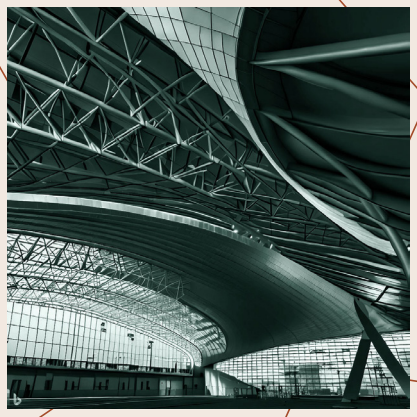
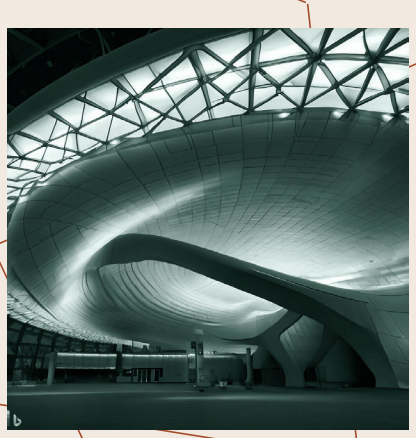
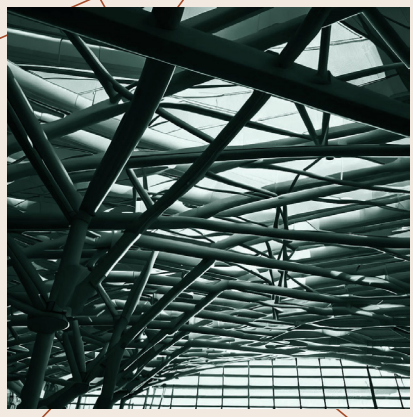
Deep Generative Design

A Deep Learning Framework for Optimized Spatial Truss Structures with Stock Constraints

P5 Version
20-06-2023

*Images on titlepage were created with the assistance of DALL-E 2

Student: Amy Sterrenberg
Mentors: Charalampos Andriotis
& Michela Turrin

TU Delft
MSc Architecture, Urbanism & Building Sciences
Building Technology Track
Sustainable Structures Theme

Student: Amy Sterrenberg
Student number: 4593057

Mentors:
Charalampos Andriotis (Sustainable Structures/Sustainable Design)
Michela Turrin (Computational Design/Design Informatics)

Delegate of the Board of Examiners:
Seyran Khademi

Acknowledgement

This thesis would not have been possible without the invaluable help and feedback of my supervisors, Charalampos Andriotis and Michela Turrin. I would like to thank them greatly for providing their knowledge and expertise in this project. I would also like to thank the board of examiners and in particular Seyran Khademi, delegate of the board of examiners, for her involvement and insights during my P2 and P4 presentations.

I am grateful to my friends and fellow students who participated in brain-storming sessions with me and offered great moral support. Thanks should also go to my family. Their belief in me has kept my motivation high in this graduation process. Lastly, I'd like to mention my pets for keeping my spirits up and offering emotional support.

Abstract

Energy use, CO2 emissions, and waste production are all significant causes of environmental issues. The building sector is a major contributor to these problems, specifically the manufacturing of (structural) steel elements. Application of reuse and/or remanufacturing, as done in a circular economy, will reduce these effects. Therefore, these techniques must to be taken into account while designing buildings and structures. However, as actors in the construction industry recognize barriers, such as time delays in the early phases of the design process, these tactics are not yet commonly used. Nonetheless, reusing structural steel components is still favoured by structural engineers, particularly when the aforementioned obstacles can be removed. This is feasible with the use of computational tools.

In this thesis, an AI based deep generative design framework is developed to optimize a 3D spatial truss structure for structural performance, material use and similarity to a defined stock of reusable materials. This workflow consist of a labelled dataset of geometries and performance indicators, a variational autoencoder (VAE), a predictive surrogate model and optimization through gradient descent. The aim of this study is to gain insight into the extent to which such a workflow can be applied in early-stage architectural and structural design exploration, especially with regards to making circular design & reuse more feasible. The case study in this thesis is a spatial truss structure supporting a flat roof. It was found that a surrogate model can be successfully trained to predict the performance of a geometry. For this, various input data representations can be used, including adjacency matrices and edge-vertex matrices. Through training of a VAE, a latent space from which meshes could be generated was successfully constructed. The VAE was able to accurately reconstruct a significant part of the geometry dataset. Through gradient descent, novel meshes were generated. Predicted performance of these meshes was increased. After assessing performance with simulations and calculations, increases in performance often remained. The largest performance increase found was 74.2%. Minor editing of meshes based on user insight demonstrated further increase in mesh performance.

Keywords: Generative Design, Structural Optimization, Multi-Objective Optimization, Artificial Intelligence, Deep Learning, Variational Autoencoder (VAE), Gradient Descent (GD)

Abbreviations

AI: Artificial Intelligence
ML: Machine Learning
NN: Neutral Network
VAE: Variational Autoencoder
GAN: Generative Adversarial Networks
EA: Evolutionary Network
GD: Gradient Descent

Contents

1. Introduction

1 Introduction

1.1 ENVIRONMENTAL IMPACTS OF THE BUILDING AND CONSTRUCTION SECTOR

The building sector is a major contributor to environmental issues such as climate change. The 2020 Global Status Report for Buildings and Construction from the Global Alliance for Buildings and Construction (GlobalABC) states that the building sector accounts for 35% of the global energy consumption and 18% of global CO2 emissions (GlobalABC, 2020). It is critical to aim for more sustainable design and construction in order to minimize and reverse these environmental impacts. These impacts include land degradation, resource depletion and water pollution (Akanbi et al., 2018, Ding et al., 2016, Ruiz et al., 2020).

Construction and demolition waste (CDW) is the most significant waste stream globally (Jin et al., 2019). It therefore contributes significantly to the aforementioned environmental impacts, CO2 emissions and energy consumption. CDW makes up over a third of the total waste production in the EU; 37.1% in 2020 (Eurostat, 2022). In the United States, 600 of the 892 tons (67%) of total waste generation was classified under construction and demolition debris (EPA, 2020). In China, CDW accounts for 30-40% of the total waste production (Huang et al., 2018).

Even though many of the components found in CDW are recyclable and reusable, only 20-30% is recycled or reused globally (World Economic Forum, 2016). Closing material loops through the use of a circular economy framework reduces the demand for virgin materials and lowers emissions and energy consumption associated with mining and manufacturing processes of these resources.

1.2 CIRCULARITY AND REUSE OF METALS

Load bearing systems contribute significantly to the embodied impacts of buildings (Allwood et al., 2012). Even though steel is 100% recyclable without loss of properties, its production still accounts for a significant part of global anthropogenic CO2 emissions: 7-9% in 2020 (World Steel Association, 2021). To reduce this, remanufacturing and reuse of steel should be considered and explored (figure 1.2.1). These methods require less reprocessing of the material, resulting in decreased energy use (Allwood et al., 2012, Iacovidou & Purnell, 2016).

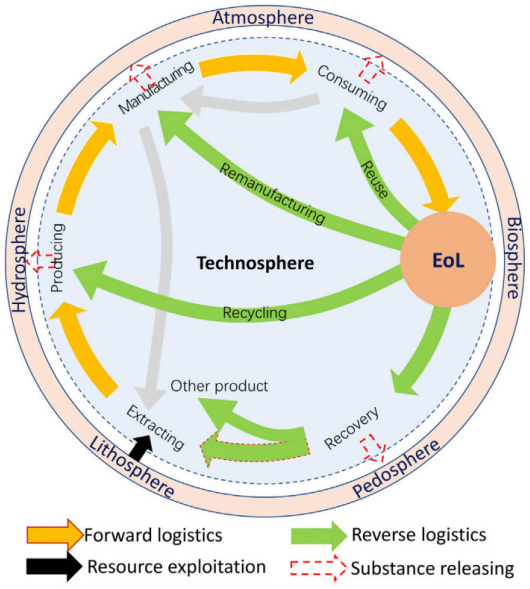


Figure 1.2.1. Schematic overview of a circular economy in the building sector.
Image Retrieved from: Zeng, X., & Li, J. (2021). Emerging anthropogenic circularity science: Principles, practices, and challenges. Iscience, 24(3), 102237.

Under the reuse principle, items are extracted directly from the existing building stock. Steel elements have a longer service life as a result of this. Following the selection of suitable steel

components, the parts must be deconstructed in a manner that does not damage the element. The items must then be stored and refurbished before being evaluated for their reuse as a structural element (Iacovidou & Purnell, 2016, Fujita & Kuki, 2016, Brütting et al., 2019).

1.2.1 DESIGN FOR REUSE

In sustainable circular design the use of virgin resources should be limited. To achieve this, designers must aim for prevention of non-essential components in their designs. Secondly, the use of reused elements should be considered, preferably in their original function (as opposed to downgrading). Finally, renewable and recycled materials should be prioritised over conventional ones. As a result, material availability is a leading factor in the design for re-use strategy. A library of available components and their properties serves as a starting point for new designs, where the size, length of available members dictate structural performance.

1.2.2 BARRIERS TO REUSE OF BUILDING ELEMENTS IN DESIGN

Even though it can aid the building and construction industry in becoming more sustainable, the reuse approach is not yet widely applied. According to a 2017 study, the main barriers perceived by structural engineers were (1) structural design principles used in existing building and (2) availability of suitable structural materials. Existing structures were found to not be designed for deconstruction, making the extraction of reusable steel elements more challenging. Local stock was also reported to be limited and, in some cases, inaccessible. Structural engineers also reported feeling pressed for time, remarking that a superior design could have been created, if they had been given more time. Despite this, structural engineers viewed reuse of steel as advantageous, especially when these obstacles can be removed (Dunant et al., 2017).

1.2.3 CASE STUDIES: STEEL REUSE

The steel reuse approach has the potential to significantly reduce environmental impacts of steel structures. A theoretical case study showed that the integration of reusable steel elements can result in a 30% decrease in embodied energy and CO2 when compared to using exclusively new steel elements (Pongiglione & Calderini, 2014).

Steel reuse has also been applied in practice, demonstrating that that steel reuse is feasible can be profitable. An early example of steel reuse is the Roy Stibbs Elementary School in Coquitlan, BC, Canada. This building was constructed in 1994 using 270 tonnes of concrete and steel salvaged from a demolished school,

Figure 1.2.2. Main barriers to steel reuse as perceived by actors in the construction industry.
Icons retrieved from (left to right):
Icon Hubs. (n.d.). Unavailable icons. Flaticon. https://www.flaticon.com/free-icons/unavailable
Winnievinzence. (n.d.). Wood icons. Flaticon. https://www.flaticon.com/free-icons/wood.
Freepik. (n.d.). Time icons. Flaticon. https://www.flaticon.com/free-icons/time.

resulting in 75% of reused steel being used in the project (Gorgolewski, 2017).

Another highly awarded case is the 1999 BedZED project; a combination of residential- and workspaces in South London. In this project, where sustainability was a main ambition, 98 tonnes of steel was reused in the structural framing. The decision to use reused steel instead new, equivalent steel, resulted in a 4% cost reduction as well as a significantly lower environmental impact, as CO2 emissions were reduced by 81,580 kg (Twinn, 2003, Tang, 2011). The 2006 Big Dig House project by John Hong demonstrates that steel can be reused from various sources. In this family house in Lexington, MA, USA, steel from a demolished highway bridge was used to build its structural framework.

More recent projects include La Cuisine, by Monteyne Architecture Works in Birds Hill Park, MB, Canada. Here, steel from demolished industrial buildings was purchased for less than 10% of the cost of new steel. This steel was then refabricated to construct the structure of the new building. The design of this building was directly based on the steel availability: form follows availability (Gorgolewski, 2017). In the 2012 London Olympic Stadium, reuse affected the design in a two ways. Firstly in its roof truss, where 2500 tons of reused steel was utilised (Brütting et al., 2019). Secondly, the stadium's upper tier was designed for deconstruction, in order to facilitate future down-scaling of the stadium and repeated reuse of its structural elements.

Another example is by Rau Architects' Alliander Office project in Duiven, the Netherlands, where steel structural components from the existing on-site building were selected and used in the new structure. This project used 90% of the material from buildings already available on site. It was the first renovation project to be awarded the BREEAM-NL outstanding sustainability certificate (Gorgolewski, 2017).

1.3 GENERATIVE DESIGN

Early design decisions often have a considerable impact on the cost, performance and quality of the final design. Design exploration therefore is a critical step in the early stages of architectural design. This can be a time-consuming process. With generative design, a large variety of design solutions can be automatically generated and evaluated. The term "Generative Design" or "Generative Design Method" (GDM) refers to a group of design methodologies based on generation and evaluation of outputs in a repetitive process. Frazer initially proposed the concept in the 1970s (Frazer, 2002). The following steps describe its structure: (Jaisawal & Agrawal, 2021)

1. A design framework, often a parametric model with defined design constraints
2. A generator that creates outputs based on this framework
3. An output evaluation method

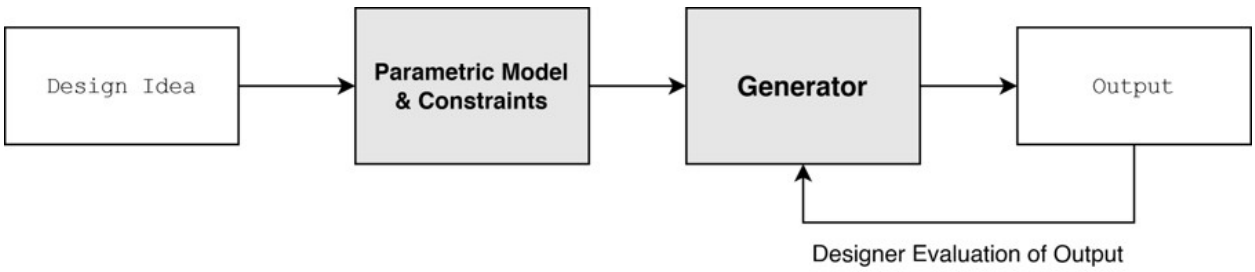


Figure 1.3.1. Flowchart illustrating the concept of generative design
Own work.

These components are illustrated in the flowchart in figure 1.3.1. This strategy is typically applied in early design stages. A design framework is established based on the initial design idea described by the designer. In architecture, this often consists of a 2D or 3D parametric model with defined design constraints. A generator then creates or modifies design geometry. These outputs are then evaluated by a designer that can select specific output and change input values, ranges and distribution. This process is repeated until a final design geometry of satisfactory quality is established. (Meintjes, 2017)

A unique quality of generative design methods is the ability to realise design solutions that go beyond the imagination of a human designer. The main advantage of generative design can therefore be described as fast and in-depth design exploration. Within this methodology, the designer's focus should be on the definition of the design and its constraints, and the evaluation and selection of a potentially large number of generated design solutions.

1.3.1 GENERATIVE DESIGN IN REUSE: METHOD COMPATIBILITY AND POTENTIAL INNOVATION

Generative design methods, as described above, are often applied in early design stages. It is this same initial design phase that is essential in design for reuse strategies. Moreover, this approach requires a flexible definition of the design concept, as designs must be adaptable to material availability. This definition consist of a set of constrains, similar to the framework used as input in generative design. Due to these commonalities, both methods are compatible. As stated, material availability is a leading factor in the design for reuse strategy,. Material availability and time limits were reported as two of the main concerns regarding steel reuse in buildings. Additionally, a constantly changing material stock can make it difficult to secure materials for construction projects. For these reasons, a rapid design process, such as that providable by generative design methods, can be beneficial and aid the effort to further incorporate reuse in building design and construction.

1.4 PROBLEM STATEMENT & RESEARCH QUESTIONS

The previous sections give context for the problem statement of this thesis. It can be concluded that the building sector is a major contributor to environmental problems, including energy consumption, CO2 emissions and waste production. Reusing structural steel elements can aid in reducing these impacts. However, this strategy is not yet widely applied, as actors in the construction industry still perceive barriers, including time delays in the design process. Still, reuse of steel elements is seen as favourable by structural engineering, especially when these obstacles can be lifted. Nonetheless, tools that can aid this process are not yet developed.

Generative Models that integrate Artificial Intelligence can be integrated into this workflow to optimize for structural performance and similarity to material stock. This introduces the design assignment addressed in this work: to gain insight into the application of AI as a tool in reuse-based design strategies by developing an AI based generative design framework that prescribes innovative designs for 3D structures consisting of linear elements, based on a circular approach where material reuse is prioritized, with an integrated computational workflow using deep learning models.

1.4.1. RESEARCH QUESTIONS

In this thesis, the following research question is answered:

Can an artificial intelligence (AI) based generative design framework generate new spatial (3D) truss design solutions, with optimized structural performance, minimized material use and that consist of linear elements that closely match elements from a reusable

material stock, in reference to the training dataset, and therefore be used as an effective tool for design exploration in early design stages of the materially circular architectural design process?

The answer to this question provides insight into the effectiveness of AI as a tool for effective integration of reused materials in the structural design process. Additionally, the following sub-questions are discussed:

1. What types of AI can be used in generative models?
2. What form of data, describing spatial truss-like 3D geometry with variable-length linear members, could be used to train the AI generative framework to generate new design solutions with similar properties to this training dataset?
3. How can a spatial truss-like 3D geometry be computationally evaluated on its structural performance, so that the outcome data can be used to train an AI generative framework?
4. How can a spatial truss-like 3D geometry be computationally evaluated on its material use, so that the outcome data can be used to train an AI generative framework?
5. How can a restrictive, yet suitable stock library be defined for materially circular design of a spatial truss?
6. How can a spatial truss-like 3D geometry be computationally evaluated on its similarity to a stock library, so that the outcome data can be used to train an AI generative framework?
7. How can an AI generative framework be designed to reconstruct input and generate new spatial truss-like 3D geometry with optimized structural performance, minimal material use and high similarity to the material stock?
8. Can a surrogate model effectively predict the performance score of datasets describing a set of spatial truss-like 3D geometry, in its encoded and decoded forms?
9. How can optimized solutions be found within the data generated by the AI generative framework?
10. How does the proposed AI generative framework compare to conventional shape and topology optimization tools?

1.4.2. METHODOLOGY

This study consist of two parts: A literature review & a designing and training the deep learning generative design framework. In these section, both parts are described.

In the first part, a literature review is done to explore AI models and their use in (structural) design optimization. This study includes the following terms: Artificial Intelligence (AI) and its subsets, Neural Networks (NN) and various types of NNs, Variational Autoencoders (VAE), Loss Function and Gradient Descent. Secondly, research is done on defining the spatial truss structures in a way that makes it suitable for training the neural network. This too done through literature review and includes information on computational mesh and wire-frame model representations. Finally, various computational tools that can be used in this process is researched, evaluated and selected.

The second part consists of the designing and training of the deep learning generative design framework. Here, an outline for this framework is described. The proposed framework can be described as having 4 components: A material library (stock), a generated dataset for training, a structural evaluation of the generated dataset, a VAE and a surrogate model. The following flowchart (figure 1.4.1) shows the proposed set-up of the framework. This set-up is inspired by a framework designed for a masterthesis by Pavlidou (2022).

The first step towards realisation of this framework is creating an input dataset. This dataset describes the spatial truss structure in

different configurations. Next, performance of the geometry variations in this dataset need to be measured. Due to the limited timeframe and scope of this study, the input dataset and design objectives for the suggested workflow are simplified. Therefore, the dataset is evaluated on three factors: (1) Structural performance, (2) material use and (3) the similarity to the stock library. These factors are translated into one numerical performance indicator. Structural performance evaluation is done using the Karamba3D Plugin in Rhinoceros Grasshopper, a parametric Finite Element Method (FEM) program (Karamba3D, n.d.). To create this dataset, literature research into the data definition of 3D mesh-like geometry is done, as mentioned previously. Details on how the structural performance is assessed are given in Chapter 2.1 of this report. In the next step, this dataset is used to train a Variational Autoencoder (VAE). This is neural network architecture was first introduced by Diederik Kingma and Max Welling in 2014. (Kingma & Welling, 2014). It is explained further in Chapter 2.1. The VAE is used to generate a new set of data. Next, geometry and performance data is put into a surrogate model. This neural network is then trained to predict the structural performance of the spatial geometry that the input data represents. This input data can be in encoded or decoded form. Finally, the VAE and the surrogate model's outputs can be used for backpropagation. This process, using a gradient descent optimizer, finds the best solutions within newly generated geometries. Finally, solutions and the framework can then be assessed and evaluated. The findings in the literature review will further specify these steps. The major steps mentioned in this methodology are outlined in the timeline (Appendix I).

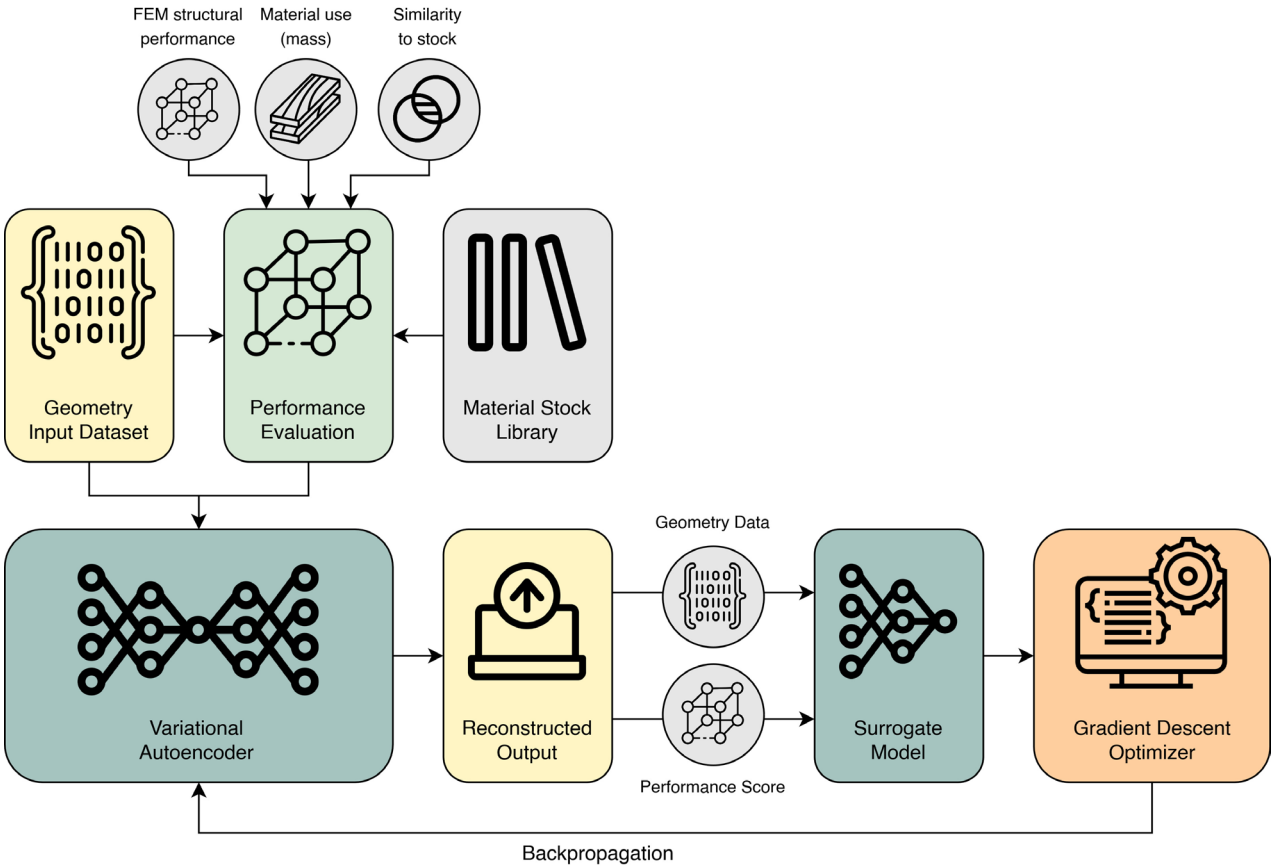


Figure 1.4.1. Diagram showing the proposed framework.
Own work. Icons retrieved from:
Wood icons created by winniwvinzence - Flaticon, from https://www.flaticon.com/free-icons/wood.
Transparency icons created by Freepik - Flaticon, from https://www.flaticon.com/free-icons/ transparency.
Matrix icons created by Freepik - Flaticon, from https://www.flaticon.com/free-icons/matrix.
Library icons created by inkubators - Flaticon, from https://www.flaticon.com/free-icons/library.
Neural network icons created by Freepik – Flaticon, from https://www.flaticon.com/free-icons/neural-network.
Output icons created by Peter Lakenbrink - Flaticon, from https://www.flaticon.com/free-icons/output.
Code icons created by Eucalyp - Flaticon, from https://www.flaticon.com/free-icons/code.

1.4.3. CASE STUDY

The chosen simulated case study is that of a non-curved triangular spatial truss. Such spatial trusses are typically used for large-span roof designs and can also be used to create curved shell structures and are made of steel. The chosen case of this study supports a planar slab (roof). Specifically, the spatial truss configuration as used in the Orange Hall (Oost Serre) and Model Hall (Zuid Serre) of the Architecture and the Built Environment Faculty of TU Delft are used (see figure 1.4.3). These spatial trusses can be described as a two layers of quadrilateral grids and an in-between triangular structure. Due to the limited timeframe and scope of this study, the input dataset defines a simplified version of the truss structure from the case study. The research will start with a 5x5 vertex grid (4x4 in quadrilateral cells). It consist of 128 linear elements. This size keeps the structure as simple as possible, so that the designed AI-model can quickly be tested, while still ensuring it is a realistic representation of the case study. An exploded view of this geometry is shown in figure 1.4.2. In the input dataset, individual linear members are replaced in order to generate design variations. This is explored further in Chapter 2 and 3 of this report.

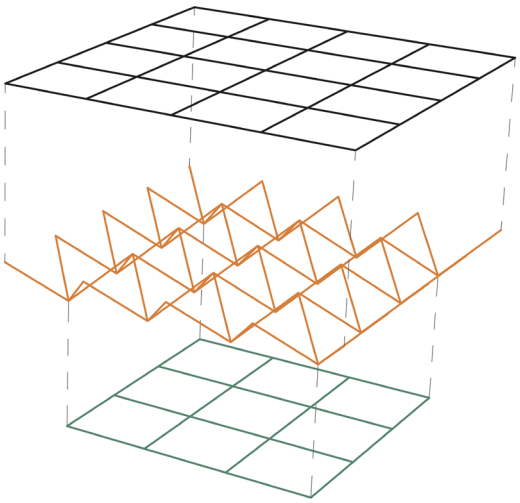


Figure 1.4.2. An exploded view of the spatial truss configuration used as a case study.
Own work.

Figure 1.4.3. TU Delft Model Hall (Zuid Serre).
Retrieved from Maison h. (2018).
A critical evolution of spaces for architectural education. http://maisonh.nl/2018/01/27/architecture-schools/

The aim of this study is to gain insight into the application of AI as a tool in reuse-based design strategies by developing an AI based generative design framework that prescribes innovative designs for 3D structures consisting of linear elements, based on a circular approach where material reuse is prioritized, with an integrated computational workflow using deep learning models like variational autoencoders.

2. Literature Review

2 Literature Review

To answer the research questions as described in the first chapter, information on artificial intelligence and the computational definitions of geom¬etry must be explored. This is done in this chapter, which is divided into three sections. In the first section, the following terms and concepts are explained:

- Artificial Intelligence (AI)
- Machine Learning (ML)
- Deep Learning
- Neural Networks (NN) and the various types of NNs.
- Variational Autoencoder (VAE)
- AI training
- Loss Function
- Gradient descent

The second section will focus on the definition of geometry to be used in deep learning algorithms. Here, the following topics are explored:

- Wireframe models
- Meshes and types of mesh representations
- Matrix representations
- Mesh Modification

Thirdly, several existing frameworks regarding generative design and design optimization through AI are presented and analysed. The following cases are studies:

- "Deep Generative Design. A Deep Learning Framework for Optimized Shell Structures" by Pavildou. (2022)
- "Design of Truss Structures Through Reuse" by Brütting et al. (2019)
- "Deep Generative Design: Integration of Topology Optimization and Generative Models" by Oh et al. (2018)

Finally, node connections and types are discussed in section 2.4.

2.1 ARTIFICIAL INTELLIGENCE AND ITS SUBSETS

The term artificial intelligence (AI) refers to the ability of machines or computers to perform activities with (simulated) human-like intelligence (Helm et al., 2020). Intelligence can be defined as the ability to perceive and process information, and use this information for reasoning, problem solving and learning. (Dalvinder & Grewal, 2014, Oxford English Dictionary, 2021)

Machine learning (ML), a subset of AI, refers specifically to the ability of computers to be trained, meaning to learn and improve the manners in which a task is performed through computational algorithms, without being explicitly (pre-)programmed for it. (Helm et al., 2020). Artificial neural networks (NN or ANN), in turn, form a subset of ML. Neural networks are described in further detail in the next section.

2.1.1 NEURAL NETWORKS AND DEEP LEARNING

A neural network within AI focusses on mimicking the learning process of a biological neural network. It consist of layers of artificial neurons with connections in between, in turn consisting of algorithms with inputs, weights, a bias and an output. The bias is also called the threshold. It can be described by the following algebraic formula (Kavlakoglu, 2020):

$$\sum_{i=1}^{m} w_i x_i + bias$$

These connections determine how information is transferred through the network. When a neutral network is trained, it learns by adjusting

these weights and biases for each variable. This learning process can be classified as a supervised one. Supervised learning includes a 'teacher' that, given the same input, decides the desired output. Comparing this desired output to the output given by the neural network gives an error margin that can be used to train the NN. The objective is to minimize error, so the NN output closely resembles the desired output. Finally, when the NN performance is acceptable or does not continue to improve, the learning is stopped. By contrast, the goal of unsupervised learning is to gain insight into the underlying data structure by modelling it, with clustering as a focal point (Kavlakoglu, 2020, Geeksforgeeks, 2022).

Finally, Deep learning is a subset of NNs. Deep learning typically consist of multilayered NNs that are trained on vast amounts of data The distinguishing feature of deep learning algorithm is for the number of layers between the input and output layer in its neural network to be at least three (Kavlakoglu, 2020).

There are several types of NN. In this review, six categories are discussed. These are:

- Perceptron
- Feed Forward Network (FF)
- Multilayer Perceptron (MLP)
- Convolutional neural network (CNN)
- Recurrent neural network (RNN)
- Autoencoders (AE)

Perceptron

The perceptron model, created in 1958 by Frank Rosenblatt, is the oldest type of NN, as well as the simplest (Sethi, 2019, Vidiyala, 2020). This type of neural network consist of only one neuron and two layers; an input and an output layer. The input layer consist of several inputs, each with a weight. The output a single (binary) variable.

Feed Forward Network (FF)

The feed forward network (FF) consist of neurons. These networks, in contrast to the perceptron model, have multiple layers, resulting in hidden layers in-between input and output layers (Vadapalli, 2020). The name of this type of NN refers to the direction in which data flows; this NN has no backward propagation. This means that the weights of the data are not updated. This architecture is shown in figure 2.1.1.

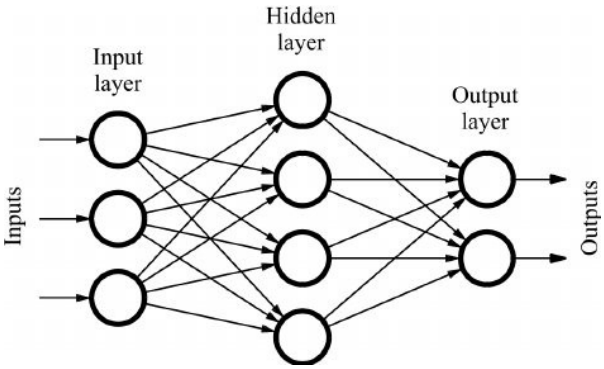


Figure 2.1.1. A diagram showing the architecture of feed forward neural networks.
Retrieved from Quiza, R., & Davim, J. P. (2011). Computational methods and optimization. In Machining of hard materials (pp. 177-208). Springer, London.

Multilayer Perceptron (MLP)

Multilayer Perceptron NNs introduce back-propagation: a method of calculating the gradient of the loss function within the neural network. This allows for training of a neural network by updating individual weights in an iterative optimization process (Wood, n.d.) Due to both forward and backward propagation, MLPs are bi-directional.

MLPs can be used in deep learning. They are a case of supervised learning, where weights are adjusted by gradient decent; an optimizing algorithm aiming for low error and maximum accuracy (Trehan, 2020). Gradient descent consists of three steps. First, initial, randomized values are needed. These values are then updated, until the slope equals zero. This term is explained in further detail in section

2.1.6 "Training Neural Networks". The chosen learning rate for this process should not be too large, as this will converge the model, so the minimum won't be reached. A learning rate that is too small, however, will result in a longer learning time.

Convolutional neural network (CNN)
Convolutional neural networks (CNN) follows the same principles as MLPs. However, they implement convolutional layers. This type of neutral networks are typically used with image and video input and pattern recognition (Sethi, 2019, IBM, n.d.). In this type, the convolutional layers are followed by pooling layers. These layers join the most important characteristics, as found by the filters, together using a variety of methods.

Recurrent neural network (RNN)
Typically, neural networks output is dependent on its input data. Recurrent Neural Networks RNN), however, addresses other states of data as well (Sethi, 2019, Vadapalli, 2020). RNNs have a layer that stores the previous inputs called state matrices. This makes it useful when data that is observed over time serves as input. This type of neural network is different from the previously mentioned types, as instead of a feed forward system, it works in loops and works with continuous, sequential data. It's main use case are predictions from time based data.

Autoencoders (AE)
The main difference between an autoencoder and the other neural network types mentioned above is its task. Autoencoders are used to process data by compression with minimal quality loss (Sethi, 2019) This is done through a hidden layer called a bottle neck. This layer consist of less neurons than the input and output layers. The main advantage of using autoencoders is that large amounts of data can be processed in a compressed state. These neural networks form the basis of the variational autoencoder, which is discussed in the next section (Rocca, 2019b).

2.1.2 SURROGATE MODELS

Computer simulations can be made to predict a system's output through an approximation model. This is useful, as simulations are typically significantly more computationally heave then statistical models. Statistical models that can approximate a system's behaviour or output are known as surrogate models, metamodels or emulators and can be used for sensitivity analysis, optimizations, or risk analysis (Guo, 2020). The goal of a surrogate model within this thesis is to accurately predict the performance score of generated geometries, so that the optimal solutions can be found after. In theory, this could also be done using the simulating tools built/used in the dataset generation process. However, a predictive model can greatly compress this task, minimizing

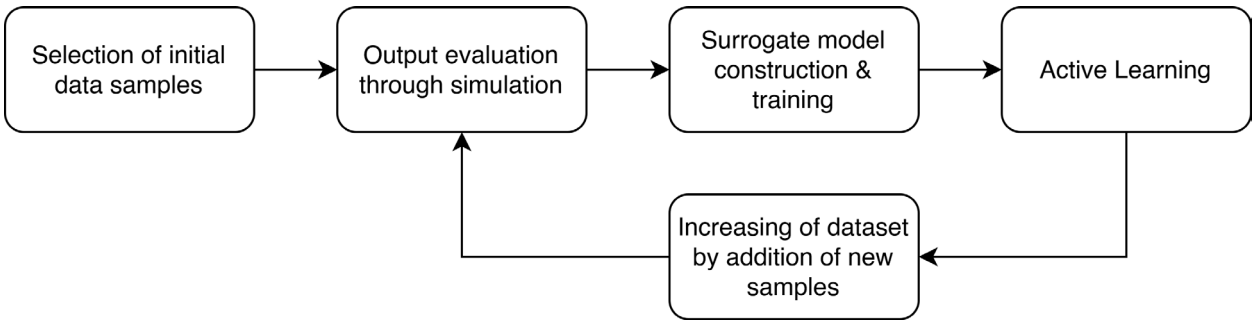


Figure 2.1.2. A flowchart showing the workflow of a surrogate model.
Own work, based on Guo, S. (2020).
An introduction to Surrogate modeling, Part I: fundamentals. Towards Data Science. https://towardsdatascience.com/an-introduction-to-surrogate-modeling-part-i-fundamentals-84697ce4d241#91b2

computation time.

The workflow of a surrogate model portrays a case of supervised machine learning, as the training data is labelled and the desired output is therefore known. The process of creating a surrogate model is shown in the flowchart in figure 2.1.2. The process starts with construction of initial training data by intentionally picking samples that evenly represent the range of the parameter space well. After these samples are selected, the output values (labels) that these samples produce need to be calculated. In this paper, this is the performance score, as mentioned in the "Methodology" in chapter 1. The samples and the output evaluations together form the initial labelled training dataset. Next, the surrogate model can be constructed and trained. This process may also include active learning; the addition of new samples to the training dataset during the training process. This method is typically applied until adequate accuracy is reached (Guo, 2020).

2.1.3 VARIATIONAL AUTOENCODER (VAE)

A Variational Autoencoder (VAE) is a generative model that consists of two neural networks, an encoder and decoder. It can be described as a self-supervised AI. Autoencoding is a form of data compression through dimensionality reduction; reducing the number of features used to describe data. As autoencoders are a form of ML, the compression and decompressions are learned automatically instead of being designed by humans, which means that only appropriate training data is required to create new autoencoders (Chollet, 2016). There are two ways in which this can be done: selection and extraction. With selection, some features are kept while others are discarded, whereas with extraction a new set of features is made entirely, based on the existing features from the input data. So, the encoder takes the input data and encodes it by reducing the number representing features through dimensionality reduction. The output of this process is encoded data of a specified dimension (Rocca, 2019b). It should be noted that autoencoders are data-specific. This means that they can be trained to compress and decompress a specific type of data, as present in the training dataset. (Chollet, 2016). The decoder – the second neural network of the VAE structure – is the exact reverse process. The input of the decoder is the encoded data. The output is encoded-decoded data of the same shape and type of the input data. This process is also shown in the flowchart in figure 2.1.3.

For this process to work optimally, the neutral networks must be created in such a way that information loss through encoding is minimized, as this will result in minimal error of reconstruction in the decoding process. The best encoding-decoding pair can

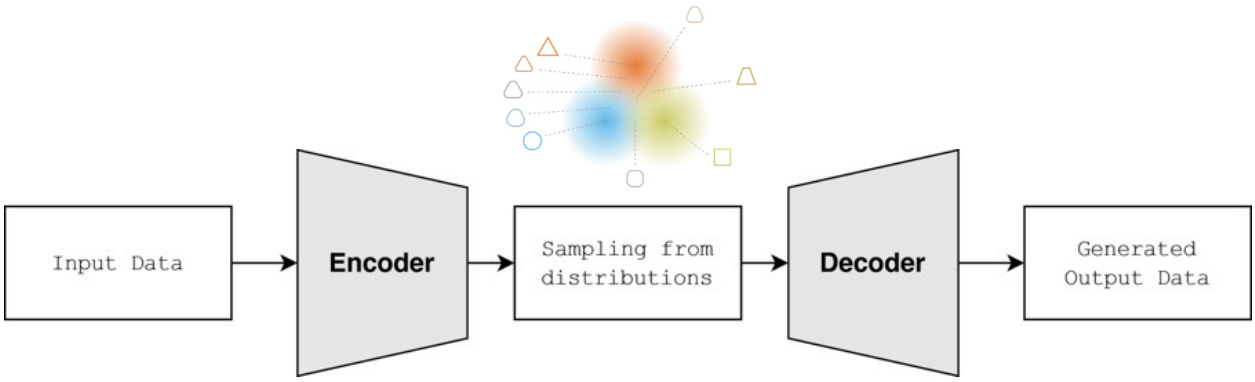


Figure 2.1.3. A flowchart explaining the architecture of Variational Autoencoders.
Own work + image above "Sampling from distributions" retrieved from Rocca, J. (2019). Understanding Variational Autoencoders (VAEs). Towards Data Science. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

be found through an iterative optimization process, where the initial data is compared to the output of the decoder. The result of this evaluation results in backpropagation of the NN architecture, resulting in updated weights.

For data generation, the latent space – the space in-between the encoder and decoder – is essential. When the encoder and decoder neutral networks are trained, this latent space should resemble a regularized gradient of information. Generation can be done by selecting randomized samples from the latent space and feeding them through the decoder. The output of the decoder will then be reconstructed input and new, generated content that resembles the initial input data.

This technique learns by iterative optimization and generates new data based on an input training dataset. The aim is to create a latent space with overlapping distributions, which can be used for data generation with a (gradient decent) optimizing algorithm. If the properties of the latent space are good – when it is regularized so a gradient of information is formed – the decoder can be used to generate new output data (Rocca, 2019b). This created latent space is visualised above "Sampling from distributions" in figure 2.1.3.

Limitations

The generation of data through the use of a VAE is dependent on how well the VAE is able to create a regular latent space (Rocca, 2019b). If the properties of a latent space are perfect, any point in the latent space can be decoded to meaningful generated data. However, the latent space of an autoencoder is not regularized by definition, as the training goals of the autoencoder is limited to minimizing loss when encoding and decoding data. The organisation of the latent space is not an explicit goal within the training process here. So, using this type of training may result in a distribution of the data that is incompatible with data generation; when the latent space is not regularized, some decoded datapoints will produce samples with no value or meaning.

Regularisation of the latent space & Kullback–Leibler (KL) divergence

It is, however, possible to explicitly regularise the latent space. This is the defining trait of a variational autoencoder (VAE) as opposed to autoencoders in general. Regularisation is defined by the continuity and completeness of the latent space. Continuity refers to the similarity of two points that are close in the latent space: these outputs should be similar when the latent space is regularised. Completeness refers to the value or meaning that decoded samples output; decoded outputs should be meaningful when the latent space is regularised (Rocca, 2019b). This way, a gradient of information is created, as visualised in figure 2.1.3.

A widely applied method to achieve this is implementing regularization into the training process. This is done by encoding inputs as a distribution rather than individual samples. Through sampling from this distribution, a reconstruction error can be calculated. This can then be used for backpropagation within the network. This results in the introduction of a "reconstruction term" on the final layer and a "regularisation term" on the latent layer (Rocca, 2019b), the second of which is defined as the Kullback–Leibler (KL) divergence (Kullback, 1959). The Kullback–Leibler divergence is used to measure the difference between to distributions by describing the expectation of the difference. This difference for two distributions $I_p(x) = -\log\ p(x)$ and $I_q(x) = -\log\ q(x)$ is defined as a log likelihood ratio (Odaibo, 2019):

$$\Delta I = I_p - I_q = -\log p(x) + \log q(x) = \log\left(\frac{q(x)}{p(x)}\right)$$

The forward KL divergence when p(x) is the reference distribution and q(x) the

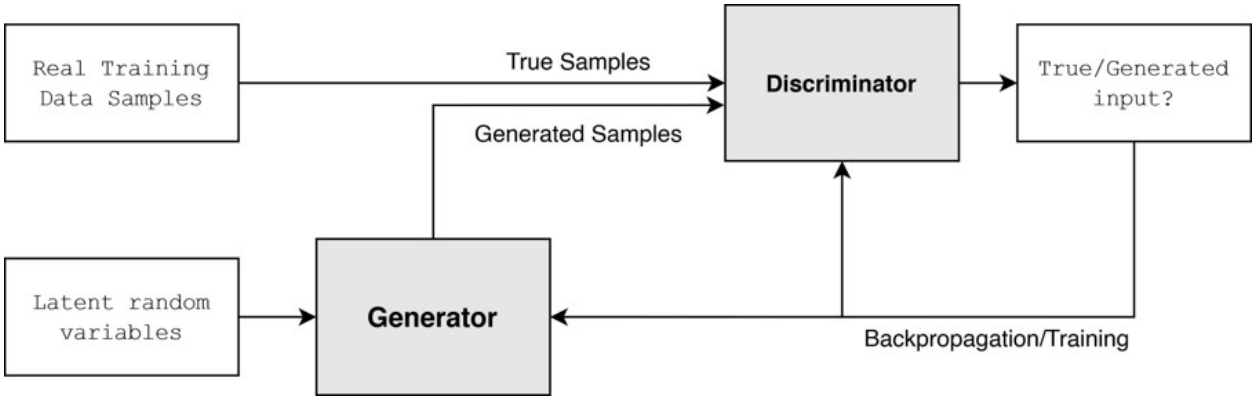


Figure 2.1.4. Flowchart showing a generative adversarial network.
Own work.

approximation is then given by: (Odaibo, 2019, Sachdeva, 2021)

$$D_{KL}(q(x)\|p[(x)) = \int (\Delta I) q(x) dx = \int q(x) \log\left(\frac{q(x)}{p(x)}\right) dx$$

The reverse KL divergence is:

$$D_{KL}(p(x)\|q[(x)) = \int (\Delta I) p(x) dx = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

It should be noted that the outcomes of these equations are not equal. The forward KL is often used in machine learning (Sachdeva, 2021). In VAEs this second distribution is defined as a standard normal distribution. This leads to a regularized latent space with continuity and completeness, as previously described. However, this method also generally results in a higher reconstruction error. These two components need to be balanced for the VAE to work optimally. This balance can be defined by a KL coefficient: (Lunot, 2019, Rocca, 2019b)

$$Loss_{total} = Loss_{reconstruction} + (KL\ Coefficient * Loss_{KL})$$

2.1.4. GENERATIVE ADVERSARIAL NETWORK (GAN)

Generative Adversarial Networks (GANs) like VAEs, are a deep-learning network. The framework was first introduced in 2014 (Goodfellow et al., 2014). GANs are a case of unsupervised learning and can be used to generate data. This is done through two neural networks; a generator and a discriminator, that contest with each other, resulting in improvement of both networks after repetition. The discriminator takes samples of data and tries to classify them as true or generated, while the generator is trained to fool this system as much as possible. This technique can learn to generate new data based on its input. The aim is for the generator to generate data that closely resembles the real data distribution (Rocca, 2019a). This is shown in figure 2.1.4.

In this model, the generative network takes a random input with a set latent dimension. After training the GAN, the generator outputs data that follows the targeted probability distribution. The discriminative network takes input with the input shape. It outputs the probability of the input set to be real data. Real data is data that is part of or similar to the original dataset (Rocca, 2019a).

The loss function can be defined as:
$$\min_G \max_D V(D,G) = \mathbb{E}_x[log\ D(x)] + \mathbb{E}_z[log(1 - D(G(z)))]$$

For the training of this network, the aim is to maximize this error for the discriminator (D). Simultaneously, when training the generative network (G), the aim is to minimize log(1-D(G(z))) (Nanos, 2013.)

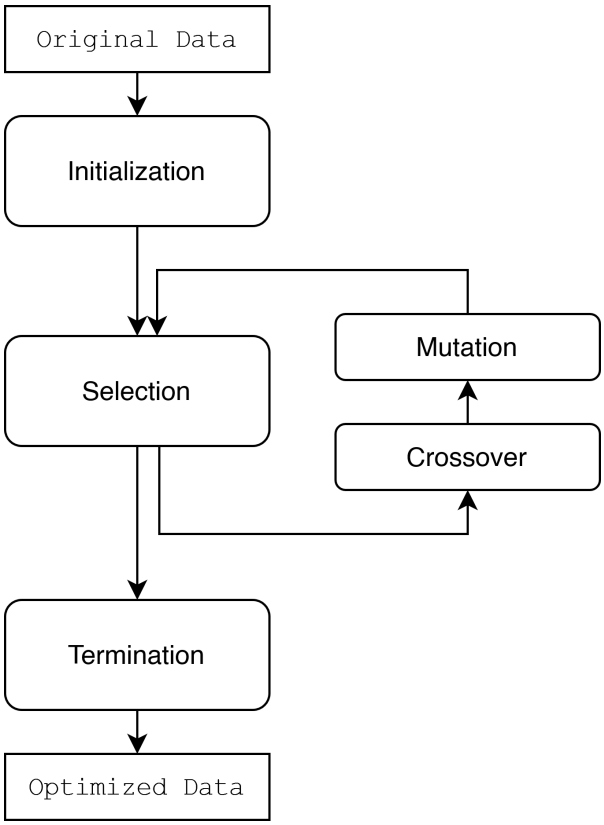


Figure 2.1.5. Flowchart showing evolutionary algorithms.
Based on Soni, 2018.

2.1.5. EVOLUTIONARY ALGORITHMS (EA)

Evolutionary algorithms (EAs) are also a subtype of machine learning (Soni, 2018). However, this method takes a different approach. Here, a model learns through selection of well-performing samples and evolution through alteration of these samples. As a result, high performing data can be found, while low performing is dismissed in each iteration/generation. This process is shown in figure 2.1.5.

One way to implement EAs in Rhinoceros Grasshopper is through the plug-in BioMorpher (Harding, n.d.). This plug-in provides an environment for Interactive Evolutionary Algorithms, which allows users to explore and optimize parametric models. In this study, evolutionary algorithms for multi-objective optimization are used for comparison of results. The main focus of this study, however, lies on deep generative models, mainly VAEs.

2.1.6 TRAINING NEURAL NETWORKS

Neural Networks have to be trained in order to perform a task. Usually, this is done using datasets. A learning curve shows how well a model has learned to predict the output.

Loss function

A loss function is a formula that defines how well a neural network is performing its task. It's mathematical definition is as follows: (Bushaev, 2017)

$$L(y, \hat{y}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

Where y equals the desired output and ŷ equals the actual output produced by the neural network. The loss function L indicates the performance of a neural network throughout the learning process. At the end of its training, the

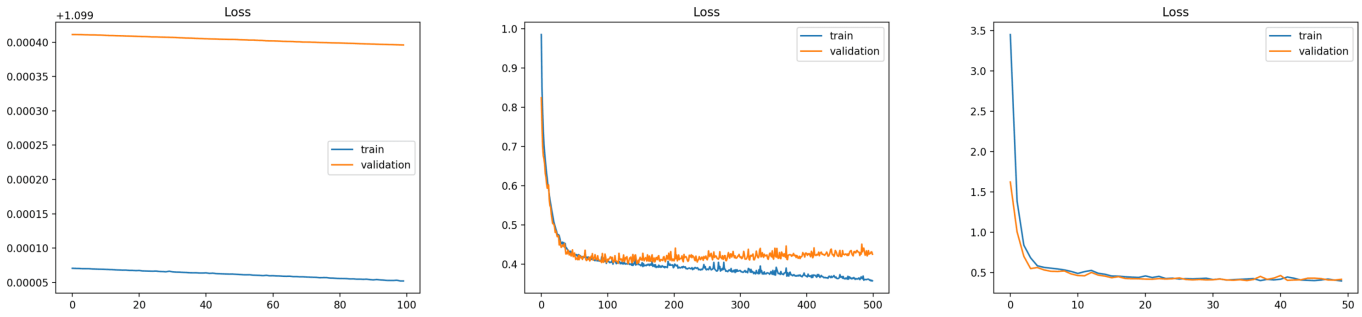


Figure 2.1.6. An underfitting, overfitting and well-fitting training learning curve.
Retrieved from Brownlee, J. (2019a).
How to use Learning Curves to Diagnose Machine Learning Model Performance. Machine Learning Mastery. https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

loss function should be minimized. Plotting the loss function over epochs – the number of training cycles the neural network has undergone – will result in a neural n¬etworks train learning curve. This curve should be compared to the validation learning curve: a learning curve that shows how well a network is generalizing. Generalization refers to the ability of a neural network to predict the output from data that was not introduced during its training (Brownlee, 2019b).

When analysing these learning curves, a model can display one of three cases: Underfit, overfit or a good fit (figure 2.1.6). An underfit model may be unable to predict outputs with a sufficiently low error and is therefore unable to learn the training dataset. This is the case when the train and validation learning curves remain flat around a loss value while training. Underfitting also occurs when the training loss continues to decrease when training is finished. In this case, the model requires further training (Brownlee, 2019a).
Learning curves are overfit, when a model has learned 'too well', meaning that errors or noise are followed too closely, leading to inaccurate outputs. This can be identified in the graph when the train learning curve continues to decrease, while the validation learning curve shows an increasing generalization error again after decreasing before (Brownlee, 2019a).
Finally, learning curves show a good fit when the circumstance as described previously do not occur. This means that the training and validation learning curves both decrease to a similar, stable level. Typically, the loss on the training curve is lower than that of the validation curve. This difference is called the generalization gap. With well-fitting curves, this gap is minimal (Brownlee, 2019a). The images on the next page shows examples of underfitting, overfitting and good fitting learning curves (figure 2.1.6.).

There are two more cases that then show up in the train and validation learning curves. These are caused by unrepresentative datasets (Brownlee, 2019a). When the training dataset is unrepresentative, both the train and validation losses may decrease, but gaps between the curves remain large. There are several reasons a training dataset is unrepresentative. However, the main reason is that a training set contains too few entries.

Gradient Descent Optimization
As described previously, when training a neural network, the goal is to minimize the loss. Therefore, the minimum of the loss function curve should be found. This can be done through gradient descent; an iterative optimization algorithm for finding this minimum. The process, as shown in figure 2.1.7, starts at a point on the curve, on which the slope is calculated. Then a step along the curve towards the decrease of that slope is taken. This is called the incremental step. This process is then repeated until the minimum is reached (Patrikar, 2019).

When the loss function presents as a convex curve, this process is relatively easy; whatever point on the curve serves as the starting point, the true minimum can be found. In non-convex curves, that is often found for loss functions of neural networks, this process is more complex. An example of such a curve is shown in figure 2.1.8.
When applying gradient descent in this case, there is a risk of finding only the nearest minima, instead of the true, global minimum. In this case, the starting point has an effect on the result. As a result, the method previously described will not suffice. Non-convex curves can also contain saddle points, defined as plateau-like parts of the curve. For all these points: $dy/dx = 0$. In the case of the loss function: $dL/dW = 0$, with L being the Loss function, and W the weights.

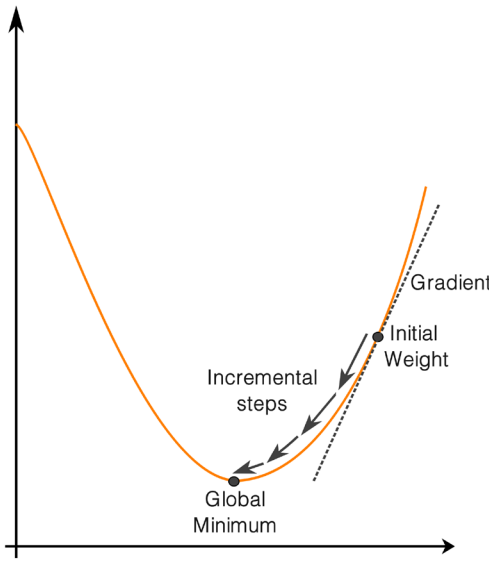


Figure 2.1.7. Gradient descent for a convex curve.
Own work, based on Patrikar, S. (2019).
Batch, Mini Batch & Stochastic Gradient Descent. Towards Data Science. https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a

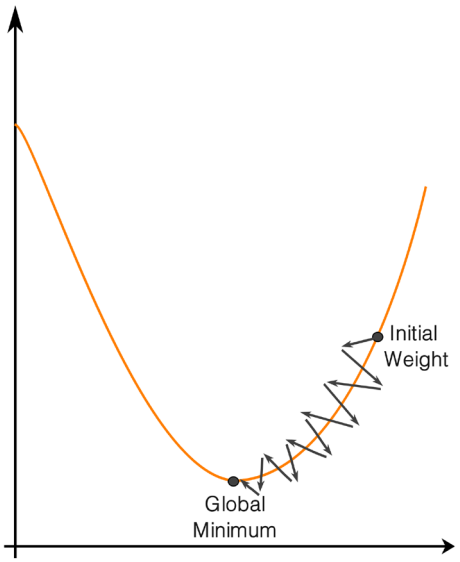


Figure 2.1.9. Stochastic gradient descent.
Own work based on Roy, A. (2020).
Introduction to Gradient Descent: Weight Initiation and Optimizers. Towards Data Science. https://towardsdatascience.com/introduction-to-gradient-descent-weight-initiation-and-optimizers-ee9ae212723f.

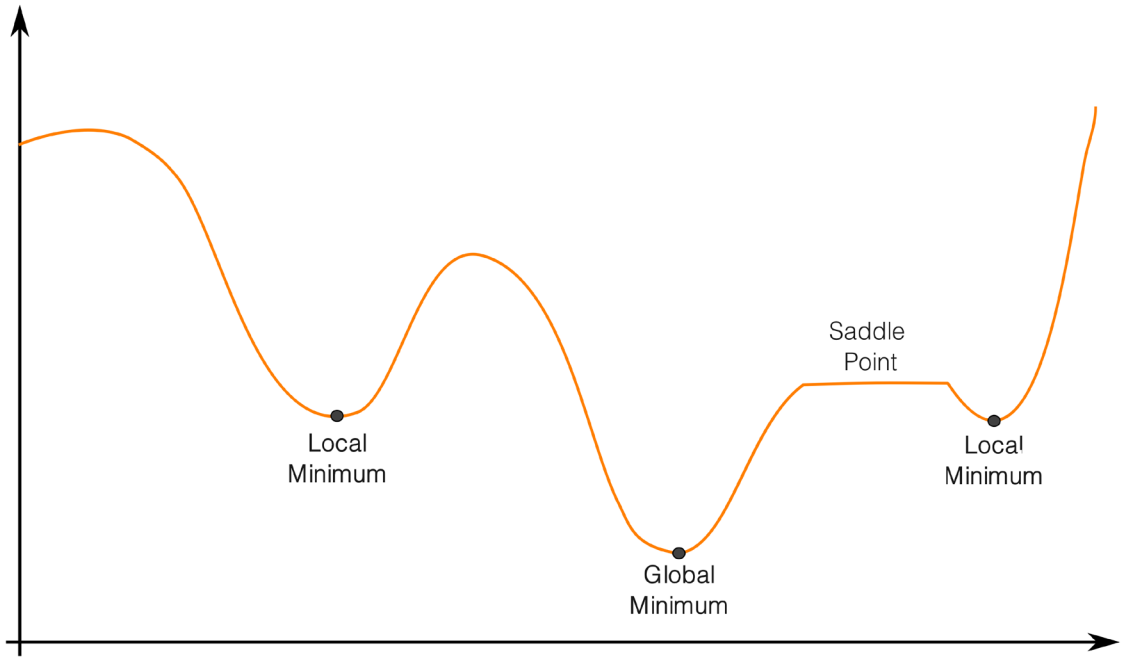


Figure 2.1.8. A non-convex curve.
Own work.

There are several methods for gradient descent optimization. Here, the following three types are described:

- Batch Gradient Descent (BGD)
- Stochastic Gradient Descent (SGD)
- Mini Batch Gradient Descent (MB-GD)

Batch Gradient Descent (BGD)
With the batch gradient descent method, all gradients are taken into account at once. The sum of all the errors is calculated and all weights are updates over one epoch (Patrikar, 2019 and Roy, 2020). This method is effective for convex curves and can be visualised as done in figure 2.1.4. It is however not efficient when the batch is large, as is the case when a large dataset is used. Using batch gradient descent in this case would be computationally heavy.

Stochastic Gradient Descent (SGD)
This issue can be avoided when stochastic gradient descent is used (figure 2.1.9). Here, one example is processed at a time. After the gradient of this starting point had been calculated, the weights are updated. These steps make up one epoch (Patrikar, 2019). Because each sample is taken into account separately, this method is useful for curves with local and global minima, as it allows for skipping over local minima (KiKaBeN, 2021). As the weights are updated after every sample, the gradient descent shows some noise (Roy, 2020).

Mini Batch Gradient Descent (MB-GD)
A mixture of the two previously mentioned methods is used in mini batch gradient descent. A batch with a fixed size, smaller than the size of the original dataset, is used (Patrikar, 2019). These are referred to as mini-batches. In one epoch one mini-batch is defined and fed to the neural network. Then, the mean gradient of this mini-batch is calculated and used to update the weights. The noise, in this case, is much less than with SGD. Additionally, the computation is less computationally heave than batch gradient descent.

2.2 DEFINING GEOMETRY FOR AI PROCESSING

As stated in section 2.1, variational autoencoders can generate new data. This data is based on the input dataset. Therefore, it is essential to define the desired geometry for this purpose. In this section methods to define the spatial truss structure are explored.

2.2.1. SELECTED SPATIAL TRUSS STRUCTURE

The spatial truss that is chosen as the case study for this thesis is that of an non-curved triangular spatial truss. The structure consist of two planar layers of quadrilaterals, connected by a triangular in-between structure. All elements in this structure are linear. Figure 2.2.1 shows an example of this truss structure. The research

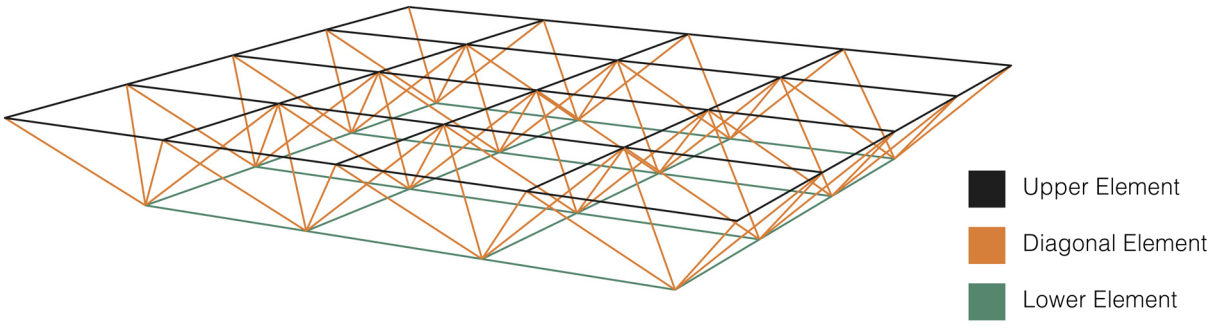


Figure 2.2.1. A diagram showing the base geometry used for the input dataset.
Own work.

will start with a five by five vertex grid, consisting of 128 linear elements. There are several ways in which such a geometry can be defined for processing in algorithms. These methods are explored in the following sections.

2.2.2. WIRE-FRAME MODEL & GRAPHS

A wire-frame model is a geometrical model consisting of vertices and lines and/or curves. Vertices are locations in 3D space. They are described by x, y and z coordinates that indicate distance from the origin. These coordinates, combined with the vertex indices are described in a vertex table. Edges describe the connection between vertices. These connections can be listed in an edge table consisting of the edge indices and the start and end vertex of each edge. This is done in the example in figure 2.2.3, where an example geometry is given.

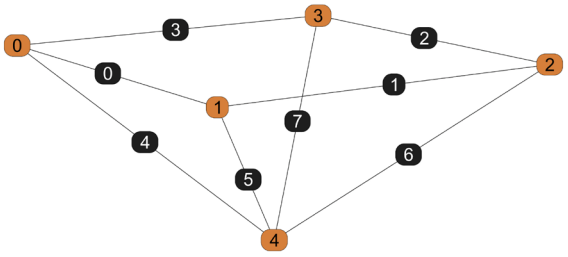


Figure 2.2.3. An example geometry with vertices (orange) and edges (black) marked.
Own work.

For this geometry:

```
Vertices = [v1, …, v4]
Edges = [e0, …, e7] = [[v0, v1], …,
[v3, v4]]
```

Another way of describing a geometry consisting or vertices and edges is a graph. Graphs are often described by adjacency matrices (Stevanović, 2014, Szabo, 2015). Adjacency matrixes describe whether two vertices are connected by an edge (adjacent) or not. The values that make up the matrix are 0 or 1. In adjacency matrices with linear edges and no self-loops, the diagonal only has 0s. Additionally, adjacency matrices for graphs with non-directional edges (undirected graphs) are symmetric (Wolfram MathWorld, n.d.). An example of an adjacency matrix is shown in figure 2.2.4.

Vertex	Coordinates
v0	0,0,1
v1	1,0,1
v2	1,1,1
v3	0,1,1,
v4	.5,.5,0

Edge	Vertices
e0	v0 v1
e1	v1 v2
e2	v2 v3
e3	v3 v0
e4	v0 v4
e5	v1 v4
e6	v2 v4
e7	v3 v4

Figure 2.2.3. A vertex and edge table describing the example geometry.
Own work.

Vertex	Coordinates
v0	0,0,1
v1	1,0,1
v2	1,1,1
v3	0,1,1,
v4	.5,.5,0

	v0	v1	v2	v3	v4
v0	0	1	0	1	1
v1	1	0	1	0	1
v2	0	1	0	1	1
v3	1	0	1	0	1
v4	1	1	1	1	0

Figure 2.2.4. A vertex table and adjacency matrix describing the example geometry.
Own work.

2.2.3. MESHES

A (polygon) mesh is another way of computationally representing a geometry. They consist of vertices, edges and, unlike wire-frame models, faces. Similarly to the wire-frame model, vertices have coordinates in a 3D space and edges describe the connection between two vertices. A closed set of edges forms a polygonal face. There are several ways in which meshes can be described. In this section, the following four types are discussed: (1) vertex-vertex meshes, (2) face-vertex meshes, (3) winged-edge meshes and (4) half-edge meshes (Axom, n.d., Wikipedia, 2023).

Vertex-vertex Meshes

Vertex-vertex (VV) meshes only represent vertices and their connections between them (figure 2.2.5). It is the simplest method to represent a geometry (Axom, n.d., Wikipedia,

2023). In this method, information on edges and vertices is not explicitly described. For this reason, this method is not considered suitable for applications where operations on edges and faces are of importance. The simplicity of this method, however, also means that the storage space needed for this data is low, and processing operations for morphing the shape is easy.

Vertex	Coordinates	Vertices
v0	0,0,1	v1 v3 v4
v1	1,0,1	v0 v2 v4
v2	1,1,1	v1 v3 v4
v3	0,1,1,	v2 v0 v4
v4	.5,.5,0	v0 v1 v2 v3

Figure 2.2.5. A vertex table describing the example geometry.
Own work.

Face-vertex Meshes

Face-vertex meshes is the most widely used method (Axom, n.d., Wikipedia, 2023). In this method, two lists of information are used to describe geometry (figure 2.2.6). The first is the face list, describing each face by listing the vertices that define it. In the vertex list, the coordinates (x, y, z) of each vertex and the faces that it is a part of. This method describes faces and vertices explicitly, which makes it easier to find faces that connect to each other. However, this is not the case for edges, which are not explicitly defined in this method. Certain mesh modifications, like the splitting or merging of faces, are difficult with this method.

Vertex	Coordinates	Faces
v0	0,0,1	f0 f1 f4
v1	1,0,1	f0 f1 f2
v2	1,1,1	f0 f2 f3
v3	0,1,1,	f0 f3 f4
v4	.5,.5,0	f1 f2 f3 f4

Face	Vertices
f0	v0 v1 v2 v3
f1	v0 v1 v4
f2	v1 v2 v4
f3	v2 v3 v4
f4	v3 v0 v4

Figure 2.2.6. A vertex and face table describing the example geometry.
Own work.

Winged-edge Meshes

With this method, introduced by Baumgart (Baumgart, 1972), three list types are used. Besides face and vertex lists, an edge list is also introduced (figure 2.2.7). This makes the meshes defined by this method easiest to dynamically modify, making it useful for modelling software. However, this method also requires the largest storage. The edge list is a table that consist of three columns. The first contains the two vertices it connects, the second holds the two faces it is linked to, and finally, the third column contains four of the outgoing edges from each of the vertices of the edge. These four edges is what give the winged-edge mesh representation method its name. Even though only four outgoing edges are noted in the edge list, there can be more outgoing edges. These are not described explicitly, however.

Vertex	Coordinates	Edges
v0	0,0,1	e0 e3 e4
v1	1,0,1	e0 e1 e5
v2	1,1,1	e1 e2 e6
v3	0,1,1,	e2 e3 e7
v4	.5,.5,0	e4 e5 e6 e7

Face	Edges
f0	e0 e1 e2 e3
f1	e0 e4 e5
f2	e1 e5 e6
f3	e2 e6 e7
f4	e3 e4 e7

Edge	Vertices	Faces	Winged Edges
e0	v0 v1	f0 f1	e3 e4 e1 e5
e1	v1 v2	f0 f2	e0 e5 e2 e6
e2	v2 v3	f0 f3	e1 e6 e3 e7
e3	v3 v0	f0 f4	e2 e7 e0 e4
e4	v0 v4	f1 f4	e0 e3 e5 e7
e5	v1 v4	f1 f2	e0 e1 e4 e6
e6	v2 v4	f2 f3	e1 e2 e5 e7
e7	v3 v4	f3 f4	e2 e3 e6 e4

Figure 2.2.7. A vertex, face and edge table describing the example geometry.
Own work.

Half-Edge Meshes

In addition to the previously mentioned representation methods, the half-edge representation uses an element called half-edges (Berkeley, n.d.). This element makes is easier to navigate the mesh. Each edge consist of two half edges which are oriented in

opposite directions. The opposite of a half edge is referred to as its twin. In this method, a face is defined by the edges that point in the counter clockwise (CCW) direction. The data linked to each half edge is its twin, the next CCW half edge, the source vertex and the edge and face it is associated with. This is shown in figure 2.2.8.

Mesh Representation in this study
For the purposes of this study, faces do not need to be taken into account. Connectivity and/ or vertex positioning do need to be defined, as the placement of vertices and the length of the edges in which this results are the essential aspects of the geometry in this study.

Vertex	Coordinates	Edges
v0	0,0,1	e0 e3 e4
v1	1,0,1	e0 e1 e5
v2	1,1,1	e1 e2 e6
v3	0,1,1,	e2 e3 e7
v4	.5,.5,0	e4 e5 e6 e7

Half Edge	Twin	Next	Source Vertex	Edge	Face
e0,1	e0,2	e1,1	v0	e0	f0
e0,2	e0,1	e4,1	v1	e0	f1
e1,1	e1,2	e2,1	v1	e1	f0
e1,2	e1,1	e5,1	v2	e1	f2
e2,1	e2,2	e3,1	v2	e2	f0
e2,2	e2,1	e6,1	v3	e2	f3
e3,1	e3,2	e0,1	v3	e3	f0
e3,2	e3,1	e7,1	v0	e3	f4
e4,1	e4,2	e5,2	v0	e4	f1
e4,2	e4,1	e3,2	v4	e4	f4
e5,1	e5,2	e6,2	v1	e5	f2
e5,2	e5,1	e0,2	v4	e5	f1
e6,1	e6,2	e7,2	v2	e6	f3
e6,2	e6,1	e5,2	v4	e6	f2
e7,1	e7,2	e4,2	v3	e7	f4
e7,2	e7,1	e2,2	v4	e7	f3

Figure 2.2.8. A vertex and half edge table describing the example geometry.
Own work.

2.3 CASE-STUDIES & EXAMPLES

In this section, recent examples of generation of (optimized) geometry are discussed. These works focus on relevant aspects to the framework in this study. In these examples, various types of generative deep learning models are applied. These models have been discussed in previous sections of this literature review.

2.3.1 DEEP GENERATIVE DESIGN: A DEEP LEARNING FRAMEWORK FOR OPTIMIZED SHELL STRUCTURES

Master thesis by Pavlidou, S. (2022).

In this master thesis, an AI based generative framework is designed for design exploration and optimization of a 2D mesh, representing a shell structure. This framework consist of similar steps as mentioned in previous sections: a Variational Autoencoder (VAE), FEM Structural Analysis and a surrogate model are used. In this research, the starting point is a control mesh. This mesh is then altered through subdivision of the mesh. This is done by merging of vertices. With these methods, a training dataset is generated. All meshes in this set where given a score based on structural performance and mass, found through FEM simulation.

A VAE is then trained with this dataset. It was successful in encoding and decoding original samples, and also capable of producing new generated samples. A surrogate model was then trained to predict the performance of data output by this VAE. Through gradient descent optimization, optimized meshes could be found. The results showed that generation of novel design solutions that outperform those in the original training dataset is possible with such a framework.

2.3.2 DESIGN OF TRUSS STRUCTURES THROUGH REUSE

Article in Journal 'Structures' by Brütting, J., Desruelle, J., Senatore, G., & Fivet, C. (2019).

This study focusses on the reuse of steel elements in new truss structures. The approach here consists of two steps. The first is optimization of the element assignment within a truss topology and the second is geometry optimization to reduce cut-off waste. The properties

of the stock that are taken into account in this study are parameters related to structural performance, length and availability. A assignment matrix is used to represent the selected topologies. These topologies are then structurally analysed and optimized. This optimization is focussed on two parameters: (1) the minimization of structural mass and (2) the maximization of the match between the structural use of an element compared to its structural capacity. This structure was successful in generating trusses that scored well in these parameters, however, it was stated that searching for the optimum outcomes within the solution space.

2.3.3 DEEP GENERATIVE DESIGN: INTEGRATION OF TOPOLOGY OPTIMIZATION AND GENERATIVE MODELS

Manuscript submitted to Journal of Mechanical Design by Oh, S., Jung, Y., Kim, S., Lee, I., Kang, N., Oh, K., & Jung, Y. (2018).

Finally, this is a study into the effectiveness of AI and deep learning within generative design frameworks. It also integrates topology optimization into the proposed framework of this study. The selected type of deep learning utilized here is that of a generative adversarial network (GAN). This framework is used in a case study for the design of a 2D wheel. It generated examples that performed well on the three selected parameters: (1) aesthetic quality, (2) diversity and (3) robustness. It was successful in generating new, well-performing designs.

2.4 NODE CONNECTIONS

The generated structural configurations presented in this thesis result in complex connections between its structural members. For the majority of the generated structures, Within the created framework, however, connection design were not explicitly taken into account. Therefore, connection types that could suit the designed outputs are discussed in this section.

2.4.1 TRADITIONAL CONNECTIONS

Suitable connection types depend on truss element cross sections. Figure 2.4.1 shows typical cross sections chosen for truss members. For exterior or exposed trusses, hollow sections are often chosen (SteelConstruction, n.d.).

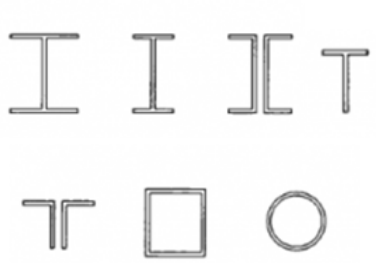


Figure 2.4.1. Typical cross sections for truss members.
Edited from SteelConstruction. (n.d.).
Trusses. https://www.steelconstruction.info/Trusses.

Bolted or welded connections are commonly used for joining structural (truss) components. Historically, riveted connections have also been applied. In these connection designs, secondary gusset or connection plates are typically introduced (Kosky et al., 2012). Gusset plates are usually rectangular or triangular in shape and constructed of steel. Gusset plates connections, both bolted or welded, are mainly used in connecting beams and girders to columns and in truss structures. Other types of connection plates that are commonly used in simple connections include end plates, fin plates, splices and bracing connections to gusset plates.

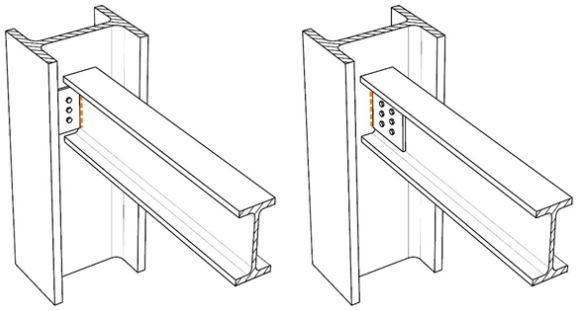


Figure 2.4.2. Examples of (left) end plate and (right) fin plate connections. The orange line indicates a weld.
Own work.

Flexible end plate connection (figure 2.4.2, left) are plates typically welded to a support member.

These plates can be either full or partially cover the depth of this member. In contrast, a fin plate (figure 2.4.2, right) is welded to the supporting member and bolted to the beam. In both cases, the welded connection can be performed before assembly in a workshop (prefab). These connection types are inexpensive and popular for use in simple beam-column connections.

Column splices (figure 2.4.3, left) are used to connect extending members oriented in the same direction as the original member. They can be used for both I-beams and hollow sections. Bracing connections are more suitable for angled connections, as shown in figure 2.4.3 (right). Bracing members are most commonly connected to other members using gusset plates, which in turn are welded to the supporting member(s).

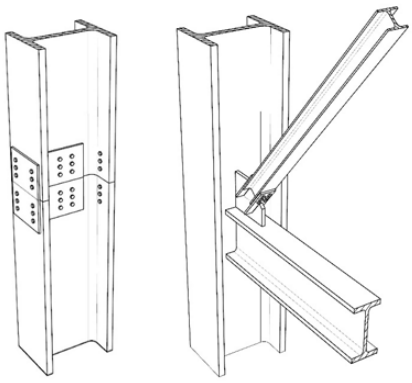


Figure 2.4.3. Examples connection with (left) splices and (right) bracing connections with gusset plates.
Own work.

The examples given above show simple form connections. However, these connection types can also be used in more complicated connection design. Examples of this are shown in figure 2.4.4. Through use of such connecting components, a variety of cross sections may be combined in one node, which can be fitting for the construction of the nodes of the generated structures.

Another joint type, especially suitable for 3D connections of multiple members is the bolt ball joint, shown in figure 2.4.5 (SAFS Steel Structure, 2021). While the manufacturing cost is higher than that of most other connection types, its assembly cost is lower. This

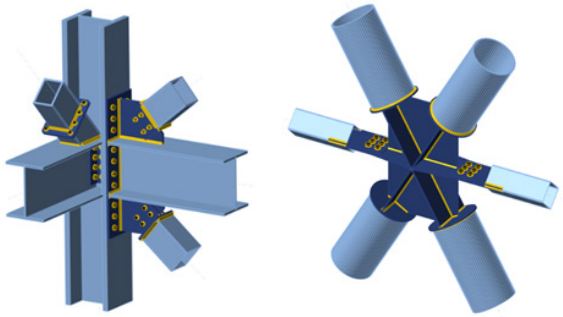


Figure 2.4.4. Examples of complex connection with end plates, fin plates, splices and braces.
Retrieved from IDEAStatiCa. (n.d.).
Sample Projects. https://www.ideastatica.com/nl/support-center-sample-projects?product=steel&label=connection&page=3&scroll=true

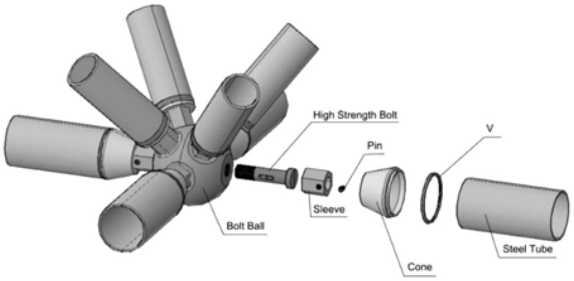


Figure 2.4.5. An example of a bolt ball connection.
Retrieved from SAFS Steel Structure. (2021).
What are the General Types of Space Frame Nodes? What are the Characteristics? https://www.safsteelstructure.com/news/what-are-the-general-types-of-space-frame-nodes-what-are-the-characteristics-of-each/

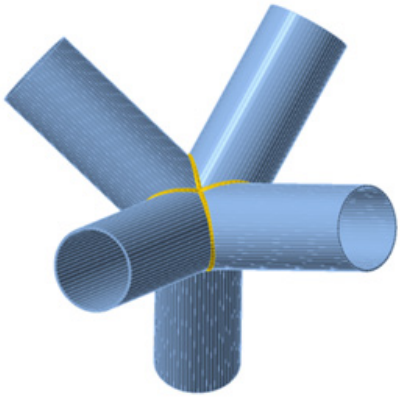


Figure 2.4.6. An example of a fully welded 3D connection.
Retrieved from IDEAStatiCa. (n.d.).
Break the limits of your steel designs. https://www.ideastatica.com/steel.

connection type is also suitable for disassembly and reassembly. As a result, this node design is consistent with the concepts of reuse and circularity presented in this thesis. It does, however, limit the cross sections of the members that can be utilised.

Finally, connections can be fully welded (figure 2.4.6). This includes multi-planar and 3D welded connections, like those in the generated trusses (Packer, 2018). These connections are mainly used for members with circular or rectangular hollow sections. Construction of such connections, however, is complex and offsets are not tolerated well.

2.4.2 STEEL CAST NODE DESIGN

The main advantages of welded cast steel joints or nodes are their aesthetical quality, as they are minimally intrusive and offer smooth transitions, usually between steel structural members with a circular cross section. Cast nodes look similar to the previously described fully welded connections. The difference, however, is that the welded seams are further away from the intersection core. This results in a lower weld stress (Wang et al., 2013, Wei et al., 2021). Using steel casting to produce the nodes of the generated geometry in this thesis, however, would not be ideal, as each node is unique in shape, necessitating the creation of unique, one-time use casts.

2.4.3 ADDITIVE MANUFACTURED NODE DESIGN

Additive Manufacturing (AM) offers a different approach to steel node design. Even though these techniques have yet to be fully integrated into the construction industry, they are viewed as promising and their potential is researched (Feucht et al., 2019, Lange et al., 2020). Using Wire Arc Additive Manufacturing (WAAM), structural and connection elements can be printed directly onto structural steel members. (Feucht et al., 2019) A possible application of AM steel components is to create topology-optimized versions of fin and end plates, stiffeners, and braces, resulting in minimal material use (Lange et al., 2020). This approach can also be used to create whole, custom nodes with high accuracy. However, Steel AM does come with a high production and equipment cost. Nonetheless, the wide range of forms that can be produced through this method makes it a suitable approach for the nodes of the generated geometries.

2.5.4 CONCLUSION

There are various ways in which steel elements can be joined. The applicability of these methods depends mainly on the element profile and the function of the structure. While all joints mentioned in this section are applicable to the case study, connections through end plates, fin plates, splices and braces work well for steel elements of varying profiles. For O-profiles, a ball bolt joint, steel cast nodes or simple welded connections are also suitable. The customizability of additive manufactured nodes make this a very versatile option, as it is possible to create unique shapes. However, the high production cost of this method should be taken into account.

3. Dataset Generation

3 Dataset Generation

For this thesis, a dataset containing training geometry needs to be created. Secondly, a material stock needs to be created. The generation of these two datasets is presented in the following section. Currently, however, this process is not yet complete.

3.1 GEOMETRY DATASET

The geometry dataset should consist of a set of vertices and their corresponding coordinates. Edge lengths should also be calculated. The form of mesh representation does not need to be generated, as this is the same for the entire structure. The geometry dataset generation consists of two steps:

1. Definition of a parametrically adjustable model
2. Generation of variations based on this model
3. Writing variation information into a file

For generation and processing of the geometry dataset, Python is used within Grasshopper/ Rhinoceros 7 is used. The flowchart in figure 3.1.12 shows how the dataset is generated. The full code can be found in Appendix II.

3.1.1 MESH MODIFICATION & GENERATION

The base mesh, as described in previous chapters, is parametrically defined. This allows for easy modification, as is necessary to generate the geometry for the input dataset. In this sections, rules and assumptions made for the intended modifications are described. The goal is for the lengths of the linear elements to vary. This way, meshes that fit the stock can be created. Figure 3.3.1 shows how the base mesh should be alterable for generation of suitable meshes.

To define these modifications, the coordinates of vertices in the base mesh should be altered. To achieve this, the mesh is divided into three layers: the upper quadrilateral grid, the lower quadrilateral grid and the in-between triangular grid, as done in figure 3.1.2. In the following sections, modifications of these layers are discussed.

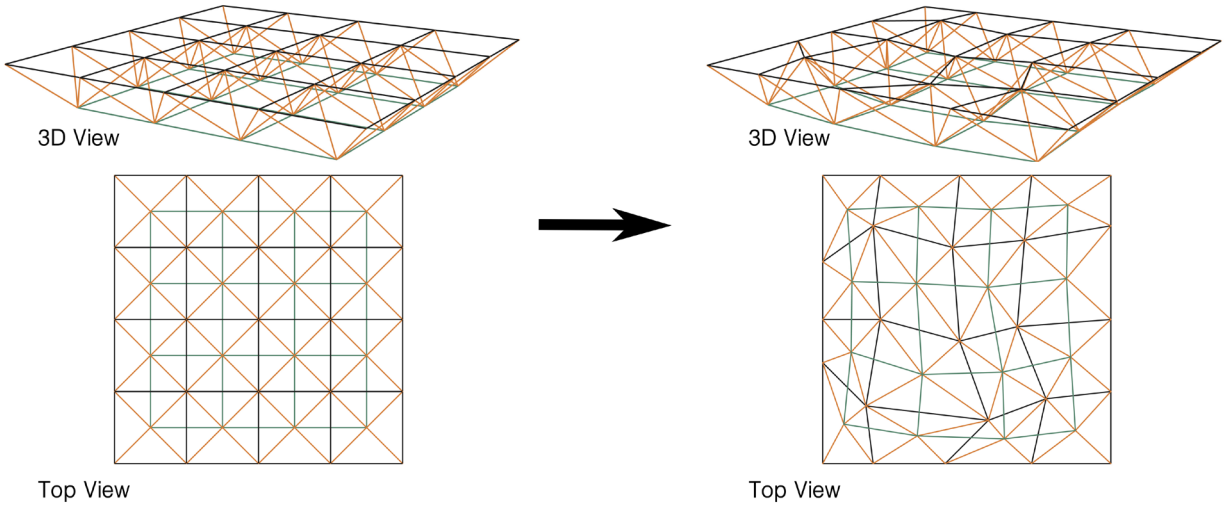


Figure 3.1.1. The base mesh & an example of a generated mesh.
Own work.

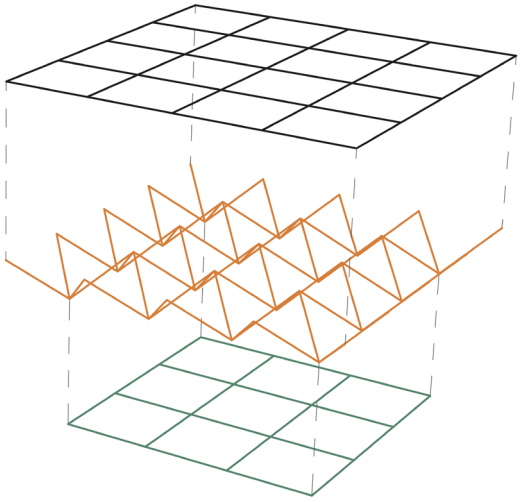


Figure 3.1.2. The base mesh split up in three layers.
Own work.

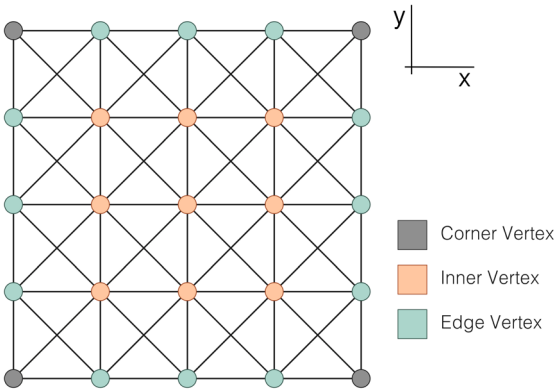


Figure 3.1.3. Base mesh. Vertices on the upper layer are highlighted.
Own work.

Top layer vertices

Modifying the mesh starts by changing the locations of the vertices of the top layer. Three vertex categories are defined on the top layer of the grid: corner vertices, inner vertices and edge vertices (figure 3.1.3)

To get generated geometries that fit within the intended parameters, the translation space of each of these vertices must be limited(figure 3.1.4). It is assumed that none of the top vertices are able to move in the z (height) direction, to assure that generated meshes can all support the same flat roof. To ensure that the total area that the truss structure it represents remains at an equal value, the outer edge of the mesh must remain the same as well. So, corner can move in neither the x or y direction and edge vertices can move in both the positive and negative direction of either x or y, as shown in figure X. Finally, inner vertices can move in the positive and negative direction of both x and y.

These translations should be limited in magnitude. Firstly, the order of the vertices should remain the same, so that the produced mesh definitions can easily be reconstructed properly. For example, vertices being assigned the same coordinates must be avoided. This would be the case if two neighbouring vertices would be generated to move 50% in opposite directions. This is undesirable, as this results in the edge length to equal zero and therefore a change in the overall shape of the mesh. The maximum magnitude of vertex translations should therefore always be less than 50% of the element length. This ensures that the shape of the top layer remains a quadrilateral grid, which is vital for this study, as not using such limitations could result in the generation of meshes that are too complex for the VAE model to learn to properly generate. So, the transformation of each vertex should be limited to less than 50% of the edge length. With these movements, the inner structure can be generated differently, while the outer edge of the mesh remains the same.

Vertex type	Vertices	Directions	Generated values
Corner	4	0	0
Edge	12	1	12
Inner	9	2	18
TOTAL	**25**		**30**

Figure 3.1.5. Number of vertices, directions these can move in and the generate directional values required.
Own work.

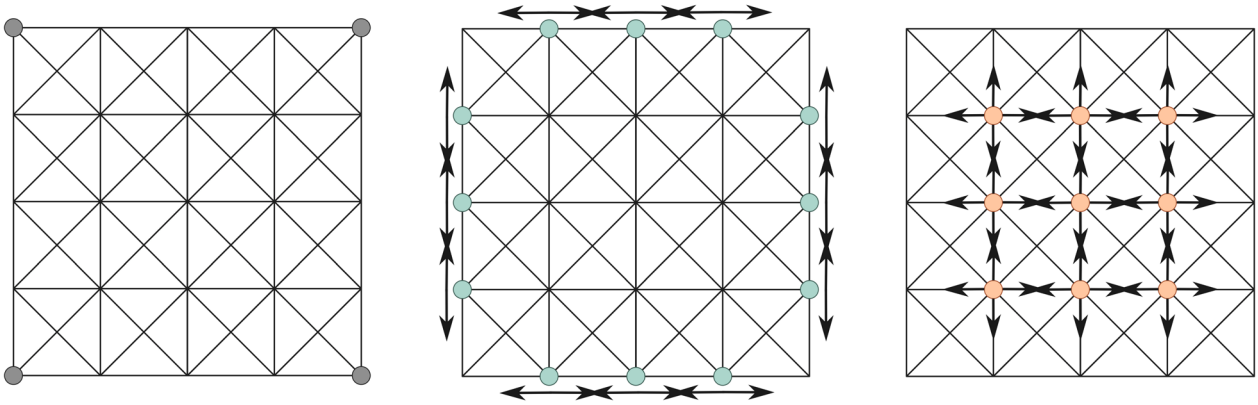


Figure 3.1.4. Vertex movement per category.
Own work.

Next, the translation needs to be defined numerically. This can be done by randomly generating values for the x and y translations of each vertex. The total number of vertices in this mesh is twenty-five. Of this, four are corner vertices, twelve are edge vertices and nine are inner vertices. Through multiplication of these numbers by the number of directions they can move in, it can be concluded that thirty unique values need to be generated (figure 3.1.5).
These randomized values can be used directly for mesh generation. However, this may cause the generator to create meshes that are almost, but not completely identical. These meshes will therefore not be easily filtered out when checking for duplicates. So, to ensure variety among the dataset samples, it is better to define a limited set of coordinates to which each vertex can move. The size of this set should be big enough to generate a large enough dataset, but not so big that meshes may become too similar and can be defined as the number of divisions taken from the translation range in the x and y direction (figure 3.1.6). To ensure that the original position of the vertex remains a possible an option for the vertex to be in, the number of divisions should be even.

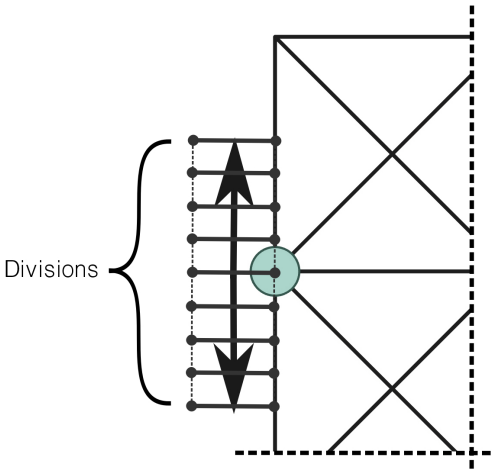


Figure 3.1.6. Divisions of the translation range. In this example, eight divisions are shown.
Own work.

To determine the total potential training set size, the total number of possible meshes is calculated for four, six, eight and ten divisions.

	Vertices	Directions	Generated Values	4 Divisions		6 Divisions		8 Divisions		10 Divisions	
				Options per vertex	Total options	Options per vertex	Total options	Options per vertex	Total options	Options per vertex	Total options
Corner	4	0	0	0	0	0	0	0	0	0	0
Edge	12	1	12	3	531,441	5	244,140,625	7	13,841,287,201	9	282,429,536,481
Inner	9	2	18	5	1,953,125	9	387,420,489	13	10,604,499,373	17	118,587,876,497
Total	25		30		1.04E+12		9.46E+16		1.47E+20		3.35E+22

Figure 3.1.7. Divisions and corresponding generation options.
Own work.

As mentioned previously, translation must be limited to avoid overlapping vertices within the mesh. As a result, the first and last option must be removed. So, using eight divisions on an edge vertex, as shown in figure 3.1.6, would result in seven location options for this vertex. The results of this are shown in 3.1.7.

Lower layer vertices

Next, the vertices on the lower layer should be addressed. These vertices are highlighted in figure 3.1.8. For the structure to be efficient, these vertices should remain within the cell created by the four surrounding top layer vertices. Therefore, placement of these vertices of the lower layer are partially defined by the upper vertex coordinates.

To allow for translation in x and y directions, the boundaries of the cells created by the upper layer are defined. Next, these are scaled down by 50%, based on the cells centre points. This is done, to avoid unrealistic geometries where the lower vertices are placed too close to the surrounding top edges. The lower vertices are then placed within these cells with the "Populate 2D" command in Grasshopper. This command populates each cell with one vertex. Figure 3.1.9 shows the area in which the vertices can be placed in blue. A possible vertex distribution marked in orange.

Finally, the translation in the z direction of these vertices need to be defined. In the base mesh, these meshes have a z-coordinate -1 compared to the top layer. This means that the translation of these points in the z direction can be -1 to 1, equal to the vertices of the top layer. These movements are shown in figure 3.1.10. These translations, like those of the top layer vertices, should be limited to ensure variety within the dataset. Therefore, the same number of distributions is used.

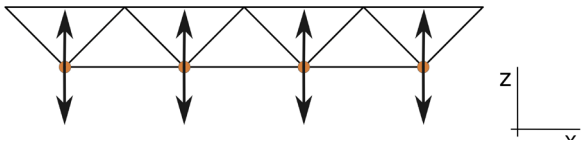


Figure 3.1.10. Base Mesh. Movement of lower vertices in z-direction.
Own work.

3.1.2 INITIAL DATASET GENERATION

Dataset generation is based on randomized inputs. Per generated dataset, three random values between 0 and 10,000 are used. These values are used as:

- "seed" input for random generation of thirty (30) values representing x,y-translation of

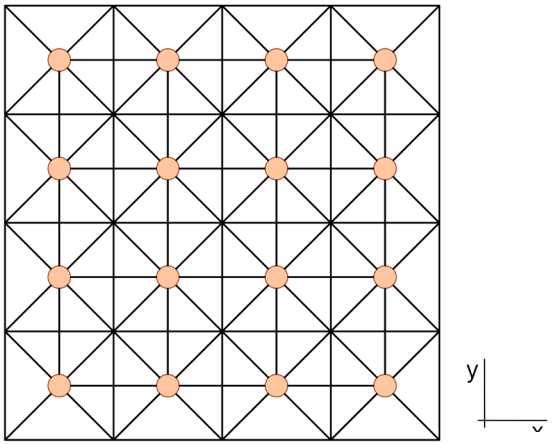


Figure 3.1.8. Base mesh. Vertices on the lower layer are highlighted in orange.
Own work.

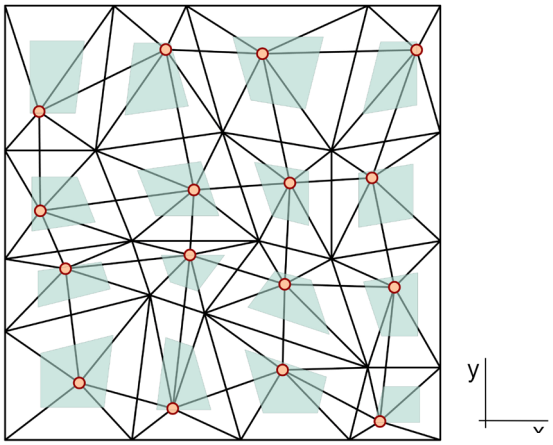


Figure 3.1.9. Example of a generated mesh. The area in which lower vertices can be placed is shown in blue. A set of lower vertices is shown in orange.
Own work.

vertices on the upper layer;

- "seed" input for random generation of sixteen (16) values representing z-translation of vertices;
- "seed" input for "Populate2D" Grasshopper component, representing x,y-translation of vertices on the lower layer.

The calculation translations are applied to the respective vertices to find the altered vertex coordinates. Though implementation of a GhPython script, the coordinates and vertex indices of each generated mesh are written to a .csv file. For the first generated mesh, an adjacency matrix is generated as well. This matrix (figure 3.1.11) is the same for all samples and is essential for correct recreation of the meshes. As the matrix is symmetric, only half needs to be used for the reconstruction of the meshes. The full Grasshopper and Python scripts can be found in Appendix II. The dataset was designed to generate at least 10,000 samples. The generator attempted to generate 10,048 meshes.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0
8	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
10	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
11	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0
12	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0
13	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
25	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
26	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
27	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0
28	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
29	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0
31	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0
32	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0
34	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0

Figure 3.1.11. Adjacency Matrix of the generated geometries.
Own work.

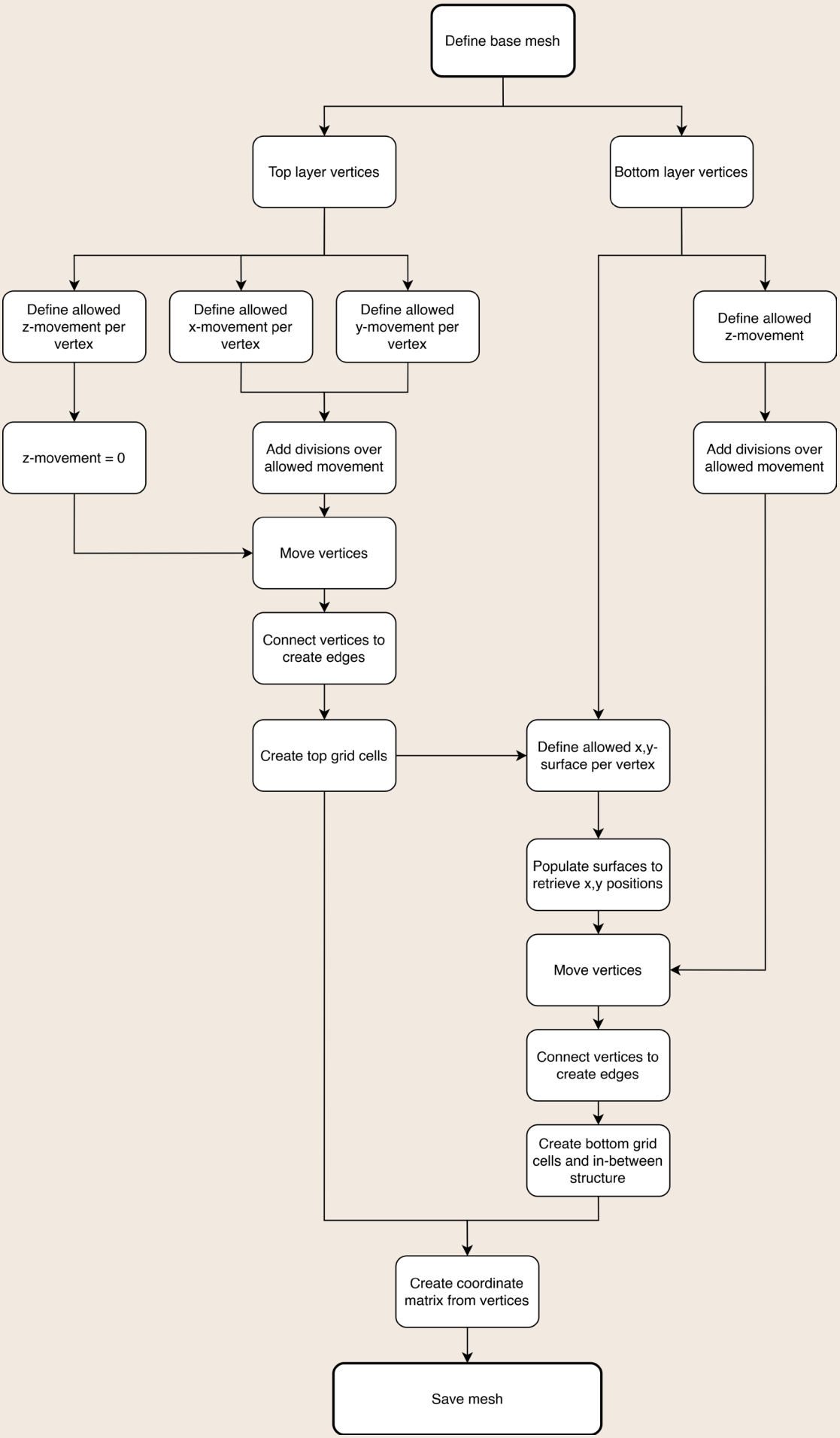


Figure 3.1.12. A flowchart for geometry generation for the dataset.
Own work.

3.2 PERFORMANCE SCORE

Each mesh is assessed on its performance. There are three aspects that are taken into account. In order in which they are addressed in this section, these are: (1) material use, (2) similarity to the stock library and (3) structural performance. A higher score means better performance. Each of the scores are normalised to allow for comparison and calculation of the overall performance score. This is done using the following equation:

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

Where,

- z_i = normalised performance indicator for mesh i
- x_i = data for mesh i
- $\min(x)$ = minimum of all data items
- $\max(x)$ = maximum of all data items

The Python code used for calculating these performance scores can be found in Appendix II.

3.2.1 MATERIAL USE

The material use is defined by the total lengths of all elements. This data is then normalised, resulting in a score between 0 and 1 for each mesh. The material use score should be higher when less material is used. As a result, the normalised data is inverted:

$$p_{material\ use_i} = 1 - z_{material\ use_i}$$

Where,

- $p_{material\ use_i}$ = normalised inverted material use performance indicator for mesh i
- $z_{material\ use_i}$ = normalised material use performance indicator for mesh i

3.2.2 MATERIAL STOCK LIBRARY

The material stock is an essential aspect for calculating the overall performance score. It contains a series of lengths, derived from a set number of steel truss beams. For these truss structures, two types were considered: flat truss beams and A-frame truss beams (figure 3.2.1). A third option; to assemble the library based on randomly generated numbers between set values, was also considered. The main difference between this third option compared to those previously mentioned is that the use of a set of trusses represents the case where the new structure is created from items of one demolition project, whereas a randomized library is a more accurate representation of a mixed stock. The minimal size of the material stock should be 128, as this is the number of linear elements in the geometry representing the truss design. A higher number, however, would be preferable, as this more closely resembles a realistic case study.

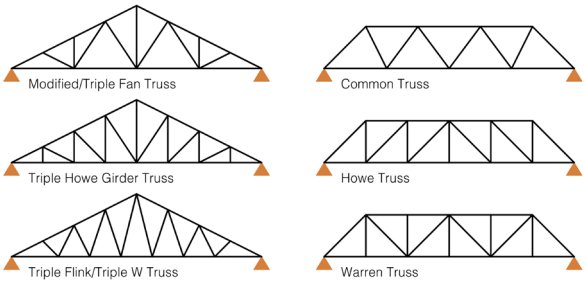


Figure 3.2.1. Examples of beam truss types that can be used as a basis for the stock library.
Own work.

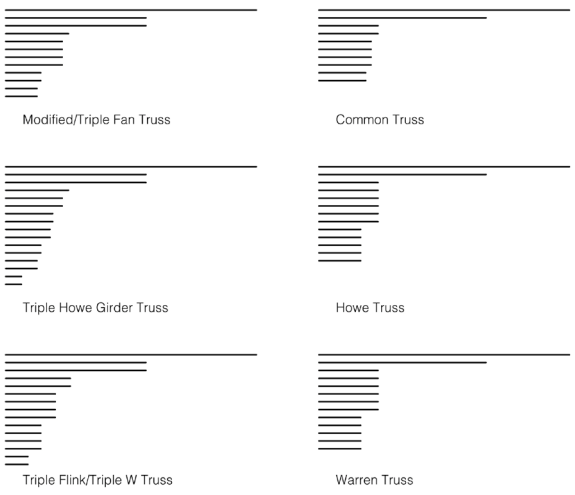


Figure 3.2.2. Examples of beam truss types that can be used as a basis for the stock library, separated into linear elements.
Own work.

For the first two options, the chosen type of truss determines the variety of items in the stock. Figure 3.2.2 shows the elements present in each of the example beam trusses as described in figure 3.2.1. While flat truss beams, as shown on the right, result in a stock with a four to five different element lengths, A frame trusses result in a larger number of different element lengths. Details on the generation of the stock library with the described methods are given later in this chapter.

The stock consist of a series of lengths. The chosen stock should reflective of the current availability of re-useable stock. For this thesis, three libraries were generated, based on the methods described previously. The first was based on a set of flat truss beams, the second on a set of a-frame truss beams and the third on randomly generated values.

To determine the size of the trusses used in the stock library, the average element length of a randomly chosen meshes was calculated. The stock library was defined so that the average element length of the chosen meshes approximately matched the average element length of the stock library. This was done to ensure suitability of the stock library elements, without using the average of all meshes, as this would result in an unrealistic case study.

Library size (number of elements in the stock library) was based on the total lengths of the elements in a mesh. The total length of the elements present in the stock library were set to be:

total length of stock library elements ≥ 1,2 * total length of basemesh elements

The total element length of the base mesh is: 238.851252 m, which means that the total length of the elements in the stock library should equal at least: 286.621502 m. This result is smaller than the total length of mesh elements for some meshes. This is reflected in their 'similarity to stock'-performance score.

Additionally, the average element length was aimed to match roughly 1.95 m. This number was found by calculating the average element length for five randomly chosen meshes (seeds 573, 2025, 5140, 8742 & 10033). This was done to ensure suitability of the stock library elements, without using the average of all meshes, as this would result in an unrealistic case study.

So, the following assumptions are made for all three stock libraries: (1) the total length of the elements in the stock library should equal at least: 286.621502 m and (2) the average element length was aimed to be around 1.95 m. For data clarity, all libraries were sorted in ascending order. In the next sections, specific assumptions and relevant library properties are described.

Stock library 1: Flat truss beam based

For stock library 1 (flat truss beams) the following specific assumptions were made. Items were generated based on a Howe/Warren truss. Properties of this truss are:

- Truss element length: 1.1 m
- Number of truss elements (horizontally): 8
- Truss height: 0.9 m

This resulted in a library set with the following properties:

- Number of truss beams: 9
- Total of element lengths: 297.631227 m
- Average Element length: 1.945302 m
- Total items in stock: 153

Stock library 2: A-frame truss beam based

For stock library 2 (a-frame truss beams) the following specific assumptions were made. Items were generated based on a Howe Girder truss. Properties of this truss are:

- Truss element length: 1.1 m

- Number of truss elements (horizontally): 8
- Truss height (highest point): 1.6 m

This resulted in a library set with the following properties:
- Number of truss beams: 9
- Total of element lengths: 316.995283 m
- Average Element length: 1.956761 m
- Total items in stock: 162

Stock library 3: Randomized Data
For stock library 3 (randomized values) the following specific assumptions were made. To assume variety within the set, a ratio of 90% short items and 10% long items was selected. Short items are defined as having a length of 0.5 to 3 m. Long items are defined as having a length of 3 to 10 m. The chosen numbers of this are based on the minimal and maximal element lengths of the five randomly chosen meshes (seeds 573, 2025, 5140, 8742 & 10033).

This resulted in a library set with the following properties:
- Number of truss beams: N/A
- Total of element lengths: 288.148285 m
- Average Element length: 1.946948 m
- Total items in stock: 148

The chosen library is the second option. This library showed a good balance between limited lengths and variety. The (162) items in this library are:
- 18x 0.4 m
- 18x 0.8 m
- 36x 1.17047 m
- 18x 1.2 m
- 18x 1.360147 m
- 9x 1.6 m
- 18x 1.627882 m
- 18x 4.68188 m
- 9x 8.8 m

3.2.3 SIMILARITY ASSESSMENT

Part of the performance score, as defined previously, is a generated sample's similarity to the material stock. For this assessment, stock element length and availability were taken into account. Other properties that could be taken into account are: Young's Modulus, material strengths, material densities and cross-section areas (Brütting et al., 2019). In this case study, however, it was assumed that these properties are equal. The structural performance is tested separately.

Similarity as defined above can be assessed in multiple ways, which is discussed in this section. Similarity, in this study, is defined as the difference in lengths between elements from the generated geometry and items from the stock, calculated as a percentage. This describes how closely the generated mesh resembles the stock. In this calculation, the items in the stock should be treated as limited, so that once they are matched to an item of the generated mesh, they cannot be used again in the same mesh. The basis of such a calculation could be described by the following:

$$Similarity = \frac{\sum_{i=1}^{128} L_{stock_i} + \left| L_{stock_i} - L_{sample_i} \right|}{\sum_{i=1}^{128} L_{stock_i}}$$

Where,
- L_{sample} = Length of sample element
- L_{stock} = Length of closest matching stock element

Here, for each item in the generated mesh the closest matching item from the stock is found. The difference in length between these items is calculated. Next, the selected closest match is removed from the library. Then, the process is repeated for the next item in the generated mesh. A flaw within this method, however, is that the outcome is dependent on the sample element order. By sorting sample and stock list, this issue can be decreased.

This method also assumes that the items in the stock are used directly as they are. In reality,

however, items can be split and used for multiple sample items instead. This case should be considered when the following is true:

$$\left|L_{stock_i} - L_{sample_i}\right| + \left|L_{stock_j} - L_{sample_j}\right| > \left|L_{split_x} - L_{sample_i} - L_{sample_j}\right|$$

Where,

- L_{sample} = Length of sample element
- L_{stock} = Length of closest matching stock element
- L_{split} = Length of a stock element that can be divided into segments

But also, when:

$$\left|L_{stock_i} - L_{sample_i}\right| + \left|L_{stock_j} - L_{sample_j}\right| + \left|L_{stock_k} - L_{sample_k}\right| > \left|L_{split_x} - L_{sample_i} - L_{sample_j} - L_{sample_k}\right|$$

This would also be the case for every other number and combination of samples that could be taken from any stock item L_{split_x}. Calculating this for every possible combination is computationally heavy, so limiting the number of calculations to be done is desirable. This can be achieved by calculating the percentual difference between L_{stock_i} and L_{sample_i}, then selecting all items over a set percentage p and only calculating possible combinations with these samples. The maximum length of the combinations can be calculated by $c = L_{stock_{max}}/L_{sample_{min}}$ for samples with a percentual length difference over p. With this, all possible combinations with lengths in range 2 to c should be considered.

Example:
For list: `[0, 1, 2, 3, 4]`
With: $c = 4$

The following combinations should be found:
`[(0, 1), (0, 2), (0, 3), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4), (0, 1, 2), (0, 1, 3), (0, 1, 4), (0, 2, 3), (0, 2, 4), (0, 3, 4), (1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4), (0, 1, 2, 3), (0, 1, 2, 4), (0, 1, 3, 4), (0, 2, 3, 4), (1, 2, 3, 4)]`

For all new combinations, the library item with the closest length should be found. In this case, the full set of unassigned stock items should be considered for each combination. So, library items should not be removed from in the process.

Next, the potential ways in which the combinations can be merged to recreate the original list should be found. For this, the original list items and their respective values of ΔL should also be considered.

Example:
For list:
`[0, 1, 2]`

Where possible combinations are:
`[[0, 1], [0, 2], [1, 2], [0, 1, 2]]`

And the original elements are:
`[[0], [1], [2]]`

Possible recreation methods are:
`[([0], [1], [2]), ([0,1], [2]), ([0,2], [1]), ([1,2], [0]), ([0,1,2])]`

For each of these combination, the total ΔL is calculated to determine with combinations fit the best. This is the combinations that results in the lowest total ΔL.

So, for this study, it was assumed that elements in the framework can be created by splitting stock items. However, joining together stock libraries is not considered, as this would result in weak points in the beams and is therefore often not desirable.

Finally, a check needs to be performed to ensure that only unique stock items are used. This can be done by finding indices of the used library items and checking if duplicates are present. If

this is the case, the duplicate library item must be replaced by its second closest match (that is not yet used elsewhere in the mesh).

To calculate the similarity performance score, the total ΔL between all items in the mesh and all selected stock items is added up to the total length of all mesh elements. This number is then divided by the total length of all mesh elements, as described at the start of this section:

$$Similarity = \frac{\sum_{i=1}^{128} L_{stock_i} + \left| L_{stock_i} - L_{sample_i} \right|}{\sum_{i=1}^{128} L_{stock_i}}$$

To limit the computational power required, the items considered for 'splitting' were limited. For items with a percentual difference between L_{stock_i} and L_{sample_i} (value p) over 0.25, the use of larger, dividable stock items was considered.

As lower similarity as calculated by the method described previously indicates better performance, the performance indicator for stock similarity was calculated using the following equation:

$$p_{stock\ similarity_i} = 1 - z_{stock\ similarity_i}$$

Where,

- $p_{stock\ similarity_i}$ = normalised inverted similarity performance indicator for mesh i
- $z_{stock\ similarity_i}$ = normalised similarity performance indicator for mesh i

In Appendix II, an example of the results output by the designed algorithm is given for randomly selected mesh 234 with a value c=5.

3.2.4 STRUCTURAL PERFORMANCE

Figure 3.2.4 shows the load case that is used to test the structural performance of the truss geometry. Each of the corners is a support point, and a force is calculated on each node. This force will result from the area (m^2) of roof that is supported by each node, and the assumed roof weight.

The supported area per node is calculated by dividing each cell into four sections, based on the midpoints of each cell edge. Then, the

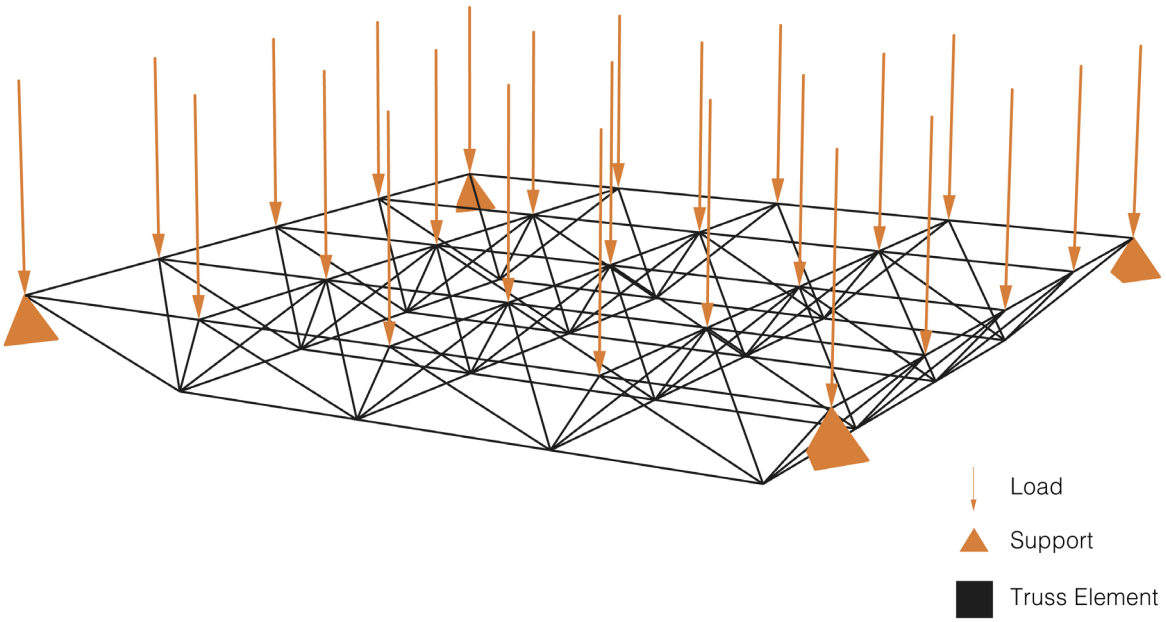


Figure 3.1.11. Adjacency Matrix of the generated geometries.
Own work.

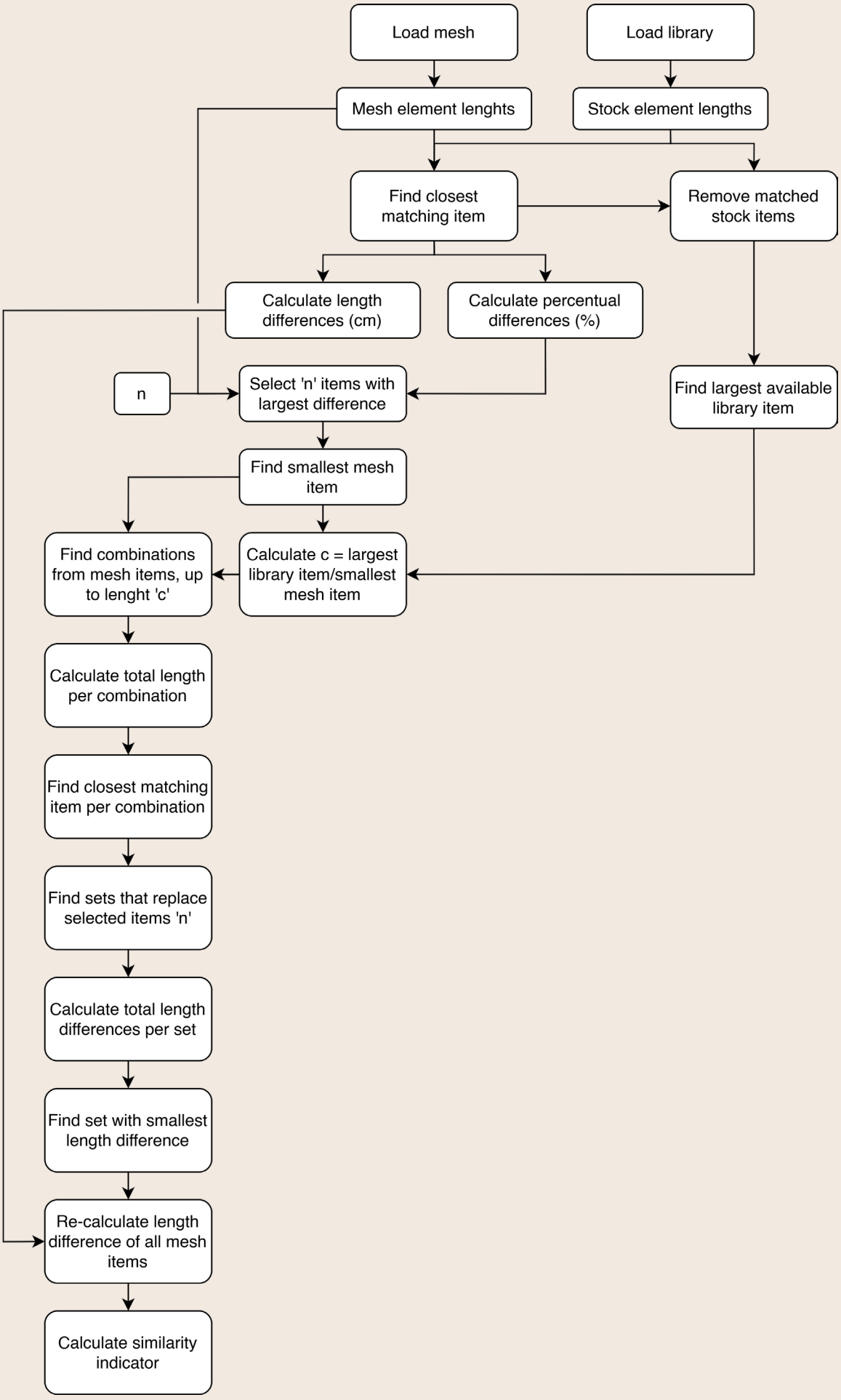


Figure 3.2.3. A flowchart for similarity performance calculation.
Own work.

divided areas are assigned to their respective vertex. Figure 3.2.5 shows an example of this for a (1) corner vertex, (2) edge vertex and (3) inner vertex.

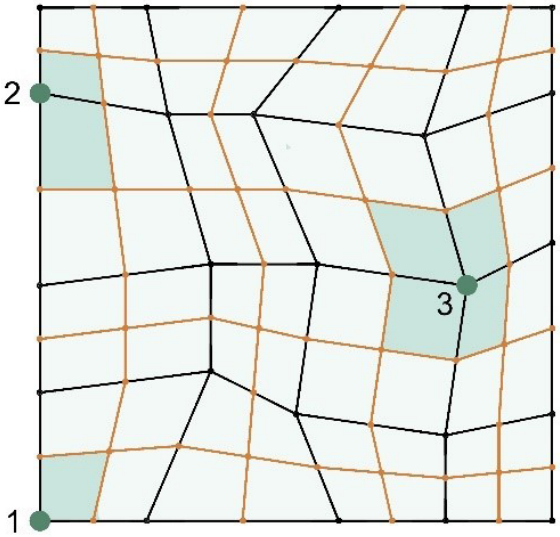


Figure 3.2.5. A diagram showing the top layer of a mesh and the calculated areas assigned to a (1) corner, (2) edge and (3) inner vertex.
Own work.

Roof weights can vary based on the types and thicknesses of used materials. Full roof systems from various sources are found to have a mass of 0.4 to 2.5 kN/m^2, the latter referring to water-holding green roof systems (Urbanscape, 2015, Sempergreen, n.d., Maguire Brothers, 2022, Eurocode, 2019). In this thesis, a self-weight of 1 kN/m^2 and a live load for accessibility and repairs of 1 kN/m^2 are assumed (Eurocode, 2019). Therefore, the total roof load used equals 2 kN/m^2. The performance score is normalised after structural calculations.

To assess structural performance, a Karamba3D framework was created within the Grasshopper environment in Rhino 7. This framework can be found in Appendix II. In this framework, a roof weight of 2 kN/m^2 was assumed. The Karamba3D simulation was used to calculate total displacement and average utilization. Good structural performance is indicated when both factors are close to zero. Therefore, each factor is normalized using the following formulas:

$$p_{displacement_i} = 1 - z_{displacement_i}$$

Where,

- $p_{displacement_i}$ = normalised inverted displacement performance indicator for mesh i
- $z_{displacement_i}$ = normalised displacement performance indicator for mesh i

$$p_{utilization_i} = 1 - z_{utilization_i}$$

Where,

- $p_{utilization_i}$ = normalised inverted utilization performance indicator for mesh i
- $z_{utilization_i}$ = normalised utilization performance indicator for mesh i

As the aim is to use items from a stock library with limited options, it is expected and allowed for utilization to be imperfect. However, it should still be optimized. Therefore, the following equation is selected for calculation of the structural performance indicator:

$$p_{structural\ performance_i} = 0.6 * p_{displacement_i} + 0.4 * p_{utilization_i}$$

3.2.5 OVERALL SCORE CALCULATION

All calculated performance scores were combined to find the overall performance indicator. This was done using the following equation:

$$p_{overall_i} = 0.2 * p_{material\ use_i} + 0.4 * p_{stock\ similarity_i} + 0.4 * p_{structural\ perfomance_i}$$

This results in a score between zero and one, in which a higher value indicates better performance.

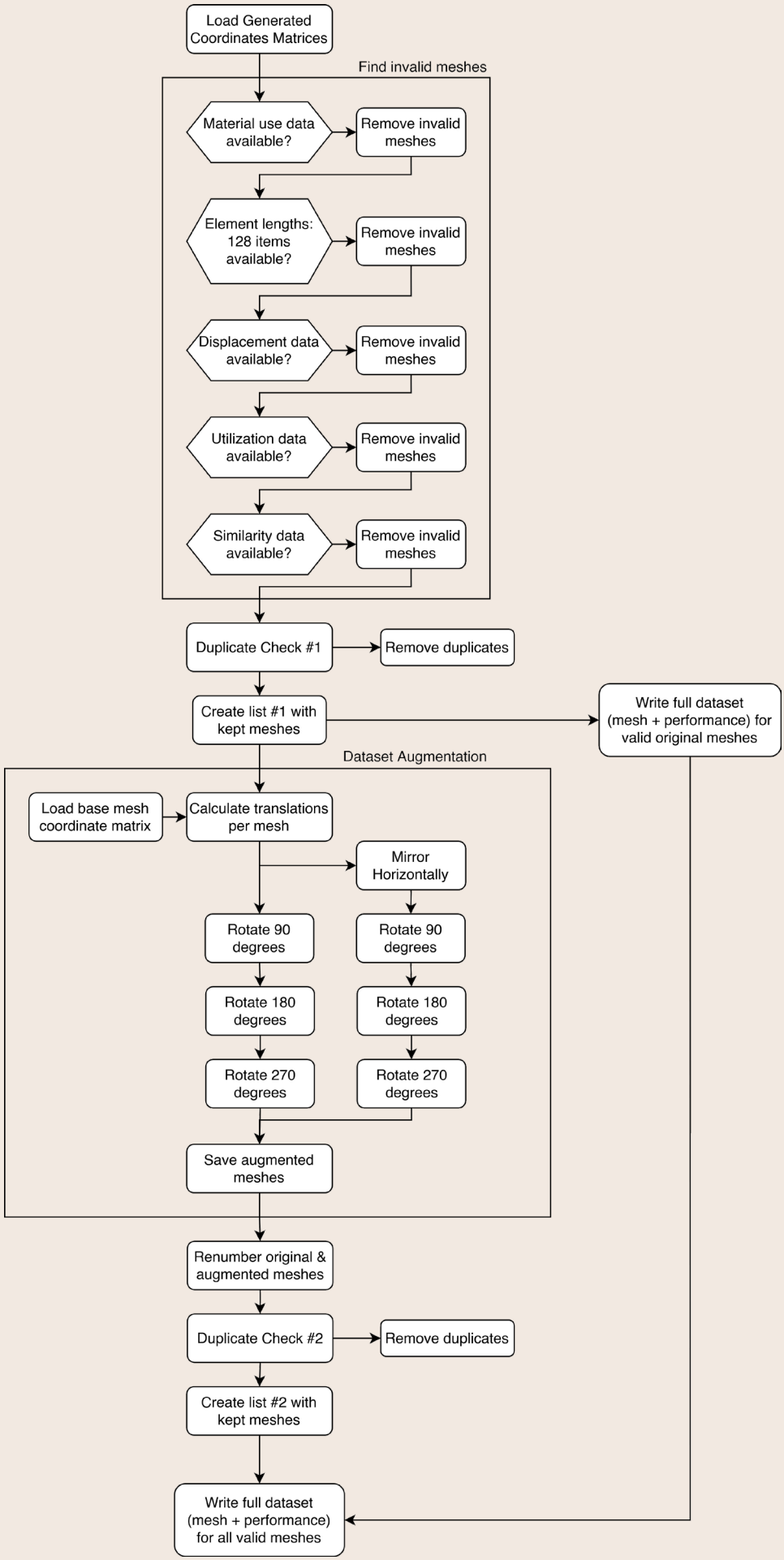


Figure 3.2.6. Flowchart showing the processing of the dataset
Own work.

3.3 PROCESSING DATASET

After all data is generated, simulated and calculated, the dataset must be filtered and processed. This process is shown in the flowchart in figure X. First, the presence and validity of all data (coordinate matrices and performance data) is checked. If one or more of this data is missing, the meshes are marked as invalid and removed. This resulted in a total of 9.703 valid meshes.

To remove potential duplicate samples, all element lengths in their specific order are compared. When duplicates are found, only the mesh with the lowest indices are kept. The code for this can be found in Appendix II. Through this method, it can be concluded that no duplicates were found.

So, all 9.703 meshes were used to normalize the performance data. This data was then written into a .csv file. In the table in figure 3.3.1, a small part of this dataset is shown. In this dataset, the following ten entries in figure 3.3.2 were found to have the highest overall performance.

The highest performing mesh has an overall score of 0.79934. The lowest score is 0.26861. The graphs in figure 3.3.3 show the distribution of the types of performance scores.

Dataset Augmentation

When generated geometries are mirrored and rotated, new samples can be created. All meshes are mirrored on the y-axis. The original and mirrored mesh are then each rotated 90°, 180° and 270°, so that a total of eight meshes

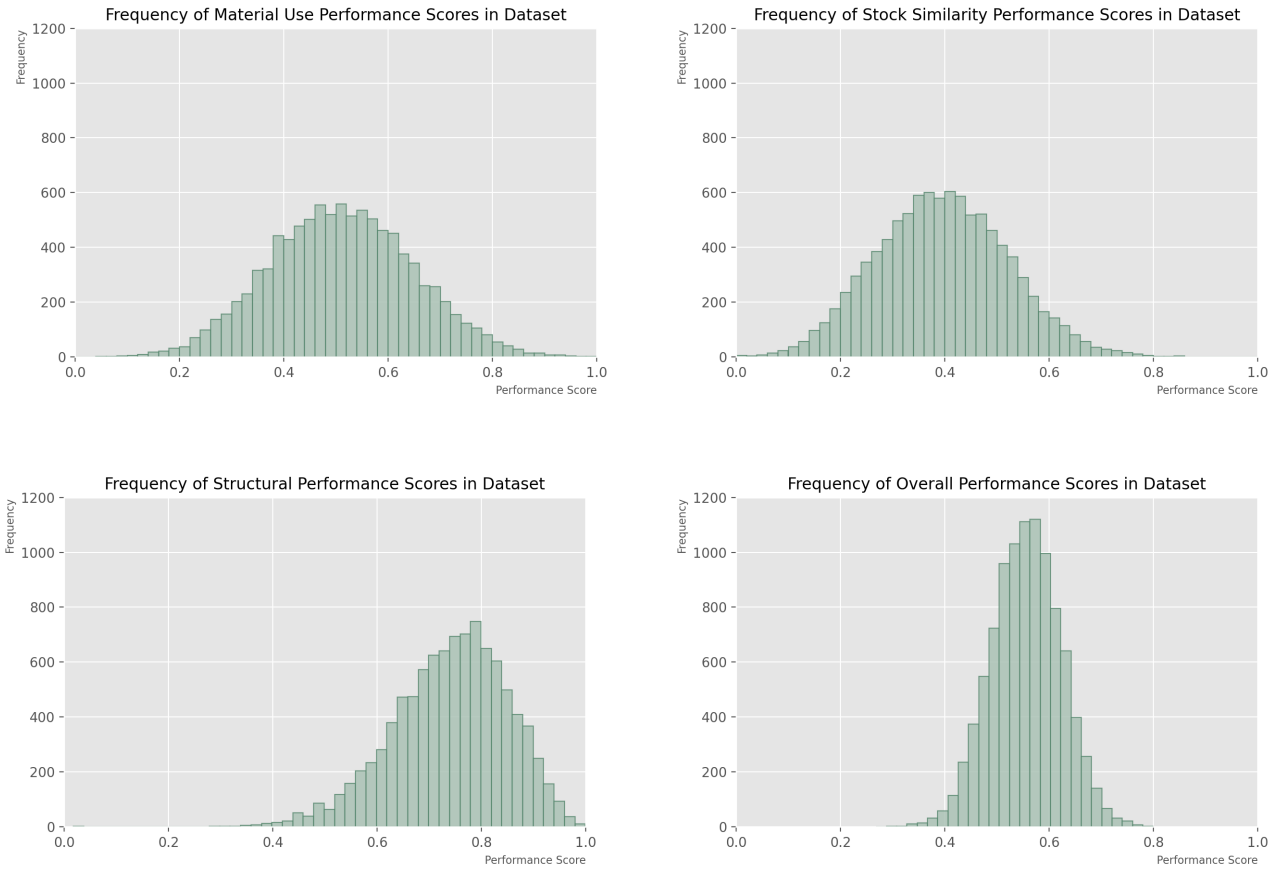


Figure 3.1.11. Adjacency Matrix of the generated geometries.
Own work.

Index	Seed	Performance Indicators					
		Material Use	Stock Similarity	Displacement	Utilization	Structural	**Overall**
0	**1**	0.55225	0.34438	0.82532	0.71787	0.78234	**0.56114**
1	**2**	0.53871	0.33137	0.71235	0.42146	0.596	**0.47869**
2	**3**	0.46073	0.31822	0.73757	0.60154	0.68316	**0.4927**
3	**4**	0.64719	0.44241	0.49164	0.35649	0.43758	**0.48143**
4	**5**	0.62315	0.43479	0.75207	0.45264	0.63229	**0.55146**
5	**6**	0.41215	0.27309	0.89061	0.67776	0.80547	**0.51385**
6	**7**	0.2711	0.11481	0.84546	0.74049	0.80347	**0.42153**
7	**8**	0.57979	0.52641	0.67451	0.3507	0.54498	**0.54451**
8	**9**	0.62806	0.45844	0.72561	0.55508	0.65739	**0.57195**
9	**10**	0.54796	0.42748	0.82927	0.56116	0.72203	**0.56939**
…	**…**	…	…	…	…	…	**…**
9700	**10046**	0.5383	0.30697	0.7707	0.75741	0.76538	**0.5366**
9701	**10047**	0.49993	0.3299	0.75955	0.57361	0.68518	**0.50602**
9702	**10048**	0.76174	0.59674	0.84789	0.66311	0.77398	**0.70064**

Figure 3.3.1. Performance of a sample of meshes from the generated dataset
Own work.

Index	Seed	Performance Indicators					
		Material Use	Stock Similarity	Displacement	Utilization	Structural	**Overall**
7631	**7902**	0.95711	1	0.61855	0.37167	0.51979	**0.79934**
1665	**1711**	0.72253	0.85983	0.83881	0.65456	0.76511	**0.79449**
5814	**6012**	0.99248	0.84584	0.72478	0.44042	0.61104	**0.78125**
1692	**1738**	0.80597	0.75371	0.85309	0.64721	0.77074	**0.77098**
9044	**9372**	0.74387	0.64727	0.92753	0.86637	0.90307	**0.76891**
4621	**4779**	0.92716	0.77425	0.73511	0.60514	0.68312	**0.76838**
8710	**9028**	1	0.73798	0.74784	0.58447	0.68249	**0.76819**
6341	**6563**	0.78716	0.70017	0.89266	0.72385	0.82514	**0.76756**
2518	**2592**	0.6535	0.68567	0.93106	0.84603	0.89705	**0.76379**
6252	**6468**	0.85304	0.71428	0.85244	0.63657	0.76609	**0.76276**

Figure 3.3.2. Performance of the ten best performing meshes from the generated dataset.
Own work.

are created, as shown in figure 3.3.4 (including the source mesh). This technique can be used to augment the dataset without requiring new simulation, as the variables to test for (structural performance, material use and similarity to the stock library) remain the same as the original geometry. The dataset augmentation was performed after all performance scores were calculated. After augmentation, duplicates were removed a second time. This process resulted in a total of 77.623 valid meshes in an augmented dataset. For this thesis, however, the original (non-augmented) dataset was used, as it was found to be of sufficient size for the purposes of this study.

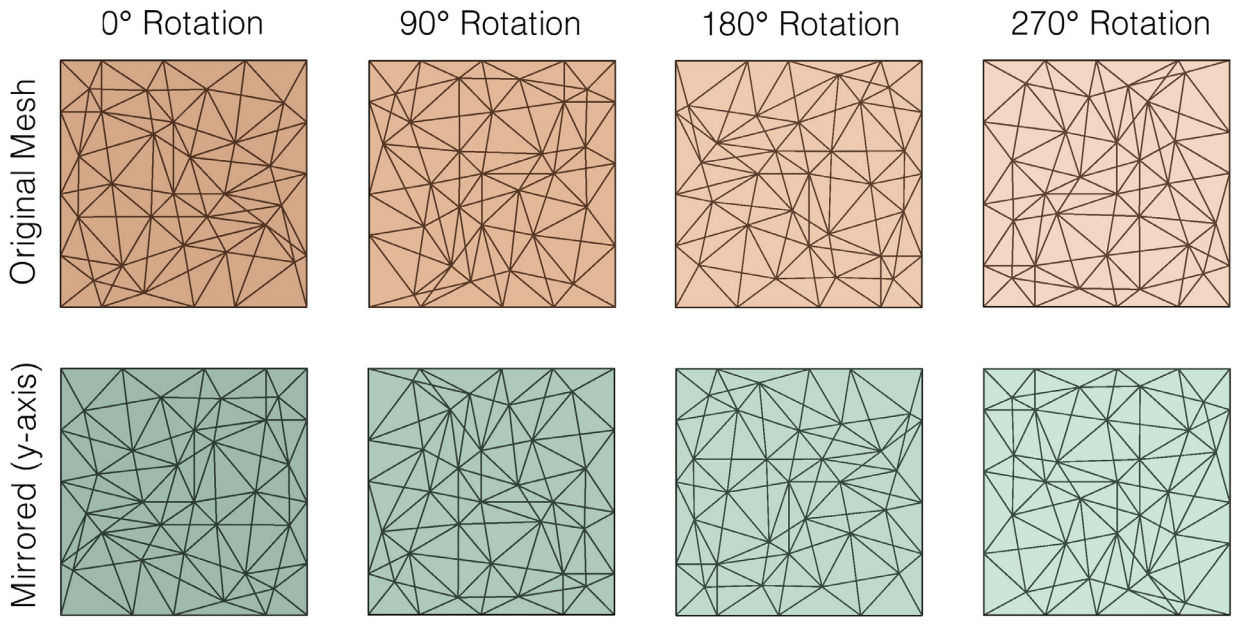


Figure 3.1.11. Adjacency Matrix of the generated geometries.
Own work.

4. Deep Learning Frameworks

4 Deep Learning Frameworks

In this chapter, the design and results of the deep learning framework are reported on and discussed. This includes the variational auto-encoder, surrogate model and gradient descent optimization. The deep learning models in this thesis were designed in Python, using the Keras API with TensorFlow. For computing, supercomputer DelftBlue of TU Delft was used.

4.1 INPUT TYPES

In this section, the various datatypes used to train the deep learning framework are listed. The following types were considered for each model:

- Coordinate input
- Adjacency matrix input
- Half adjacency matrix input
- Edge-vertex input
- Movement (vector) input
- Movement step input

4.1.1 COORDINATE INPUT

The coordinate input data consists of the 41 vertices in the mesh. This results in 41*3 = 123 inputs for each mesh. As a result, the input shape for x_train is (41, 3) for floating-point numbers. An example of this input type for mesh 80 is given in figure 4.1.1.

	x	y	z
0	0.00	0.00	0.00
1	1.67	0.00	0.00
2	4.00	0.00	0.00
...	...	...	...
38	3.00	7.57	-0.67
39	4.28	7.09	-1.33
40	7.31	6.72	-1.00

Figure 4.1.1. An example of coordinate data input.
Own work.

4.1.2 ADJACENCY MATRIX INPUTS

Instead of the 41x3 coordinate input, an adjacency matrix input based on a fine mesh can be utilised. This required recreation of each mesh based on a square 3D grid. This process is visualised in figure 4.1.2.

For this method, three fine meshes were tested. For each, the accuracy was calculated as follows:

$$accuracy = 1 - \frac{\sum edge\ length\ differences}{\sum edge\ lengths} * 100$$

This resulted in the following:

Adjacency Matrix Shape		Accuracy
4375x4375	19,140,625	>99%
1445x1445	2,088,025	~97%
243x243	59,049	~94%

The selected adjacency matrix input data is of shape (243, 243) and consists of binary input. An example of an adjacency matrix for the small example mesh used in chapter 2 is shown in figure 2.1.3.

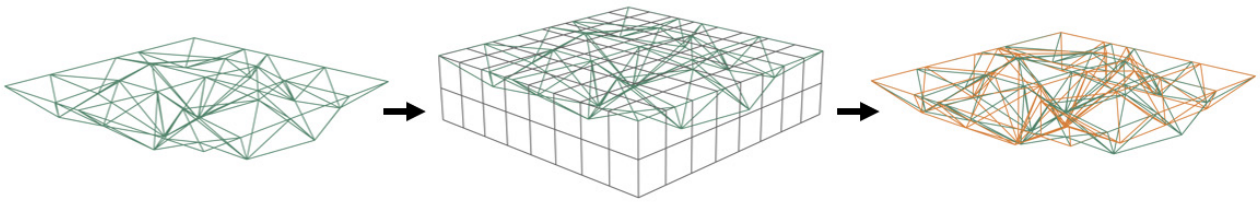


Figure 4.1.2..Application of the fine mesh for adjacency matrices. This example shows a 94% accuracy.
Own work.

	0	1	2	3	4
0	0	1	0	0	0
1	1	0	1	0	0
2	0	1	0	1	0
3	0	0	1	0	1
4	0	0	0	1	0

Figure 4.1.3. An example of adjacency matrix input.
Own work.

Half adjacency matrix input

As adjacency matrices are symmetric over the diagonal, it is possible to only use half of the data (only unique data items) as input as a list. This is visualised in figure 4.1.4. In this example, this would translate to the following list:

```
[1,0,1,0,0,1,0,0,0,1]
```

For the tested geometry, these lists have a shape of (29403, 1) and consist of binary data.

	0	1	2	3	4
0	0	1	0	0	0
1	1	0	1	0	0
2	0	1	0	1	0
3	0	0	1	0	1
4	0	0	0	1	0

Figure 4.1.4. An example of an adjacency matrix. Unique values are highlighted.
Own work.

4.1.3 EDGE-VERTEX MATRIX INPUT

A third type of tested input are edge-vertex matrices with binary data. These can be generated using the previously described adjacency matrices. For the tested geometry, the edge-vertex matrices have a shape of (243, 128). This is significantly smaller than the adjacency matrices.

Figure 4.1.5 shows the adjacency matrix for the example mesh used in chapter 2 on the left. Both the rows and columns are the vertices in this mesh. In the same figure, the matrix on the right shows the edge-vertex matrix for the same mesh. In this matrix, rows indicate edges, while vertices are shown in the columns. As each edge is connected to two vertices, each row has two values of '1'. The circled values indicate how an edge-vertex matrix can be obtained from an adjacency matrix. This example mesh has more edges than vertices. As a result, this edge-vertex matrix is larger than its adjacency matrix. However, the tested geometry has fewer edges (128) than vertices (243), so the edge-vertex matrices are smaller than the adjacency matrices.

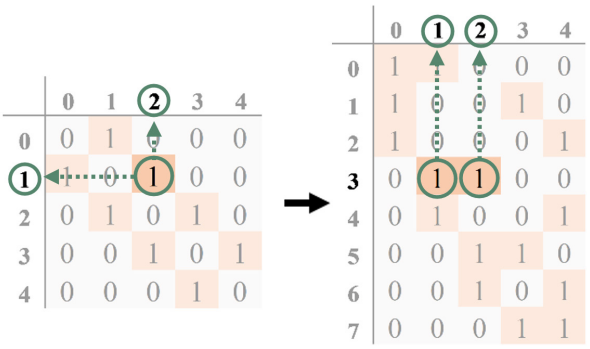


Figure 4.1.5. An example of an adjacency matrix (left) & edge-vertex matrix (right).
Own work.

4.1.4 VERTEX MOVEMENT INPUTS

As all geometries in the dataset are created from a base mesh and contain the same overall structure and number of edges, they can be accurately represented by the vector of movement per vertex in the x, y and z direction. This results in data of type 'float' with shape (41, 3), as shown in figure 4.1.6. This input is defined by calculating the difference between the vertex position as defined in the coordinate matrices and the base mesh coordinates.

	x	y	z
0	0.00	0.00	0.00
1	-0.33	0.00	0.00
2	-0.67	0.00	0.00
...	...	...	...
38	0.39	-0.42	0.67
39	-0.13	-0.06	0.67
40	-0.63	0.57	0.33

Figure 4.1.6. An example of movement data input.
Own work.

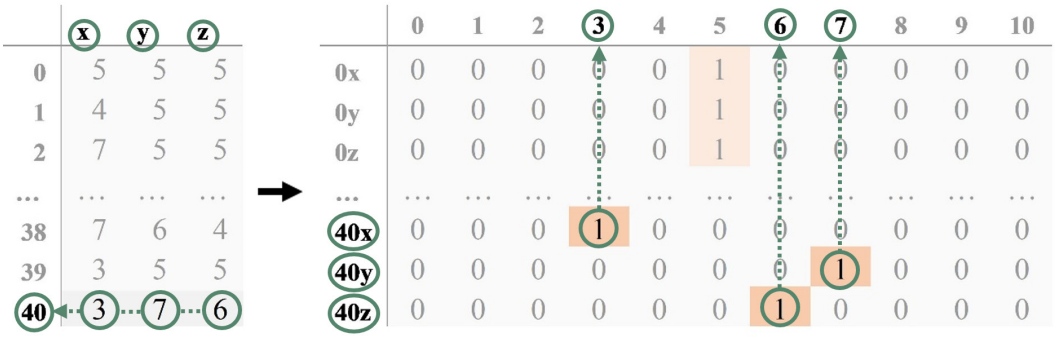


Figure 4.1.7. Creation of 'vertex stepped movement input'. The movement data (as shown in figure 4.1.6) is first converted to a number of steps (table, left). Then, it is converted to binary data (table, right).
Own work.

Vertex Step Movement Input

The same data can be defined through binary values. It is defined by rewriting the movement as a number of steps, and entering a 1 value in the column of the correct number of steps for each vertex in the x, y and z directions. The steps are first simplified as:

```
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4 ,5]
```

Then, this is converted to indices of:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

So, for the meshes in the dataset, the input data shape is (41 * 3, 11) = (123, 11).

4.2 SURROGATE MODEL

The surrogate model is a simple neural network. Its purpose is to map design geometries to the performance indicator found. In this section, all researched options for the surrogate model are described. In each case, the prepared dataset is split into a training and testing dataset. 80% of samples are randomly selected and added to the training set. The other 20% is used as test data. The generated meshes are represented by coordinate data or adjacency matrices in `x_train`. The performance score data serves as labelling, represented as floats

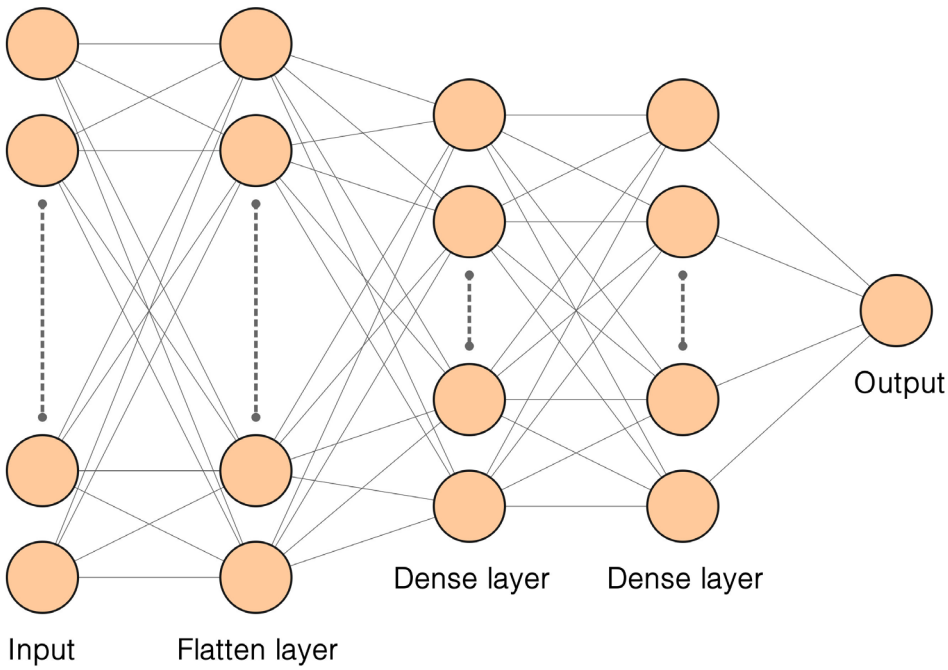


Figure 4.2.1. A schematic representation of the architecture of the surrogate model.
Own work.

in `y_train`. The figure below (4.2.1) shows a schematic representation of the surrogate model's architecture. The full code for the surrogate model can be found in Appendix III.

4.2.1 SURROGATE MODEL DESIGN OPTIONS

As stated previously, this performance indicator is calculated using separate, normalised performance indicators for structural performance, in turn split into 'displacement' and 'utilization', material use (mass) and similarity to the stock library. The desired output is a single value; the overall performance indicator. There are several ways in which the surrogate model can be designed, as the performance scores that need to be predicted can be separated. Below, all options are explained.

Option 1: Overall Performance Score Model

Firstly, the model can be designed to directly calculate the overall performance score for each mesh. This method only requires one NN and is therefore simple. It is visualised schematically in figure 4.2.2.

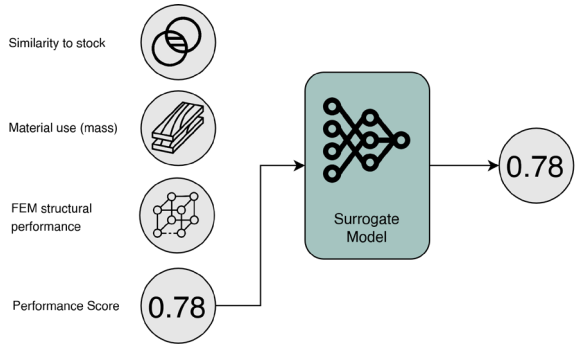


Figure 4.2.2. Schematics of surrogate model option 1. Own work*.

Option 2: Structural + Material Use & Similarity Models

As shown in the work of Pavildou (2022), a surrogate model can be trained to successfully predict structural performance and material use for 2D meshes based on adjacency matrices. This architecture can be used here as well (figure 4.2.3). As stock similarity performance has no direct relation to the structural performance of a mesh, using this data in a separate NN could result in a lower loss. Within this option it is also possible for the structural and material use scores to be calculated by a surrogate model, but to use the existing code for similarity score calculation, instead of making an NN to assess this.

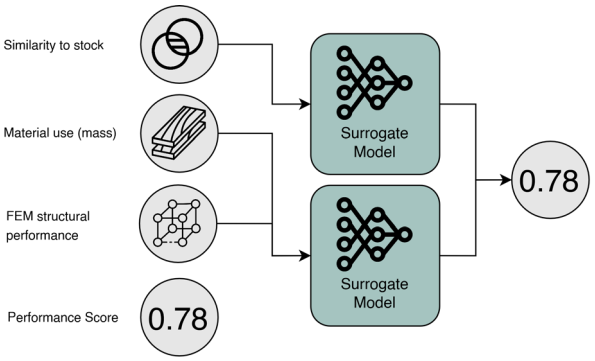


Figure 4.2.3. Schematics of surrogate model option 2. Own work*.

Option 3: Fully Separate Models

Further separating the aspects that make up the overall performance score could result in up to three separate NNs in the surrogate model (figure 4.2.3). This can give more insight into which aspects the NNs are able to predict. It also allows for varying NN designs to fit each data type. Finally, it is possible to use NNs for some predictions and combine this with calculations from the existing performance simulation models.

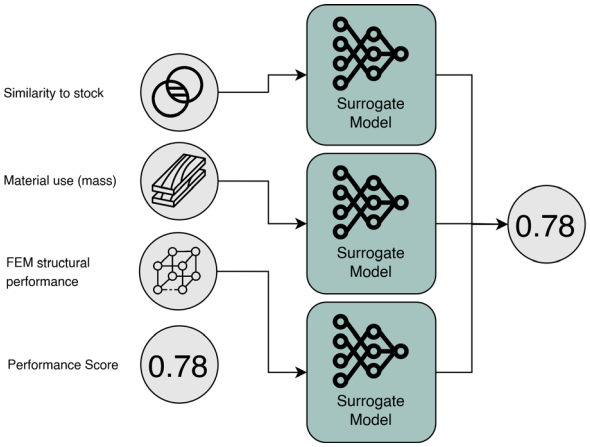


Figure 4.2.4. Schematics of surrogate model option 3. Own work*.

*In figure 4.2.2-4.2.4, icons are retrieved from:
Wood icons created by winniwvinzence - Flaticon, from https://www.flaticon.com/free-icons/wood.
Transparency icons created by Freepik - Flaticon, from https://www.flaticon.com/free-icons/ transparency.
Matrix icons created by Freepik - Flaticon, from https://www.flaticon.com/free-icons/matrix.
Neural network icons created by Freepik – Flaticon, from https://www.flaticon.com/free-icons/neural-network.

4.2.2 TRAINING DATASET

The total dataset size is 9.703. For the training data; a total of 60 items are removed, resulting in a dataset size of 9.643. 80% of the overall dataset (7.714 samples) is used as training data, 20% (1.929 samples) is used for validation/test data. The removed meshes include the ten best performing meshes as previously mentioned and 50 randomly chosen samples. These samples are used to demonstrate the accuracy of the trained models later in this chapter. For this set, the following samples were selected:

```
[9012, 2337, 838, 3730, 6116, 8979, 3517,
348, 7093, 9699, 107, 1690, 1071, 4188,
2538, 7919, 1926, 5450, 1701, 5615, 1605,
8055, 6704, 256, 3059, 1875, 9392, 3108,
1652, 7019, 4519, 3234, 1042, 2072, 709,
7338, 8721, 4013, 5177, 2868, 3995, 822,
4180, 8437, 5005, 2057, 5199, 1397, 2543,
3530].
```

The frequency of these performance scores are shown in the graphs in figure 4.2.5. The numerical values of the scores of these samples can be found in Appendix II.

4.2.3 COORDINATE INPUT

The first input tested first for the surrogate model consist of a series of coordinates for the 41 vertices in the mesh. This results in 41*3 = 123 inputs for each mesh. As a result, the input shape for `x_train` is (41, 3). `y_train` is (1) and the output is a single numerical value; the performance score. The tested architecture included the input layer, and finally the output layer. The layer type used is 'Dense'. This layer expects a row vector as input, where each column is a feature of the input. As a result, the input layer is flattened, so the 2D shape is converted to 1D.

Exploration

The accuracy of the surrogate model depends on various aspects in the model architecture. Therefore, several versions of the model were

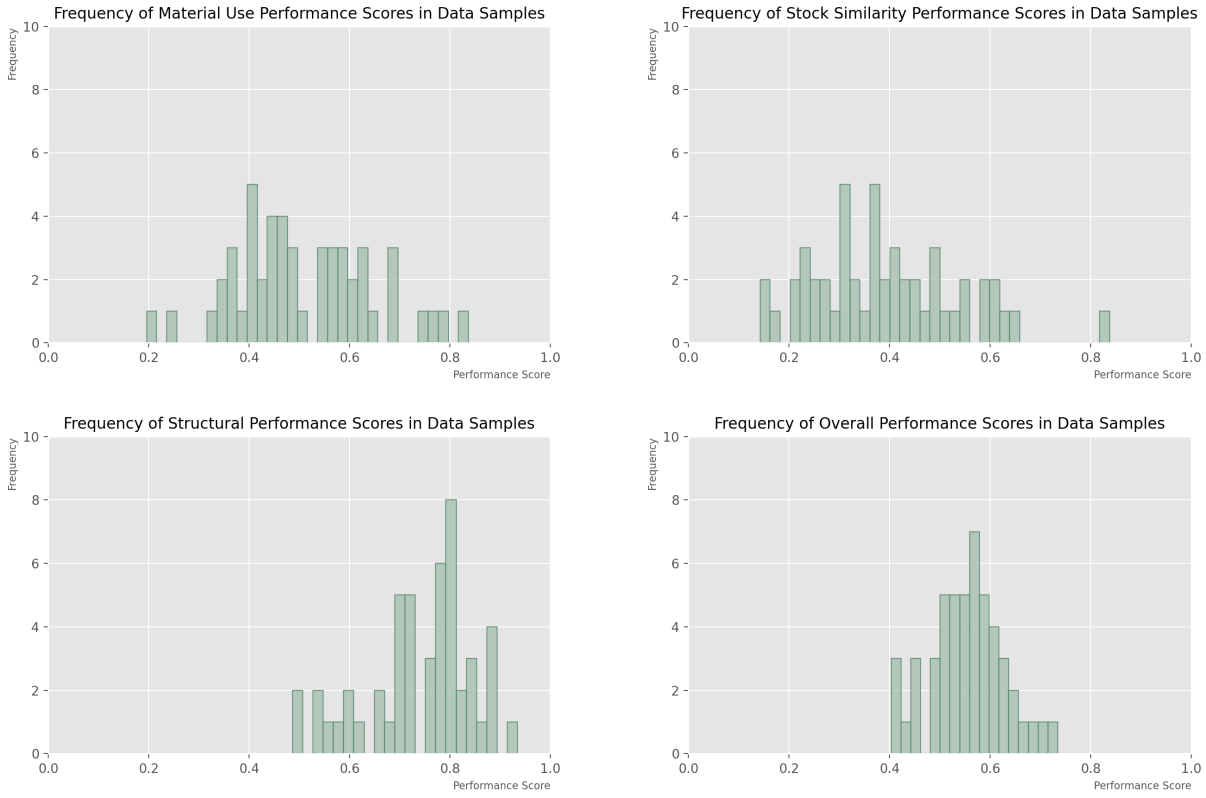


Figure 4.2.5. Frequency histograms of the performance scores. Bar width = 0.02.
Own work.

run to find the best performing model. The following aspects were explored:

Batch size:
- 16
- 32
- 64
- 128

Epoch at which training is stopped:
- Before validation loss stops decreasing
- At the point where validation loss stops decreasing
- Some epochs after validation loss stops decreasing

Learning rates:
- 0.001
- 0.0001
- 0.00001
- 0.000001

Loss types:
- Mean absolute error
- Mean squared error

Layer activation types:
- Relu
- Sigmoid
- Softmax
- Softsign

Metrics:
- 'accuracy'
- 'top_k_categorical_accuracy'

Optimizer types:
- SGD
- Adam

Clipnorm: (to avoid exploding gradients)
- 1.0
- No clipnorm

Addition of normalization layers:
`(tf.keras.layers.experimental.preprocessing.Normalization(axis=1))`
- No additional normalization layers
- One normalization layer
- Multiple normalization layers

Addition of dropout layers
`(model.add(Dropout(0.25)))`
- No dropout layers
- One dropout layer

Architectures
- '128-42-1'
- '64-64-1'
- '64-32-1'
- '32-32-1'
- '32-16-1'
- '16-16-1'

The types of architecture are compared are visualised in figure 4.2.6.

The models in this section are labelled as 'Exploration 0-26'. Each model is trained to predict the overall performance score, as described under 'option 1' in section 4.2.1. A control model, labelled with ID 0, was created to compare variations of the model to. The attributes of this model are shown in the table in figure 4.2.7.

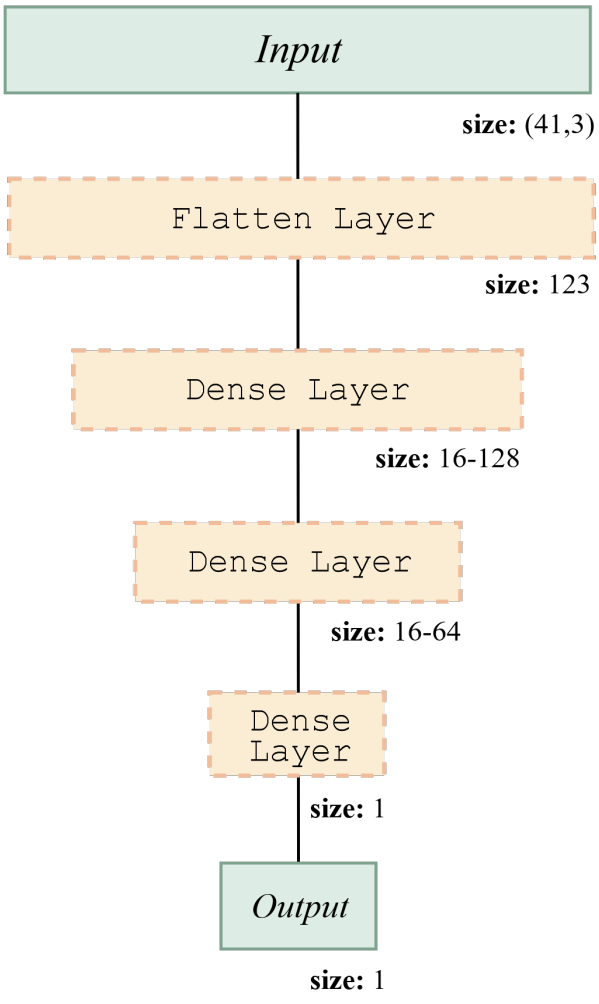


Figure 4.2.6. Schematics of 'surrogate model with coordinate input' architectures.
Own work.

Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

Figure 4.2.7. The attributes of the control model for surrogate models with coordinate input.
Own work.

Results

To analyse the data, several graphs were plotted. In the main text, only a selection of these graphs is shown. However, all graphs can be found in Appendix IV. Here, the graphs titles 'Model Loss' show the training (train) loss and validation (test) loss over the number of epochs that the model was run for, as discussed in Chapter 2. The first graph shows the model run for a standardized number of epochs. In the second graph, the model is rerun for a model-specific number of epochs. The graphs titled 'Distribution of Performance Scores' and 'Distribution of Prediction Error' give information on the distribution of the input data and model predictions. The first of these graphs shows a histogram in which the frequencies of the predicted and calculated scores are plotted. In the second graph, the prediction error, defined as the absolute or percentual difference between calculated and predicted scores, is plotted. The meshes in this graph are sorted on error in ascending order. The averages of each error type are also plotted. This graph provides information on the number of predictions that is within a certain error margin. Finally, in the graphs labelled 'Calculated and Predicted Performance Scores of Samples' and 'Calculated and Predicted Performance Scores of All Meshes', predicted and calculated performance scores for the 50 selected meshes (which were excluded from the training dataset) and the full dataset are plotted. In both graphs, the data is sorted by the predicted scores. Unless explicitly specified, the training of each model is stopped when validation loss stops decreasing. The lower 'Model Loss' graph shows the epoch that was selected for each model. For example, the base model produces an average loss of around 0.005 (figure 4.2.8, left). As the validation loss (test loss) stops decreasing after 80 epochs, this is where training is stopped (figure 4.2.8, right).

As both the test and train loss decrease until a set value is reached, it can be concluded that the model trains. However, the distribution of the predicted performance score differs significantly from the calculated scores; the prediction range was more limited than the calculated score range (figure 4.2.9). This

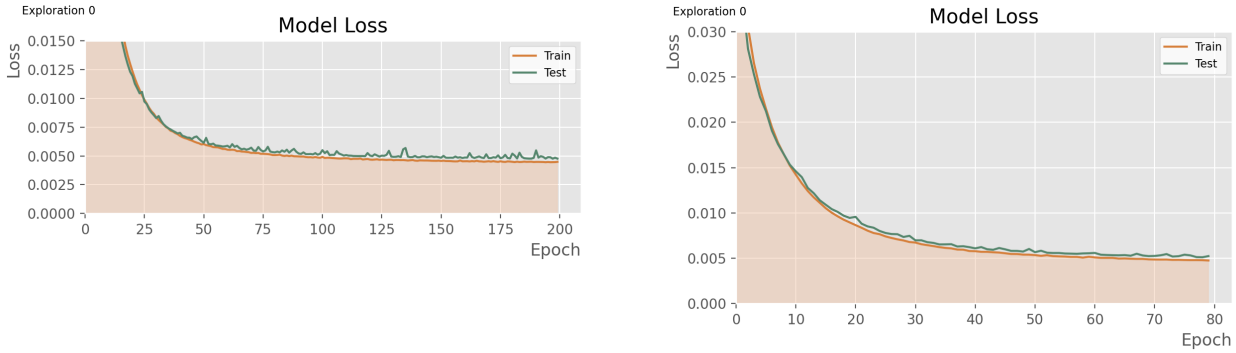


Figure 4.2.8. Model loss of surrogate model exploration 0 (control model, coordinate input) for 200 (left) and 80 (right).
Own work.

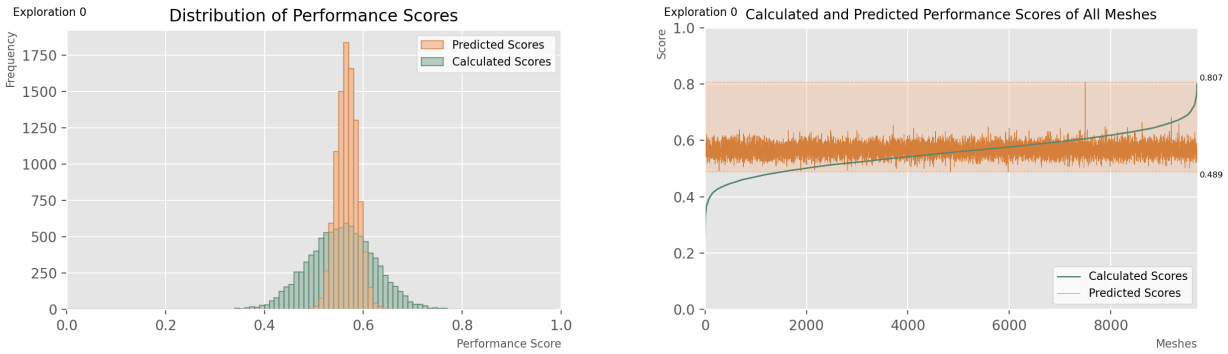


Figure 4.2.9. 'Distribution of performance scores' (left) and 'Calculated and predicted performance scores of all meshes' (right) of surrogate model exploration 0 (control model, coordinate input).
Own work.

indicates that the model is unable to accurately predict the performance scores outside this range. It can also be concluded that predictions are mostly within a range of 0.5-0.6, regardless of the calculated (train) value. Running the same model for varying periods of time showed that overtraining the model led to narrowing of the predicted score range (figure 4.2.10, top right). In some explorations, however, the distribution of predicted scores matched that of the calculated scores much more closely (figure 4.2.10, top left). Note that the scales on the 'Frequency' axis are different.

However, there is still no correlation between calculated and predicted performance scores; a wider range of predicted scores was found regardless of corresponding calculated score, as shown in figure 4.2.10 (bottom left). In the case of overtraining, each prediction was made in an extremely narrow range, as shown in figure 4.2.10 (bottom right). In this example, all values were the same; 0.556.

Figure 4.2.10. Distributions of performance scores & performance score plots for exploration 4 and 12. Note the scales on the 'Frequency' axis.
Own work.

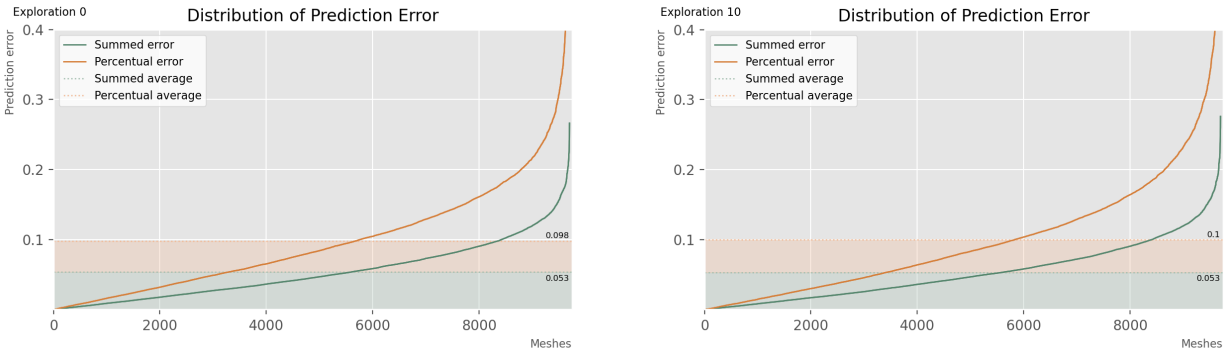


Figure 4.2.11. Distribution of prediction error for exploration 0 (left) and 10 (right).
Own work.

The control model (0) produced an average summed error of 0.099 and an average percentual error of 0.056 (figure 4.2.11, left). The minimum summed and average error found in exploration 0-24 are 0.094 and 0.053 respectively. These results were found in exploration 10. The difference in error between most explorations, however, was limited: 0.094 to 0.0113 and 0.053 to 0.062. This indicates that many of the tested variables do not impact model performance significantly. However, models four, fifteen and twenty had significantly higher errors compared to the control model. As a result, the following was observed. First, stopping training before validation loss stops decreasing results in higher errors. This is found in the results of exploration 4. Secondly, using the SGD optimizer results in higher errors (exploration 15). Finally, adding one dropout layer (after the first layer) results in higher errors as well. This is found in the results of exploration 20.

As shown in grey in figure 4.2.12, the distribution of the performance data is mainly centred around 0.45-0.65. In previous explorations 0-24, the predicted scores tended to fall within this range also. Therefore, the effects of alteration of the data distribution were examined. Firstly, the overall performance data

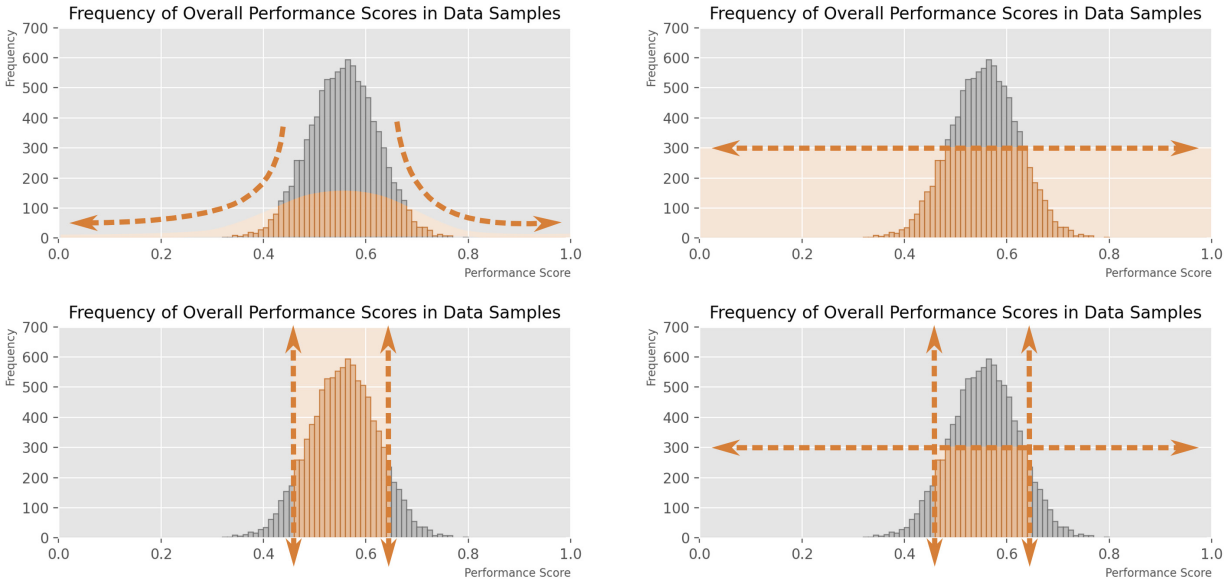


Figure 4.2.12. Schemes indicating how data is selected for the exploration on alteration of the data distribution.
Own work.

was re-normalised, so that it ranged from zero to one. This is visualised in the top left image in figure 4.2.12. The model trained on this data is exploration 27. In this case, the dataset size remained 9.703. In the other three attempts parts of the data were removed to alter the distribution. In exploration 28, the data was limited vertically as shown in the bottom left in figure 4.2.12. Here, all meshes that scored outside of the 0.45-0.65 range were removed. This resulted in a dataset size of 8.397. Next, the original dataset was altered so that each set of 0.01 performance score range only include no more than 300 samples (top right of figure 4.2.12). This dataset contains 7.029 items and is tested in exploration 29. Finally, in exploration 30, the dataset was limited both vertically and horizontally (bottom right of figure 4.2.12). The dataset size in this case was 5.723.

Results

These four models (exploration 27-30) trained differently. For this reason, they were stopped at different epochs. Results are shown in figure 4.2.13. The models of exploration 28 and 30, where the performance score range was limited, produced the smallest loss. This loss is smaller than control model 0. It should be noted, however, that is it improbable that these model are able to make accurate predictions outside of a 0.45-0.65 range, since those scores are not represented in the training data. This can also be seen in the full reports on these explorations, which can be found in Appendix IV. In exploration 29, the training loss reaches similar levels to the control. However, the validation loss stays significantly higher. These attempts were therefore not explored further.

Finally, the control model is compared to the model with re-normalized data (27) in the top images in figure 4.2.13. However, as the range of the data is changed by the re-normalization, the quality of these models is best represented by comparison of calculated and predicted data (figure 4.2.14), rather than loss function. These

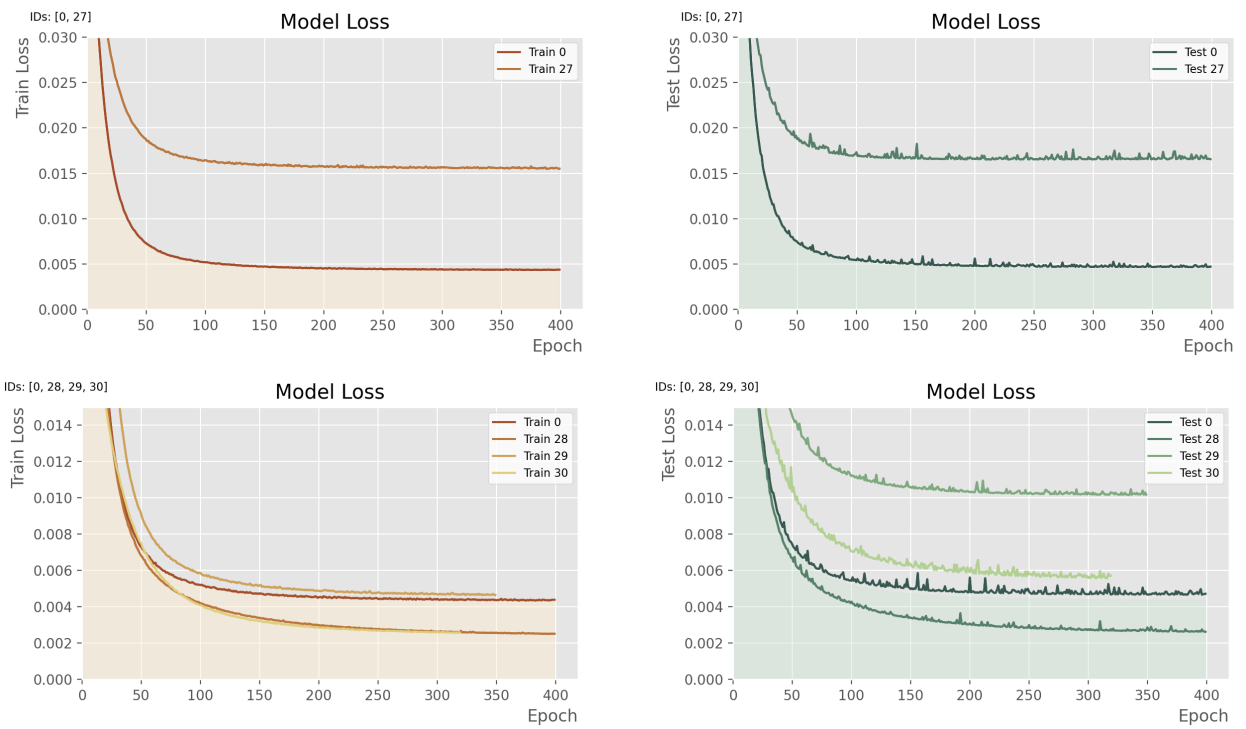


Figure 4.2.13. Model loss comparison for exploration 27-30 and control group 0. The figures on the left show train loss and those on the right show test loss. The area under curves for exploration 0 are shaded.
Own work.

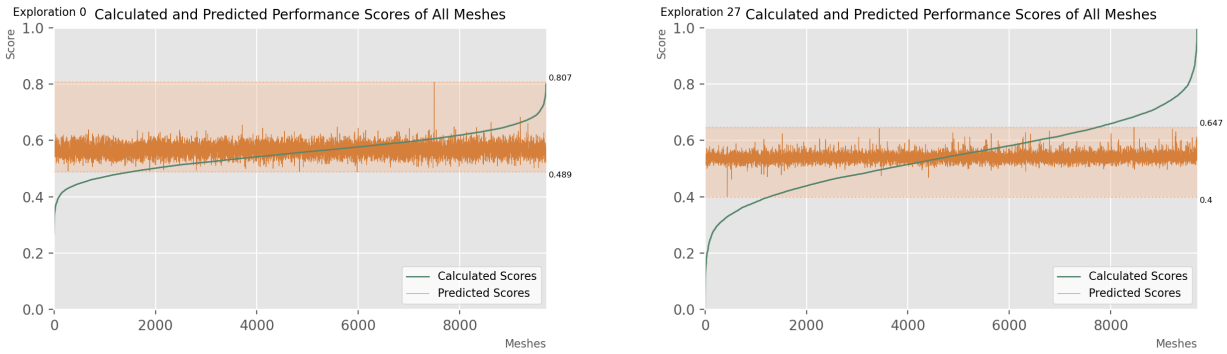


Figure 4.2.14. Calculated and predicted performance scores of all meshes for exploration 0 (left) and 27 (right).
Own work.

graphs show that the model for exploration 27 does not produce more accurate predictions compared to the control model. For this reason, this method is abandoned also.

4.2.4 ADJACENCY MATRIX INPUT

A second set of models is run with adjacency matrix input. With this input, similar exploration attributes were explored. However, different architectures were tested. This is due to the input data shape being larger. The architectures that were tested are:

Architectures

- '256-128-64-1'
- '128-64-32-1'
- '128-64-1'
- '64-64-1'
- '64-32-1'
- '32-16-1'

In the graphs in figure 4.2.15, the model loss of the control models for coordinate input (left) and adjacency matrix input (right) are shown. With the adjacency matrix input, the training curves reach much values. There is a larger difference between the train and validation curves. Still, the test loss reaches a lower level with adjacency matrix input compared to coordinate input. The next set of plots (figure 4.2.16) show the calculated and predicted performance scored of the meshes. This includes all training data and the fifty meshes that were excluded from training. The predictions, shown in orange, on the exploration 31 graph have a range that closely matches the calculated range. Additionally, the trend of the predicted score plot follows the calculated curve here, whereas in the coordinate input plot of exploration 0, the predictions are close to the same value regardless of the calculated scores. This indicates better performance of the model with adjacency matrix input.
This is also confirmed in the graphs in

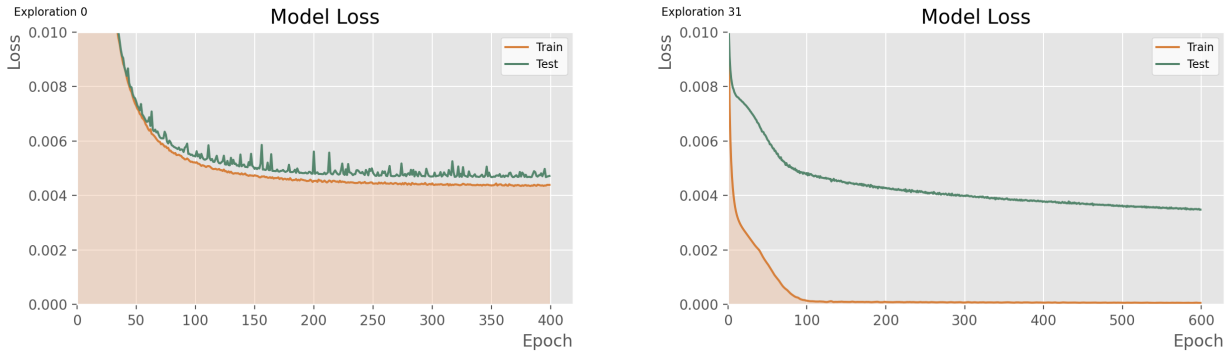


Figure 4.2.15. The model loss (test and train) for exploration 0 with coordinate input and exploration 31 with adjacency matrix input.
Own work.

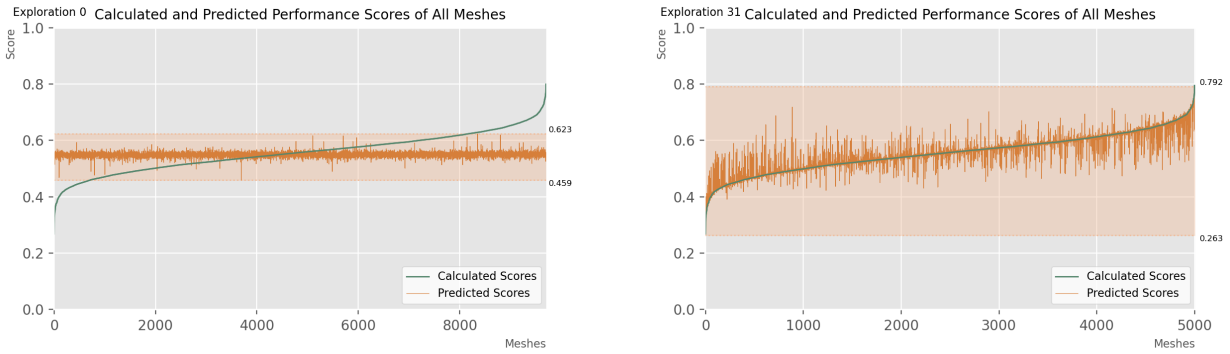


Figure 4.2.16. Calculated and predicted performance scores of all meshes for exploration 0 with coordinate input (left) and 31 with adjacency matrix input (right).
Own work.

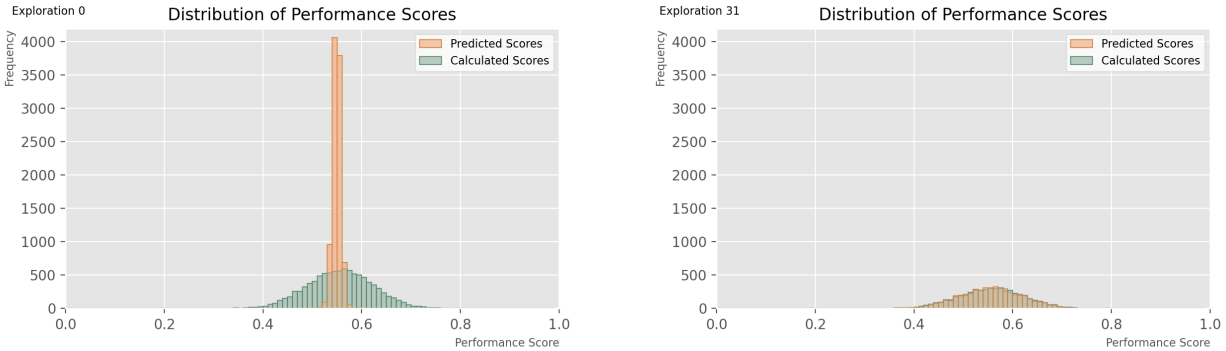


Figure 4.2.17. Distribution of performance scores for exploration 0 with coordinate input (left) and 31 with adjacency matrix input (right).
Own work.

figure 4.2.17, showing the distribution of the calculated and predicted performance scores. The distribution of predicted scores in the adjacency matrix surrogate model resembles the distribution of the calculated scores closely, indicating better predictions.

In model 31-57, the impact of batch size, learning rate, activations, metrics, optimizers, clipnorms, normalization layers, dropout layers and architectures. The full report on all models can be found in Appendix IV. The main graphs for the control model are shown in figure 4.2.15 and 4.2.16. Firstly, batch size changes the time in which the model is trained. Models with a smaller batch size learn the data in less epochs. With a batch size of 16, the test loss of the model nears zero at around 60 epochs. With a batch size of 128, this happens after 350-400 epochs. Smaller batch sizes also resulted in lower validation loss. The models with batch size of 32 and 16 reached lowest train and validation losses.

The validation loss of most models with adjacency matrix input decreases for a long time, as can be seen in figure 4.2.18 (exploration 36), where the model was run for 2000 epochs. Stopping the model earlier results in higher train and validation losses and less accurate predictions.

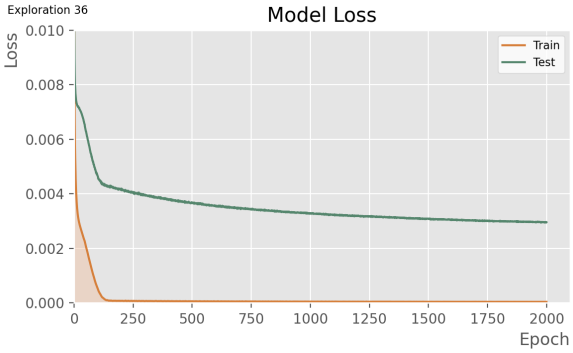


Figure 4.2.18. Model loss of exploration 36 (2000 epochs).
Own work.

Secondly, a larger learning rate results in faster

decrease of the train and test loss. The lowest validation loss was achieved with a learning rate of 0.00001 (exploration 40). However, models with higher learning rates (0.001 and 0.0001) train significantly faster and still showed good performance. In the graphs in figure 4.2.19 show the predictions for model 38 (lr=0.0001) and 40 (lr=0.000001). Out of the models used to compare the impact of learning rate, the predictions of model 38 are consistently the closest to the respective calculated values. Additionally, some predictions in model 40 are significantly too low, reaching the lower range of all predictions.

Test loss of the model run for mean absolute error (exploration 41) remained slightly higher than train and test loss for the control model trained for mean squared error (figure 4.2.20, left). As predictions made by the model using 'mean squared error' were overall closer to the calculated values, it was used in the final models (figure 4.2.20, right).

Layer activations, however, did have a significant impact on the loss curves. These were tested in exploration 41-44 and the control model. A summary of the results is shown in figure 4.2.21. A 'relu' activation on all layers, as used in the control mode, was found to perform best (figure 4.2.21, top left). 'sigmoid' activation resulted in similar train and test loss values after 75-100 epochs However, after 100 epochs this model showed an increasing gap between train and test loss, indicating poor performance (figure 4.2.21, top right). The 'softmax' layer activation proved to be unsuitable for this model. Results showed much slower decreases of losses (figure 4.2.21, bottom left). 'softsign' activation performed similarity to the control model. Train and test loss decreased over epochs and reached similar

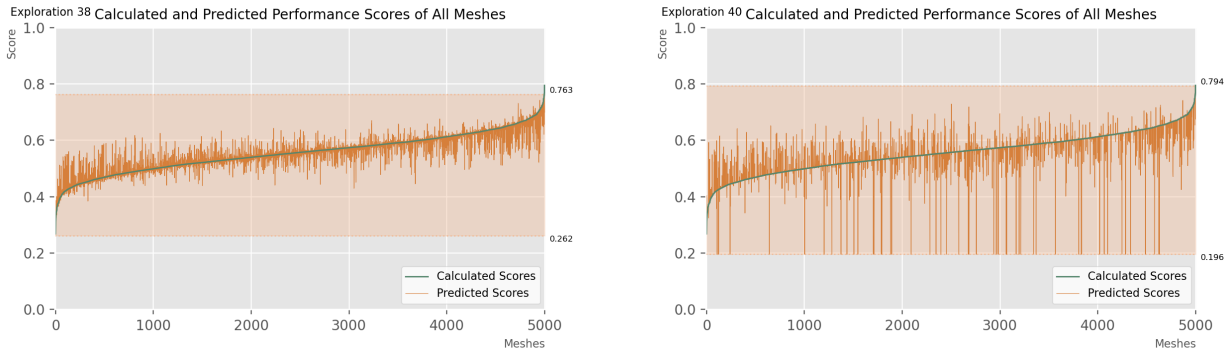


Figure 4.2.19. The calculated and predicted performance scores of all meshes for exploration 38 (lr=0.0001) and 40 (lr=0.000001) with adjacency matrix input.
Own work.

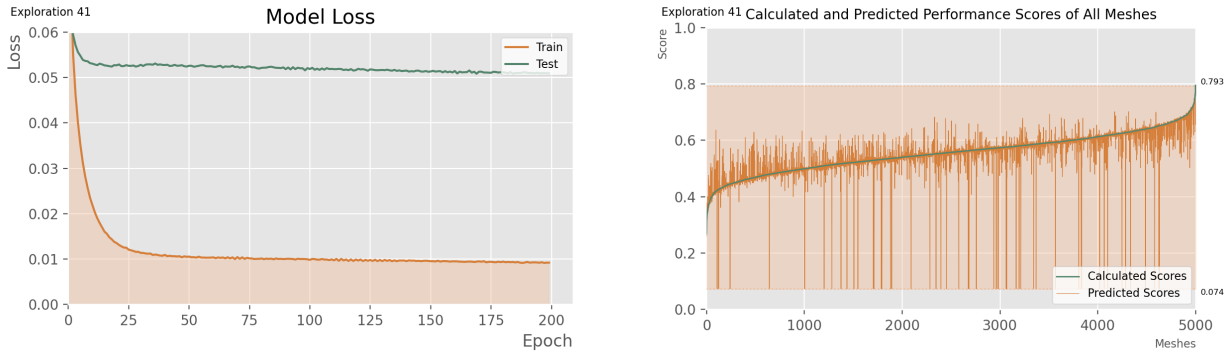


Figure 4.2.20. The model losses and calculated and predicted performance scores of all meshes for exploration 41.
Own work.

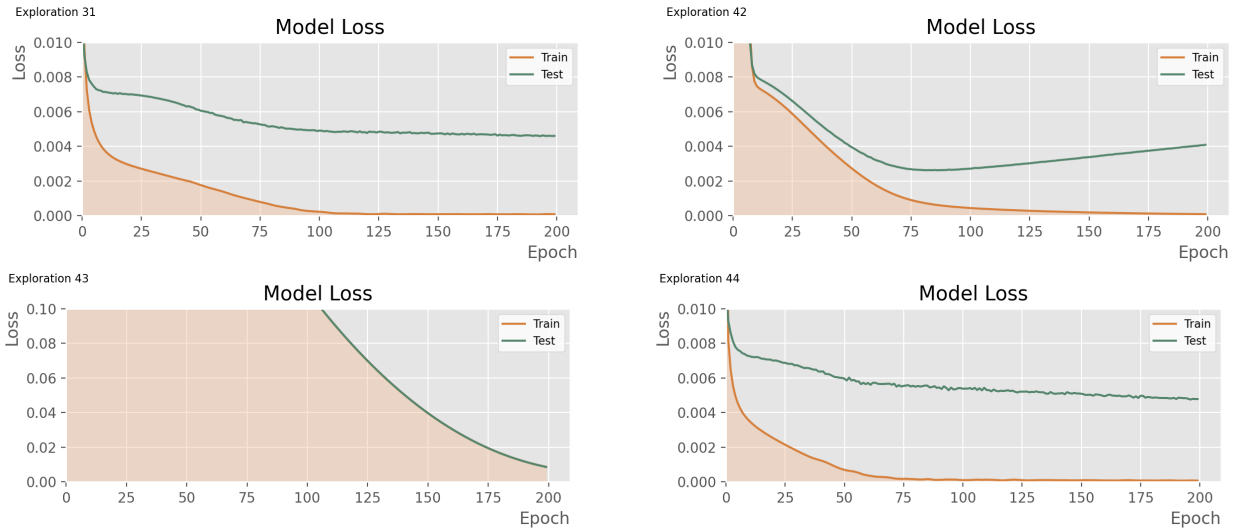


Figure 4.2.21. Model loss for exploration 31 (top left, 'relu'), 42 (top right, 'sigmoid'), 43 (bottom left, 'softmax') and 44 (bottom right, 'softsign').
Own work.

levels over 200 epochs. However, the test loss curve showed slightly more noise (figure 4.2.21, top left). The graphs showing calculated and predicted performance scores of all meshes, which can be found in the Appendix, confirm these results.

The control model was trained on metric 'accuracy'. Training on metric 'top_k_categorical_accuray', as done in exploration 45, sped up the training process slightly. In 200 epochs, it reached a lower test loss. Prediction accuracy was similar to that of the control model.

Next, the use of SGD as the optimizer was tested in exploration 46. This resulted in much slower training and a significantly larger final test and train loss (figure 4.2.22, left). The model using the RMSprop optimizer (exploration 47, figure 4.2.22, right) showed very similar training curved to the control model. This model,

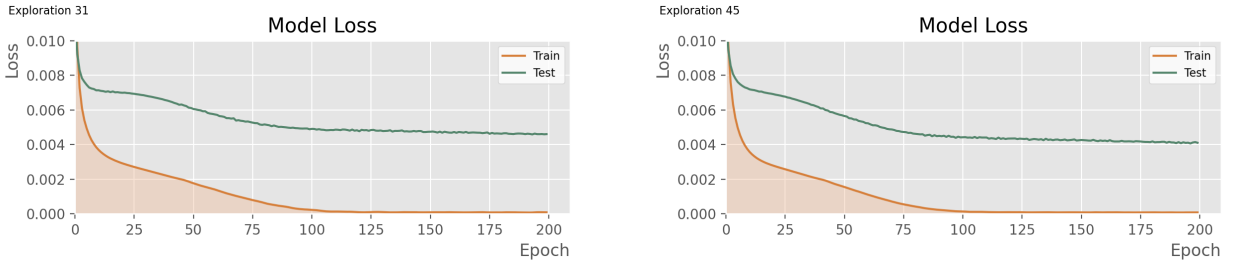


Figure 4.2.22. Model loss for exploration 31 (left, 'accuracy') and 45 (right, 'top_k_categorical_accuracy)
Own work.

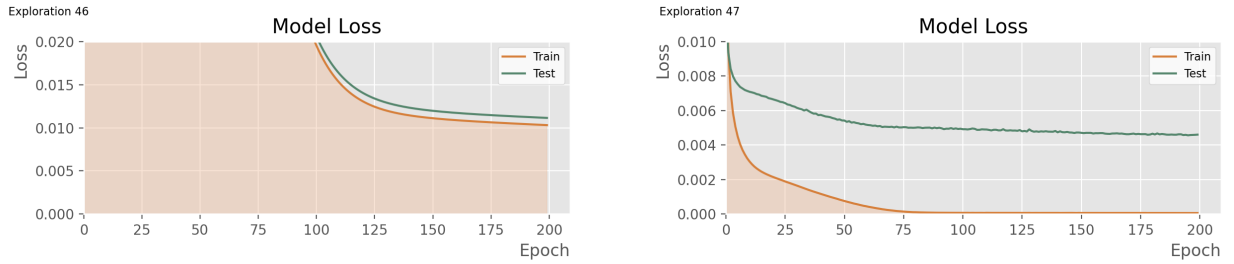


Figure 4.2.22. Model loss for exploration 46 (left, 'SGD') and 47 (right, 'RMSprop)
Own work.

however, trained slightly faster.

In exploration 48 the model was run without a clipnorm. Removing the clipnorm of 1 from the model slightly slowed down training. However, it resulted in a smaller test loss after 200 epochs. Prediction results were similar as well. The addition of normalization layers before and in-between the dense layers also had insignificant effects on training (explorations 49 and 50). For this reason, these graphs are not shown in the main text. They can, however, be found in Appendix IV.

Adding one dropout layer, as done in exploration 51, resulted in a significantly different learning curve (figure 4.2.23). After 200 epochs, the loss is significantly lower compared to the control model. However, this model tended to make predictions that were consistently too low, as can be seen in the graph showing the calculated and predicted performance scores. When multiple dropout layers were added, the predictions were generally too high for low calculated values, and too low for higher calculated values. In conclusion, the control model performed better.

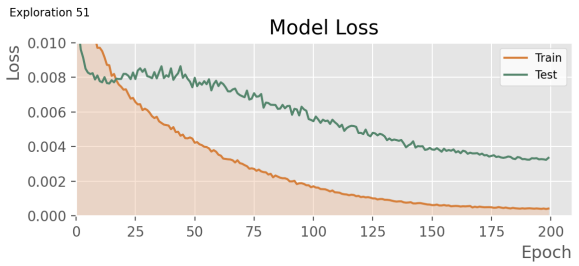


Figure 4.2.23. Model loss of exploration 51 (dropout layer).
Own work.

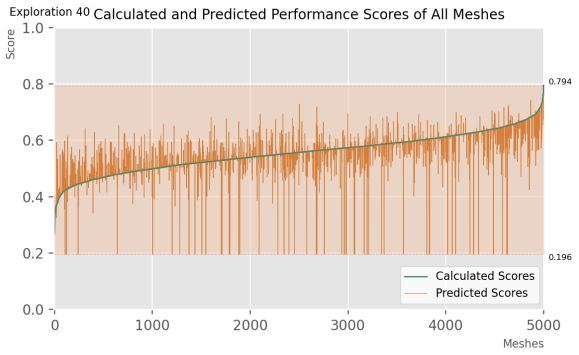


Figure 4.2.24. calculated and predicted performance scores of all meshes for exploration 41.
Own work.

Finally, of the six architectures tested in exploration 52-27 and control model 31, those with larger layer dimensions performed better than limited architectures. The learning curved reached lower test loss values and the predictions were closer to the calculated scores overall. Predictions were also more accurate for models with larger architectures, as shown in figure 4.2.25 and 4.2.26. The model with architecture '256-128-64-1' (exploration 53) had an average percentual prediction error of 2.00%, whereas the model with architecture '32-16-1' (exploration 57) had an average percentual prediction error of 2.8% . However, it should be noted that the larger models took significantly longer to compute.

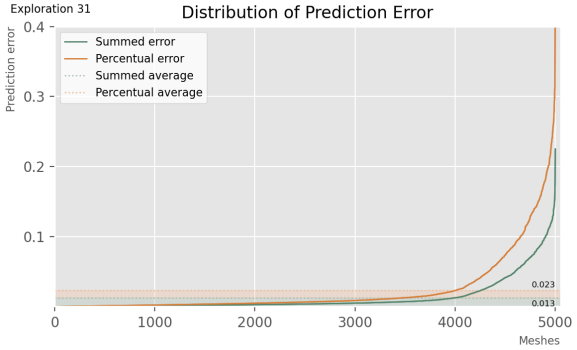


Figure 4.2.25. Distribution of prediction error for model 53 with architecture '256-128-1'.
Own work.

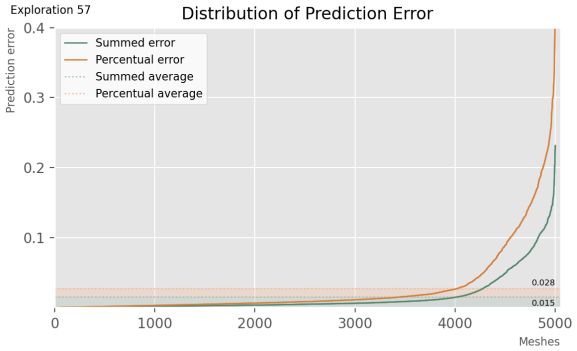
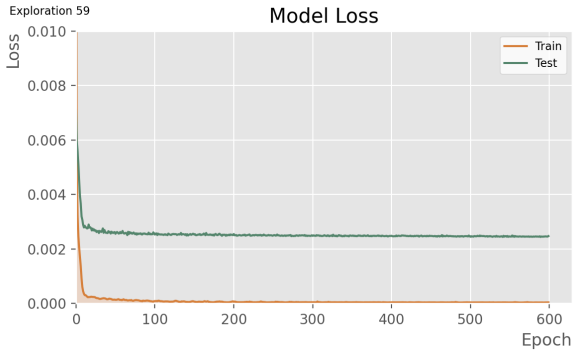


Figure 4.2.26. Distribution of prediction error for model 57 with architecture '32-16-1'.
Own work.

Option Comparison

Final model

Using the results from the previously discussed explorations, two more models were run (58 and 59). As model 59 produced the most accurate prediction, this was selected as a final model for adjacency matrix input. Details on model 58 can be found in Appendix IV.

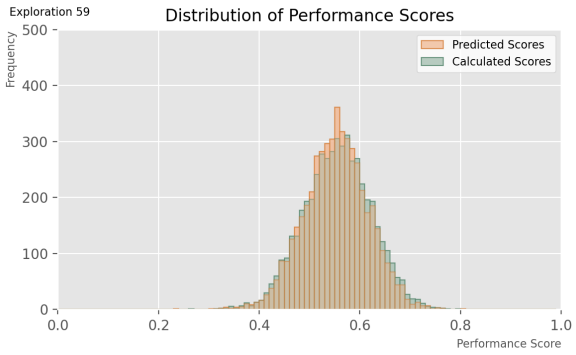


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.

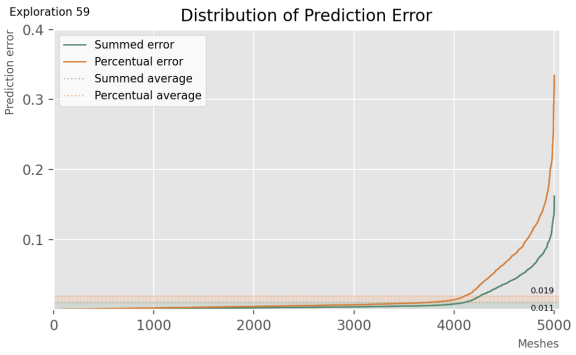
SURROGATE MODEL: 59

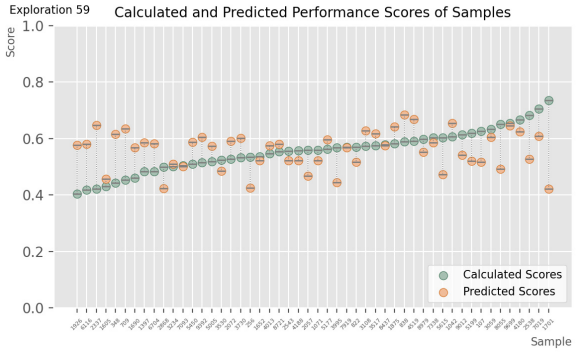
The attributes of this model are:

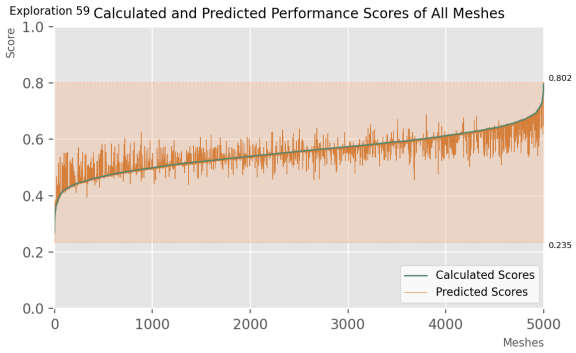
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.







The results in figure 4.2.29 show percentual error for each of the previously discussed surrogate model configurations. In figure 4.2.28, the losses of these models are compared. As training data differs, however, these can not be compared directly. The full results for the individual models (run to train on the various aspects of the overall performance score) can be found in Appendix IV under exploration 59 and 62-65.

The percentual error is calculated as follows:

$$percentual\ error = \frac{|predicted\ score - calculated\ score|}{calculated\ score}$$

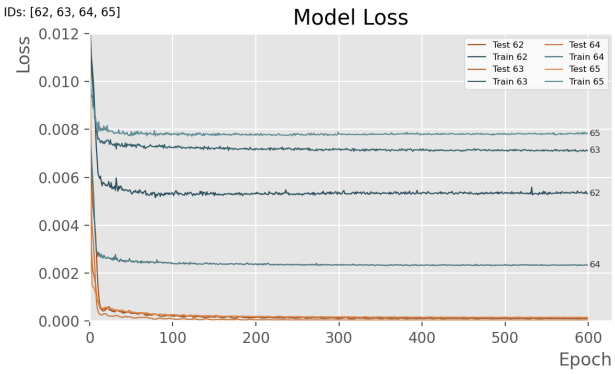


Figure 4.2.28. The model losses for explorations 62-65.
Own work.

Using the aforementioned options (1-3), the calculated performance scores (training data)

$$error_1 = average\ error_{overall}$$

$$error_2 = \frac{average\ error_{structural+material\ use} + average\ error_{stock\ similarity}}{2}$$

$$error_3 = \frac{average\ error_{structural} + average\ error_{material\ use} + average\ error_{stock\ similarity}}{3}$$

Figure 4.2.27. The formulas used to calculate errors shown in figure 4.2.28.
Own work.

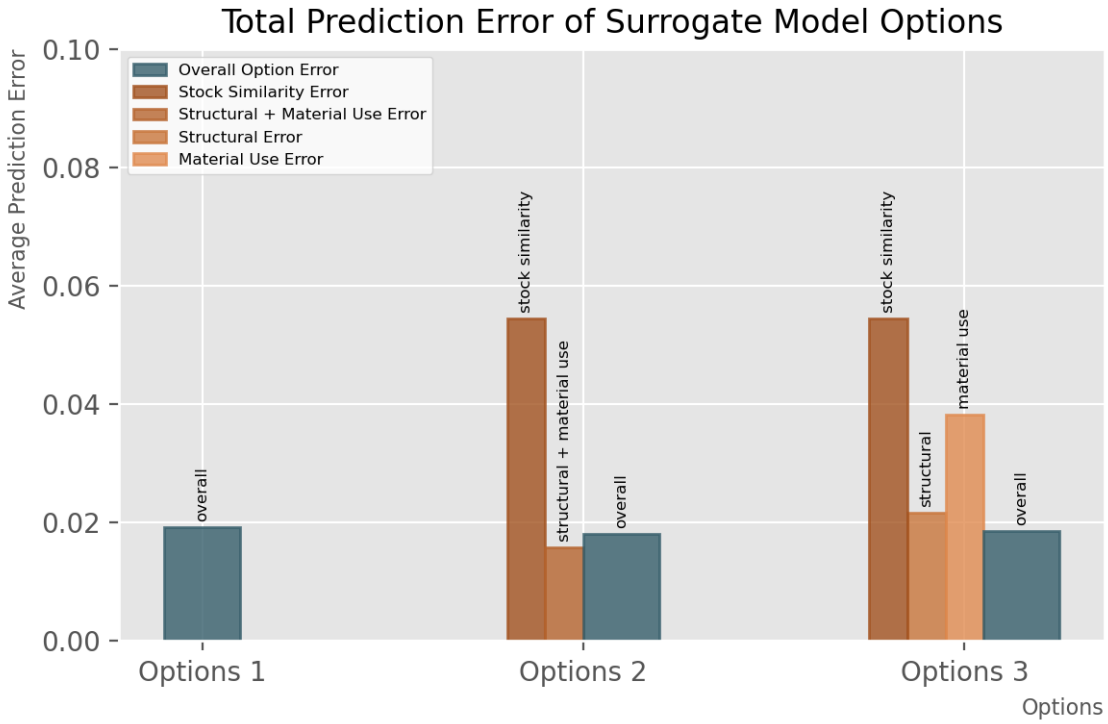


Figure 4.2.28. Total prediction error of surrogate model options for adjacency matrix input.
Own work.

were compared to the output of each option. The resulting total error of each option was calculated using the average error for each model architecture, as shown in the formulas in figure 4.2.27.

Figure 4.2.29 shows that all three options predict the overall performance score with a similar accuracy. Option 2 and 3 perform slightly better than option 1. However, these options do require multiple models to run, increasing computing time. Additionally, some further calculations are required to retrieve the overall performance score. On the other hand, option 2 and 3 allow for predictions of individual performance scores, which can be useful when optimizing or selecting designs.

4.2.5 HALF ADJACENCY MATRIX INPUT

The best working architecture (exploration 59) was also tested with half adjacency matrix input (exploration 74). The results of this are plotted in figure 4.2.30 The test and train loss both decrease to a set level, indicating that the model trains well. The right graph in figure 4.2.30 shows all predictions. This models accuracy is comparable to the surrogate models trained on full adjacency matrix input.

Option Comparison

For this input type, four more surrogate models were trained to predict the separated performance scores (exploration 75-78). The graph in figure 4.2.31 shows the model loss comparison (left) and the average prediction errors for each combination option (right), calculated as described in the previous section. All options have similar average prediction errors.

4.2.6 EDGE-VERTEX MATRIX INPUT

Edge-vertex matrix input is smaller per mesh than adjacency matrix input. For this input, the best performing architecture of the adjacency

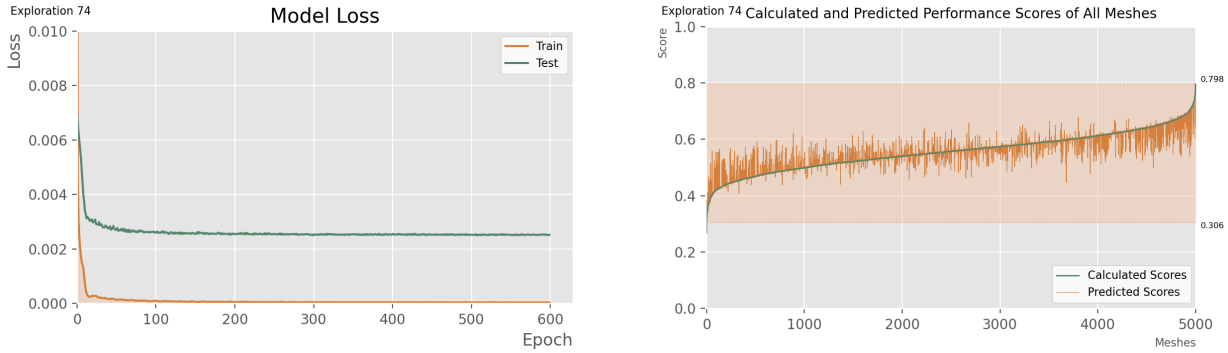


Figure 4.2.30. The model loss and calculated and predicted performance scores of all meshes for exploration 74.
Own work.

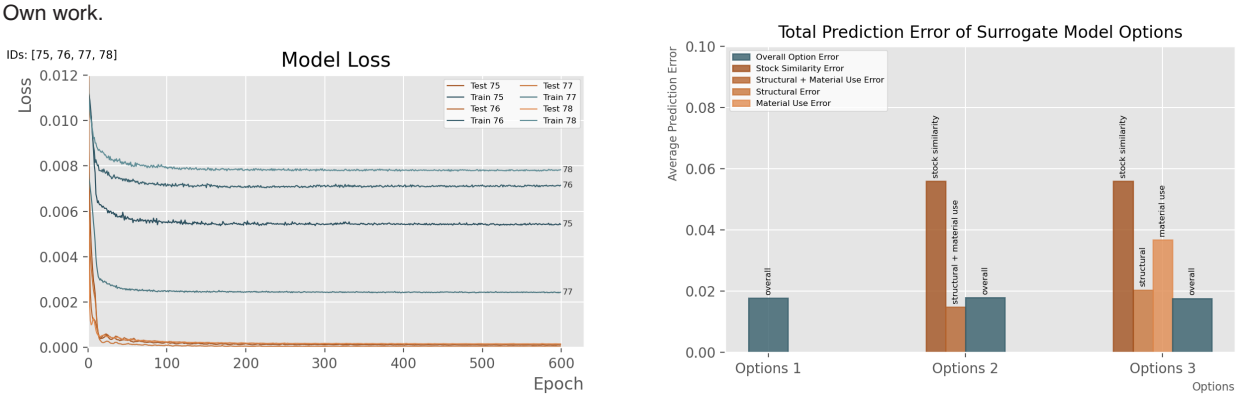


Figure 4.2.31. The model losses and Total prediction error of surrogate model options for half adjacency matrix input (exploration 75-78).
Own work.

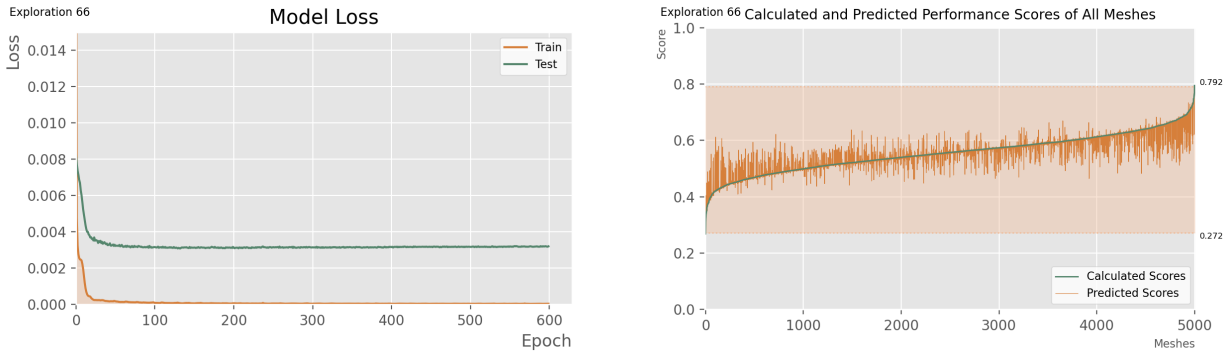


Figure 4.2.32. Model loss (left) and calculated and predicted performance scores of all meshes (right) for exploration 66 with edge-vertex input data.
Own work.

matrix-based surrogate models was used (exploration 59). Model loss and prediction performance for this model (exploration 66) are shown in figure 4.2.32. This showed that the data can also be used to train the surrogate model successfully; it's performance is comparable to adjacency matrix and half-adjacency matrix input.

Option Comparison

Again, a test was performed to assess the performance of the models for separated surrogate models (option 1-3). Figure 4.2.33 shows the average prediction error values with surrogate models trained on edge-vertex matrix data. These values are around 0.02 for each option. This indicates that all options predict the overall performance scores with similar accuracy. Therefore, all options can be used in the workflow.

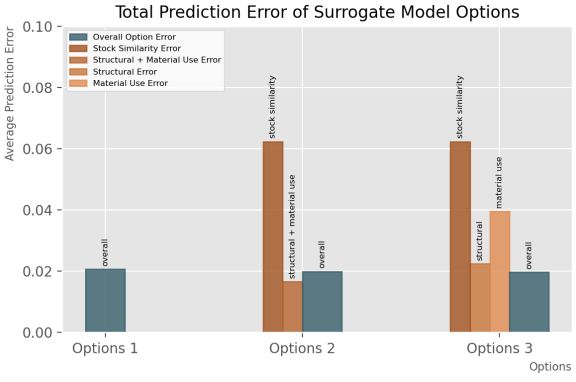


Figure 4.2.33. Total prediction error of surrogate model options for edge-vertex matrix input (exploration 67-70).
Own work.

4.3 VARIATIONAL AUTOENCODER

The variational autoencoder is trained to create a latent space from which new samples can be generated. In this section, various architectures and inputs will be tested to achieve this. In all cases, the data is split into 80% train data and 20% validation data. The validation data (or test data) is used to assess the ability of the encoder and decoder to encode and accurately reconstruct meshes. So, the input data should be encoded in such a way that the decoder output closely resembles the encoder input. The schematic in figure 4.3.1 shows the general architecture of the VAE (for coordinate input). Layer and latent space dimensions are specified later in this section.

4.3.1 COORDINATE INPUT

The VAE was first trained to generate based on coordinate input. This input was of the shape (41, 3, 1) and type 'float' with five decimals.

The loss function was defined as:

$$\frac{1}{2}\left[\left(\sum_{i=1}^{z}\mu_i^2+\sum_{i=1}^{z}\sigma_i^2\right)-\sum_{i=1}^{z}\left(log(\sigma_i^2)+1\right)\right]$$

In these explorations (0-16), the effects of the following parameters was tested:

- Architecture
- Dimension of the latent space
- Value b in $b * kl\_loss$

The following variations were made:

Latent space dimensions:
- 2
- 7
- 13
- 19
- 30
- 40
- 50
- 60

b -values in b*kl_loss:
- 0.01
- 0.1
- 0.5
- 1
- 5
- 10

Architectures:
- '82-42'
- '41'
- '82-41-21'
- '64-32'

The full reports on these explorations can be found in Appendix V.

The properties of the control model are:

Latent space dimensions: 7
b -values in b*kl_loss: 1
Architectures: '82-42'

This model is visualised in figure 4.3.1.

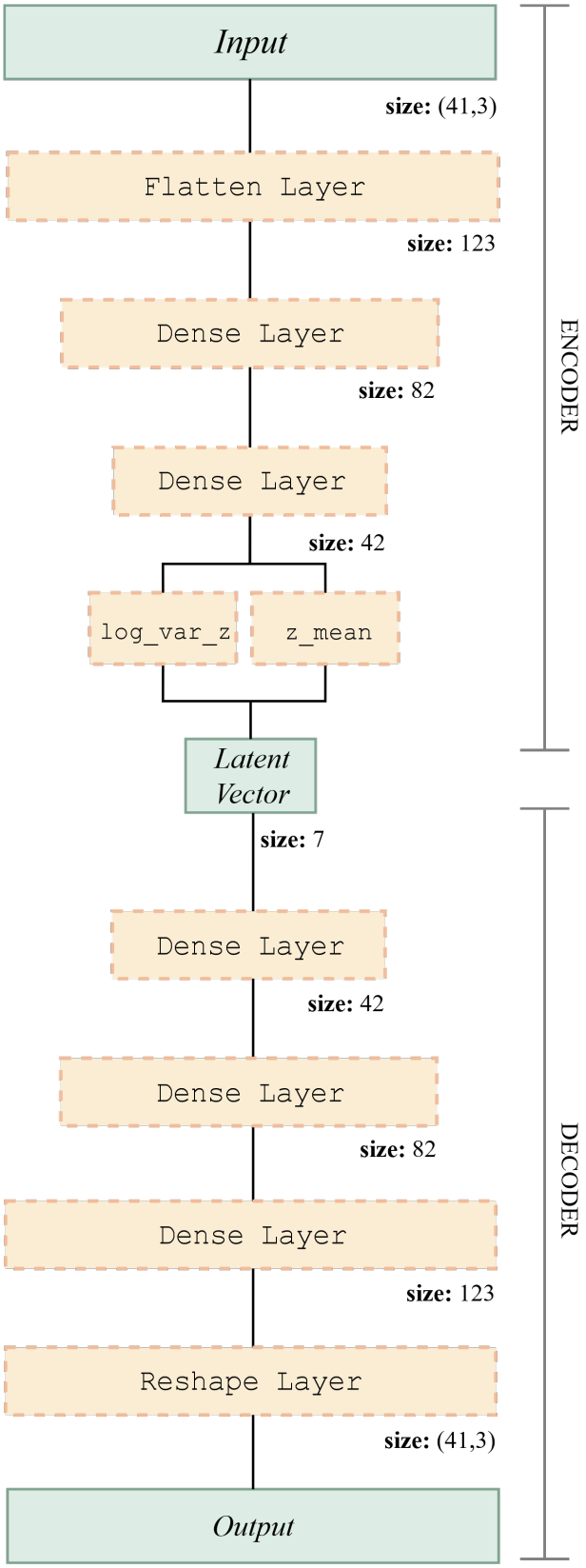


Figure 4.3.1. Schematics of 'VAE with coordinate input' architectures (control model).
Own work.

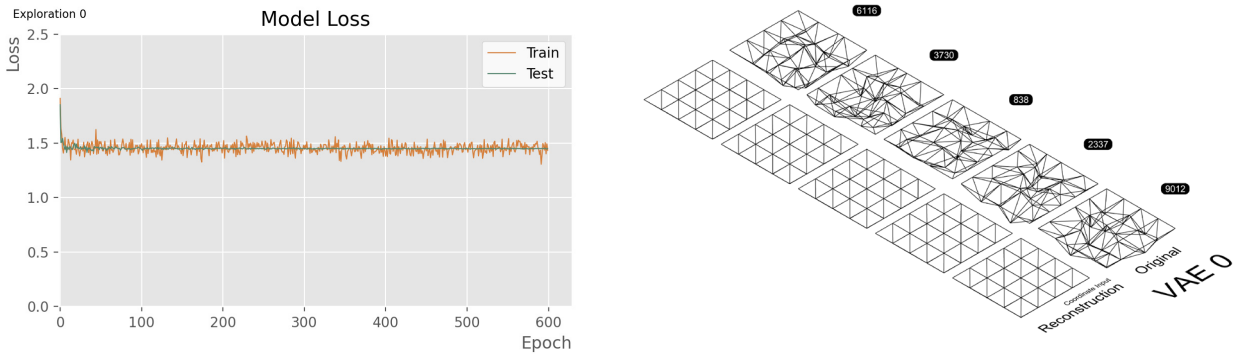


Figure 4.3.2. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 0 (the control model). Own work.

Results

Overall, the VAE models with coordinate input were unsuccessful in reconstructing the dataset meshes. The results of the control model (0) are shown in figure 4.3.2. For this and most other exploration models, output was largely similar to the "base mesh". Train and validations loss curves quickly flattened and the model did not train further.

Varying the dimension of the latent space and multiplication of the KL-loss did result in variation in some outputs. VAE model 10 and 11 – with highest latent dimension and lowest KL-loss multiplication respectively – gave the most variation in reconstructions (figure 4.3.3, top & bottom right). The model loss curves of these models also showed cleared training compared to the control model (figure 4.3.3, top & bottom left). In both cases, the validation loss is lower than the test loss on some parts of the curve, which could indicate that the model is not training properly.

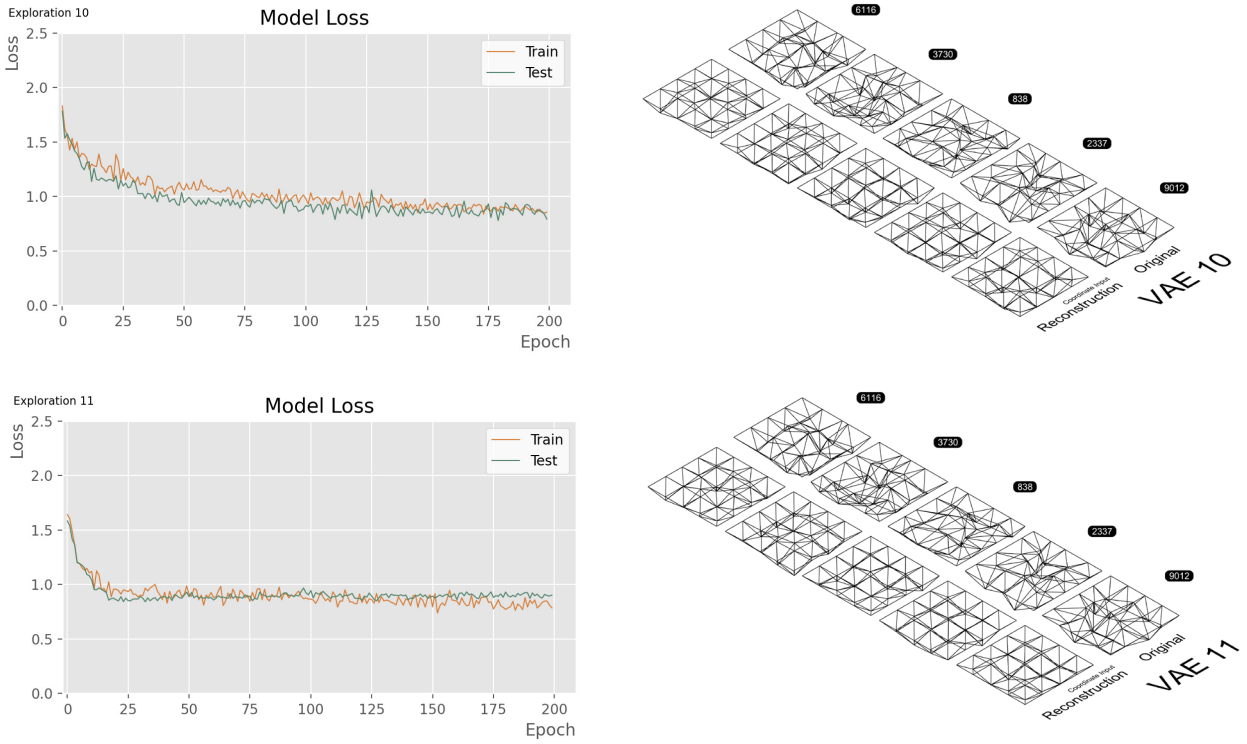


Figure 4.3.3. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 10 (top) and 11 (bottom). Own work.

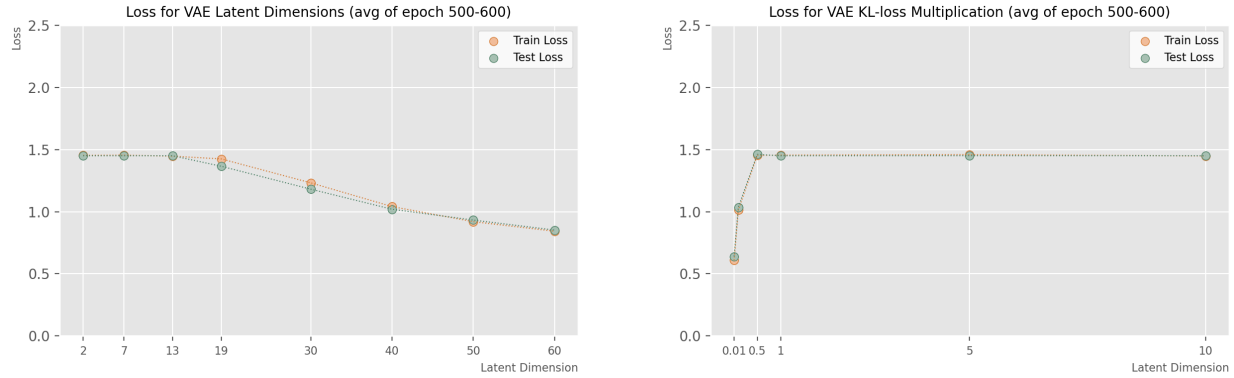


Figure 4.3.4. Loss for varying latent dimensions (left) and KL-loss multiplication (value b) for models with coordinate input. For these models, the average values of the final 100 epochs are used.
Own work.

The graphs showing the average loss of the final one-hundred epochs for models with varying latent dimension and KL-loss multiplication (figure 4.3.4) show that for this input type, a higher latent dimension and lower KL-loss multiplication result in lower test and train losses after training. Therefore, high latent dimensions and low KL-loss multiplications would be selected for further use. However, as reconstructions produced by these models are not representative of the input meshes. These models therefore are not used in the workflow.

Final layer activation

In the previous explorations, the last layer of the encoder was designed to have a 'sigmoid' activation type. As these models did not produce accurate reconstructions, another variation was tested. Here, the final layer was given a 'relu' activation. A comparison of the model loss graphs is shown in figure 4.3.4. The loss was similar for the two models, but the loss curves progress differently over the epochs. For further exploration, models were created with 'sigmoid' activation on the final encoder layer.

4.3.2 ADJACENCY MATRIX INPUT

The (243, 243) adjacency matrix input was used to train a second set of VAE models. Here, the same three parameters (architecture, latent dimension, KL-loss multiplication) were adjusted. The chosen values are largely similar to those tested with coordinate input. However, different architectures were tested to accommodate the larger input data.
These are:

- '972-243'
- '1944-972'
- '1944-972-486-243'
- '3888-1944-243'

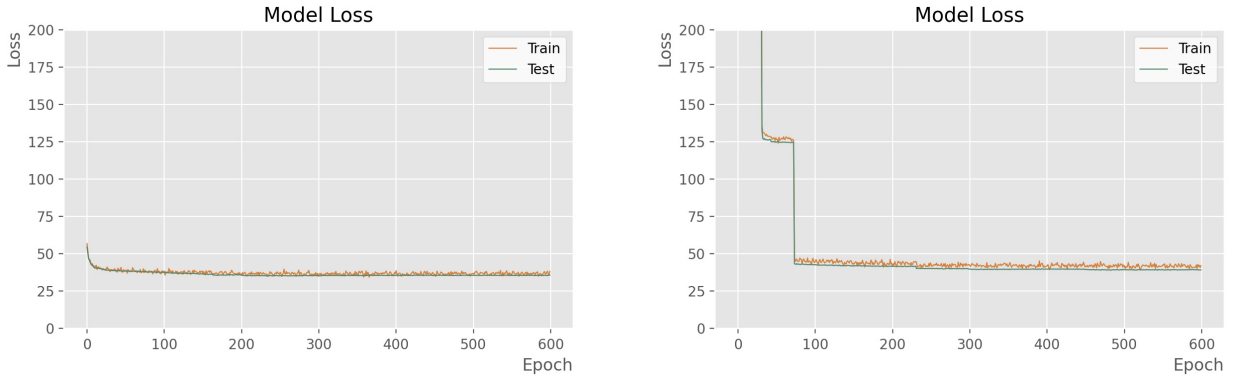


Figure 4.3.4. Model loss for the control model with 'sigmoid' activation on the final layer (left) and the control model with 'relu' activation on the final layer (right).
Own work.

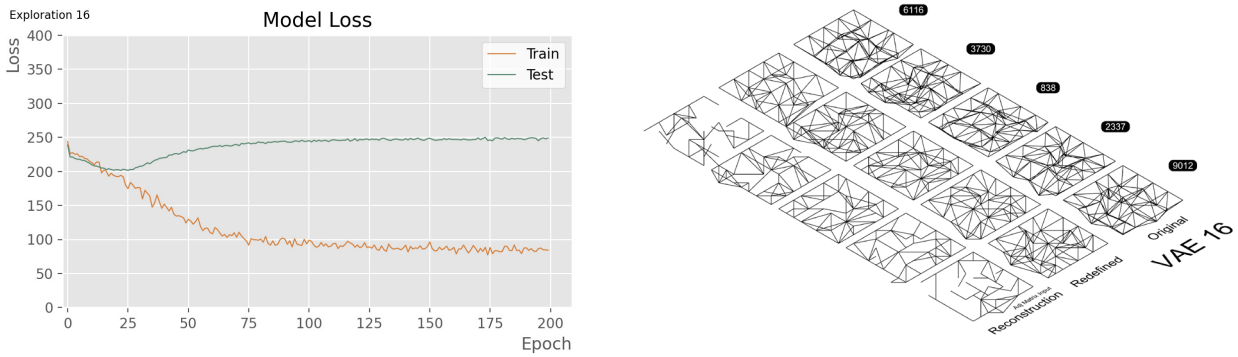


Figure 4.3.5. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 16 (the control model for adjacency matrix input).
Own work.

For the latent space dimension and parameter b in $b * kl\_loss$, the same values were tested as mentioned under section '4.3.1 Coordinate Input'.

Results

Training with adjacency matrix input gave vastly different results to training with coordinate matrix input. Generally, only part of the meshes was reconstructed correctly (figure 4.3.5). The training and validation loss curves indicate that the model was unable to learn (part of) the dataset well. This was the case for almost all explorations. The point at which the separation of the train and test curves starts varies among explorations with different parameters. In the graphs in figure 4.3.6, where the averages of the final epochs of these models are plotted, distance between train and test loss is relatively large as well.

For the model with KL-loss multiplication set as '10' (exploration 31, marked with a circle in figure 4.3.6) the curves are not (yet) diverging significantly. Nonetheless, the predictions of this model (exploration 31, figure 4.3.7) were significantly less accurate than its control model (exploration 16, figure 4.3.6).

Figure 4.3.7. Model loss for VAE exploration 31 (with KL-loss multiplication = 10)
Own work.

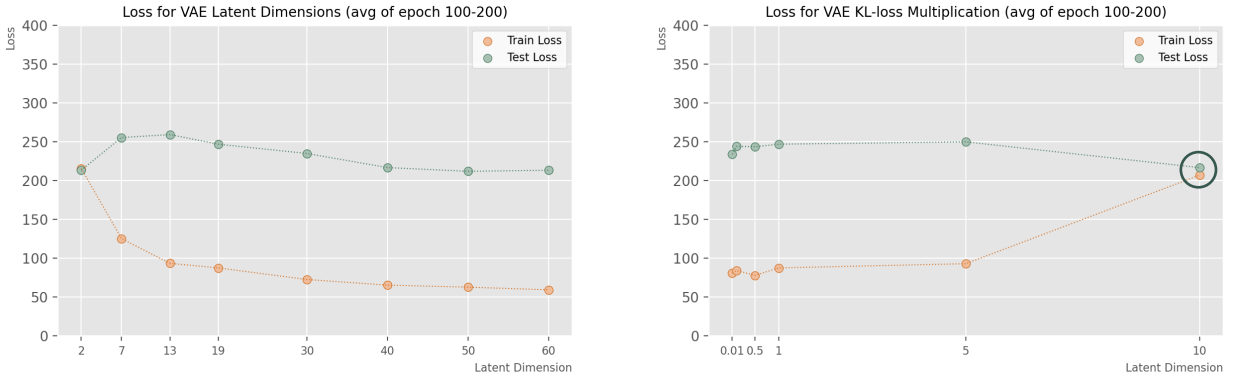


Figure 4.3.6. Loss for varying latent dimensions (left) and KL-loss multiplication (value b) for models with adjacency matrix input. For these models, the average values of the final 100 epochs are used.
Own work.

Reconstruction Threshold

To reconstruct the meshes, a threshold was used. If the values in the decoded data were larger than this threshold, the value was rounded to '1', indicating a connection. In the previous reconstructions shown in this section, a threshold of 0.5 was used. Figure 4.3.8 shows the reconstruction of a mesh with exploration 16 using various thresholds.

When a lower threshold is used, more edges are reconstructed. However, these models (using adjacency matrix input and mean_squared_error) were not able to produce reconstructions that accurately represent the input meshes.

Binary crossentropy vs. mean squared error

The same test set was run with a binary_crossentropy loss function. Full results for this can be found in Appendix V under VAE exploration 85-100. The train and test loss curves produced by these models display divergence, similar to those trained on mean squared error (figure 4.3.9). The diverging, however, reaches more extreme values. This can be seen in figure 4.3.9, where control model 16 is compared to control model 85. The mesh reconstructions of the models with binary crossentropy loss functions contain more edges than those from models trained on mean squared error. However, reconstructions are no more accurate. Some models with larger architectures or varying latent dimensions or KL-loss multiplication factors, are able to accurately reconstruct mesh 838. Still, none of these models was able to reconstruct a significant part of the geometry dataset with adequate accuracy. So, the models trained with (full) adjacency matrix data did not appear to be able to train and reconstruct meshes well.

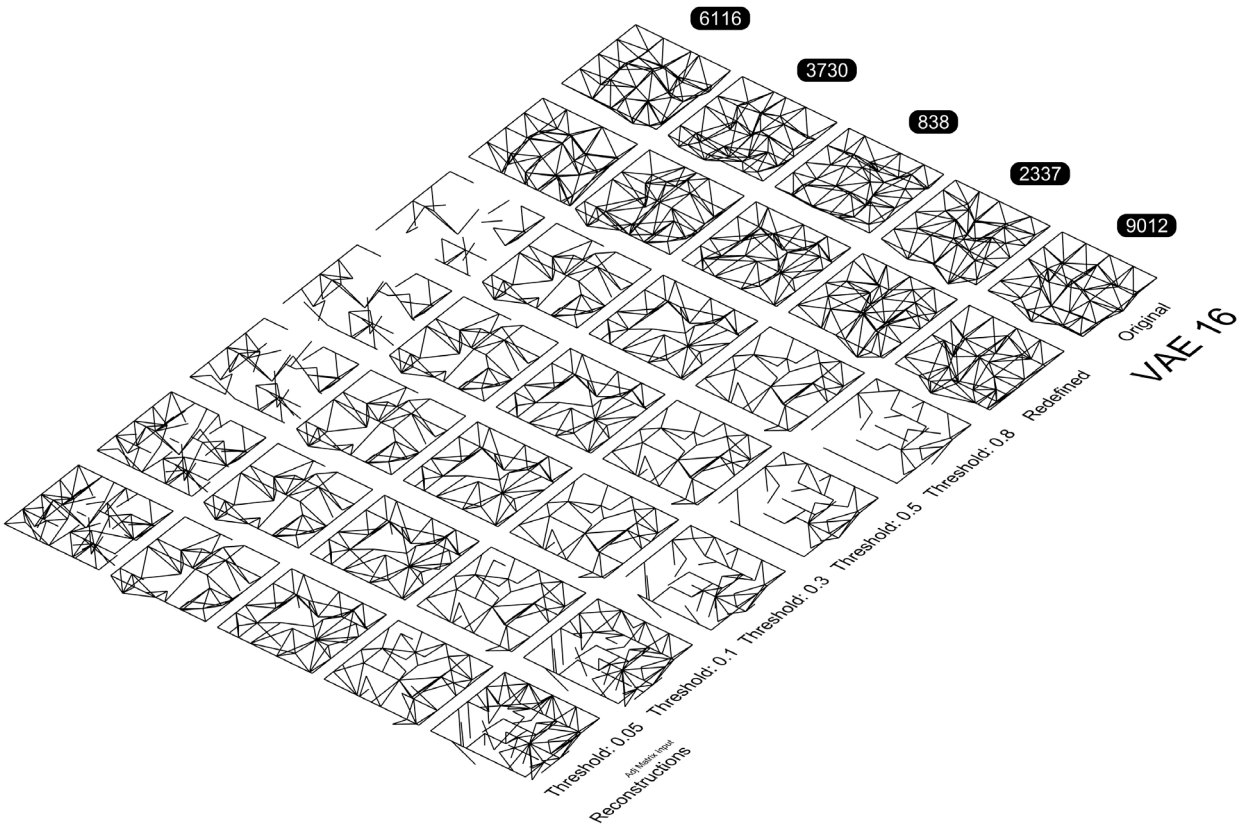


Figure 4.3.8. Sample reconstructions from VAE exploration 16 with threshold values of 0.05 to 0.8.
Own work.

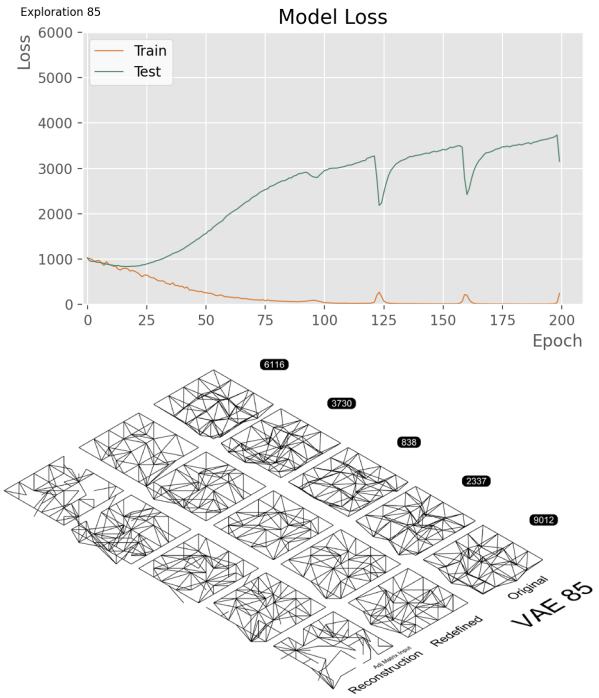


Figure 4.3.9. Model loss (top) and reconstructions of mesh samples (bottom) for VAE exploration 85 (the control model adjacency matrix input trained on binary crossentropy).
Own work.

Convolutional Layers

The previously discussed model sets are both based on architectures with densely-connected neural network layers. However, the adjacency matrix data is also suitable for use in networks with 2D convolution layers. These layers are typically used with image input. In a convolutional variational autoencoder, the input shape is defined as 4+D `batch_shape + (channels, rows, cols)` (Keras, n.d.). In this case, the number of channels is 1 (binary data). The rows and columns (cols) are both 243.

Versions of these models were trained with 'binary_crossentropy' (exploration 57) and 'mean_squared_error' (exploration 58) as loss functions. The architecture of these models is shown in figure 4.3.10. In the model loss graph for exploration 57 in figure 4.3.11, test loss starts to slightly increase after epoch 3. In the graph for exploration 58, train loss decreases as well. Here, test loss starts to increase again after epoch 5. Both models were able to make

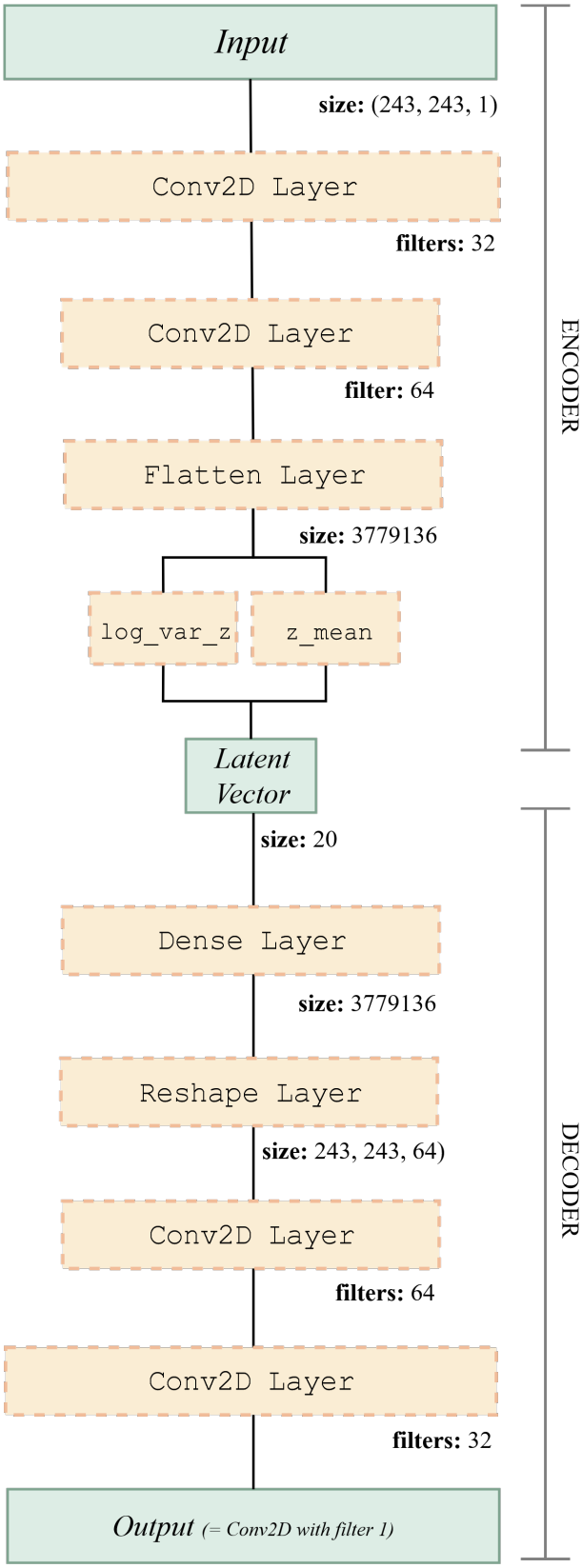


Figure 4.3.10. Schematics of 'Convolutional VAE with adjacency matrix input' architectures (control model).
Own work.

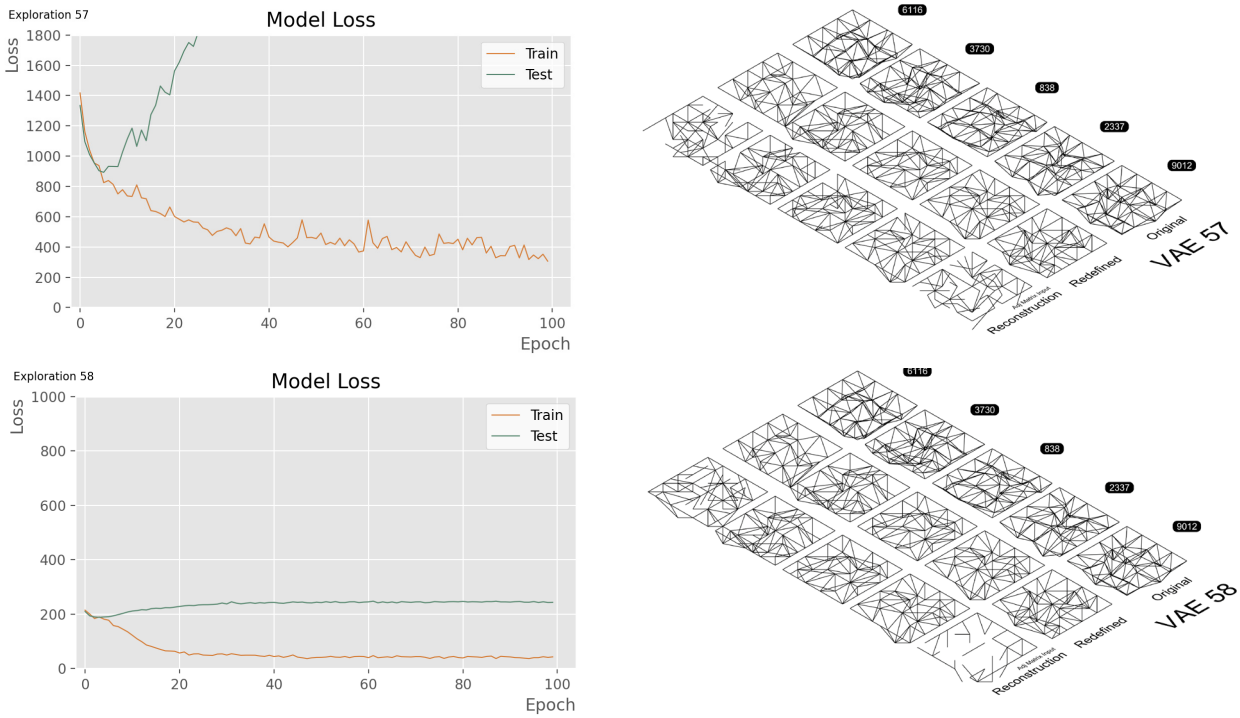


Figure 4.3.11. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 57 (top) and 58 (bottom). Own work.

accurate reconstructions for some meshes, including mesh 3730, 838 and 2337. However, neither produced accurate reconstructions for other meshes, including 6116 and 9012. As the model trained with mean squared error performed better, a full set of models was run with this loss function. These are exploration 117-132 (see full report in Appendix V).

Most of the models in this set show similar performance to the control model; reconstructions for sample meshes 3730, 838 and 2337 were mostly successful. Some models, including VAE 126 (figure 4.3.12) produced partly accurate reconstructions for mesh 6116 and 9012 too. However, these reconstructions still contained false edges and were missing significant parts of the meshes.

The model loss graph (figure 4.3.11, left & figure 4.3.12, left) show some divergence between train and validation loss. However, this effect is significantly less extreme compared to the densely connected VAE models.

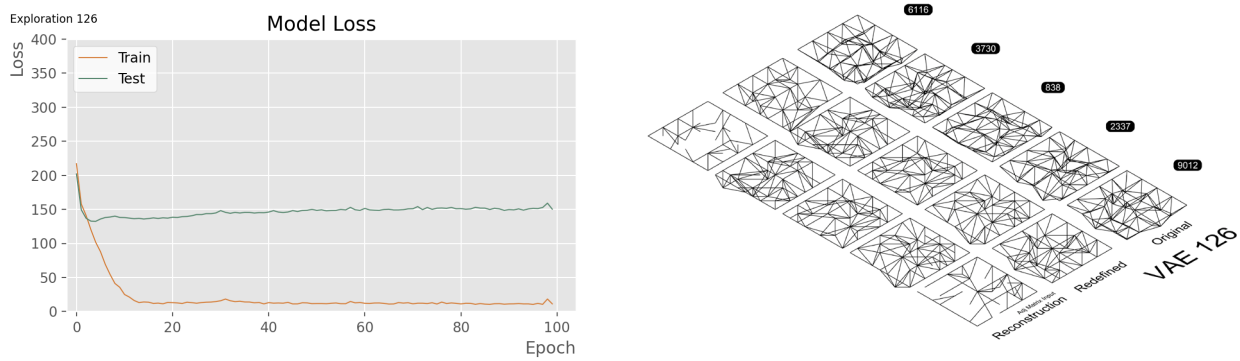


Figure 4.3.11. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 57 (top) and 58 (bottom). Own work.

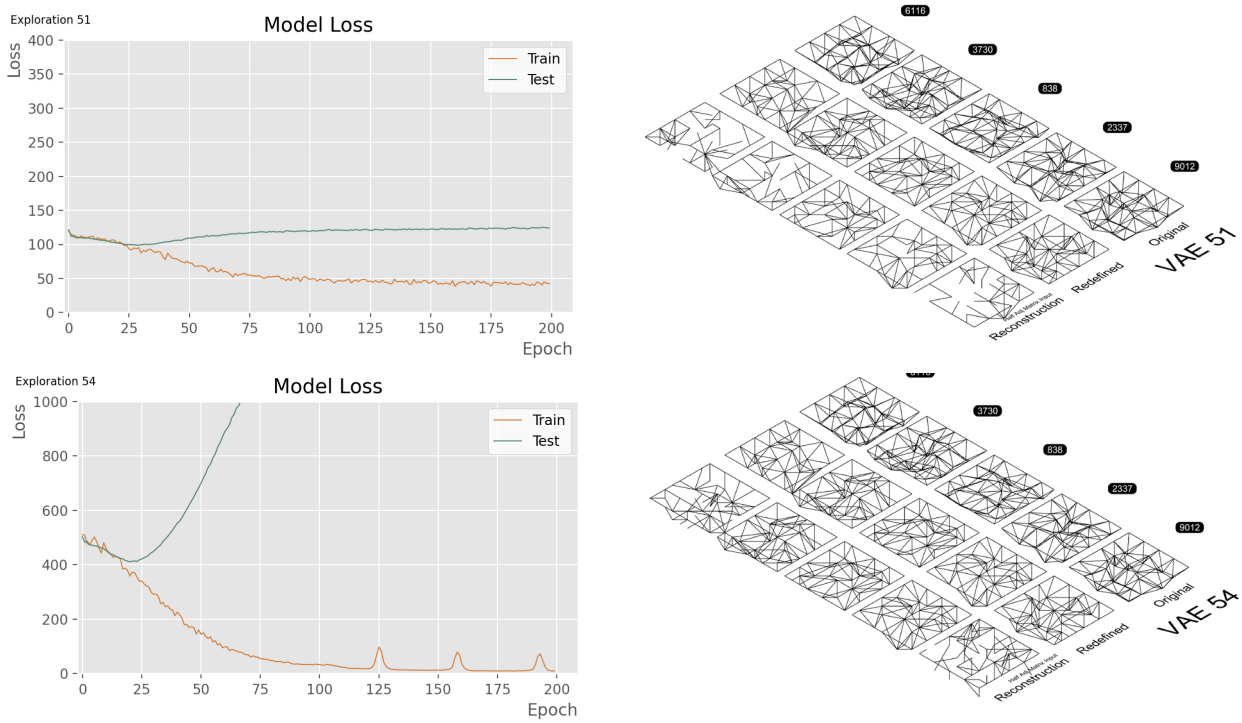


Figure 4.3.12. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 51 (top) and 54 (bottom). Own work.

4.3.3 HALF ADJACENCY MATRIX INPUT

In another exploration, half of the adjacency matrix was used as input. These models were trained with 'binary_crossentropy' loss functions (figure 4.3.12, right). In the left graph of figure 4.3.12, 'mean_squared_error' was used. The loss curves of these models differ significantly. In both cases, the train and test loss curves diverge after approximately 25 epochs. However, the divergence is again more extreme in the model where 'binary_crossentropy' is used. Still, this model produces more accurate recreations of the sample meshes.

A full set of explorations was therefore done (60-75) with binary crossentropy. In this set, the following architectures were tested:

- '1089-121'
- '1089-363-121'
- '3267-1089-121'
- '3267-1089-363-121'

For the latent space dimension and parameter b in $b * kl\_loss$, the same values were tested as mentioned under section '4.3.1 Coordinate Input'.

Figure 4.3.13 on the next page shows train and test loss for the control model. These plots look similar to those of the models trained on full adjacency matrix input. However, reconstructions are accurate for a significant part of the test meshes, as shown on the right in figure 4.3.13 (reconstruction threshold = 0.5). The reconstructions of mesh 3730, 6116 and 9012 were highly accurately with this input.

The reconstructions under different thresholds is shown in figure 4.3.14. Overall, meshes that are reconstructed with a lower threshold had more edges. While mesh 3730, 838 and 2337 are accurately reconstructed under any threshold, mesh 6116 and 9012 do not get reconstructed well regardless of threshold value. New edges that are generated with a lower threshold value do not represent the edges in the original mesh.

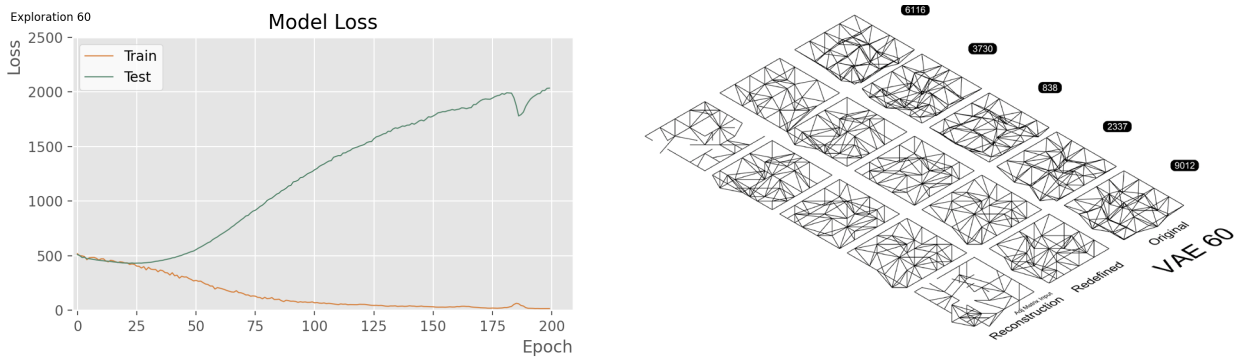


Figure 4.3.13. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 60 (the control model half adjacency matrix input).
Own work.

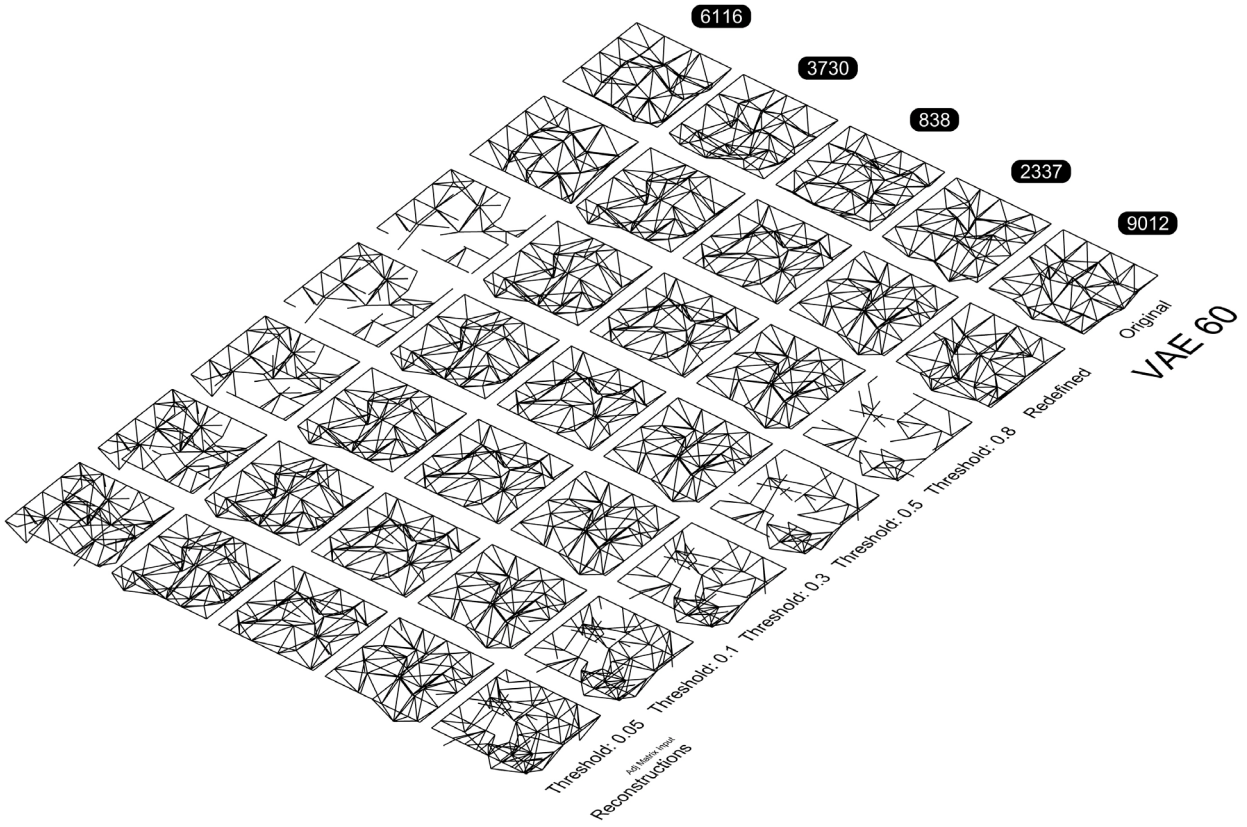


Figure 4.3.14. Sample reconstructions from VAE exploration 60 with threshold values of 0.05 to 0.8.
Own work.

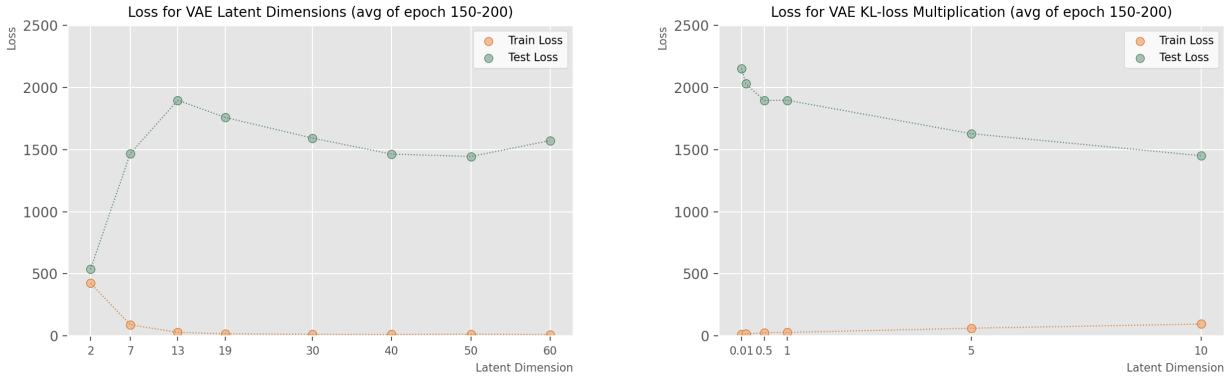


Figure 4.3.15. Loss for varying latent dimensions (left) and KL-loss multiplication (value b) for models with half adjacency matrix input and binary crossentropy. For these models, the average values of the final 50 epochs are used.
Own work.

Figure 4.3.15 shows the average loss over the epoch 150 to 200 (the final 50 epochs) for models with varying latent dimensions and KL-loss multiplications. It should be noted that train and test loss curves of most models diverge at different starting points and rates.

A latent dimension of 2 (figure 4.3.16, top left) reconstructs very few meshes, mainly in the four corners of the mesh. A latent dimension of 7 (figure 4.3.16, top right), produces more accurate results. Best reconstruction, however, is reached at latent dimensions of 30+ (figure 4.3.16, bottom left). Still, higher latent dimensions of 40, 50 or 60 do now show further improvement in the mesh samples (figure 4.3.16, bottom right), even though the graph in figure 4.3.15 (left) shows that validation loss is lower at latent dimensions of 40 or 50.

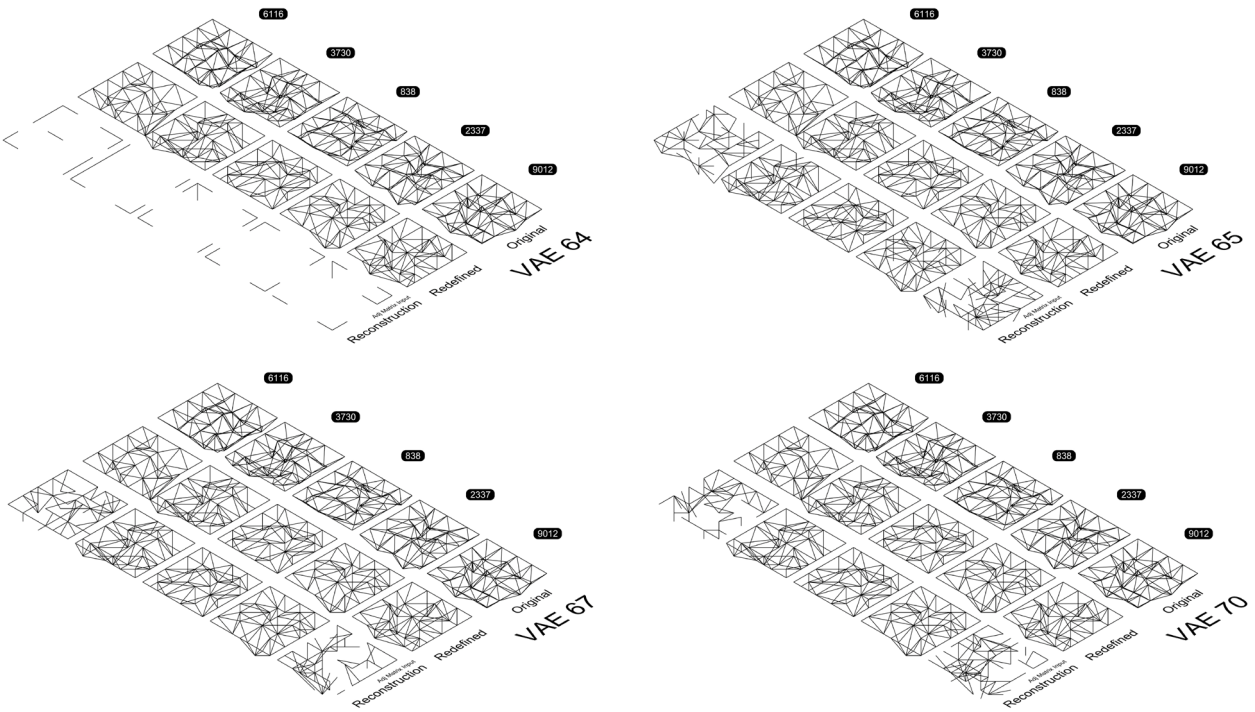


Figure 4.3.16. Mesh sample reconstructions for VAE explorations 64 (lat_dim=2), 65 (lat_dim=7), 66 (lat_dim=30) and 70 (lat_dim=60)
Own work.

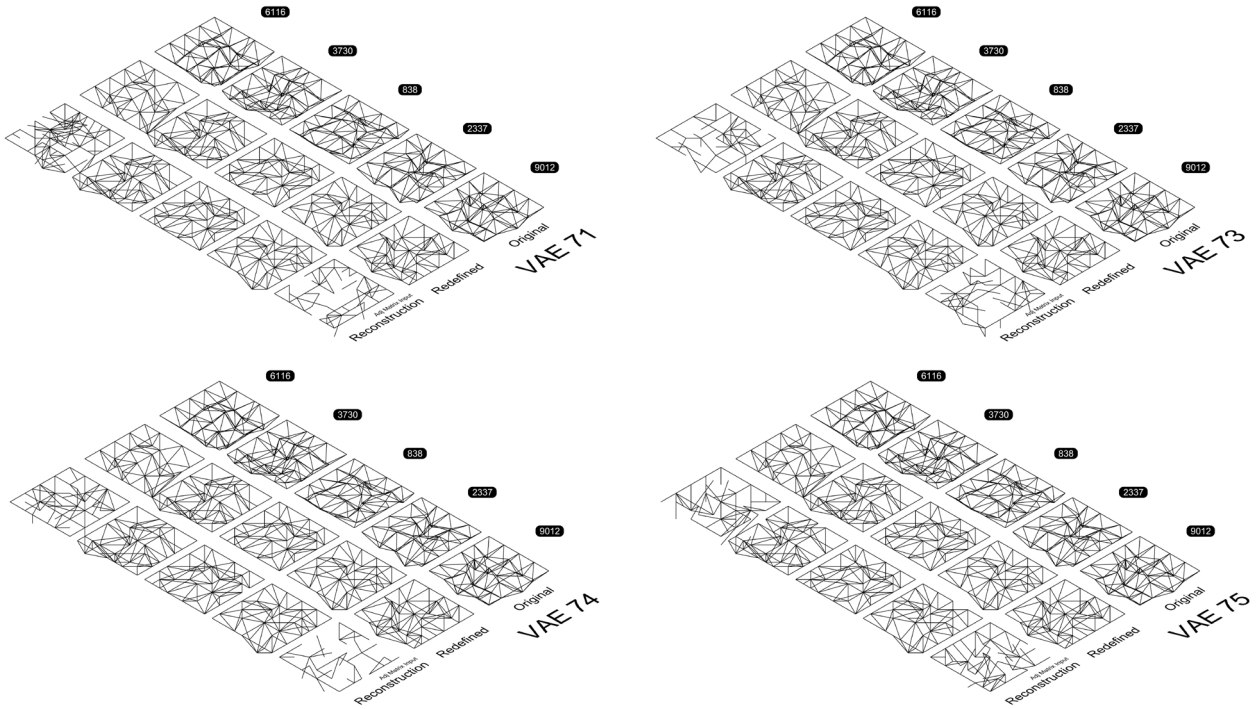


Figure 4.3.17. Mesh sample reconstructions for VAE explorations 71 (b=0.01), 73 (b=0.5), 74 (b=5) and 75 (b=10)
Own work.

similar to previous models sets, test loss is lower for smaller multiplications factors in KL-loss. However, the opposite is true for validation loss; a larger KL-loss multiplication produces smaller validation loss.

Sample mesh reconstructions (figure 4.3.17) indicate that KL-loss multiplications values of 1 or lower produce the most accurate results. The reconstructions of mesh 3730, 838 and 2337 are highly accurate in exploration 60 (control) and 71-73 . in reconstructions from exploration 74 and 75, with a b-value of 5 and 10 respectively, false and missing edges were generated. Other samples (including 6116 and 9012) were not accurately reconstructed by any of the tested models. While none of these reconstructions are considered representative, exploration model 63 was the most accurate (figure 4.3.18). Here, some edges, especially near the corners and sides of the meshes, were correctly reconstructed. For this reason, the architecture '1089-363-121' was concluded to be the best. The results presented here do not represent the full exploration. All results can be found in Appendix V.

4.3.4 EDGE-VERTEX MATRIX INPUT

The (128,243) edge-vertex matrix input was used to train a fourth set of VAEs; exploration 32-47. This set was trained with 'mean_squared_error'. Another set, trained with 'binarycrossentropy', will be presented later in this section.

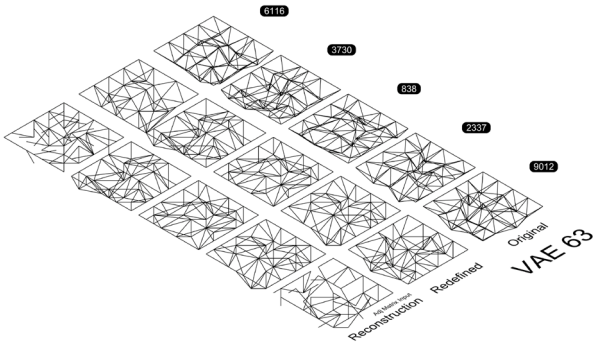


Figure 4.3.18. Mesh sample reconstructions for VAE exploration 63 (lat_dim =13, b=1, architecture='1089-363-121')
Own work.

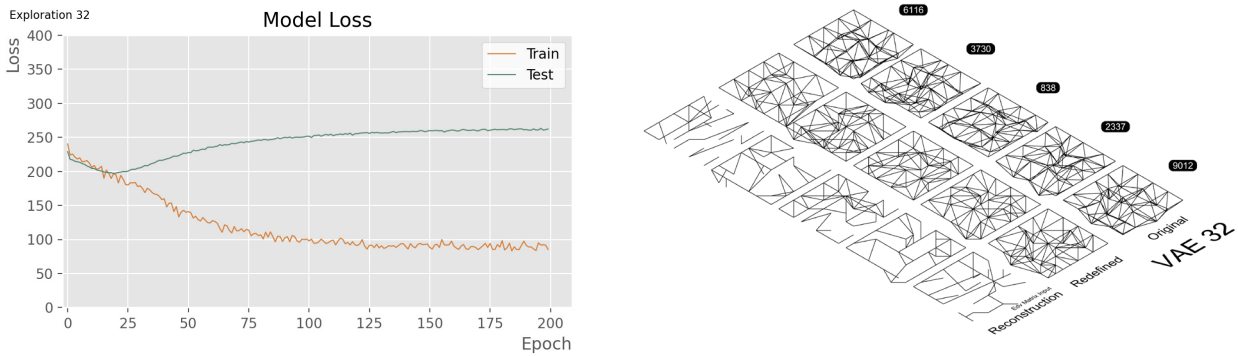


Figure 4.3.19. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 32 (the control model for edge vertex matrix input on mean squared error).
Own work.

In these sets, the following architectures were tested:

- '972-243'
- '1944-972'
- '1944-972-486-243'
- '972-486-243'

For the latent space dimension and parameter b in $b * kl\_loss$, the same values were tested as mentioned under section '4.3.1 Coordinate Input'.

The loss curves for training of models with the edge-vertex matrix input were similar to those of the models trained with adjacency matrix input; curves show divergence after varying points, depending on their parameters. This is shown in figure 4.3.19 and Appendix V. Reconstructions from decoded data of the control model (figure 4.3.19) shows that this input type results in partly reconstructed meshes. Reconstructed meshes from these models also contain edged that are not present in the original input mesh more often than the control model for adjacency matrix input.

Latent dimension and KL-loss multiplication graphs (figure 4.3.20) look similar to those from the VAE models with adjacency input as well. Again, the training and validations loss curves of the model with a KL-loss multiplication of '10' are not (yet) diverged as much as the models it is compared too, explaining why the train loss and test loss curves in this graph show convergence.

Overall, the explorations of this model type did not perform significantly better than its control model. However, some models were able to reconstruct were able to reconstruct sample

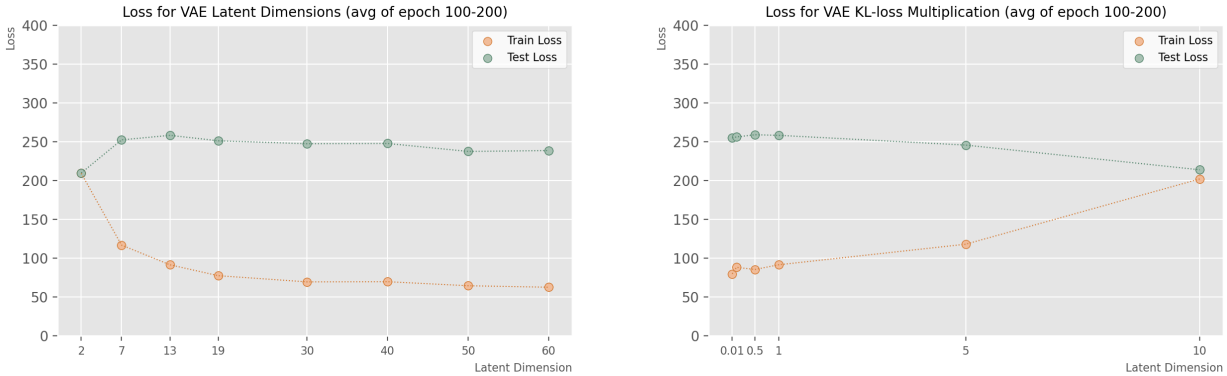


Figure 4.3.20. Loss for varying latent dimensions (left) and KL-loss multiplication (value b) for models with edge vertex matrix input and mean squared error. For these models, the average values of the final 100 epochs are used.
Own work.

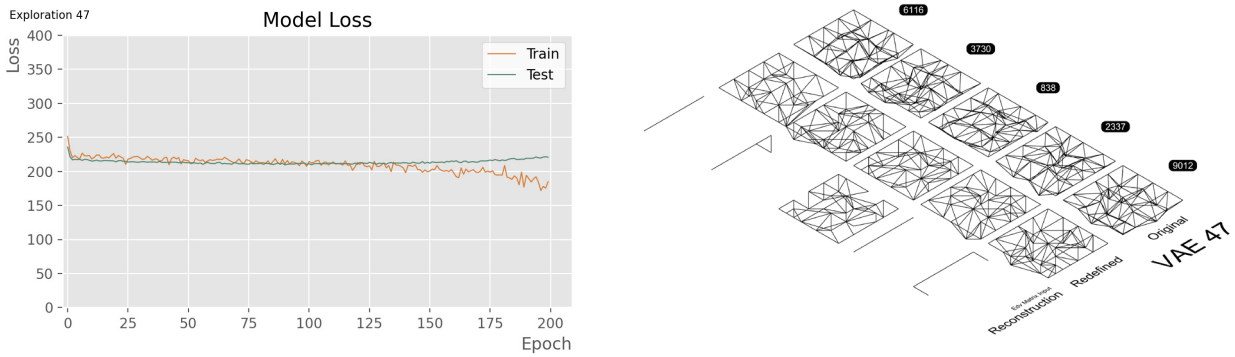


Figure 4.3.21. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 47 (the control model for edge vertex matrix input on mean squared error).
Own work.

'838' more closely compared to the other test samples (figure 4.3.21). This was the most clear in model 47 (latent dimension 13 and KL-loss multiplication of 10).

Binary crossentropy vs. mean squared error

As stated, a set with the same parameters was trained with binary crossentropy; exploration 101-116. Comparing the reconstructions of the control model of this set (exploration 101, figure 4.3.22) to those of the control model with mean squared error shows that neither model is succesfully reconstructing a majority of mesh samples. However, in exploration 101 (with binary crossentropy) mesh 838 was reconstructed significantly better.

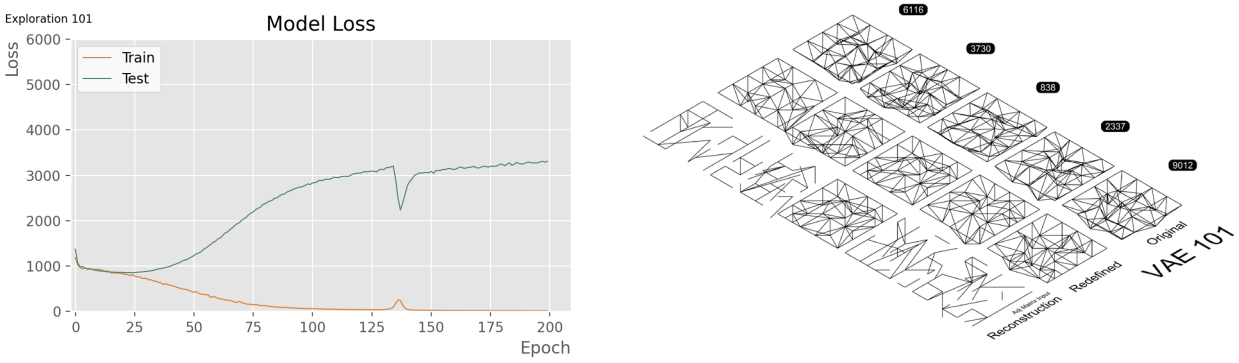


Figure 4.3.22. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 101, the control model for edge-vertex matrix input and binary crossentropy.
Own work.

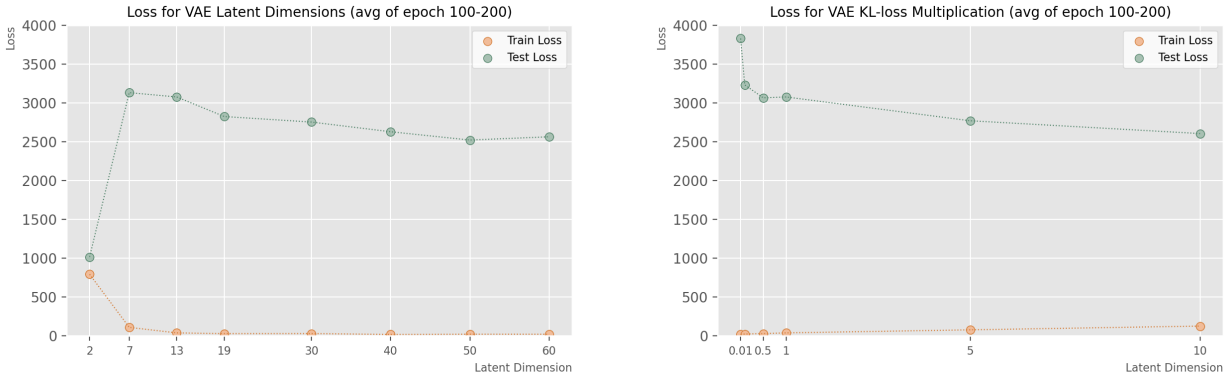


Figure 4.3.23. Loss for varying latent dimensions (left) and KL-loss multiplication (value b) for models with edge vertex matrix input and binary crossentropy. For these models, the average values of the final 100 epochs are used.
Own work.

Figure 4.3.23 shows loss for varying latent dimensions and KL-loss multiplications. These graphs follow the same pattern as the models trained on (half) adjacency matrix input and binary crossentropy.

4.3.5 VERTEX MOVEMENT & STEP MOVEMENT INPUT

Two models were created to test training on vertex movement ((41,3), float-type data) and vertex step movement ((123,11), binary data). Full documentation on these models can be found in Appendix V.

Both models, however, performed similarly to the models with coordinate inputs; train and test loss did not drop significantly and outputs were not representative of the input mesh. As a results, these input types were not explored further.

4.3.6 SURROGATE MODEL ON ENCODED DATA

The previously found best performing surrogate model architecture (section 4.2) was used again to evaluate surrogate model performance on VAE encoded data. The VAE used here was trained on half adjacency matrix data with architecture '1089-363-121', latent dimension 13, KL-loss multiplication 0.2 and binary crossentropy.

The resulting model loss for the surrogate model is shown in figure 4.3.25. Both train and validation data decrease in this model, indicating successful training. Compared to the various models shown in section 4.2, however, test loss remains slightly higher. In the case of exploration 59 and 74 (trained on adjacency and half-adjacency matrix data) for example, test loss quickly decreased to levels below 0.003. (figure 4.2.30). In exploration 66, trained on edge-vertex matrix data, test loss reached levels below 0.004 (figure 4.2.32).

Option Comparison

For this input type, four more surrogate models were trained to predict the separated performance scores (exploration 80-83). The graph in figure 4.3.26 shows the model loss

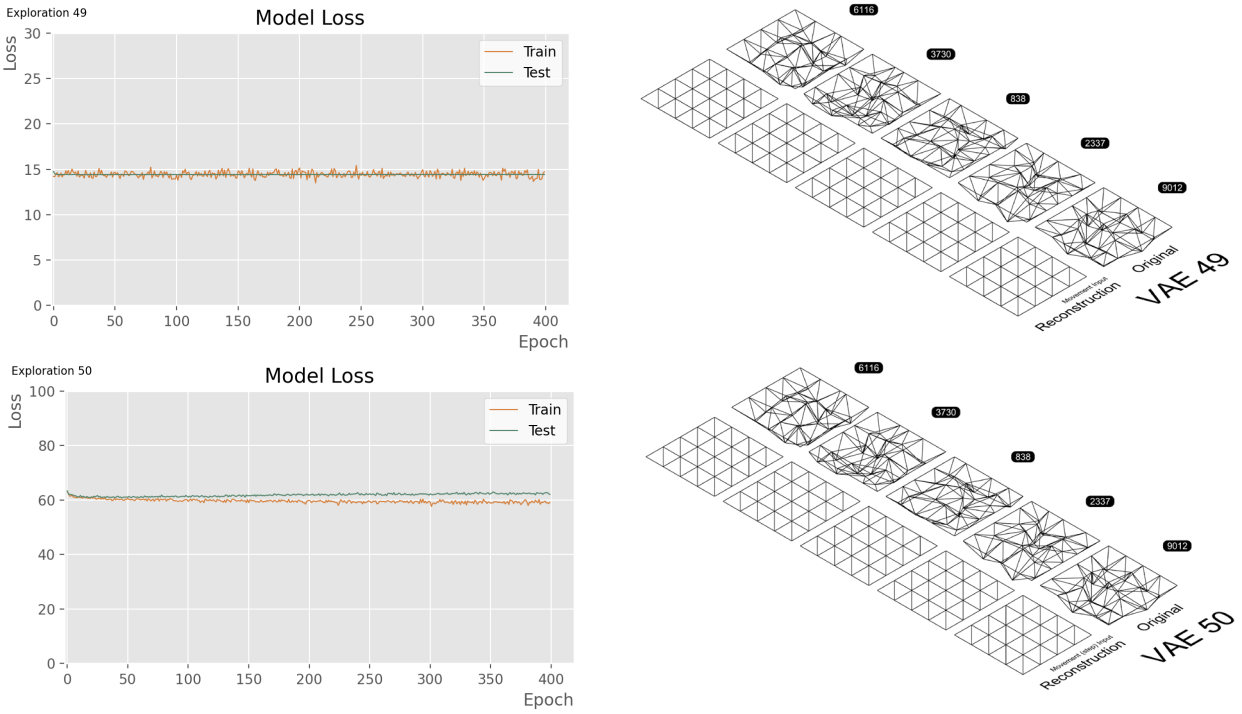


Figure 4.3.24. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 49 and 50.
Own work.

comparison (left) and the average prediction errors for each combination option (right), calculated as described in the previous section. In previous comparisons, all options resulted in similar average prediction errors. However, this is not the case here; option 2 and 3, where results from various surrogate models are combined, are significantly higher. This can be explained by the average (percentual) error per model. These are shown below. Between brackets, the same value is shown for the surrogate models trained on half adjacency input as described in section 4.2.

Overall performance:

- 0.032 (compared to 0.018)

Structural performance:

- 0.038 (compared to 0.020)

Material use performance:

- 0.075 (compared to 0.037)

Structural + material use performance:

- 0.031 (compared to 0.015)

Stock similarity performance:

- 0.096 (compared to 0.056)

So, for the model trained on encoded VAE data all error margins are larger. The 'calculated and predicted performance scores' graphs (found in appendix IV) show these larger margins as well. This can have an effect on the overall scores as calculated from multiple, seperate surrogate models. As each aspect in the calculation of the predicted overall performance score has a larger error margin, multiplying them can more easily result in larger error for the predicted overall values. This could be an explanation for the high values found in figure 4.3.26 for surrogate model option 2 and 3. Nonetheless, as the surrogate models tested in section 2.4 show higher accuracy, those were used for the overall workflow.

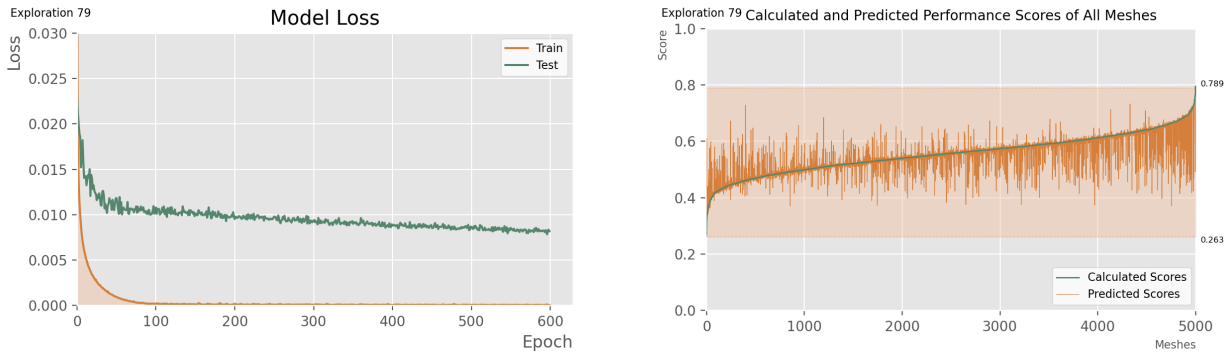


Figure 4.3.25. The model loss and calculated and predicted performance scores of all meshes for exploration 79.
Own work.

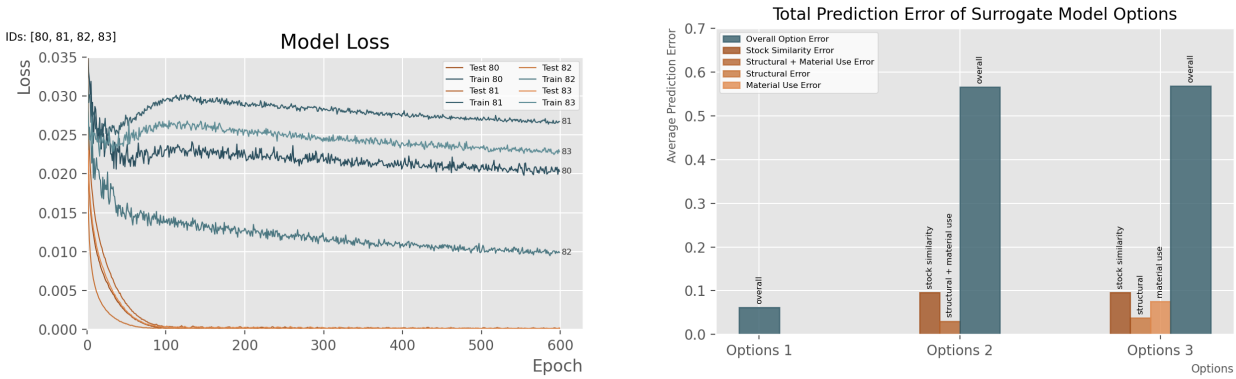


Figure 4.3.26. The model losses and Total prediction error of surrogate model options for encoded VAE input (exploration 80-83).
Own work.

4.3.7 LATENT DIMENSION MINIMIZATION

Limiting the dimension of the latent space can be done to minimize the computational complexity of the VAE models. Therefore, another set of explorations was done (exploration 133-139). These models had the following properties:

Latent space dimensions: 3-9
b -values in b*kl_loss: 0.3
Architectures: '1089-363-121'

Full results are presented in Appendix V. Figure 4.3.27 shows the model losses and mesh reconstructions for VAE 133, 135 and 137 with latent dimensions of 3, 5 and 7 respectively. For latent dimensions under 4, the sample meshes are not reconstructed accurately. In these models, the meshes are missing a significant number of edges. These mesh representations can not be used in for realistic performance assessment. Latent dimensions between 5 and 7 perform better; in three of five cased, a majority of edges is reconstructed accurately. However, edges are missing and false edges are generated still. Latent dimensions of 7 and up produce highly accurate representations (again, for three of five sample meshes). Therefore, a latent dimension of 7 is applied in the final VAE.

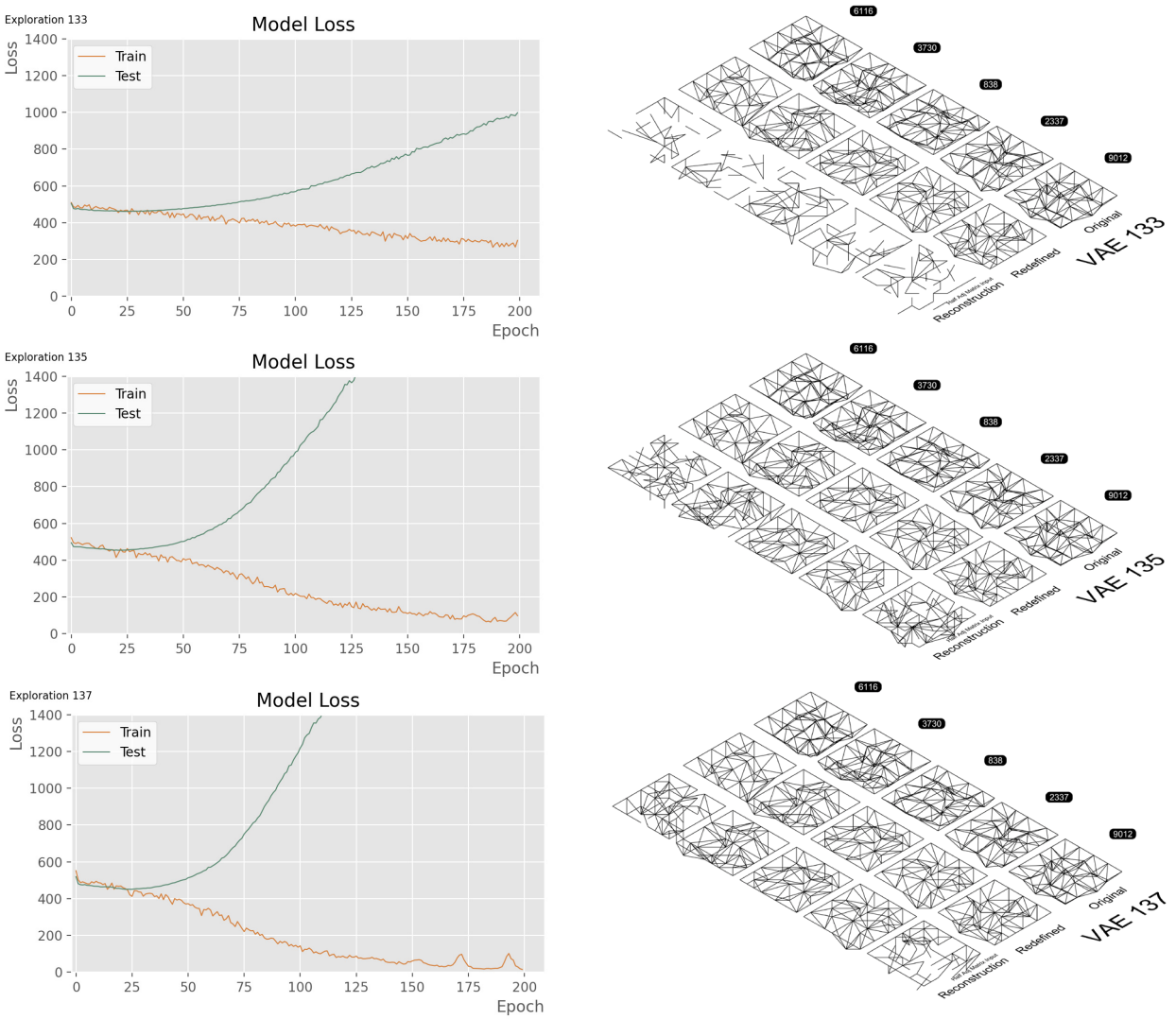


Figure 4.3.27. Model loss (left) and reconstructions of mesh samples (right) for VAE exploration 133, 135 and 137. Own work.

4.3.8 SPLIT VARIATIONAL AUTOENCODER

As the 3D structure of the case study presented in this study may be too complex for the workflow, another approach was tested, with a focus on the top and bottom layers of the case study geometry. Here, each layer is generated by a separate VAE; the first VAE generates the 2D top layer, the second generated the 3D bottom layer. In a secondary part of the workflow, the in-between structure is automatically generated based on this data. This data then provides the (half) adjacency matrix data for the full geometry, which can then be send to any of the surrogate model configurations tested in this work. This output can then be used in two sets of gradient descent, each backpropagating to one VAE, so that an optimized structure can be found. The full workflow with implementation of this approach is shown in figure 4.3.28.

A major benefit of this approach, compared to the previously documented approached, is that it ensures the geometry structure always matches that of the base mesh, so that, in all cases the in-between structure connects the corner points of a upper layer cell to one vertex on the bottom layer. Secondly, it is expected that the latent dimensions of each VAE in this approach can be decreased compared to previous attempts. This simplifies the process of gradient descent, which could result in better optimization results. The full code for this can be found in Appendix III.

To ensure that the workflow would run, the VAE models were tested with the following hyperparameter settings:

VAE 1 (2D top structure):

Latent space dimensions:	7
b -values in b*kl_loss:	0.5
Architectures:	'1089-363-121'

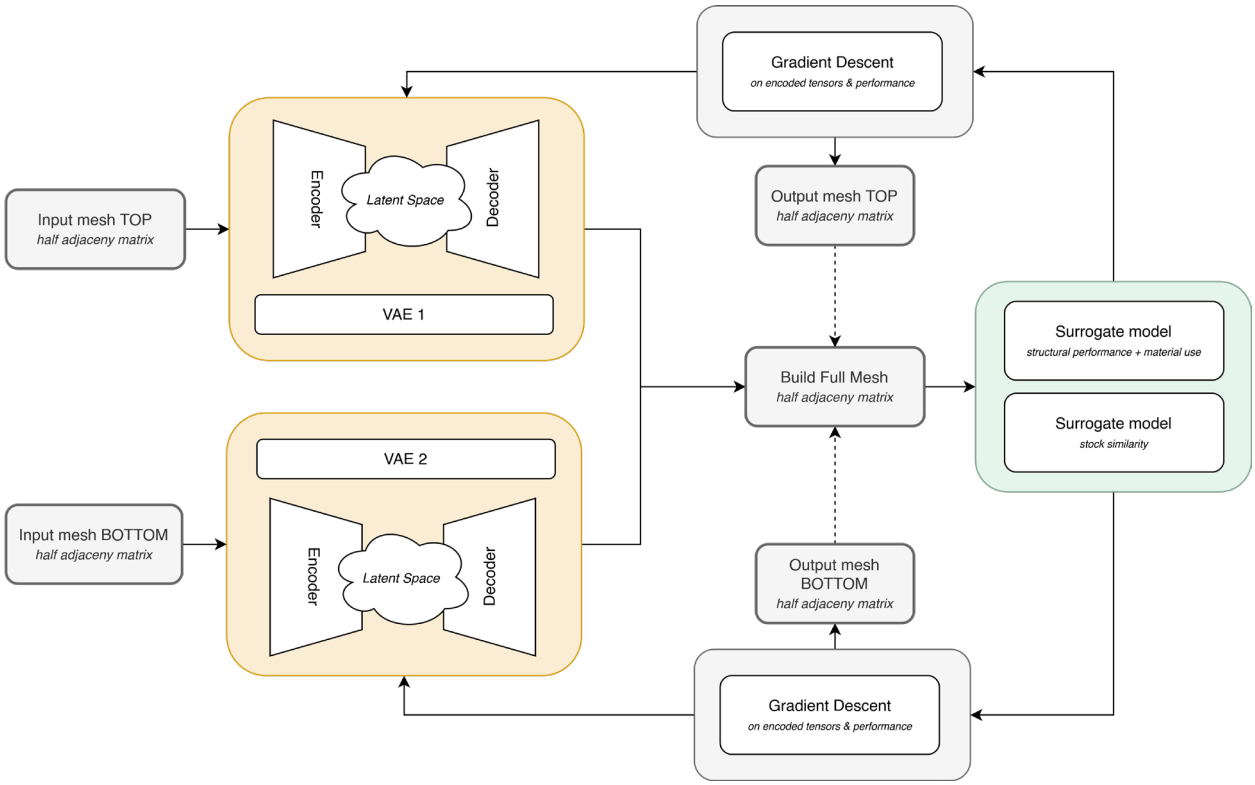


Figure 4.3.27. The workflow with integration of two separate VAEs; one for the top layer of the mesh and one for the bottom layer of the mesh.
Own work.

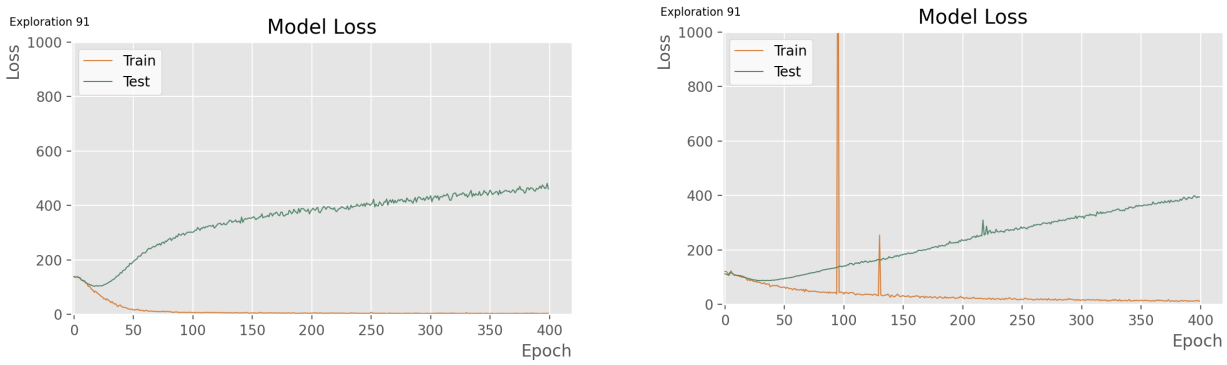


Figure 4.3.28. Model losses for split VAE models for top and bottom structures.
Own work.

VAE 2 (3D top structure):

Latent space dimensions:	13
b -values in b*kl_loss:	0.5
Architectures:	'1089-363-121'

The model loss curves for these VAE models are shown in figure 4.3.28 As this method introduces limits to the applicability of the workflow, this method was not applied in the final workflow.

4.4 GRADIENT DESCENT

Gradient descent (GD) is used for optimization of the VAE output. Since a large performance score indicates better overall performance, gradient descent was used to locate maxima. Meshes that were reconstructed well by the VAE were used as input for the gradient descent algorithm. First, two meshes from the test set were selected:

- Mesh 838
- Mesh 2337

The final goal of the gradient descent algorithm and the AI workflow together, is to produce meshes that perform better than the original randomly generated meshes.

The ten best performing manually generated meshes were therefore excluded from the dataset. These were therefore not part of the test nor training data of the VAE. Rather, these

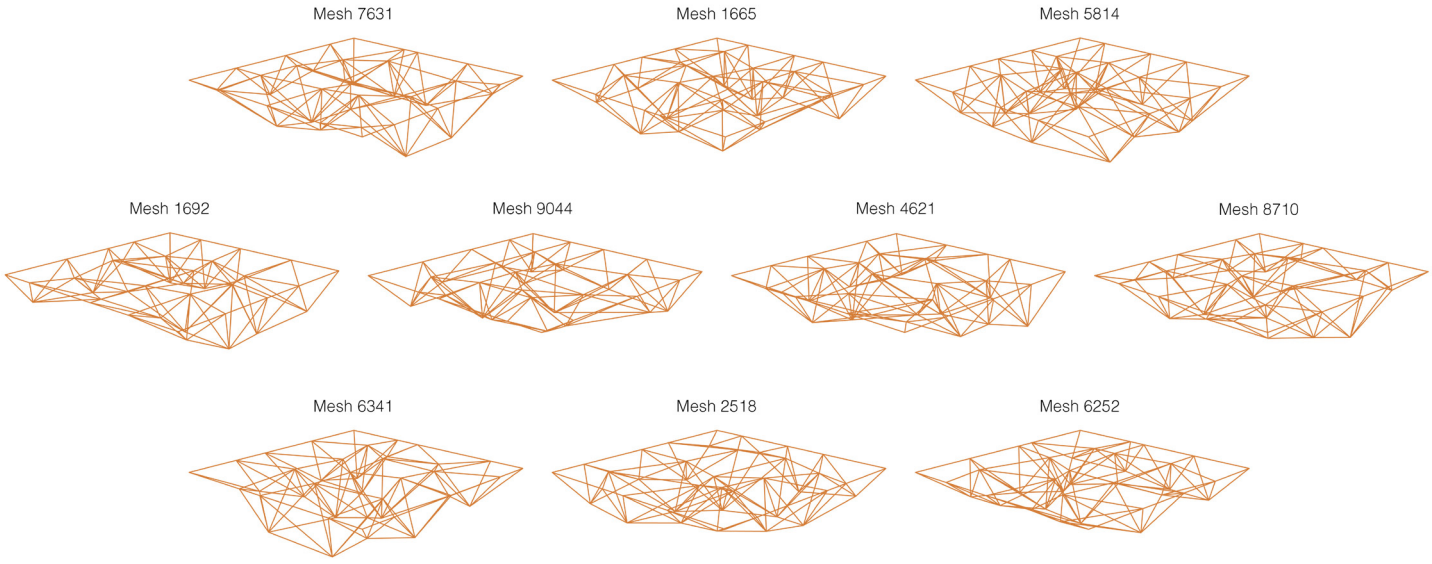


Figure 4.4.1. The ten best performing meshes from the original dataset.
Own work.

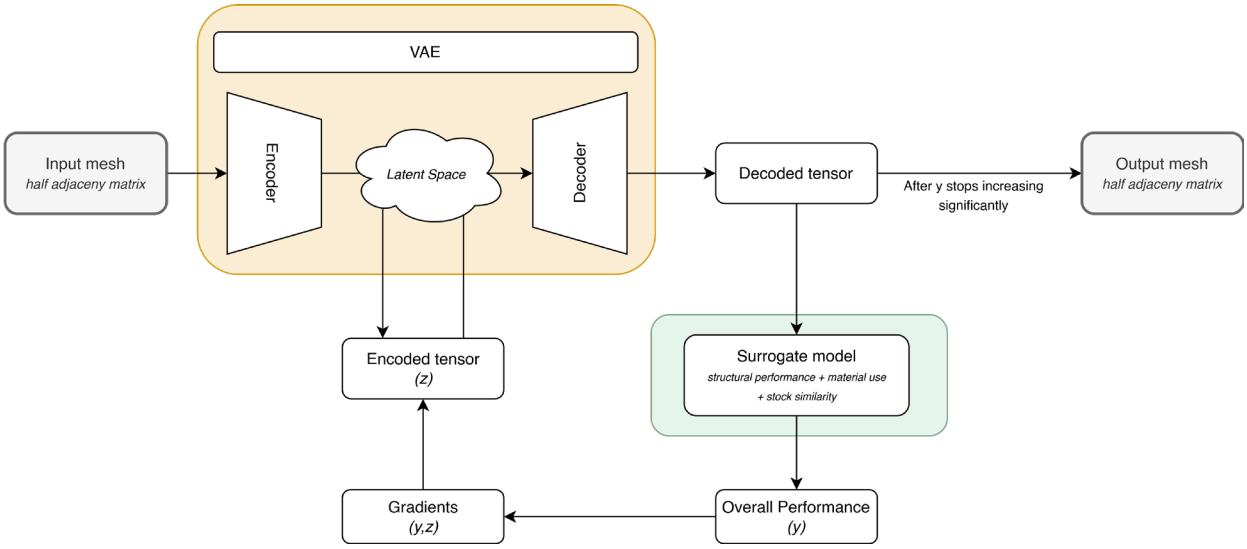


Figure 4.4.2. Gradient Descent and the integration into the full generative framework.
Own work.

samples were used to compare to the best samples generated by the VAE at the GD stage. These ten samples are: `[7631, 1665, 5814, 1692, 9044, 4621, 8710, 6341, 2518, 6252]`. They are shown in the schemes in image 4.4.1. From the ten best performing meshes in the dataset, three meshes were selected:

- Mesh 1665
- Mesh 1692
- Mesh 2518

These meshes were chosen, because the reconstructions produced by the various VAE architecture was generally the best for these meshes.
The gradient descent model is set to run for a minimum of 100 and a maximum of 1000 iterations. If the increase in predicted performance score becomes less than 0.001%, the algorithm is stopped early. The learning rate varies per mesh. The process is shown in figure 4.4.2.

4.4.1 HALF ADJACENCY MATRIX INPUT – DENSELY CONNECTED VAE – SINGLE SURROGATE OPTIMIZER

First, optimizing through a single surrogate model (option 1) was performed. Here, the best performing densely connected VAE architecture was used.
For each time the GD model was run. gradients, the newly generated meshes, and their predicted performance scores are reviewed. The full documentation of this can be found in Appendix VI.

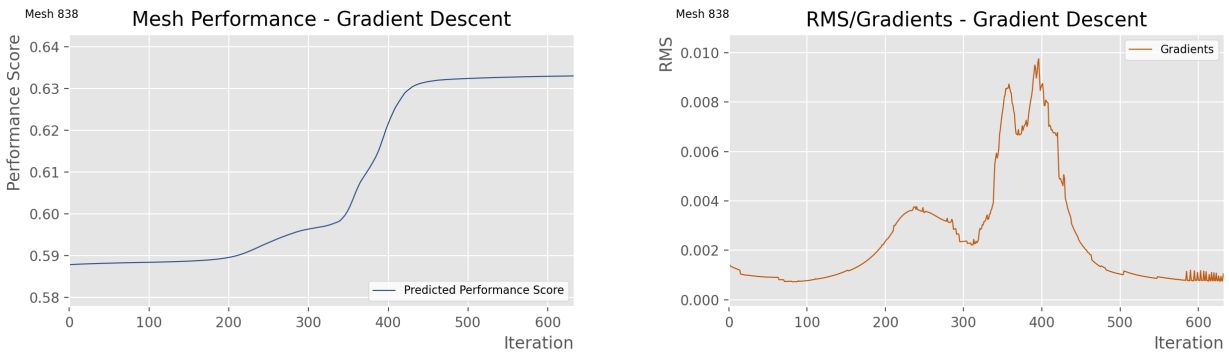


Figure 4.4.3. Predicted Mesh Performance and RMS Gradients for GD with starting mesh 838 (model ID: 0)
Own work.

For all five tested samples, gradient descent worked successfully; the predicted performance score of the model architecture (parameter y) increased in all cases (figure 4.4.3). It should be noted, that this does not mean that the newly generated meshes performed better. This is verified through calculation of actual performance scores with the simulations/ models described in Chapter 2

In these tests, three learning rates were chosen: 0.5, 2.5 and 5. The best performing learning rate varied per mesh. For mesh 838, a learning rate of 0.5 worked well.

FURTHER EXPLORATION

For this architecture, further exploration was done to assess the impacts of VAE regularisation loss multiplication (b), VAE latent dimension, GD learning rate and the effects of rounding the mesh matrices before inputting them into the surrogate model. A control set was also created. Full documentation again can be found in Appendix VI. Results of this control model were comparable to the previously discussed results. All meshes' predicted performance improved over iterations and generated outputs were different from the original input meshes.

Regularisation term

Tested regularisation terms are 0.01, 0.3 (control model) and 5. Low regularisation loss multiplications resulted in smaller predicted improvements. As a result, the iterations were limited to 100. Regardless, valid outputs were produced. A high value, however, produces some invalid meshes with missing edges. An examples of this is shown in figure 4.4.5.

Latent dimension

As stated previously, VAEs with lower latent dimensions sometimes recreated meshes with slight inaccuracies. However, lowering latent dimension simplifies the GD process. Therefore, models with varying latent dimensions were explored. Besides the control model with latent dimension 13, two models with latent dimension 5 and 20 were reviewed. Although graphs look promising, the model with low latent dimensions produced meshes with missing edges. An example of this is shown for input mesh 1665 in figure 4.4.6. Using a VAE with high latent dimensions, however, showed a similar issue, as shown in figure 4.4.7 for input mesh 2337.

Learning rate

Various learning rates were also explored, namely: 0.001, 2.5, 10 and 30. Predicted performance and gradient graphs for various learning rates presented differently. Lower learning rates produced smoother curves for predicted performance. Examples are shown in figure 4.4.9 for mesh 838. Generated outputs, however, were very similar to the control model (figure 4.4.8, for input mesh 838).

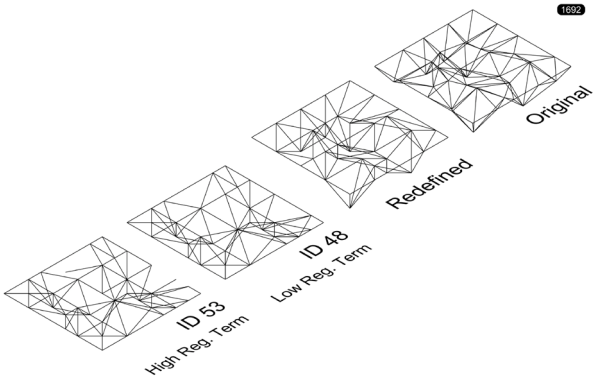


Figure 4.4.5. Mesh 1692 GD output for low and high regularisation loss multiplications.
Own work.

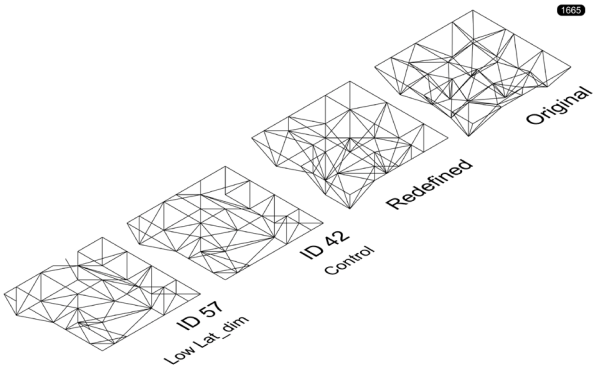


Figure 4.4.6. Mesh 1665 GD output for low latent dimension, compared to the control mode.
Own work.

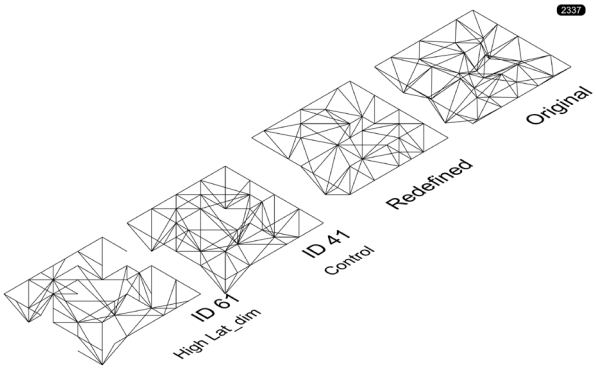


Figure 4.4.7. Mesh 1665 GD output for low latent dimension, compared to the control mode.
Own work.

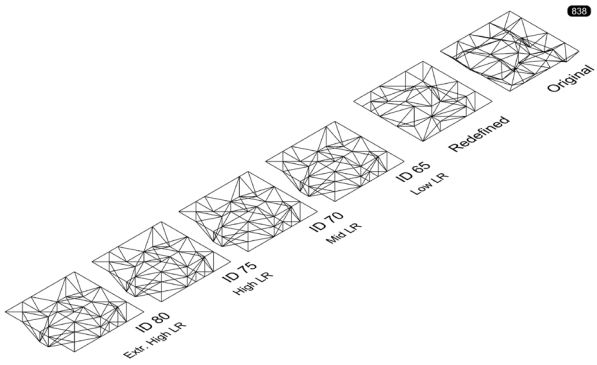


Figure 4.4.8. Mesh 838 GD output for various learning rates.
Own work.

Rounding matrix for surrogate model

In a final exploration, decoded matrices were rounded to binary data before functioning as input for the surrogate model. This, however, had insignificant impact on the graphs or output (documentation in Appendix VI).

4.4.2 HALF ADJACENCY MATRIX INPUT – DENSELY CONNECTED VAE – DOUBLE SURROGATE OPTIMIZER

With a double surrogate model (option 2), stock similarity can be assessed separately to material use and structural performance. Application of this was researched in exploration 23-27 (Appendix VI). Here, a learning rate of 0.5 was used. Results for mesh 838 are shown in figure 4.4.10. The GD algorithm worked well in this case also; predicted performance was again increased.

4.4.3 FULL ADJACENCY MATRIX INPUT – 2D CONVOLUTION VAE – SINGLE SURROGATE OPTIMIZER

As the created convolutional VAE models produced adequate reconstructions of the test meshes, optimization through gradient descent was also attempted on these models. These models, however, were found to be too computationally heavy for the scope of this thesis.

4.4.4 ACTUAL PERFORMANCE CALCULATION

The output mesh of gradient descent with half adjacency matrix input, with a densely connected VAE and single surrogate model was tested on material use, structural performance

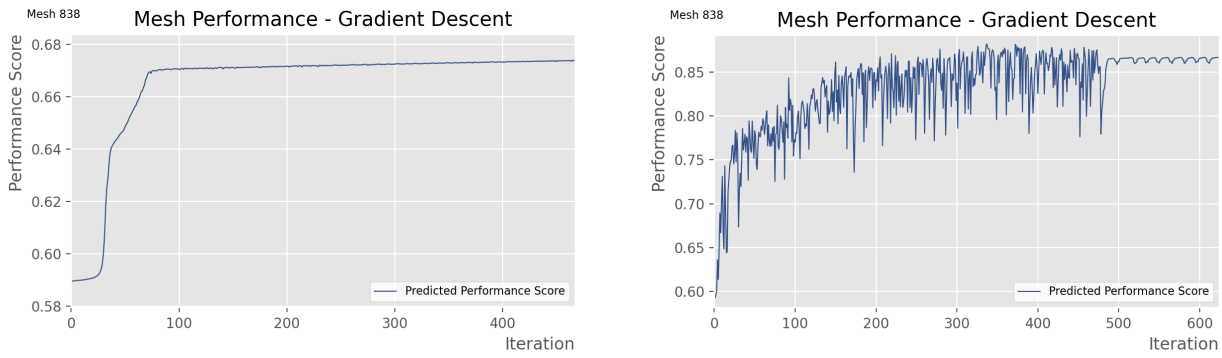


Figure 4.4.9. Mesh 838 GD output graph showing predicted performance for learning rate 2.5 and 30.
Own work.

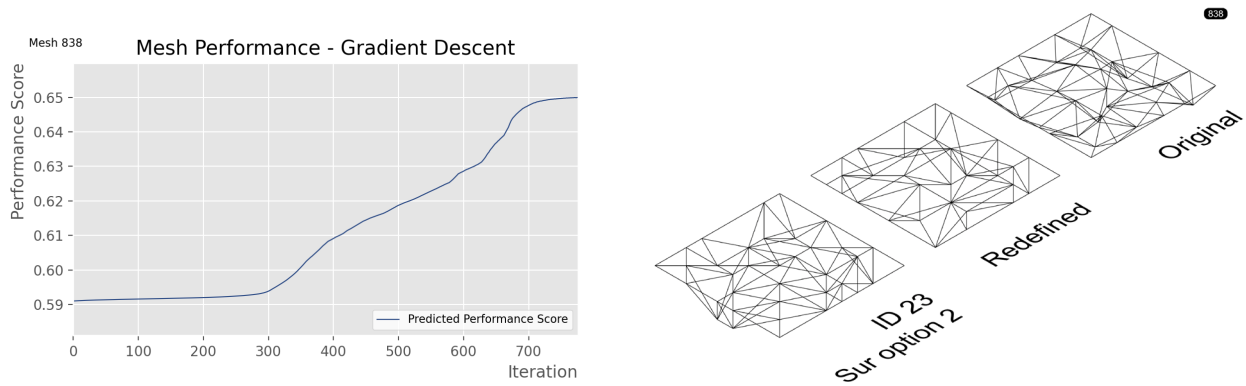


Figure 4.4.10. Mesh 838 GD output(right) and predicted Mesh Performance (left) with starting mesh 838, with separated surrogate models (model ID: 23)
Own work.

and stock similarity. This data was compared to the original scores of original redefined meshes. This is done, because these are the input meshes on which the optimization is based. This results in the following:

Mesh 838

The properties of the original mesh are:

Material use performance:	248.934781
Structural performance:	
Displacement:	4.753726
Utilization:	0.343162
Stock Similarity:	1.32502
Overall Score:	**0.3692**

The properties of the optimized mesh (figure 4.4.11) are:

Material use performance:	249.66248
Structural performance:	
Displacement:	6.219296
Utilization:	0.443601
Stock Similarity:	1.339972
Overall Score:	**0.15044**

For this mesh, no improvement was found.

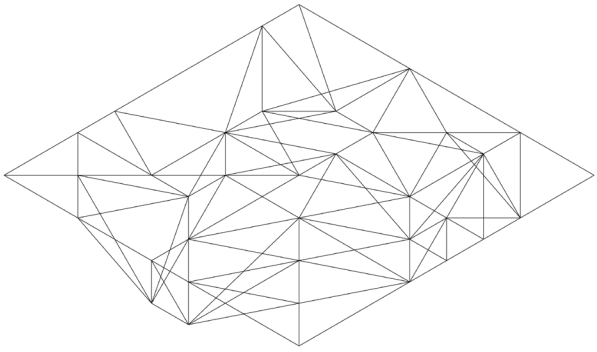


Figure 4.4.11. Optimized mesh for sample 838 (GD output)
Own work.

Mesh 2337

The properties of the original mesh are:

Material use performance:	251.267066
Structural performance:	
Displacement:	4.89553
Utilization:	0.413054
Stock Similarity:	1.34057
Overall Score:	**0.23212**

The properties of the optimized mesh (figure 4.4.12) are:

Material use performance:	269.59015
Structural performance:	
Displacement:	4.493735
Utilization:	0.38814
Stock Similarity:	1.384053
Overall Score:	**-0.03039**

For this mesh, no improvement was found.

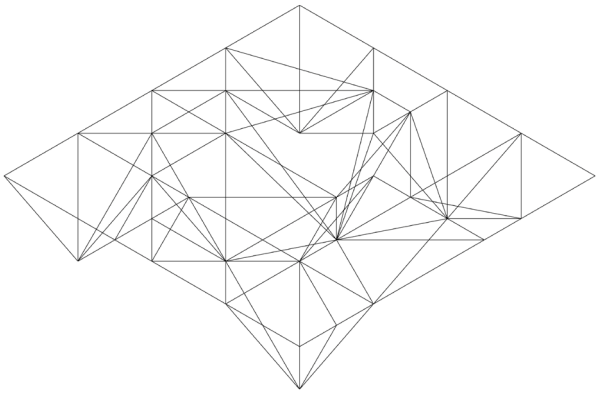


Figure 4.4.12. Optimized mesh for sample 2337 (GD output)
Own work.

Mesh 1665

The properties of the original mesh are:

Material use performance:	269.107138
Structural performance:	
Displacement:	1.075159
Utilization:	0.152099
Stock Similarity:	1.34057
Overall Score:	**0.51253**

The properties of the optimized mesh (figure 4.4.13) are:

Material use performance:	244.40203
Structural performance:	
Displacement:	2.269986
Utilization:	0.263927
Stock Similarity:	1.314961
Overall Score:	**0.6308**

For this mesh, some improvement was found in material use and stock similarity. Structural aspects, however, did not improve. Still, the overall score improved, meaning this mesh was succesfully optimized.

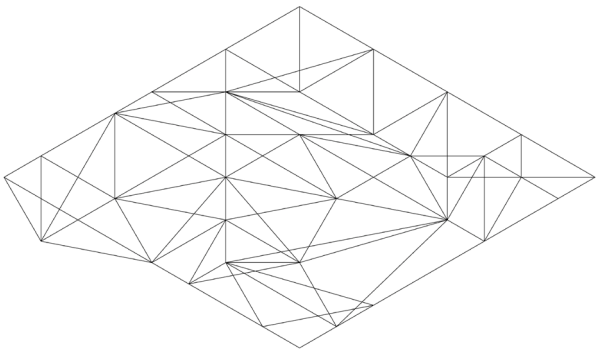


Figure 4.4.13. Optimzied mesh for sample 1665 (GD output)
Own work.

Mesh 1692

The properties of the original mesh are:

Material use performance:	258.579212
Structural performance:	
Displacement:	1.75926
Utilization:	0.203708
Stock Similarity:	1.37262
Overall Score:	**0.35819**

The properties of the optimized mesh (figure 4.4.14) are:

Material use performance:	250.13364
Structural performance:	
Displacement:	2.168047
Utilization:	0.266164
Stock Similarity:	1.337768
Overall Score:	**0.49954**

For this mesh improvement was found in material use and stock similarity again. Structural components slighly decreased here, but not significantly. The overall score was improved, compared to the originial mesh.

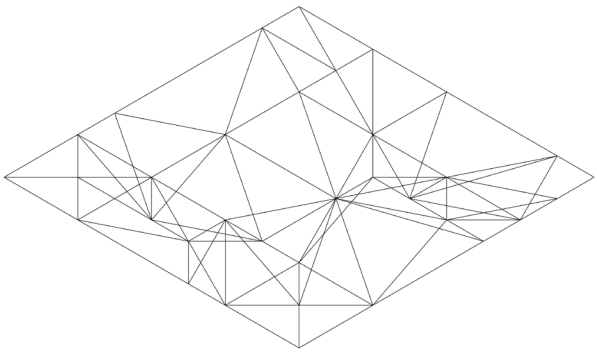


Figure 4.4.14. Optimzied mesh for sample 1692 (GD output)
Own work.

Mesh 2518
The properties of the original mesh are:

Material use performance:	260.771266
Structural performance:	
Displacement:	1.978422
Utilization:	0.21161
Stock Similarity:	1.35943
Overall Score:	**0.38714**

The properties of the optimized mesh (figure 4.4.15) are:

Material use performance:	255.67396
Structural performance:	
Displacement:	0.617542
Utilization:	0.102124
Stock Similarity:	1.334872
Overall Score:	**0.67445**

For this mesh, a larger overall improvement was found. In addition, all individual aspects are also improved in this case.

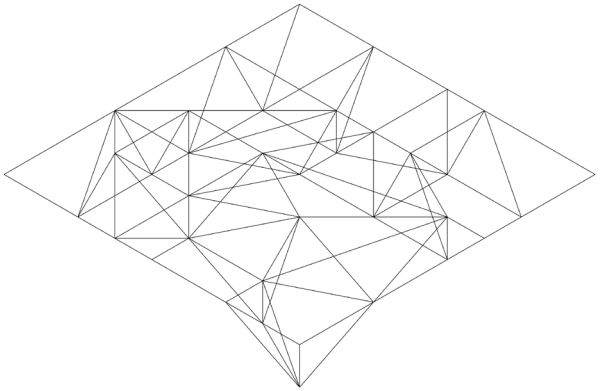


Figure 4.4.15. Optimzied mesh for sample 2518 (GD output)
Own work.

EDITING PRODUCED MESHES

Meshes generated by the framework sometimes followed very different topologies than the meshes from the original dataset. While this is not necessarily an issue, two of the produced meshes (with input mesh 2337 and high-performing input mesh 1665) were slightly edited to fit the originally selected topology. This was done to assess whether a model with a different set-up would be worth researching.

In this case, some vertices that are typically placed on the lower layer of the structure were generated on the top layer. This resulted in a dense top layer for these mehses. Bottom layers, however, contain less edges than the original meshes. This caused major decrease of structrual performance.

Results for edited mesh with input 2337
The properties of the original mesh are:

Material use performance:	251.267066
Structural performance:	
Displacement:	4.89553
Utilization:	0.413054
Stock Similarity:	1.34057
Overall Score:	**0.23212**

The properties of the initial optimized mesh (figure 4.4.12) are:

Material use performance:	269.59015
Structural performance:	
Displacement:	4.493735
Utilization:	0.38814
Stock Similarity:	1.384053
Overall Score:	**-0.03039**

The properties of the optimized & edited mesh (figure 4.4.16) are:

Material use performance:	270.87662
Structural performance:	
Displacement:	0.374842
Utilization:	0.092052
Stock Similarity:	1.378316
Overall Score:	**0.41164**

For this mesh, significant improvement was found, especially compared to the initial optimization. Compared to the original mesh, material use and stock similarity performance scores have worsened, but they have slightly improved compared to the inital optimized

mesh. Structural performance indicators have improved compared to both meshes.

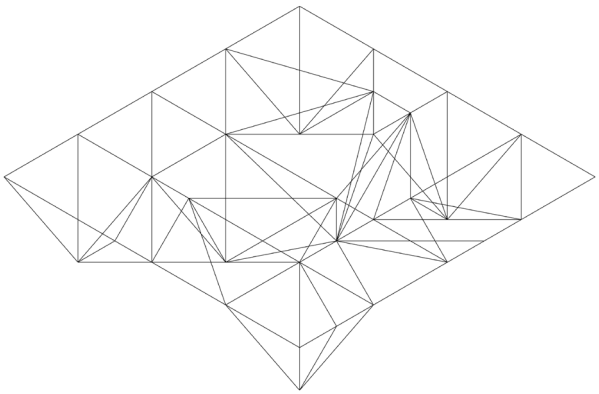


Figure 4.4.16. Optimzied & edited mesh for sample 2337 (GD output)
Own work.

Results for edited mesh with input 1665

The properties of the original mesh are:

Material use performance:	269.107138
Structural performance:	
Displacement:	1.075159
Utilization:	0.152099
Stock Similarity:	1.34057
Overall Score:	**0.51253**

The properties of the initial optimized mesh (figure 4.4.13) are:

Material use performance:	244.40203
Structural performance:	
Displacement:	2.269986
Utilization:	0.263927
Stock Similarity:	1.314961
Overall Score:	**0.6308**

The properties of the optimized & edited mesh (figure 4.4.17) are:

Material use performance:	255.7416
Structural performance:	
Displacement:	0.629073
Utilization:	0.101264
Stock Similarity:	1.325075
Overall Score:	**0.71786**

For this mesh, significant improvement on structural performance was found compared to both the original mesh and the initial optimization. Stock similarity and material use, however decreased in performance compared to the initial optimization. They have, however, still improved compared to the original mesh. Overall performance therefore increased.

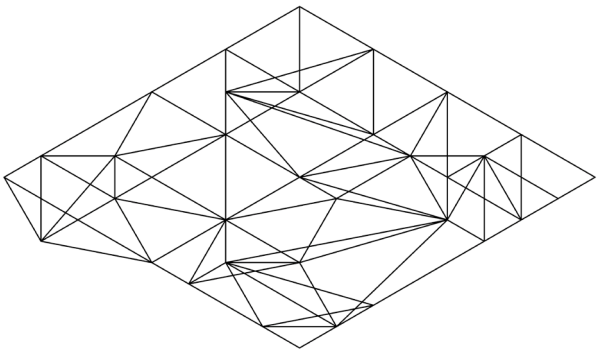


Figure 4.4.17. Optimzied & edited mesh for sample 1665 (GD output)
Own work.

In conclusion, the workflow was able to produce improved mesh structures in the tested samples. Slight editing of the meshes proved to be succesfull in further optimization of the meshes.

RE-RUN WITH UPDATED DATASET

As there were clear issues found within the previously presented results, an assessment of the initial dataset was done. It was found that in the original process of converting the coordinate data to adjacancy matrix, the top layer of the geometry was converted to contain a mix of quadrilateral and triangular cells in the top layer. While these still present as valid structures that could be applied, it can be assumed that the original calculated performance score is no longer representative of actual performance. Therefore, the conversion method was altered to ensure that the quadrilateral grid of the top layer was maintained.

The following results show the performance of this newly trained model. For a learning rate of 0,5 it was found that the majority of meshes did not change from the original input. Therefore,

higher learning rated were used (up to 100). However, meshes still did not show significant change from the original input. For this reason and the limited scope of this work, this method was discontinued.

4.5 GENERATIVE ADVERSARIAL NETWORK

A second type of generative deep learning architecture that was explored was through a generative adversarial network (GAN). As explained in the literature review, a GAN consist of a generator and discriminator.

For the purposes of this study, GANs can be used in two ways. Firstly, the surrogate model can be combined with a GAN to form the full workflow. Here, the GAN needs the mesh geometry (half adjacency matrix data) as input. Data from the training dataset is labeled 'True' (with a value of '1'). A second set of fake data is created, labelled 'False or '0'. As the generator and discriminator compete, both models are trained. The trained generator can then be used to generate new data. This new data can be send to the surrogate model to assess when better performing meshes can be found.

In a second method that can be used to optimize outputs, the performance indicators are also used as input in the GAN model. This way, the GAN can directly predict the performance indicator. The generation space can then be explored and generated meshes with performance over a set value can be saved.

GAN (unlabelled model)

A generator was sucessfully trained, as is shown in figure 4.5.1. However, the losses for the discriminator quickly reached infinite values. Various attempts were made to limit this (simplifying the architecture, altering learning rates, using various optimizers, adding clipnorms). However, no well-training model was created.

Figure 4.5.1. Generator loss for a trained GAN model. Discriminator loss exploded to infinite values.
Own work.

A second architecture that was tested was Wasserstein GANs with Gradient Penalty (WGAN-GP). The Wasserstein GAN was first introduced in 2017 (Arjovsky et al., 2017). This method proposes the use of a different loss function based on Wasserstein-1 distance, which has been shown to improve training stability. The results of such a model trained on the unlabelled geometry data are shown in figure 4.5.2.

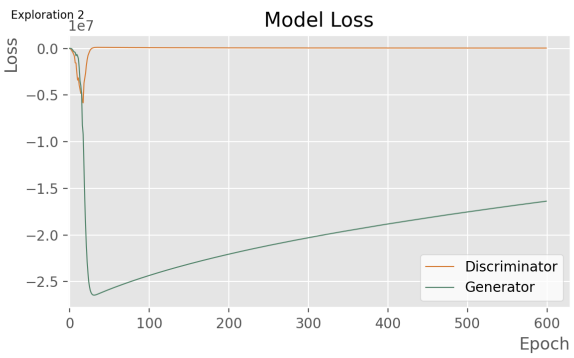


Figure 4.5.2. Generator and discriminator loss for a trained Wasserstein GAN model.
Own work.

Although this model is promising, further development is required to properly asses its applicability and performance. This was, however, beyond the scope of this thesis.

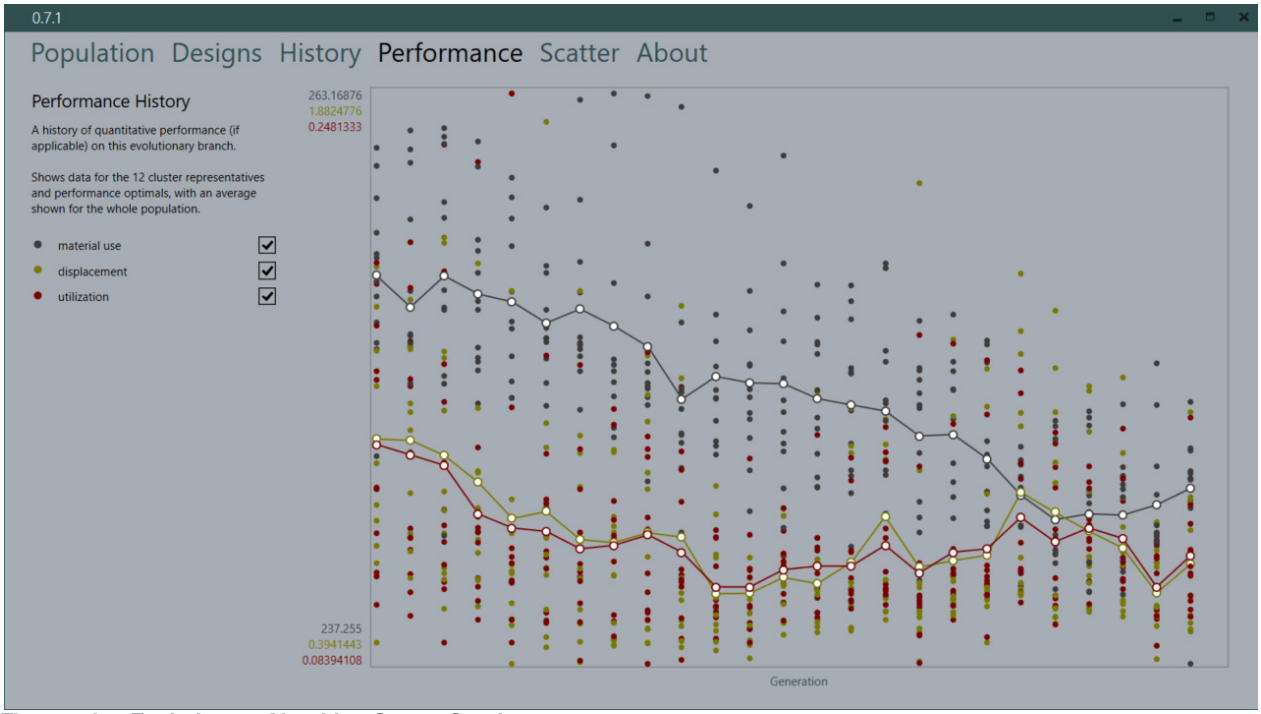


Figure 4.6.1. Evolutionary Algorithm Output Graph
Own work.

ID	Material use	Displacement	Utilization	Similarity to stock	Overall
0	237.255003	0.814472	0.124237	1.3248	0.80112
1	243.160903	0.517798	0.099262	1.33418	0.75771
2	242.865796	0.549619	0.107809	1.33062	0.76797
3	242.890614	0.560134	0.108309	1.32393	0.79688
4	240.979901	0.541356	0.100983	1.33146	0.78013
5	240.632539	0.517727	0.103293	1.3299	0.78892
6	241.809288	0.540046	0.095185	1.32247	0.81946
7	240.804746	0.575395	0.109791	1.32862	0.78617
8	240.797853	0.58395	0.110846	1.32623	0.79575
9	247.376331	0.449664	0.091407	1.33532	0.73706

Figure 4.6.2. Evolutionary Algorithm Performance Results for 10 generated meshes.
Own work.

4.6 EVOLUTIONARY ALGORITHM

Finally, a fully different approach to multi-objective optimization was explored. This was done through the BioMorpher plug-in in Rhinoceros Grasshopper. This was done mainly for comparison to the VAE- and GAN-based workflows developed in this thesis.

The designed algorithm first focuses to optimize for material use and structural performance (displacement and utilization). From this, a set of well-performing geometries is saved. Next, this list of geometries is send to a separate assessment for similarity of elements to the stock library with a GhPython script. This was done as the similarity assessment took significantly longer to calculate in the Grasshopper environment compared to the other parameters. As the evolutionary algorithm requires a generation size of twelve or higher and several generations need to be run, this choice significantly reduces computing time. When, for each selected sample, the similarity score was calculated, the best performing meshes could be found.

The model starts with 'Generation 0'. This generation consists of 24 sets of randomly generated values. Next, generation of 24 meshes each are generated. The full history of the model van be found in the Appendix. As well-performing geometries start to be formed, they are selected in the menu. The characteristics (genes) of these 'parent'-geometries is used to create the next generation of designs. After running 24 generation of this algorithm, the performance was plotted (figure 4.6.1). Over generations, all three performance indicators start to decrease. This means that the algorithm successfully optimizes the geometry. Of all meshes produced by this algorithm, ten of the highest performing meshes from generation 17 to 24 were assessed on stock similarity (figure 4.6.2).

The results of the overall best performing mesh found through this method (figure 4.6.3) are:

Material use performance:	241.809288
Structural performance:	
Displacement:	0.540046
Utilization:	0.095185
Stock Similarity:	1.32247
Overall Score:	**0.81946**

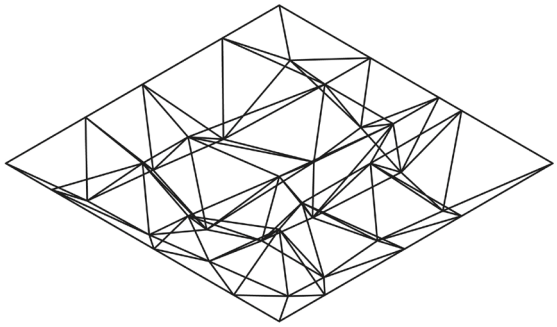


Figure 4.6.3. Optimized mesh created by an evolutionary algorithm.
Own work.

This is higher than any of the meshes in the dataset used for training the AI frameworks. (In the dataset, the highest found score is 0.79934). This indicates that an evolutionary algorithm performs better optimization than random generation.Compared to the AI based framework, this method has a much larger design freedom; all possible coordinates can be used. When the model is converted to those that the VAE can produce, the performance changes to:

Material use performance:	257.017232
Structural performance:	
Displacement:	0.624517
Utilization:	0.104095
Stock Similarity:	1.33354
Overall Score:	**0.67092**

It also should be noted, that most output of this method look similar. This is, in part, the goal of EAs, however. Still, it also causes the generation space to be limited early on in the process. This effect can however be decreased if the process is restarted a number of times. As it is based on a randomly generated fist selection, this will likely lead to differing outputs.

5. Application

5 Application

5.1 AI FRAMEWORK APPLICATION

In this section, the applications of the designed framework and generative AI in general are discussed. This includes the integration of such workflows within the architectural and structural design process.

The aim of the designed framework is to generate an optimized design based on material use, structural performance and similarity to a stock library. A structure generated by the framework is shown in the render in figure 5.1.1.
The framework is intended for use in early stages of the design process. The tool can also be use to assess feasibility of the use of a set library for a design task by checking the number of valid meshes produced.

COMPUTATIONAL REQUIREMENTS

The full, final framework runs on the following properties on the DelftBlue cluster:

```
#SBATCH      --time=05:00:00
#SBATCH      --ntasks=1
#SBATCH      --cpus-per-task=1
#SBATCH      --partition=compute
#SBATCH      --mem-per-cpu=32GB
```

It should be noted that the five hour time is an upper limit. In practice, the model ran for 2-4 hours depending on the set hyperparameters.

CURRENT APPLICATION

The designed framework can be used in practice in its current form, as long as its limits are considered in the design process. The exact framework created in this work can be used only to create a mesh of the selected type and with the selected stock library. A new or modified mesh type or stock library requires the creation of a new dataset and new training of the neural networks. This requires specialist knowledge that is likely not yet available to actors in the building design sector. Additionally it could be a time-consuming process in itself. Further development of the method would help mitigate these problems.

The case study presented in this thesis can primarily be used as a demonstrator of the use of generative deep learning models in architecture generally. This is a necessary step in

Figure 5.1.1. A rendered view of mesh 2337 as used in a structure.
Own work. Parts of the background were generated using the Adobe Photoshops generative AI feature.

the current research, as these models are not yet widely used in architectural and structural design. In addition, this research required a case study that is relatively simple, as the validity and applications of such models was tested. However, the model's task was also designed to not be one that can already be effectively and accurately performed by existing (computational) tools. This method therefore demonstrates both the potential uses of generative deep learning models in these fields and provides innovative, effective solutions to a design problem that is currently challenging to tackle.

RECOMMENDED ALTERATIONS FOR BETTER WORKFLOW INTEGRATION

AI generative models as the one discussed in this thesis require datasets to work. The availability of such data is currently limited. The data required consists of:

- A geometry dataset, consisting of (half) adjacency matrix data.
- A performance indicator dataset (labelling)
- A material stock library dataset with material properties to be considered.

Geometry datasets can be generated through for example Python or Rhinoceros Grasshopper. These sets need to be generated by the user and are specific to the design assignment. The performance indicator dataset should be retrieved by (simulated) testing of the geometries. For stock similarity assessment, The material stock should be based on locally available stock. This data can be more challenging to retrieve, as it requires detailed documentation of available stocks.

As the creation of the workflow for a new design task can be time-consuming, this work should ideally be started in early stages of the design process. So, these datasets are also required early on. This is were an information gap is currently present; properties of reusable materials are unknown or unavailable at the required times. Therefore, a standardized procedure for documentation and distribution of this data would be beneficial.

To be able to access the required data on available stock, an 'existing buildings as material banks' (E-BAMB) approach can be explored (Rose & Stegermann, 2018).

BAMB Approach & Managing of stocks

The BAMB project was started in 2015. It is part of the EU-funded Horzion 2020 program (European Energy Innovation, 2017). The project describes the steps required to shift to a more circular building industry (BAMB2020, n.d.). One of these steps is to gather material data in 'Material Passports'. This material data is organised in several categories, including physical, chemical and biological material properties (Heinrich & Lang, 2018). Physical data includes dimensions, weight, density and other properties relating to building physics. Chemical attributes include chemical composition, lifespan information and durability. Biological data describes decomposability, renewability and recycle and reuse potentials. Information on material health is also documented.

To enable reuse, information on the location of the project to be demolished and/or material storage location(s) is also useful. This is for example done in the Oogstkaart project, which aims to map urban mining potential in the Netherlands. (Oogstkaart, n.d.). This project can be seen in figure 5.1.2. Similar data is stored in the 'Transportation and Logistics' category in the BAMB material passports.

BAMB information & tools can be integrated in Building Information Modeling (BIM). This way, material information and reuse potential is defined at the design stages of a building's life-cycle. This allows the data required for generative AI workflows to be accessable at the

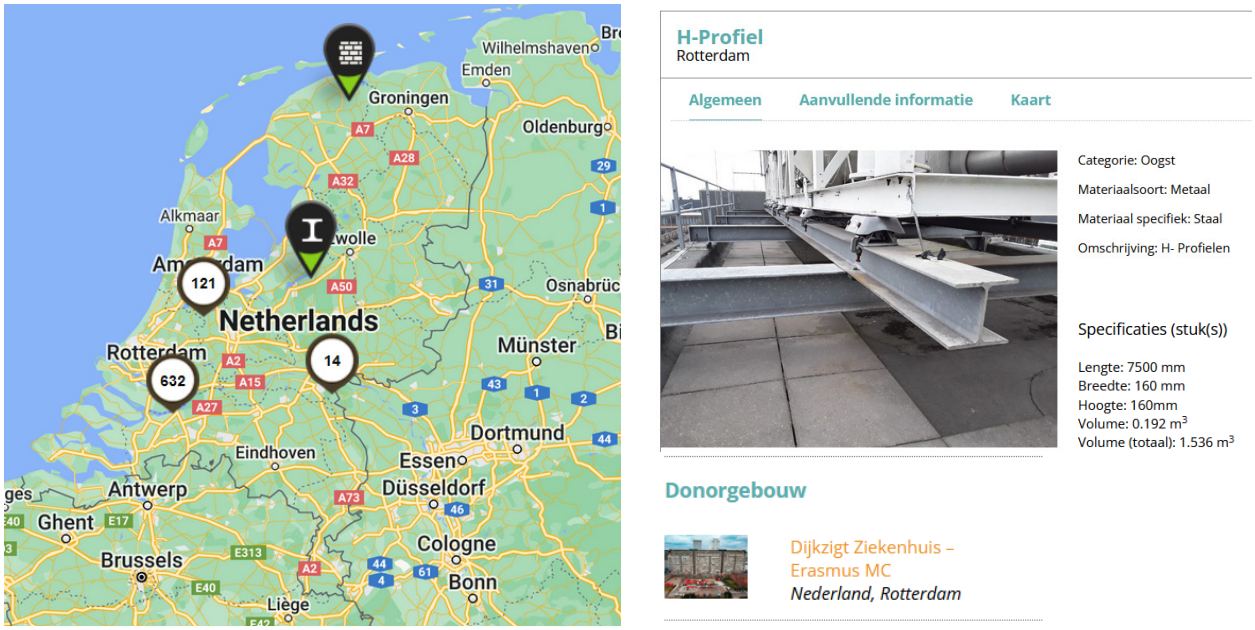


Figure 5.1.2. The Oogstkaart Project with a map (left) and example of available reusable structural H-profiles (right).
Retrieved from: Oogstkaart. (n.d.). Oogstkaart: De urban mining potentie van NL. https://www.oogstkaart.nl/.

required time (in the early design stages).

BAMB data can be gathered in various ways. Rose & Stegermann (2018) describe the following:

- Original building design analysis.
- Built assets records (public).
- In-use building component estimation.
- In-use building component measurement.
- Identification of components after demolition.

As stated previously, it is preferable for documentation to be accurate and available before demolition (preferably earlier). For existing buildings, estimation or measurement of building components would therefore be most preferable. For future projects, integration of BAMB and BIM in the design stages would be recommended. In either case, however, evaluation of building components after use should still take place to determine reuse potential. To best serve the proposed AI framework in this thesis, quick identification of building materials is essential. Therefore, documentation on building components before demolition is preferable. This way, the AI framework can be applied on material stock data early in the design process. In addition, when a reuse-based design is created before demolition, disassembly techniques can be adjusted to ensure suitable component retrieval.

Law & Regulations

Various governments worldwide have set goals regarding sustainability and circularity. In the Netherlands, the goal is to have a fully circular economy before 2050 (Rijksoverheid, n.d.). While a timeline and documentation on recommended measures to be take before 2030 have been published, limited laws and regulations have been put in place. Still, an agreement regarding retrieval of raw materials has been signed by over 400 parties (Rijksoverheid, 2021) These agreements, however, do not impose regulations to all relevant sector participants. This is why regulations should be updated, particularly in local municipality level governments, as it is most sustainable and feasible to source materials locally. Still, international law and

regulation should also be considered, as many material and waste streams are on a continental or worldwide scale.

As regulations on circularity are not yet well-integrated in law, incentive for users to apply circular principles in their building projects is lacking. While sustainable goals are often encouraged by governments and publicly valued more in recent years, improving governmental regulation on circularity in the built environment is an essential step towards circularity.

Societal Aspects

Integration of reusable elements in new buildings and structures can contribute to making the building sector more sustainable. Besides benefits to sustainable goals as mentioned in previous sections, it also impacts societal aspects of buildings. Buildings serve people and therefore hold value in society. The functionality and use a building directly impacts the way it is viewed A highly valued scientific or cultural institution. a historic religious building or even a beloved family home; buildings hold societal and personal importance. As these buildings reach the end of their lifetime, they could be restored. However, as the built environment and society keep changing, this may not always be the preferred choice. Demolition, however, would remove all societal value. Alternatively, building elements can be collected and saved to be used in new projects that better serve the new (functional) requirements of society, while still keeping parts of retired buildings in place. To building owners and users, this aspect could make reuse an advantageous option in a societal sense, in addition to the sustainable benefits of this approach.

Additionally, applying reuse in building design can lead to creative and unconventional design choices, resulting in unique features of a building (figure 5.1.3). This is, for example, the case in this thesis. When new items would be used for the design of a spatial truss, it is most likely that a regular grid would be chosen, as is the case in the Model and Orange Halls of the TU Delft Faculty of Architecture, where the case study truss is present.

Figure 5.1.3. Casa Collage by S+PS Architects; a project where building components are reused to create a unique facade.
ArchDaily. (2021). Deconstruct, Do Not Demolish: The Practice of Reuse of Materials in Architecture. https://www.archdaily.com/974056/deconstruct-do-not-demolish-the-practice-of-re-use-of-materials-in-architecture.

5.2 COMPARISON TO OTHER OPTIMIZATION TOOLS

As stated previously, the aim of the designed framework is multi-objective optimization of a 3D truss structure for material use, structural performance and stock similarity. However, tools for (part of) such optimization exist. In this section, the differences between some optimization methods are discussed.

MULTI-OBJECTIVE OPTIMIZATION

The proposed framework can be classified as a form of multi-objective optimization. This means that the output score indicates performance for multiple parameters of the design. The output geometry therefore reflects the best configuration of elements based on these parameters. Results of a 2017 study on optimization in architectural projects concluded that many architects expressed a need for such tools. 78% of respondents in

this study rated multi-target optimization as preferable over single-target tools. Additionally, 91% declared a need for choice regarding the optimized outputs (Cichoka et al., 2017). The gradient descent algorithm in the framework of this thesis offers this. It should be noted, however, that many multi-optimization tools, including the workflow presented in this thesis, have limits. While multiple objectives are taken into account, it may not include all relevant objectives. Therefore, the human designer should still evaluate the outcome of optimization tools. Additionally, in this work, the balancing (weighing) of all parameters is set to a chosen ratio. In reality, this ratio may vary depending on the design task. Altering parameters (such as the input mesh in the presented worflow) may be applied to find a series optimized outcomes, which can then be reviewed through other methods and on other relevant aspects. Extending multi-optimization tools to include all relevant performances is another possibility. However, this likely adds complexity to the optimization process and should therefore be carefully assessed.

COMPUTATIONAL COMPLEXITY AND COST

Many (existing) simulations/optimizatio ntools are computationally complex and high-cost. This was, for example, found in the evolutionary algorithm in BioMorpher in this thesis. This is especially true for multi-objective optimization tools. As a result, they can be time-consuming as well. In addition, a tool as designed in this work is time-consuming to produce in itself. However, once the designed architecture works to an acceptable degree, the optimization is extensive (multi-objective) and relatively quick. The five hour timeframe mentioned previously includes the surrogate model, VAE and GD, run on DelfBlue (single node, 32 GB RAM). However, when the surrogate model and VAE are saved and called in the GD algorithm, this time can decrease to under an hour, which is comparable to the time it would take to optimize for structural performance, material use and similarity to the stock individually. However, no multi-objective tool that combines these factors was found. Therefore, a direct comparison could not be made.

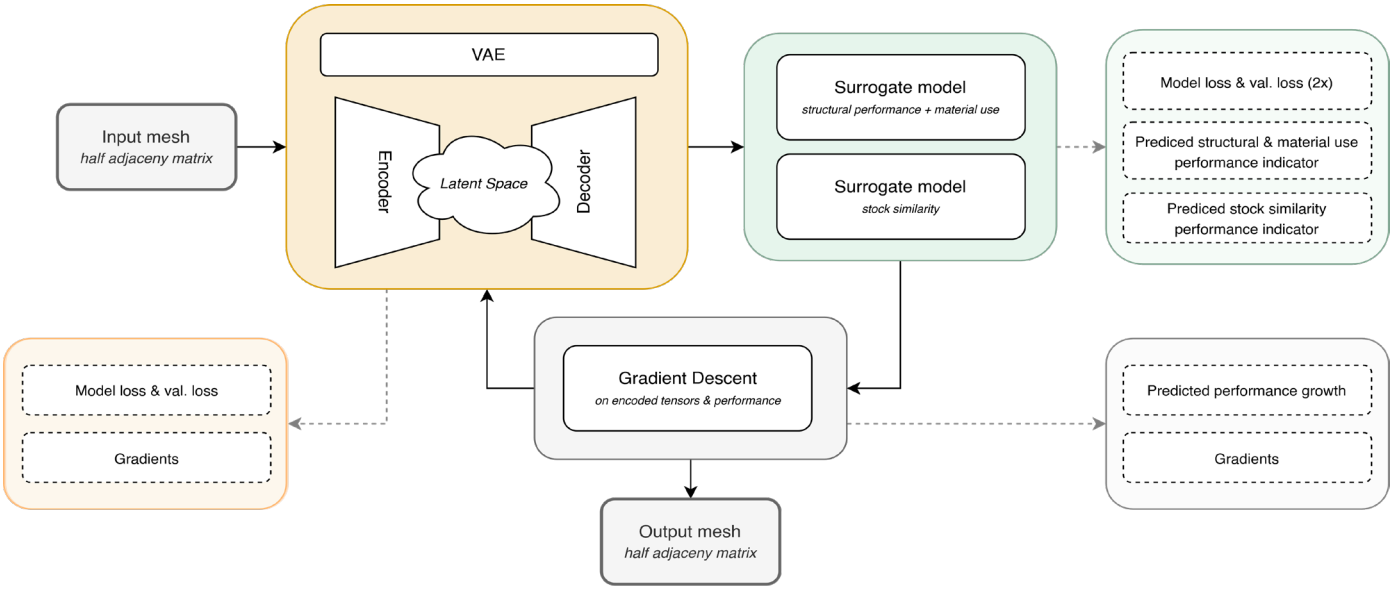


Figure 5.2.1. Visualisation of user insight into the Generative Framework. The simplified framework process is indicated by solid arrows and the user insights are indicated by dashed arrows.
Own work.

BLACK BOX SIMULATION

Many existing optimization tools, such as Kangeroo, Galapagos and BioMorpher in Rhinoceros Grasshopper, are restrictive due to their 'black box'. The user can input parameters and the plug-in or software produces an output. However, the underlying principles and calculations may not be clear.

This can also be said of the framework designed in this work. It is, for example, difficult to visualise the organised latent space of the VAE. Still, the GD optimizer proves that this space is well-structured, since valid output is produced. The diagram in figure 5.2.1 visualises the insight the user has into the designed framework. This was done for the framework that uses a separated surrogate model.

6. Conclusions

6 Conclusions

6.1 CONCLUSIONS

In this thesis, the following research question was addressed:

Can an artificial intelligence (AI) based generative design framework generate new spatial (3D) truss design solutions, with optimized structural performance, minimized material use and that consist of linear elements that closely match elements from a reusable material stock, in reference to the training dataset, and therefore be used as an effective tool for design exploration in early design stages of the materially circular architectural design process?

Furthermore, the following sub questions were asked:

1. What types of AI can be used in generative models?
2. What form of data, describing spatial truss-like 3D geometry with variable-length linear members, could be used to train the AI generative framework to generate new design solutions with similar properties to this training dataset?
3. How can a spatial truss-like 3D geometry be computationally evaluated on its structural performance, so that the outcome data can be used to train an AI generative framework?
4. How can a spatial truss-like 3D geometry be computationally evaluated on its material use, so that the outcome data can be used to train an AI generative framework?
5. How can a restrictive, yet suitable stock library be defined for materially circular design of a spatial truss?
6. How can a spatial truss-like 3D geometry be computationally evaluated on its similarity to a stock library, so that the outcome data can be used to train an AI generative framework?
7. How can an AI generative framework be designed to reconstruct input and generate new spatial truss-like 3D geometry with optimized structural performance, minimal material use and high similarity to the material stock?
8. Can a surrogate model effectively predict the performance score of datasets describing a set of spatial truss-like 3D geometry, in its encoded and decoded forms?
9. How can optimized solutions be found within the data generated by the AI generative framework?
10. How does the proposed AI generative framework compare to conventional shape and topology optimization tools?

GENERATIVE AI FRAMEWORK ARCHITECTURE & INPUT

The literature review (Chapter 2) suggests two main model types that may be utilised for generative design: Generative Adversarial Networks (GANs) and Variational Autoencoders (VAE). In this study, two sub-types of variational autoencoders where explored: (1) a densely-connected variational autoencoder and a convolutional variational autoencoder. (sub question 1)

Within these models, various data types and configurations can be used to describe spatial truss-like 3D geometry. This data can consist of binary or float (decimal format) values. Float-type data can be used to describe vertex (x, y, z) coordinates or the (x ,y, z) vector movement of a vertex compared to its base position. These datatypes contain no information on connecting edges between the vertices, which makes in unsuitable in some cases. The case study chosen for this thesis, however, always contained the

same overall topology; vertices were able to move with respect to their base position, but always connected to the same set of vertices.
Binary data can be used in the form of adjacency matrices, edge-vertex matrices or 'stepped movement' matrices. In adjacency matrices, vertices are listed in both the rows and columns. When two vertices are connected by an edge, a value of 1 is entered in the matrix. Otherwise, the value 0 is given. As these matrices are symmetrical over the diagonal axis, it is also possible to use a list of half of the values from the original adjacency matrix. In edge-vertex matrices, vertices are represented in the columns only. The rows contain edge indices. A value of '1' is inserted when a vertex is either the start of end of an edge. This means that each edge, and therefore each row, contains two values of '1'. This data representation can be larger or smaller than the adjacency matrix representation, depending on the number of vertices and edges in the geometry. In the case presented in this thesis, the tested geometry contained 40 vertices and 128 edges. As a result, the edge-vertex matrices for this geometry were larger than the adjacency matrices. The data type labelled 'step movement matrices' are a binary representation of the (x, y, z) vector data. The float values are divided into steps, which form the columns of the matrix. The rows contain the x. y and z aspects of all vertices. As a result, each row contains one value of '1'.

In the surrogate model and generative (VAE) model architectures, the various datatypes mentioned previously were used as input. Overall, data based on the position of the vertices (coordinate data, movement data) resulted in unrepresentative outputs. This was the case regardless of data type (float or binary). Data representations based on topology/edge connectivity (adjacency matrix data, half adjacency matrix data, edge-vertex matrix date) resulted in more accurate performance predictions and geometry reconstructions. Out of these, (half) adjacency matrix data performed best. (sub question 2).

PERFORMANCE INDICATOR CALCULATION & SIMULATION

Finite Element Analysis can be used to assess structural performance of geometries. This can be computationally heavy and time consuming, especially for large datasets. Calculation of displacement and utilization by Karamba3D results in a representative indicator for structural performance of a mesh. (sub question 3). As the assumption was made that all steel elements used in the structure have the same profile and material properties, material use can be assessed by calculating the total of element lengths. (sub question 4).

To assess the extent to which the elements from a generated geometry match those from a stock library, a custom algorithm was written. In reality, a suitable stock can be implemented into this directly. For this thesis, a self-defined material stock library was created. The required stock library was defined so that the generated geometries in the dataset would present a range of performances. Secondly, this stock library was defined to be realistic; it was based on a set of beam trusses. The number of these available trusses was limited, so that the total number of elements in the library was slightly larger than the number of elements required for building the generated dataset geometries. This was based on the total length of the library elements compared to the total length of elements in the generated geometries, so that the library would always contain enough material for use in the generated structures. The average length of elements in the stock was also balanced to the average length of geometry elements. In this process, proportions of the beam-trusses used to create the library were kept realistic by limiting the length to height ratio of the designed beam trusses. (sub question 5).

Using this library, similarity assessment is performed. In this thesis, similarity is defined by difference between the length of geometry elements and the corresponding library element lengths. A margin is left to allow for splitting of some library items. This way, unproportional negative impacts from single outliers on the performance score of otherwise well-performing meshes are avoided.
Normalization of data of all three performance score types makes it possible to combine the performance scores into one. Weights are implemented in this calculation as well. (sub question 6).

MESH GENERATION (VARIATIONAL AUTOENCODER)

As described in the methodology, the generative framework consist of a labelled dataset, variational autoencoder (VAE), surrogate model and an optimizer (gradient descent). The variational autoencoder is the generative model in this workflow. The performance of the variational autoencoder architectures evaluated for this thesis varied substantially. Training models on data describing the vertices' locations ensured that all edges would be reconstructed. However, the model was not able to reconstruct input meshes accurately; all sampled reconstructions matched the base mesh exactly or fairly closely.

With full adjacency matrix data, the densely connected VAE often failed to reconstruct all edges of the input meshes. It also frequently generated edges that were not part of the original input mesh. Similar results were found for edge-vertex input. In this case, the generated edges were often significantly longer than those in the original meshes. Therefore, neither of this inputs was considered to produce acceptable results. The densely connected VAE trained on half adjacency matrix, however, was able to reconstruct a significant part of the original dataset correctly. However it still produced poor reconstructions for a many of the training meshes.

Convolutional VAE architectures are trained on full adjacency matrix data. This architecture was able to reproduce input meshes with adequate accuracy for a similar part of meshes as the densely-connected VAE trained on (half) adjacency matrix. However, the overall reconstruction quality was slightly lower (with similar latent dimensions); a larger latent dimension was required for similar accuracy. For this reason, the densely-connected VAE was chosen for the final workflow. (sub question 7).

PREDICTIVE MESH PERFORMANCE EVALUATION (SURROGATE MODEL)

The goal of the surrogate model within this thesis is to accurately predict the performance score of generated geometries, so that the optimal solutions can be found after. Half adjacency matrix data was found to produce the best predictions. With the best performing architecture, parameters and input data type, the surrogate model predicts overall performance indicators with 98.2% accuracy (percentual error = 0.0018). When results from models trained to predict separate aspects of the performance score (material use score, structural performance score and stock similarity performance score) were used to (re) calculate the overall performance score, predictions were made with similar accuracy. (sub question 8).

MESH OPTIMIZATION (GRADIENT DESCENT)

A gradient descent algorithm was used for optimization. Here, the best performing VAE an surrogate model architectures were used. As stated previously, the VAE did not produce accurate results for all input meshes. Still, the gradient descent optimizer worked successfully on the inputs that were reconstructed with high accuracy. The GD model was run with various hyperparameters and in combination with various surrogate models and VAEs. In the tested cases, the GD was always able to produce

novel meshes with a higher performance score (as predicted by the surrogate model). The meshes produces in this optimization did in part of the cases perform better. In one case, the optimized mesh performed better on all optimization goals (material use, structural and stock similairy) when assessed using the original simulation. However, in most cases the model improved only part of the researched aspects. Still, this often resulted in an improved mesh. In one tested case, the resulting optimization should be viewed as invalid, as calculated performance decreased to a point that was significantly below any other generated model (including the original dataset). Slight editing of this model, however, resulted in an increase in performance compared to the original input. Slight editing of the mesh also proved to be beneficial to an already improved mesh.

None of the tested meshes were found to be improved compared to the inital training set. However, it should be noted that a much larger degree of design freedom was used in this inital training dataset. Therefore, this dataset is not perfectly suitable for comparison. The largest performance indicator for an optimized mesh is 0.67445. This mesh also demonstrated the greatest improvement, starting at 0.38714. This means a 74.2% increase was reached. For an optimized and edited mesh, this is 0.71786 (starting point: 0.51253, percentual increase: 40.1%).

Therefore, it can be concluded that the AI workflow is successful in optimizing the input meshes. Further research can however be performed to further increase the workflows success rate and performance (sub question 9).

In the optimizing step (GD), an input can be selected by the human designer. The output of the optimizer is likely to still contain properties present in this original mesh. As a result, a human designer can design with their own chosen goals, even those that are not optimized by the framework, such as aesthetic quality. After application of the framework, the mesh is likely to have (largely) maintained these qualities. This is similar to many optimization tools, especially those present in parametric modelling. A notable aspect of this framework is its ability to integrate structural performance, material use and stock similarity in a multi-targeted optimization. Many currently available tools provide single objective optimizations. If the framework is applied with split surrogate models, the framework gains more usability for targeted optimization. In this case, the user is able to apply weights to more accurately direct which of the integrated aspects to optimize for.

An evolutionary algorithm based multi-objective optimization tool, designed using the BioMorpher plug-in in Rhinoceros Grasshopper, however, did prove to find higher performing messages, compared to both the training data set and the VAE-based workflow. The found mesh had a performance score of 0.81946. Again, it should be noted that this is likely in part due to the larger degree of design freedom available to the EA, as when the design freedom is limited through the same process, the found mesh has a performance score of 0.67092 which is slightly under the best (unedited) optimized mesh by the VAE workflow.

In addition, the generative framework in its current form requires (basic) knowledge of the applied deep learning models. Its use is therefore not as accessible as various other optimization tools. Additionally, it can be computationally heavy; the full framework can take 2-4 hours to run on 32GB RAM. It was found that other optimization methods, such as an evolutionary algorithm, produce improved results, compared to random mesh generation (sub question 10).

6.2 DISCUSSION

In this section, the approached used in dataset generation, and the design of the surrogate model, variational autoencoder and gradient descent optimization are discussed and reflected on.

DATASET GENERATION & LABELLING

The dataset initially consisted of coordinate data and one adjacency matrix, which together described 9.703 3D mesh structures. For simplification purposes, this geometry was limited to a bounding box. Vertex movement was also limited. In future work, these limitations could be decreased.

This dataset was later converted to various datatypes, including adjacency matrices and edge-vertex matrices. To create these datatypes, redefinition of the meshes was required. In this step, meshes were re-created with >94% accuracy. For future work, a recommendation would be to first define the geometry through adjacency matrices, as this datatype proved effective in the deep learning models. Additionally, adjacency matrix data can be converted to coordinate data without loss.

The dataset was labelled on three performance types: material use, structural performance and stock similarity. Therefore, the structures were optimized based on these factors. Other criteria were not considered.

VARIATIONAL AUTOENCODER

A regularised space containing all meshes of the dataset and the gradients between them was created through a variational autoencoder (VAE). This was the selected architecture for the generative model. A significant property of VAEs is the latent space dimension. Generally, lower latent spaces are preferable. The lowest latent dimension found in a well performing model in this work is 5. To further compress the framework and decrease computational cost, further decrease of this parameter can be researched.

In this work, densely-connected VAE and convolutional VAE architectures were explored. In future work, a graph VAE architecture could be explored also. Additionally, other generative deep learning models can be explored, including generative adversarial networks (GANs), as these have been shown to train on this type of data as well in this thesis.

SURROGATE MODEL

In the framework, performance score calculation on the original dataset is done through algorithms and FEM simulation. If these methods were applied in the gradient descent optimization step, this would be a time-consuming process with high computational cost. So, to compress this job, a predictive surrogate model was defined.

This model predicts the performance scores of meshes from the dataset with good accuracy. Still, calculation of performance should still be done on newly generated models to check if improvement of performance was achieved. Especially since the model accuracy slightly decreased near at the extremes of the labelling range. This can be improved through dataset augmentation. Alternatively, well-performing newly generated models could be added to the surrogate model's training data to increase its performance on high scoring meshes..

GRADIENT DESCENT

Gradient Descent (GD) was used to find well-performing meshes in the VAE's latent space through retrieval of gradients in predicted performance. These gradients were then used to find (local) maxima. In this work, the framework was able to produce new mesh designs, based on an input mesh. These meshes were predicted to perform better than the input mesh by the surrogate model. Re-calculation of the performance scores of these meshes was unable to confirm this, however.

The GD algorithm can be improved through improvement of the surrogate model and VAE, as these models are both called in the GD model. Limiting the latent space dimension in the VAE model can decrease the GD algorithm complexity. Another suggestion is exploration of Keras Optimizers such as Adam and RMSprop. The main recommendation for improvement, however, is simplification of the design task or improvement of the VAE. This is because the main issue with newly generated meshes was due to highly different topolgies; essential edges were not generated.

6.3 LIMITATIONS, APPLICATION & FUTURE DEVELOPMENT

In this section, further limitations, practical application and transferability of the work to the present and future building industry are discussed.

REFURBISHMENT AND REPAIRS IN STOCK

Realistic stock can include steel elements of insufficient quality for direct reuse. These elements could require refurbishment and/or repairs, which adds cost and time to the project. When the framework as developed in this thesis is applied directly, refurbishments and/or repairs and the associated costs should be evaluated separately. Firstly, it is possible to exclude elements that require repairs or refurbishment from the initial dataset used as stock. This way, the refurbishment time and costs are limited upfront. It is also possible to take a sample of the best performing meshes based on the full stock and evaluate these based on the required refurbishments of the respective elements of each mesh to find the best balance between cost and performance. Therefore, even though the method does not explicitly take refurbishment and repairs into account, the evaluation process can be added as an extra step before or after the application of the method.

ALTERING OF ELEMENTS RELATED TO SIMILARITY

Currently, the similarity assessment process takes the possibility for splitting items into account. This was done to allow the meshes to be recreated more closely and match the actions that would realistically be taken to achieve this. This might not be desirable for some projects, as these longer items may be used better in other structures and splitting items adds cost. In this case, the similarity assessment code should be slightly altered to remove this feature.

In addition, the joining of elements is not considered in the similarity assessment. The primary reason for this is to keep the similarity assessment simple to maximize the chances of the surrogate model and variational autoencoder producing accurate outputs. Secondly, welding together steel elements results in inconsistent structural quality throughout the element. Through small additions to the similarity assessment code, however, the possibility for joining items together could easily be implemented into the process, should this be a suitable property for another case study.

STOCK ELEMENT PROPERTIES

In this thesis, the status and dimensions of these elements are not taken into account. This limits the applications of the current method. It is however still directly applicable in early stages of the design process to aid with topology exploration. In this case, the method should to be applied to a stock that is of adequate quality. Additionally, the way the framework is set up allows for relatively easy integration of attributes such as cross section dimensions, type and structural performance. These can be added as markers in the library dataset and taken into account in the FEM model for structural assessment of generated meshes. However, it is very probable that this data also needs to be input in both the surrogate NN model and the variational autoencoder to gain accurate results. The addition of these material characteristics would complicate the data, thus making this process more difficult.

In this thesis, the main aim is to gain insight into the use of these models for generation and optimization of structural meshes. Therefore it was chosen to only include the lengths of the stock elements.

CIRCULARITY

As mentioned in Chapter 1, the circular economy model describes a hierarchy of three approaches to handling waste materials: (1) reuse, (2) remanufacturing and (3) recycling. Therefore, the focus of this thesis is on reuse. In practice, reuse of elements is still rare. This means that stocks of reusable are not generally available. As the framework developed in this work requires a clear inventory of available items, this could limit the use of the framework in practice. However, as the industry moves towards sustainability, reusing materials might become more commonplace in the building sector. This framework could provide an effective and quick way to integrate reuse of a specific stock into the architectural design process.

Another factor that is not taken into account in this work, however, is the suitability of the stock element for a specific case. Ideally, all structures that are created in the same timeframe in an area should be evaluated on stock use at the same time. This can be implemented into the similarity assessment in its current form; the elements of all generated meshes for all structural cases can be added to one list to be used as input. This, however, would significantly increase the options that can be made, therefore increasing complexity of the overall task. This approach should therefore be researched separately to ensure its accuracy and applicability.

PROPOSED FUTURE CHANGES TO THE WORKFLOW

As concluded, the workflow does not produce meshes that perform better than their original input, even though the surrogate model predicts

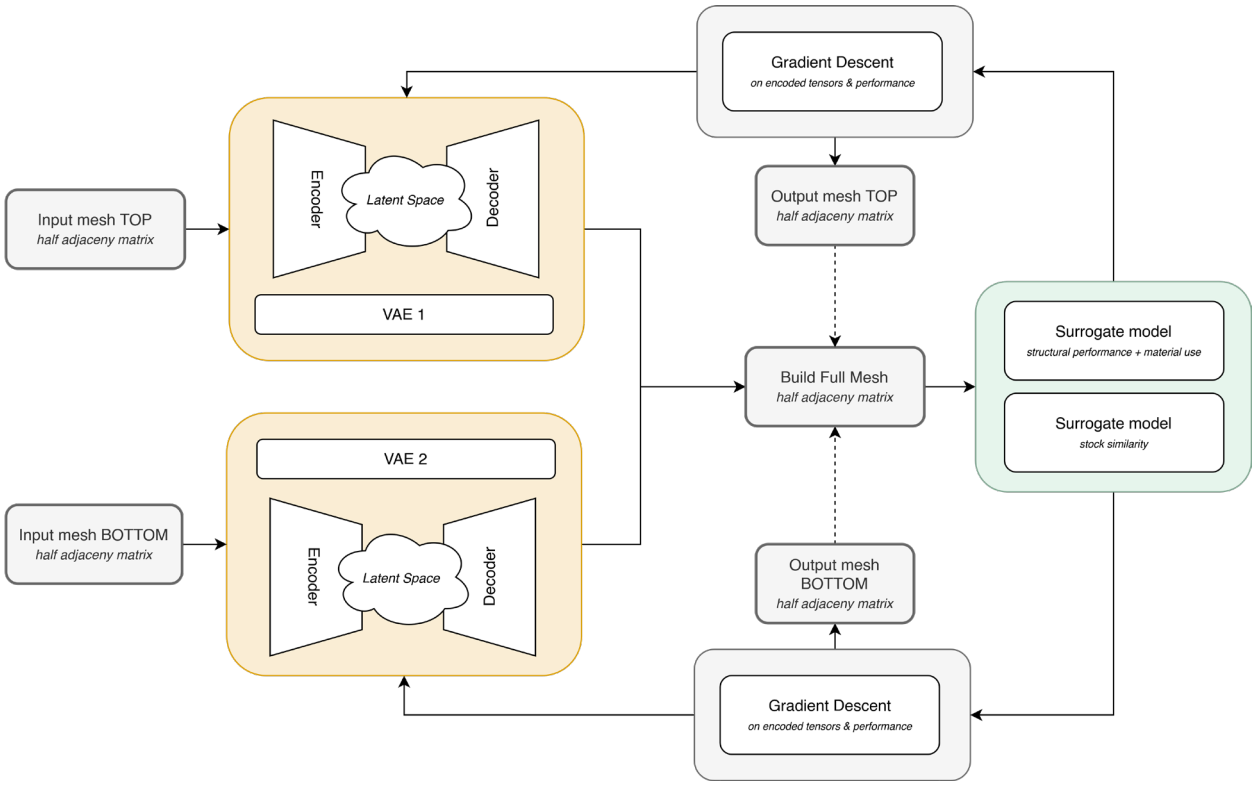


Figure 6.3.1. A proposed framework for future work
Own work.

better output. One of the possible reasons for this was identified to be movement of vertices to the top layer. These movements change the topology of the structure significantly. To counter this, a new workflow may be used. The proposed design of this framework is shown in figure 6.3.1. Here, the structure is split up in a top and bottom layer. The in-between structure can be generated from this data. By adjusting the adjacency matrix sizes, the locations of vertices can be limited.

A disadvantage of such a framework would be that more pre-processing of data is needed, as separate datasets describing top and bottom layers of each mesh need to be created. Secondly, a new tool needs to be created for rebuilding of the meshes. Finally, two separate VAE models are required in this framework. This could cause the framework to be more computationally heavy. However, it is probable that these VAE models utilise smaller architectures, since the input data shape will be significantly smaller. Finally, it should be noted that this new workflow is specifically applicable to the case study used in this work; it cannot consistently be applied to any 3D structure. The workflow as originally proposed could be more easily extended to various 3D structures consisting of linear elements. Still, many layered structures exist in architecture and construction. Therefore, this proposed workflow could still serve as a useful tool.

The main advantage of this proposed workflow is simplification of the task performed by the VAE models and GD optimization, resulting in a functional workflow that produces better performing mesh structures. Therefore, this workflow could be considered and tested in future work.

ADDITIONAL FUTURE APPLICATION OF THE WORK

The method described in this work is a stepping stone towards the use of AI-guided processes in architectural and structural design. Therefore there are various possible future applications and developments that can be researched. Some of these have been mentioned previously, such as the adjustment of the similarity assessment process to fit any specific design case, the addition of various material properties to the stock data, an the addition of stock data regarding refurbishment and repair needs.

Furthermore, the framework could be tested on other structural types or more elaborate or complicated structures. This would further show the possible applications and limits of this method. Similarly, it could be tested on more simple meshes, to get a more clear idea on the data representations and their effectivity in the deep learning/NN models.

Eventually, the method could be expanded to various aspects of structure design, including (complex) roofs, building shells, bridges, etc. It could also be expanded to include structural types that do not consist of linear elements only, but curved elements or even 3D shaped elements.

In conclusion, the framework designed in this thesis can be directly applied in the early design stages. However, its limits, as described above, should be taken into consideration. The main aim of this thesis however is exploration for the development of generative AI tools for architectural and structural design purposes.

7. References

7 References

- Akanbi, L. A., Oyedele, L. O., Akinade, O. O., Ajayi, A. O., Delgado, M. D., Bilal, M., & Bello, S. A. (2018). Salvaging building materials in a circular economy: A BIM-based whole-life performance estimator. Resources, Conservation and Recycling, 129, 175-186.
- Allwood, J. M., Cullen, J. M., Carruth, M. A., Cooper, D. R., McBrien, M., Milford, R. L., ... & Patel, A. C. (2012). Sustainable materials: with both eyes open (Vol. 2012). Cambridge, UK: UIT Cambridge Limited.
- Arjovsky, M., Chintala, S., & Bottou, L. (2017, July). Wasserstein generative adversarial networks. In International conference on machine learning (pp. 214-223). PMLR.
- Artificial Intelligence. (2021, December 23). In Oxford English Dictionary. https://www.oed.com/viewdictionaryentry/Entry/271625
- Axom. (n.d.). Mesh Representation. https://axom.readthedocs.io/en/develop/axom/mint/docs/sphinx/sections/mesh_representation.html
- Baker-Brown, D. (2019). Brummen Town Hall and a new HQ for Alliander, by RAU Architects and Turntoo. In The Re-Use Atlas (pp. 127-132). RIBA Publishing.
- BAMB2020. (n.d.). Material Passports. https://www.bamb2020.eu/topics/materials-passports/.
- Baumgart, B. G. (1972). Winged edge polyhedron representation. STANFORD UNIV CA DEPT OF COMPUTER SCIENCE.
- Berkeley. (n.d.). An Introduction to Half-Edge Data Structure. https://cs184.eecs.berkeley.edu/sp20/article/17/an-introduction-to-half-edge-dat
- Brownlee, J. (2019a). How to use Learning Curves to Diagnose Machine Learning Model Performance. Machine Learning Mastery. https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/
- Brownlee, J. (2019b). Overfitting and Underfitting With Machine Learning Algorithms. Machine Learning Mastery. https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/
- Brütting, J., Desruelle, J., Senatore, G., & Fivet, C. (2019). Design of Truss Structures Through Reuse. Structures, 18, 128–137. https://doi.org/10.1016/j.istruc.2018.11.006
- Brütting, J., Desruelle, J., Senatore, G., & Fivet, C. (2019, April). Design of truss structures through reuse. In Structures (Vol. 18, pp. 128-137). Elsevier.
- Bushaev, V. (2017). How do we 'train' neural networks?. Towards Data Science. https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73
- Cichocka, J. M., Browne, W. N., & Rodriguez, E. (2017). Optimization in the architectural practice. In Protocols, Flows and Glitches: Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) 2017 (pp. 387-396). The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA).
- Cui, J., & Tang, M. X. (2017). Towards generative systems for supporting product design. International Journal of Design Engineering, 7(1), 1-16.
- Dalvinder, P., & Grewal, S. (2014). A Critical Conceptual Analysis of Definitions of Artificial Intelligence as Applicable to Computer Engineering (Issue 2). Ver. I. www.iosrjournals.org
- Ding, Z., Wang, Y., & Zou, P. X. (2016). An agent based environmental impact assessment of building demolition waste management: Conventional versus green management. Journal of Cleaner Production, 133, 1136-1153.
- Dunant, C. F., Drewniok, M. P., Sansom, M., Corbey, S., Allwood, J. M., & Cullen, J. M. (2017). Real and perceived barriers to steel reuse across the UK construction value chain. Resources, Conservation and Recycling, 126, 118–131. https://doi.org/10.1016/j.resconrec.2017.07.036
- EPA (2020). 2018 Facts and Figures Fact Sheet. https://www.epa.gov/facts-and-figures-about-materials-waste-and-recycling/advancing-sustainable-materials-management
- European Energy Innovation. (2017). Buildings As Material Banks: Where Are We Now?. https://www.europeanenergyinnovation.eu/Latest-Research/Summer-2017/BAMB-Buildings-As-Material-Banks-Where-are-we-now.
- Eurostat (2022). Waste Statistics. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Waste_statistics#Total_waste_generation
- Feucht, T., Lange, J., & Erven, M. (2019). 3-D-Printing with Steel: Additive Manufacturing of Connection Elements and Beam Reinforcements.

ce/papers, 3(3-4), 343-348.
- Frazer, J. (2002). Creative design and the generative evolutionary paradigm. In Creative evolutionary systems (pp. 253-274). Morgan Kaufmann.
- Fujita, M., & Kuki, K. (2016). An evaluation of mechanical properties with the hardness of building steel structural members for reuse by NDT. Metals, 6(10), 247.
- Geeksforgeeks. (2022). Neural Networks | A beginners guide. https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/
- GlobalABC. (2020). The 2020 Global Status Report for Buildings and Construction. United Nations Environmental Programme.
- Goodfellow, I., Pouget-Abadie, J. Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems, 27, 2672-2680.
- Gorgolewski, M. (2017). Resource salvation: The architecture of reuse. John Wiley & Sons.
- Guo, S. (2020). An introduction to Surrogate modeling, Part I: fundamentals. Towards Data Science. https://towardsdatascience.com/an-introduction-to-surrogate-modeling-part-i-fundamentals-84697ce4d241#91b2.
- Harding, J. (n.d.). Biomorpher. Grasshopper. https://www.grasshopper3d.com/group/biomorpher
- Heinrich, M., & Lang, W. (2019). Materials Passports - Best Practice. Technische Universität München, Fakultät für Architektur.
- Helm, J. M., Swiergosz, A. M., Haeberle, H. S., Karnuta, J. M., Schaffer, J. L., Krebs, V. E., Spitzer, A. I., & Ramkumar, P. N. (2020). Machine Learning and Artificial Intelligence: Definitions, Applications, and Future Directions. In Current Reviews in Musculoskeletal Medicine (Vol. 13, Issue 1, pp. 69–76). Springer. https://doi.org/10.1007/s12178-020-09600-8
- Iacovidou, E., & Purnell, P. (2016). Mining the physical infrastructure: Opportunities, barriers and interventions in promoting structural components reuse. Science of the Total Environment, 557, 791-807.
- IBM. (n.d.) What are Neural Networks?. https://www.ibm.com/topics/neural-networks#:~:text=Neural%20networks%2C
- Jaisawal, R., & Agrawal, V. (2021). Generative Design Method (GDM) – A State of Art. IOP Conference Series: Materials Science and Engineering, 1104(1), 012036. https://doi.org/10.1088/1757-899x/1104/1/012036
- Jin, R., Yuan, H., & Chen, Q. (2019). Science mapping approach to assisting the review of construction and demolition waste management research published between 2009 and 2018. Resources, Conservation and Recycling, 140, 175-188.
- Karamba3D. (n.d.). Welcome to Karamba3D. https://manual.karamba3d.com/
- Kavlakoglu, E. (2020). AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?. IBM. https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks
- KiKaBeN. (2021). Gradient Descent Optimizers: Understanding SGD, Momentum, Nesterov Momentum, AdaGrad, RMSprop, AdaDelta, and ADAM – Made Easy. https://kikaben.com/gradient-descent-optimizers/.
- Kingma, D. P., & Welling, M. (2014). Auto-encoding variational Bayes. 2nd international conference on learning representations (ICLR2014). Preprint, submitted December, 23, 2013.
- Kosky, P., Balmer, R., Balmer, R. T., Keat, W. D., & Wise, G. (2012). Exploring engineering: an introduction to engineering and design. Academic Press.
- Kuang, W. (2018). Types of Steel Connections and their Classifications. SkyCiv. https://skyciv.com/technical/types-of-steel-connections-and-their-classifications/.
- Kullback, S. (1959). Information theory and statistics. john riley and sons. Inc. New York.
- Lange, J., Feucht, T., & Erven, M. (2020). 3D printing with steel: Additive Manufacturing for connections and structures. Steel Construction, 13(3), 144-153.
- Lunot, V. (2019). On the use of the Kullback–Leibler divergence in Variational Autoencoders. https://www.vincent-lunot.com/post/on-the-use-of-the-kullback-leibler-divergence-in-variational-autoencoders/
- Meintjes, K. (2017). "Generative Design" – What's That?. CIMdata. https://www.cimdata.com/en/news/item/8402-generative-design-what-s-that.
- Nanos, G. (2013). Baeldung CS. Generative Adversarial Networks: Discriminator's Loss and Generator's Loss. https://www.baeldung.com/cs/gan-loss.
- Odaibo, S. (2019). Tutorial: Deriving the standard variational autoencoder (vae) loss function. arXiv preprint arXiv:1907.08956.
- Oogstkaart. (n.d.). Oogstkaart: De urban mining porentie van NL. https://www.oogstkaart.nl/.
- Packer, J. A. (2018). Multi-planar Welded Connections. Steel Tube Institute.

https://steeltubeinstitute.org/resources/multi-planar-welded-connections/.
- Patrikar, S. (2019). Batch, Mini Batch & Stochastic Gradient Descent. Towards Data Science. https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a
- Pavlidou, S. (2022). Deep Generative Designs: A Deep Learning Framework for Optimized Shell Structures.
- Pongiglione, M., & Calderini, C. (2014). Material savings through structural steel reuse: A case study in Genoa. Resources, Conservation and Recycling, 86, 87-92.
- Rijksoverheid. (n.d.). Nederland circulair in 2015. https://www.rijksoverheid.nl/onderwerpen/circulaire-economie/nederland-circulair-in-2050.
- Rijksoverheid. (2021). Uitvoeringsprogramma Circulaire Economie 2021-2023. https://www.rijksoverheid.nl/onderwerpen/circulaire-economie/documenten/rapporten/2021/10/18/uitvoeringsprogramma-circulaire-economie.
- Rocca, J. (2019a). Understanding Generative Adversarial Networks (GANs). Towards Data Science. https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29
- Rocca, J. (2019b). Understanding Variational Autoencoders (VAEs). Towards Data Science. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
- Rose, C. M., & Stegemann, J. A. (2018, September). Characterising existing buildings as material banks (E-BAMB) to enable component reuse. In Proceedings of the Institution of Civil Engineers-Engineering Sustainability (Vol. 172, No. 3, pp. 129-140). Thomas Telford Ltd.
- Roy, A. (2020). Introduction to Gradient Descent: Weight Initiation and Optimizers. Towards Data Science. https://towardsdatascience.com/introduction-to-gradient-descent-weight-initiation-and-optimizers-ee9ae212723f
- Ruiz, L. A. L., Ramón, X. R., & Domingo, S. G. (2020). The circular economy in the construction and demolition waste sector–A review and an integrative model approach. Journal of Cleaner Production, 248, 119238.
- Sachdeva, K. (2021, January 26). KL Divergence - CLEARLY EXPLAINED! [Video]. YouTube. https://www.youtube.com/watch?v=9_eZHt2qJs4
- SAFS Steel Structure. (2021). What are the General Types of Space Frame Nodes? What are the Characteristics? https://www.safsteelstructure.com/news/what-are-the-general-types-of-space-frame-nodes-what-are-the-characteristics-of-each/
- Sethi, V. (2019). Types of Neural Networks (and what each one does!) Explained. Towards Data Science. https://towardsdatascience.com/types-of-neural-network-and-what-each-one-does-explained-d9b4c0ed63a1
- Soni, D. (2018). Introduction to Evolutionary Algorithms. https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac.
- SteelConstruction. (n.d.). Simple connections. https://www.steelconstruction.info/Simple_connections.
- SteelConstruction. (n.d.). Trusses. https://www.steelconstruction.info/Trusses.
- Stevanović, D. (2014). Spectral radius of graphs. Academic Press.
- Szabo, F. (2015). The linear algebra survival guide: illustrated with Mathematica. Academic Press.
- Tang, H., Hu, Q. G., Xu, Y. Y., & Yang, Y. H. (2011). Reuse of Reclaimed Materials in Construction-From the Process Analysis of BedZED. In Applied Mechanics and Materials (Vol. 99, pp. 433-439). Trans Tech Publications Ltd.
- Taylor, P. (2010). Energy Technology Perspectives. International Energy Agency.
- Tingley, D. D., & Davison, B. (2013). Minimizing the Environmental Impact of Steel Structures. In Design, Fabrication and Economy of Metal Structures (pp. 651-656). Springer, Berlin, Heidelberg.
- Trehan, D. (2020). Gradient Descent Explained. Towards Data Science. https://towardsdatascience.com/gradient-descent-explained-9b953fc0d2c
- Twinn, C. (2003). BedZED. Arup Journal, 38(1), 10-16.
- Vadapalli, P. (2020). 7 Types of Neural Networks in Artificial Intelligence Explained. upGrad. https://www.upgrad.com/blog/types-of-neural-networks/
- Vidiyala, R. (2020). 6 Types of Neural Networks Every Data Scientist Must Know. Towards Data Science. https://towardsdatascience.com/6-types-of-neural-networks-every-data-scientist-must-know-9c0d920e7fce
- Wang, L., Jin, H., Dong, H., & Li, J. (2013). Balance fatigue design of cast steel nodes in tubular steel structures. The Scientific World Journal, 2013.
- Wei, Z., Pei, X., & Jin, H. (2021). Evaluation of welded cast steel joint fatigue data using structural stress methods. Journal of Constructional Steel Research, 186, 106895.
- Wikipedia (2023). Polygon mesh. https://en.wikipedia.org/wiki/Polygon_mesh
- Wolfram MathWorld. (n.d.). Adjacency Matrix. https://mathworld.wolfram.com/AdjacencyMatrix.html.

- Wood, T. (n.d.). Backproagation. DeepAI. https://deepai.org/machine-learning-glossary-and-terms/backpropagation
- World Steel Association. (2021). Climate change and the production of iron and steel.
- Zeng, X., & Li, J. (2021). Emerging anthropogenic circularity science: Principles, practices, and challenges. Iscience, 24(3), 102237
- Urbanscape. (2015). How much does a Green Roof weight?. https://www.urbanscape-architecture.com/how-much-does-a-green-roof-weigh/.
- Sempergreen. (n.d.). How much does a green roof weigh?. https://www.sempergreen.com/en/solutions/green-roofs/frequently-asked-questions-green-roof/how-much-does-a-green-roof-weigh.
- Maguire Brothers. (2022). Weight Loading On A Roof. https://www.maguirebrothers.co.uk/weight-loading-on-a-roof.
- Eurocode. (2019). NEN-EN 1991-1-1:2002+C11:2019+NB:2019 Eurocode 1: Belastingen op constructies Deel 1-1: Algemene belastingen - Volumieke gewichten, eigengewicht en gebruiksbelastingen voor gebouwen.
- Chollet, F. (2016). Building Autoencoders in Keras. The Keras Blog. https://blog.keras.io/building-autoencoders-in-keras.html

8. Appendix

8 Appendix

In this appendix, the Grasshopper model, deep learning framework and full results can be found. The Appendix consist of the following sections:

- Appendix I: Graduation Timeline
- Appendix II: Dataset Generation & Performance Assessment
- Appendix III: AI Framework
- Appendix IV: Surrogate model results
- Appendix V: VAE model results
- Appendix: VI: Gradient descent results

Appendix I

Graduation Timeline

GRADUATION TIMELINE

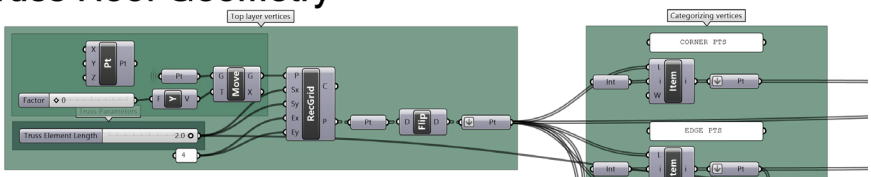
	Graduation Topic Oriented Research
22-Nov-22	**P1 Presentation**
	Literature Study on AI Generative Models
	Literature Study on Case Studies & Application
	Experiment with Generative Models & Tutorials
	Study Grasshopper plug-ins & Python libraries to use in training dataset generation
	Set up framework for training dataset generation
	Set up framework for creating stock library
	Write Report
25-Jan-23	**P2 Presentation**
	Generate training dataset
	Create stock library
05-Feb-23	Finish training dataset & stock library
	Set up framework for structural performance analysis
12-Feb-23	Finish structural performance analysis
	Set up framework for similarity to stock assessment
19-Feb-23	Finish similarity to stock assessment
	Create Surrogate Model
	Train & Evaluate Surrogate Model
	Create Variational Autoencoder Framework
	Train & Evaluate the VAE
11-Apr-23	**P3 Presentation**
	Further train the NN models
	Create Gradient Descent Optimizer
	Train & Evaluate Final Surrogate Model/VAE
	Use Gradient Descent Optimizer
	Process results
01-May-23	Finish results
	Write conclusions, discussion & finish report
23-May-23	**P4 Presentation**
	Final improvements
27-Jun-23	**P5 Presentation**

Appendix II

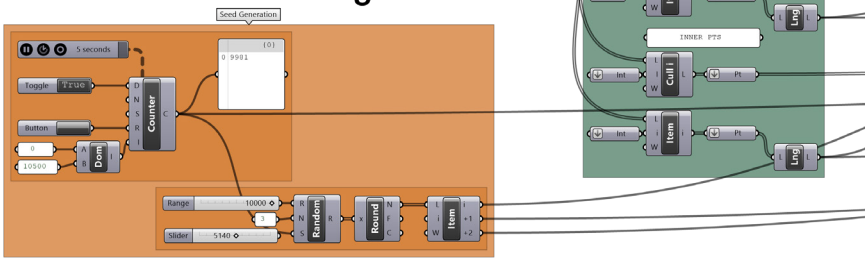
Dataset Generation & Performance Assessment

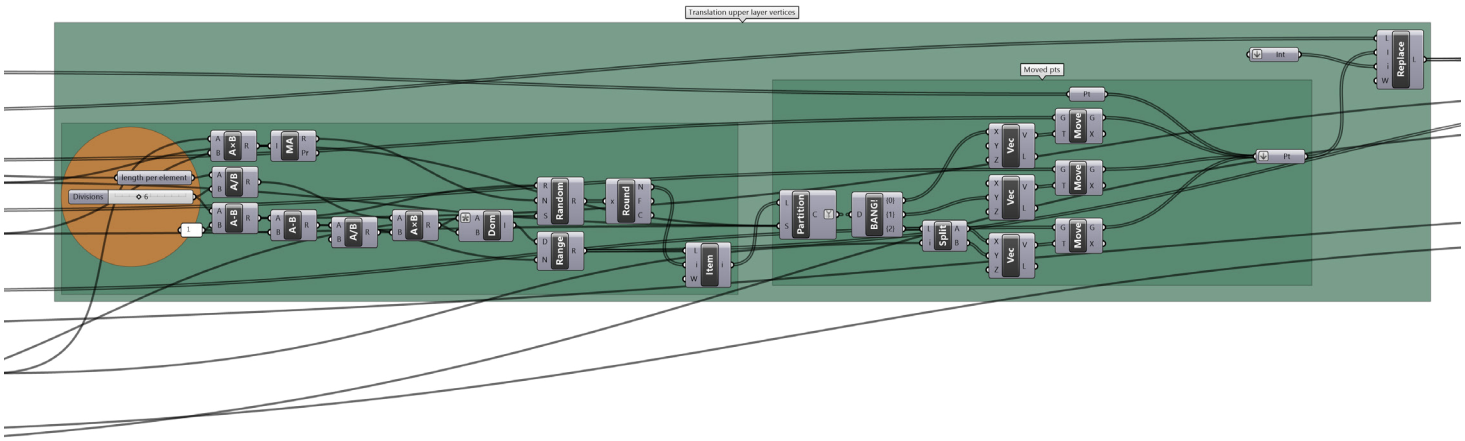
Dataset Generation – Grasshopper

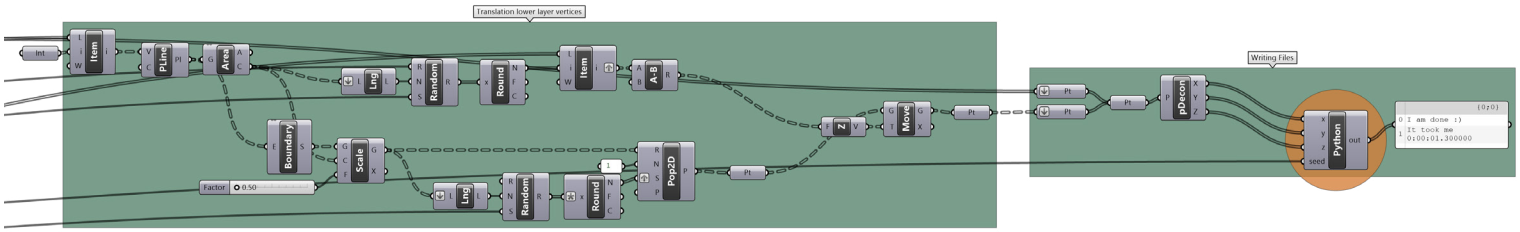
Truss Roof Geometry



Auto Generation Settings







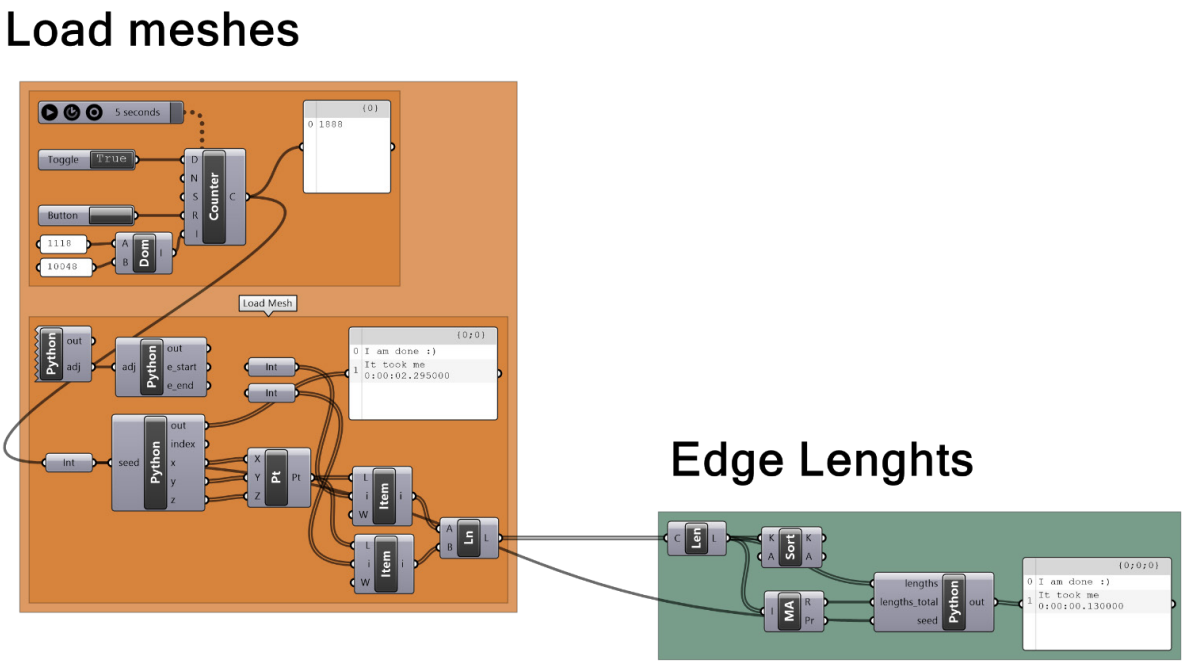
Dataset Generation – GhPython: Adjacency Matrix

```
import rhinoscriptsyntax as rs
import csv
import scriptcontext
np = scriptcontext.sticky['numpy']

#GET ADJACENCY MATRIX
adj = np.zeros((41, 41))

for i in range(len(e_start)):
    adj[e_start[i]][e_end[i]] = 1
    adj[e_end[i]][e_start[i]] = 1

np.savetxt("C:/Users/amyst/Desktop /dataset/AdjMatrix.csv", adj, fmt=b"%i", delimiter =
",")
```

Dataset Generation – GhPython: Coordinate Matrix

```
import rhinoscriptsyntax as rs
import csv
import datetime as dt
import scriptcontext
np = scriptcontext.sticky['numpy']

t_start = dt.datetime.now()

#COORDINATE MATRIX
xyz_coordinates = np.zeros((41,4))
for i in range(len(x)):
    xyz_coordinates[i][0] = i
    xyz_coordinates[i][1] = x[i]
    xyz_coordinates[i][2] = y[i]
    xyz_coordinates[i][3] = z[i]

np.savetxt("C:/Users/amyst/Desktop/dataset/mesh%s.csv" %seed, xyz_coordinates,
fmt=b"%.5f", delimiter = ",")

t_end = dt.datetime.now()

print("I am done :)")
print("It took me %s" %(t_end-t_start))
```

Dataset Generation – Grasshopper: Finding Element Lengths & Total Lengths



Dataset Generation – GhPython: Load Adjacency Matrix

```
import rhinoscriptsyntax as rs
import csv
import scriptcontext
np = scriptcontext.sticky['numpy']

#LOAD ADJACENCY MATRIX
adj = np.loadtxt("C:/Users/amyst/Desktop/dataset/AdjMatrix_simplified.csv", delimiter =
",")
```

Dataset Generation – GhPython: Process Adjacency Matrix

```
import rhinoscriptsyntax as rs
import csv
import datetime as dt
import scriptcontext
np = scriptcontext.sticky['numpy']

t_start = dt.datetime.now()

e_start = []
e_end = []

for i in range(len(adj)):
    print(adj[i])
    for j in range(len(adj[0])):
        adj[i][j] = getattr(adj[i][j], "tolist", lambda: adj[i][j])()
        if adj[i][j] == 1:
                e_start.append(i)
                e_end.append(j)

t_end = dt.datetime.now()

print("I am done :)")
print("It took me %s" %(t_end-t_start))
```

Dataset Generation – GhPython: Save Element Lengths & Material Use

```
import rhinoscriptsyntax as rs
import csv
import datetime as dt
import scriptcontext
np = scriptcontext.sticky['numpy']

t_start = dt.datetime.now()

#SAVE TOTAL & INDIVIDUAL LENGTHS
material_use = np.zeros((1,2))

material_use[0][0] = seed
material_use[0][1] = lengths_total

np.savetxt("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_material_use.csv"
%seed, material_use,  fmt=b"%.5f", delimiter = ",")
np.savetxt("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_lengths.csv" %seed,
lengths,  fmt=b"%.5f", delimiter = ",")

t_end = dt.datetime.now()

print("I am done :)")
print("It took me %s" %(t_end-t_start))
```

Dataset Generation – GhPython: Delete Invalid & Duplicate Meshes & Create Full Performance Dataset

```
import numpy as np
from functools import partial

# NORMALIZED PERFORMANCE SCORE CALCULATION DEF
def normalize(value_list, name):
    max_value = max(value_list)
    min_value = min(value_list)

    print('\n')
    print('Max total length:')
    print(max_value)
    print('Min total length:')
    print(min_value)
    print('\n')

    normalized_values = []

    for i in value_list:
        normalized_values.append(1 - ((i - min_value) / (max_value - min_value)))

    print('Normalized %s per valid mesh:' %name)
    for i in range(len(normalized_values)):
        print(indices_kept[i], normalized_values[i])
    print('\n')

    return normalized_values

# SAVE PERFORMANCE DATA DEF
def save_performance_scores(p_value_list, name):
    performance_array = np.zeros((1, 2))
    for i in range(len(p_value_list)):
        performance_array[0][0] = indices_kept[i]
```

```
        performance_array[0][1] = p_value_list[i]


np.savetxt("C:/Users/amyst/Desktop/dataset/performance_data/test_files/mesh%s_normalized_
%s_score.csv" %
                   (indices_kept[i], name ), performance_array, fmt='%s',
                   delimiter=",")

# FIND DUPLICATES DEF
def list_duplicates_of(seq,
                       item):  # def source:
```

*Code source: Stack Overflow. (2015). Index of duplicates items in a python list. https://stackoverflow.com/questions/5419204/index-of-duplicates-items-in-a-python-list

```
    start_at = -1
    locs = []
    while True:
        try:
            loc = seq.index(item, start_at + 1)
        except ValueError:
            break
        else:
            locs.append(loc)
            start_at = loc
    return locs

# DELETE INVALID MESHES DEF
def delete_invalid_meshes(indices_to_remove, mesh_indices):
    for i in indices_to_remove:
        if i in mesh_indices:
            mesh_indices.remove(i)
    return mesh_indices


# LOADING OF ELEMENT LENGHTS
mesh_indices = list(range(0, 10049))
indices_to_remove = []
indices_kept = []

for i in mesh_indices:  # change number to full sample size
    material_use =
np.loadtxt("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_material_use.csv" % i,
                               delimiter=",")
    if str(material_use[1]) == "nan":
        print("Mesh %s is removed" % i)  # REMOVE MESHES FROM FAILED MESH GENERATION
        indices_to_remove.append(i)
    else:
        indices_kept.append(i)


# DELETE INVALID MESHES
mesh_indices = delete_invalid_meshes(indices_to_remove, mesh_indices)

# LOADING OF TOTAL LENGHTS
lengths_full = []
indices_kept = []

for i in mesh_indices:
    lengths =
tuple(np.loadtxt("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_lengths.csv" %
```

```
i, delimiter=","))
    if len(lengths) == 128:
        lengths_full.append(lengths)
        indices_kept.append(i)
    else:
        print("Mesh %s is removed" % i)  # REMOVE MESHES FROM FAILED MATERIAL USE
CALCULATION
        indices_to_remove.append(i)

# DELETE INVALID MESHES
mesh_indices = delete_invalid_meshes(indices_to_remove, mesh_indices)


# REMOVE MESHES FROM FAILED STRUCTURAL PERFORMANCE CALCULATION
indices_kept = []

for i in mesh_indices:
    structural_perf =
np.loadtxt("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_structural_data.csv" %
i,
                               delimiter=",")
    if str(structural_perf[0]) == "nan" or str(structural_perf[1]) == "nan":
        print("Mesh %s is removed" % i)  # REMOVE MESHES FROM FAILED STRUCTURAL
PERFORMANCE SIMULATION
        indices_to_remove.append(i)
    else:
        indices_kept.append(i)

# DELETE INVALID MESHES
mesh_indices = delete_invalid_meshes(indices_to_remove, mesh_indices)


# REMOVE MESHES FROM FAILED STOCK SIMILARITY CALCULATION
indices_kept_material_and_structural = indices_kept.copy()
indices_kept = []
indices_kept_similarity =
np.loadtxt("C:/Users/amyst/Desktop/dataset/supporting_files/valid_indices_similarity.csv"
, delimiter=",")

indices_kept_similarity = list(indices_kept_similarity)

for i in range(10049):
    if i in indices_kept_material_and_structural and i in indices_kept_similarity:
        indices_kept.append(i)
    else:
        indices_to_remove.append(i)

# DELETE INVALID MESHES
mesh_indices = delete_invalid_meshes(indices_to_remove, mesh_indices)

# FIND DUPLICATES
dup_len = partial(list_duplicates_of, lengths_full)
unique_len = list(set(lengths_full))

if len(unique_len) == len(lengths_full):
    print('\n')
    print("No duplicates found!")
    unique_indices = mesh_indices
```

```
    print('Indices of unique, valid meshes:')
    print(unique_indices)
else:
    unique_indices = []

    print('\n')
    print('List of duplicates per valid mesh:')
    for i in range(len(unique_len)):
        print(unique_len[i], dup_len(unique_len[i]))
        unique_indices.append(dup_len(unique_len[i])[0])  # items to keep (indices)
    print('\n')
    unique_indices.sort()
    print('Indices of unique, valid meshes:')
    print(unique_indices)

np.savetxt("C:/Users/amyst/Desktop/dataset/performance_data/test_files/valid_mesh_indices
.csv", unique_indices, fmt='%s',
           delimiter=",")
np.savetxt("C:/Users/amyst/Desktop/dataset/performance_data/test_files/invalid_mesh_indic
es.csv", indices_to_remove, fmt='%s',
           delimiter=",")

#COLLECT DATA FOR VALID INDICES TO CREATE DATASET (UNAUGMENTED)

# 0 Mesh Index (mesh_index)
# 1 Original Mesh Seed (mesh_seed)
# 2 Material Use Performance Score (p_material_use)
# 3 Stock Similarity Perfomance Score (p_stock_similarity)
# 4 Displacement Perfomance Score (p_displacement)
# 5 Utilization Performance Score (p_utilization)
# 6 Structural Performance Score (p_structural)
# 7 Overall Performance Score (p_overall)

mesh_seed = mesh_indices.copy()
mesh_index = list(range(len(mesh_seed)))
dataset_size = len(mesh_seed)
full_dataset_unaugmented = np.zeros((dataset_size, 8))

material_use_full = []
displacement_full = []
utilization_full = []
stock_similarity_full = []

for i in mesh_indices:
    material_use =
np.loadtxt("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_material_use.csv" % i,
                               delimiter=",")
    material_use_full.append(material_use[1])
    structural_perf =
np.loadtxt("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_structural_data.csv" %
i,
                               delimiter=",")
    displacement_full.append(structural_perf[0])
    utilization_full.append(structural_perf[1])
    similarity_perf =
np.loadtxt("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_similarity.csv" % i,
                               delimiter=",")
    stock_similarity_full.append(similarity_perf)
```

```
# NORMALIZED DATA
p_material_use = normalize(material_use_full, "Material Use")
p_displacement = normalize(displacement_full, "Displacement")
p_utilization = normalize(utilization_full, "Utilization")
p_stock_similarity = normalize(stock_similarity_full, "Similarity")

# CALCULATE TOTAL STRUCTURAL PERFORMANCE SCORE
p_structural = []
for i in range(len(p_displacement)):
    p_structural.append(0.6 * p_displacement[i] + 0.4 * p_utilization[i])


#CHECK IF ALL PERFORMANCE LISTS ARE THE SAME LENGTH:
print(len(mesh_index), len(mesh_seed), len(p_material_use), len(p_stock_similarity),
len(p_displacement), len(p_utilization),len(p_structural))

#CALCULATE OVERALL PERFORMANCE SCORE
p_overall = []
for i in range(len(mesh_indices)):
    p_overall.append(0.2 * p_material_use[i] + 0.4 * p_stock_similarity[i] + 0.4 *
p_structural[i])

# SAVE FULL DATASET
for i in range(dataset_size):
    full_dataset_unaugmented[i][0] = mesh_index[i]
    full_dataset_unaugmented[i][1] = mesh_seed[i]
    full_dataset_unaugmented[i][2] = p_material_use[i]
    full_dataset_unaugmented[i][3] = p_stock_similarity[i]
    full_dataset_unaugmented[i][4] = p_displacement[i]
    full_dataset_unaugmented[i][5] = p_utilization[i]
    full_dataset_unaugmented[i][6] = p_structural[i]
    full_dataset_unaugmented[i][7] = p_overall[i]

np.savetxt("C:/Users/amyst/Desktop/dataset/full_dataset/full_file/dataset_unaugmented.csv
", full_dataset_unaugmented, fmt=b"%.5f", delimiter=",")
```

Dataset Generation – Python: Dataset Augmentation

```
import numpy as np

def rotate90(coordinate_matrix_base, translations):
    # ROTATING MESH BY 90 DEGREES
    vertex_order = [20, 15, 10, 5, 0, 21, 16, 11, 6, 1, 22, 17, 12, 7, 2, 23, 18, 13, 8,
3, 24, 19, 14, 9, 4, 37, 33, 29, 25, 38, 34, 30, 26, 39, 35, 31, 27, 40, 36, 32, 28]
    translations_new = translations.copy()

    for i in range(len(translations_new)):
        x = -translations[i, 1]
        y = translations[i, 0]
        translations_new[i, 0] = x
        translations_new[i, 1] = y

    vertex_new = np.zeros((41, 4))

    for i in range(len(vertex_order)):
        vertex_new[i, 1:4] = coordinate_matrix_base[vertex_order[i]]
        vertex_new[i, 1:4] = vertex_new[i, 1:4] + translations_new[i]
        vertex_new[i, 0] = vertex_order[i]

    vertex_new_sorted = vertex_new[vertex_new[:, 0].argsort()]
    translations_new = coordinate_matrix_base - vertex_new_sorted[:, 1:4]

    return translations_new, vertex_new_sorted


#LOADING COORDINATE MATRICES
unaugmented_data =
np.loadtxt("C:/Users/amyst/Desktop/dataset/full_dataset/full_file/dataset_unaugmented.csv
", delimiter=",").astype(int)
mesh_indices = []

for i in unaugmented_data:
        mesh_indices.append(i[1])


print("Number of meshes in dataset: %s" %len(mesh_indices))
# mesh_indices = mesh_indices_full_list[9000:] #line used for testing - only selecting
part of the data

coordinate_matrix_base = np.loadtxt("C:/Users/amyst/Desktop/dataset/meshbase.csv",
delimiter=",")
coordinate_matrix_base = coordinate_matrix_base[:, 1:4]

for mesh in mesh_indices:
    coordinate_matrix = np.loadtxt("C:/Users/amyst/Desktop/dataset/mesh%s.csv" % mesh,
delimiter=",")

    translations = coordinate_matrix_base - coordinate_matrix[:, 1:4]
    vertex_indices = list(range(0, 41))


    #MIRRORING MESH (HORIZONTALLY)
    vertex_order_mirrored = [4, 3, 2, 1, 0, 9, 8, 7, 6, 5, 14, 13, 12, 11, 10, 19, 18,
17, 16, 15, 24, 23, 22, 21, 20, 28, 27, 26, 25, 32, 31, 30, 29, 36, 35, 34, 33, 40, 39,
38, 37]
```

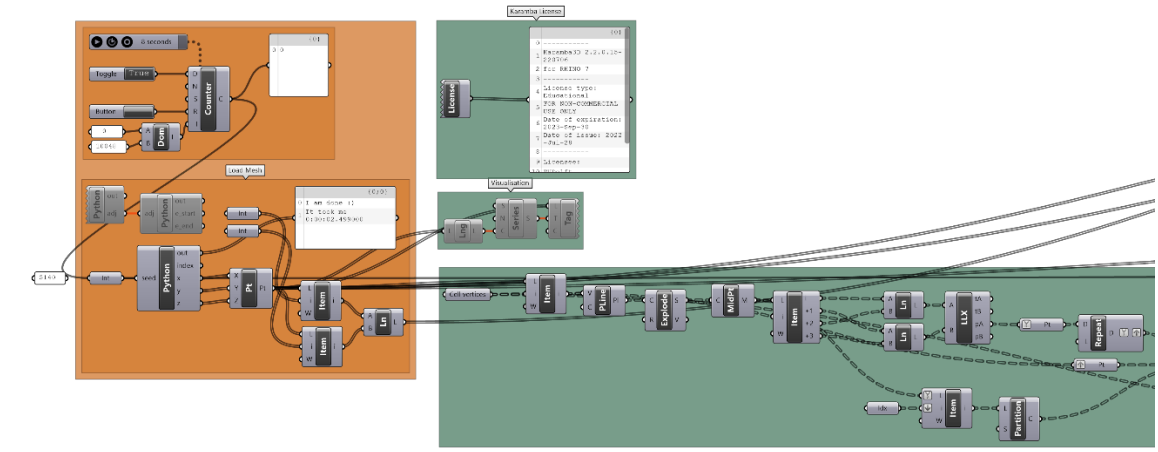
```
    translations_mirrored = translations.copy()


    for i in range(len(translations_mirrored)):
        y = -translations[i, 1]
        translations_mirrored[i, 1] = y


    vertex_mirrored = np.zeros((41, 4))
    for i in range(len(vertex_order_mirrored)):
        vertex_mirrored[i, 1:4] = coordinate_matrix_base[vertex_order_mirrored[i]]
        vertex_mirrored[i, 1:4] = vertex_mirrored[i, 1:4] + translations_mirrored[i]
        vertex_mirrored[i, 0] = vertex_order_mirrored[i]

    vertex_mirrored_sorted = vertex_mirrored[vertex_mirrored[:, 0].argsort()]
    translations_mirrored = coordinate_matrix_base - vertex_mirrored_sorted[:, 1:4]

    np.savetxt("C:/Users/amyst/Desktop/dataset/augmented_set/mesh%s_mirrored.csv" % mesh,
vertex_mirrored_sorted,  fmt=b"%.5f", delimiter = ",")


    #ROTATING MIRRORED MESH (90 DEGREES)
    translations_mirrored_rotated_90, vertex_mirrored_rotated_90_sorted =
rotate90(coordinate_matrix_base, translations_mirrored)

np.savetxt("C:/Users/amyst/Desktop/dataset/augmented_set/mesh%s_mirrored_rotated_90.csv"
% mesh, vertex_mirrored_rotated_90_sorted,  fmt=b"%.5f", delimiter = ",")

    #ROTATING MIRRORED MESH (180 DEGREES)
    translations_mirrored_rotated_180, vertex_mirrored_rotated_180_sorted =
rotate90(coordinate_matrix_base, translations_mirrored_rotated_90)

np.savetxt("C:/Users/amyst/Desktop/dataset/augmented_set/mesh%s_mirrored_rotated_180.csv"
% mesh, vertex_mirrored_rotated_180_sorted,  fmt=b"%.5f", delimiter = ",")

    #ROTATING MIRRORED MESH (270 DEGREES)
    translations_mirrored_rotated_270, vertex_mirrored_rotated_270_sorted =
rotate90(coordinate_matrix_base, translations_mirrored_rotated_180)

np.savetxt("C:/Users/amyst/Desktop/dataset/augmented_set/mesh%s_mirrored_rotated_270.csv"
% mesh, vertex_mirrored_rotated_270_sorted,  fmt=b"%.5f", delimiter = ",")

    #ROTATING MESH (90 DEGREES)
    translations_rotated_90, vertex_rotated_90_sorted = rotate90(coordinate_matrix_base,
translations)
    np.savetxt("C:/Users/amyst/Desktop/dataset/augmented_set/mesh%s_rotated_90.csv" %
mesh, vertex_rotated_90_sorted,  fmt=b"%.5f", delimiter = ",")

    #ROTATING MESH (180 DEGREES)
    translations_rotated_180, vertex_rotated_180_sorted =
rotate90(coordinate_matrix_base, translations_rotated_90)
    np.savetxt("C:/Users/amyst/Desktop/dataset/augmented_set/mesh%s_rotated_180.csv" %
mesh, vertex_rotated_180_sorted,  fmt=b"%.5f", delimiter = ",")

    #ROTATING MESH (270 DEGREES)
    translations_rotated_270, vertex_rotated_270_sorted =
rotate90(coordinate_matrix_base, translations_rotated_180)

    np.savetxt("C:/Users/amyst/Desktop/dataset/augmented_set/mesh%s_rotated_270.csv" %
mesh, vertex_rotated_270_sorted,  fmt=b"%.5f", delimiter = ",")
```

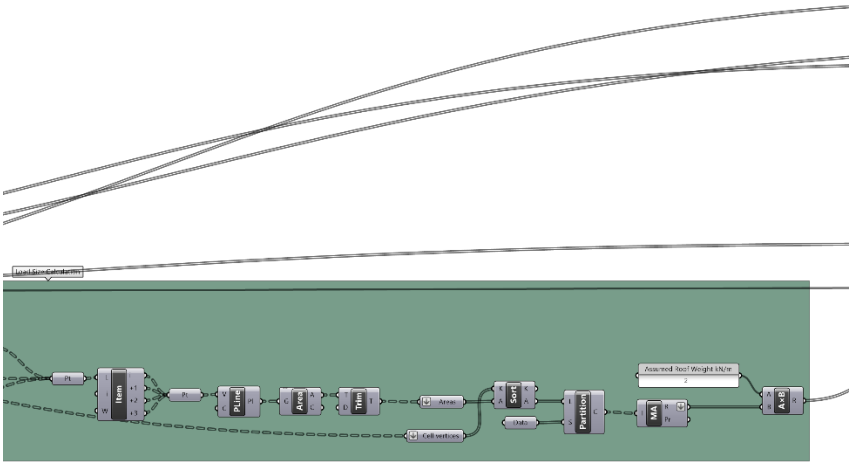
Dataset Generation – Python: Renumbering and Second Duplicate Check

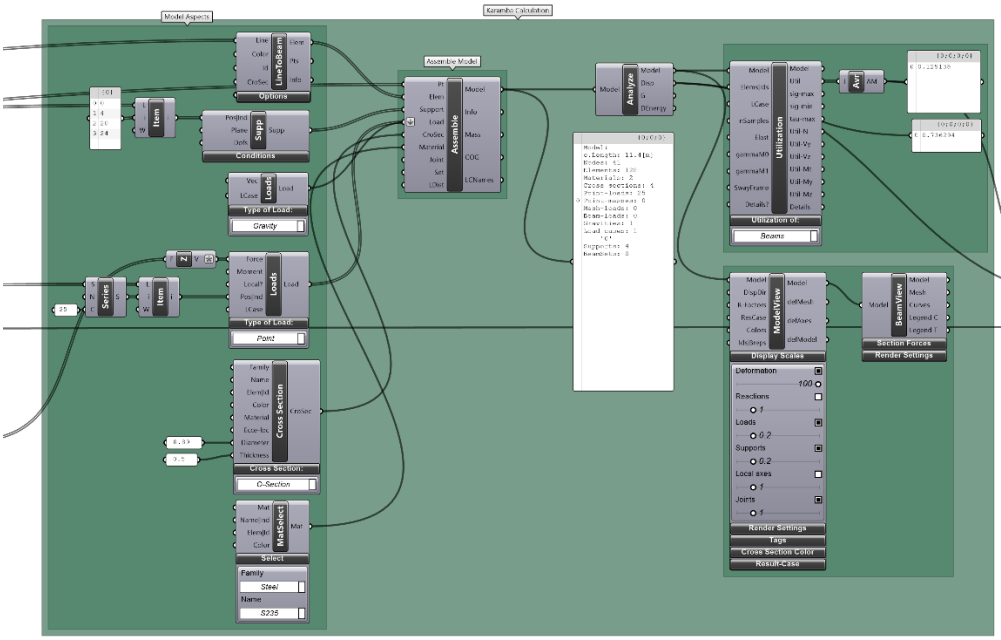
```
import numpy as np
from functools import partial


#LOAD VALID MESH INDICES
valid_mesh_indices_full_list =
np.loadtxt("C:/Users/amyst/Desktop/dataset/supporting_files/valid_mesh_indices.csv",
delimiter=",")
new_mesh_indices_full_list = list(range(len(valid_mesh_indices_full_list)))

valid_mesh_indices = valid_mesh_indices_full_list  # line was used for testing; to only
select part of the data
new_mesh_indices = new_mesh_indices_full_list # line was used for testing; to only select
part of the data


# RENUMBERING OF ORIGINAL DATASET
for i in range(len(valid_mesh_indices)):
    loaded_mesh = np.loadtxt("C:/Users/amyst/Desktop/dataset/mesh%s.csv"
%valid_mesh_indices[i], delimiter=",")
    np.savetxt("C:/Users/amyst/Desktop/dataset/full_dataset/mesh%s.csv"
%(new_mesh_indices[i]*8), loaded_mesh, fmt=b"%.5f", delimiter=",")

# RENUMBERING OF AUGMENTED DATASET
augmented_mesh_types = ["rotated_90", "rotated_180", "rotated_270", "mirrored",
"mirrored_rotated_90",
                        "mirrored_rotated_180", "mirrored_rotated_270"]

for i in range(len(valid_mesh_indices)):
    for j in range(len(augmented_mesh_types)):
        loaded_mesh =
np.loadtxt("C:/Users/amyst/Desktop/dataset/augmented_set/mesh%s_%s.csv" %
(valid_mesh_indices[i], augmented_mesh_types[j]), delimiter=",")
        np.savetxt("C:/Users/amyst/Desktop/dataset/full_dataset/mesh%s.csv" %
((j+1)+new_mesh_indices[i]*8), loaded_mesh, fmt=b"%.5f", delimiter=",")

# RENUMBERING OF PERFORMANCE DATA
for i in range(len(valid_mesh_indices)):
    loaded_mesh =
np.loadtxt("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_lengths.csv"
%valid_mesh_indices[i], delimiter=",")

np.savetxt("C:/Users/amyst/Desktop/dataset/reordered_performance_data/mesh%s_to_%s_lenght
s.csv" %((new_mesh_indices[i]*8), ((new_mesh_indices[i]+1)*8-1)), loaded_mesh,
fmt=b"%.5f", delimiter=",")
```

FEM Analysis & Structural Perfomance – Grasshopper

Load meshes







Structural Perfomance

Dataset Generation – GhPython: Load Adjacency Matrix

```
import rhinoscriptsyntax as rs
import csv
import datetime as dt
import scriptcontext
np = scriptcontext.sticky['numpy']

t_start = dt.datetime.now()

#SAVE TOTAL LENGTH
structural_performance = np.zeros((1,2))

structural_performance[0][0] = displacement
structural_performance[0][1] = utilization

np.savetxt("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_structural_data.csv"
%seed, structural_performance,  fmt=b"%f", delimiter = ",")


t_end = dt.datetime.now()

print("I am done :)")
print("It took me %s" %(t_end-t_start))
```

Similarity Assessment – Python

```
import numpy as np
from itertools import combinations
import os

#LOAD MESH
mesh_indices = list(range(0, 10049))
valid_meshes = []
library_set =
np.loadtxt("C:/Users/amyst/Desktop/dataset/library/stocklibrary_lengths_aframetrussbeams.
csv", delimiter=",")

for mesh in mesh_indices:  # change number to full sample size
    if
os.path.getsize("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_lengths.csv" %
mesh) > 0:
        sample_set =
np.loadtxt("C:/Users/amyst/Desktop/dataset/performance_data/mesh%s_lengths.csv" % mesh,
delimiter=",")
        if len(sample_set) != 128:
            print("Mesh %s is invalid." %mesh)
        else:
            print("STOCK SIMILARITY CALCULATION FOR MESH %s:" % mesh)
            valid_meshes.append(mesh)

        # CHANGE TYPES FROM ARRAY TO LIST
            sample_set = list(sample_set)
            library_set = list(library_set)

        # SORT LIST
            sample_set.sort()


            library_closest = []

            # FIND CLOSEST MATCH
            library_instock = library_set.copy()
            l_difference = []

            for i in sample_set:
                library_array = np.asarray(library_instock)
                closest_idx = (np.abs(library_array - i)).argmin()
                # print(closest_idx)
                library_closest.append(library_instock[closest_idx])
                l_difference.append(abs(i - library_instock[closest_idx]))
                del library_instock[closest_idx]

            sample_set_print = [round(n, 2) for n in sample_set]
            library_closest_print = [round(n, 2) for n in library_closest]
            l_difference_print = [round(n, 2) for n in l_difference]
            print("The original set of mesh element lengths: %s" %sample_set_print)
            print("Set of the closest matching items from library: %s"
%library_closest_print)
            print("The difference between these sets (in cm) is: %s" %l_difference_print)
            # print("The items left in stock are: %s" %library_instock)
            print('\n')

            #FIND WHERE L_DIFFERENCE IS LARGER THAN P% OF ORIGINAL ELEMENT LENGTH
```

```
            # p = 0.6 # number representing percentage (0-1)

            p_difference = []
            for i in range(len(sample_set)):
                p_difference.append(l_difference[i]/sample_set[i])

            p_difference_print = [round(n, 2) for n in p_difference]
            print("The difference between these sets (in percentage) is: %s"
%p_difference_print)
            print('\n')

            # FIND N LARGEST DIFFERENCES (ALTERNATIVE TO PREVIOUS SECTION)
            n = 5

            largest_dif_inx = sorted(range(len(l_difference)), key = lambda sub:
l_difference[sub])[-n:]

            largest_dif = []
            for i in largest_dif_inx:
                largest_dif.append(sample_set[i])

            p_dif_large_index = sorted(range(len(p_difference)), key=lambda sub:
p_difference[sub])[-n:]
            sample_set_index = list(range(len(sample_set)))

            print("The items considered for replacement are: %s" %p_dif_large_index)
            print("The number of these items is: %s" % len(p_dif_large_index))
            print('\n')

            # FIND SMALLEST ITEM FROM THESE
            items_p_dif_large = []

            for i in p_dif_large_index:
                items_p_dif_large.append(sample_set[i])

            items_p_dif_large_sorted = items_p_dif_large.copy()
            items_p_dif_large_sorted.sort()

            smallest_items_p_dif_large = min(items_p_dif_large_sorted)

            items_p_dif_large_print = [round(n, 2) for n in items_p_dif_large]
            smallest_items_p_dif_large_print = round(smallest_items_p_dif_large, 2)
            print("Items with large p_difference: %s" %items_p_dif_large_print)
            print("Smallest item within large p_difference: %s"
%smallest_items_p_dif_large_print)

            # FIND LARGEST LIBRARY ITEM IN STOCK
            largest_item_library_instock = max(library_instock)
            print("Largest item within library items left in stock: %s"
%largest_item_library_instock)
            print('\n')

            # FIND HOW MANY TIMES SMALLEST STRUCTURE ITEM FITS IN LARGEST LIBRARY ITEM ->
FACTOR C
            c = round(largest_item_library_instock/smallest_items_p_dif_large)

            if len(items_p_dif_large) < c:
                c = len(items_p_dif_large)
```

```
            print("Factor c (number of times the smallest structure item fits in largest
library item): %s" %c)
            print('\n')

            # #ADD UP ITEMS IN ALL WAYS POSSIBLE (2 TO C) #UNUSED!
            # # So, for list [0, 1, 2]:
            # # [0,1], [0,2], [0, 1, 2], [1,2]
            # items_left = items_p_dif_large
            # combined_lenghts = []
            # combination_index = []
            #
            # for i in range(len(items_left) - 1):
            #     for j in range(1, c-1):
            #         temp = [items_left[i]]
            #         temp_index = [i]
            #         for k in range(1, c):
            #             if i < len(items_left)-j and i+k < len(items_left):
            #                 temp.append(items_left[i+k])
            #                 temp_index.append(i+k)
            #             else:
            #                 temp = []
            #             if len(temp) != 0:
            #                 print("temp list is: %s" % temp)
            #                 print("temp index list is: %s" % temp_index)
            #                 temp_index_tuple = tuple(temp_index)
            #                 combination_index.append(temp_index_tuple)
            #                 combined_lenghts.append(sum(temp))
            #
            # print("List of combined lenghts are: %s" %combined_lenghts)
            # print("List of combined element indices are: %s" %combination_index)
            # print('\n')

            #ADD UP ITEMS IN ALL WAYS POSSIBLE (2 TO C):
            # So, for list [0, 1, 2]:
            # [0,1], [0,2], [0, 1, 2], [1,2]

            items_left = items_p_dif_large
            combination_lengths_tuples = []
            combined_lengths = []
            combination_index_tuples = []

            for i in range(2, c + 1):
                combination_lengths_tuples += list(combinations(items_left, i))
                combination_index_tuples +=
list(combinations(list(range(len(items_left))), i))

            combination_lengths = []
            combination_index = []

            for i in range(len(combination_lengths_tuples)):
                combination_lengths.append(list(combination_lengths_tuples[i]))
                combination_index.append(list(combination_index_tuples[i]))
                combined_lengths.append(sum(list(combination_lengths[i])))

            combination_lengths_print = [[round(n, 2) for n in nested] for nested in
combination_lengths]
            combined_lengths_print = [round(n, 2) for n in combined_lengths]
```

```
            print("List of combined element indices are: %s" %combination_index)
            print("List of lengths to combine are: %s" %combination_lengths_print)
            print("List of combined lengths are: %s" %combined_lengths_print)
            print('\n')


            #FIND CLOSEST MATCHING LENGTHS COMPARED TO AVAILABLE STOCK LENGTHS
            l_difference_new = []
            library_closest_new = []
            print("The items left in stock are: %s" %library_instock)


            for i in combined_lengths:
                library_array_new = np.asarray(library_instock)
                closest_idx_new = (np.abs(library_array_new - i)).argmin()
                # print(closest_idx)
                library_closest_new.append(library_instock[closest_idx_new])
                l_difference_new.append(abs(i - library_instock[closest_idx_new]))

            print("The original set of reconsidered mesh element lengths: %s" %
combined_lengths_print)
            library_closest_new_print = [round(n, 2) for n in library_closest_new]
            l_difference_new_print = [round(n, 2) for n in l_difference_new]
            print("Set of the closest matching items from library: %s"
%library_closest_new_print)
            print("The difference between these sets (in cm) is: %s"
%l_difference_new_print)
            print('\n')

            #RE-CALCULATE L_DIFFERENCE & REPLACE ITEMS
            combination_original_index = []

            for i in range(len(combination_index)):
                temp = []
                for j in range(len(combination_index[i])):
                    temp.append(p_dif_large_index[combination_index[i][j]])
                combination_original_index.append(temp)

            p_dif_large_index.sort()

            print("The original indices present in the combinations are: %s" %
combination_original_index)



            # FIND COMBINATIONS THAT WILL RECREATE THE ORIGINAL SEQUENCE
            # So, for list [0, 1, 2]:
            # possible combinations are: [0,1], [0,2], [0, 1, 2], [1,2]
            # original elements are: [0], [1], [2]
            # return: [0]+[1]+[2], [0,1] + [2], [0,2]+[1], [1,2]+[0], [0,1,2]

            p_dif_large_index_nestedlist = []
            l_difference_for_p_dif_large = []
            for i in p_dif_large_index:
                p_dif_large_index_nestedlist.append([i])
                l_difference_for_p_dif_large.append(l_difference[i])

            segment_options = p_dif_large_index_nestedlist + combination_original_index
```

```
            l_difference_segmented_options = l_difference_for_p_dif_large +
l_difference_new
            print("List of the segment options to combine: %s" % segment_options)
            l_difference_segmented_options_print = [round(n, 2) for n in
l_difference_segmented_options]
            print("List of l_differences for each option: %s" %
l_difference_segmented_options_print)
            print('\n')

            goal_option = p_dif_large_index
            print("The goal combination is: %s" % goal_option)

            segmented_combination_options_tuples = []
            segmented_combination_options = []
            segmented_combination_options_index_tuples = []
            segmented_combination_options_index = []

            for i in range(1, len(goal_option)+1):
                segmented_combination_options_tuples +=
list(combinations(segment_options, i))
                segmented_combination_options_index_tuples +=
list(combinations(list(range(len(segment_options))), i))

            for i in range(len(segmented_combination_options_tuples)):

segmented_combination_options.append([*segmented_combination_options_tuples[i]])

segmented_combination_options_index.append([*segmented_combination_options_index_tuples[i
]])

            # print("Possible combinations of segments: %s" %
segmented_combination_options)
            # print("Indices of these combinations are: %s" %
segmented_combination_options_index)

            goal_option_index = []
            for i in range(len(segmented_combination_options)):
                temp = []
                for j in range(len(segmented_combination_options[i])):
                    for k in range(len(segmented_combination_options[i][j])):
                        temp.append(segmented_combination_options[i][j][k])
                temp.sort()
                if temp == goal_option:
                    # print("The goal combination was found for i = %s" % (str(i)))
                    goal_option_index.append(i)

            print("The goal combination was found for i = %s" % (goal_option_index))
            print('\n')
            print("Indices in the segmented_combination_options list that match the
goal_option are: %s" %goal_option_index)

            final_segment_combination_options = []
            final_segment_combination_options_index = []
            for i in goal_option_index:

final_segment_combination_options.append(segmented_combination_options[i])

final_segment_combination_options_index.append(segmented_combination_options_index[i])
```

```
            print("Final valid segment combination options: %s"
%final_segment_combination_options)
            print("Final valid segment combination options indices: %s"
%final_segment_combination_options_index)

            l_difference_per_final_segment_combination_option = []
            l_difference_sum_per_final_segment_combination_option = []
            for i in range(len(final_segment_combination_options_index)):
                temp = []
                for j in range(len(final_segment_combination_options_index[i])):

temp.append(l_difference_segmented_options[final_segment_combination_options_index[i][j]]
)
                l_difference_per_final_segment_combination_option.append(temp)
                l_difference_sum_per_final_segment_combination_option.append(sum(temp))

            print('\n')
            l_difference_sum_per_final_segment_combination_option_print = [round(n, 2)
for n in l_difference_sum_per_final_segment_combination_option]
            print("List of total l_differences per combination: %s"
%l_difference_sum_per_final_segment_combination_option_print)

            # FIND BEST FITTING COMBINATION
            min_l_difference_option =
np.min(l_difference_sum_per_final_segment_combination_option)
            min_l_difference_option_index =
np.where(l_difference_sum_per_final_segment_combination_option ==
min_l_difference_option)[0][0]

            min_l_difference_option_print = round(min_l_difference_option, 2)
            print("Smallest l_difference from options: %s" %
min_l_difference_option_print)
            print("Index of this option: %s" % min_l_difference_option_index)

            min_option = final_segment_combination_options[min_l_difference_option_index]
            print("Option with smallest l_difference: %s" %min_option)
            print('\n')

            # REMOVE L_DIFFERENCE VALUES FOR REASSIGNED ITEMS
            l_difference_values_to_delete = []
            for i in sorted(p_dif_large_index, reverse=True):
                l_difference_values_to_delete.append(l_difference[i])
                del l_difference[i]

            l_difference_values_to_delete_print = [round(n, 2) for n in
l_difference_values_to_delete]
            print("L_difference for reassighned items (to delete): %s" %
l_difference_values_to_delete_print)
            # print("L_difference after removing values for reassigned items: %s" %
l_difference)

            # ADD L_DIFFERENCE OF NEWLY FOUND COMBINATION
            l_difference.append(min_l_difference_option)
            print("L_difference for reassighned items (to add): %s"
%min_l_difference_option_print)
            l_difference_print = [round(n, 2) for n in l_difference]
            print("L_difference list after adding values for chosen segment combination:
```

```
%s" %l_difference_print)
            print('\n' * 2)

            # CALCULATE TOTAL L_DIFFERENCE &  TOTAL ELEMENT LENGTHS
            print("Final data for mesh %s is:" %mesh)

            l_difference_total = sum(l_difference)
            print("Total length difference: %s" %round(l_difference_total, 2))
            l_element_total = sum(sample_set)
            print("Total element length: %s" %round(l_element_total, 2))

            # CALCULATE L_DIFFERENCE_TOTAL/L_ELEMENT_TOTAL
            p_similarity = (l_element_total + l_difference_total)/l_element_total
            print("Similarity score is: %s" %round(p_similarity, 2))
            p_similarity_array = np.zeros((1,1))
            p_similarity_array[0][0] = p_similarity

            # SAVE SIMILARITY SCORE

np.savetxt("C:/Users/amyst/Desktop/dataset/performance_data/test_files/mesh%s_similarity.
csv" %mesh,
            p_similarity_array, fmt=b"%f", delimiter=",")

            print('\n' * 6)

    else:
        print("Mesh %s is invalid." % mesh)
        print('\n')

np.savetxt("C:/Users/amyst/Desktop/dataset/supporting_files/valid_indices_similarity.csv"
,
           valid_meshes, fmt='%s',
           delimiter=",")
```

Similarity Assessment – Example of Similarity Calculation Report
Results of type 'float' were rounded for reporting purposes, full values are used in calculation.

```
STOCK SIMILARITY CALCULATION FOR MESH 234:
The original set of mesh element lengths: [0.67, 0.79, 0.82, 1.15, 1.17, 1.23, 1.24,
1.27, 1.27, 1.28, 1.29, 1.33, 1.33, 1.33, 1.33, 1.33, 1.33, 1.34, 1.35, 1.36, 1.37, 1.37,
1.38, 1.4, 1.41, 1.46, 1.49, 1.49, 1.49, 1.53, 1.6, 1.6, 1.62, 1.63, 1.65, 1.66, 1.67,
1.67, 1.67, 1.68, 1.69, 1.7, 1.7, 1.71, 1.74, 1.77, 1.77, 1.78, 1.8, 1.8, 1.8, 1.81,
1.84, 1.85, 1.88, 1.89, 1.89, 1.89, 1.89, 1.91, 1.93, 1.94, 1.95, 1.95, 1.96, 1.97, 1.98,
2.0, 2.0, 2.0, 2.02, 2.03, 2.04, 2.06, 2.1, 2.11, 2.11, 2.12, 2.13, 2.13, 2.13, 2.15,
2.15, 2.16, 2.18, 2.19, 2.23, 2.24, 2.26, 2.29, 2.32, 2.33, 2.33, 2.33, 2.36, 2.39, 2.4,
2.43, 2.43, 2.45, 2.46, 2.48, 2.51, 2.51, 2.51, 2.53, 2.61, 2.62, 2.64, 2.66, 2.67, 2.67,
2.67, 2.69, 2.69, 2.69, 2.71, 2.75, 2.75, 2.85, 3.0, 3.05, 3.09, 3.26, 3.29, 3.33, 3.35,
3.59]
Set of the closest matching items from library: [0.8, 0.8, 0.8, 1.17, 1.17, 1.2, 1.2,
1.2, 1.2, 1.36, 1.36, 1.36, 1.36, 1.36, 1.36, 1.36, 1.36, 1.36, 1.36, 1.36, 1.36, 1.36,
1.36, 1.36, 1.36, 1.36, 1.6, 1.6, 1.6, 1.6, 1.6, 1.6, 1.63, 1.63, 1.63, 1.63, 1.63, 1.63,
1.63, 1.63, 1.63, 1.63, 1.63, 1.63, 1.63, 1.63, 1.63, 1.63, 1.63, 1.6, 1.6, 1.6,
1.36, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.2, 1.17, 1.17,
1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17,
1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17, 1.17,
1.17, 1.17, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8,
4.68, 4.68, 4.68, 4.68, 4.68, 4.68, 4.68, 4.68, 4.68, 4.68, 4.68]
The difference between these sets (in cm) is: [0.13, 0.01, 0.02, 0.02, 0.0, 0.03, 0.04,
0.07, 0.07, 0.08, 0.07, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.02, 0.01, 0.0, 0.01, 0.01,
0.02, 0.04, 0.05, 0.1, 0.11, 0.11, 0.11, 0.07, 0.0, 0.0, 0.01, 0.0, 0.02, 0.03, 0.04,
0.04, 0.04, 0.06, 0.06, 0.07, 0.07, 0.09, 0.11, 0.14, 0.14, 0.15, 0.17, 0.17, 0.2, 0.21,
0.24, 0.49, 0.68, 0.69, 0.69, 0.69, 0.69, 0.71, 0.73, 0.74, 0.75, 0.75, 0.76, 0.77, 0.78,
0.8, 0.83, 0.83, 0.85, 0.86, 0.87, 0.89, 0.93, 0.94, 0.94, 0.95, 0.96, 0.96, 0.96, 0.98,
0.98, 0.99, 1.01, 1.02, 1.06, 1.07, 1.08, 1.12, 1.15, 1.16, 1.16, 1.16, 1.19, 1.22, 1.23,
1.26, 1.26, 1.28, 1.29, 1.31, 1.71, 1.71, 1.71, 1.73, 1.81, 1.82, 1.84, 1.86, 1.87, 1.87,
1.87, 1.89, 1.89, 1.89, 1.91, 1.93, 1.93, 1.83, 1.68, 1.63, 1.6, 1.42, 1.39, 1.35, 1.33,
1.09]


The difference between these sets (in percentage) is: [0.2, 0.01, 0.02, 0.02, 0.0, 0.03,
0.03, 0.06, 0.06, 0.06, 0.06, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.01, 0.0, 0.01,
0.01, 0.02, 0.03, 0.03, 0.07, 0.07, 0.07, 0.07, 0.04, 0.0, 0.0, 0.0, 0.0, 0.01, 0.02,
0.02, 0.02, 0.02, 0.03, 0.03, 0.04, 0.04, 0.05, 0.07, 0.08, 0.08, 0.09, 0.09, 0.1, 0.11,
0.12, 0.13, 0.26, 0.36, 0.36, 0.36, 0.37, 0.37, 0.37, 0.38, 0.38, 0.39, 0.39, 0.39, 0.39,
0.39, 0.4, 0.41, 0.41, 0.42, 0.42, 0.43, 0.43, 0.44, 0.44, 0.44, 0.45, 0.45, 0.45, 0.45,
0.45, 0.46, 0.46, 0.46, 0.46, 0.48, 0.48, 0.48, 0.49, 0.5, 0.5, 0.5, 0.5, 0.5, 0.51,
0.51, 0.52, 0.52, 0.52, 0.52, 0.53, 0.68, 0.68, 0.68, 0.68, 0.69, 0.69, 0.7, 0.7, 0.7,
0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.64, 0.56, 0.53, 0.52, 0.44, 0.42, 0.4, 0.4,
0.3]


The items considered for replacement are: [114, 115, 118, 117, 116]
The number of these items is: 5


Items with large p_difference: [2.69, 2.69, 2.75, 2.75, 2.71]
Smallest item within large p_difference: 2.69
Largest item within library items left in stock: 8.8


Factor c (number of times the smallest structure item fits in largest library item): 3
```

List of combined element indices are: [[0, 1], [0, 2], [0, 3], [0, 4], [1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4], [0, 1, 2], [0, 1, 3], [0, 1, 4], [0, 2, 3], [0, 2, 4], [0, 3, 4], [1, 2, 3], [1, 2, 4], [1, 3, 4], [2, 3, 4]]
List of lengths to combine are: [[2.69, 2.69], [2.69, 2.75], [2.69, 2.75], [2.69, 2.71], [2.69, 2.75], [2.69, 2.75], [2.69, 2.71], [2.75, 2.75], [2.75, 2.71], [2.75, 2.71], [2.69, 2.69, 2.75], [2.69, 2.69, 2.75], [2.69, 2.69, 2.71], [2.69, 2.75, 2.75], [2.69, 2.75, 2.71], [2.69, 2.75, 2.71], [2.69, 2.75, 2.75], [2.69, 2.75, 2.71], [2.69, 2.75, 2.71], [2.75, 2.75, 2.71]]
List of combined lengths are: [5.38, 5.44, 5.44, 5.4, 5.44, 5.44, 5.4, 5.5, 5.46, 5.46, 8.13, 8.13, 8.09, 8.18, 8.14, 8.14, 8.19, 8.15, 8.15, 8.2]

The items left in stock are: [0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 4.68188, 4.68188, 4.68188, 4.68188, 4.68188, 4.68188, 4.68188, 8.8, 8.8, 8.8, 8.8, 8.8, 8.8, 8.8, 8.8, 8.8]
The original set of reconsidered mesh element lengths: [5.38, 5.44, 5.44, 5.4, 5.44, 5.44, 5.4, 5.5, 5.46, 5.46, 8.13, 8.13, 8.09, 8.18, 8.14, 8.14, 8.19, 8.15, 8.15, 8.2]
Set of the closest matching items from library: [4.68, 4.68, 4.68, 4.68, 4.68, 4.68, 4.68, 4.68, 4.68, 4.68, 4.68, 8.8, 8.8, 8.8, 8.8, 8.8, 8.8, 8.8, 8.8, 8.8, 8.8, 8.8]
The difference between these sets (in cm) is: [0.7, 0.75, 0.75, 0.71, 0.76, 0.76, 0.72, 0.81, 0.77, 0.77, 0.67, 0.67, 0.71, 0.62, 0.66, 0.66, 0.61, 0.65, 0.65, 0.6]

The original indices present in the combinations are: [[114, 115], [114, 118], [114, 117], [114, 116], [115, 118], [115, 117], [115, 116], [118, 117], [118, 116], [117, 116], [114, 115, 118], [114, 115, 117], [114, 115, 116], [114, 118, 117], [114, 118, 116], [114, 117, 116], [115, 118, 117], [115, 118, 116], [115, 117, 116], [118, 117, 116]]
List of the segment options to combine: [[114], [115], [116], [117], [118], [114, 115], [114, 118], [114, 117], [114, 116], [115, 118], [115, 117], [115, 116], [118, 117], [118, 116], [117, 116], [114, 115, 118], [114, 115, 117], [114, 115, 116], [114, 118, 117], [114, 118, 116], [114, 117, 116], [115, 118, 117], [115, 118, 116], [115, 117, 116], [118, 117, 116]]

List of l_differences for each option: [1.89, 1.89, 1.91, 1.93, 1.93, 0.7, 0.75, 0.75, 0.71, 0.76, 0.76, 0.72, 0.81, 0.77, 0.77, 0.67, 0.67, 0.71, 0.62, 0.66, 0.66, 0.61, 0.65, 0.65, 0.6]

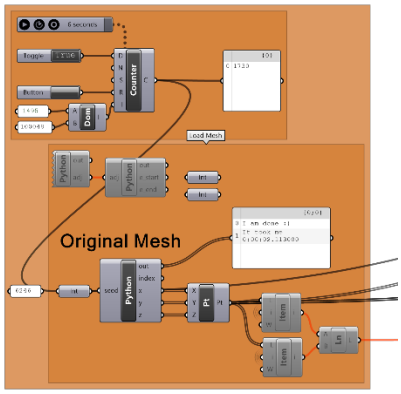
The goal combination is: [114, 115, 116, 117, 118]
The goal combination was found for i = [153, 170, 186, 201, 215, 228, 240, 251, 261, 270, 347, 366, 388, 409, 485, 498, 510, 616, 638, 659, 690, 706, 721, 865, 886, 901, 917, 933, 1097, 1112, 1128, 1159, 1303, 1335, 1350, 2634, 2656, 2677, 2883, 2904, 3115, 4651, 4672, 4883, 6420, 15275]

Indices in the segmented_combination_options list that match the goal_option are: [153, 170, 186, 201, 215, 228, 240, 251, 261, 270, 347, 366, 388, 409, 485, 498, 510, 616, 638, 659, 690, 706, 721, 865, 886, 901, 917, 933, 1097, 1112, 1128, 1159, 1303, 1335, 1350, 2634, 2656, 2677, 2883, 2904, 3115, 4651, 4672, 4883, 6420, 15275]
Final valid segment combination options: [[[114, 115], [118, 117, 116]], [[114, 118], [115, 117, 116]], [[114, 117], [115, 118, 116]], [[114, 116], [115, 118, 117]], [[115, 118], [114, 117, 116]], [[115, 117], [114, 118, 116]], [[115, 116], [114, 118, 117]], [[118, 117], [114, 115, 116]], [[118, 116], [114, 115, 117]], [[117, 116], [114, 115, 118]], [[114], [115], [118, 117, 116]], [[114], [116], [115, 118, 117]], [[114], [117], [115, 118, 116]], [[114], [118], [115, 117, 116]], [[114], [115, 118], [117, 116]], [[114], [115, 117], [118, 116]], [[114], [115, 116], [118, 117]], [[115], [116], [114, 118, 117]], [[115], [117], [114, 118, 116]], [[115], [118], [114, 117, 116]], [[115],

```
[114, 118], [117, 116]], [[115], [114, 117], [118, 116]], [[115], [114, 116], [118,
117]], [[116], [117], [114, 115, 118]], [[116], [118], [114, 115, 117]], [[116], [114,
115], [118, 117]], [[116], [114, 118], [115, 117]], [[116], [114, 117], [115, 118]],
[[117], [118], [114, 115, 116]], [[117], [114, 115], [118, 116]], [[117], [114, 118],
[115, 116]], [[117], [114, 116], [115, 118]], [[118], [114, 115], [117, 116]], [[118],
[114, 117], [115, 116]], [[118], [114, 116], [115, 117]], [[114], [115], [116], [118,
117]], [[114], [115], [117], [118, 116]], [[114], [115], [118], [117, 116]], [[114],
[116], [117], [115, 118]], [[114], [116], [118], [115, 117]], [[114], [117], [118], [115,
116]], [[115], [116], [117], [114, 118]], [[115], [116], [118], [114, 117]], [[115],
[117], [118], [114, 116]], [[116], [117], [118], [114, 115]], [[114], [115], [116],
[117], [118]]]

Final valid segment combination options indices: [[5, 24], [6, 23], [7, 22], [8, 21], [9,
20], [10, 19], [11, 18], [12, 17], [13, 16], [14, 15], [0, 1, 24], [0, 2, 21], [0, 3,
22], [0, 4, 23], [0, 9, 14], [0, 10, 13], [0, 11, 12], [1, 2, 18], [1, 3, 19], [1, 4,
20], [1, 6, 14], [1, 7, 13], [1, 8, 12], [2, 3, 15], [2, 4, 16], [2, 5, 12], [2, 6, 10],
[2, 7, 9], [3, 4, 17], [3, 5, 13], [3, 6, 11], [3, 8, 9], [4, 5, 14], [4, 7, 11], [4, 8,
10], [0, 1, 2, 12], [0, 1, 3, 13], [0, 1, 4, 14], [0, 2, 3, 9], [0, 2, 4, 10], [0, 3, 4,
11], [1, 2, 3, 6], [1, 2, 4, 7], [1, 3, 4, 8], [2, 3, 4, 5], [0, 1, 2, 3, 4]]


List of total l_differences per combination: [1.3, 1.4, 1.4, 1.32, 1.42, 1.42, 1.34,
1.53, 1.44, 1.44, 4.38, 4.4, 4.47, 4.47, 3.42, 3.42, 3.42, 4.42, 4.48, 4.48, 3.42, 3.42,
3.42, 4.51, 4.51, 3.42, 3.42, 3.42, 4.58, 3.41, 3.41, 3.41, 3.41, 3.41, 3.41, 6.5, 6.49,
6.49, 6.49, 6.49, 6.47, 6.49, 6.49, 6.47, 6.47, 9.56]
Smallest l_difference from options: 1.3
Index of this option: 0
Option with smallest l_difference: [[114, 115], [118, 117, 116]]
L_difference for reassighned items (to delete): [1.93, 1.93, 1.91, 1.89, 1.89]
L_difference for reassighned items (to add): 1.3

L_difference list after adding values for chosen segment combination: [0.13, 0.01, 0.02,
0.02, 0.0, 0.03, 0.04, 0.07, 0.07, 0.08, 0.07, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.02,
0.01, 0.0, 0.01, 0.01, 0.02, 0.04, 0.05, 0.1, 0.11, 0.11, 0.11, 0.07, 0.0, 0.0, 0.01,
0.0, 0.02, 0.03, 0.04, 0.04, 0.04, 0.06, 0.06, 0.07, 0.07, 0.09, 0.11, 0.14, 0.14, 0.15,
0.17, 0.17, 0.2, 0.21, 0.24, 0.49, 0.68, 0.69, 0.69, 0.69, 0.69, 0.71, 0.73, 0.74, 0.75,
0.75, 0.76, 0.77, 0.78, 0.8, 0.83, 0.83, 0.85, 0.86, 0.87, 0.89, 0.93, 0.94, 0.94, 0.95,
0.96, 0.96, 0.96, 0.98, 0.98, 0.99, 1.01, 1.02, 1.06, 1.07, 1.08, 1.12, 1.15, 1.16, 1.16,
1.16, 1.19, 1.22, 1.23, 1.26, 1.26, 1.28, 1.29, 1.31, 1.71, 1.71, 1.71, 1.73, 1.81, 1.82,
1.84, 1.86, 1.87, 1.87, 1.87, 1.89, 1.83, 1.68, 1.63, 1.6, 1.42, 1.39, 1.35, 1.33, 1.09,
1.3]



Final data for mesh 234 is:
Total length difference: 86.26
Total element length: 256.1
Similarity score is: 1.34
```

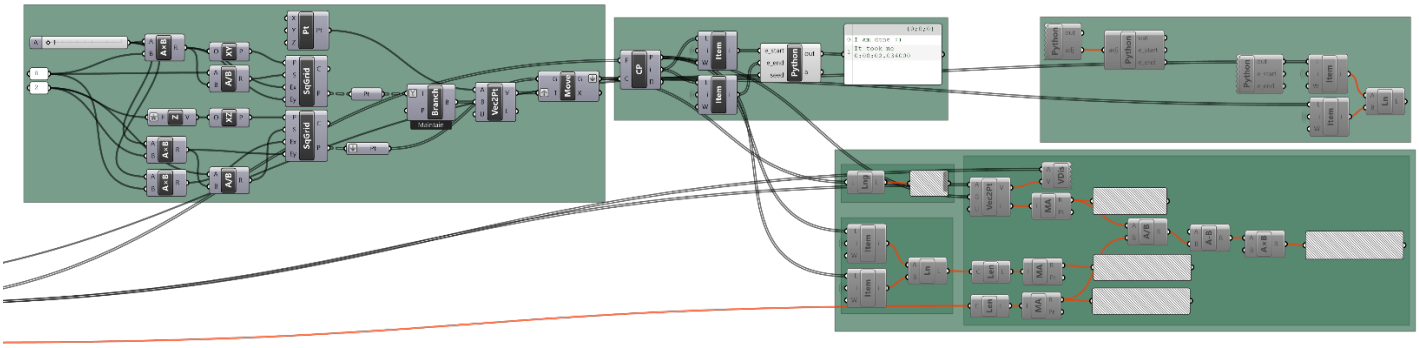
Create Adjacency Matrices based on Fine Mesh – Grasshopper:

Load meshes



Fine Mesh

Recreated Mesh



Create Adjacency Matrices based on Fine Mesh – GHPython:

```
import rhinoscriptsyntax as rs
import csv
import datetime as dt
import scriptcontext
np = scriptcontext.sticky['numpy']

t_start = dt.datetime.now()

#GET ADJACENCY MATRIX
adj = np.zeros((243, 243))

for i in range(len(e_start)):
    adj[e_start[i]][e_end[i]] = 1
    adj[e_end[i]][e_start[i]] = 1

np.savetxt("C:/Users/amyst/Desktop/dataset/adjmatrix/AdjMatrix243-Mesh%s.csv" %seed, adj,
fmt=b"%i", delimiter=",")

t_end = dt.datetime.now()

print("I am done :)")
print("It took me %s" %(t_end-t_start))
```

50 Samples from Performance Score Dataset

Index	Seed	Performance Indicators					
		Material Use	Stock Similarity	Displacement	Utilization	Structural	**Overall**
1926	**1979**	0.33359	0.21518	0.6951	0.52282	0.62619	**0.40327**
6116	**6324**	0.40105	0.29458	0.60139	0.47153	0.54945	**0.41782**
2337	**2402**	0.41286	0.31088	0.69733	0.30027	0.53851	**0.42233**
1605	**1651**	0.36267	0.172	0.7456	0.68609	0.72179	**0.43005**
348	**357**	0.35557	0.15748	0.84381	0.66673	0.77298	**0.44329**
709	**733**	0.24001	0.3222	0.75277	0.59799	0.69086	**0.45323**
1690	**1736**	0.41501	0.14279	0.83968	0.74445	0.80159	**0.46075**
1397	**1439**	0.45585	0.48556	0.52887	0.43878	0.49283	**0.48253**
6704	**6943**	0.4376	0.20873	0.82058	0.72224	0.78124	**0.48351**
2868	**2956**	0.42291	0.23098	0.84627	0.73895	0.80334	**0.49831**
3234	**3335**	0.34876	0.24237	0.90445	0.73347	0.83606	**0.50112**
7093	**7348**	0.43588	0.24326	0.86328	0.70748	0.80096	**0.50487**
5450	**5633**	0.47169	0.23984	0.86255	0.69685	0.79627	**0.50878**
9392	**9734**	0.19608	0.34767	0.88593	0.77178	0.84027	**0.51439**
5005	**5175**	0.4919	0.27122	0.8464	0.67421	0.77752	**0.51788**
3530	**3639**	0.38483	0.309	0.83211	0.77403	0.80888	**0.52411**
2072	**2129**	0.40485	0.23372	0.89826	0.85854	0.88237	**0.52741**
3730	**3853**	0.43942	0.31155	0.82805	0.75746	0.79981	**0.53243**
256	**262**	0.44485	0.4061	0.76574	0.61542	0.70561	**0.53366**
1652	**1698**	0.36472	0.27461	0.90176	0.84965	0.88092	**0.53516**
4013	**4149**	0.54178	0.32465	0.81343	0.71127	0.77256	**0.54724**
8721	**9040**	0.3763	0.31998	0.92447	0.7986	0.87412	**0.5529**
2543	**2618**	0.59299	0.43423	0.75099	0.5206	0.65883	**0.55582**
4188	**4330**	0.41424	0.42118	0.85357	0.62835	0.76348	**0.55671**
2057	**2114**	0.46041	0.39821	0.82867	0.67419	0.76688	**0.55812**
1071	**1103**	0.56569	0.4023	0.8205	0.5539	0.71386	**0.5596**
5177	**5353**	0.4868	0.37945	0.82225	0.72331	0.78268	**0.56221**
3995	**4130**	0.58671	0.36227	0.84046	0.65365	0.76574	**0.56854**
7919	**8198**	0.6775	0.49201	0.63637	0.52649	0.59242	**0.56927**
822	**846**	0.62856	0.4126	0.76863	0.59077	0.69749	**0.56975**
3108	**3202**	0.48376	0.37597	0.86377	0.74133	0.81479	**0.57306**
3517	**3626**	0.56023	0.31693	0.89562	0.75312	0.83862	**0.57427**
8437	**8741**	0.60714	0.45548	0.7638	0.57522	0.68837	**0.57897**
1875	**1926**	0.55483	0.44801	0.82668	0.58271	0.7291	**0.58181**
838	**862**	0.75803	0.60845	0.54981	0.39133	0.48642	**0.58955**
4519	**4673**	0.46012	0.46422	0.84349	0.6884	0.78146	**0.59029**
8979	**9307**	0.46693	0.37582	0.93144	0.81493	0.88483	**0.59765**
7338	**7603**	0.65664	0.63098	0.62416	0.43217	0.54736	**0.60267**
5615	**5805**	0.58172	0.4998	0.79439	0.6049	0.7186	**0.6037**
1042	**1073**	0.62426	0.39927	0.83672	0.75756	0.80506	**0.60658**
9012	**9340**	0.61475	0.61775	0.67402	0.50596	0.6068	**0.61277**

5199	**5376**	0.69135	0.50978	0.80388	0.5212	0.6908	**0.6185**
107	**110**	0.51078	0.37633	0.95817	0.89963	0.93475	**0.62659**
3059	**3153**	0.68909	0.54444	0.7787	0.57025	0.69532	**0.63372**
8055	**8338**	0.5522	0.5286	0.85331	0.77712	0.82283	**0.65101**
9699	**10045**	0.7471	0.5413	0.76931	0.64479	0.7195	**0.65374**
4180	**4322**	0.57016	0.58278	0.87009	0.69103	0.79846	**0.66653**
2538	**2613**	0.78412	0.65061	0.75455	0.52157	0.66136	**0.68161**
7019	**7272**	0.63401	0.58459	0.91872	0.77738	0.86218	**0.70551**
1701	**1747**	0.8369	0.83762	0.65973	0.4615	0.58044	**0.7346**

Appendix III

AI Framework

AI Framework (Surrogate Model, VAE & Gradient Descent)

Parts of this code are inspired by and/or retrieved from: Pavlidou, S. (2022). Deep Generative Designs: A Deep Learning Framework for Optimized Shell Structures.

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, activations
import random
import matplotlib.style
import matplotlib.pyplot as plt
import os

# ID
attempt = 39
value = 'p_overall'
train_size = 5000
job_directory = os.getcwd()
save_location = os.path.join(job_directory, "total_%s" % attempt)

# BEST PERFORMING MESHES (TO REMOVE)
best = [7631, 1665, 5814, 1692, 9044, 4621, 8710, 6341, 2518, 6252]
test = [9012, 2337, 838, 3730, 6116, 8979, 3517, 348, 7093, 9699, 107, 1690, 1071, 4188,
2538, 7919, 1926, 5450, 1701,
        5615, 1605, 8055, 6704, 256, 3059, 1875, 9392, 3108, 1652, 7019, 4519, 3234,
1042, 2072, 709, 7338, 8721, 4013,
        5177, 2868, 3995, 822, 4180, 8437, 5005, 2057, 5199, 1397, 2543, 3530]

# LOAD PERFORMANCE INDICATOR DATASET
column_names = ["index", "seed", "p_material_use", "p_stock_similarity",
"p_displacement", "p_utilization",
                "p_structural", "p_structural_material_use", "p_overall", "material_use",
"displacement", "utilization"]
dataset_path = "dataset/dataset_unaugmented-extended2-fix.csv"
loc_dataset = os.path.join(job_directory, dataset_path)
dataset = pd.read_csv(loc_dataset, names=column_names)

calculated_scores_set_full = dataset[value]
calculated_scores_set = calculated_scores_set_full[:train_size]
dataset = dataset[:train_size]

# LOAD ADJ MATRIX DATASET
loaded_adj_set_path = "dataset/half_adj_matrix_243_set.csv"
loc_adj_set = os.path.join(job_directory, loaded_adj_set_path)
loaded_adj_set = np.loadtxt(loc_adj_set, delimiter=",")
loaded_adj_set = loaded_adj_set[:train_size]

loaded_adj_set_test_path = "dataset/half_adj_matrix_243_set_test.csv"
loc_adj_set_test = os.path.join(job_directory, loaded_adj_set_test_path)
loaded_adj_set_test = np.loadtxt(loc_adj_set_test, delimiter=",")

# Y_TRAIN (PERFORMANCE DATA - LABELS)
train_labels = dataset[value]
train_labels_np = np.array(train_labels)
train_labels_np = train_labels_np.astype('float32')
train_labels_np = train_labels_np.reshape(len(dataset), 1)
y_train = train_labels_np
print("Shape of y_train is: ")
print(y_train.shape)
```

```
# X_TRAIN (GEOMETRY DATA)
train_shape = (train_size, 29403)
input_shape = (29403)
train_data = loaded_adj_set.reshape(train_shape)
x_train = train_data
print("Shape of x_train is: ")
print(x_train.shape)

# ================================================================================================
# SURROGATE MODEL
# ================================================================================================

normalizer = tf.keras.layers.experimental.preprocessing.Normalization(axis=1)


def build_and_compile_model(normalizer):
    model = keras.Sequential([
        keras.Input(shape=(input_shape)),
        layers.Flatten(),
        layers.Dense(256, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(1)
    ])
    return model


print("Starting Training of Surrogate Model!")
sur_model = build_and_compile_model(normalizer)
sur_model.compile(loss=["mse", "mse"], optimizer=tf.keras.optimizers.Adam(0.0001,
clipnorm=1.), metrics=['accuracy'])

history = sur_model.fit(x_train,
                        y_train,
                        validation_split=0.2,
                        verbose=1,
                        epochs=600,
                        batch_size=32
                        )

# Get loss history
loss = history.history['loss']

print("Training Of Surrogate Model is Finished")

# SAVE MODEL
history_path = "history"
model_path = "model/"
loc_model = os.path.join(save_location, model_path)
sur_model.save(loc_model, save_format='HDF5')
loc_history_loss = os.path.join(save_location, history_path, "loss.txt")
loc_history_val_loss = os.path.join(save_location, history_path, "val_loss.txt")
np.savetxt(loc_history_loss, history.history['loss'], delimiter=",")
np.savetxt(loc_history_val_loss, history.history['val_loss'], delimiter=",")

# PREDICT SAMPLES
samples_to_test = loaded_adj_set_test
```

```
sample_shape = (len(test), 29403)
samples_to_test = np.asarray(samples_to_test)
samples_to_test = samples_to_test.reshape(sample_shape)

fullset_to_test = np.asarray(loaded_adj_set)
fullset_shape = (train_size, 29403)
fullset_to_test = fullset_to_test.reshape(fullset_shape)

calculated_scores = np.take(calculated_scores_set_full, test, axis=0)
predicted_scores = []
predicted_scores.append(sur_model.predict(samples_to_test, batch_size=1))

calculated_scores_fullset = calculated_scores_set
predicted_scores_fullset = []
predicted_scores_fullset.append(sur_model.predict(fullset_to_test, batch_size=1))

plot_path = "plots"
loc_calculated = os.path.join(save_location, plot_path, "calculated_attempt%s.txt" %
attempt)
loc_predicted = os.path.join(save_location, plot_path, "predicted_attempt%s.txt" %
attempt)
np.savetxt(loc_calculated, calculated_scores, delimiter=",")
np.savetxt(loc_predicted, predicted_scores[0], delimiter=",")

loc_calculated_fullset = os.path.join(save_location, plot_path,
"calculated_fullset_attempt%s.txt" % attempt)
loc_predicted_fullset = os.path.join(save_location, plot_path,
"predicted_fullset_attempt%s.txt" % attempt)
np.savetxt(loc_calculated_fullset, calculated_scores_fullset, delimiter=",")
np.savetxt(loc_predicted_fullset, predicted_scores_fullset[0], delimiter=",")

# ============================================================================================
# VAE
# ============================================================================================

latent_dim = 13
b = 0.2  # KL-loss weight


# VAE CLASS
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder

    def train_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            # reconstruction =tf.round(self.decoder(z))
            reconstruction = self.decoder(z)
            reconstruction_loss = tf.reduce_mean(keras.losses.binary_crossentropy(data,
reconstruction))
            reconstruction_loss *= 29403

            kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
```

```
            kl_loss = tf.reduce_mean(kl_loss)
            kl_loss *= -b

            total_loss = reconstruction_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))

        return {
            "loss": total_loss,
            "reconstruction_loss": reconstruction_loss,
            "kl_loss": kl_loss,
        }

    def test_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        z_mean, z_log_var, z = self.encoder(data)
        reconstruction = self.decoder(z)
        reconstruction_loss = tf.reduce_mean(keras.losses.binary_crossentropy(data,
reconstruction))
        reconstruction_loss *= 29403

        kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
        kl_loss = tf.reduce_mean(kl_loss)
        kl_loss *= -b
        total_loss = reconstruction_loss + kl_loss

        return {
            "loss": total_loss,
            "reconstruction_loss": reconstruction_loss,
            "kl_loss": kl_loss,
        }


#  SAMPLING LAYER
class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon


# ENCODER
encoder_inputs = keras.Input(shape=(29403))
x = layers.Dense(1089, activation="relu")(encoder_inputs)
x = layers.Dense(363, activation="relu")(x)
x = layers.Dense(121, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
z = Sampling()([z_mean, z_log_var])
encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")

# DECODER
latent_inputs = keras.Input(shape=(latent_dim,))
x = layers.Dense(121, activation="relu")(latent_inputs)
x = layers.Dense(363, activation="relu")(x)
```

```
x = layers.Dense(1089, activation="relu")(x)
decoder_outputs = layers.Dense(29403, activation="sigmoid")(x)
decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
decoder.summary()

# TRAINING
print("Starting Training of Variational Autoencoder!")
vae = VAE(encoder, decoder)
vae.compile(optimizer=keras.optimizers.Adam(), metrics=['accuracy'])
history = vae.fit(train_data, epochs=200, batch_size=64, validation_split=0.2, verbose=2)
print("Training Of Surrogate Model is Finished")

# GET HISTORY
history.history.keys()
history.history['loss']
history_path = "history"
loc_history_loss = os.path.join(save_location, history_path, "loss.txt")
loc_history_val_loss = os.path.join(save_location, history_path, "val_loss.txt")
np.savetxt(loc_history_loss, history.history['loss'], delimiter=",")
np.savetxt(loc_history_val_loss, history.history['val_loss'], delimiter=",")

# SAVE VAE
vae_save_path = "model"
vae_save_loc = os.path.join(save_location, vae_save_path)

encoder.save_weights(vae_save_loc + "VAE_encoder.h5")
decoder.save_weights(vae_save_loc + "VAE_decoder.h5")

samples_to_test = loaded_adj_set_test
sample_shape = (len(test), 29403, 1)
samples_to_test = np.asarray(samples_to_test)
samples_to_test = samples_to_test.reshape(sample_shape)

predicted_scores_encoder = encoder.predict(samples_to_test)
predicted_scores_decoder = decoder.predict(predicted_scores_encoder[2])

predicted_scores_encoder = np.asarray(predicted_scores_encoder[2])
predicted_scores_encoder_shape = (len(test), (latent_dim))
predicted_scores_encoder =
predicted_scores_encoder.reshape(predicted_scores_encoder_shape)
predicted_scores_decoder = np.asarray(predicted_scores_decoder)
predicted_scores_decoder_shape = (len(test) , 29403)
predicted_scores_decoder =
predicted_scores_decoder.reshape(predicted_scores_decoder_shape)

predict_path = "predict"
loc_predicted_encoder = os.path.join(save_location, predict_path,
"predicted_encoder_ID%s.csv" % attempt)
loc_predicted_decoder = os.path.join(save_location, predict_path,
"predicted_decoder_ID%s.csv" % attempt)
np.savetxt(loc_predicted_encoder, predicted_scores_encoder, delimiter=",")
np.savetxt(loc_predicted_decoder, predicted_scores_decoder, delimiter=",")

# ============================================================================================
# GRADIENT DESCENT
# ============================================================================================

gradients = []
```

```
performance = []
generated_meshes = []

input_mesh_ID = 838
learning_rate = 0.5

# input_mesh_path = "input_mesh/input_mesh%s.csv" %input_mesh_ID
# loc_input_mesh = os.path.join(job_directory, input_mesh_path)
# input_mesh = pd.read_csv(loc_input_mesh, names=column_names)

mesh_shape = (1, 29403)

input_mesh = loaded_adj_set[input_mesh_ID]
input_mesh = input_mesh.astype('float32')
input_mesh = input_mesh.reshape(mesh_shape)

x = vae.encoder(input_mesh)
z = x[2]

for i in range(1000):
    with tf.GradientTape() as tape:
        tape.watch(z)
        decoded_tensor = vae.decoder(z)
        decoded_tensor = layers.Reshape((1, 29403))(decoded_tensor)
        y = sur_model(decoded_tensor)
        gradient = tape.gradient(y, z)
    gradients_array = gradient.numpy().reshape(latent_dim)
    rms = np.sqrt(np.mean((gradients_array ** 2)))
    gradients.append(rms)

    y_arr = y.numpy().reshape(1)
    performance.append(y_arr[0])

    decoded_tensor = vae.decoder(z)
    decoded_tensor = tf.round(decoded_tensor)
    decoded_tensor_save = layers.Reshape((1, 29403))(decoded_tensor)

    generated_meshes.append(decoded_tensor_save)
    z = z + (learning_rate * gradient)
    if len(performance) > 100 and (abs(performance[i - 1] - performance[i]) /
performance[i - 1]) * 100 < 0.0001:
        break

generated_meshes = np.asarray(generated_meshes)
mesh_shape = (len(generated_meshes), 29403)
generated_meshes = generated_meshes.reshape(mesh_shape)

# SAVE GENERATED MESHES
generated_meshes_path = "generated_meshes"
loc_generated_meshes = os.path.join(save_location, generated_meshes_path,
"generated_meshes_ID%s.csv" % attempt)
loc_performance = os.path.join(save_location, generated_meshes_path,
"performance_ID%s.csv" % attempt)
loc_gradients = os.path.join(save_location, generated_meshes_path, "gradients_ID%s.csv" %
attempt)
np.savetxt(loc_generated_meshes, generated_meshes, delimiter=",")
np.savetxt(loc_performance, performance, delimiter=",")
np.savetxt(loc_gradients, gradients, delimiter=",")
```

Full VAE-based workflow code with split top and bottom structures:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, activations
import random
import matplotlib.style
import matplotlib.pyplot as plt
import os

# ID
attempt = 91
value = 'p_overall'
train_size = 5000
job_directory = os.getcwd()
save_location = os.path.join(job_directory, "total_%s" % attempt)

adj_size_full = 29403
adj_size_top = 3240
adj_size_bot = 13041

# BEST PERFORMING MESHES
best = [7631, 1665, 5814, 1692, 9044, 4621, 8710, 6341, 2518, 6252]
test = [34, 167, 454, 477, 783, 335, 749, 290, 607, 544]

# LOAD PERFORMANCE INDICATOR DATASET
column_names = ["index", "seed", "p_material_use", "p_stock_similarity",
"p_displacement", "p_utilization",
                "p_structural", "p_structural_material_use", "p_overall", "material_use",
"displacement", "utilization"]
dataset_path = "dataset/dataset_unaugmented-extended2-fix.csv"
loc_dataset = os.path.join(job_directory, dataset_path)
dataset = pd.read_csv(loc_dataset, names=column_names)

calculated_scores_set_full = dataset[value]
calculated_scores_set = calculated_scores_set_full[:train_size]
dataset = dataset[:train_size]

# LOAD ADJ MATRIX DATASET
loaded_adj_set_path = "dataset/AdjMatrix243_full-half_set_Mesh0-4999.csv"
loc_adj_set = os.path.join(job_directory, loaded_adj_set_path)
loaded_adj_set = np.loadtxt(loc_adj_set, delimiter=",")
loaded_adj_set = loaded_adj_set[:train_size]

loaded_adj_set_test_path = "dataset/half_adj_matrix_243_set_test.csv"
loc_adj_set_test = os.path.join(job_directory, loaded_adj_set_test_path)
loaded_adj_set_test = np.loadtxt(loc_adj_set_test, delimiter=",")

# LOAD ADJ MATRIX TOP DATASET
loaded_adj_set_top_path = "dataset/AdjMatrix243_full-top_set_Mesh0-4999.csv"
loc_adj_set_top = os.path.join(job_directory, loaded_adj_set_top_path)
loaded_adj_set_top = np.loadtxt(loc_adj_set_top, delimiter=",")
loaded_adj_set_top = loaded_adj_set_top[:train_size]

loaded_adj_set_top_test_path = "dataset/AdjMatrix243_top-half-set_test.csv"
loc_adj_set_top_test = os.path.join(job_directory, loaded_adj_set_top_test_path)
```

```
loaded_adj_set_top_test = np.loadtxt(loc_adj_set_top_test, delimiter=",")

# LOAD ADJ MATRIX BOT DATASET
loaded_adj_set_bot_path = "dataset/AdjMatrix243_full-bot_set_Mesh0-4999.csv"
loc_adj_set_bot = os.path.join(job_directory, loaded_adj_set_bot_path)
loaded_adj_set_bot = np.loadtxt(loc_adj_set_bot, delimiter=",")
loaded_adj_set_bot = loaded_adj_set_bot[:train_size]

loaded_adj_set_bot_test_path = "dataset/AdjMatrix243_bot-half-set_test.csv"
loc_adj_set_bot_test = os.path.join(job_directory, loaded_adj_set_bot_test_path)
loaded_adj_set_bot_test = np.loadtxt(loc_adj_set_bot_test, delimiter=",")

# Y_TRAIN (PERFORMANCE DATA - LABELS)
train_labels = dataset[value]
train_labels_np = np.array(train_labels)
train_labels_np = train_labels_np.astype('float32')
train_labels_np = train_labels_np.reshape(len(dataset), 1)
y_train = train_labels_np
print("Shape of y_train is: ")
print(y_train.shape)

# X_TRAIN (GEOMETRY DATA - FULL)
train_shape = (train_size, adj_size_full)
input_shape = (adj_size_full)
x_train = loaded_adj_set.reshape(train_shape)
print("Shape of x_train is: ")
print(x_train.shape)

# X_TRAIN (GEOMETRY DATA - TOP)
train_shape = (train_size, adj_size_top)
input_shape_top = (adj_size_top)
x_train_top = loaded_adj_set_top.reshape(train_shape)
print("Shape of x_train is: ")
print(x_train_top.shape)

# X_TRAIN (GEOMETRY DATA - BOT)
train_shape = (train_size, adj_size_bot)
input_shape_bot = (adj_size_bot)
x_train_bot = loaded_adj_set_bot.reshape(train_shape)
print("Shape of x_train is: ")
print(x_train_bot.shape)

#=============================================================================================
# SURROGATE MODEL
#=============================================================================================

normalizer = tf.keras.layers.experimental.preprocessing.Normalization(axis=1)


def build_and_compile_model(normalizer):
    model = keras.Sequential([
        keras.Input(shape=(input_shape)),
        layers.Flatten(),
        layers.Dense(256, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(1)
    ])
```

```
    return model


print("Starting Training of Surrogate Model!")
sur_model = build_and_compile_model(normalizer)
sur_model.compile(loss=["mse", "mse"], optimizer=tf.keras.optimizers.Adam(0.0001,
clipnorm=1.), metrics=['accuracy'])

history = sur_model.fit(x_train,
                        y_train,
                        validation_split=0.2,
                        verbose=1,
                        epochs=600,
                        batch_size=32
                        )

loss = history.history['loss']

print("Training Of Surrogate Model is Finished")

# SAVE MODEL
history_path = "history"
model_path = "model/"
loc_model = os.path.join(save_location, model_path)
sur_model.save(loc_model, save_format='HDF5')
loc_history_loss = os.path.join(save_location, history_path, "loss.txt")
loc_history_val_loss = os.path.join(save_location, history_path, "val_loss.txt")
np.savetxt(loc_history_loss, history.history['loss'], delimiter=",")
np.savetxt(loc_history_val_loss, history.history['val_loss'], delimiter=",")

# PREDICT SAMPLES
fullset_to_test = np.asarray(loaded_adj_set)
fullset_shape = (train_size, adj_size_full)
fullset_to_test = fullset_to_test.reshape(fullset_shape)

calculated_scores_fullset = calculated_scores_set
predicted_scores_fullset = []
predicted_scores_fullset.append(sur_model.predict(fullset_to_test, batch_size=1))

plot_path = "plots"

loc_calculated_fullset = os.path.join(save_location, plot_path,
"calculated_fullset_attempt%s.txt" % attempt)
loc_predicted_fullset = os.path.join(save_location, plot_path,
"predicted_fullset_attempt%s.txt" % attempt)
np.savetxt(loc_calculated_fullset, calculated_scores_fullset, delimiter=",")
np.savetxt(loc_predicted_fullset, predicted_scores_fullset[0], delimiter=",")

# ================================================================================================
# VAE 1: TOP LAYER
# ================================================================================================

latent_dim_top = 13
b_top = 0.5  # KL-loss weight


# VAE CLASS
class VAE_TOP(keras.Model):
```

```
    def __init__(self, encoder_top, decoder_top, **kwargs):
        super(VAE_TOP, self).__init__(**kwargs)
        self.encoder_top = encoder_top
        self.decoder_top = decoder_top

    def train_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder_top(data)
            # reconstruction =tf.round(self.decoder_top(z))
            reconstruction = self.decoder_top(z)
            reconstruction_loss = tf.reduce_mean(keras.losses.binary_crossentropy(data,
reconstruction))
            reconstruction_loss *= adj_size_top

            kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
            kl_loss = tf.reduce_mean(kl_loss)
            kl_loss *= -b_top

            total_loss = reconstruction_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))

        return {
            "loss": total_loss,
            "reconstruction_loss": reconstruction_loss,
            "kl_loss": kl_loss,
        }

    def test_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        z_mean, z_log_var, z = self.encoder_top(data)
        reconstruction = self.decoder_top(z)
        reconstruction_loss = tf.reduce_mean(keras.losses.binary_crossentropy(data,
reconstruction))
        reconstruction_loss *= adj_size_top

        kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
        kl_loss = tf.reduce_mean(kl_loss)
        kl_loss *= -b_top
        total_loss = reconstruction_loss + kl_loss

        return {
            "loss": total_loss,
            "reconstruction_loss": reconstruction_loss,
            "kl_loss": kl_loss,
        }


#  SAMPLING LAYER
class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
```

```
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon


# ENCODER
encoder_inputs = keras.Input(shape=(adj_size_top))
x = layers.Dense(1089, activation="relu")(encoder_inputs)
x = layers.Dense(363, activation="relu")(x)
x = layers.Dense(121, activation="relu")(x)
z_mean = layers.Dense(latent_dim_top, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim_top, name="z_log_var")(x)
z_top = Sampling()([z_mean, z_log_var])
encoder_top = keras.Model(encoder_inputs, [z_mean, z_log_var, z_top], name="encoder_top")

# DECODER
latent_inputs = keras.Input(shape=(latent_dim_top,))
x = layers.Dense(121, activation="relu")(latent_inputs)
x = layers.Dense(363, activation="relu")(x)
x = layers.Dense(1089, activation="relu")(x)
decoder_outputs = layers.Dense(adj_size_top, activation="sigmoid")(x)
decoder_top = keras.Model(latent_inputs, decoder_outputs, name="decoder_top")
decoder_top.summary()

# TRAINING
print("Starting Training of Variational Autoencoder (TOP)!")
vae_top = VAE_TOP(encoder_top, decoder_top)

clip_norm = 1.0
learning_rate = 0.001
adam_optimizer = keras.optimizers.Adam(learning_rate=learning_rate, clipnorm=clip_norm)
vae_top.compile(optimizer=adam_optimizer, metrics=['accuracy'])

# FIT
history = vae_top.fit(x_train_top, epochs=400, batch_size=64, validation_split=0.2,
verbose=2)

print("Training Of Variational Autoencoder (TOP) is Finished")

# GET HISTORY
history.history.keys()
history.history['loss']
history_path = "history"
loc_history_loss = os.path.join(save_location, history_path, "loss_top.txt")
loc_history_val_loss = os.path.join(save_location, history_path, "val_loss_top.txt")
np.savetxt(loc_history_loss, history.history['loss'], delimiter=",")
np.savetxt(loc_history_val_loss, history.history['val_loss'], delimiter=",")

# SAVE VAE
vae_save_path = "model"
vae_save_loc = os.path.join(save_location, vae_save_path)

encoder_top.save_weights(vae_save_loc + "VAE_TOP_encoder.h5")
decoder_top.save_weights(vae_save_loc + "VAE_TOP_decoder.h5")

samples_to_test_top = loaded_adj_set_top_test
sample_shape = (len(test), adj_size_top, 1)
samples_to_test_top = np.asarray(samples_to_test_top)
samples_to_test_top = samples_to_test_top.reshape(sample_shape)
```

```
predicted_scores_encoder = encoder_top.predict(samples_to_test_top)
predicted_scores_decoder = decoder_top.predict(predicted_scores_encoder[2])

predicted_scores_encoder = np.asarray(predicted_scores_encoder[2])
predicted_scores_encoder_shape = (len(test), (latent_dim_top))
predicted_scores_encoder =
predicted_scores_encoder.reshape(predicted_scores_encoder_shape)
predicted_scores_decoder = np.asarray(predicted_scores_decoder)
predicted_scores_decoder_shape = (len(test), adj_size_top)
predicted_scores_decoder =
predicted_scores_decoder.reshape(predicted_scores_decoder_shape)

predict_path = "predict"
loc_predicted_encoder = os.path.join(save_location, predict_path,
"predicted_encoder_TOP_ID%s.csv" % attempt)
loc_predicted_decoder = os.path.join(save_location, predict_path,
"predicted_decoder_TOP_ID%s.csv" % attempt)
np.savetxt(loc_predicted_encoder, predicted_scores_encoder, delimiter=",")
np.savetxt(loc_predicted_decoder, predicted_scores_decoder, delimiter=",")

# ============================================================================================
# VAE 2: BOTTOM LAYER
# ============================================================================================

latent_dim_bot = 13
b_bot = 0.5  # KL-loss weight


# VAE CLASS
class VAE_BOT(keras.Model):
    def __init__(self, encoder_bot, decoder_bot, **kwargs):
        super(VAE_BOT, self).__init__(**kwargs)
        self.encoder_bot = encoder_bot
        self.decoder_bot = decoder_bot

    def train_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder_bot(data)
            # reconstruction =tf.round(self.decoder_bot(z))
            reconstruction = self.decoder_bot(z)
            reconstruction_loss = tf.reduce_mean(keras.losses.binary_crossentropy(data,
reconstruction))
            reconstruction_loss *= adj_size_bot

            kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
            kl_loss = tf.reduce_mean(kl_loss)
            kl_loss *= -b_bot

            total_loss = reconstruction_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))

        return {
            "loss": total_loss,
            "reconstruction_loss": reconstruction_loss,
            "kl_loss": kl_loss,
```

```
        }

    def test_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        z_mean, z_log_var, z = self.encoder_bot(data)
        reconstruction = self.decoder_bot(z)
        reconstruction_loss = tf.reduce_mean(keras.losses.binary_crossentropy(data,
reconstruction))
        reconstruction_loss *= adj_size_bot

        kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
        kl_loss = tf.reduce_mean(kl_loss)
        kl_loss *= -b_bot
        total_loss = reconstruction_loss + kl_loss

        return {
            "loss": total_loss,
            "reconstruction_loss": reconstruction_loss,
            "kl_loss": kl_loss,
        }


#  SAMPLING LAYER
class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon


# ENCODER
encoder_inputs = keras.Input(shape=(adj_size_bot))
x = layers.Dense(1089, activation="relu")(encoder_inputs)
x = layers.Dense(363, activation="relu")(x)
x = layers.Dense(121, activation="relu")(x)
z_mean = layers.Dense(latent_dim_bot, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim_bot, name="z_log_var")(x)
z_bot = Sampling()([z_mean, z_log_var])
encoder_bot = keras.Model(encoder_inputs, [z_mean, z_log_var, z_bot], name="encoder_bot")

# DECODER
latent_inputs = keras.Input(shape=(latent_dim_bot,))
x = layers.Dense(121, activation="relu")(latent_inputs)
x = layers.Dense(363, activation="relu")(x)
x = layers.Dense(1089, activation="relu")(x)
decoder_outputs = layers.Dense(adj_size_bot, activation="sigmoid")(x)
decoder_bot = keras.Model(latent_inputs, decoder_outputs, name="decoder_bot")
decoder_bot.summary()

# TRAINING
print("Starting Training of Variational Autoencoder (BOT)!")
vae_bot = VAE_BOT(encoder_bot, decoder_bot)

clip_norm = 1.0
learning_rate = 0.001
```

```
adam_optimizer = keras.optimizers.Adam(learning_rate=learning_rate, clipnorm=clip_norm)
vae_bot.compile(optimizer=adam_optimizer, metrics=['accuracy'])

# FIT
history = vae_bot.fit(x_train_bot, epochs=400, batch_size=64, validation_split=0.2,
verbose=2)

print("Training Of Variational Autoencoder (BOT) is Finished")

# GET HISTORY
history.history.keys()
history.history['loss']
history_path = "history"
loc_history_loss = os.path.join(save_location, history_path, "loss_bot.txt")
loc_history_val_loss = os.path.join(save_location, history_path, "val_loss_bot.txt")
np.savetxt(loc_history_loss, history.history['loss'], delimiter=",")
np.savetxt(loc_history_val_loss, history.history['val_loss'], delimiter=",")

# SAVE VAE
vae_save_path = "model"
vae_save_loc = os.path.join(save_location, vae_save_path)

encoder_bot.save_weights(vae_save_loc + "VAE_BOT_encoder.h5")
decoder_bot.save_weights(vae_save_loc + "VAE_BOT_decoder.h5")

samples_to_test_bot = loaded_adj_set_bot_test
sample_shape = (len(test), adj_size_bot, 1)
samples_to_test_bot = np.asarray(samples_to_test_bot)
samples_to_test_bot = samples_to_test_bot.reshape(sample_shape)

predicted_scores_encoder = encoder_bot.predict(samples_to_test_bot)
predicted_scores_decoder = decoder_bot.predict(predicted_scores_encoder[2])

predicted_scores_encoder = np.asarray(predicted_scores_encoder[2])
predicted_scores_encoder_shape = (len(test), (latent_dim_bot))
predicted_scores_encoder =
predicted_scores_encoder.reshape(predicted_scores_encoder_shape)
predicted_scores_decoder = np.asarray(predicted_scores_decoder)
predicted_scores_decoder_shape = (len(test), adj_size_bot)
predicted_scores_decoder =
predicted_scores_decoder.reshape(predicted_scores_decoder_shape)

predict_path = "predict"
loc_predicted_encoder = os.path.join(save_location, predict_path,
"predicted_encoder_BOT_ID%s.csv" % attempt)
loc_predicted_decoder = os.path.join(save_location, predict_path,
"predicted_decoder_BOT_ID%s.csv" % attempt)
np.savetxt(loc_predicted_encoder, predicted_scores_encoder, delimiter=",")
np.savetxt(loc_predicted_decoder, predicted_scores_decoder, delimiter=",")

# ============================================================================================
# GRADIENT DESCENT
# ============================================================================================

gradients_top = []
gradients_bot = []
performance = []
generated_meshes_top = []
```

```
generated_meshes_bot = []

input_mesh_ID = 838
learning_rate = 1

mesh_shape_top = (1, adj_size_top)
mesh_shape_bot = (1, adj_size_bot)

input_mesh_top = loaded_adj_set_top[input_mesh_ID]
input_mesh_top = input_mesh_top.astype('float32')
input_mesh_top = input_mesh_top.reshape(mesh_shape_top)

input_mesh_bot = loaded_adj_set_bot[input_mesh_ID]
input_mesh_bot = input_mesh_bot.astype('float32')
input_mesh_bot = input_mesh_bot.reshape(mesh_shape_bot)

x_top = vae_top.encoder_top(input_mesh_top)
z_top = x_top[2]

x_bot = vae_bot.encoder_bot(input_mesh_bot)
z_bot = x_bot[2]

top_start = [0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8,
8, 8, 8, 9, 9, 9, 10, 10, 10,
             11, 11, 11, 11, 12, 12, 12, 12, 13, 13, 13, 13, 14, 14, 14, 15, 15, 15, 16,
16, 16, 16, 17, 17, 17, 17, 18,
             18, 18,
             18, 19, 19, 19, 20, 20, 21, 21, 21, 22, 22, 22, 23, 23, 23, 24, 24]
top_end = [1, 5, 0, 6, 2, 1, 7, 3, 2, 8, 4, 3, 9, 0, 6, 10, 1, 5, 7, 11, 2, 6, 8, 12, 3,
7, 9, 13, 4, 8, 14, 5, 11,
           15, 6, 10, 12, 16, 7, 11, 13, 17, 8, 12, 14, 18, 9, 13, 19, 10, 16, 20, 11,
15, 17, 21, 12, 16, 18, 22, 13,
           17, 19,
           23, 14, 18, 24, 15, 21, 20, 16, 22, 21, 17, 23, 22, 18, 24, 19, 23]
bot_start = [0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 8,
8, 8, 9, 9, 9, 9, 10, 10, 10,
             10, 11, 11, 11, 12, 12, 13, 13, 13, 14, 14, 14, 15, 15]
bot_end = [1, 4, 0, 5, 2, 1, 6, 3, 2, 7, 0, 5, 8, 1, 4, 6, 9, 2, 5, 7, 10, 3, 6, 11, 4,
9, 12, 5, 8, 10, 13, 6, 9, 11,
           14, 7, 10, 15, 8, 13, 9, 12, 14, 10, 13, 15, 14, 11]

break_out_flag = False

for i in range(1000):
    with tf.GradientTape() as tape:
        tape.watch(z_top)
        tape.watch(z_bot)
        decoded_tensor_top = vae_top.decoder_top(z_top)
        adj_top = decoded_tensor_top
        decoded_tensor_bot = vae_bot.decoder_bot(z_bot)
        adj_bot = decoded_tensor_bot

        # BUILD FULL MESH
        # VALIDITY CHECK:
        check = 0
        for i in adj_top:
            for j in i:
                if j == 1:
```

```
            check += 1

if check != 81:
    print("Check Error:")
    print("  Check (top) is " + str(check))
    break_out_flag = True
    break

if break_out_flag:
    break

check = 0
for i in adj_bot:
    for j in i:
        if j == 1:
            check += 1

if check != 48:
    print("Check Error:")
    print("  Check (bot) is " + str(check))
    break

if break_out_flag:
    break

# ADD TOP
e_start_top = []
e_end_top = []

for i in range(len(adj_top)):
    for j in range(len(adj_top[0])):
        adj_top[i][j] = getattr(adj_top[i][j], "tolist", lambda: adj_top[i][j])()
        if adj_top[i][j] == 1:
            e_start_top.append(i)
            e_end_top.append(j)

e_start_idx_top = []
e_end_idx_top = []

for i in e_start_top:
    e_start_idx_top.append(int(top_v[i]))

for i in e_end_top:
    e_end_idx_top.append(int(top_v[i]))

# ADD BOT
e_start_bot = []
e_end_bot = []

for i in range(len(adj_bot)):
    # print(adj_bot[i])
    for j in range(len(adj_bot[0])):
        adj_bot[i][j] = getattr(adj_bot[i][j], "tolist", lambda: adj_bot[i][j])()
        if adj_bot[i][j] == 1:
            e_start_bot.append(i)
            e_end_bot.append(j)

e_start_idx_bot = []
```

```
e_end_idx_bot = []

for i in e_start_bot:
    e_start_idx_bot.append(int(bot_v[i]))

for i in e_end_bot:
    e_end_idx_bot.append(int(bot_v[i]))

e_top = set()
e_bot = set()

for i in e_start_idx_top:
    e_top.add(i)

for i in e_end_idx_top:
    e_top.add(i)

for i in e_start_idx_bot:
    e_bot.add(i)

for i in e_end_idx_bot:
    e_bot.add(i)

e_top = list(e_top)
e_top.sort()
e_bot = list(e_bot)
e_bot.sort()

# SPLIT LIST TO ORDER
e_top_split = np.array_split(e_top, len(e_top) / 5)
e_bot_split = np.array_split(e_bot, (len(e_bot) / 4))

vertices = list(range(243))
vertices = np.array_split(vertices, len(vertices) / 3)
vertices = np.array_split(vertices, len(vertices) / 9)

sections_full = []

for index in range(len(vertices)):
    section_set = []
    for j in vertices:
        j = list(j)
        section = []
        section.append(list(j[index]))
        for i in section:
            for k in i:
                section_set.append(k)
    sections_full.append(section_set)

e_top_split_idx = []
for vertice_idx in e_top_split:
    section_idx = []
    for x in vertice_idx:
        for i in range(len(sections_full)):
            if x in sections_full[i]:
                section_idx.append(i)
    e_top_split_idx.append(section_idx)
```

```
        e_bot_split_idx = []
        for vertice_idx in e_bot_split:
            section_idx = []
            for x in vertice_idx:
                for i in range(len(sections_full)):
                    if x in sections_full[i]:
                        section_idx.append(i)
            e_bot_split_idx.append(section_idx)

        e_top_split_reorder_set = []
        for i in range(len(e_top_split_idx)):
            order = sorted(range(len(e_top_split_idx[i])), key=lambda k:
e_top_split_idx[i][k])
            e_top_split_reorder = []
            for j in order:
                e_top_split_reorder.append(list(e_top_split[i])[j])
            e_top_split_reorder_set.append(e_top_split_reorder)

        e_bot_split_reorder_set = []
        for i in range(len(e_bot_split_idx)):
            order = sorted(range(len(e_bot_split_idx[i])), key=lambda k:
e_bot_split_idx[i][k])
            e_bot_split_reorder = []
            for j in order:
                e_bot_split_reorder.append(list(e_bot_split[i])[j])
            e_bot_split_reorder_set.append(e_bot_split_reorder)

        e_top_list = []
        e_bot_list = []

        for i in e_top_split_reorder_set:
            for j in i:
                e_top_list.append(j)

        for i in e_bot_split_reorder_set:
            for j in i:
                e_bot_list.append(j)

        # ADD TOP & BOTTOM LAYERS
        top_v_start = []
        top_v_end = []
        bot_v_start = []
        bot_v_end = []

        for i in top_start:
            top_v_start.append(e_top_list[i])

        for i in top_end:
            top_v_end.append(e_top_list[i])

        for i in bot_start:
            bot_v_start.append(e_bot_list[i])

        for i in bot_end:
            bot_v_end.append(e_bot_list[i])

        # ADD INBETWEEN LAYER
        inbtop_v_split = np.array_split(inbtop_v, len(inbtop_v) / 4)
```

```
inb_v_start = []
inb_v_end = []

for i in range(len(e_bot_list)):
    for inb_idx in inbtop_v_split[i]:
        inb_v_start.append(e_bot_list[i])
        inb_v_end.append(e_top_list[int(inb_idx)])

# ADD ALL VERTICES TOGETHER
e_start_full = top_v_start + inb_v_start + inb_v_end + bot_v_start
e_end_full = top_v_end + inb_v_end + inb_v_start + bot_v_end

# GET ADJACENCY MATRIX
adj = np.zeros((243, 243))

for i in range(len(e_start_full)):
    adj[e_start_full[i]][e_end_full[i]] = 1
    adj[e_end_full[i]][e_start_full[i]] = 1

# VALIDITY CHECK 2:
check = 0
for i in adj:
    for j in i:
        if j == 1:
            check += 1

if check != 256:
    print("Check Error:")
    print("  Check (top) is " + str(check))
    break

if break_out_flag:
    break

adj_half_matrix = []
for row in range(len(adj)):
    if row != 0:
        idx = list(range(row))
        new_row = []
        for i in idx:
            new_row.append(adj[row][i])
        for i in new_row:
            adj_half_matrix.append(int(i))

# VALIDITY CHECK 3:
check = 0
for i in adj_half_matrix:
    if i == 1:
        check += 1

if check != 128:
    print("Check Error:")
    print("  Check (half adj) is " + str(check))
    break

if break_out_flag:
    break
```

```
        y = sur_model(adj_half_matrix)
        gradient_top = tape.gradient(y, z_top)
        gradient_bot = tape.gradient(y, z_bot)

    # TOP LAYER GRADIENTS
    gradients_array_top = gradient_top.numpy().reshape(latent_dim_top)
    rms_top = np.sqrt(np.mean((gradients_array_top ** 2)))
    gradients_top.append(rms_top)

    # BOTTOM LAYER GRADIENTS
    gradients_array_bot = gradient_bot.numpy().reshape(latent_dim_bot)
    rms_bot = np.sqrt(np.mean((gradients_array_bot ** 2)))
    gradients_bot.append(rms_bot)

    # PREDICTED PERFORMANCE
    y_arr = y.numpy().reshape(1)
    performance.append(y_arr[0])

    # SAVE MESHES
    decoded_tensor_top_save = vae_top.decoder(z_top)
    decoded_tensor_bot_save = vae_bot.decoder(z_bot)

    generated_meshes_top.append(decoded_tensor_top_save)
    generated_meshes_bot.append(decoded_tensor_bot_save)

    # UPDATE
    z_top = z_top + (learning_rate * gradient_top)
    z_bot = z_bot + (learning_rate * gradient_top)

    if len(performance) > 100 and (abs(performance[i - 1] - performance[i]) /
performance[i - 1]) * 100 < 0.0001:
        break

generated_meshes_top = np.asarray(generated_meshes_top)
mesh_shape = (len(generated_meshes_top), adj_size_full)
generated_meshes_top = generated_meshes_top.reshape(mesh_shape)

generated_meshes_bot = np.asarray(generated_meshes_bot)
mesh_shape = (len(generated_meshes_bot), adj_size_full)
generated_meshes_bot = generated_meshes_bot.reshape(mesh_shape)

# SAVE GENERATED MESHES
generated_meshes_path = "generated_meshes"
loc_generated_meshes_top = os.path.join(save_location, generated_meshes_path,
"generated_meshes_top_ID%s.csv" % attempt)
loc_generated_meshes_bot = os.path.join(save_location, generated_meshes_path,
"generated_meshes_bot_ID%s.csv" % attempt)
loc_performance = os.path.join(save_location, generated_meshes_path,
"performance_ID%s.csv" % attempt)
loc_gradients_top = os.path.join(save_location, generated_meshes_path,
"gradients_top_ID%s.csv" % attempt)
loc_gradients_bot = os.path.join(save_location, generated_meshes_path,
"gradients_bot_ID%s.csv" % attempt)
np.savetxt(loc_generated_meshes_top, generated_meshes_top, delimiter=",")
np.savetxt(loc_generated_meshes_bot, generated_meshes_bot, delimiter=",")
np.savetxt(loc_performance, performance, delimiter=",")
```

```
np.savetxt(loc_gradients_top, gradients_top, delimiter=",")
np.savetxt(loc_gradients_bot, gradients_bot, delimiter=",")
```

Full VAE-based workflow code with split top and bottom structures:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, activations
import random
import matplotlib.style
import matplotlib.pyplot as plt
import os

from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Reshape, Flatten, Input
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras import backend as K

# ID
attempt = 4
value = 'p_overall'
job_directory = os.getcwd()
train_size = 5000
adj_size = 29403
save_location = os.path.join(job_directory, "gan_%s" % attempt)
test = [34, 167, 454, 477, 783, 335, 749, 290, 607, 544]

# LOAD PERFORMANCE INDICATOR DATASET
column_names = ["index", "seed", "p_material_use", "p_stock_similarity",
"p_displacement", "p_utilization",
                "p_structural", "p_structural_material_use", "p_overall", "material_use",
"displacement", "utilization"]
dataset_path = "dataset/dataset_unaugmented-extended2-fix.csv"
loc_dataset = os.path.join(job_directory, dataset_path)
dataset = pd.read_csv(loc_dataset, names=column_names)

calculated_scores_set_full = dataset[value]
calculated_scores_set = calculated_scores_set_full[:train_size]
dataset = dataset[:train_size]

# PERFORMANCE DATA - LABELS
train_labels = dataset[value]
train_labels_np = np.array(train_labels)
train_labels_np = train_labels_np.astype('float32')
train_labels_np = train_labels_np[:train_size]
train_labels_np = train_labels_np.reshape(train_size, 1)
labels = train_labels_np

# LOAD ADJ MATRIX DATASET
loaded_adj_set_path = "dataset/AdjMatrix243_full-half_set_Mesh0-4999.csv"
loc_adj_set = os.path.join(job_directory, loaded_adj_set_path)
loaded_adj_set = np.loadtxt(loc_adj_set, delimiter=",")
loaded_adj_set = loaded_adj_set[:train_size]

loaded_adj_set_test_path = "dataset/AdjMatrix243_full-half-set_test.csv"
loc_adj_set_test = os.path.join(job_directory, loaded_adj_set_test_path)
loaded_adj_set_test = np.loadtxt(loc_adj_set_test, delimiter=",")
```

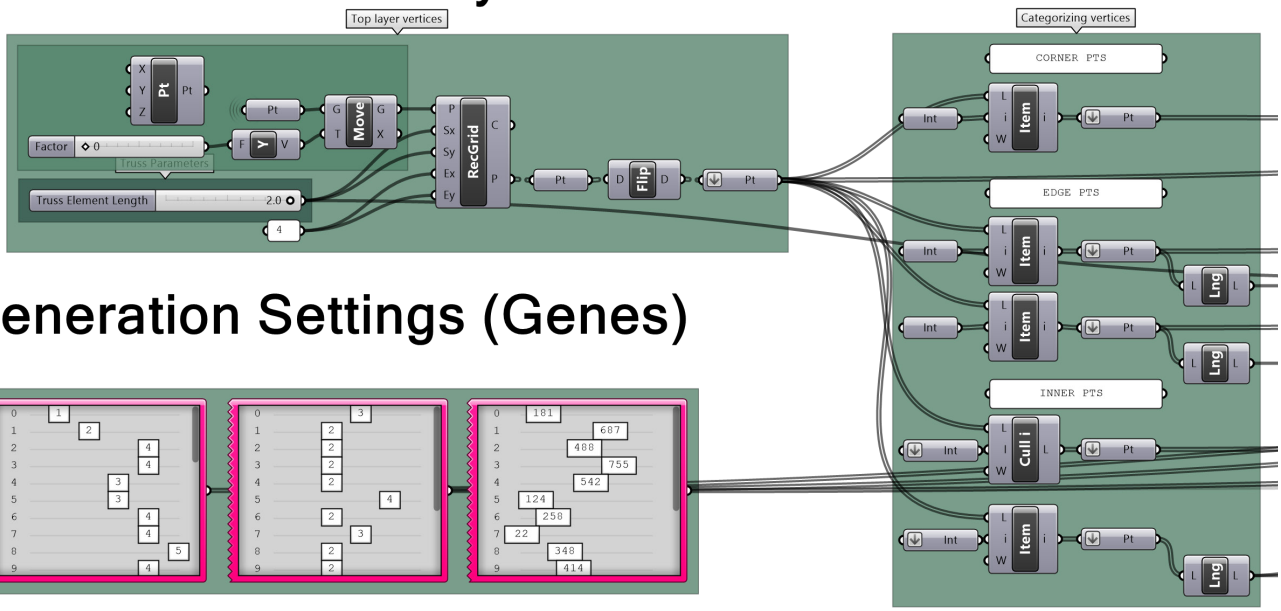
```
# TRAIN DATA
train_shape = (train_size, adj_size)
input_shape = (adj_size)
train_data = loaded_adj_set.reshape(train_shape)


#=============================================================================================
# SURROGATE MODEL
#=============================================================================================

normalizer = tf.keras.layers.experimental.preprocessing.Normalization(axis=1)


def build_and_compile_model(normalizer):
    model = keras.Sequential([
        keras.Input(shape=(input_shape)),
        layers.Flatten(),
        layers.Dense(256, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(1)
    ])
    return model


print("Starting Training of Surrogate Model!")
sur_model = build_and_compile_model(normalizer)
sur_model.compile(loss=["mse", "mse"], optimizer=tf.keras.optimizers.Adam(0.0001,
clipnorm=1.), metrics=['accuracy'])

history = sur_model.fit(x_train,
                        y_train,
                        validation_split=0.2,
                        verbose=1,
                        epochs=600,
                        batch_size=32
                        )

# Get loss history
loss = history.history['loss']

print("Training Of Surrogate Model is Finished")

# SAVE MODEL
history_path = "history"
model_path = "model/"
loc_model = os.path.join(save_location, model_path)
sur_model.save(loc_model, save_format='HDF5')
loc_history_loss = os.path.join(save_location, history_path, "loss.txt")
loc_history_val_loss = os.path.join(save_location, history_path, "val_loss.txt")
np.savetxt(loc_history_loss, history.history['loss'], delimiter=",")
np.savetxt(loc_history_val_loss, history.history['val_loss'], delimiter=",")

# PREDICT SAMPLES
fullset_to_test = np.asarray(loaded_adj_set)
fullset_shape = (train_size, adj_size_full)
fullset_to_test = fullset_to_test.reshape(fullset_shape)
```

```
calculated_scores_fullset = calculated_scores_set
predicted_scores_fullset = []
predicted_scores_fullset.append(sur_model.predict(fullset_to_test, batch_size=1))

plot_path = "plots"

loc_calculated_fullset = os.path.join(save_location, plot_path,
"calculated_fullset_attempt%s.txt" % attempt)
loc_predicted_fullset = os.path.join(save_location, plot_path,
"predicted_fullset_attempt%s.txt" % attempt)
np.savetxt(loc_calculated_fullset, calculated_scores_fullset, delimiter=",")
np.savetxt(loc_predicted_fullset, predicted_scores_fullset[0], delimiter=",")

# ============================================================================================
# GAN
# ============================================================================================
noise_dim = 100
gradient_penalty_weight = 10

def wasserstein_loss(y_true, y_pred):
    return K.mean(y_true * y_pred)

def gradient_penalty_loss(y_true, y_pred, averaged_samples):
    gradients = K.gradients(y_pred, averaged_samples)[0]
    gradients_sqr = K.square(gradients)
    gradients_sqr_sum = K.sum(gradients_sqr, axis=np.arange(1, len(gradients_sqr.shape)))
    gradient_l2_norm = K.sqrt(gradients_sqr_sum)
    gradient_penalty = K.square(1 - gradient_l2_norm)
    return K.mean(gradient_penalty)

def build_generator():
    model = Sequential()
    model.add(Dense(256, input_dim=noise_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(1024))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(29403, activation='tanh'))
    return model

def build_discriminator():
    model = Sequential()
    model.add(Dense(1024, input_dim=29403))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(256))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(1))
    return model

def build_gan(generator, discriminator):
    z = Input(shape=(noise_dim,))
```

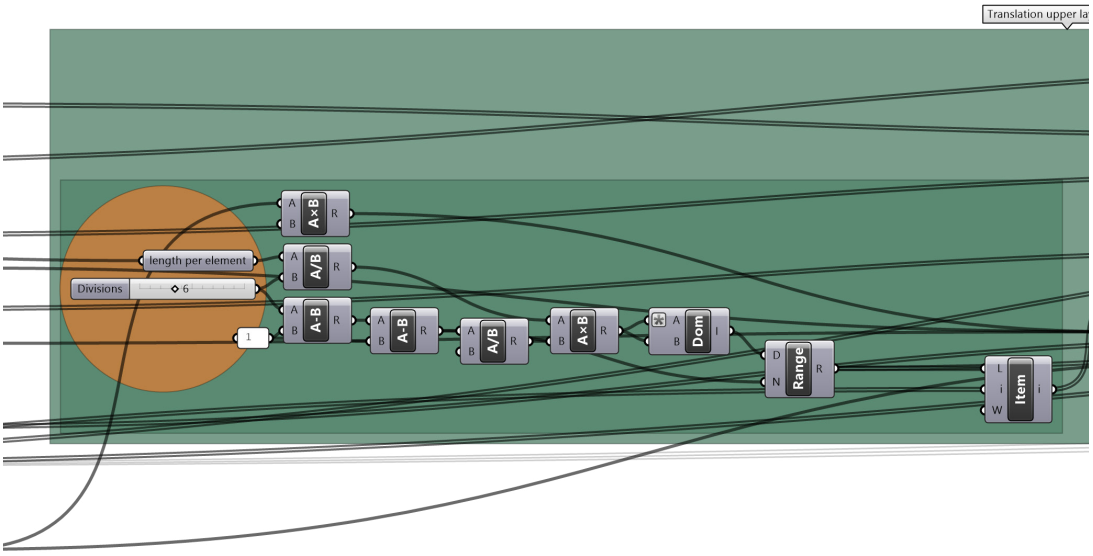
```
    data = generator(z)
    discriminator.trainable = False
    real = discriminator(data)
    return Model(z, real)


# BUILD GAN MODEL
train_data = train_data * 2 - 1

optimizer = RMSprop(lr=0.0001)

discriminator = build_discriminator()
discriminator.compile(loss=wasserstein_loss, optimizer=optimizer)

generator = build_generator()
generator.compile(loss=wasserstein_loss, optimizer=optimizer)

gan = build_gan(generator, discriminator)
gan.compile(loss=wasserstein_loss, optimizer=optimizer)

# GAN TRAINING
batch_size = 64
epochs = 600

discriminator_losses = []
generator_losses = []

for epoch in range(epochs):
    for _ in range(batch_size):
        noise = np.random.normal(0, 1, (batch_size, noise_dim))

        idx = np.random.randint(0, train_data.shape[0], batch_size)
        real_datas = train_data[idx]
        fake_datas = generator.predict(noise)

        d_loss_real = discriminator.train_on_batch(real_datas, -np.ones((batch_size, 1)))
        d_loss_fake = discriminator.train_on_batch(fake_datas, np.ones((batch_size, 1)))
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # Train the generator
        noise = np.random.normal(0, 1, (batch_size, noise_dim))
        g_loss = gan.train_on_batch(noise, -np.ones((batch_size, 1)))

    discriminator_losses.append(d_loss)
    generator_losses.append(g_loss)

    if epoch % 10 == 0:
        print("%d [D loss: %f] [G loss: %f]" % (epoch, d_loss, g_loss))

# GET HISTORY
save_location = os.path.join(job_directory, "gan_%s" % attempt)
loc_history_dis_loss = os.path.join(save_location, history_path,
"discriminator_losses.csv")
loc_history_gen_loss = os.path.join(save_location, history_path, "generator_losses.csv")
np.savetxt(loc_history_dis_loss, discriminator_losses, delimiter=',')
np.savetxt(loc_history_gen_loss, generator_losses, delimiter=',')

# SAVE GAN
```

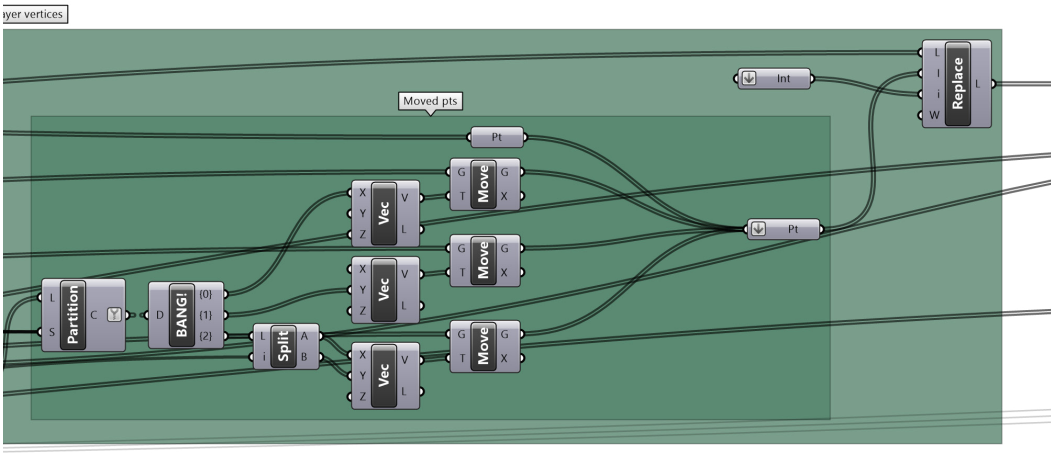
```
gan_save_path = "model"
gan_save_loc = os.path.join(save_location, gan_save_path)

gan.save(gan_save_loc + 'gan_model.h5')

# GENERATE VALUES
y_to_improve = 0.75
optimized_y_list = []
generated_data_list = []
step = 0
optimized_y = 0

while optimized_y < y_to_improve and step < 1000:
    noise = np.random.normal(0, 1, (10, noise_dim))
    fake_labels = np.random.randint(0, 10, (10, 1))
    generated_data = generator.predict(noise, fake_labels)

    y = sur_model(generated_data)
    optimized_y = max(y)
    max_index = y.index(optimized_y)
    optimized_y_list.append(optimized_y)
    optimized_generated_data = generated_data[max_index]
    generated_data_list.append(optimized_generated_data)
    step += 1

# SAVE GENERATED DATA
predict_path = "predict"
loc_generated = os.path.join(save_location, predict_path, "generated_optimized_ID%s.csv"
% attempt)
loc_performance = os.path.join(save_location, predict_path,
"performance_optimized_ID%s.csv" % attempt)
np.savetxt(loc_generated, generated_data_list, delimiter=",")
np.savetxt(loc_performance, optimized_y_list, delimiter=",")
```

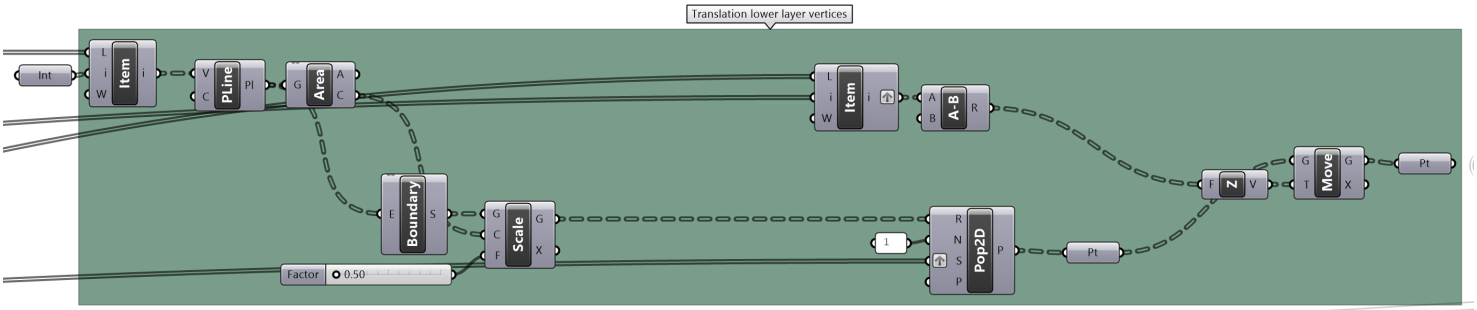
Truss Roof Geometry



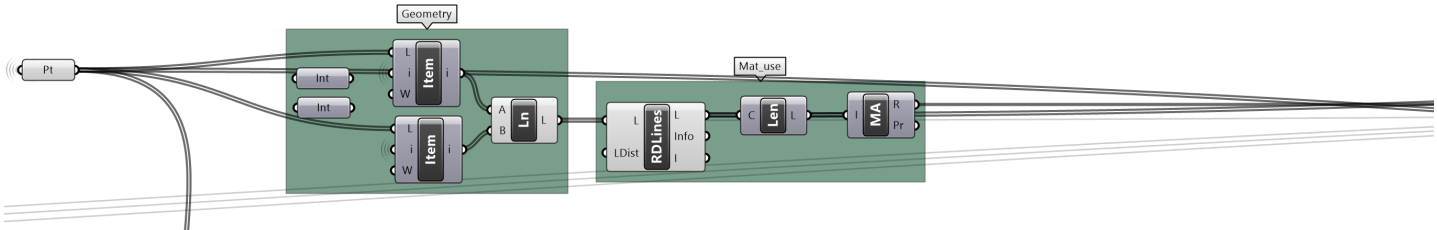
Generation Settings (Genes)



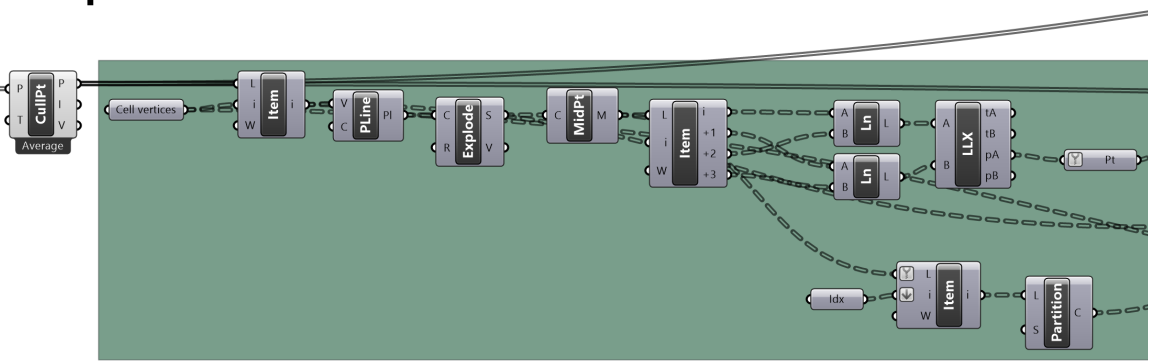


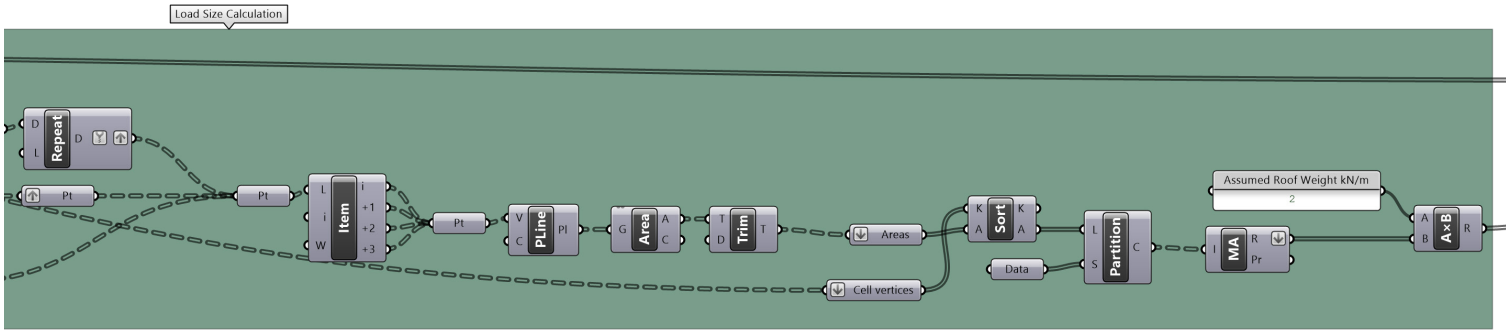


Material Use

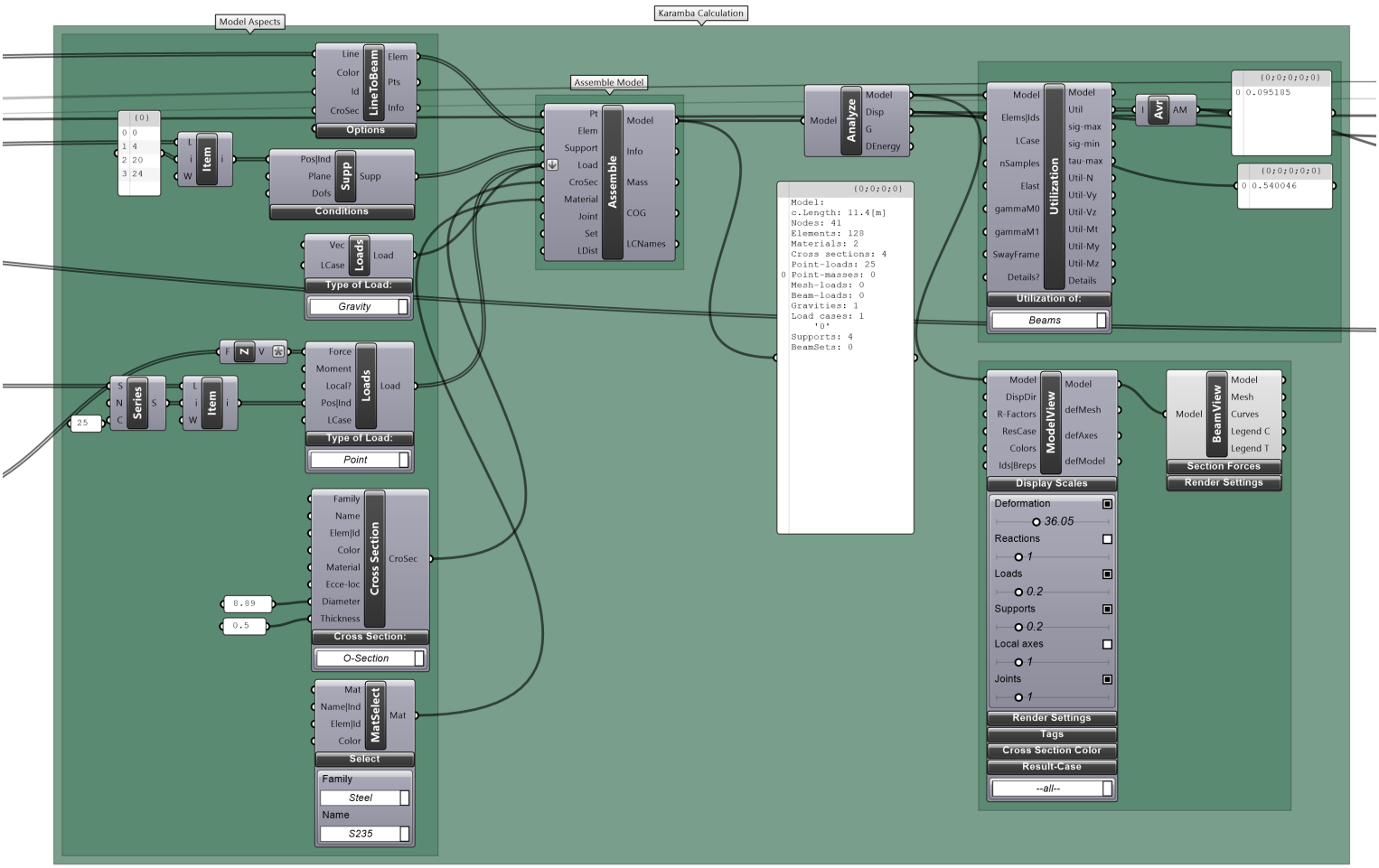


Preperation for Structural Assessment

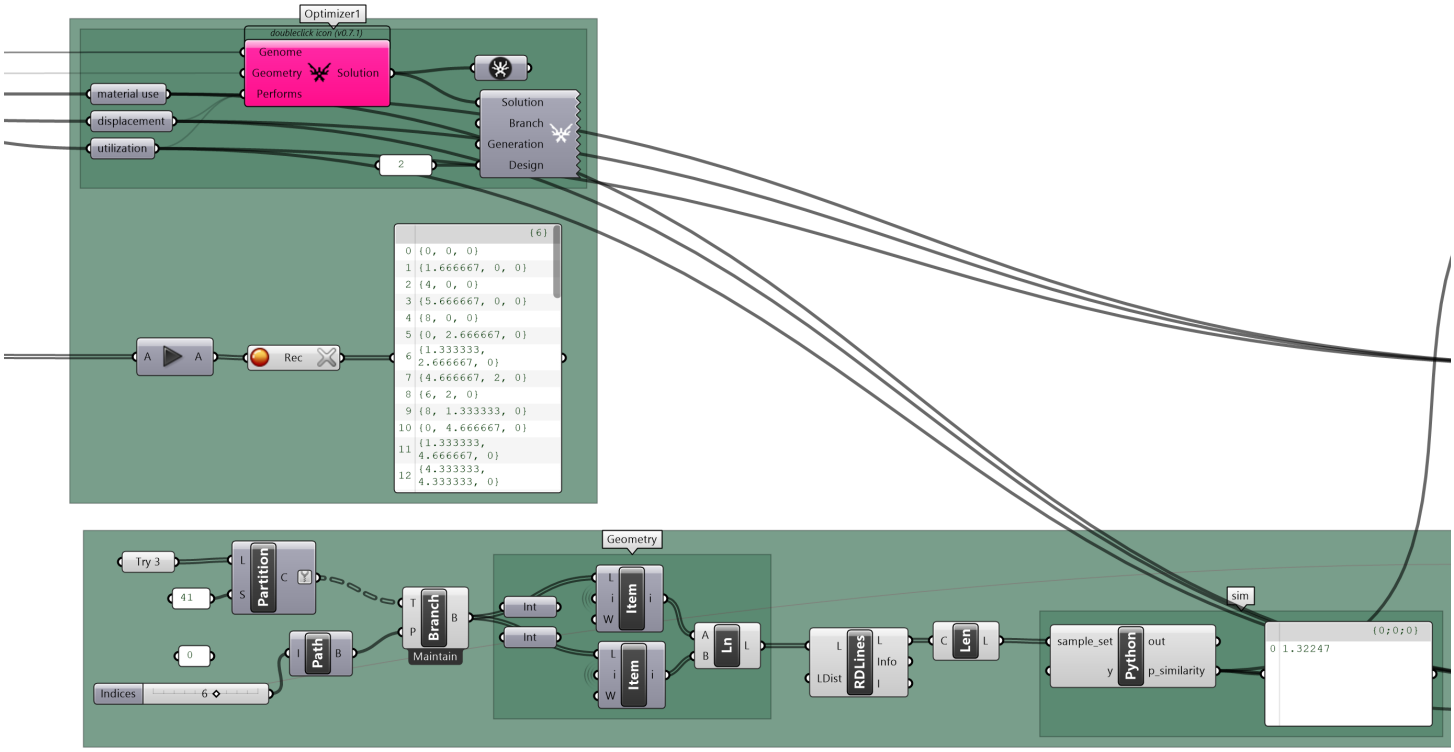




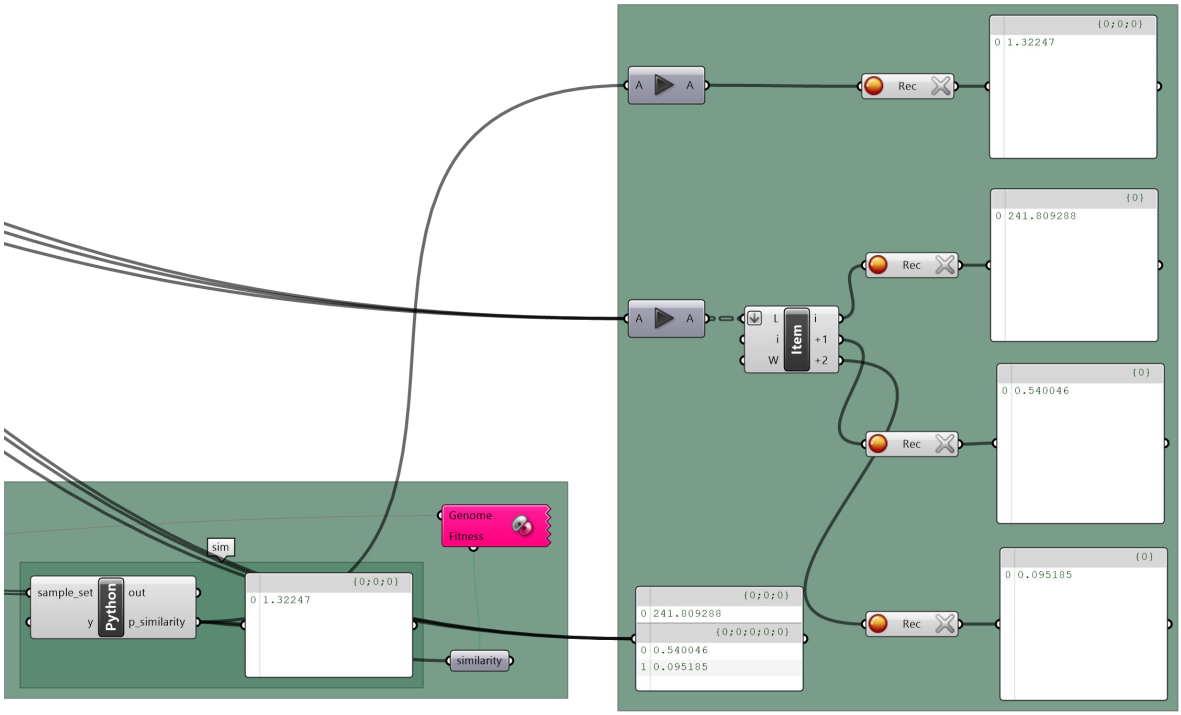
Structural Assessment



Evolutionary Algorithm



Save Results in GH



Appendix IV

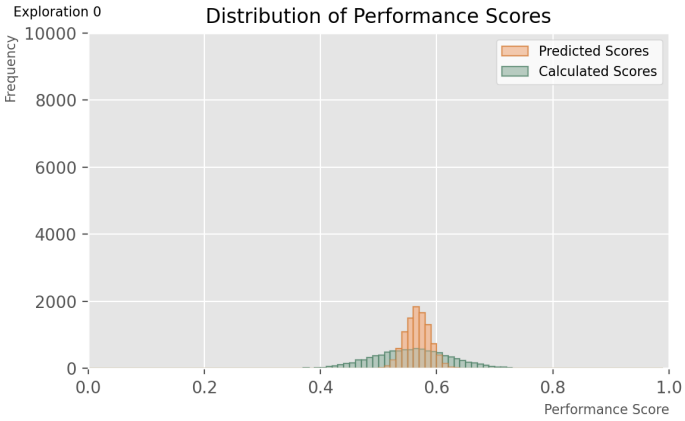
Surrogate model results

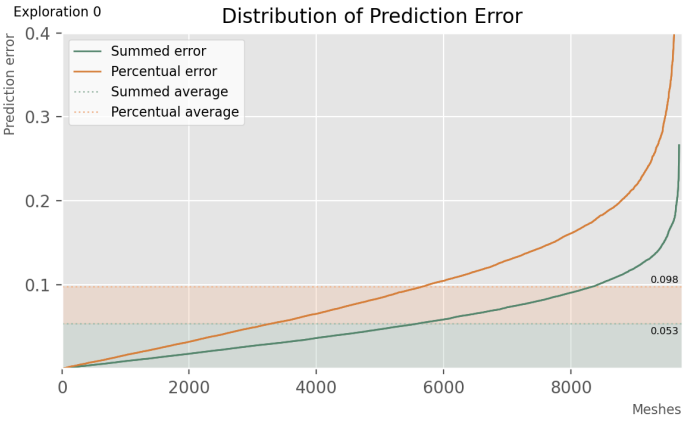
SURROGATE MODEL: EXPLORATION 0

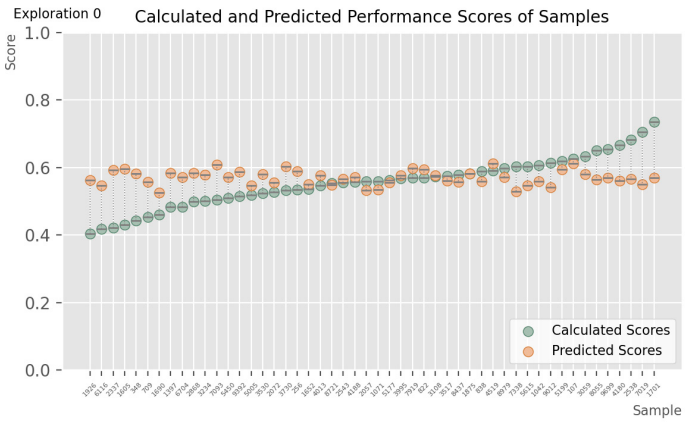
The attributes of this model are:

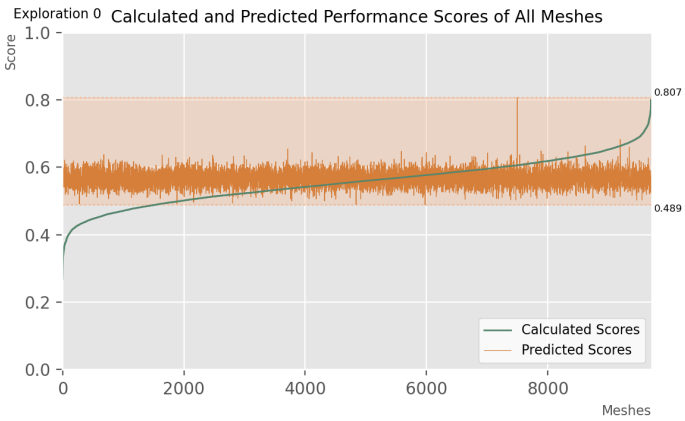
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.







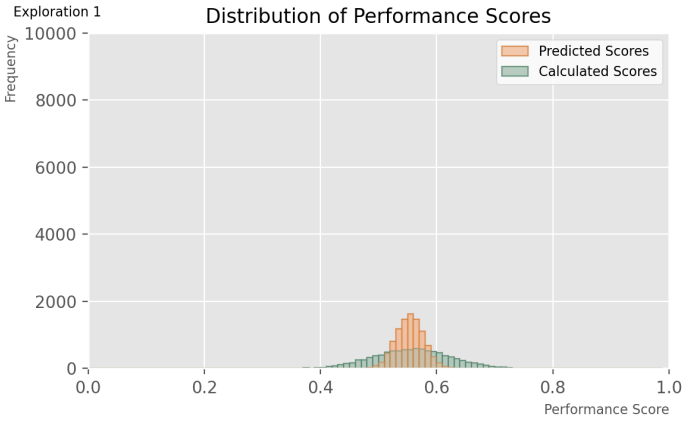


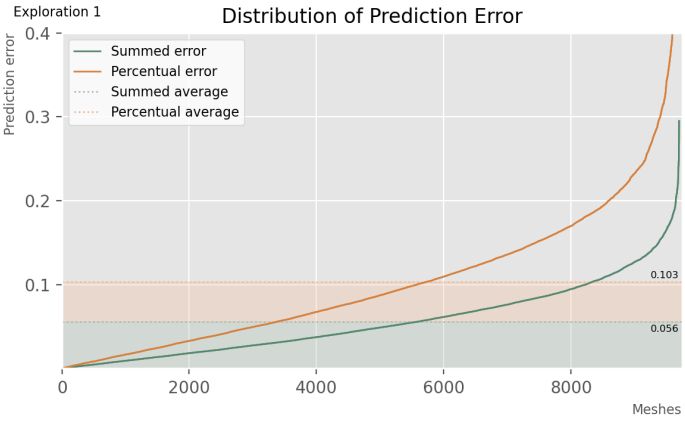
SURROGATE MODEL: EXPLORATION 1

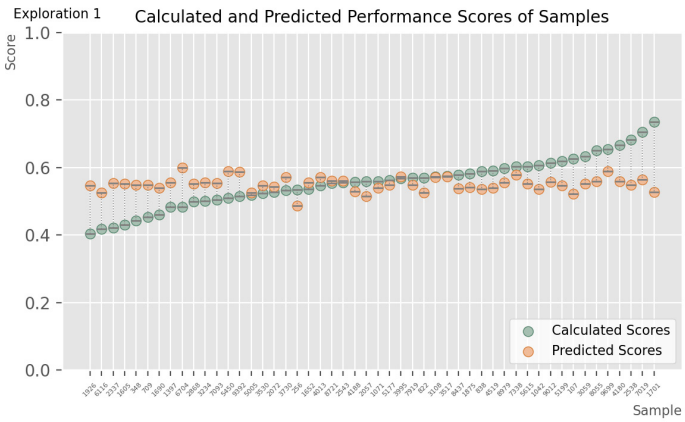
The attributes of this model are:

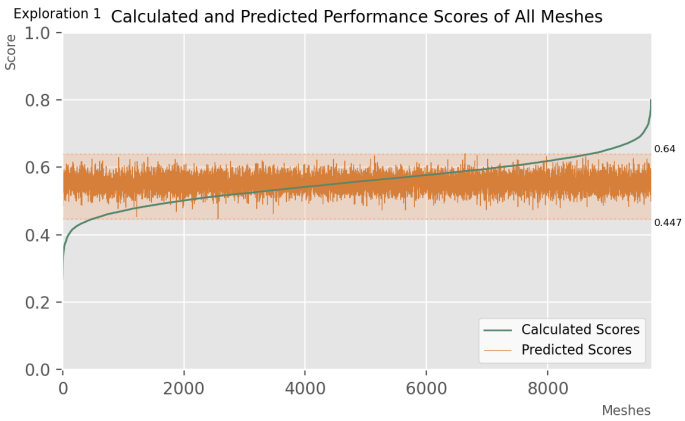
Batch size:	**16**
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.







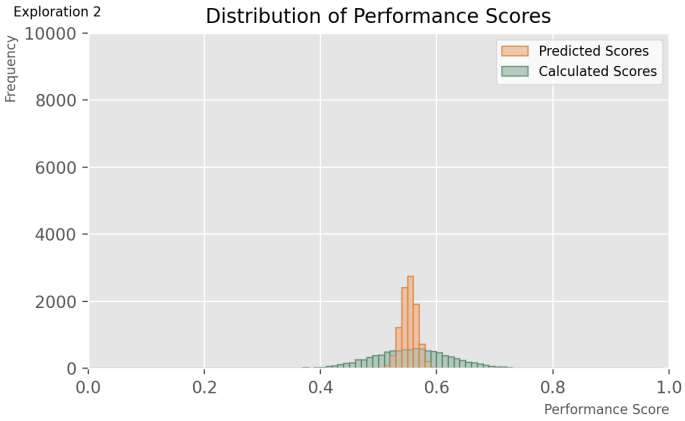


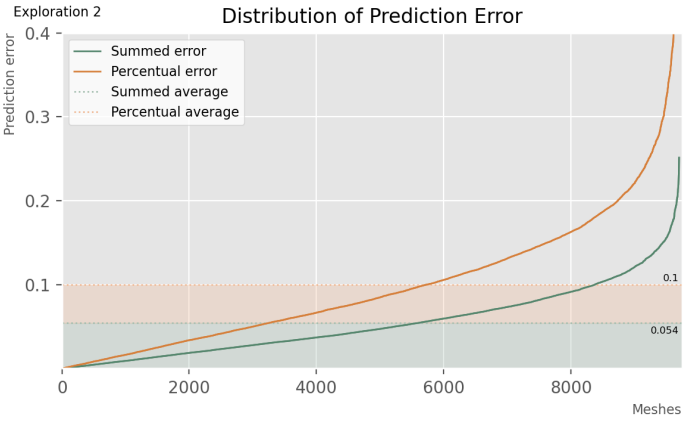
SURROGATE MODEL: EXPLORATION 2

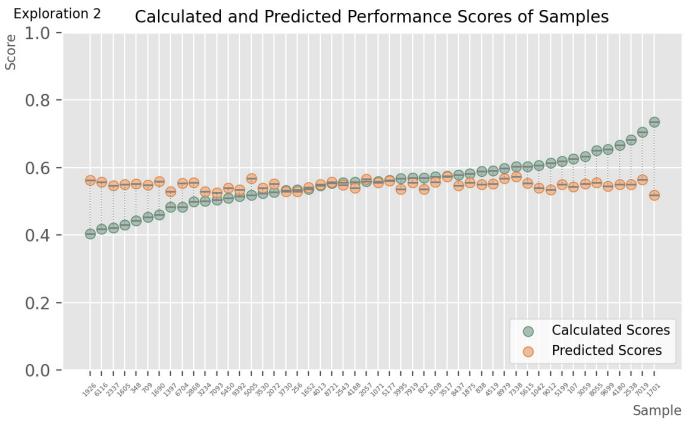
The attributes of this model are:

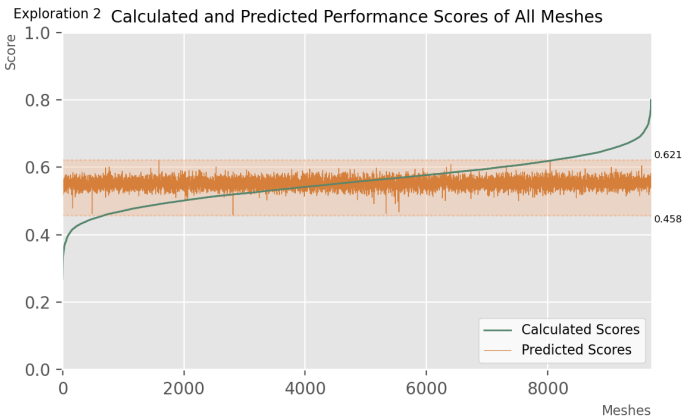
Batch size:	64
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





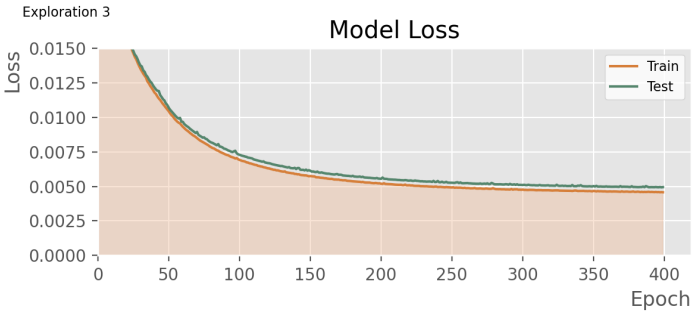




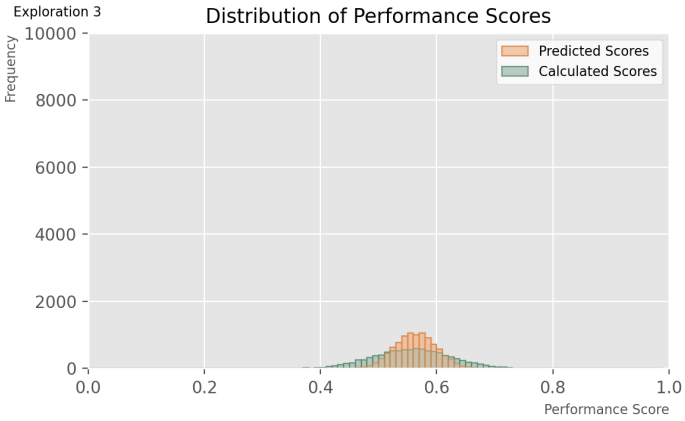
SURROGATE MODEL: EXPLORATION 3

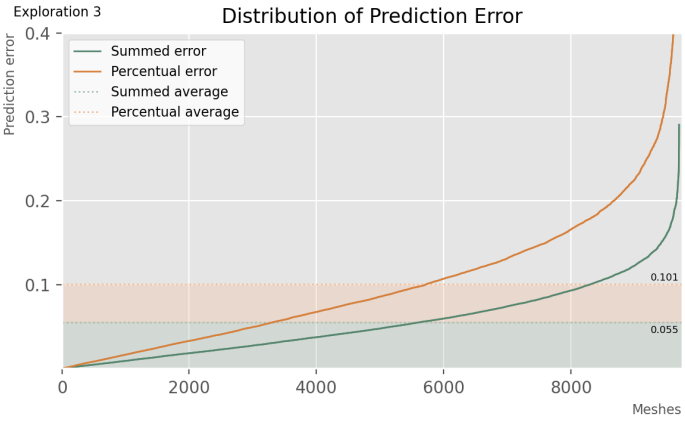
The attributes of this model are:

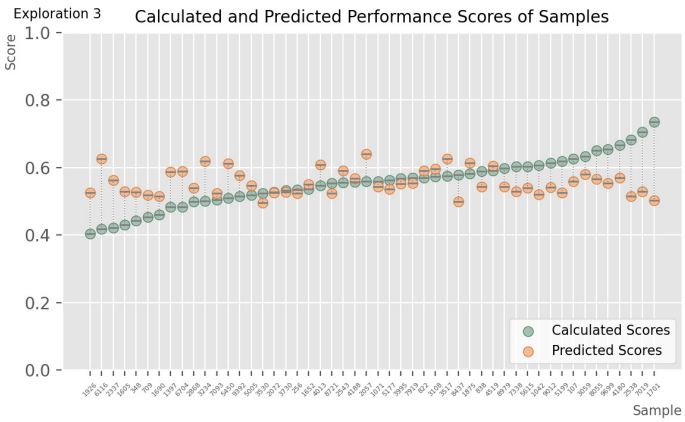
Batch size:	**128**
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

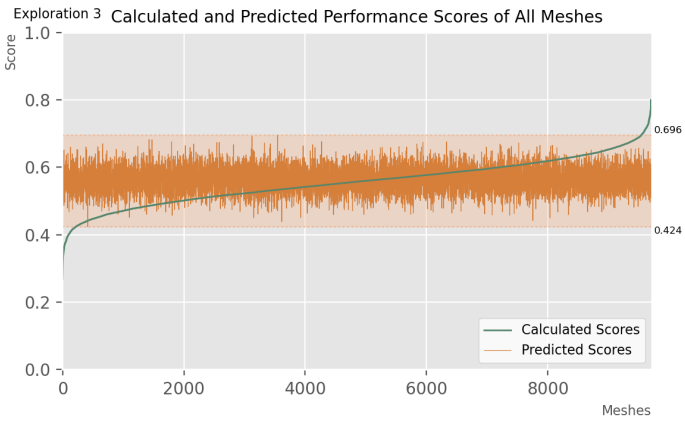


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.







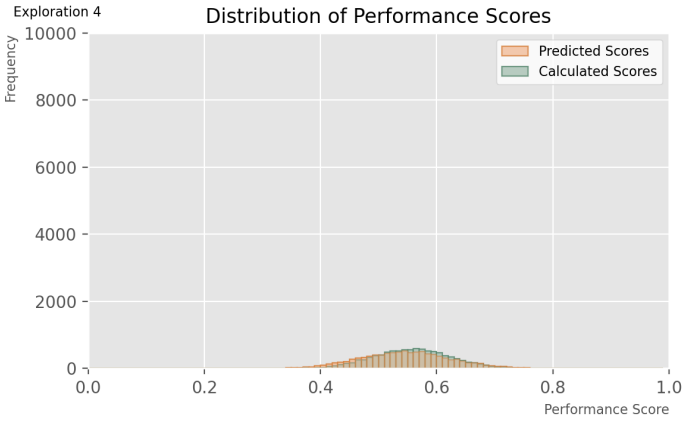


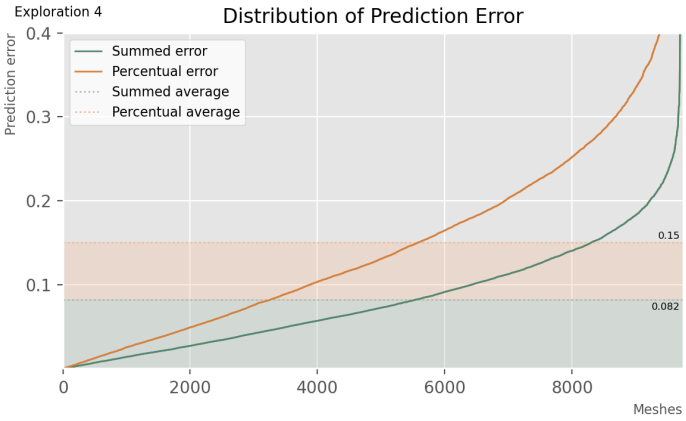
SURROGATE MODEL: EXPLORATION 4

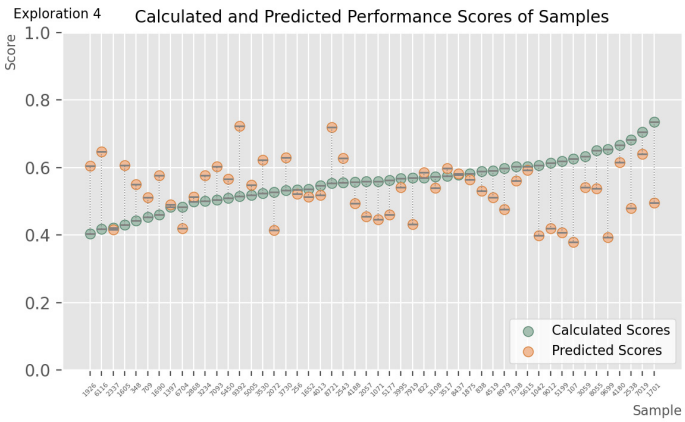
The attributes of this model are:

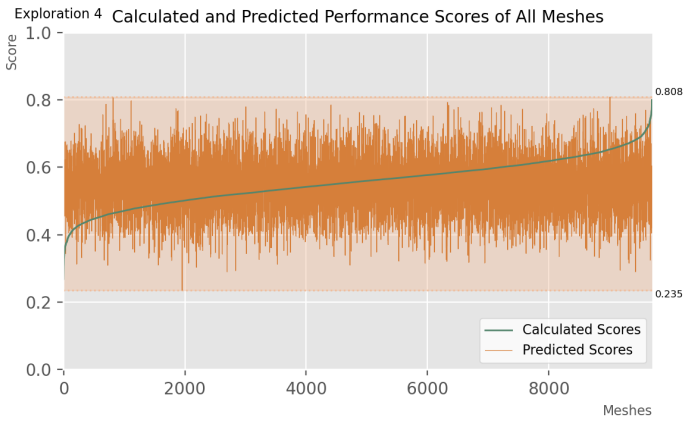
Batch size:	32
Epoch at which training is stopped:	**Before** the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

***NOTE:** For the above graph, limits of both axes were adapted to fit data.





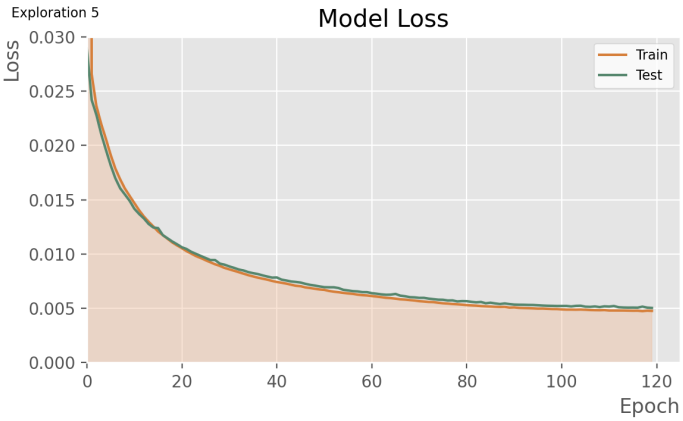




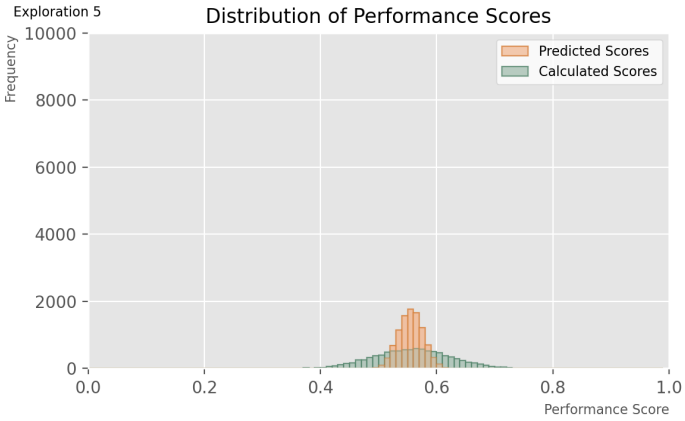
SURROGATE MODEL: EXPLORATION 5

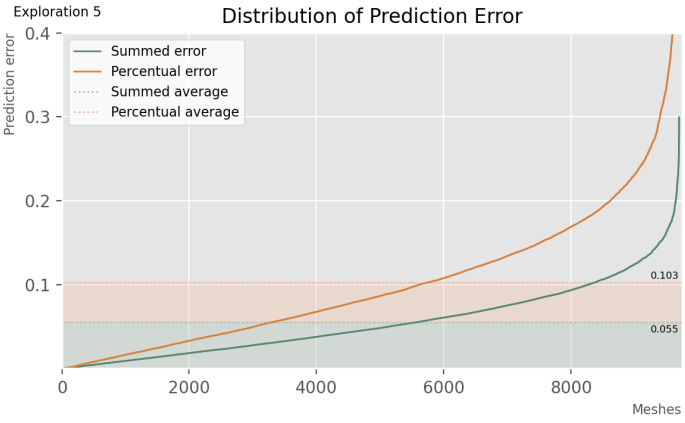
The attributes of this model are:

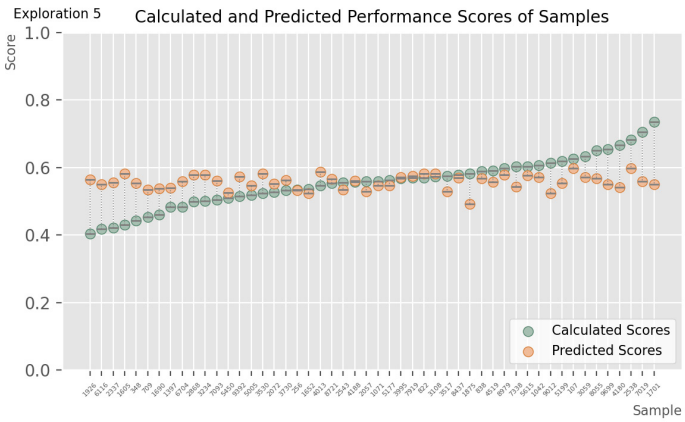
Batch size:	32
Epoch at which training is stopped:	After the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

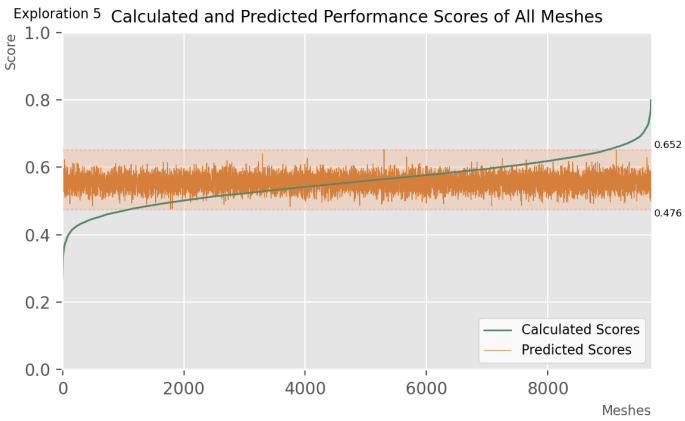


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





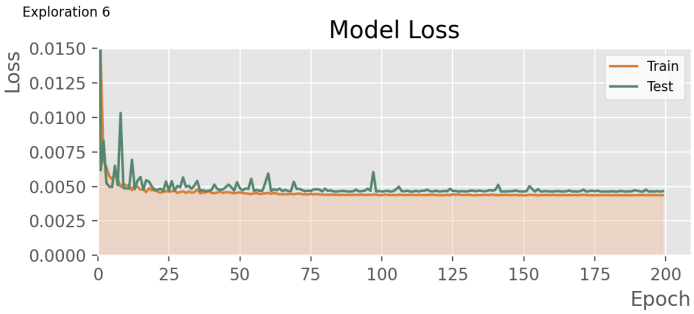


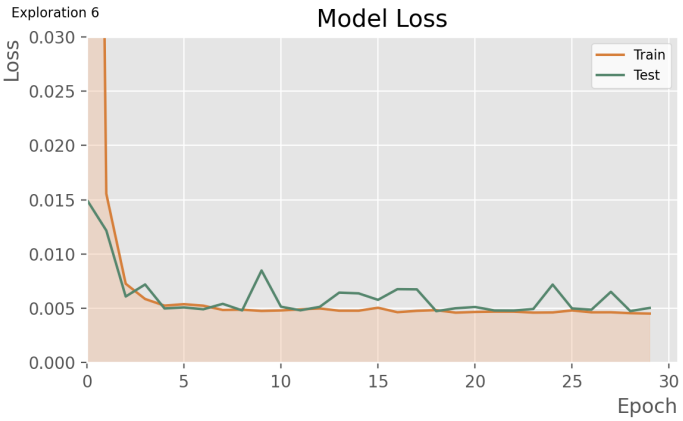


SURROGATE MODEL: EXPLORATION 6

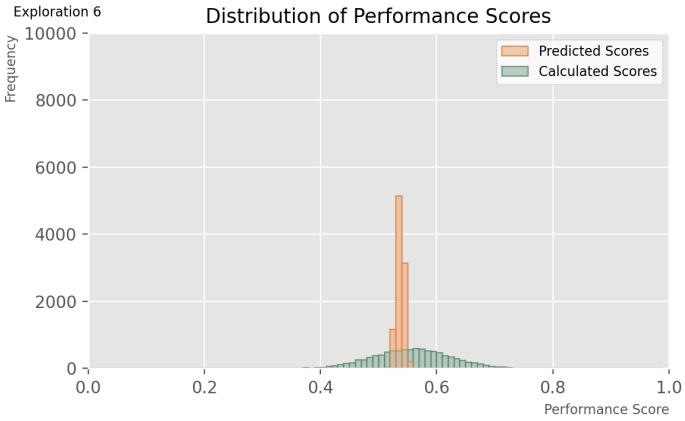
The attributes of this model are:

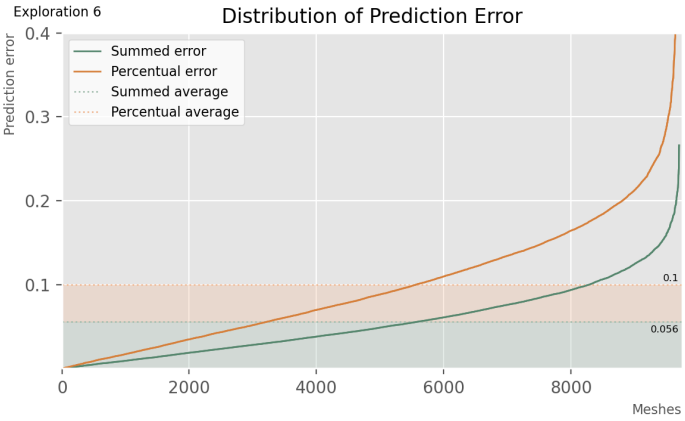
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

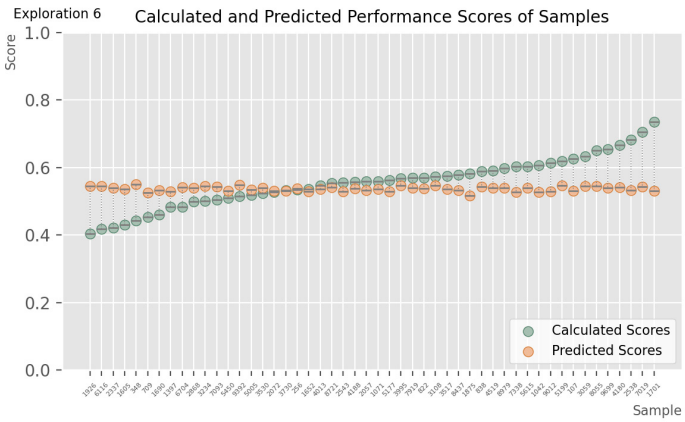


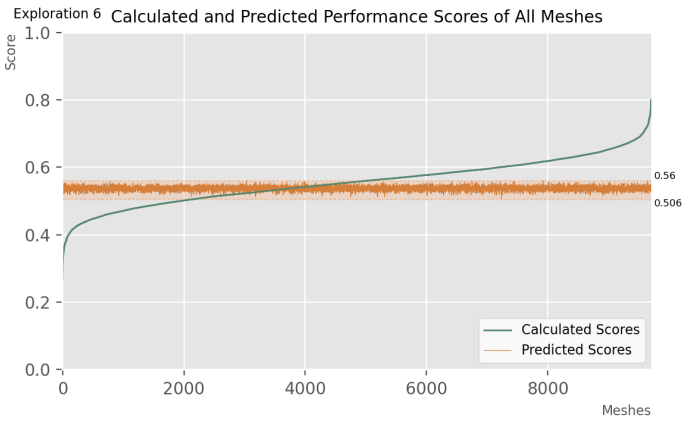


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





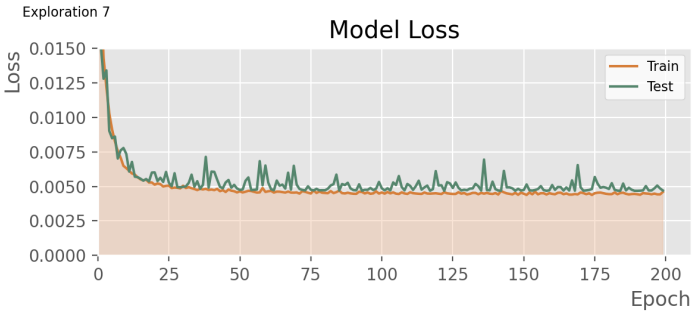




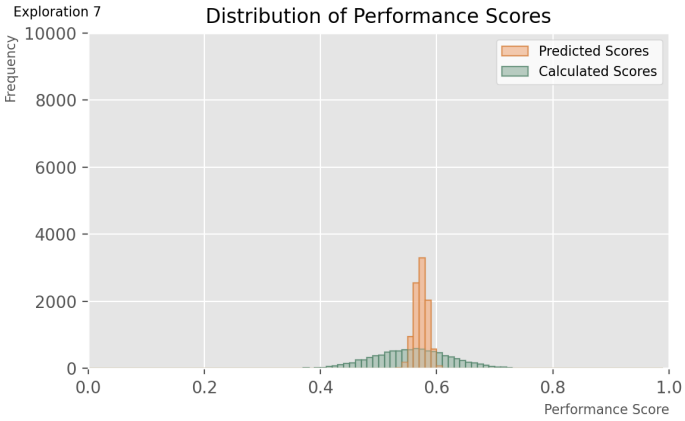
SURROGATE MODEL: EXPLORATION 7

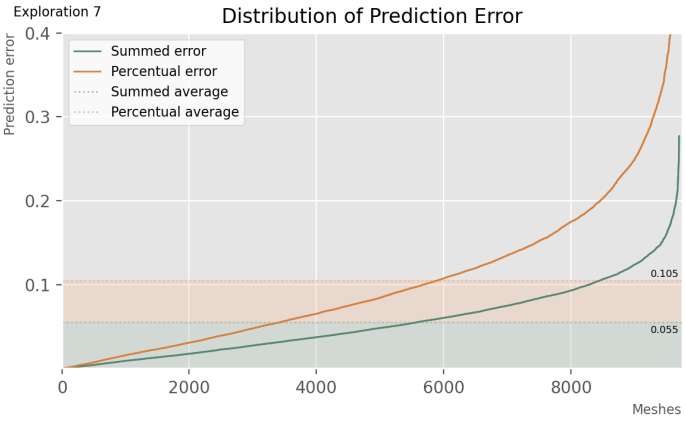
The attributes of this model are:

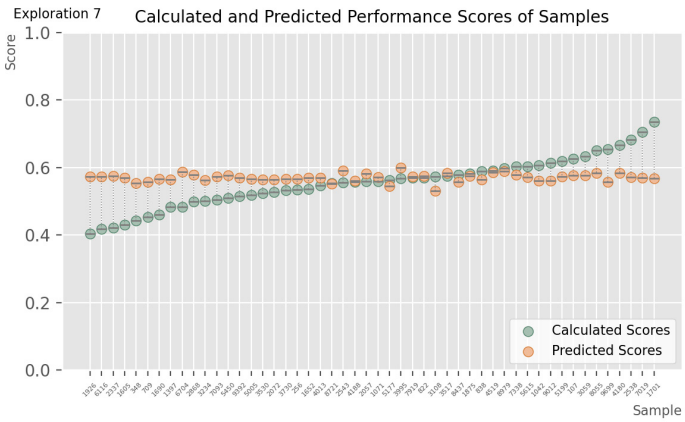
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.0001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

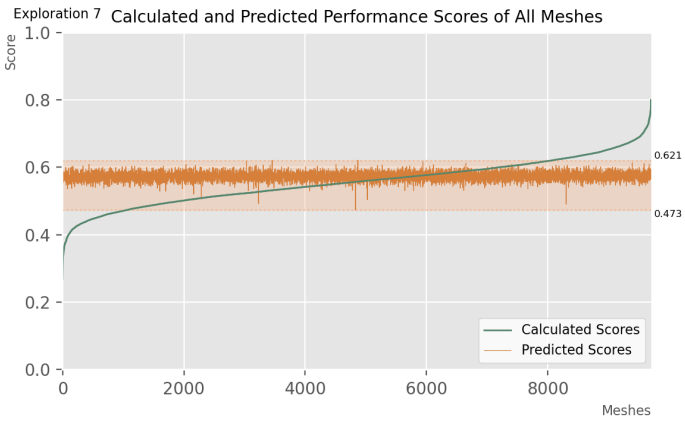


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.







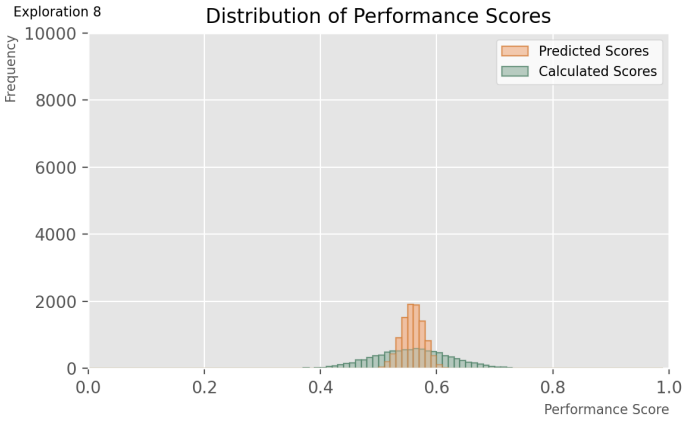


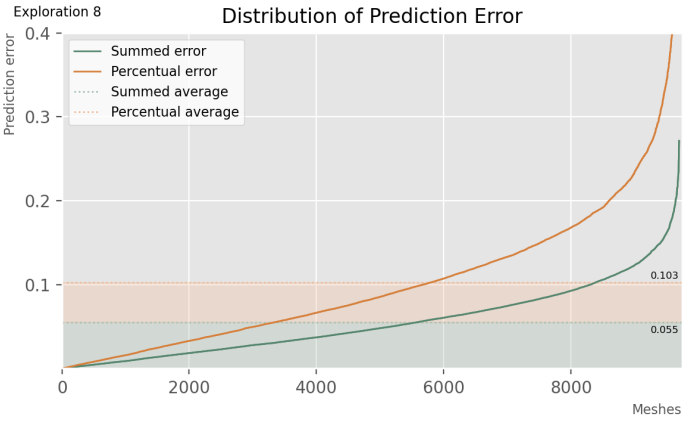
SURROGATE MODEL: EXPLORATION 8

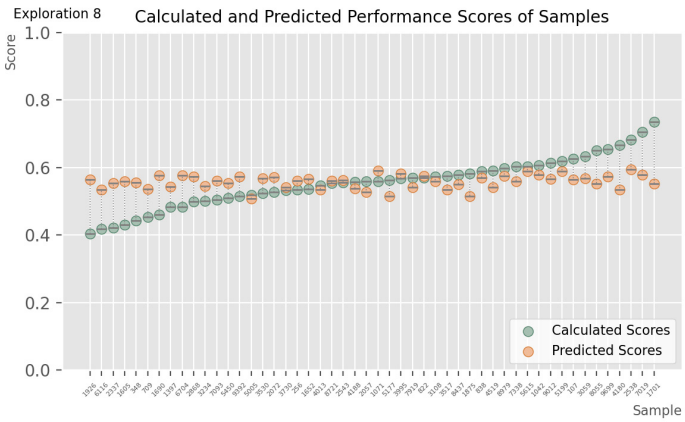
The attributes of this model are:

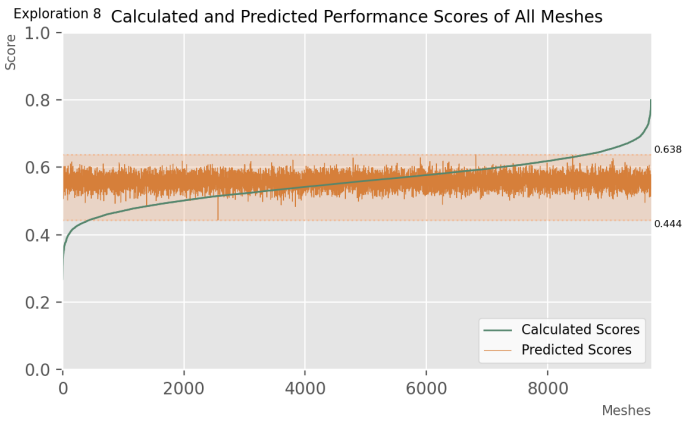
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.000005
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





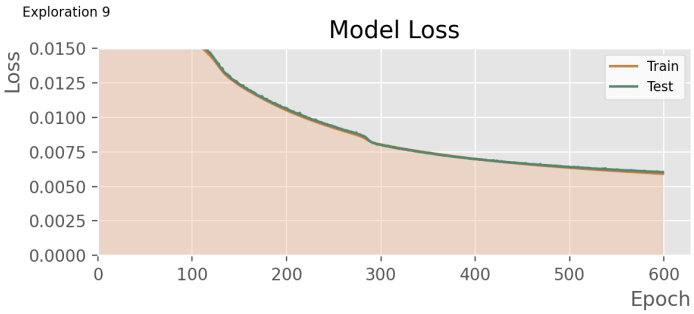




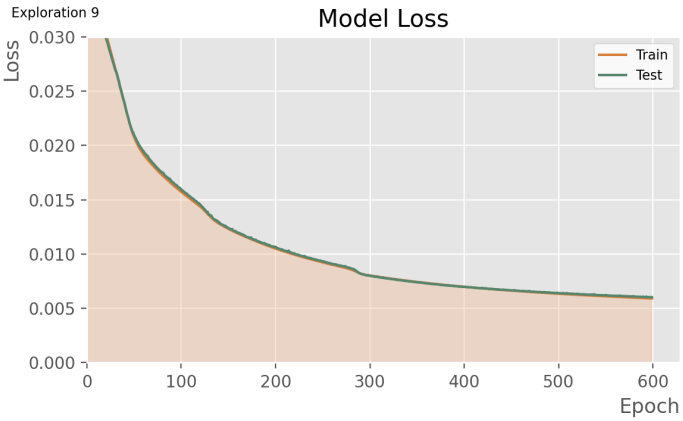
SURROGATE MODEL: EXPLORATION 9

The attributes of this model are:

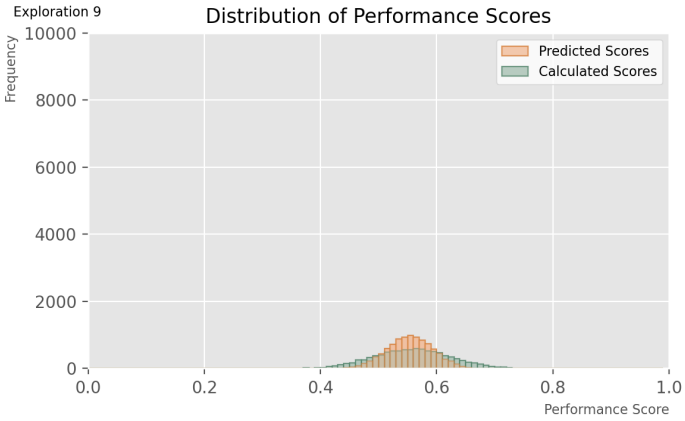
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.000001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

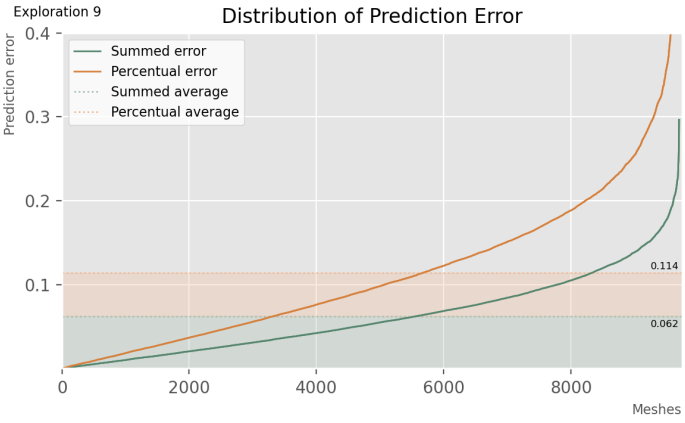


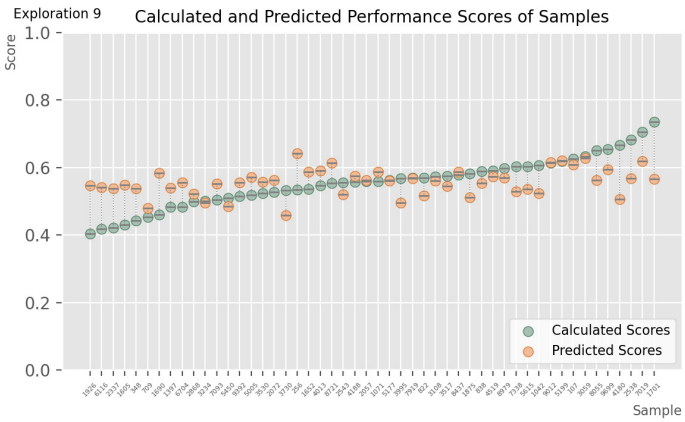
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.

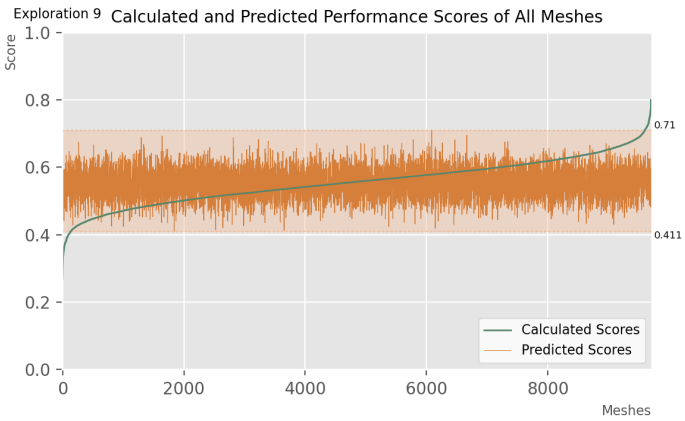


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.









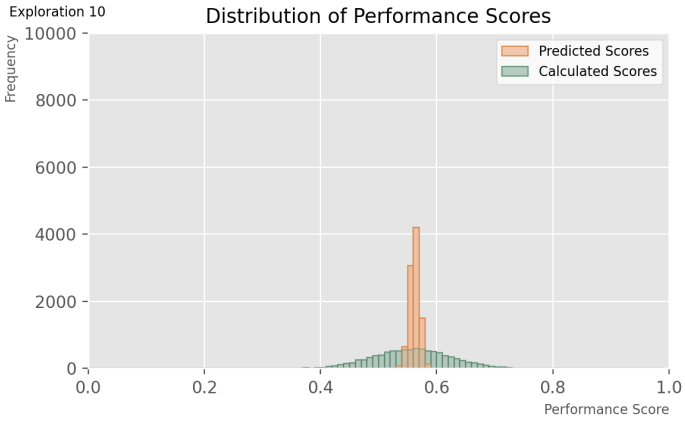
SURROGATE MODEL: EXPLORATION 10

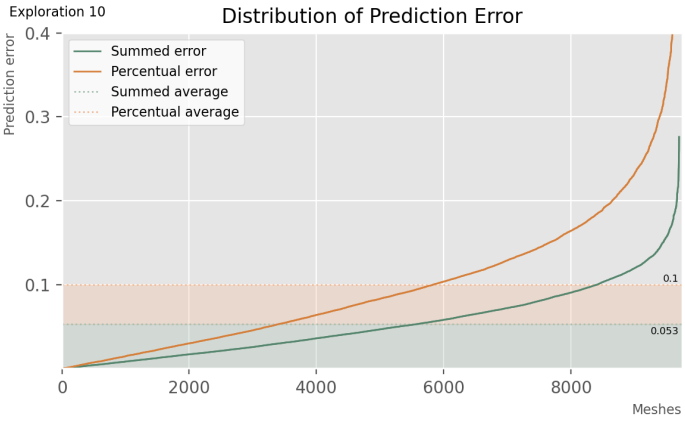
The attributes of this model are:

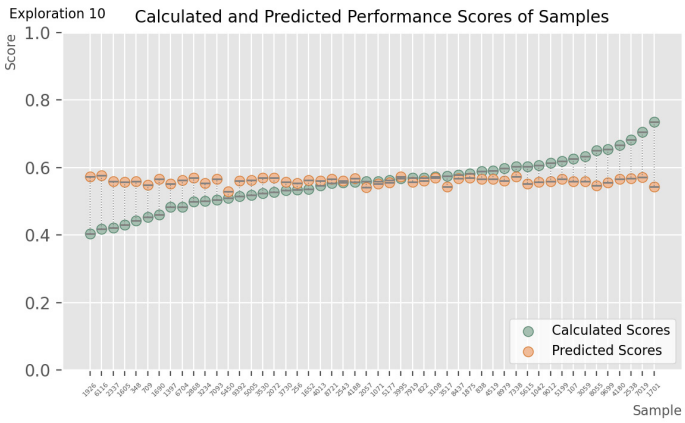
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	**Mean average error**
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

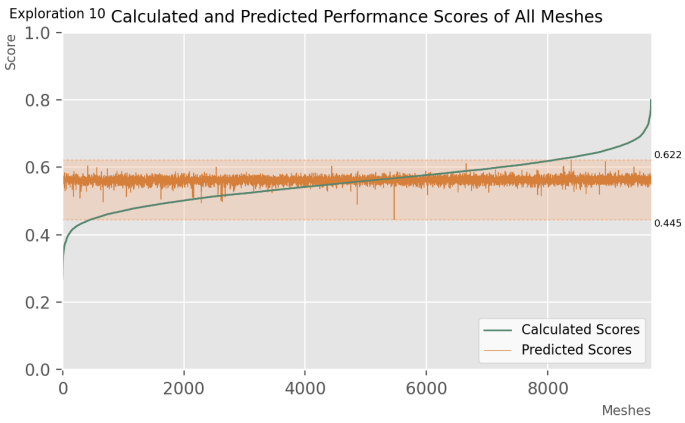
***NOTE:** For the above graph, limits of both axes were adapted to fit data.

***NOTE:** For the above graph, limits of both axes were adapted to fit data.





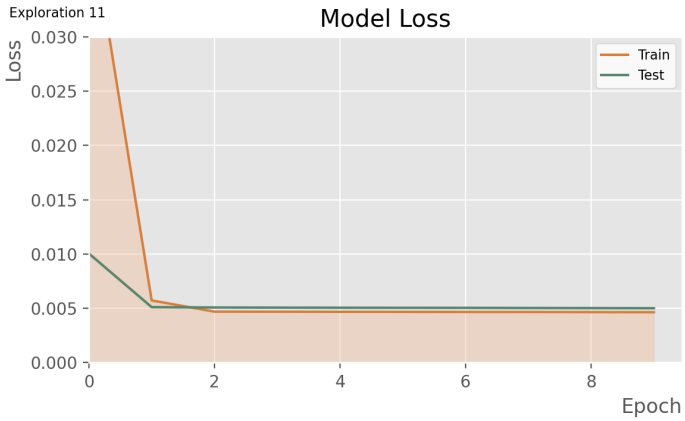




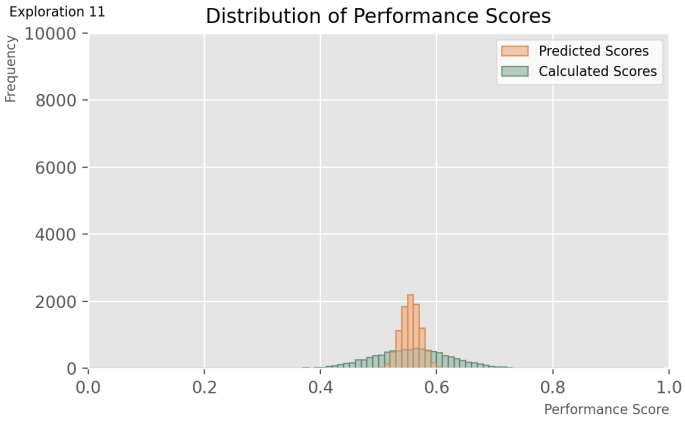
SURROGATE MODEL: EXPLORATION 11

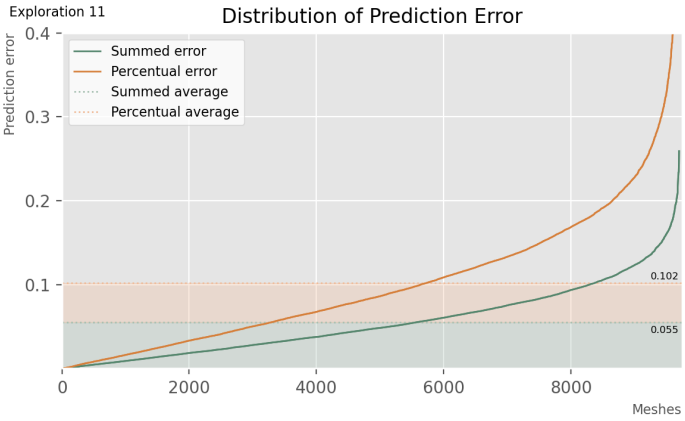
The attributes of this model are:

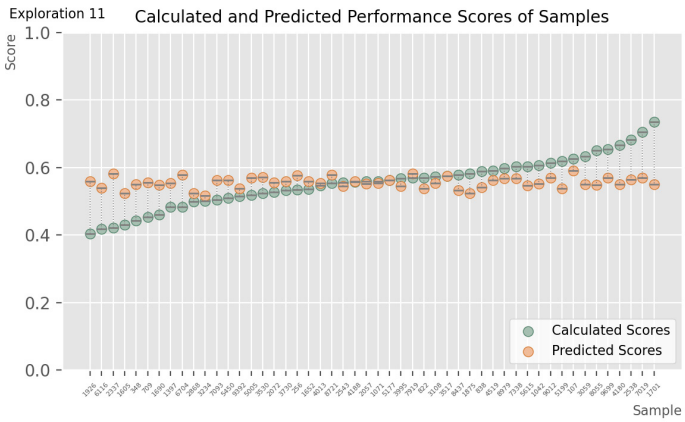
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	**Sigmoid**
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

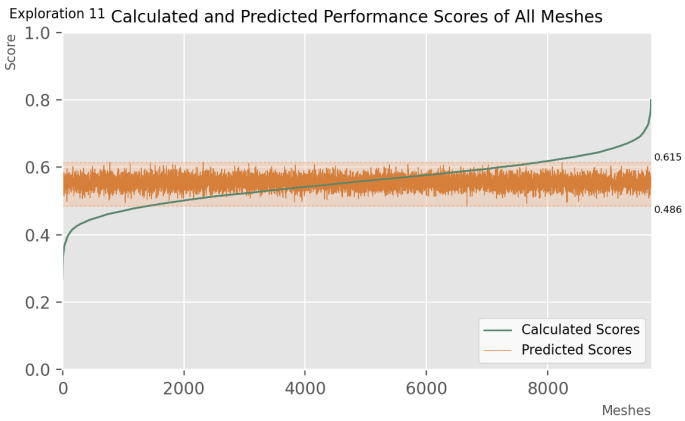


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





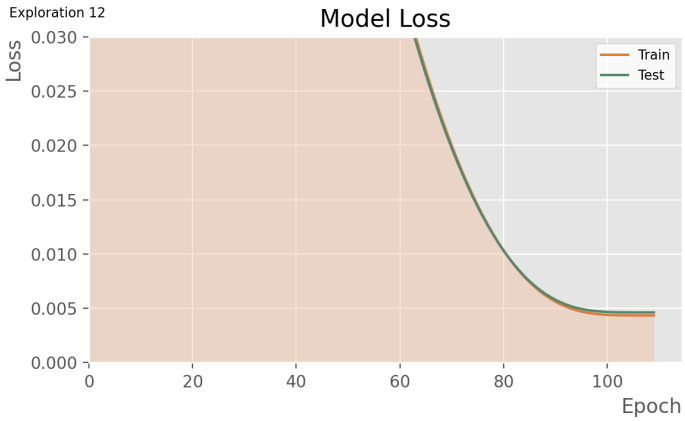




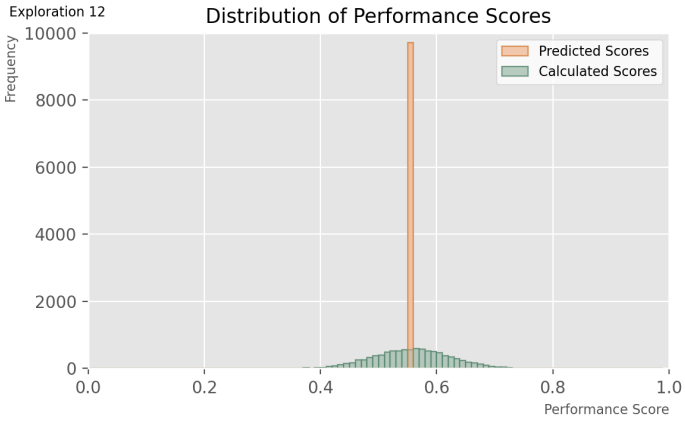
SURROGATE MODEL: EXPLORATION 12

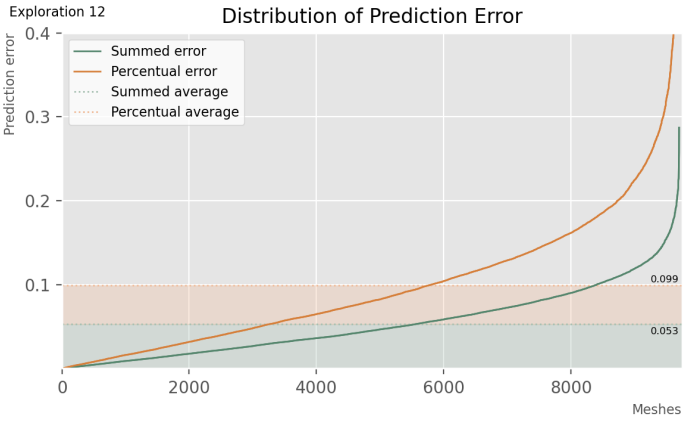
The attributes of this model are:

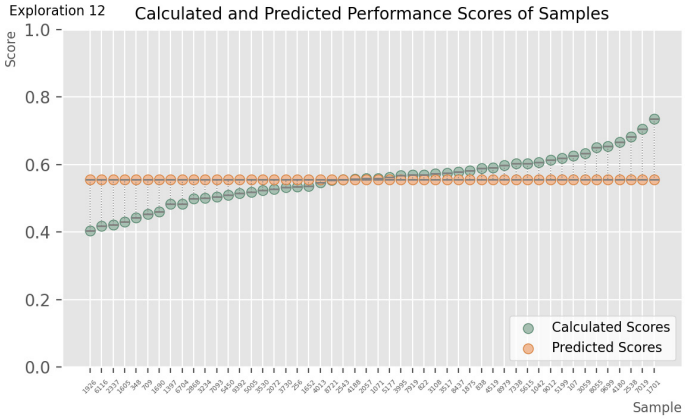
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	**Softmax**
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

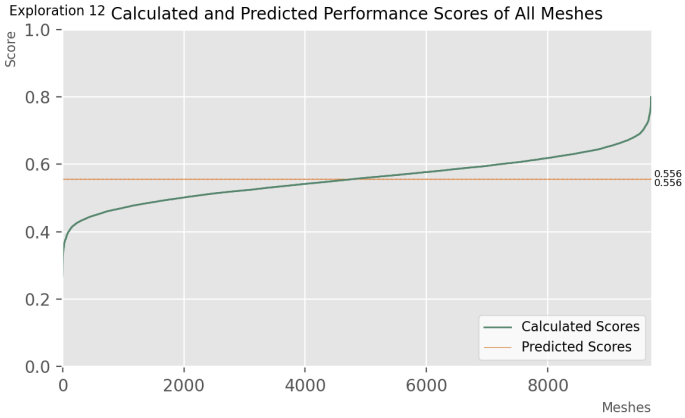


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





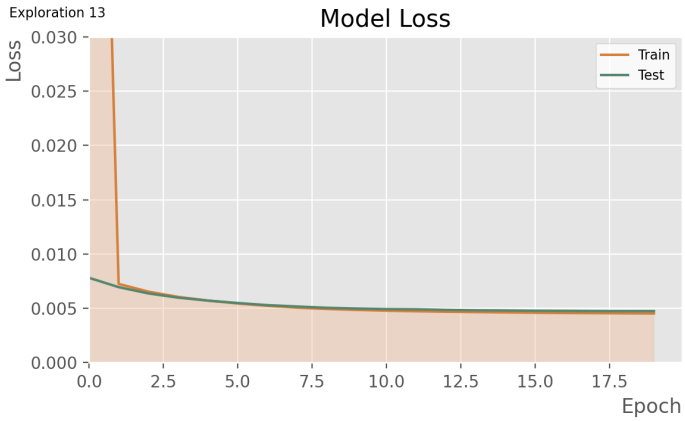




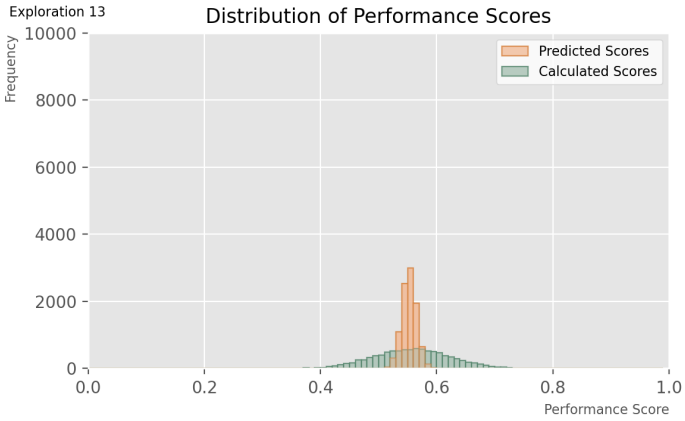
SURROGATE MODEL: EXPLORATION 13

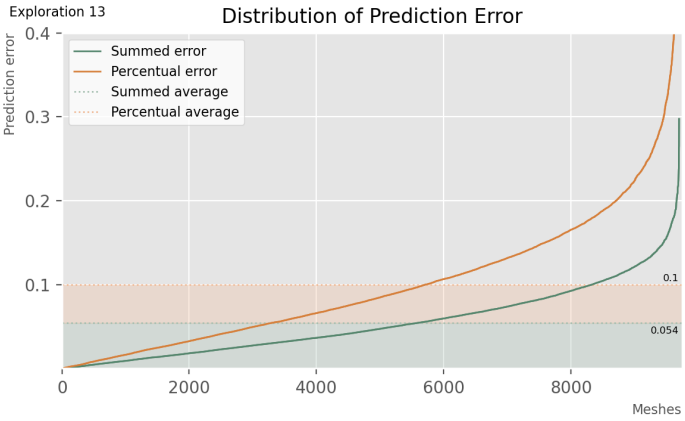
The attributes of this model are:

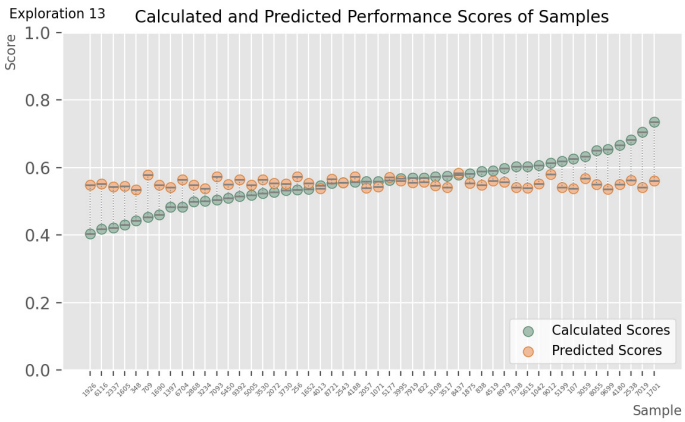
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	**Softsign**
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

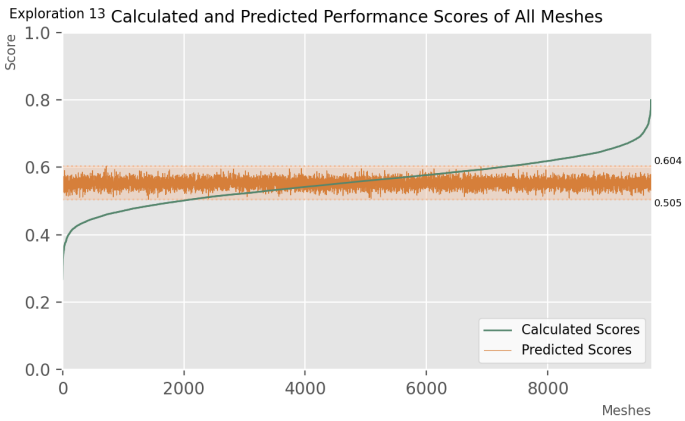


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





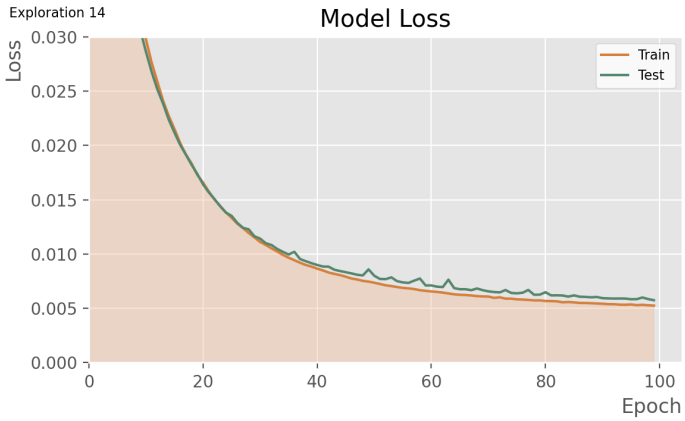




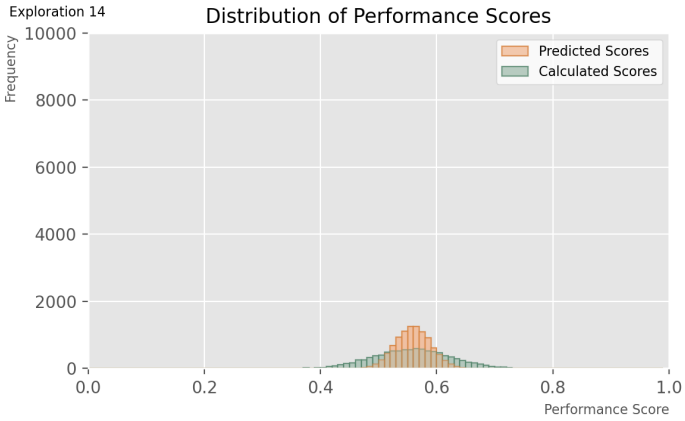
SURROGATE MODEL: EXPLORATION 14

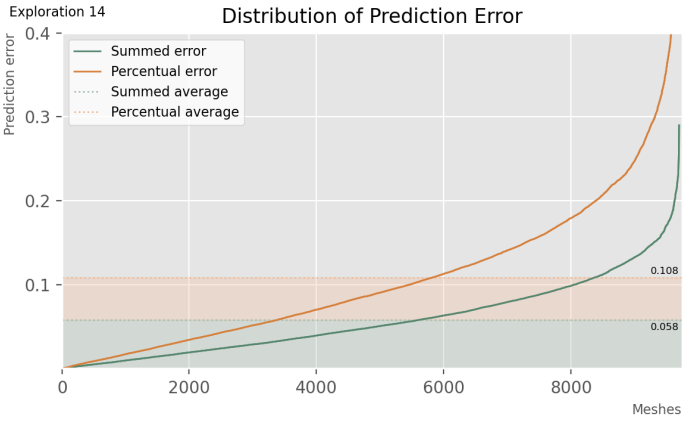
The attributes of this model are:

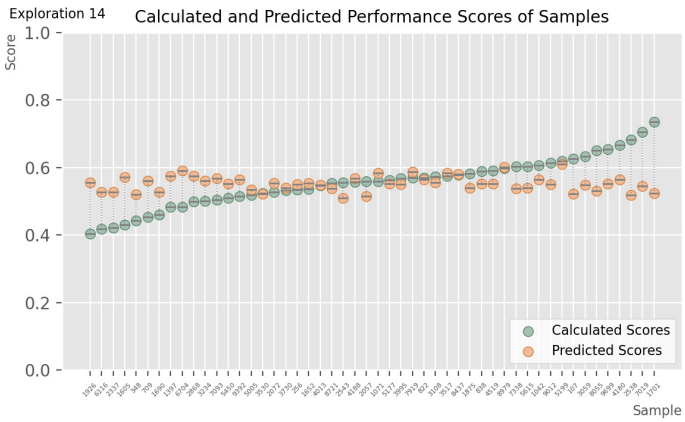
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	top_k_categorical_accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

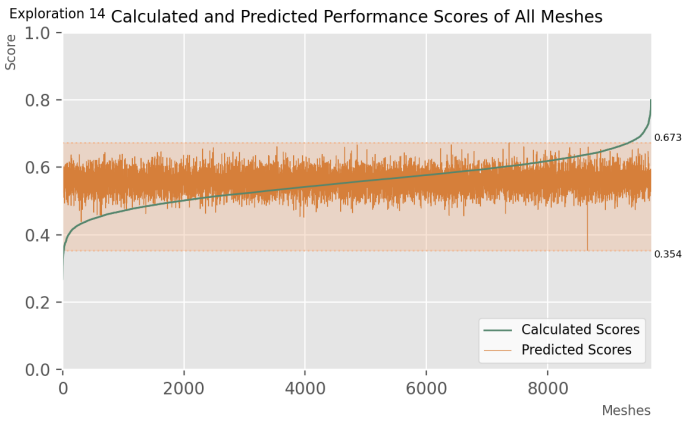


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.







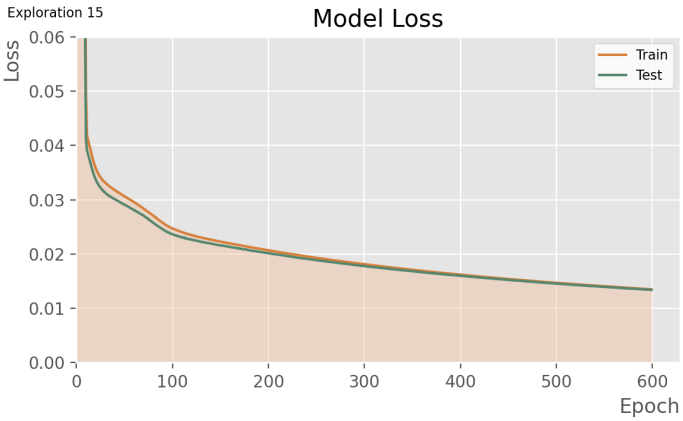


SURROGATE MODEL: EXPLORATION 15

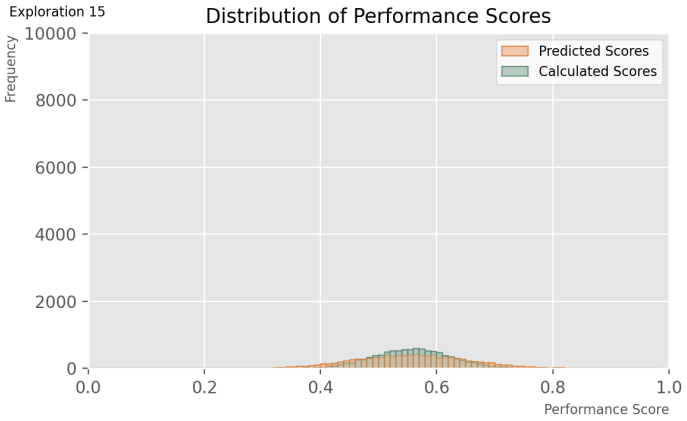
The attributes of this model are:

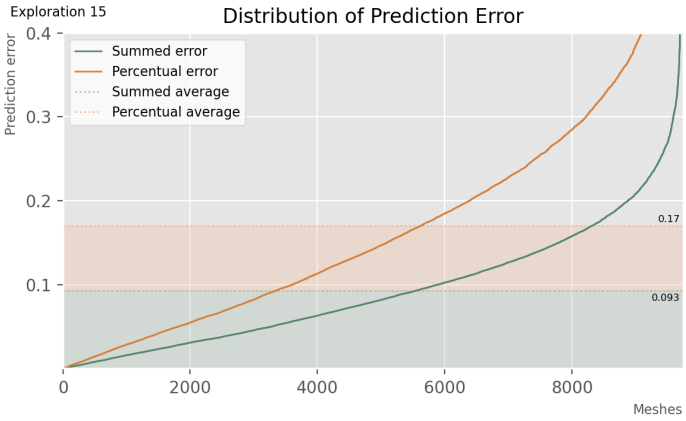
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	SGD
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

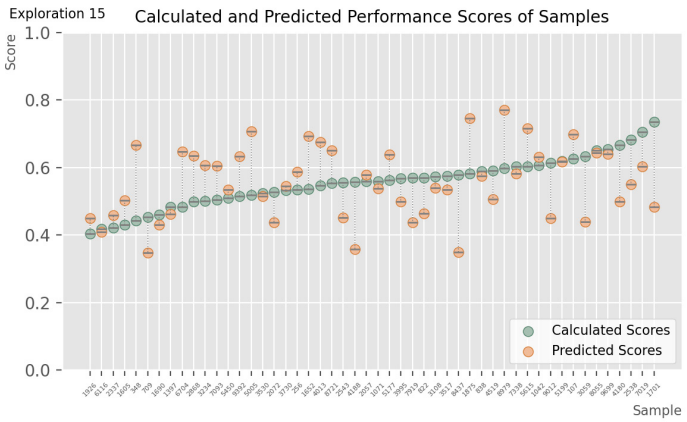
***NOTE:** For the above graph, limits of both axes were adapted to fit data.

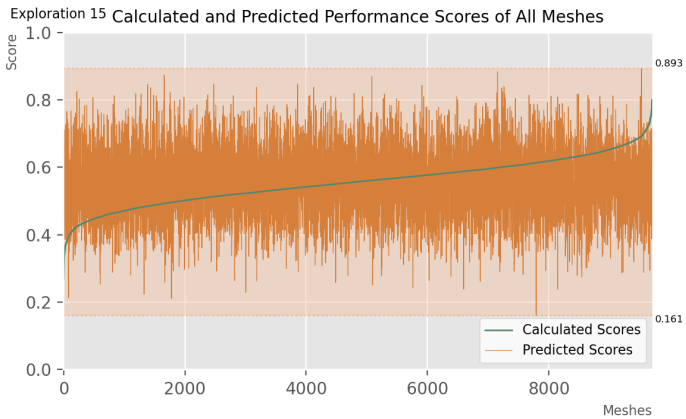


***NOTE:** For the above graph, limits of both axes were adapted to fit data.







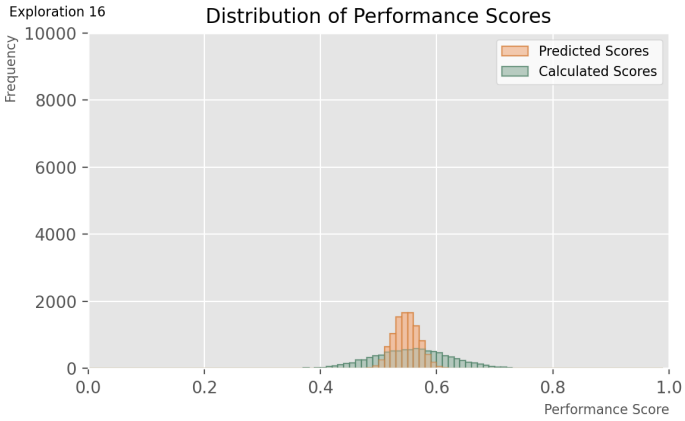


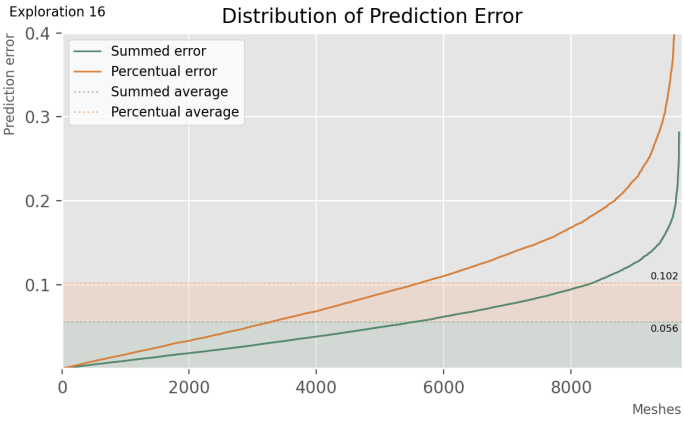
SURROGATE MODEL: EXPLORATION 16

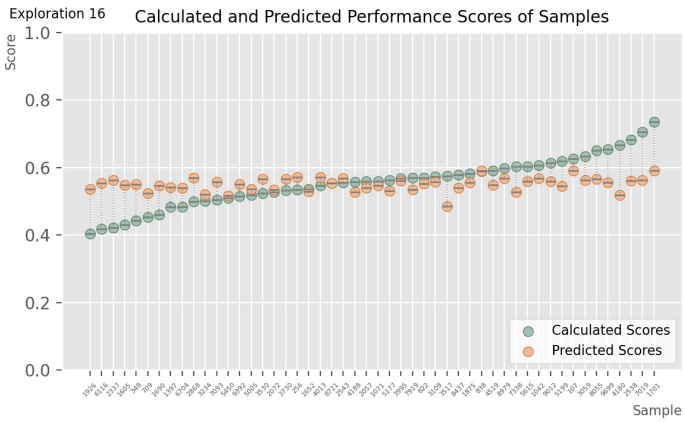
The attributes of this model are:

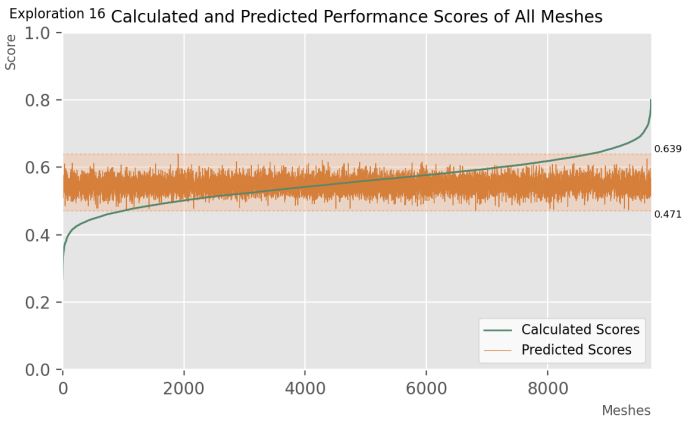
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	**RMSprop**
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





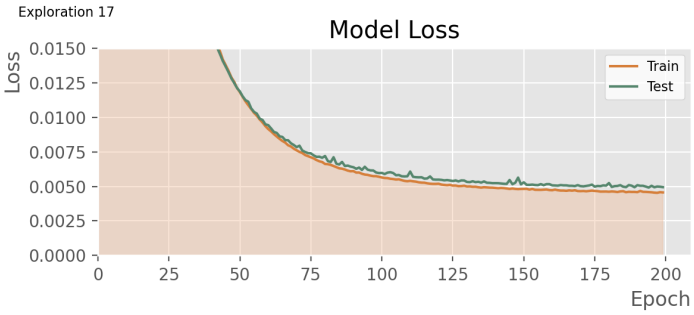


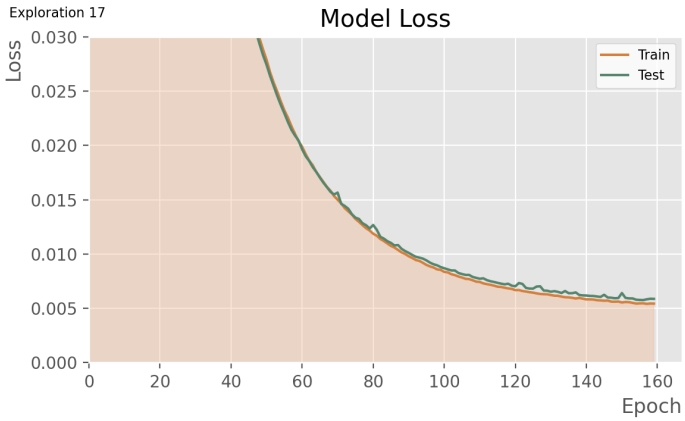


SURROGATE MODEL: EXPLORATION 17

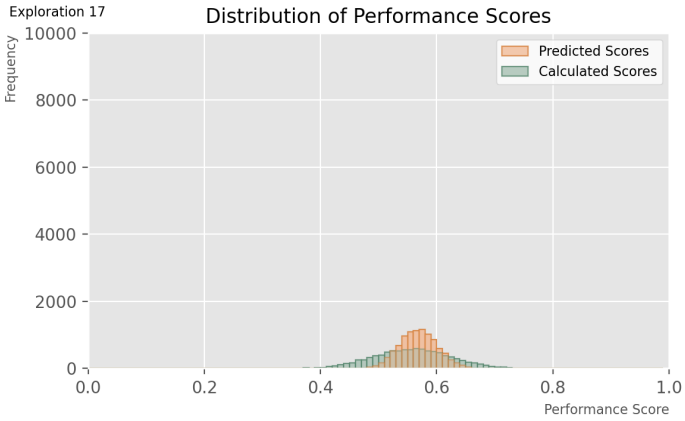
The attributes of this model are:

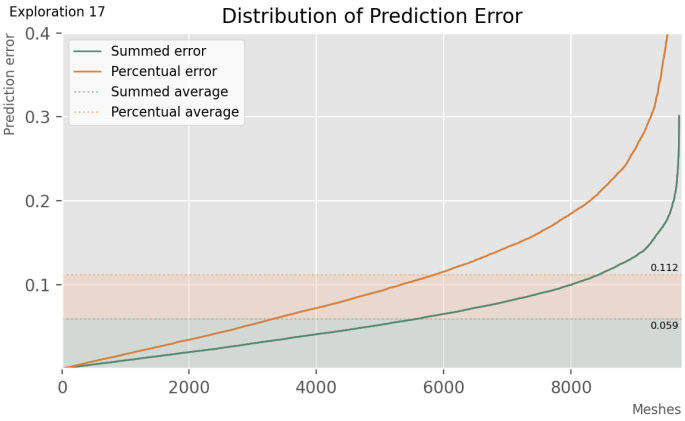
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	None
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

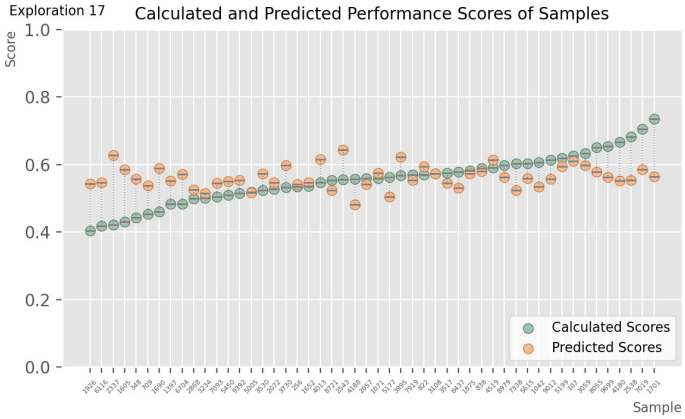


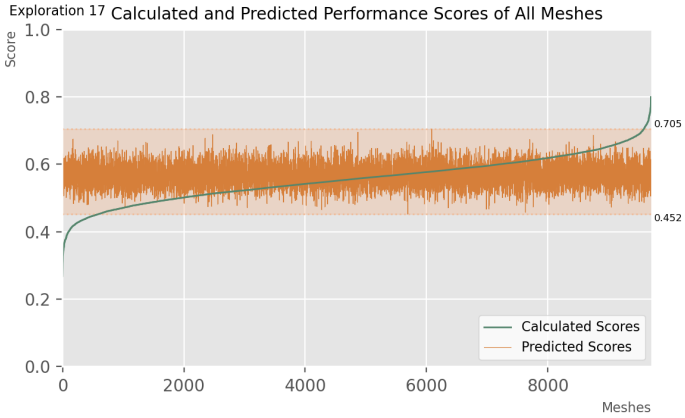


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.







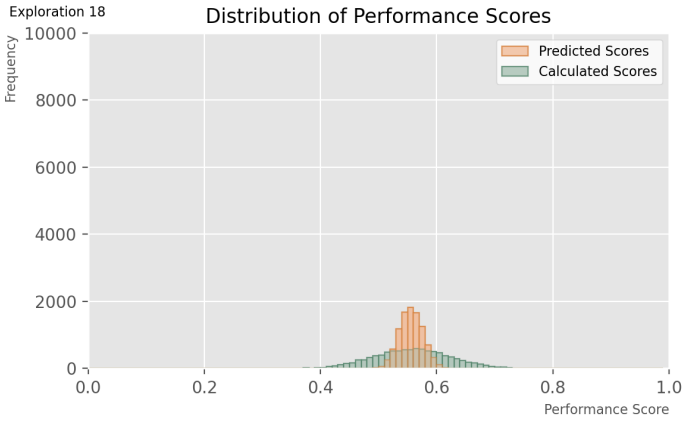


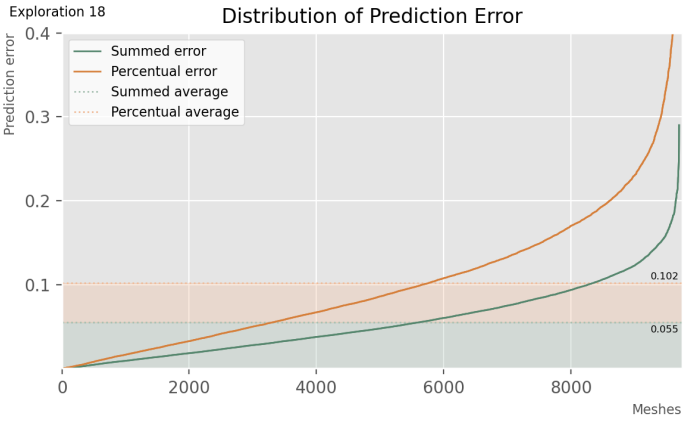
SURROGATE MODEL: EXPLORATION 18

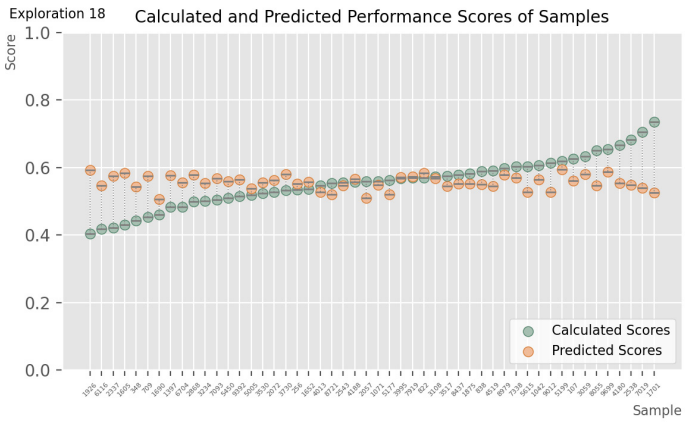
The attributes of this model are:

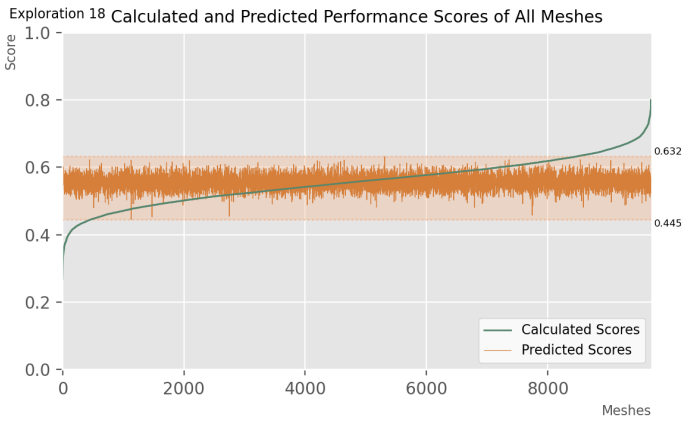
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	One additional normalization layer
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





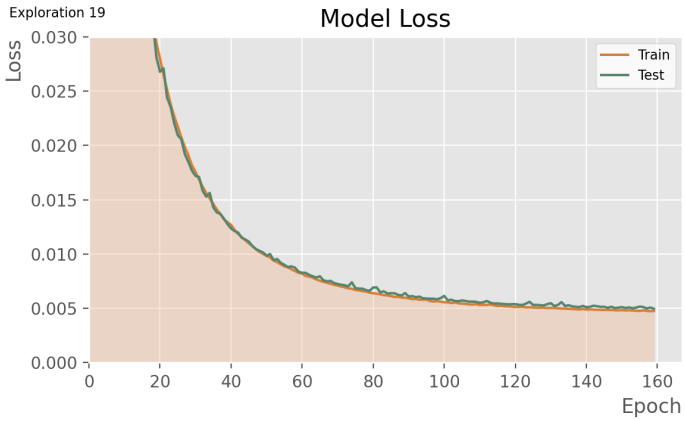




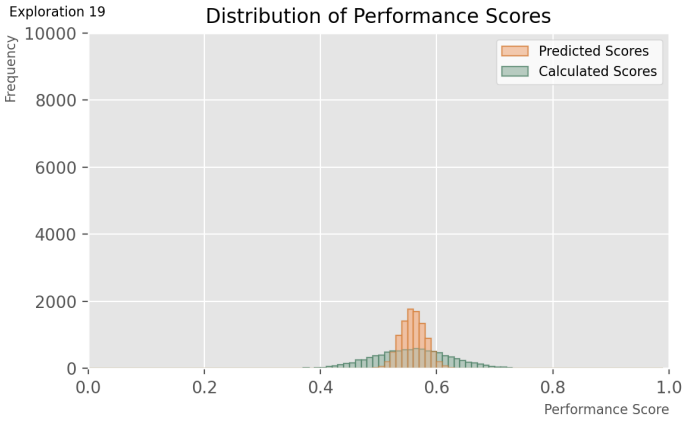
SURROGATE MODEL: EXPLORATION 19

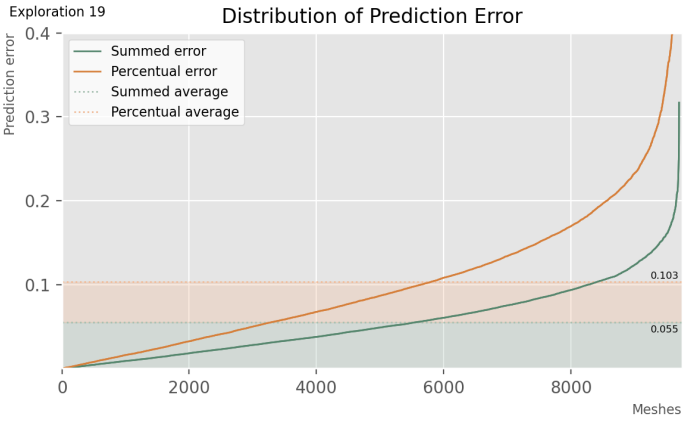
The attributes of this model are:

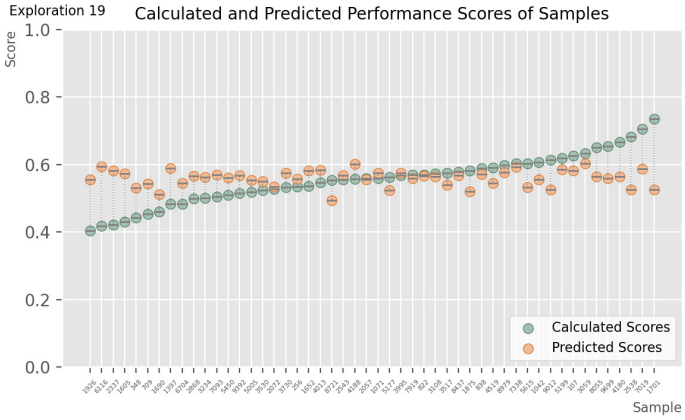
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	**Multiple** additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

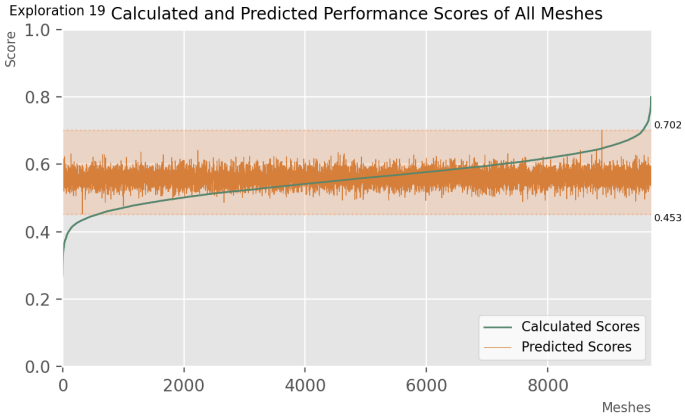


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





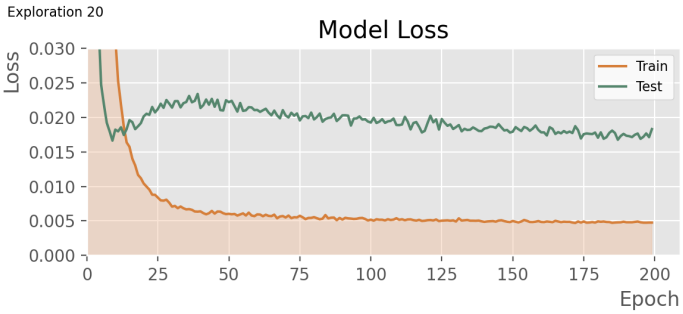




SURROGATE MODEL: EXPLORATION 20

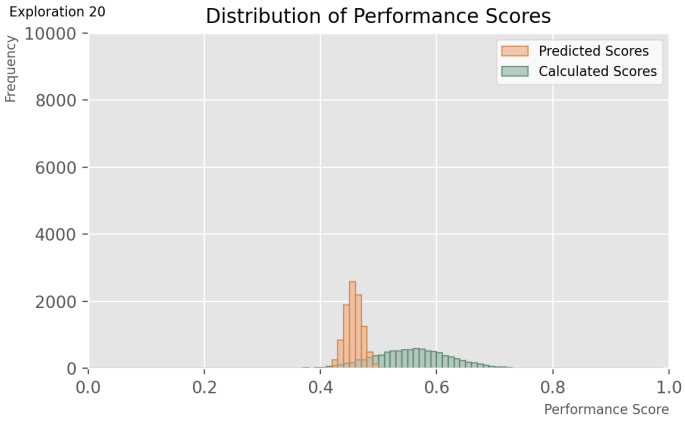
The attributes of this model are:

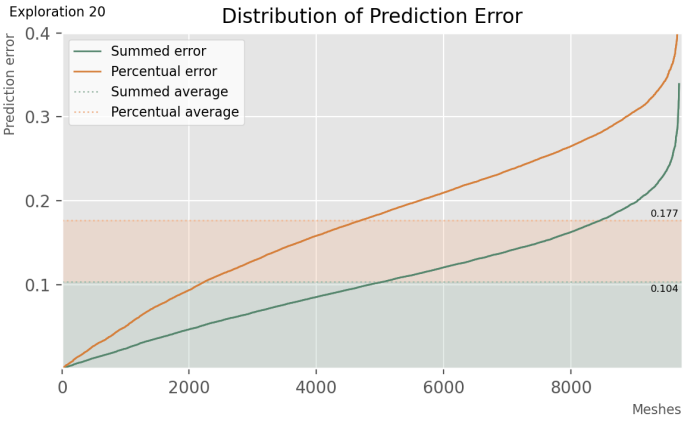
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	One dropout layers
Architecture:	'64-32-1'

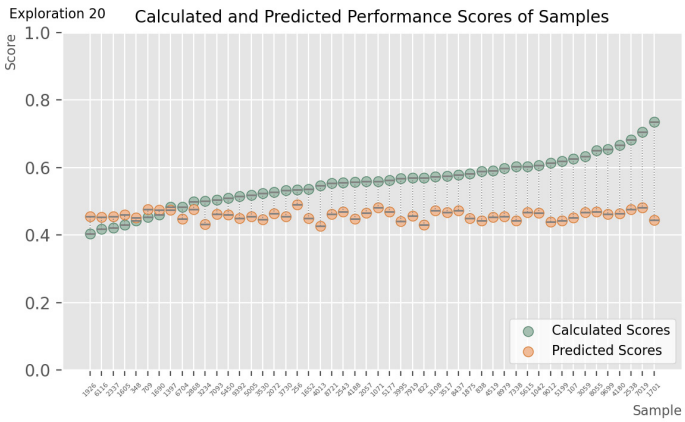


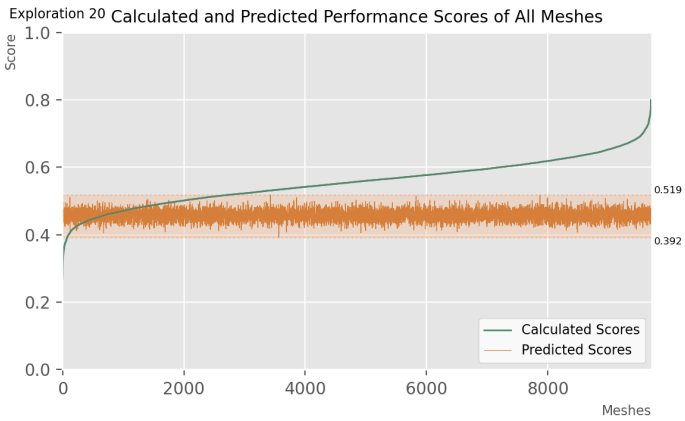
***NOTE:** For the above graph, limits of the 'loss' axis were adapted to fit data.

***NOTE:** For the above graph, limits of both axes were adapted to fit data.







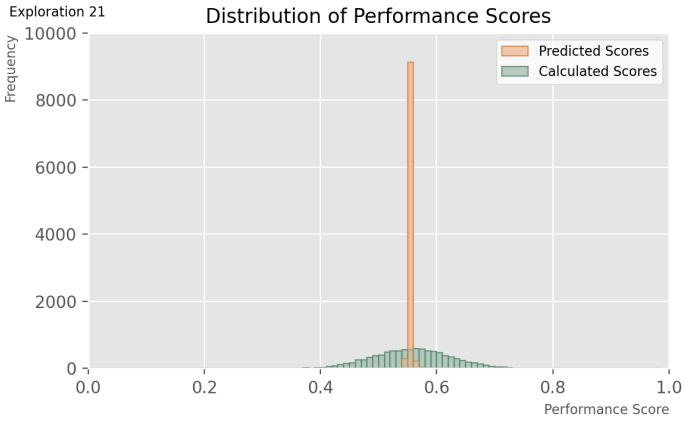


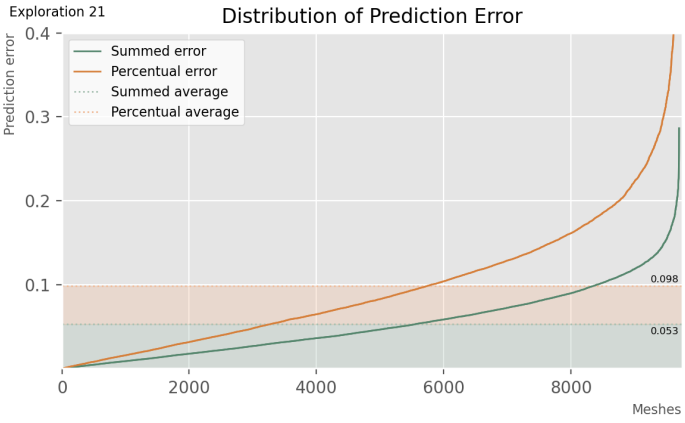
SURROGATE MODEL: EXPLORATION 21

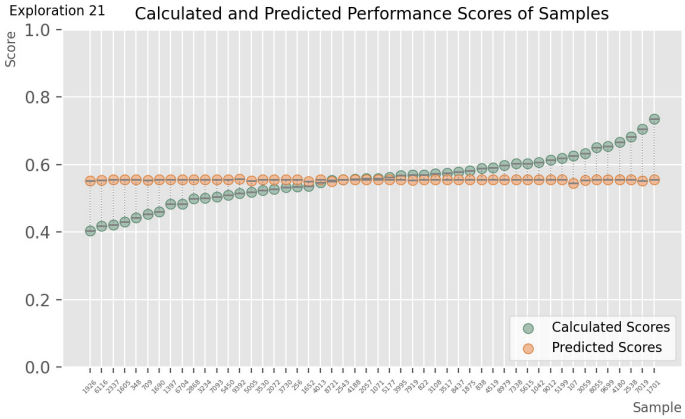
The attributes of this model are:

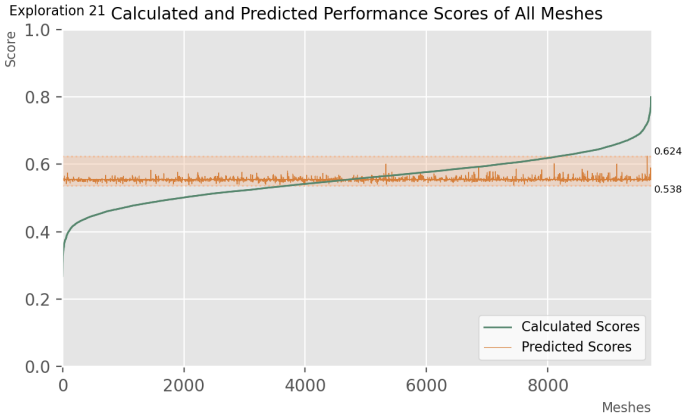
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	**Multiple** dropout layers
Architecture:	'64-32-1'

***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.







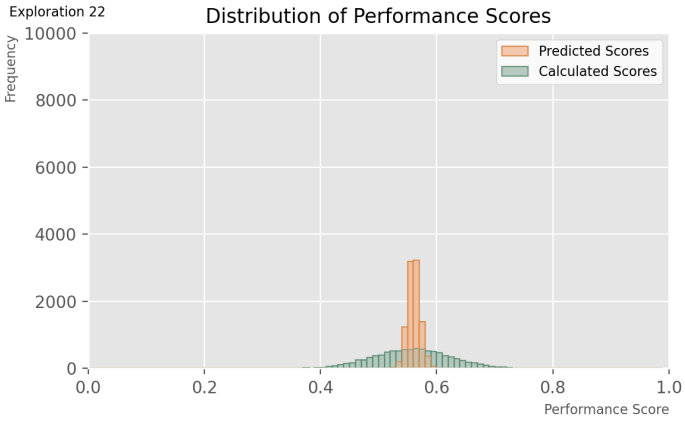


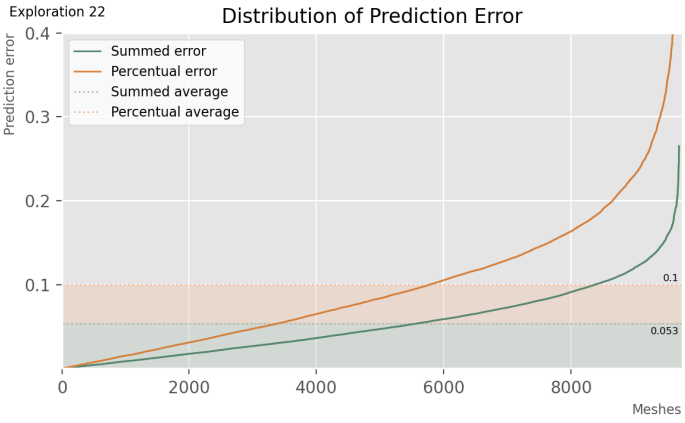
SURROGATE MODEL: EXPLORATION 22

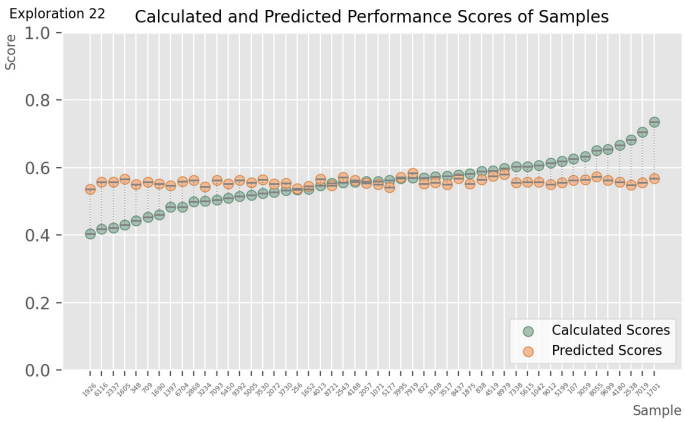
The attributes of this model are:

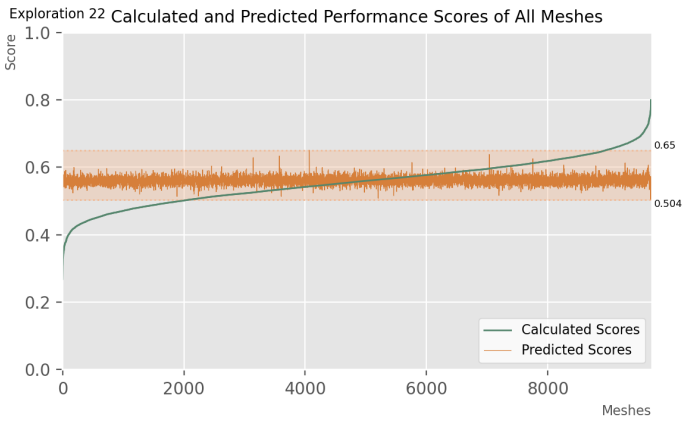
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	**'128-64-1'**

***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





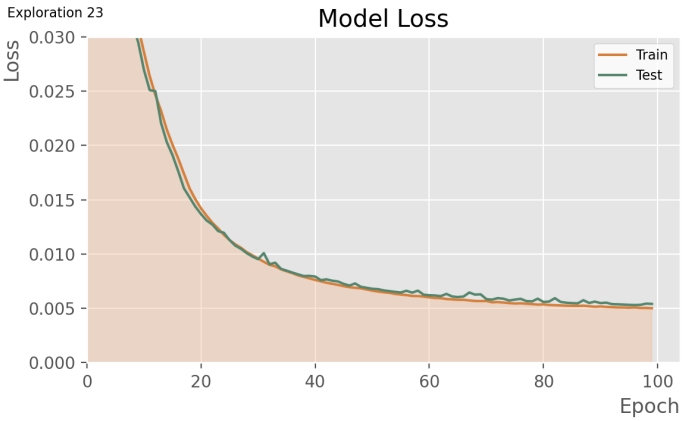




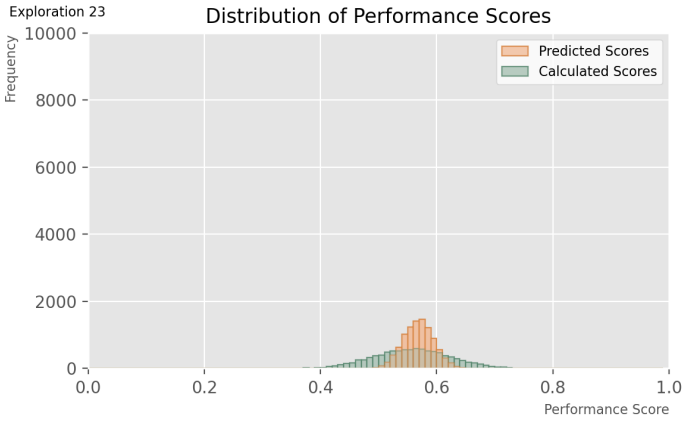
SURROGATE MODEL: EXPLORATION 23

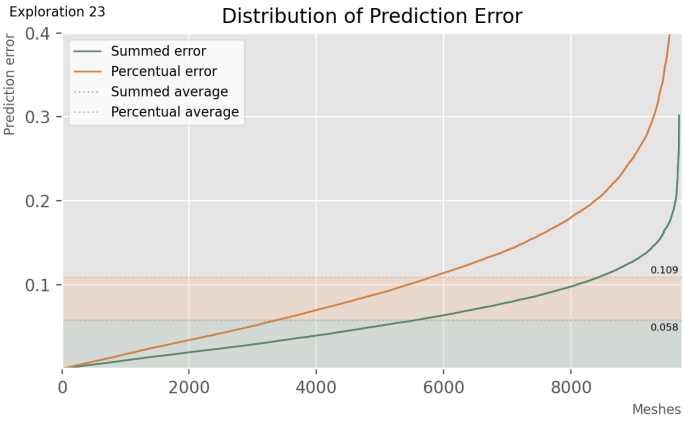
The attributes of this model are:

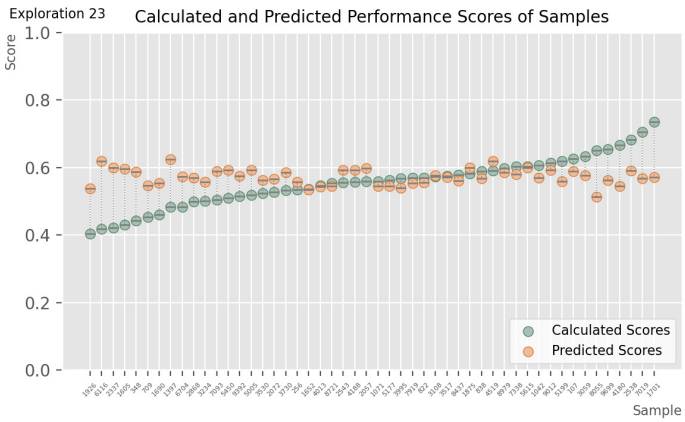
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	**'64-64-1'**

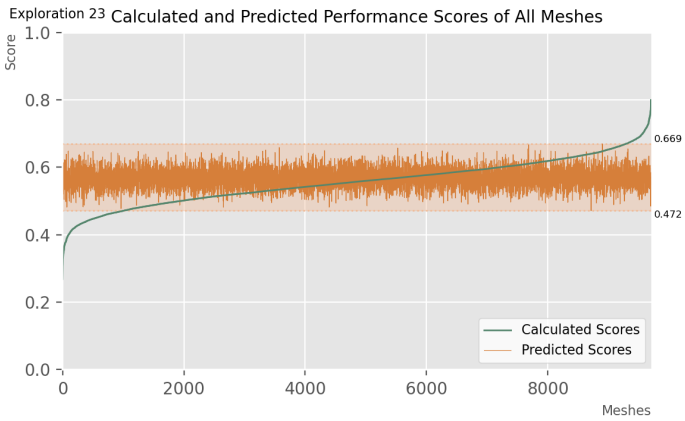


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.







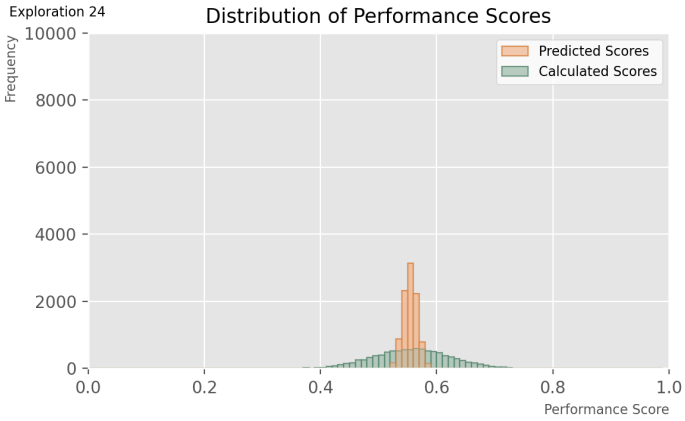


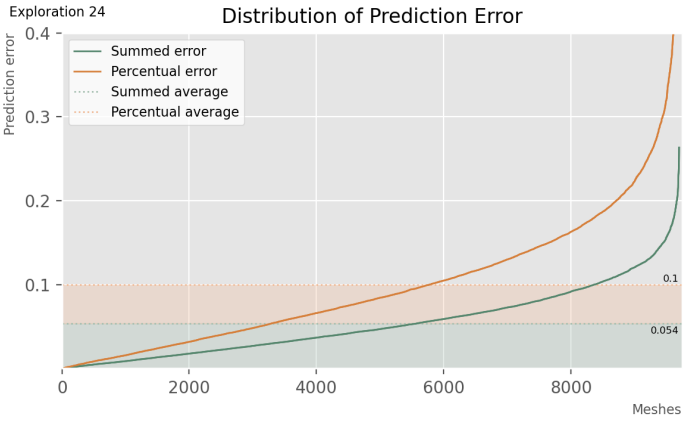
SURROGATE MODEL: EXPLORATION 24

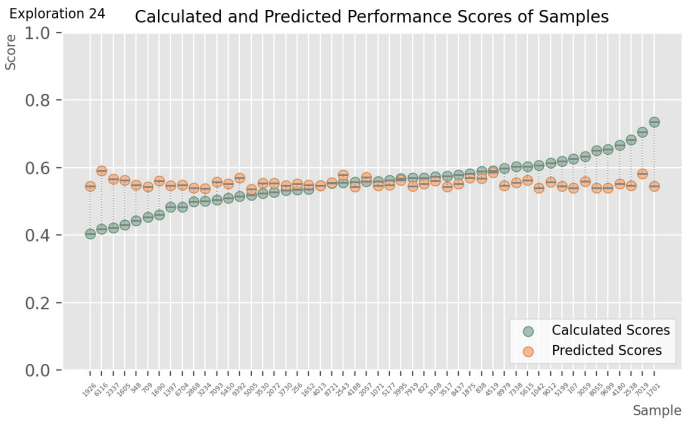
The attributes of this model are:

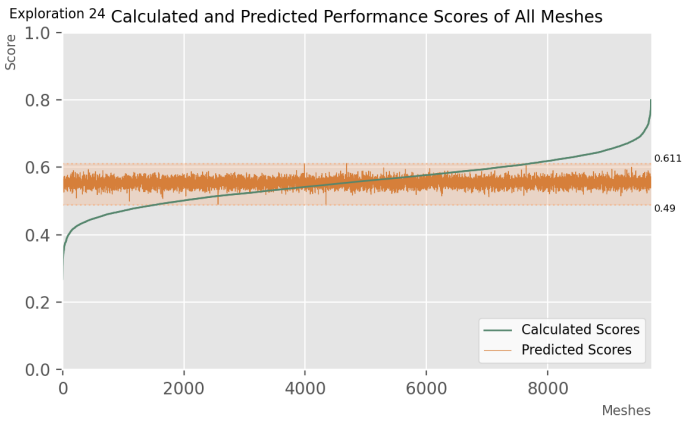
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	**'32-32-1'**

***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





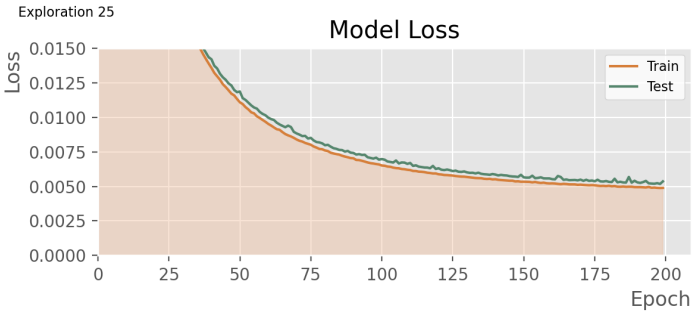


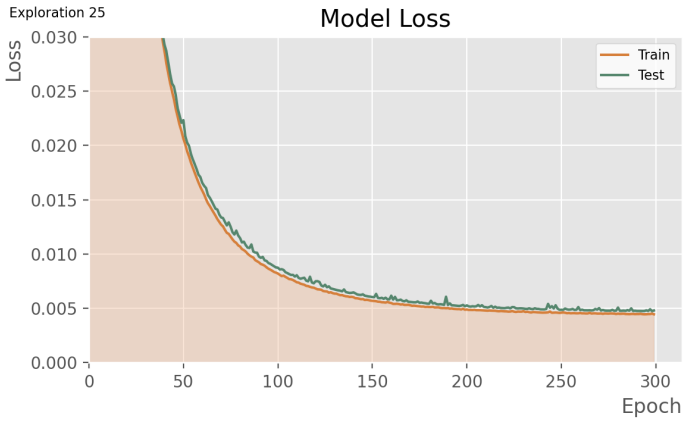


SURROGATE MODEL: EXPLORATION 25

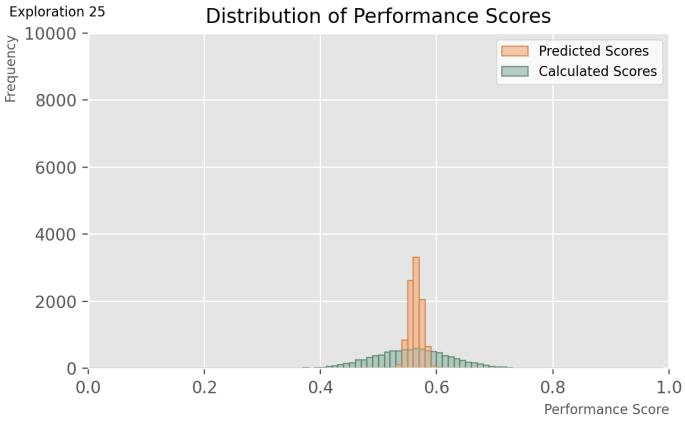
The attributes of this model are:

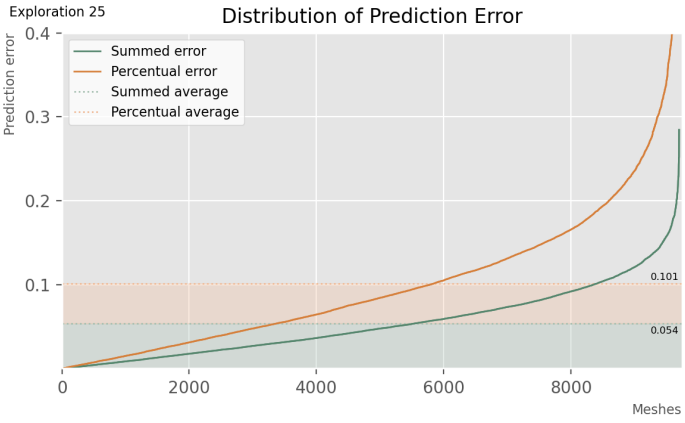
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	**'32-16-1'**

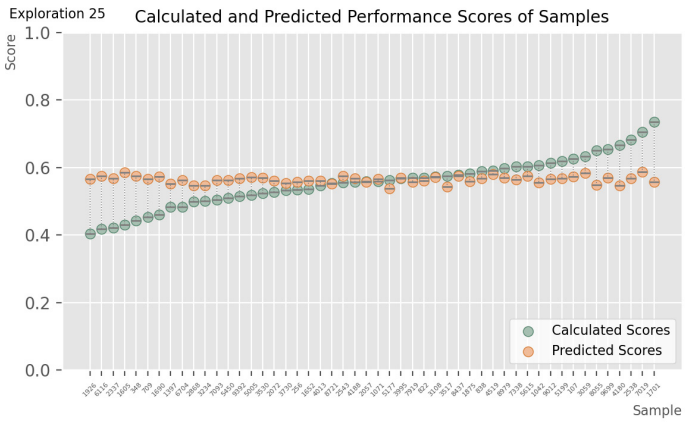


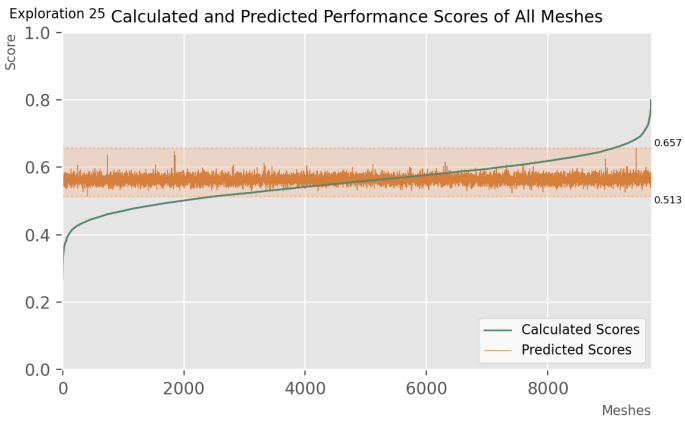


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





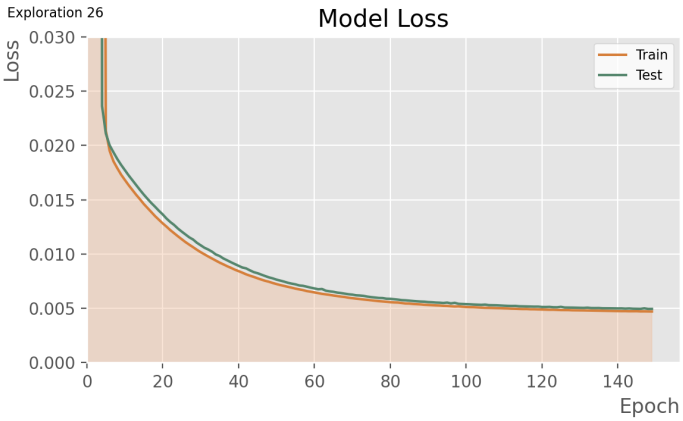




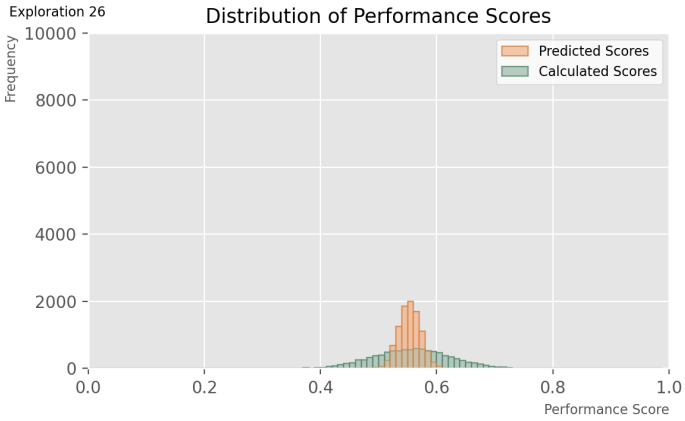
SURROGATE MODEL: EXPLORATION 26

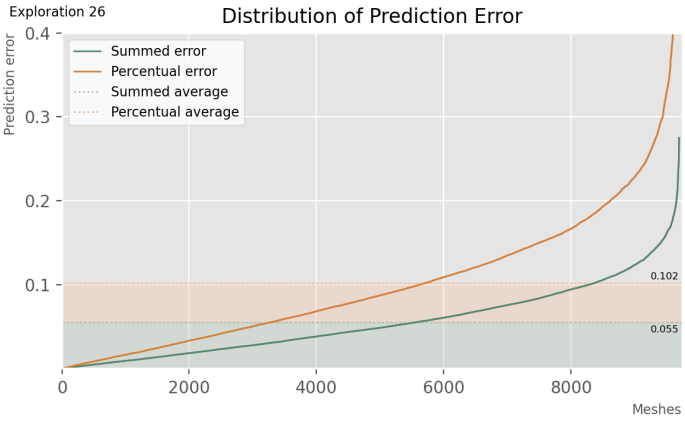
The attributes of this model are:

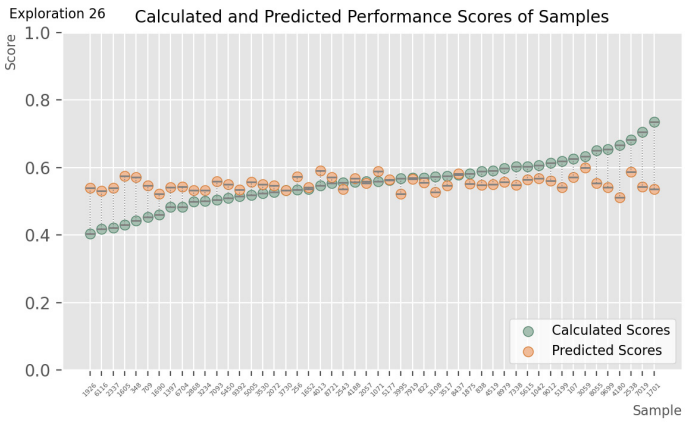
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	**'16-16-1'**

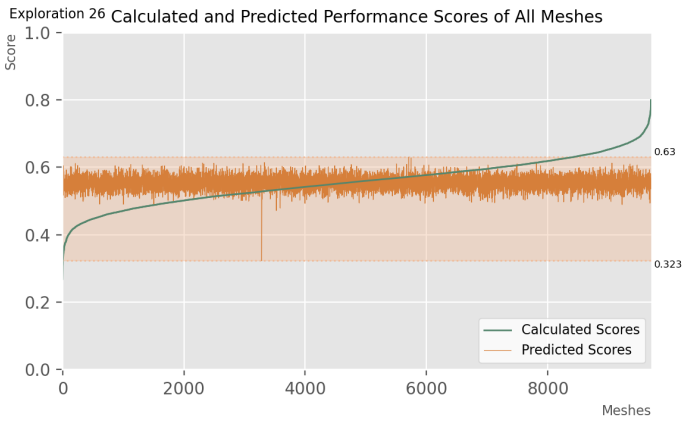


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





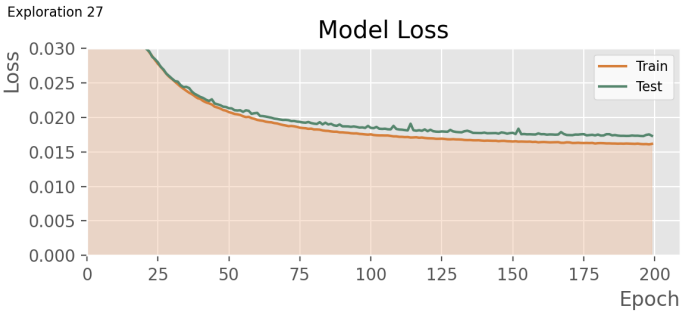




SURROGATE MODEL: EXPLORATION 27

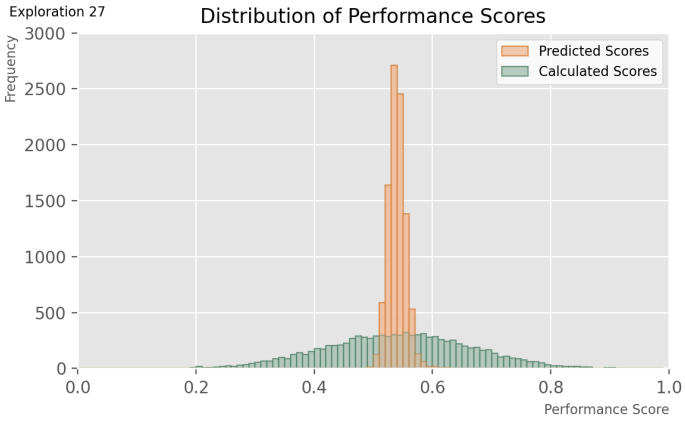
The attributes of this model are:

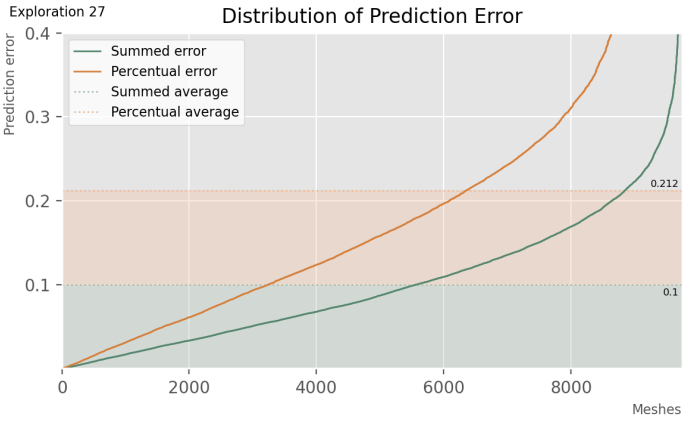
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

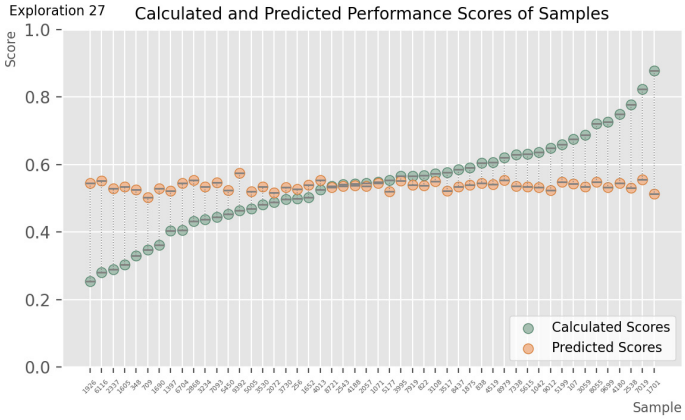


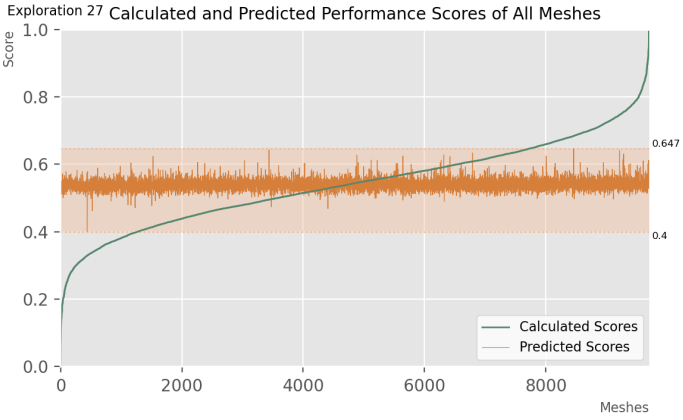
***NOTE:** For the above graph, limits of the 'loss' axis were adapted to fit data.

***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.







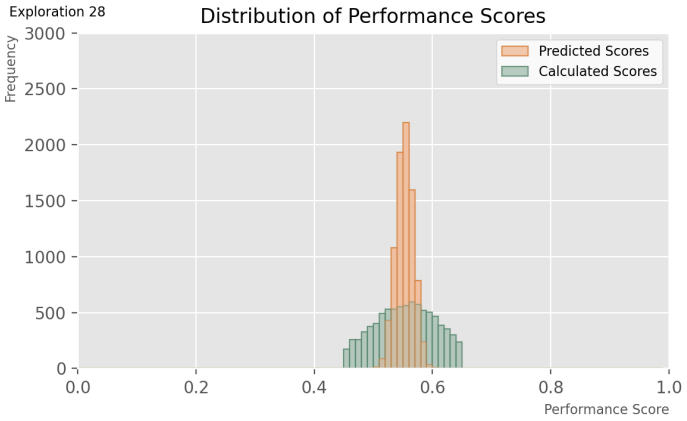


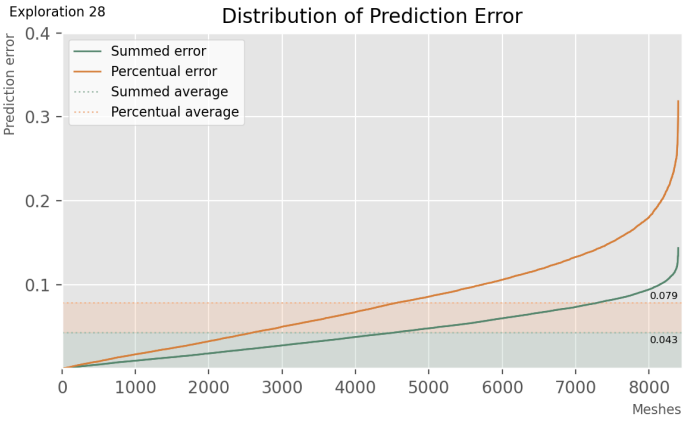
SURROGATE MODEL: EXPLORATION 28

The attributes of this model are:

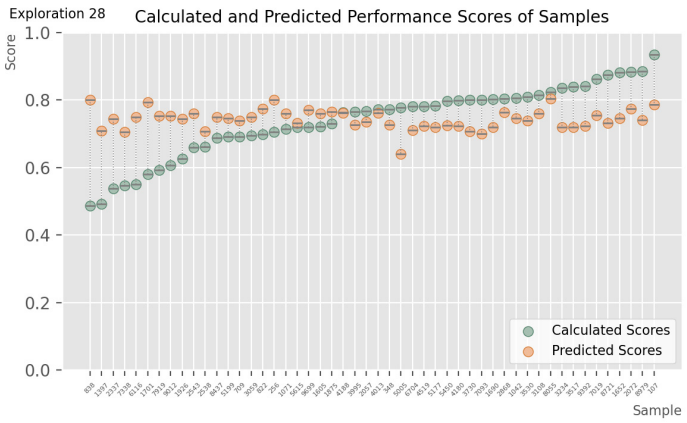
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

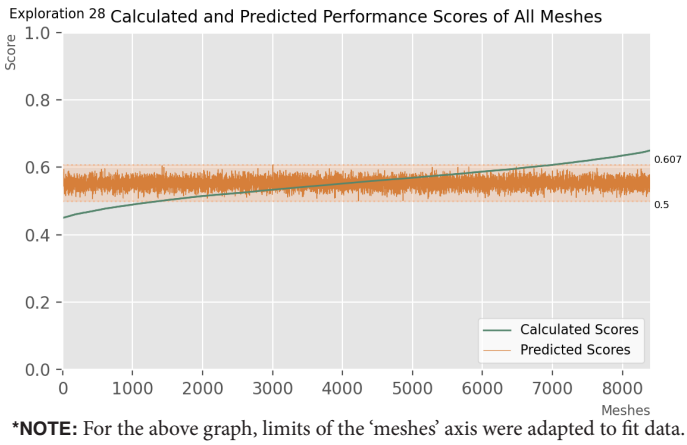
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





***NOTE:** For the above graph, limits of the 'meshes' axis were adapted to fit data.



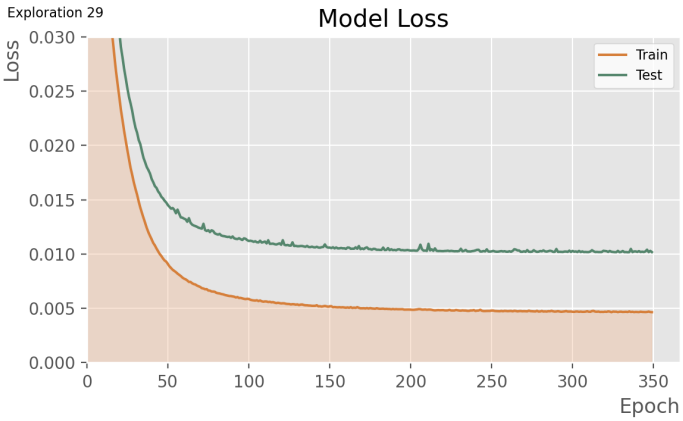


***NOTE:** For the above graph, limits of the 'meshes' axis were adapted to fit data.

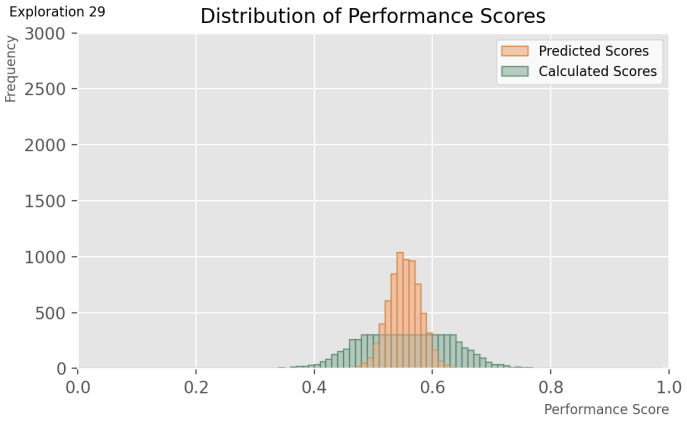
SURROGATE MODEL: EXPLORATION 29

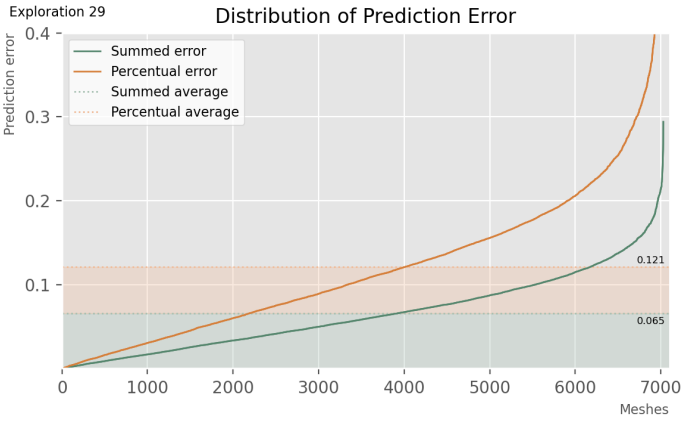
The attributes of this model are:

Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

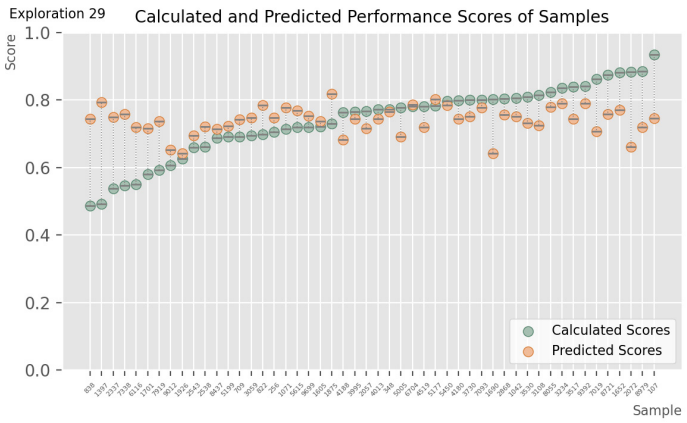


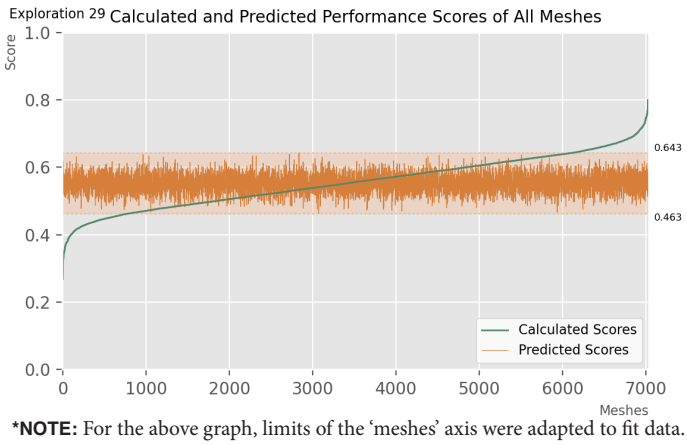
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





***NOTE:** For the above graph, limits of the 'meshes' axis were adapted to fit data.



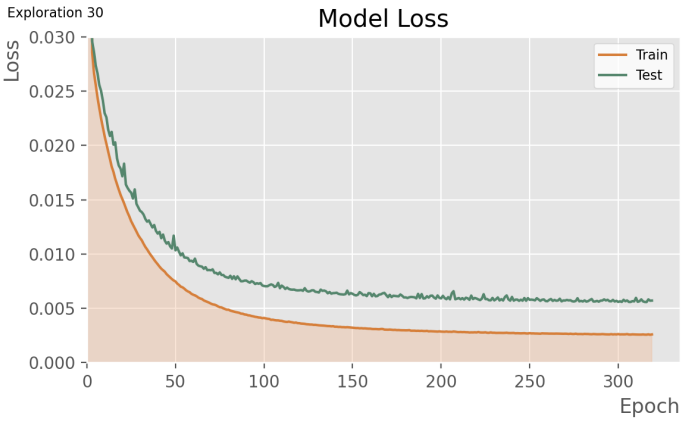


***NOTE:** For the above graph, limits of the 'meshes' axis were adapted to fit data.

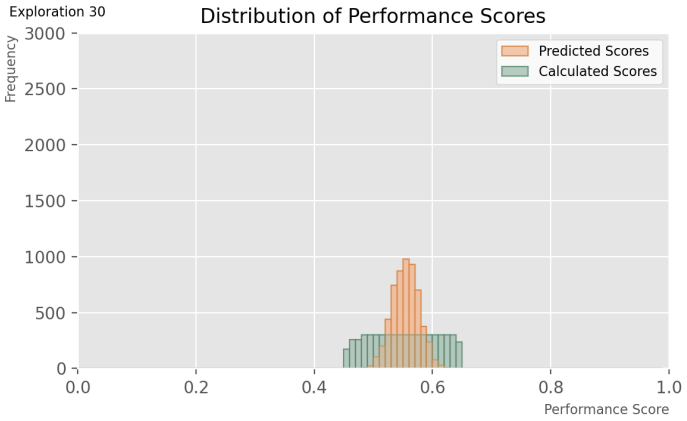
SURROGATE MODEL: EXPLORATION 30

The attributes of this model are:

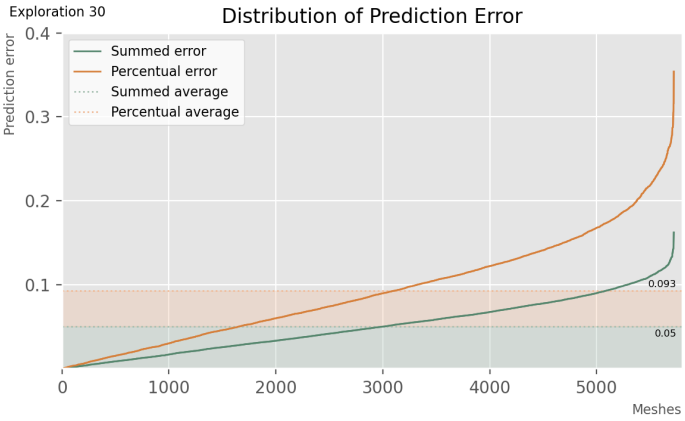
Batch size:	32
Epoch at which training is stopped:	At the point where validation loss stops decreasing
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'



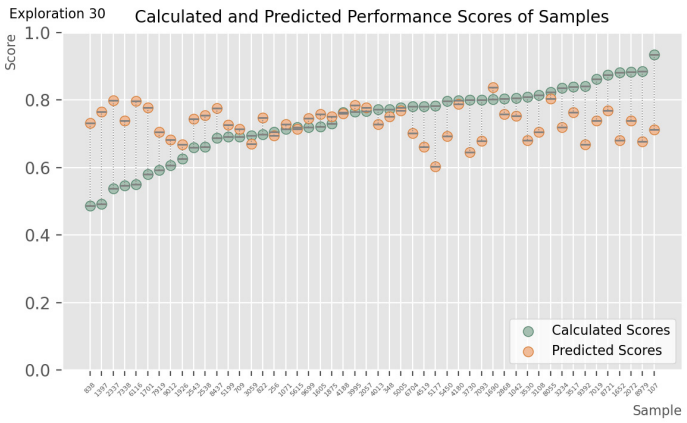
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.

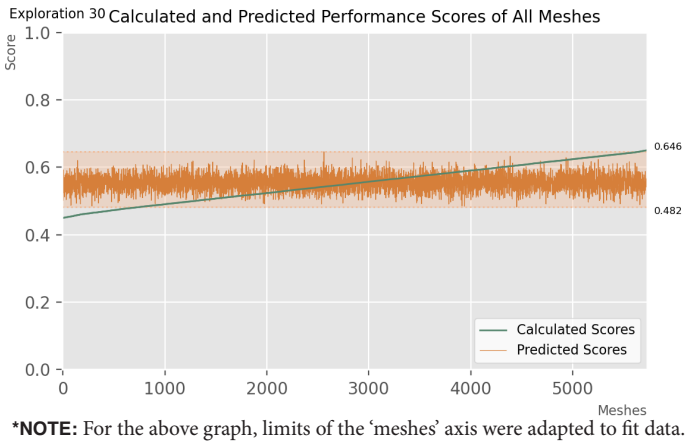


***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'meshes' axis were adapted to fit data.



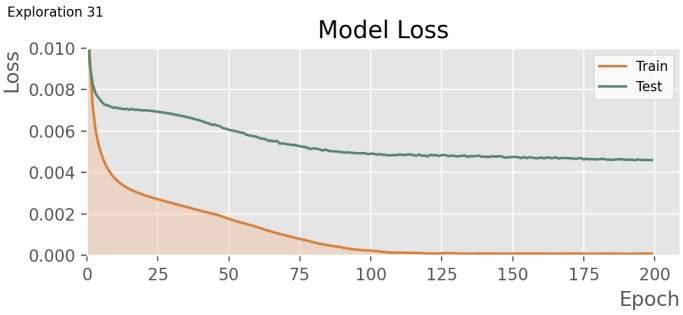


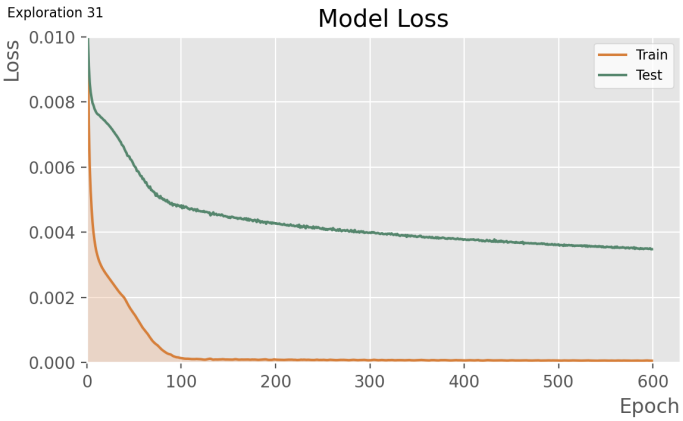
***NOTE:** For the above graph, limits of the 'meshes' axis were adapted to fit data.

SURROGATE MODEL: EXPLORATION 31

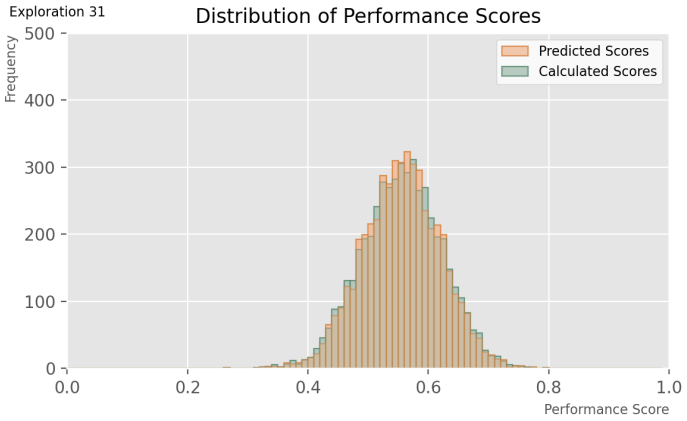
The attributes of this model are:

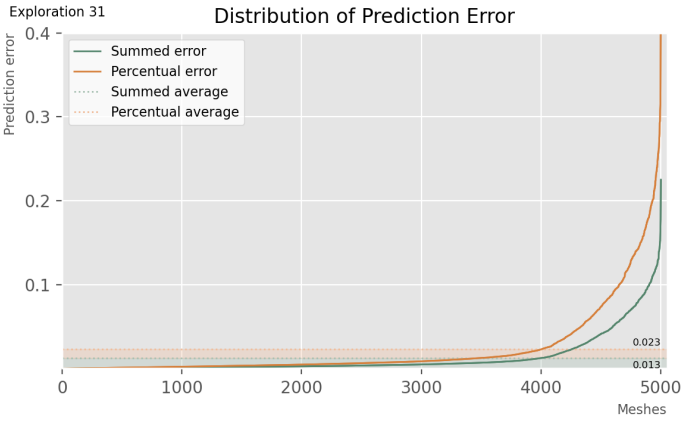
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

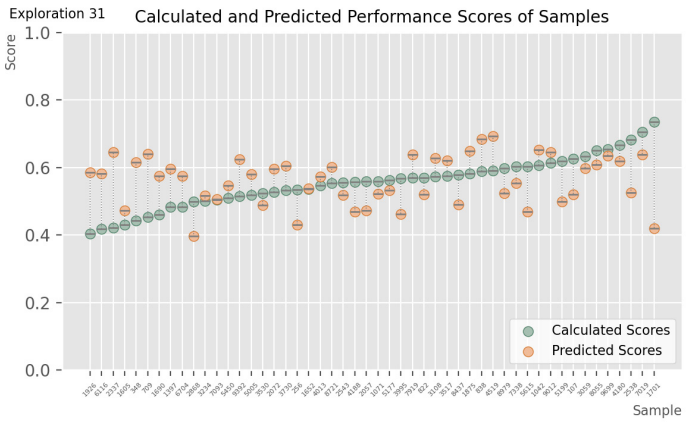


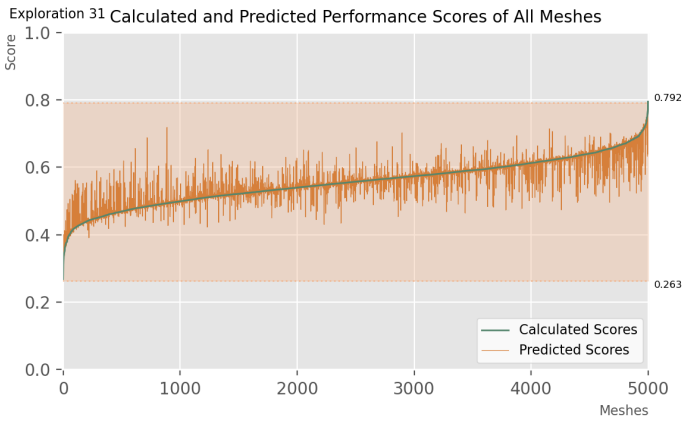


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





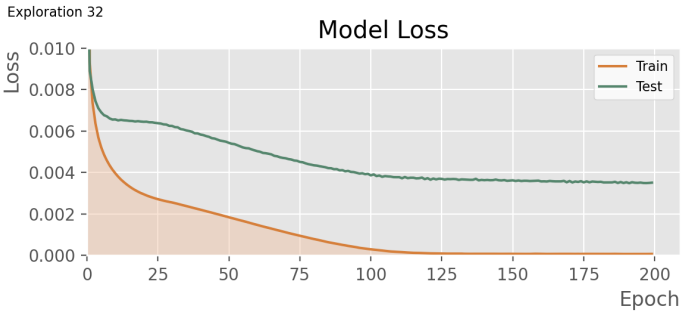


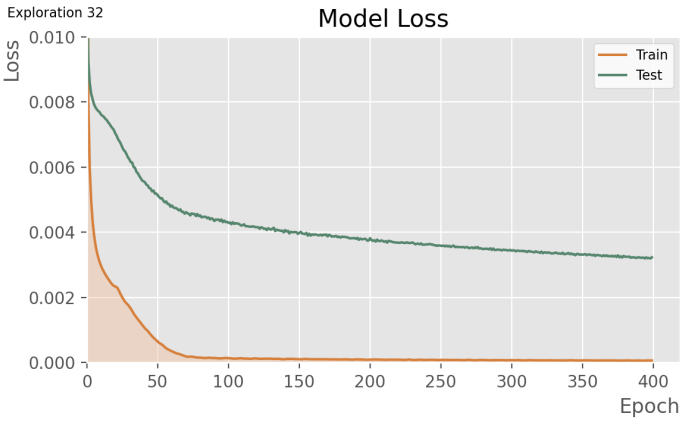


SURROGATE MODEL: EXPLORATION 32

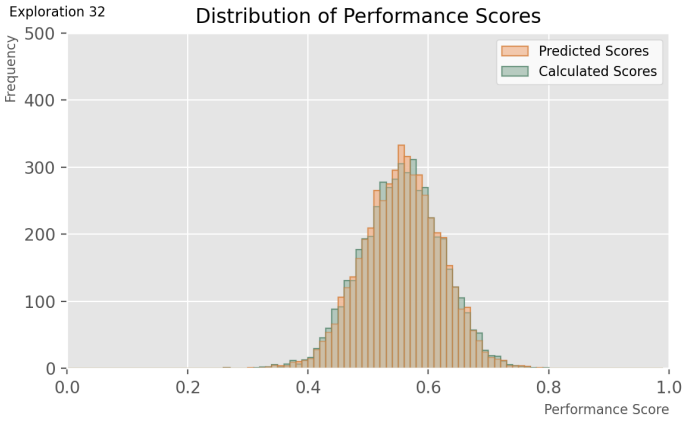
The attributes of this model are:

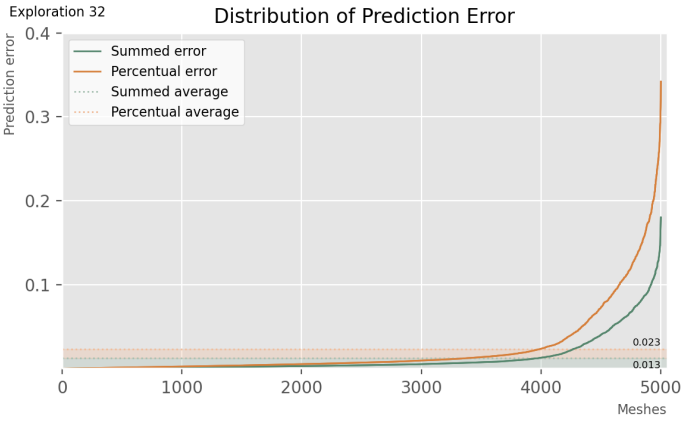
Batch size:	**16**
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

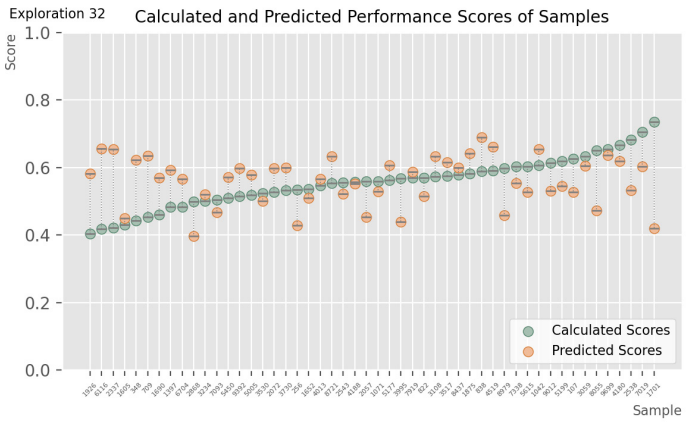


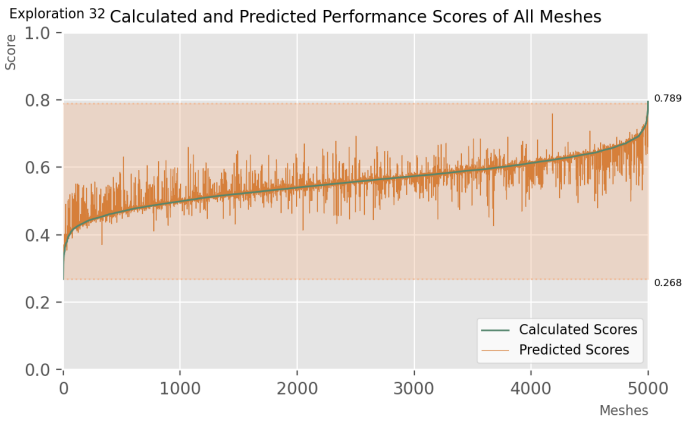


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





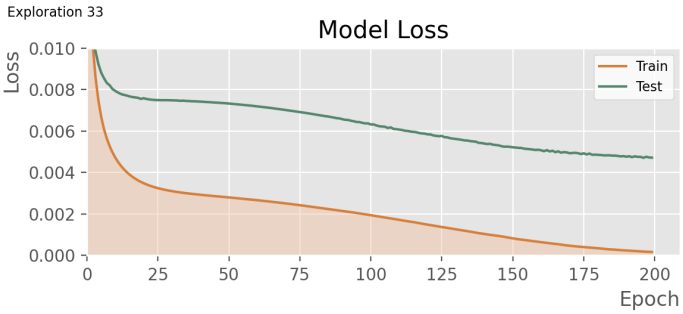




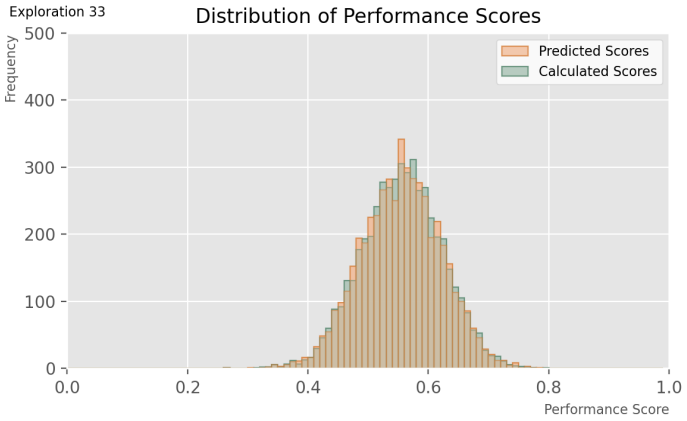
SURROGATE MODEL: EXPLORATION 33

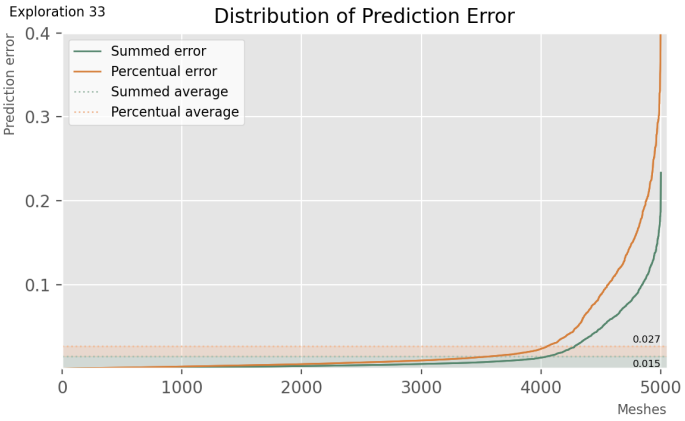
The attributes of this model are:

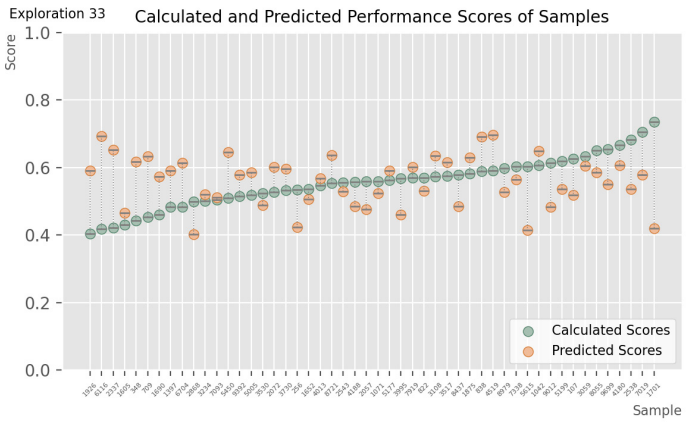
Batch size:	64
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

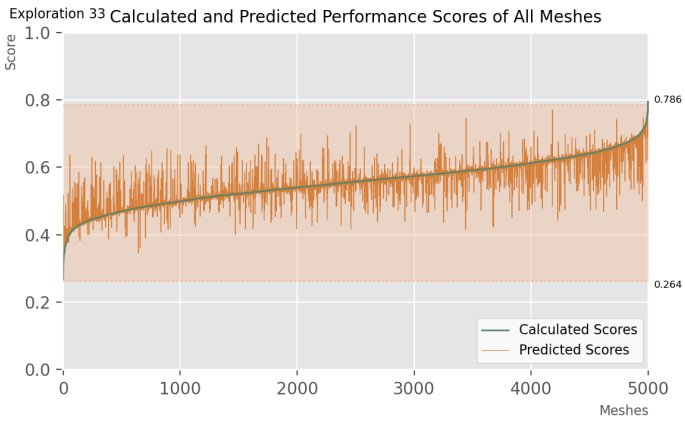


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





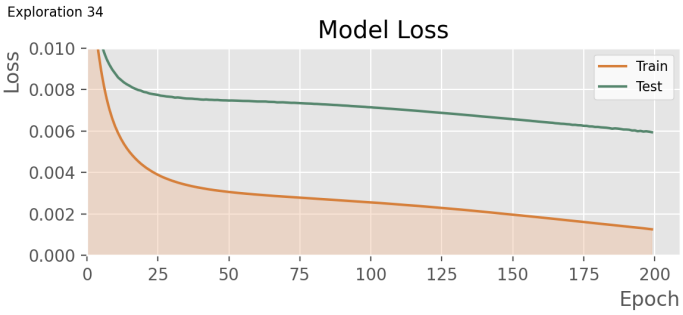




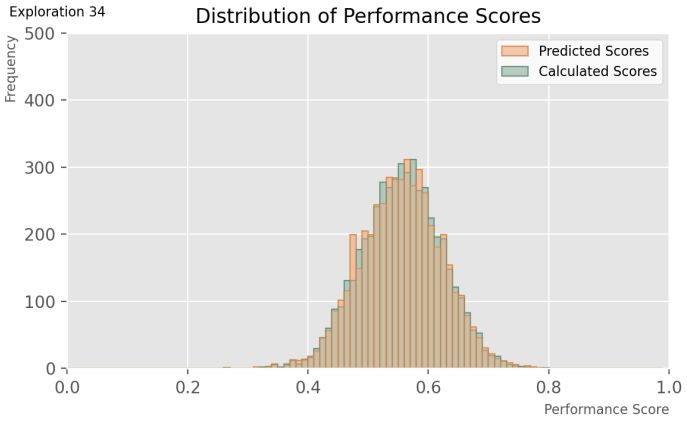
SURROGATE MODEL: EXPLORATION 34

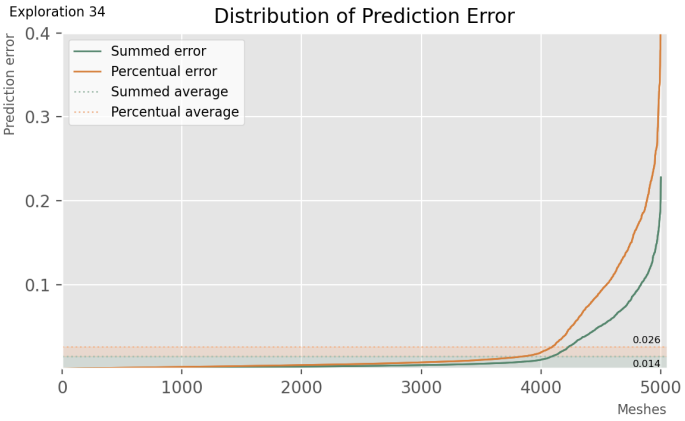
The attributes of this model are:

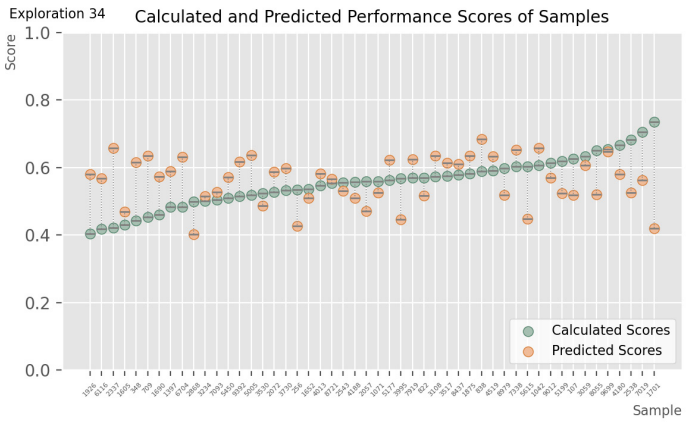
Batch size:	**128**
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

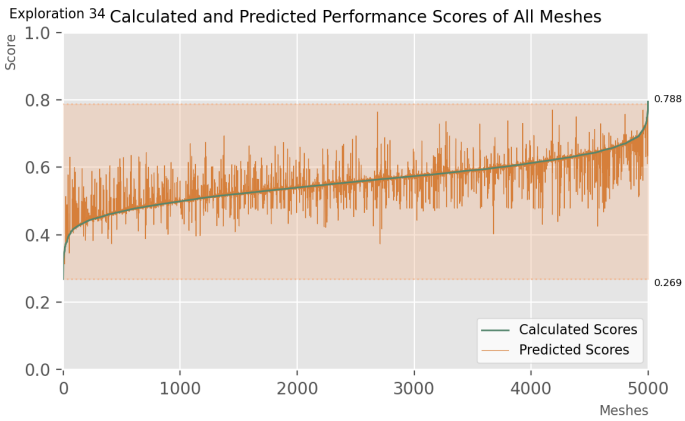


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





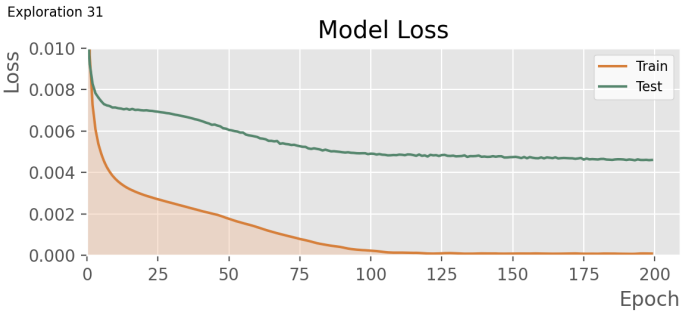


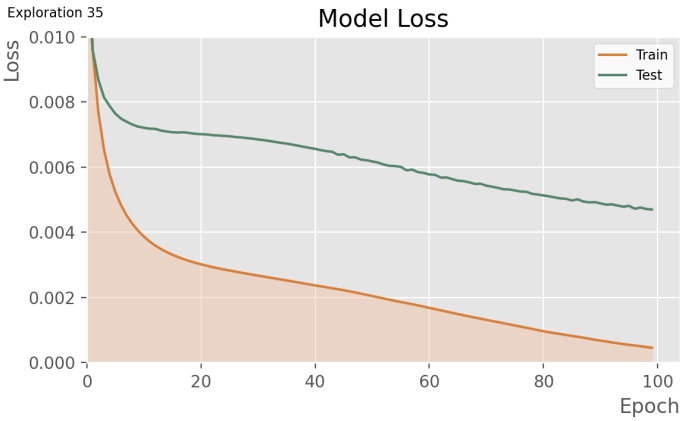


SURROGATE MODEL: EXPLORATION 35

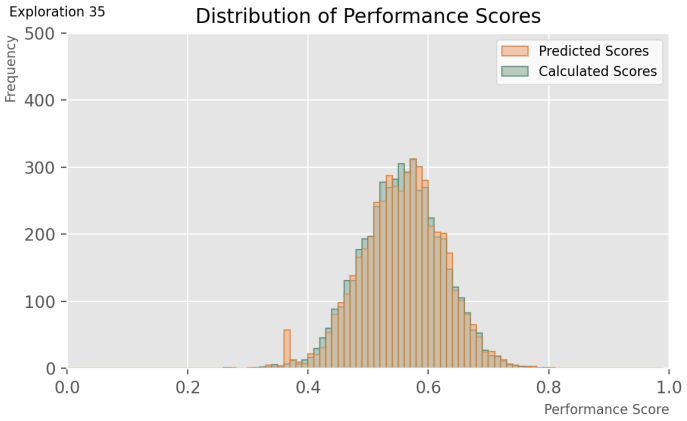
The attributes of this model are:

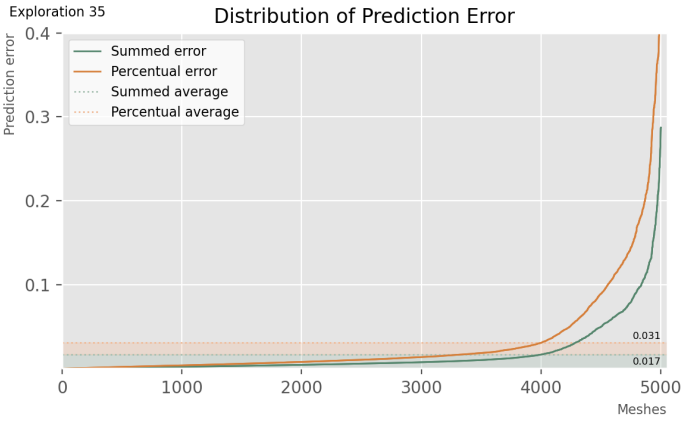
Batch size:	32
Epoch at which training is stopped:	**100**
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

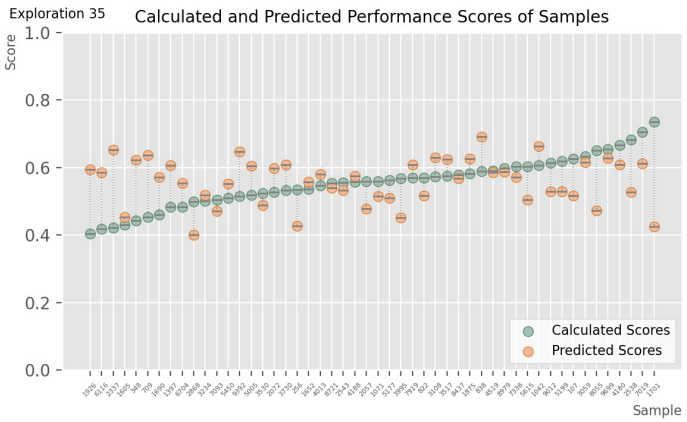


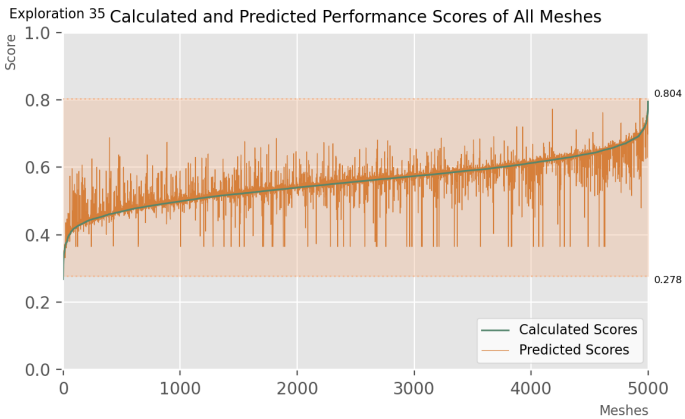


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





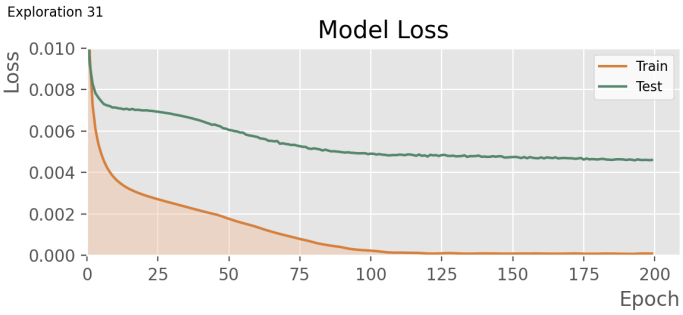




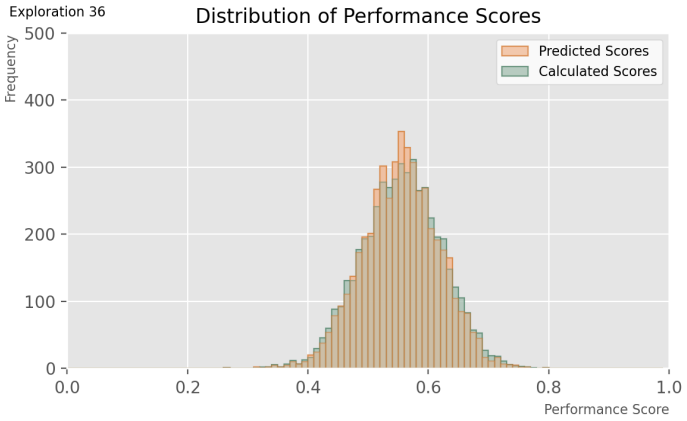
SURROGATE MODEL: EXPLORATION 36

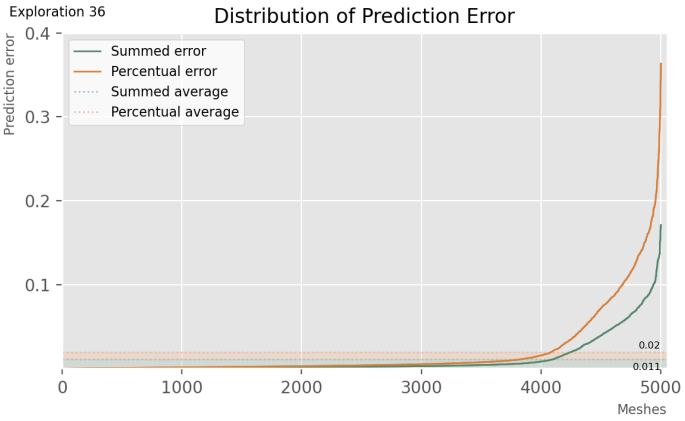
The attributes of this model are:

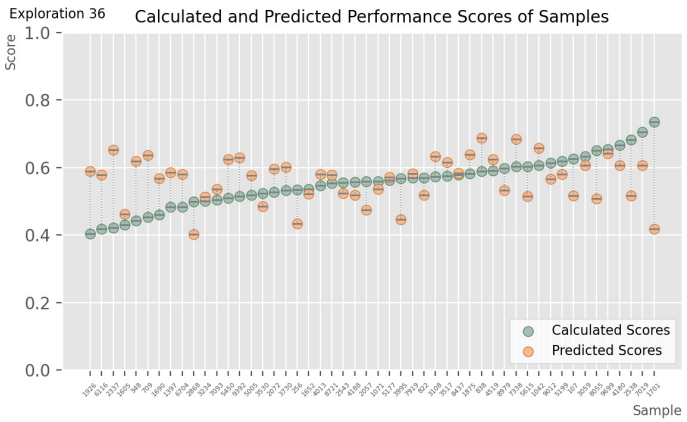
Batch size:	32
Epoch at which training is stopped:	**2000**
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

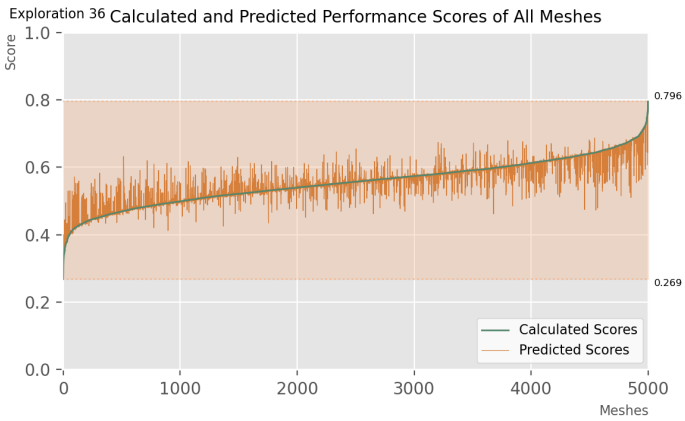


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





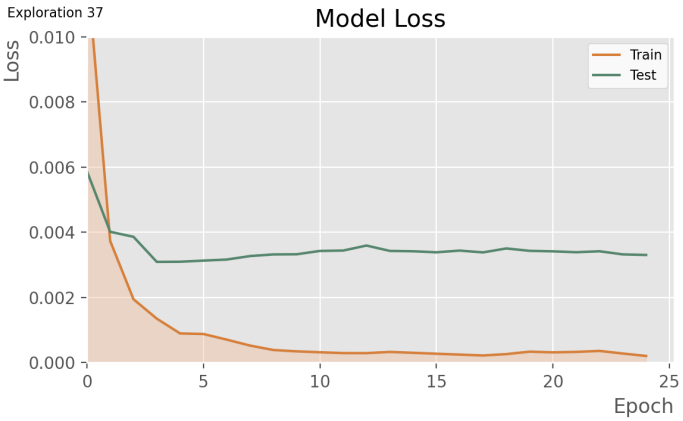




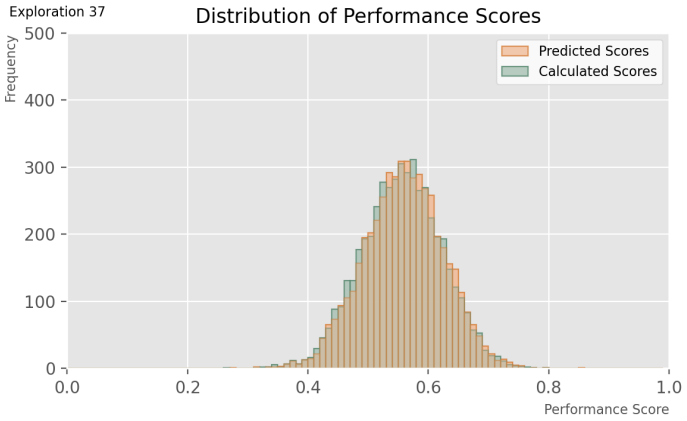
SURROGATE MODEL: EXPLORATION 37

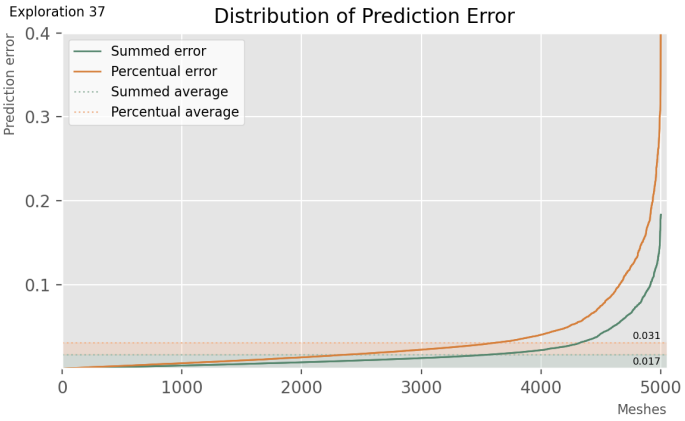
The attributes of this model are:

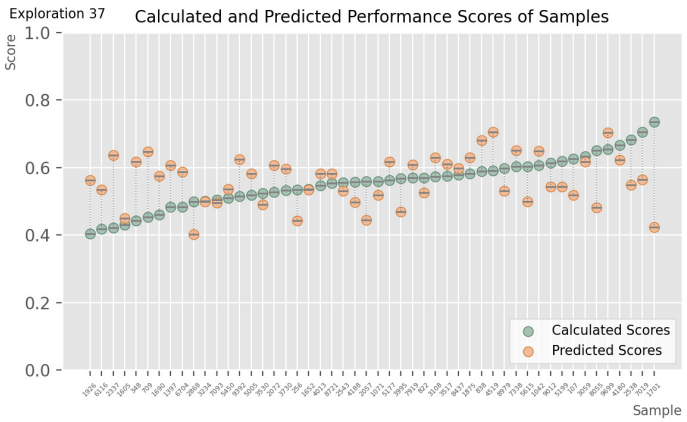
Batch size:	32
Epoch at which training is stopped:	25
Learning rate:	0.001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

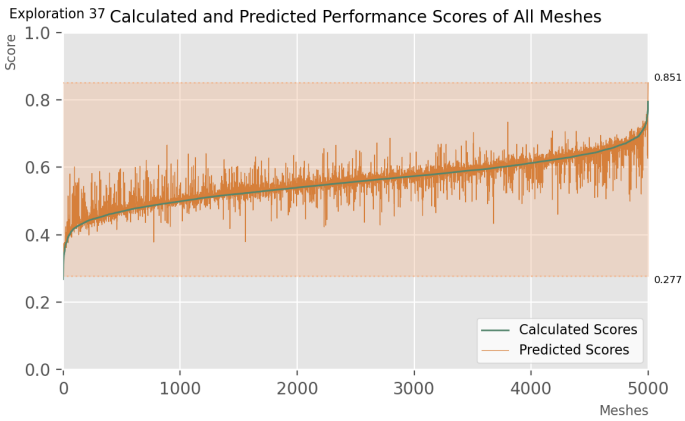


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





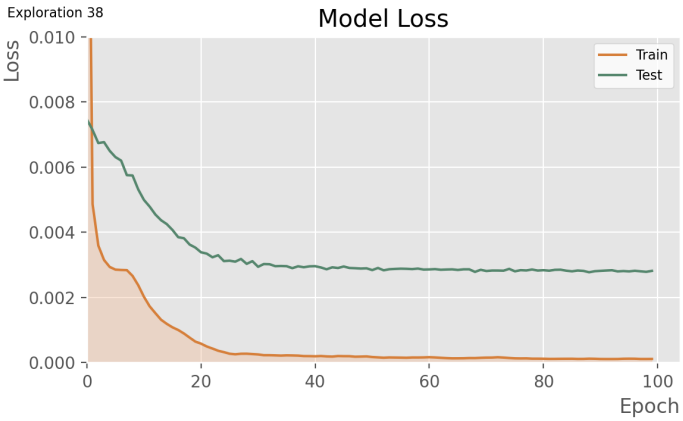




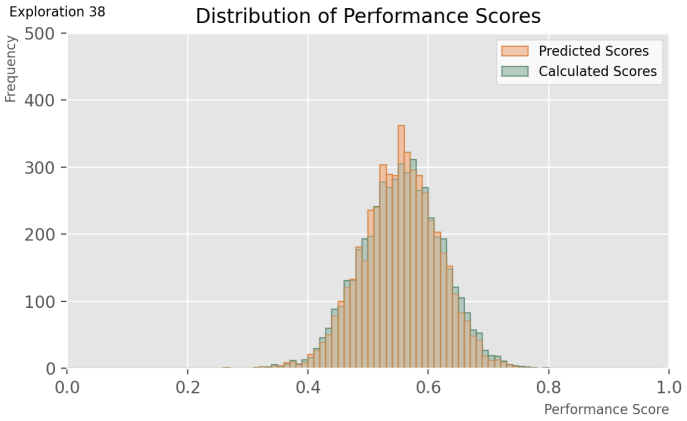
SURROGATE MODEL: EXPLORATION 38

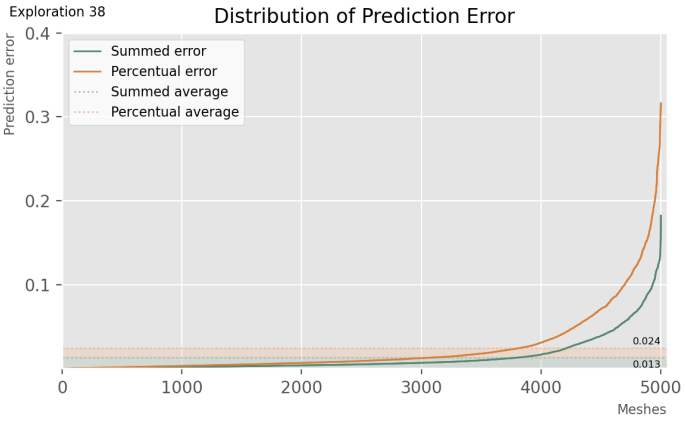
The attributes of this model are:

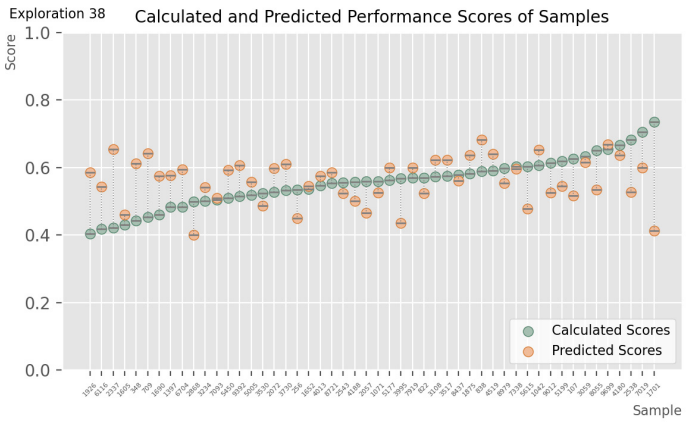
Batch size:	32
Epoch at which training is stopped:	100
Learning rate:	0.0001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

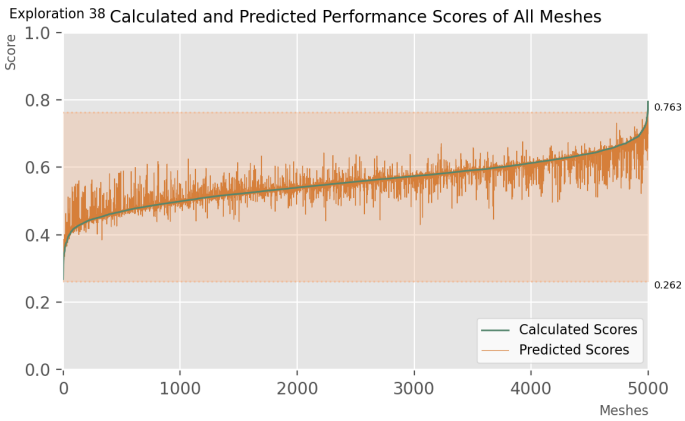


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





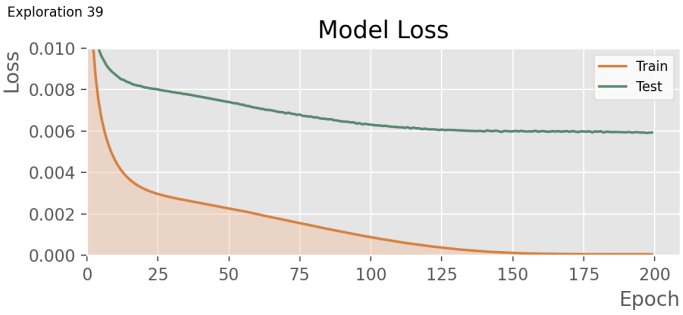


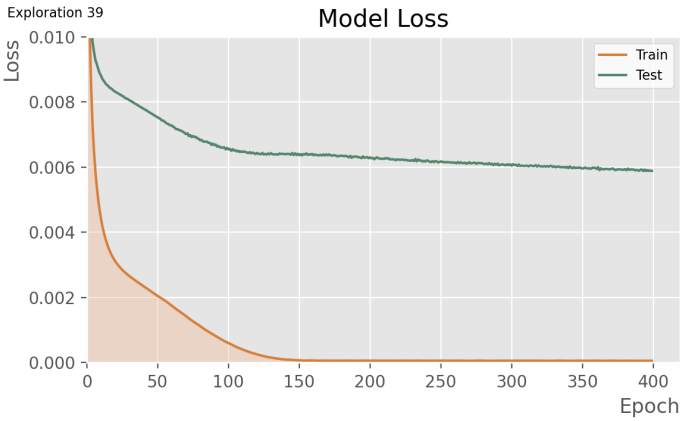


SURROGATE MODEL: EXPLORATION 39

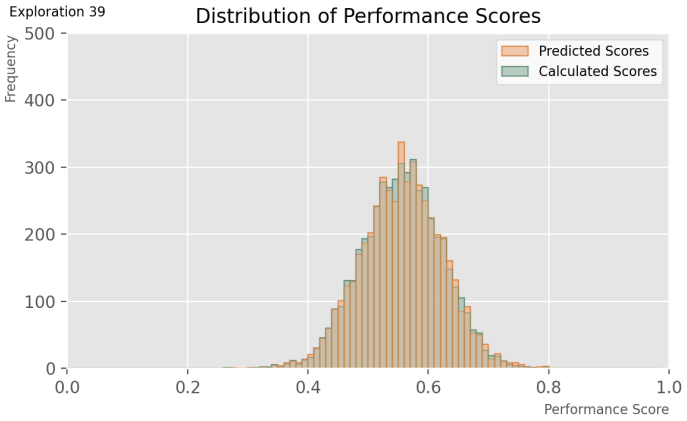
The attributes of this model are:

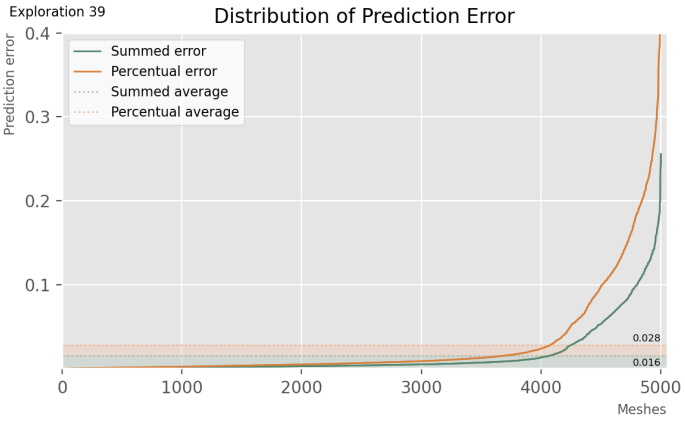
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.000005
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

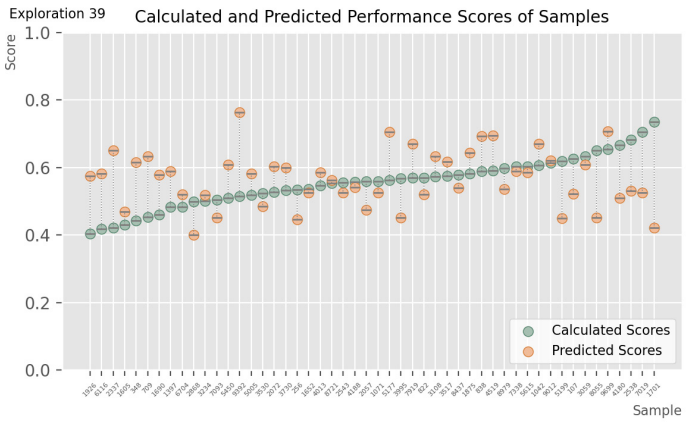


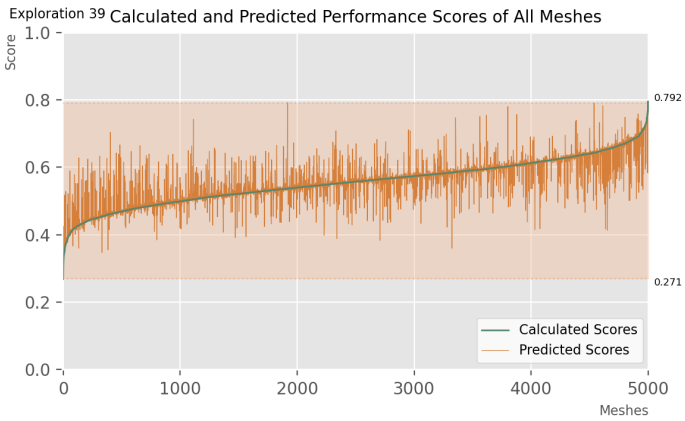


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





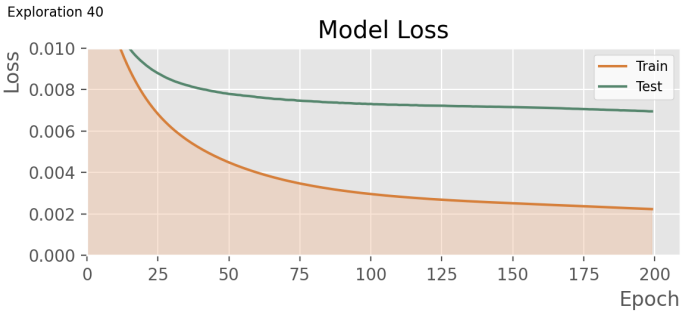




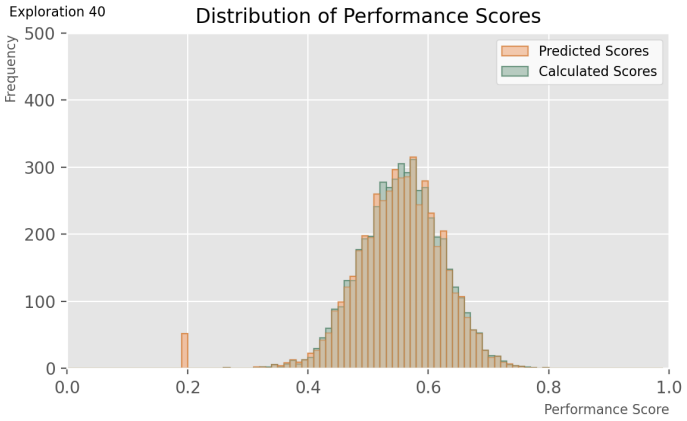
SURROGATE MODEL: EXPLORATION 40

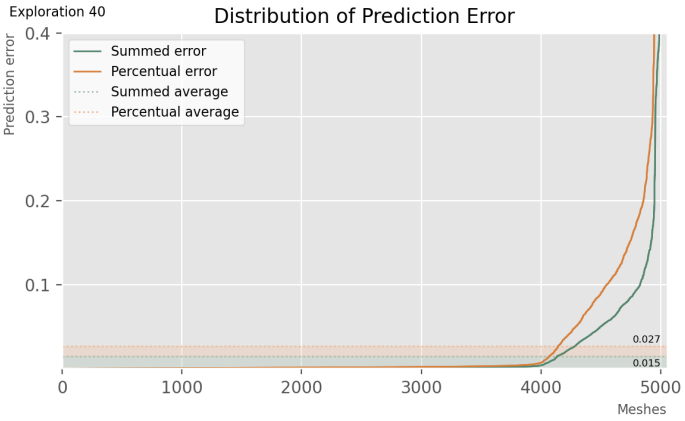
The attributes of this model are:

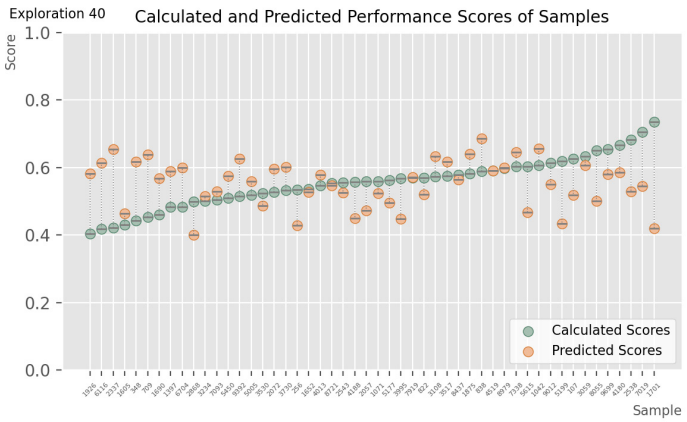
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.000001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

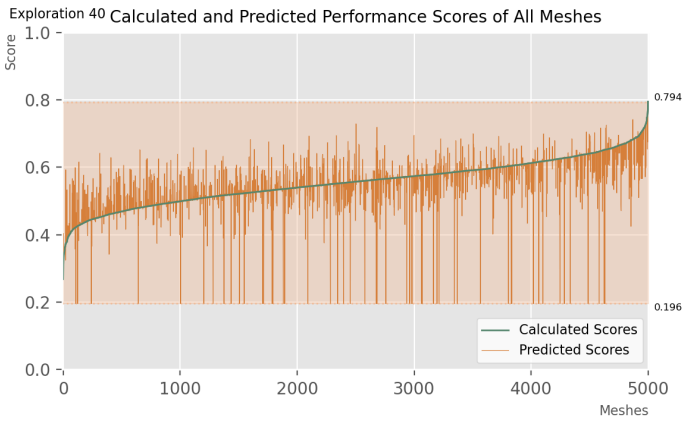


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





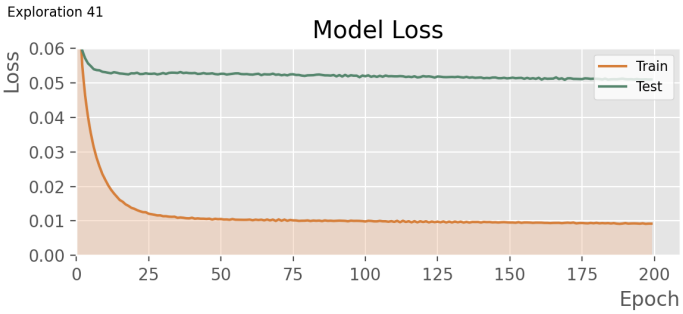




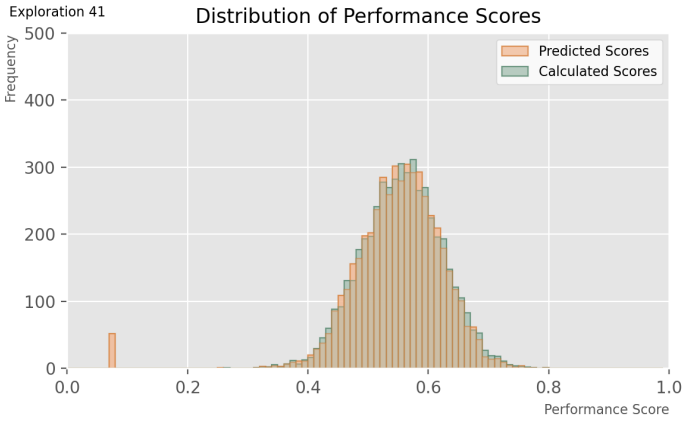
SURROGATE MODEL: EXPLORATION 41

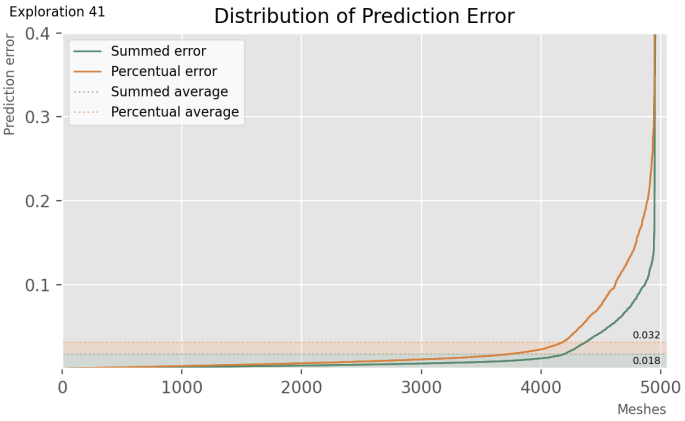
The attributes of this model are:

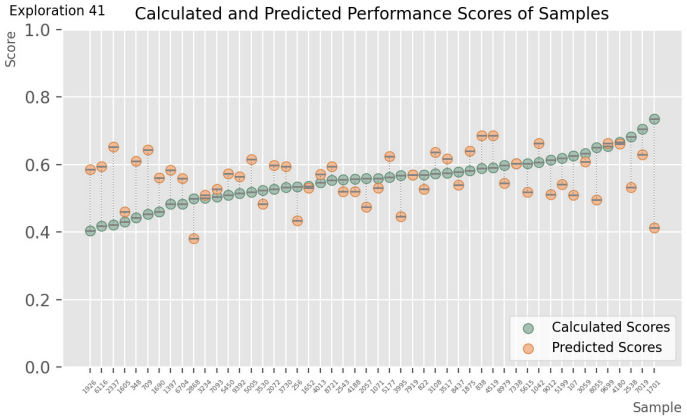
Batch size:	32
Epoch at which training is stopped:	200
Learning rate:	0.00001
Loss types:	**Mean average error**
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

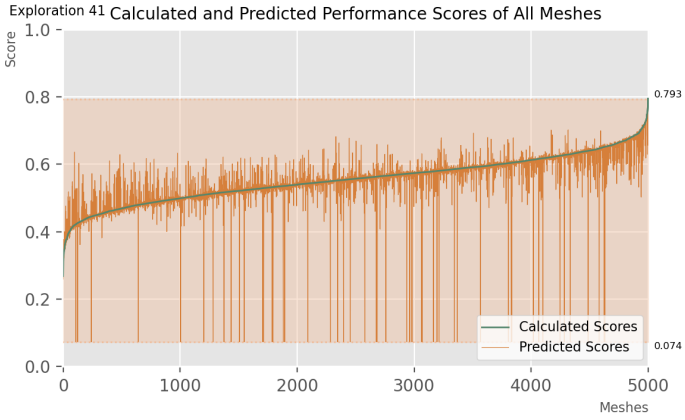


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





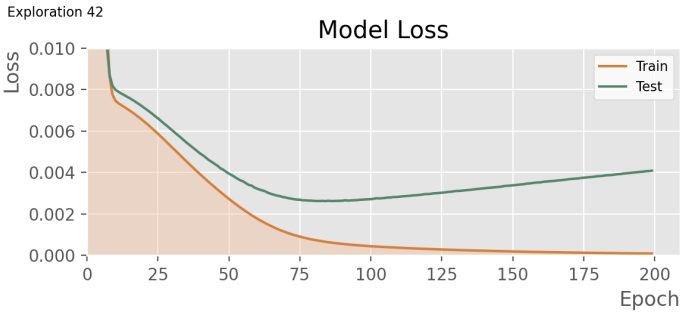


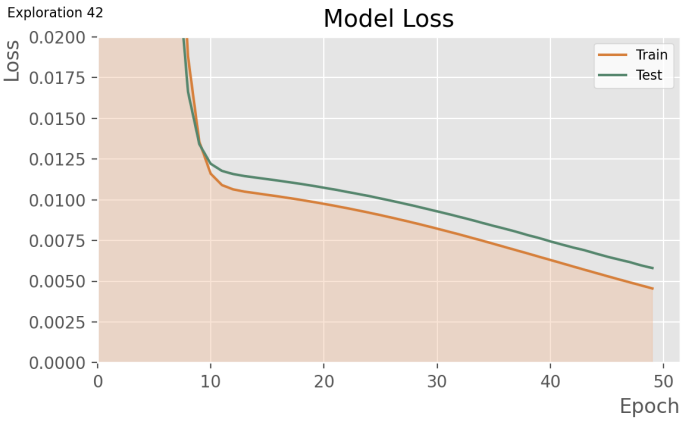


SURROGATE MODEL: EXPLORATION 42

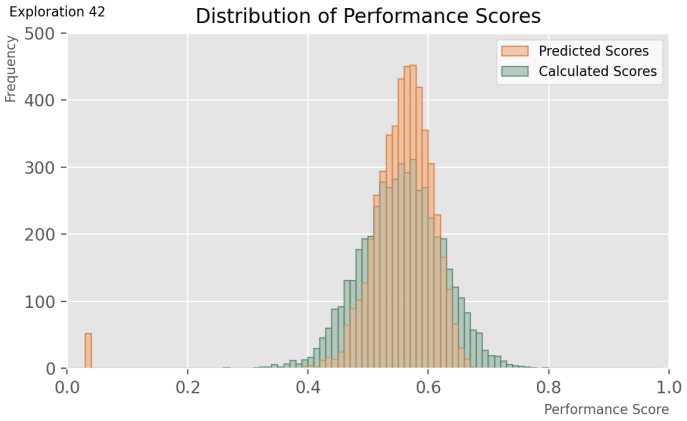
The attributes of this model are:

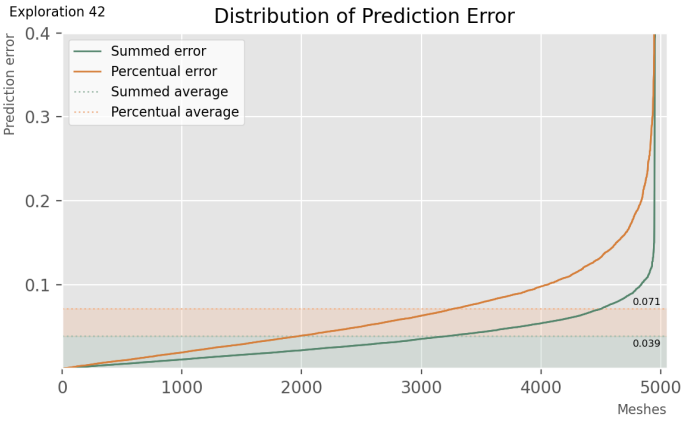
Batch size:	32
Epoch at which training is stopped:	50
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	**Sigmoid**
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

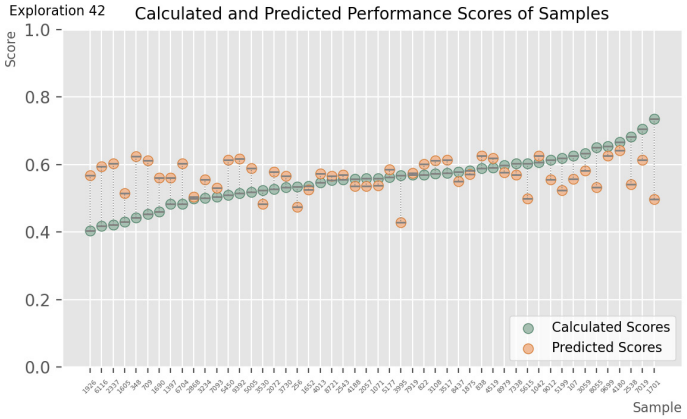


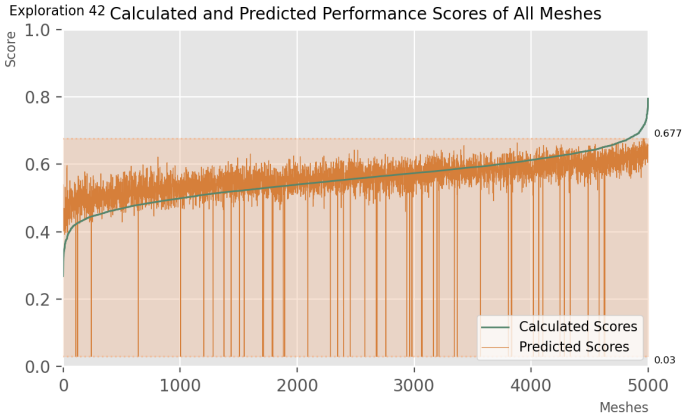


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





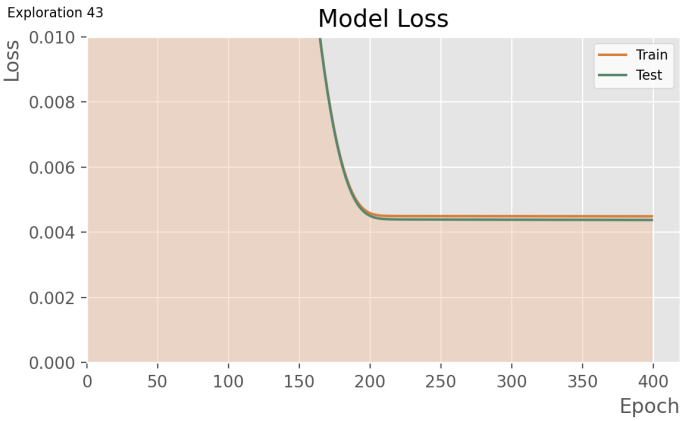




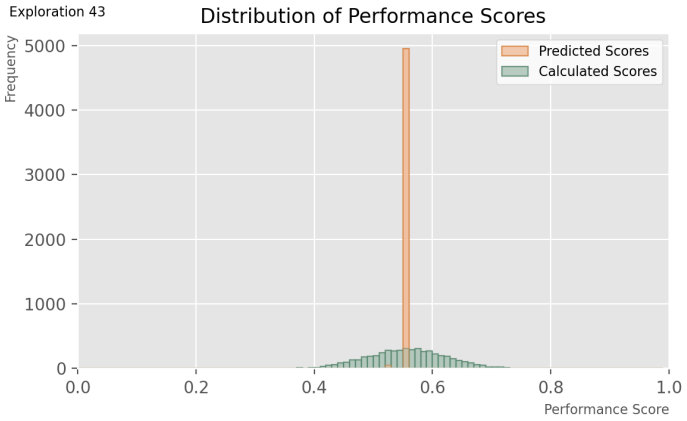
SURROGATE MODEL: EXPLORATION 43

The attributes of this model are:

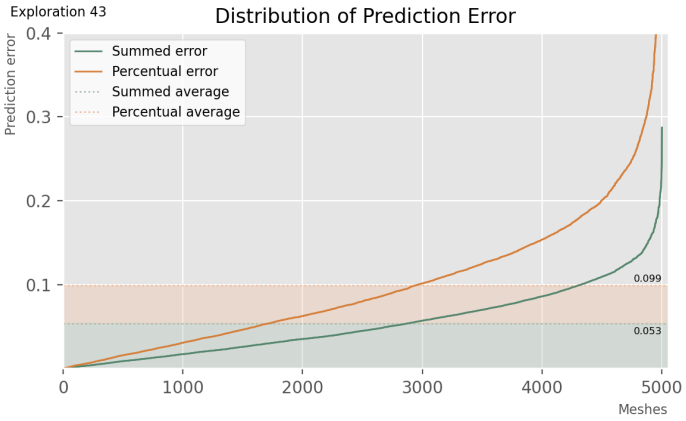
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	**Softmax**
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

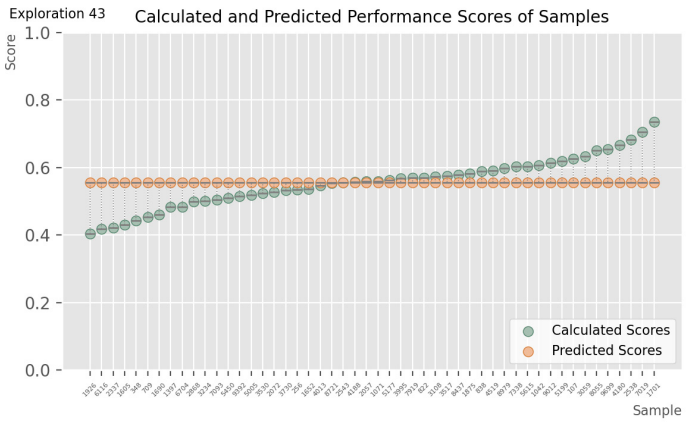


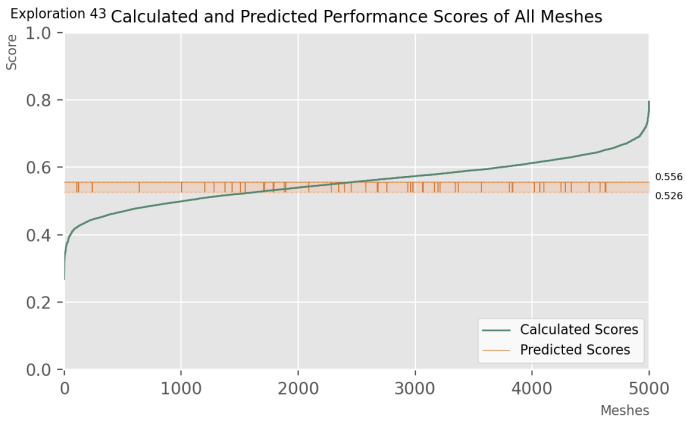
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





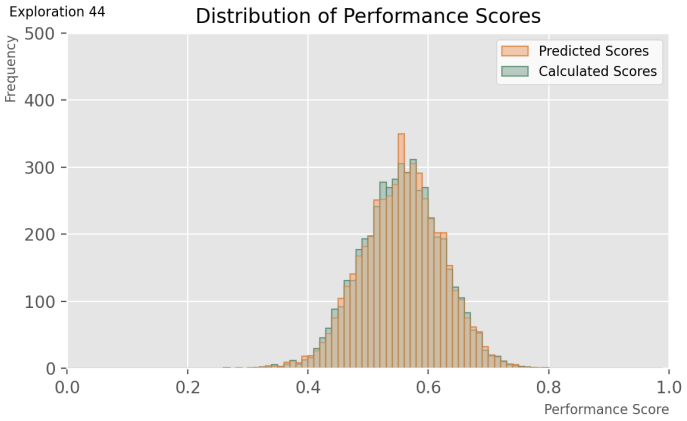


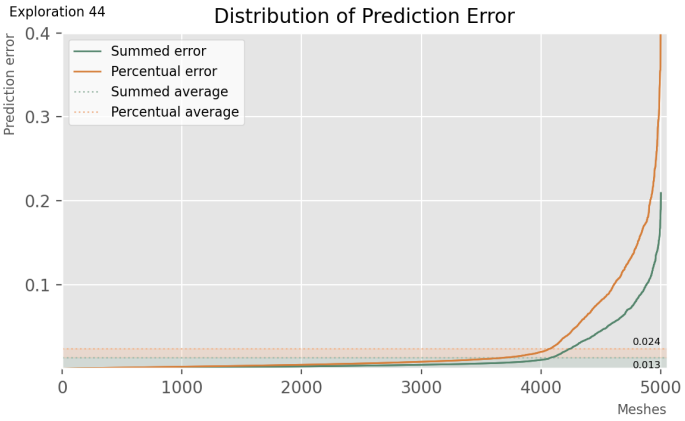
SURROGATE MODEL: EXPLORATION 44

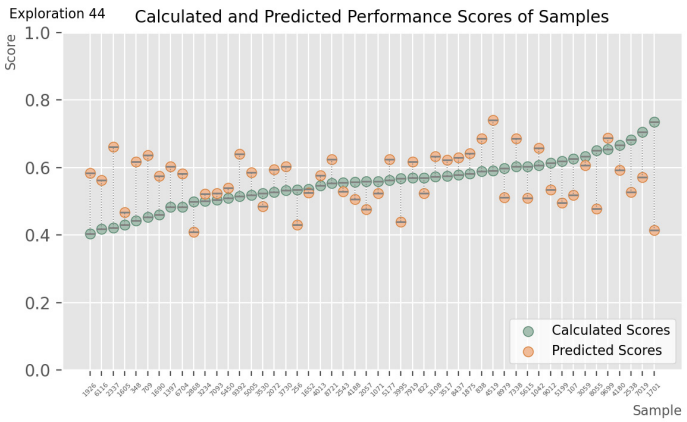
The attributes of this model are:

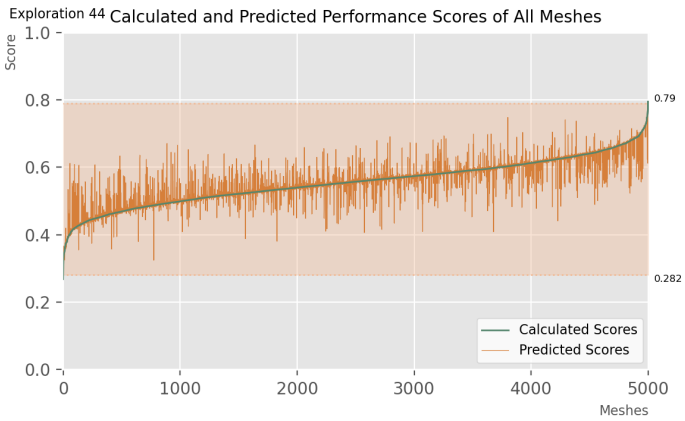
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	**Softsign**
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





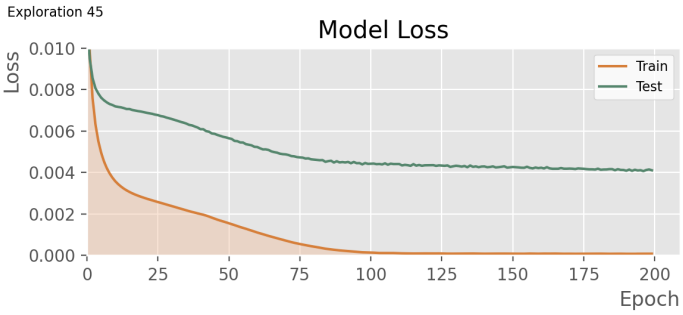




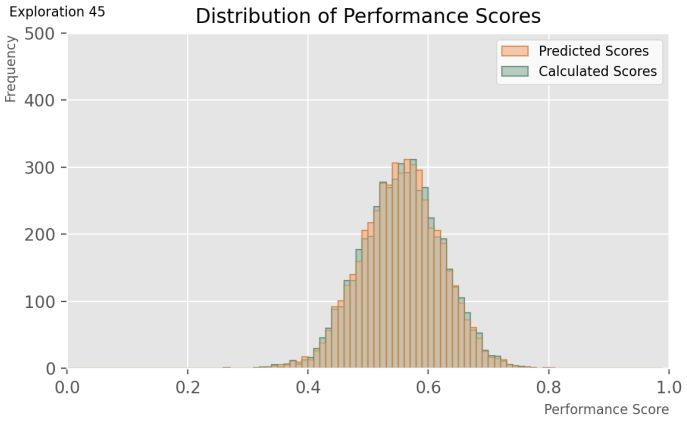
SURROGATE MODEL: EXPLORATION 45

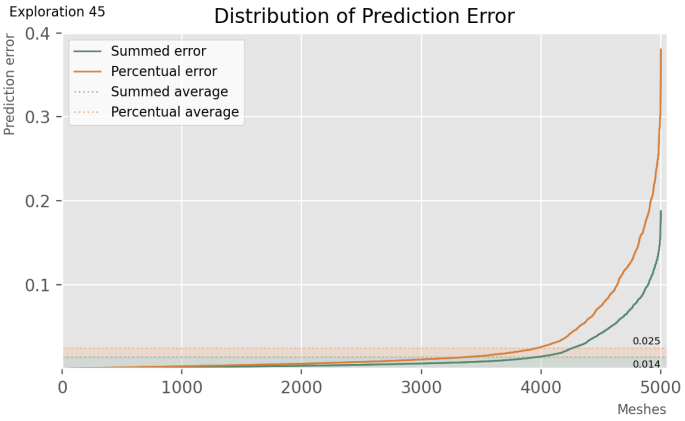
The attributes of this model are:

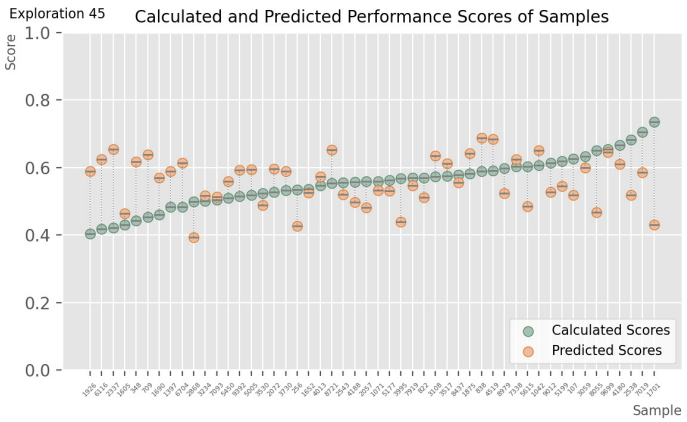
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	top_k_categorical_accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

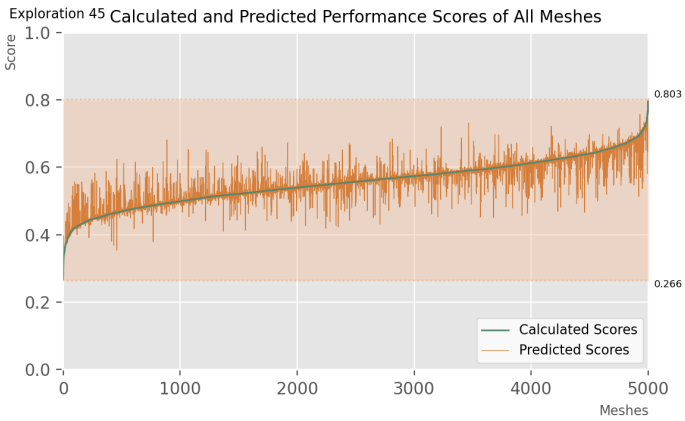


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





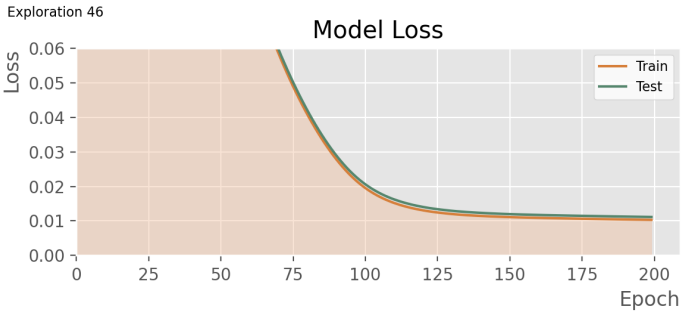




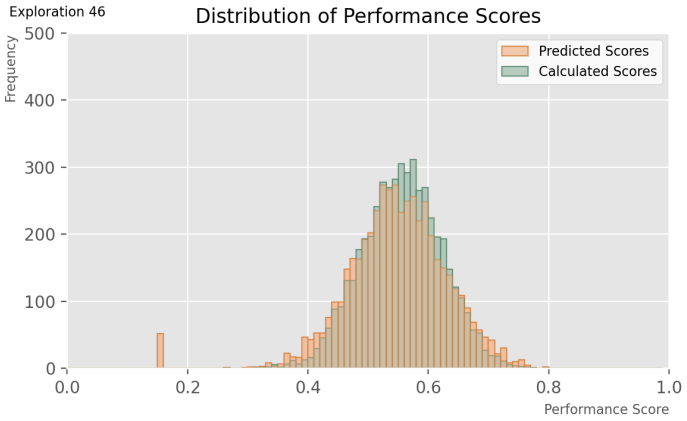
SURROGATE MODEL: EXPLORATION 46

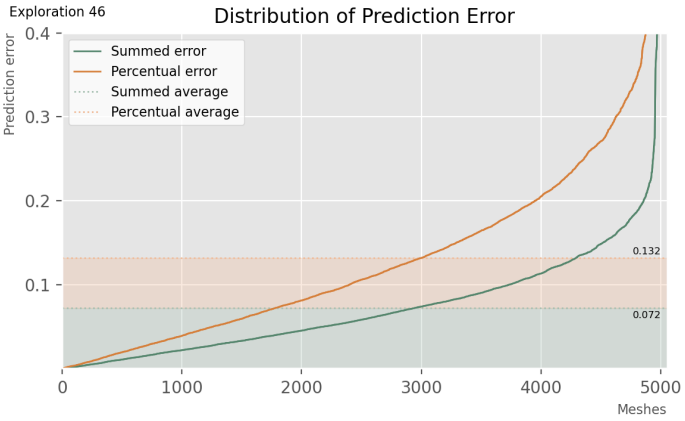
The attributes of this model are:

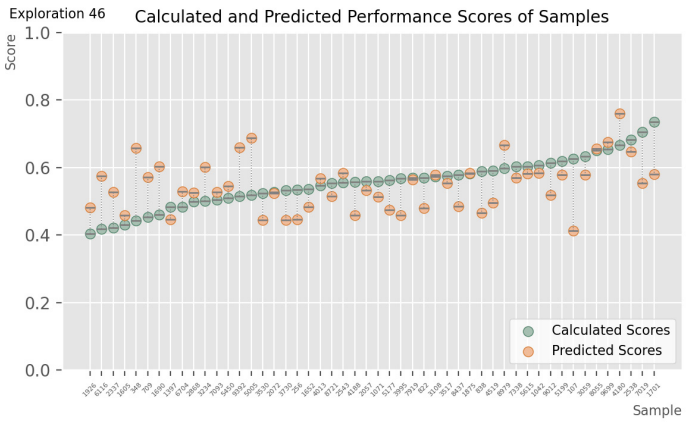
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	SGD
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

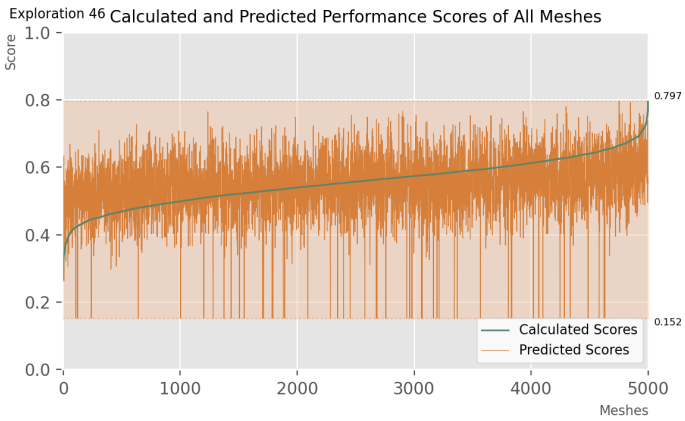


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





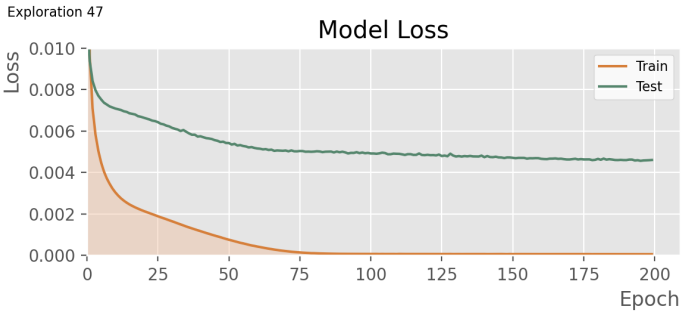




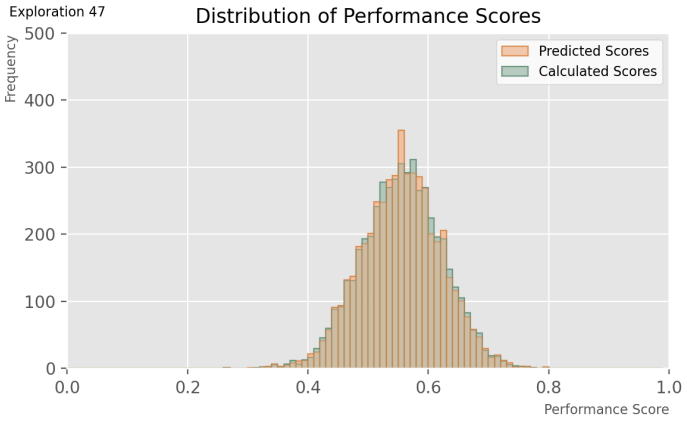
SURROGATE MODEL: EXPLORATION 47

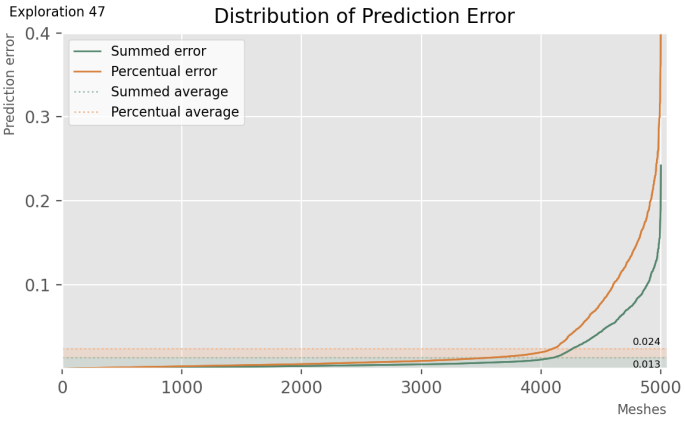
The attributes of this model are:

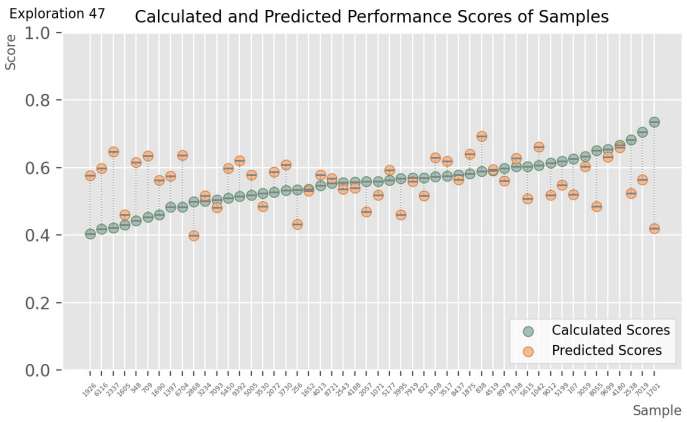
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	**RMSprop**
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

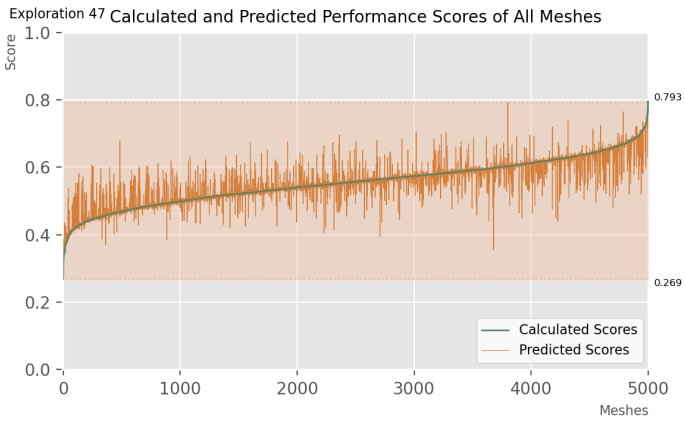


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





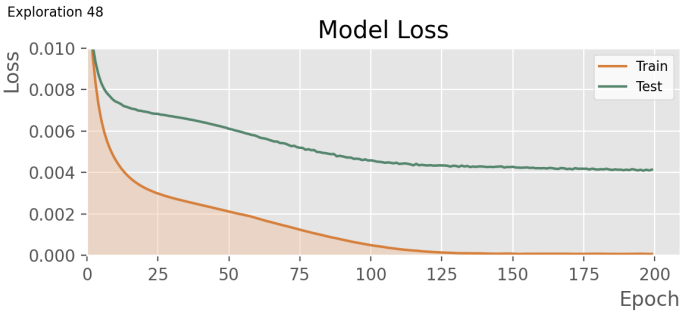




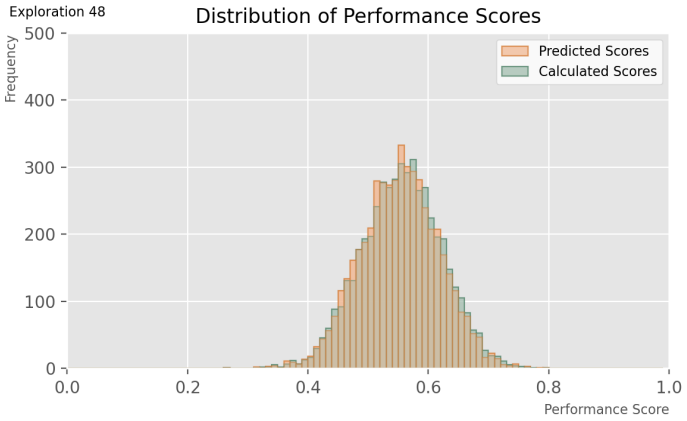
SURROGATE MODEL: EXPLORATION 48

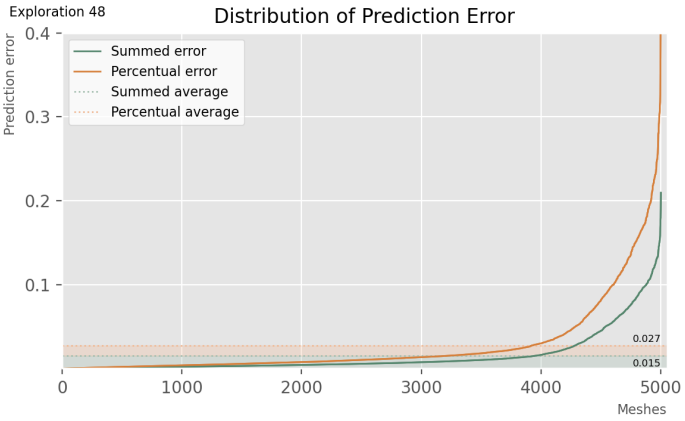
The attributes of this model are:

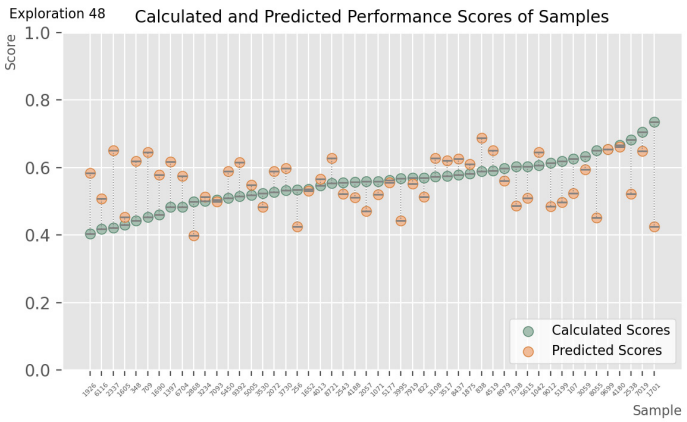
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	None
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

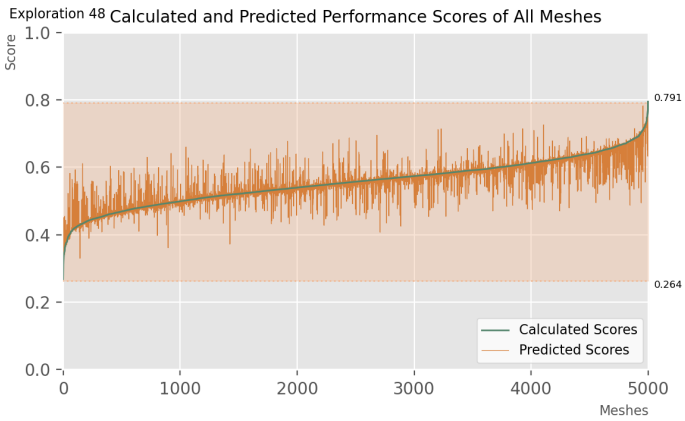


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





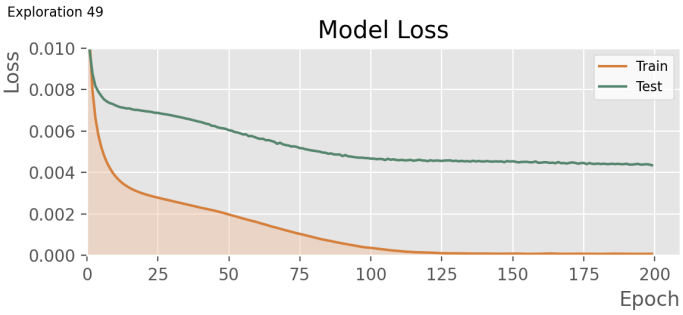




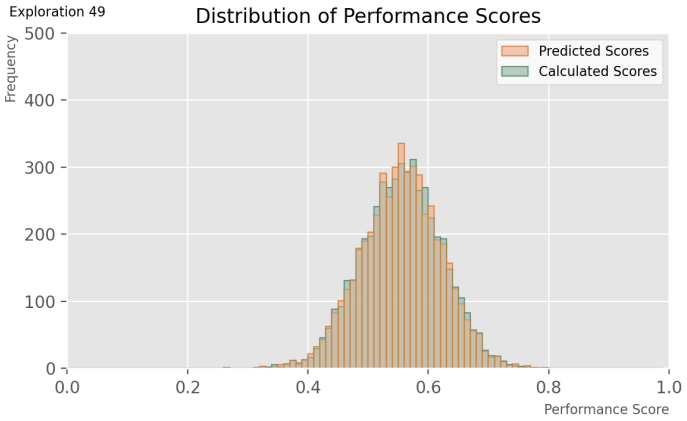
SURROGATE MODEL: EXPLORATION 49

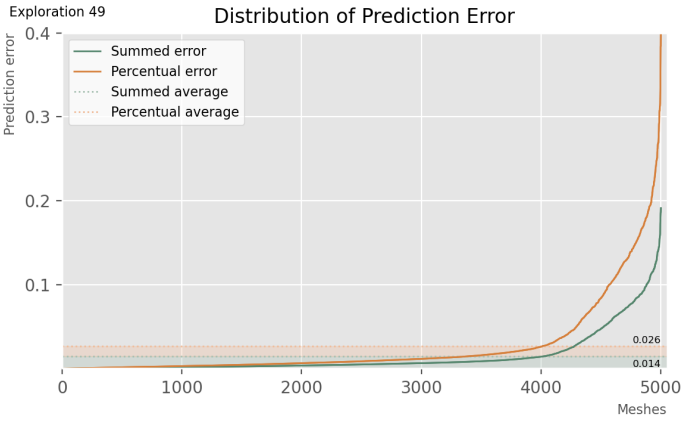
The attributes of this model are:

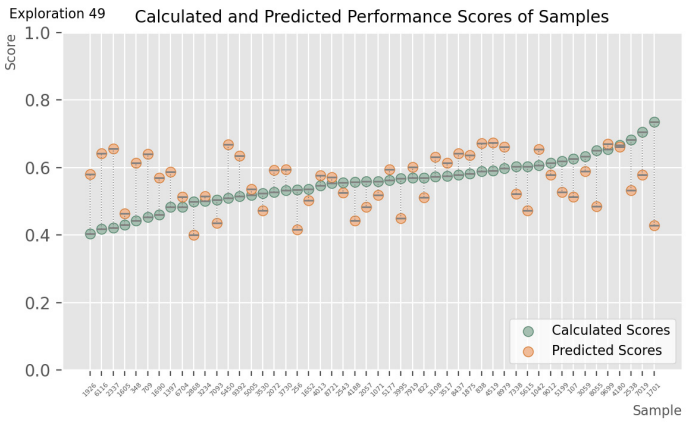
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	**One** additional normalization layer
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

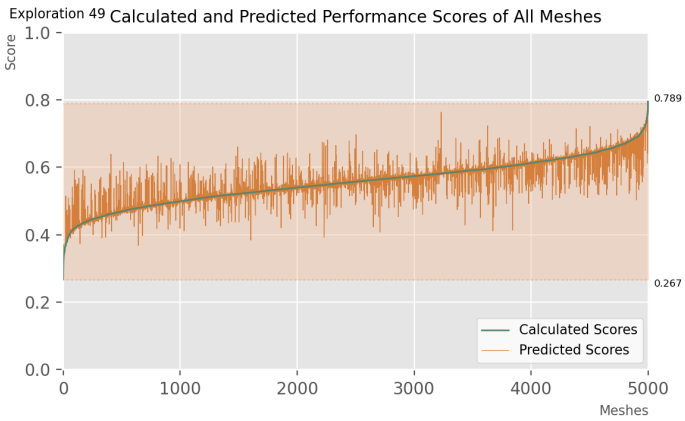


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





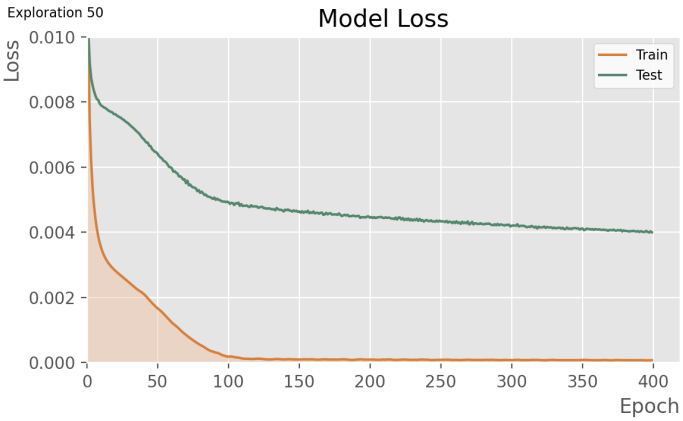




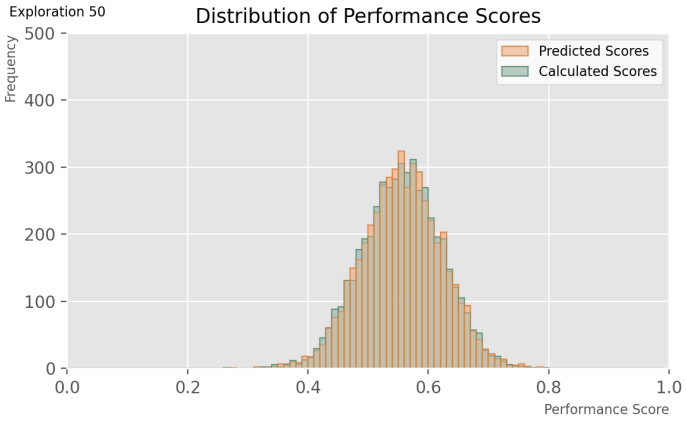
SURROGATE MODEL: EXPLORATION 50

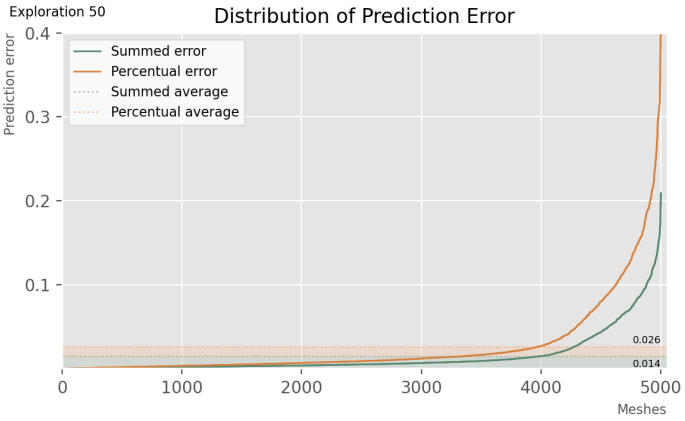
The attributes of this model are:

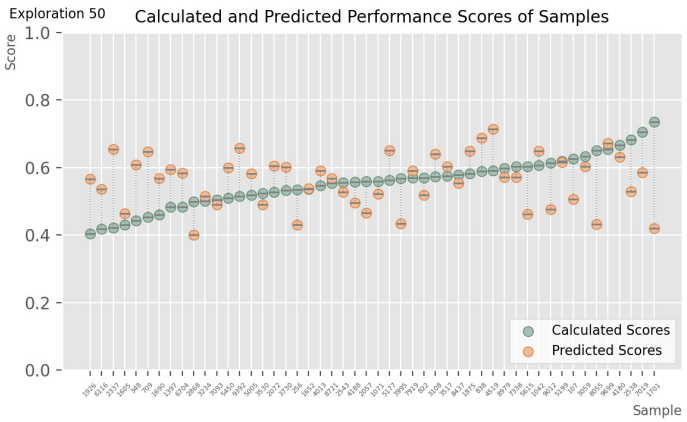
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	**Multiple** additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'64-32-1'

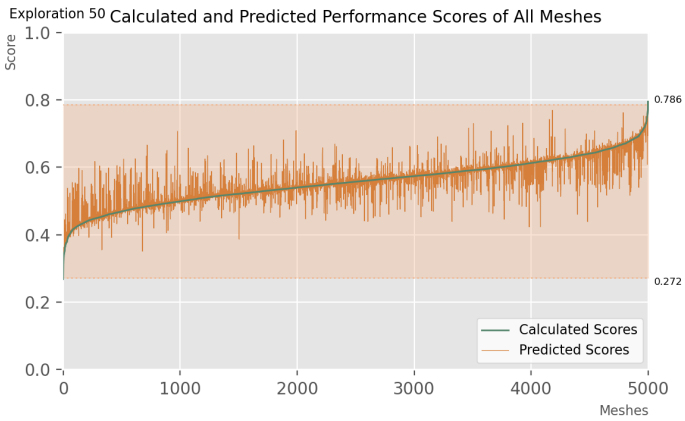


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





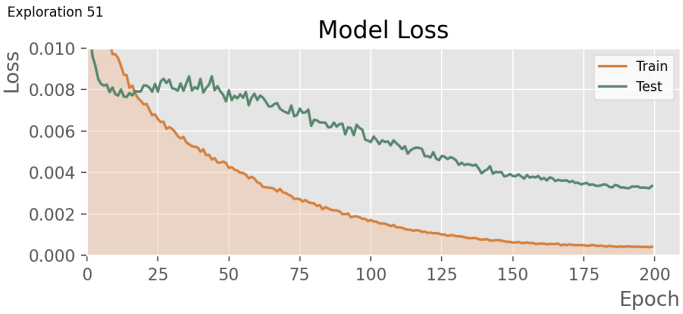




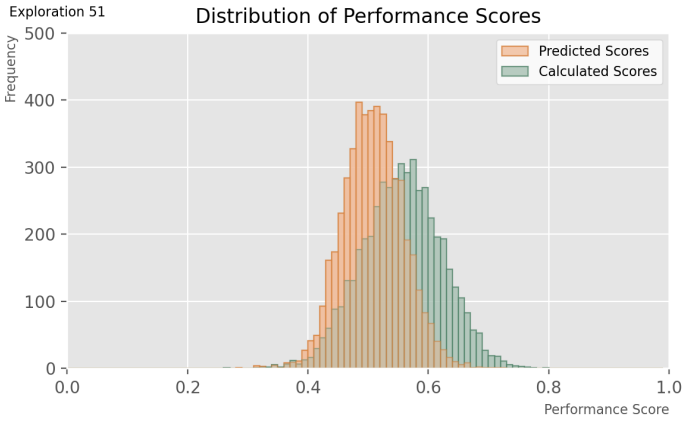
SURROGATE MODEL: EXPLORATION 51

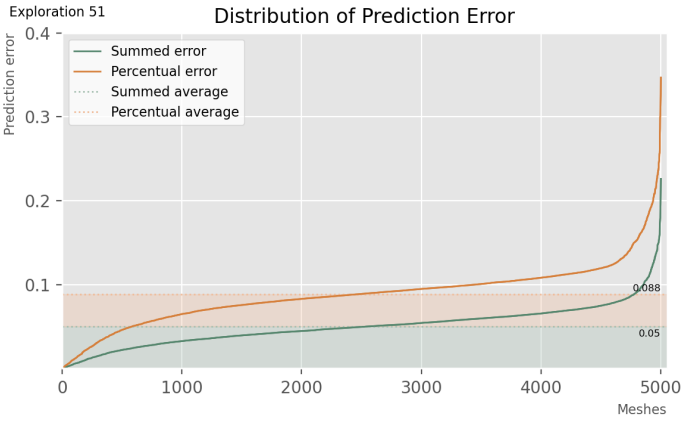
The attributes of this model are:

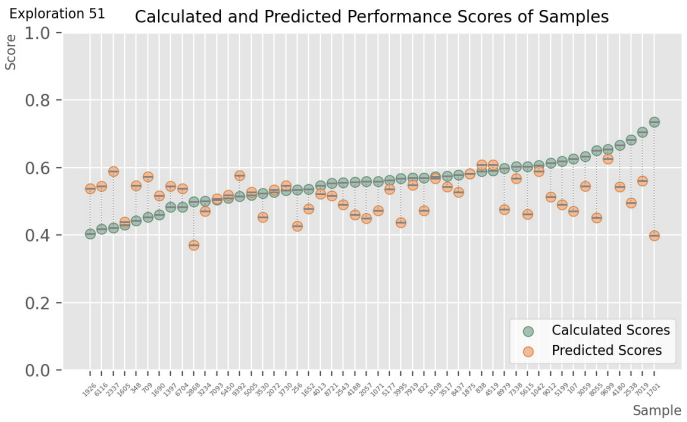
Batch size:	32
Epoch at which training is stopped:	300
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	One dropout layers
Architecture:	'64-32-1'

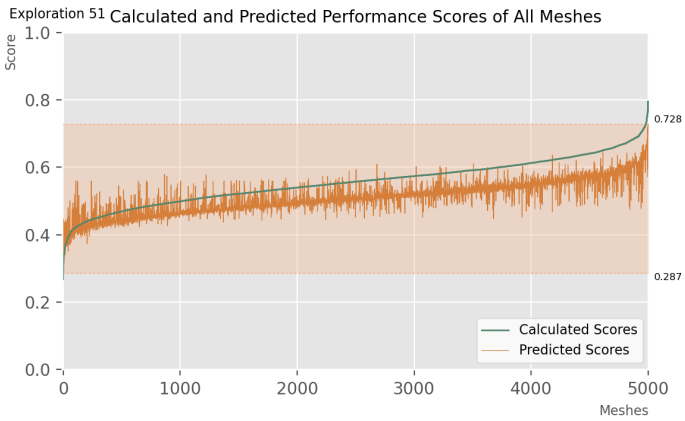


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





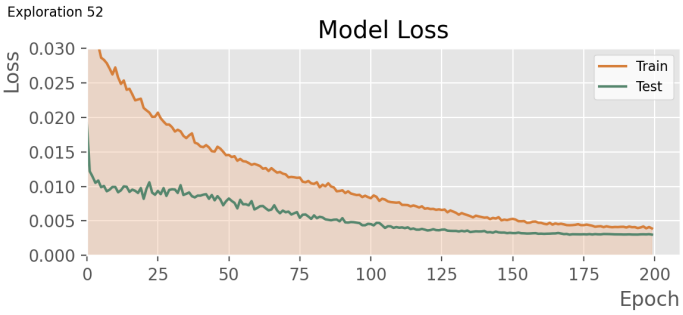




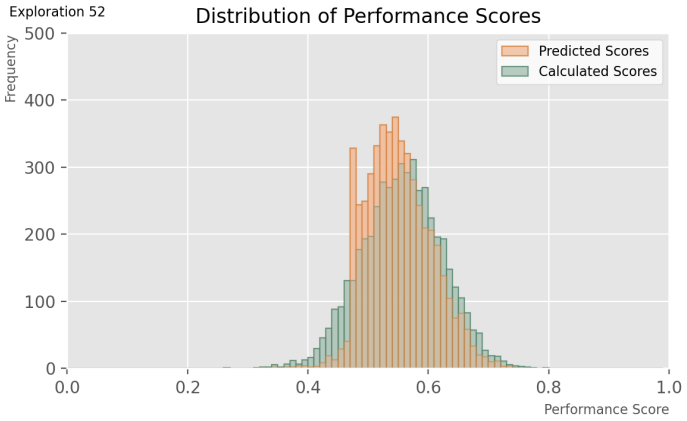
SURROGATE MODEL: EXPLORATION 52

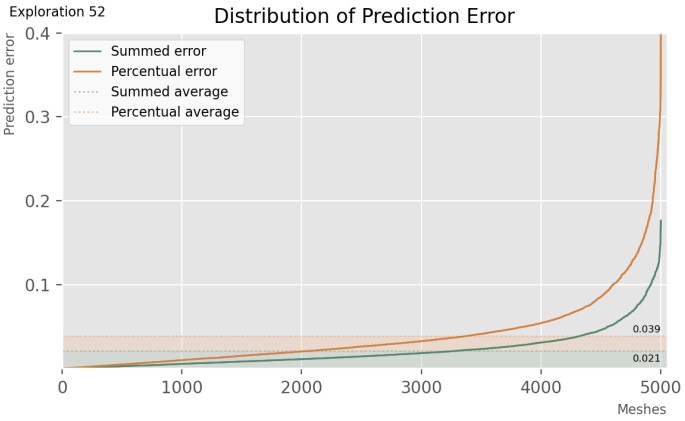
The attributes of this model are:

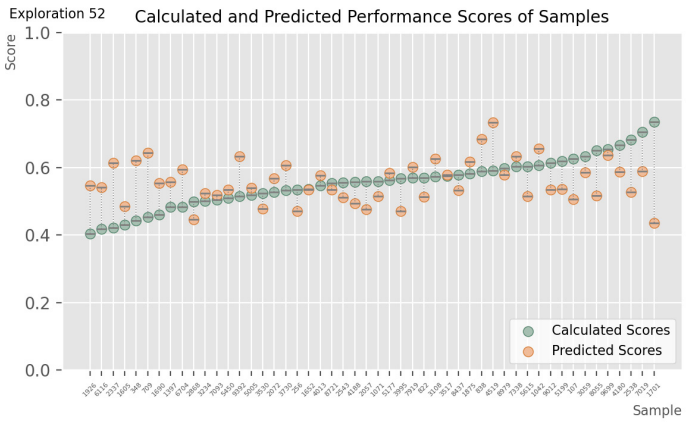
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	**Multiple** dropout layers
Architecture:	'64-32-1'

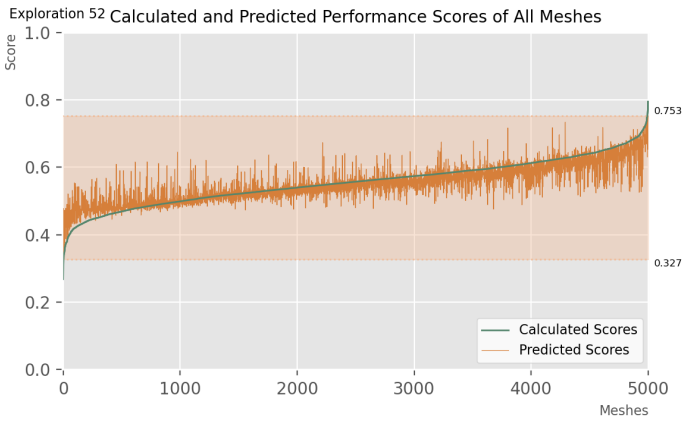


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





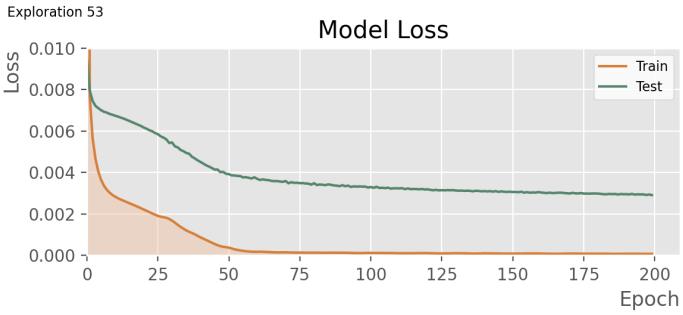


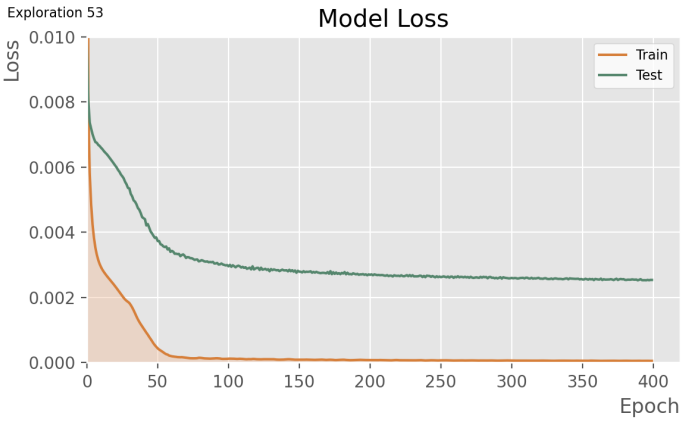


SURROGATE MODEL: EXPLORATION 53

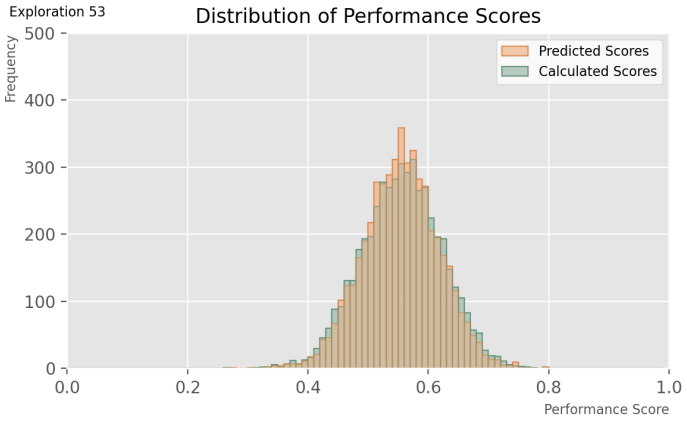
The attributes of this model are:

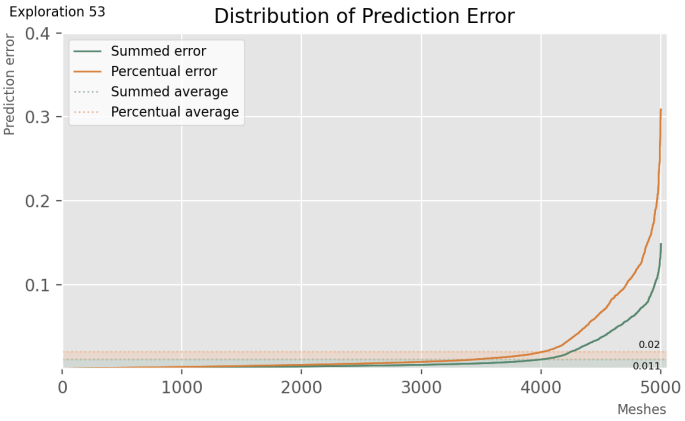
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	**'256-128-64-1'**

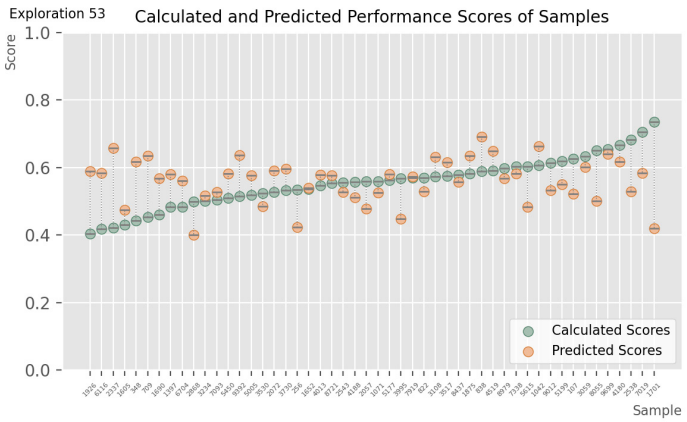


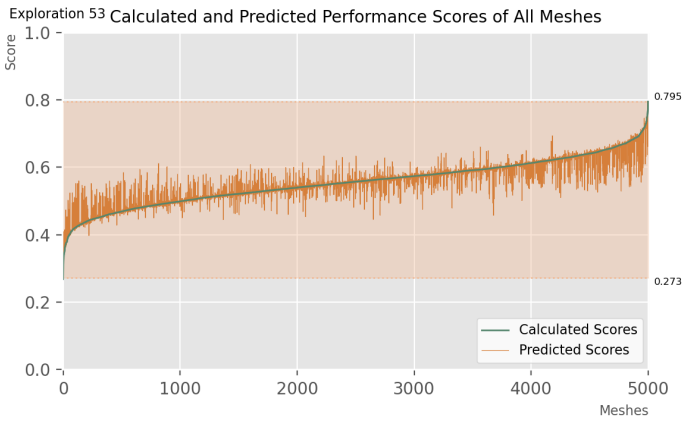


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





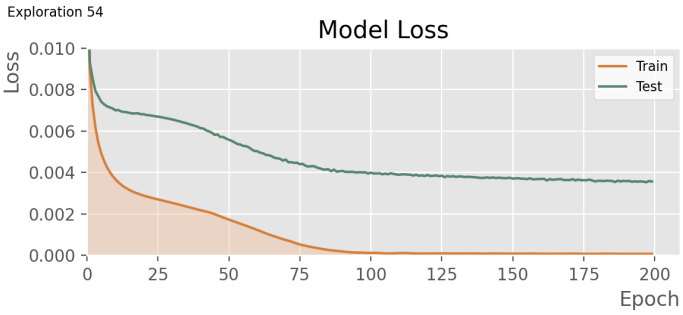


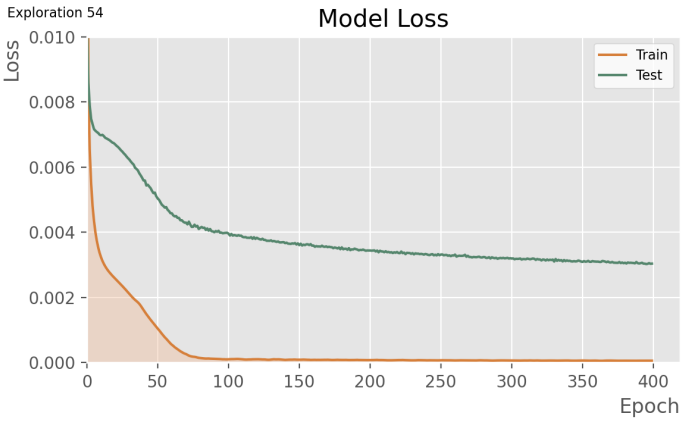


SURROGATE MODEL: EXPLORATION 54

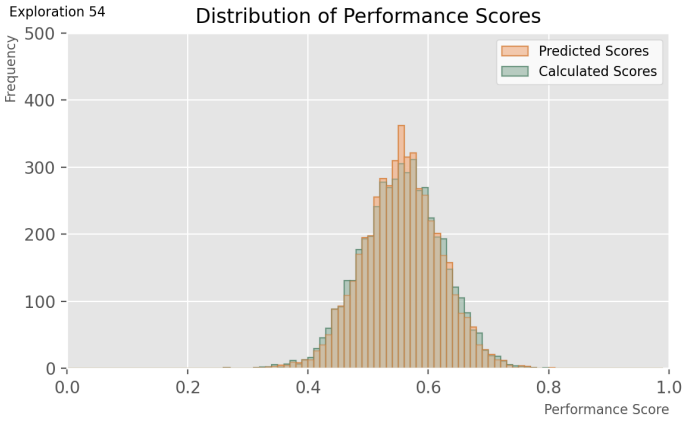
The attributes of this model are:

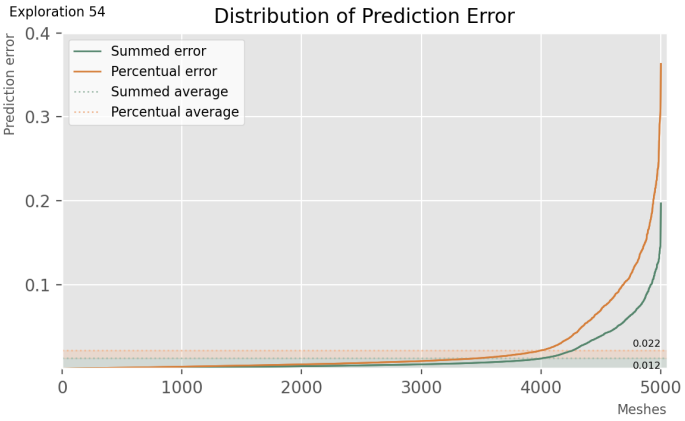
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	**'128-64-32-1'**

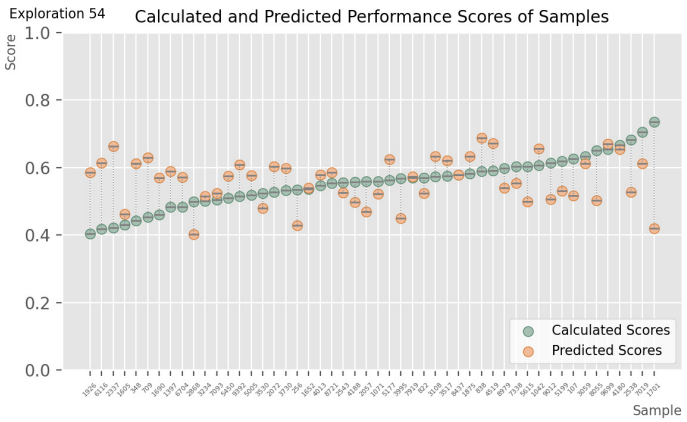


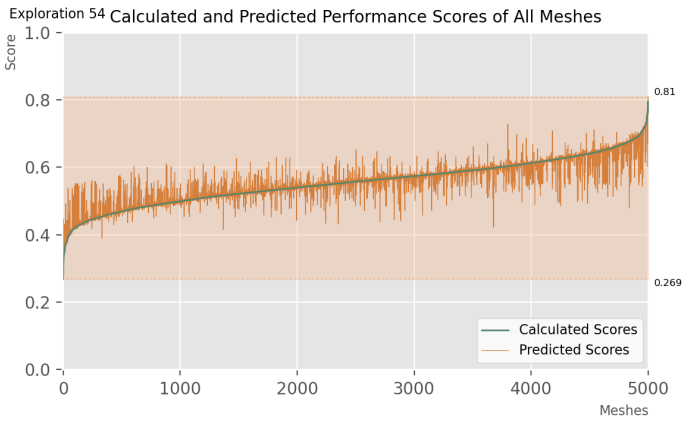


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





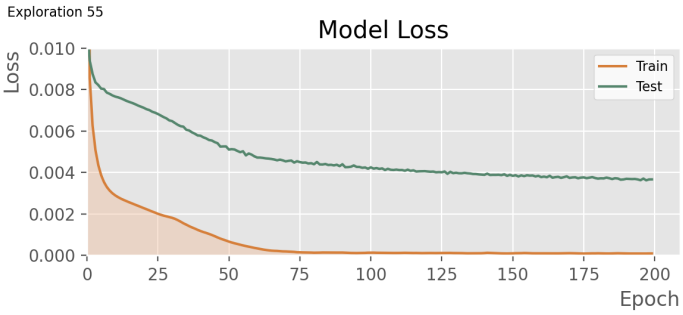


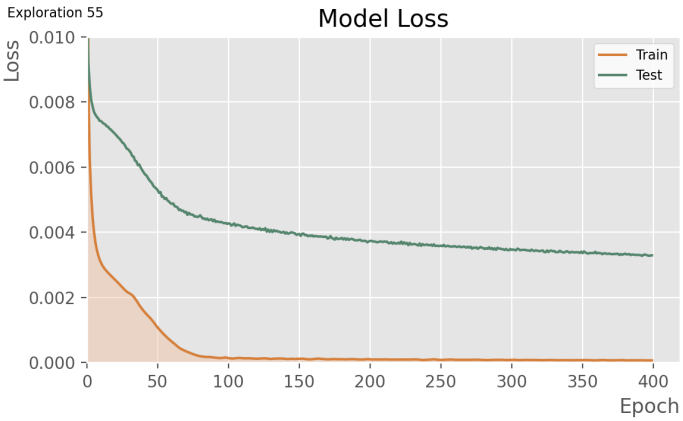


SURROGATE MODEL: EXPLORATION 55

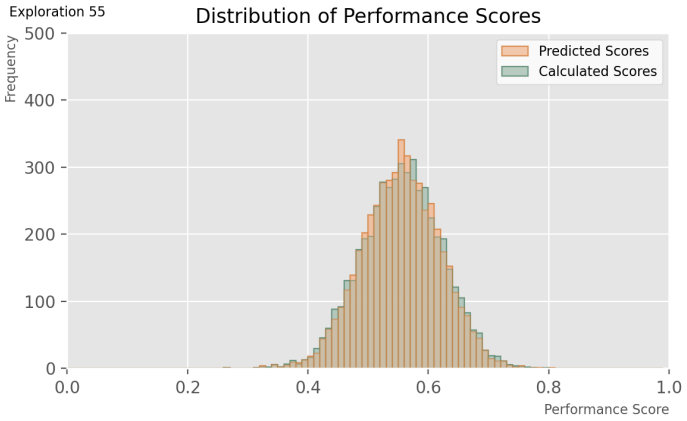
The attributes of this model are:

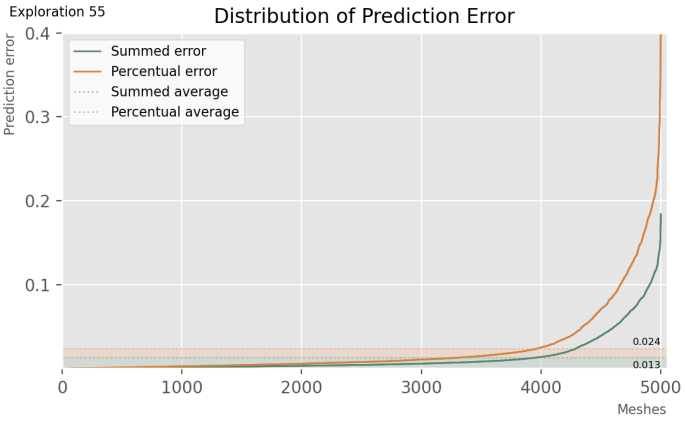
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	**'128-64-1'**

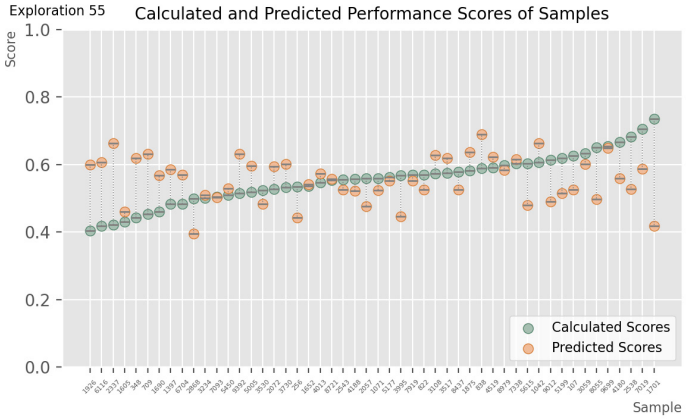


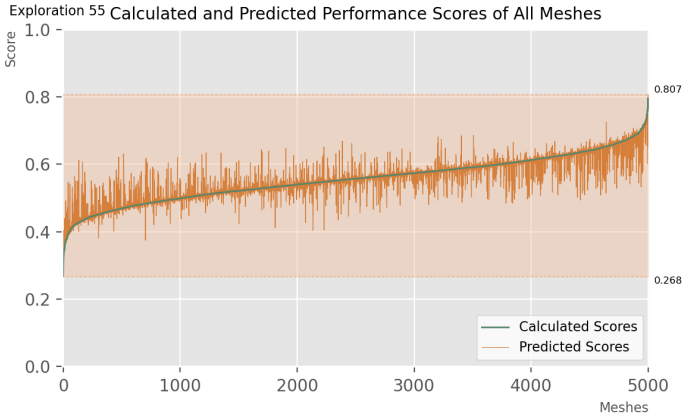


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





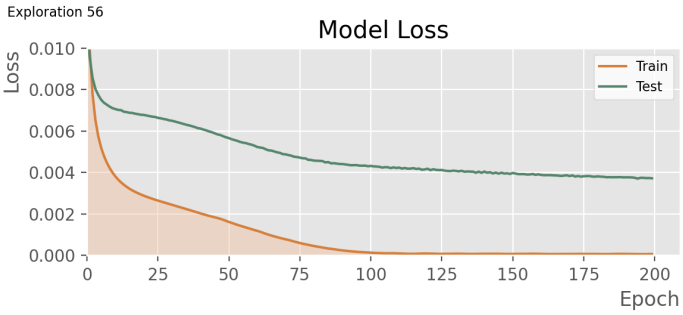


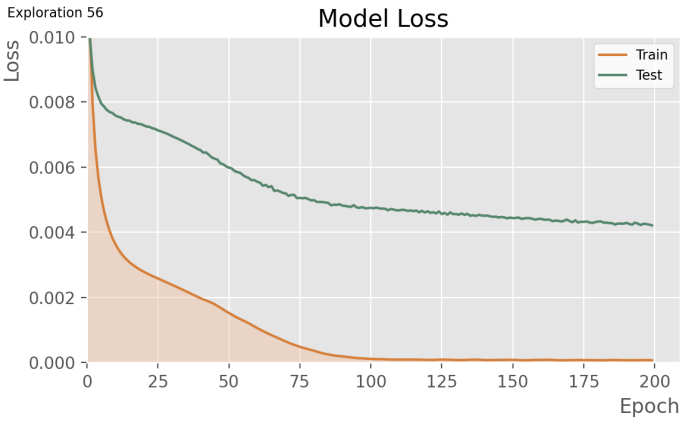


SURROGATE MODEL: EXPLORATION 56

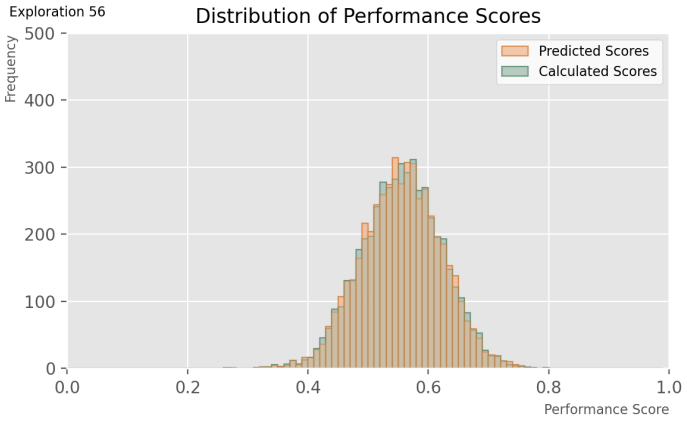
The attributes of this model are:

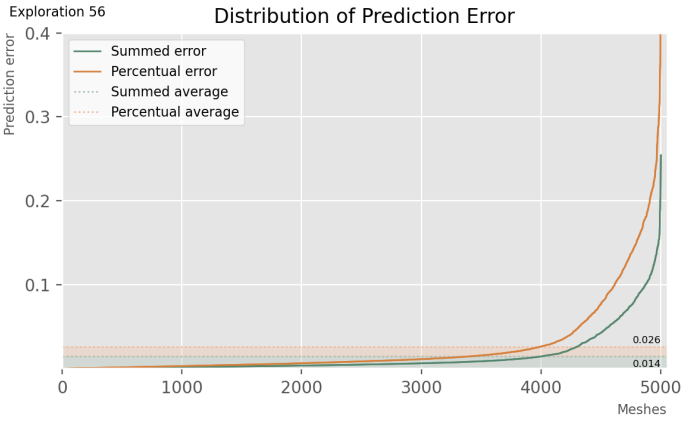
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	**'64-64-1'**

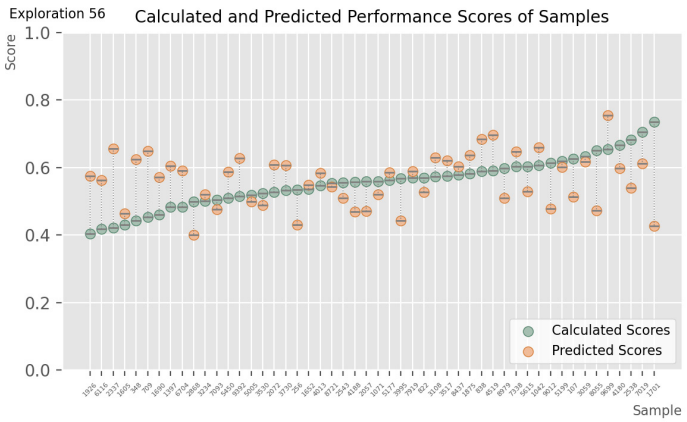


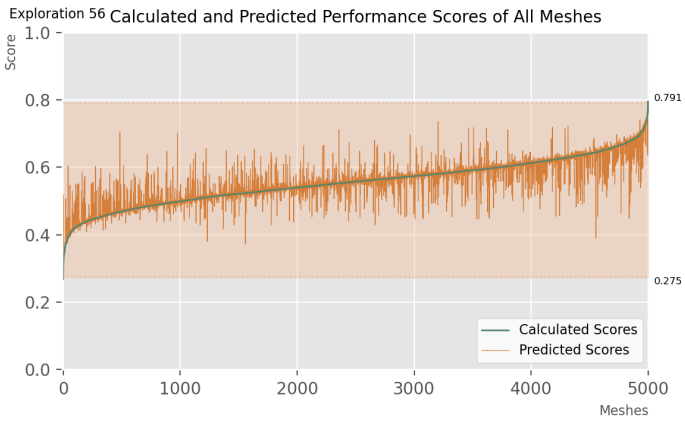


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





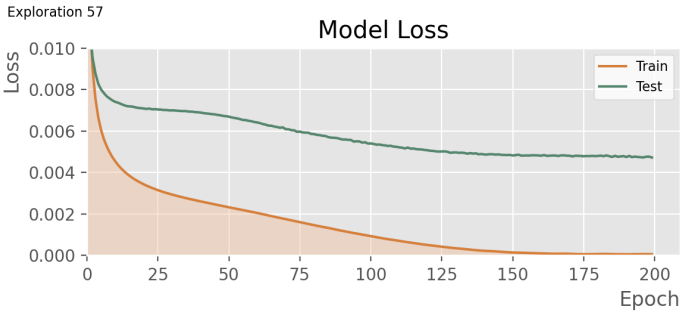




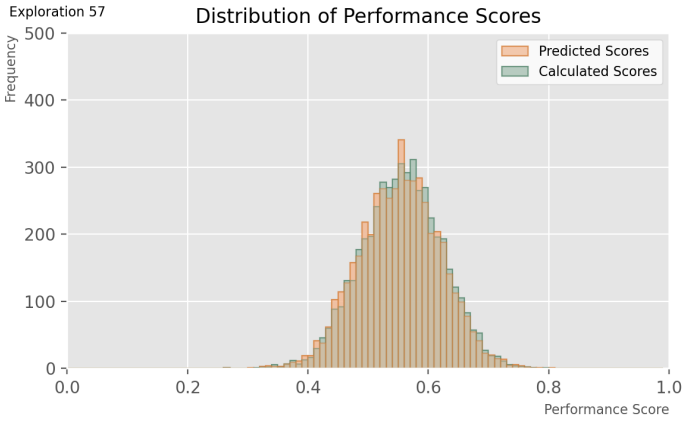
SURROGATE MODEL: EXPLORATION 57

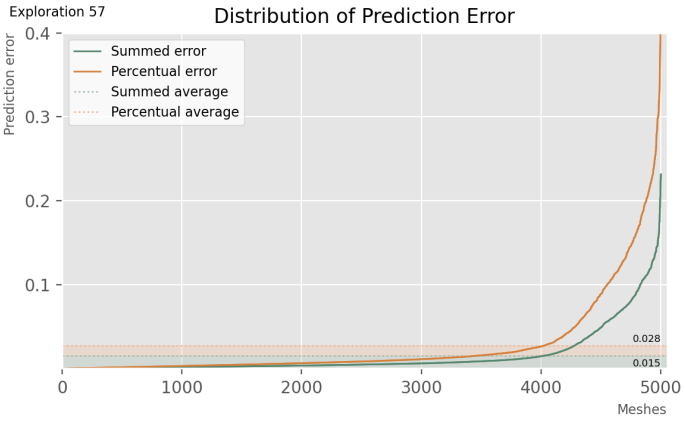
The attributes of this model are:

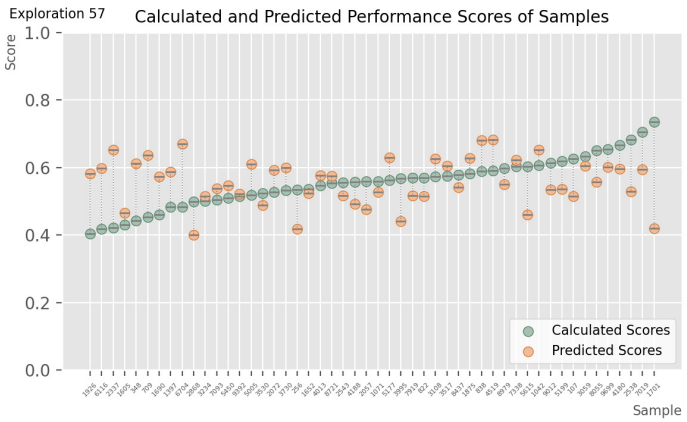
Batch size:	32
Epoch at which training is stopped:	400
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	**'32-16-1'**

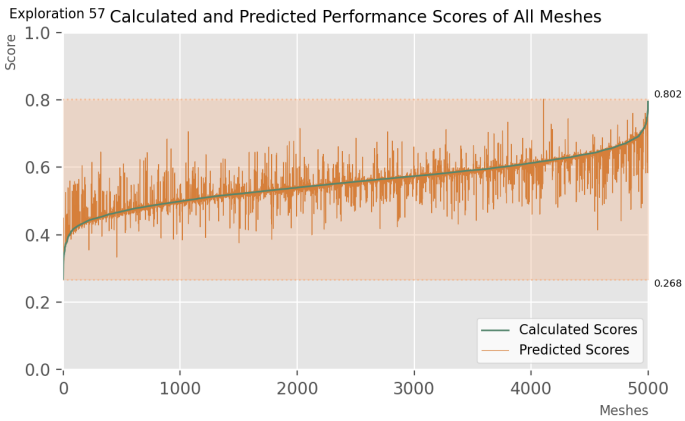


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.





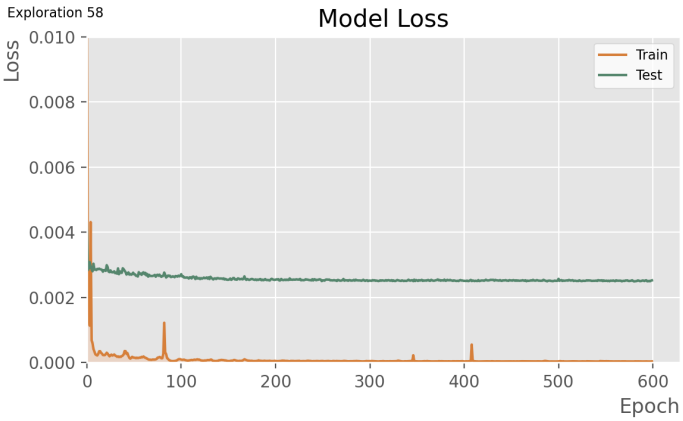




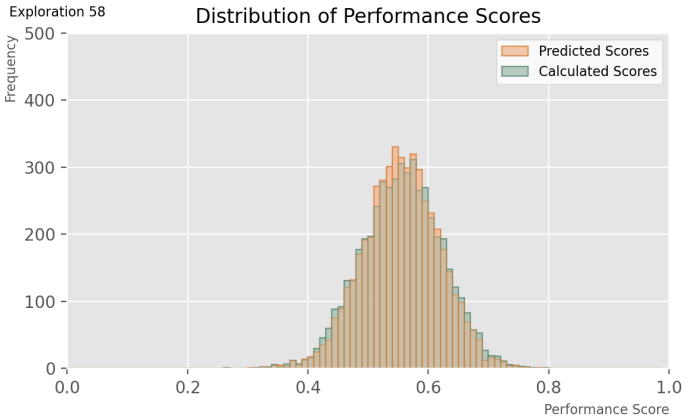
SURROGATE MODEL: 58

The attributes of this model are:

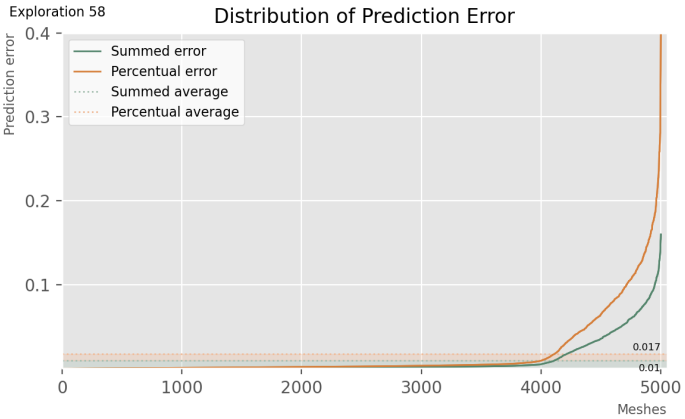
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

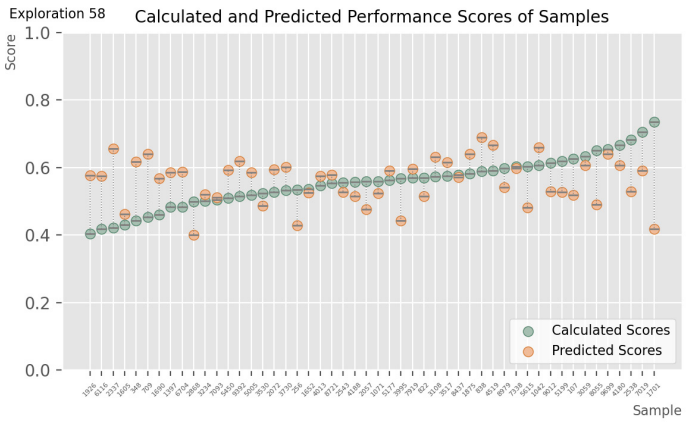


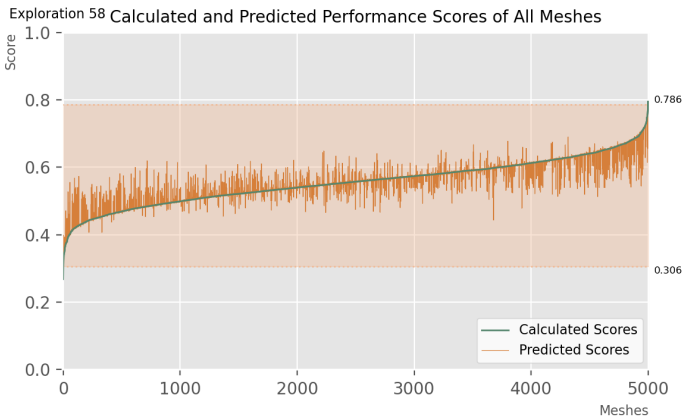
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.



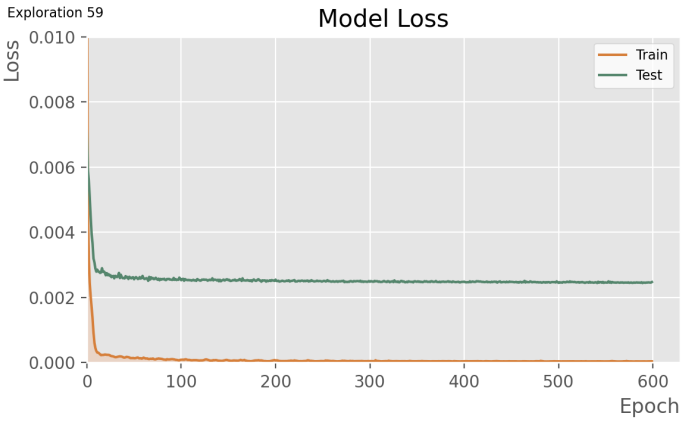




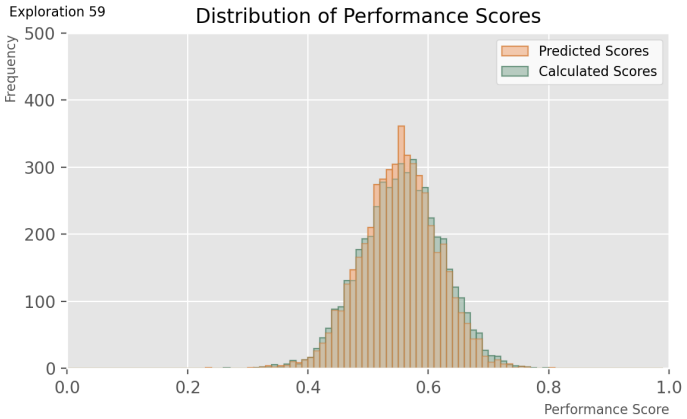
SURROGATE MODEL: 59

The attributes of this model are:

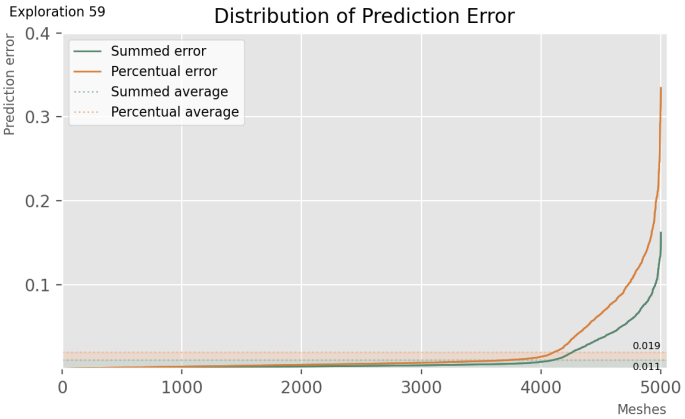
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

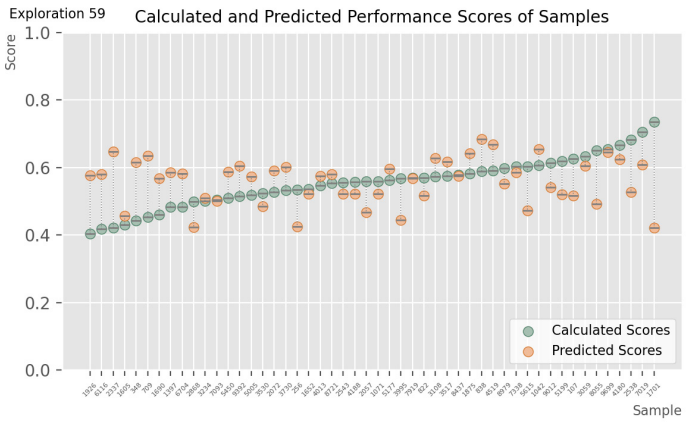


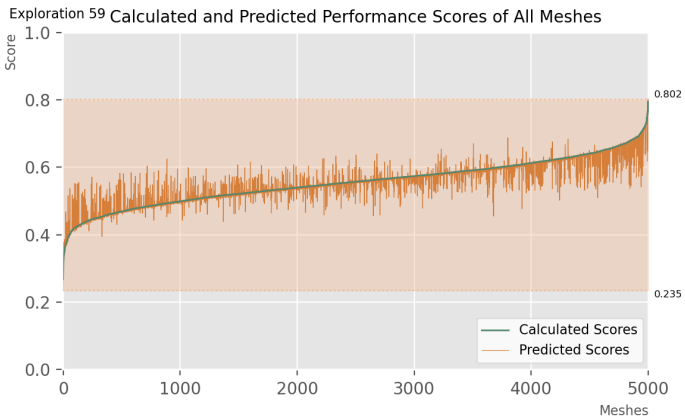
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





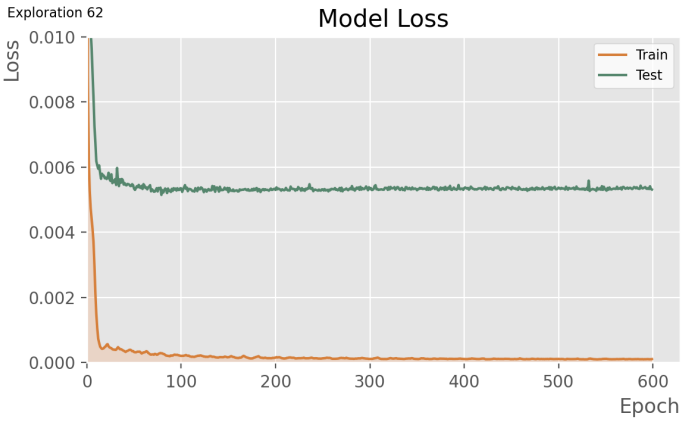


SURROGATE MODEL: 62

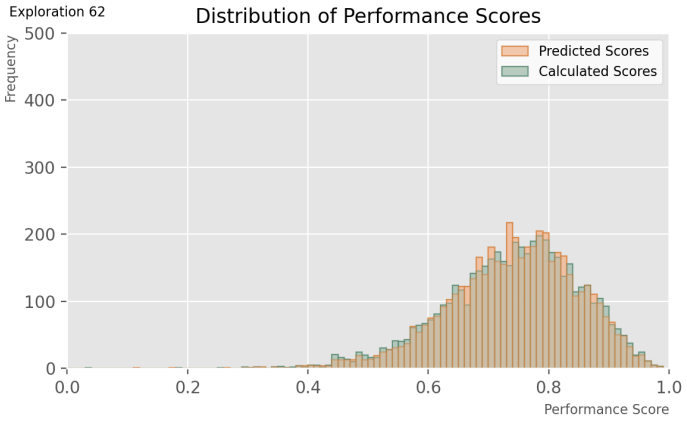
The attributes of this model are:

Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

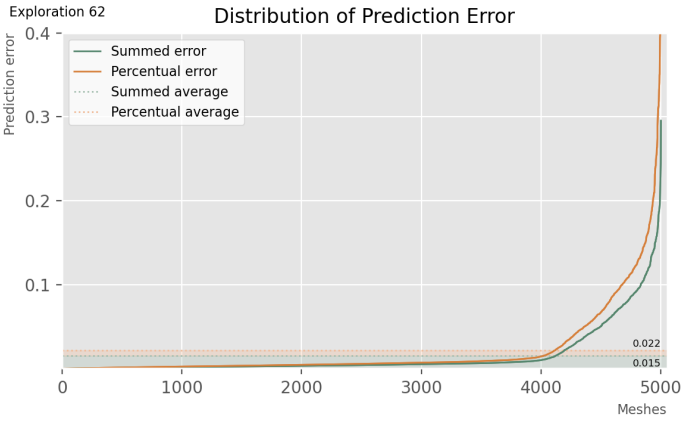
***TRAINED FOR STRUCTURAL PERFORMANCE, ON ADJACENCY MATRIX**

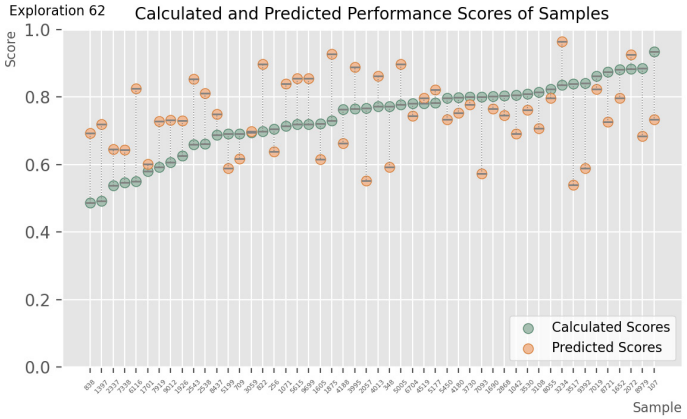


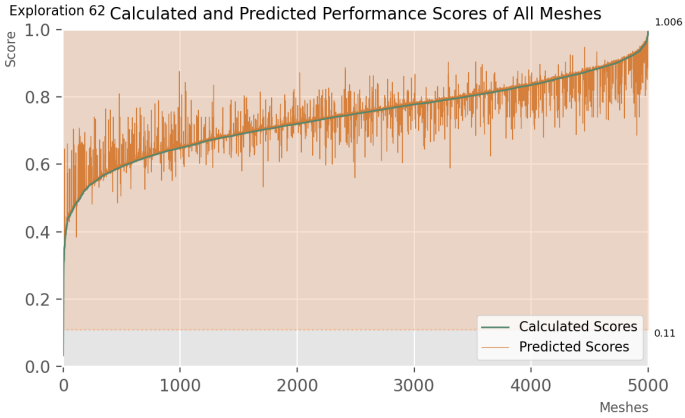
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





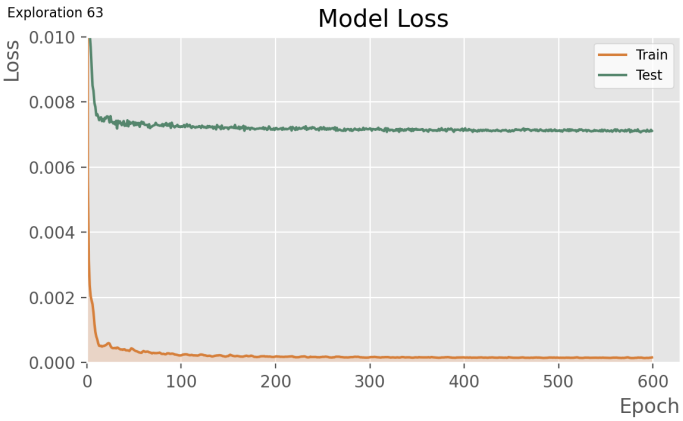


SURROGATE MODEL: 63

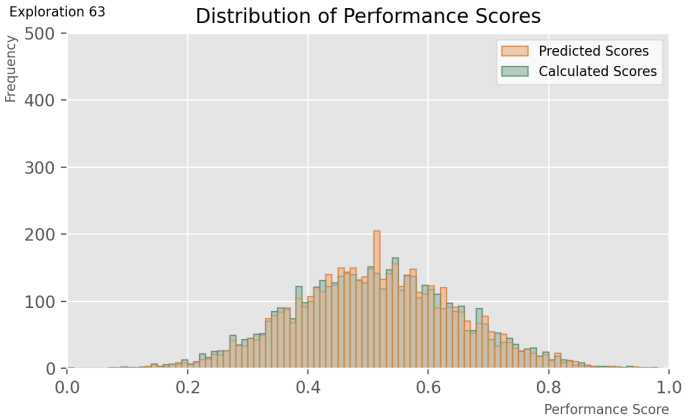
The attributes of this model are:

***TRAINED FOR MATERIAL USE PERFORMANCE, ON ADJACENCY MATRIX**

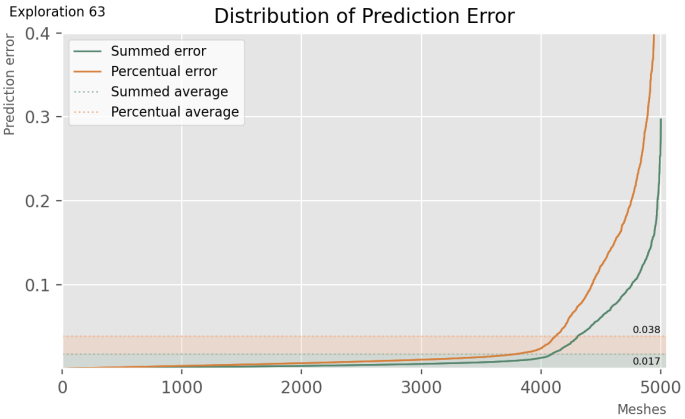
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

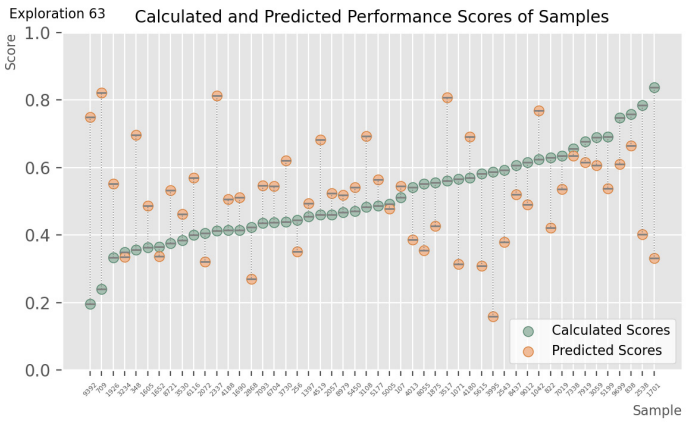


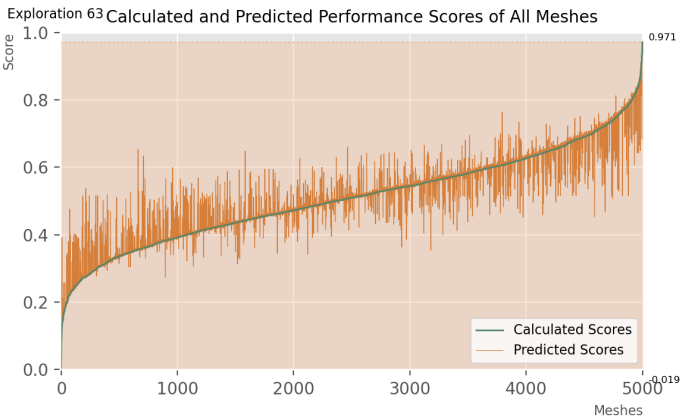
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





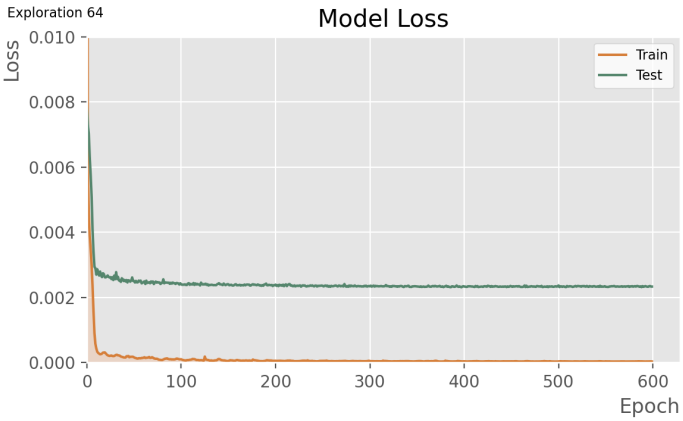


SURROGATE MODEL: 64

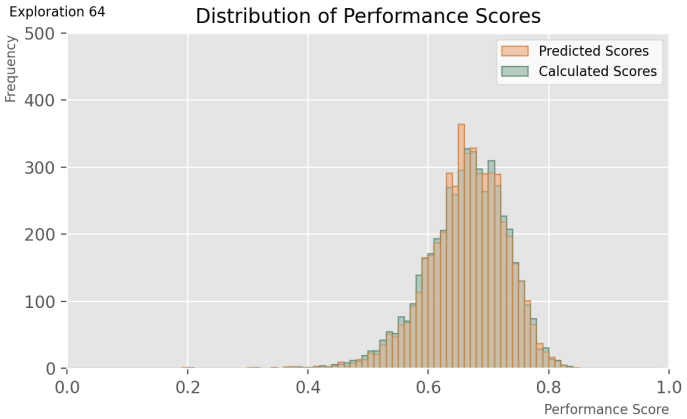
The attributes of this model are:

***TRAINED FOR STRUCTURAL + MATERIAL USE PERFORMANCE, ON ADJACENCY MATRIX**

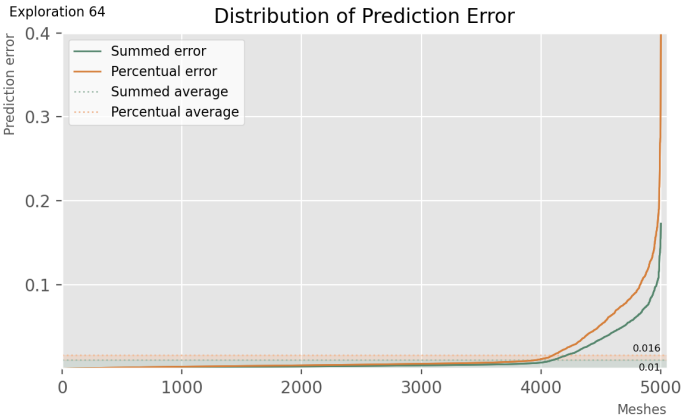
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

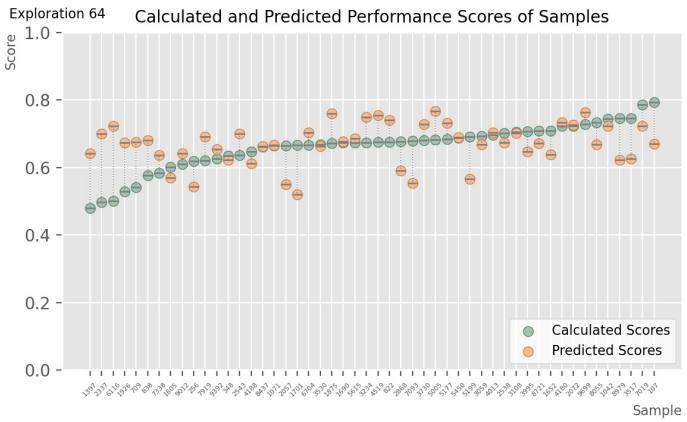


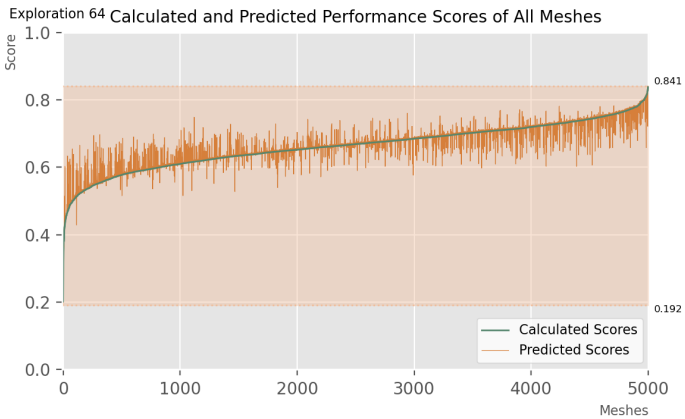
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





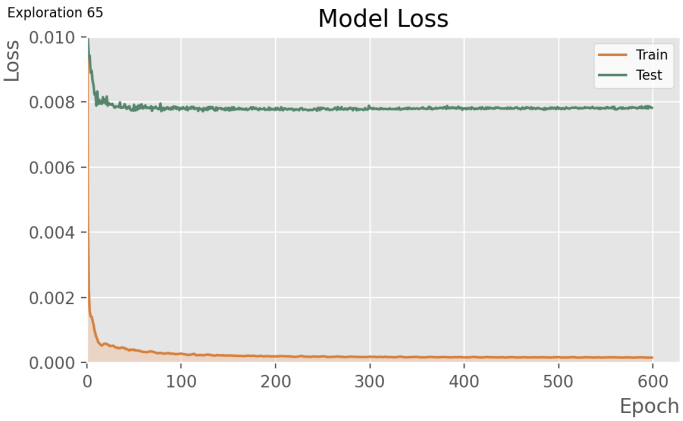


SURROGATE MODEL: 65

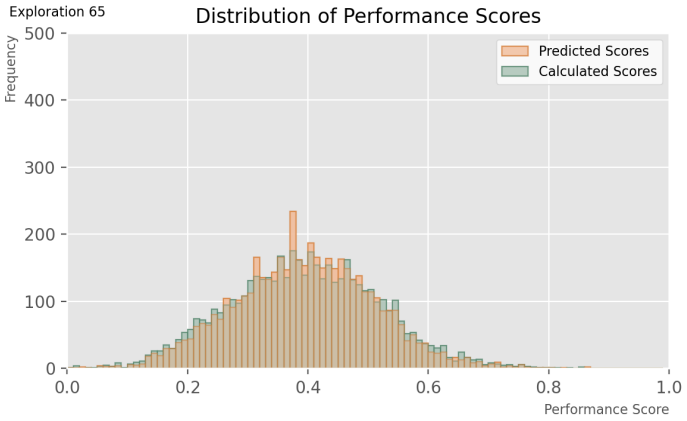
The attributes of this model are:

***TRAINED FOR STOCK SIMILARITY PERFORMANCE, ON ADJACENCY MATRIX**

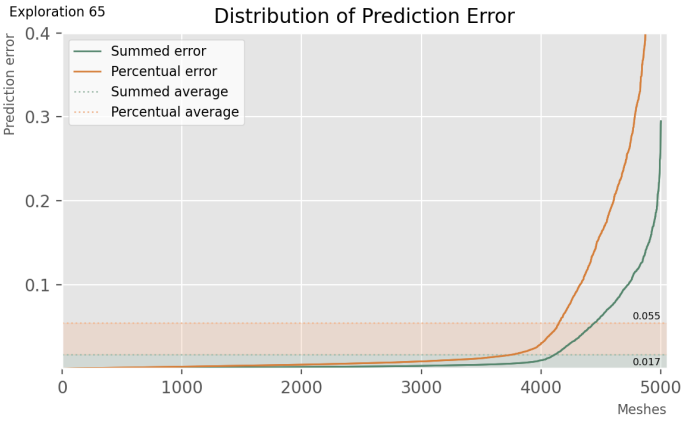
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

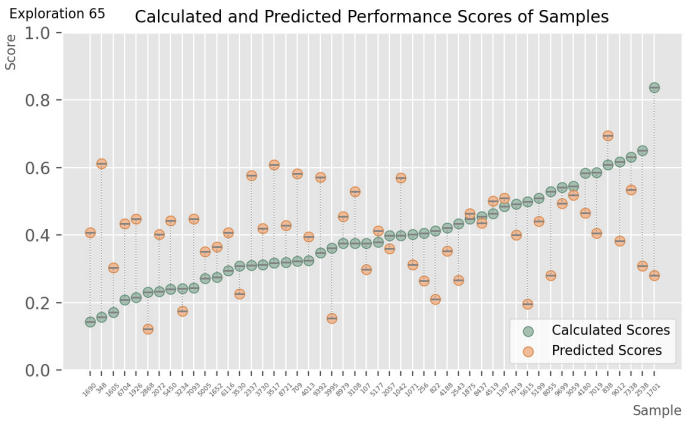


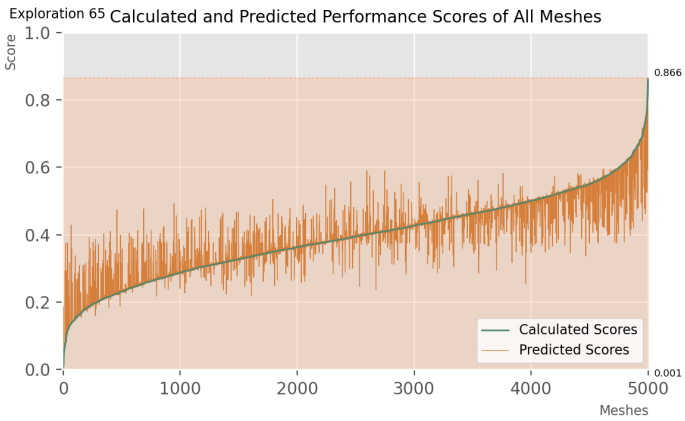
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





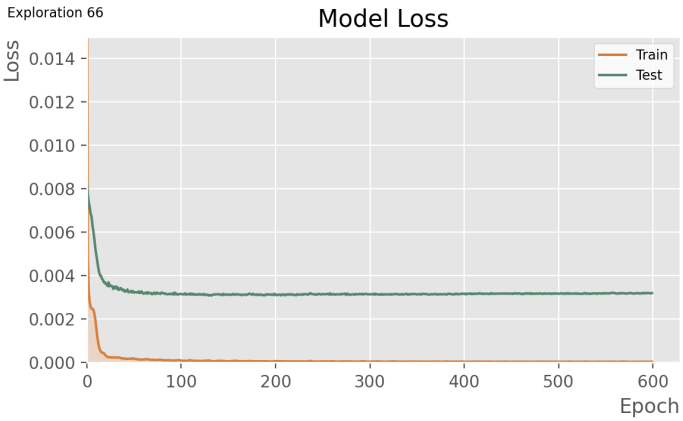


SURROGATE MODEL: 66

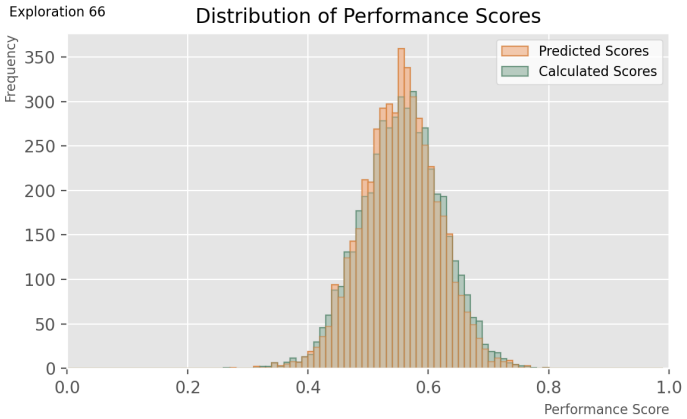
The attributes of this model are:

*** TRAINED FOR OVERALL PERFORMANCE, ON EDGE-VERTEX MATRIX**

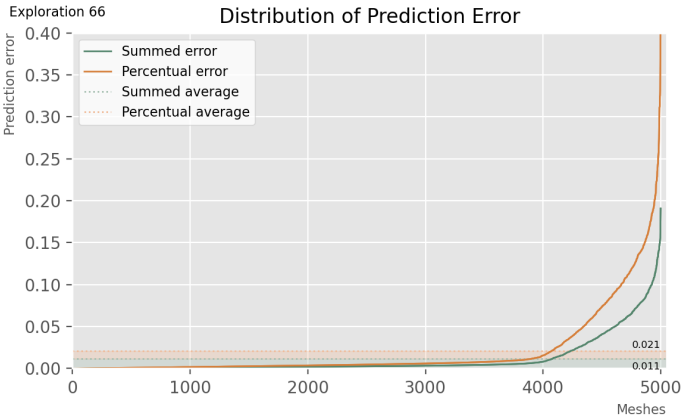
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

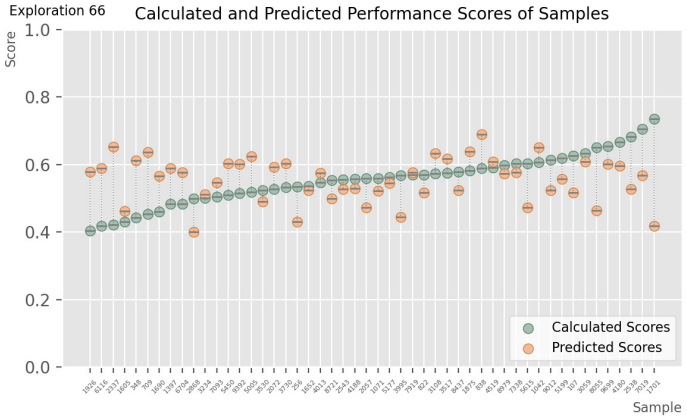


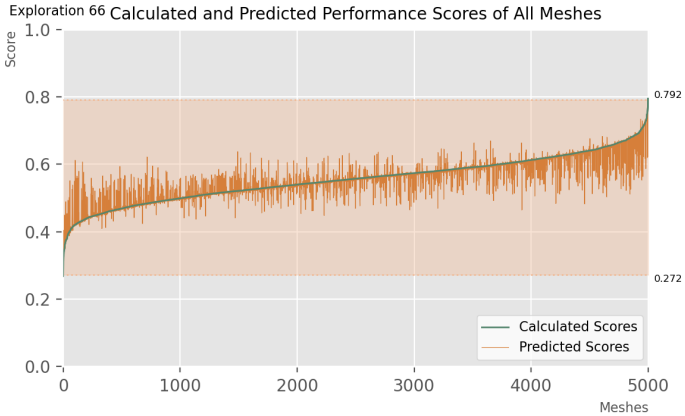
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





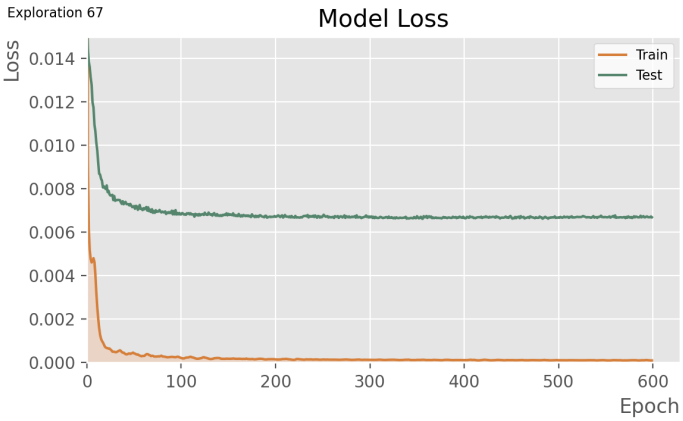


SURROGATE MODEL: 67

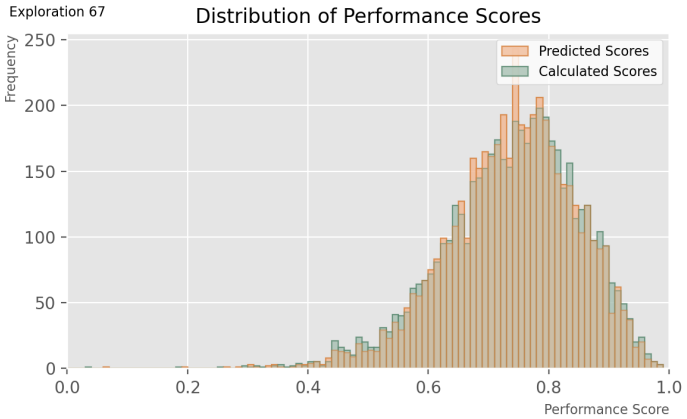
The attributes of this model are:

Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

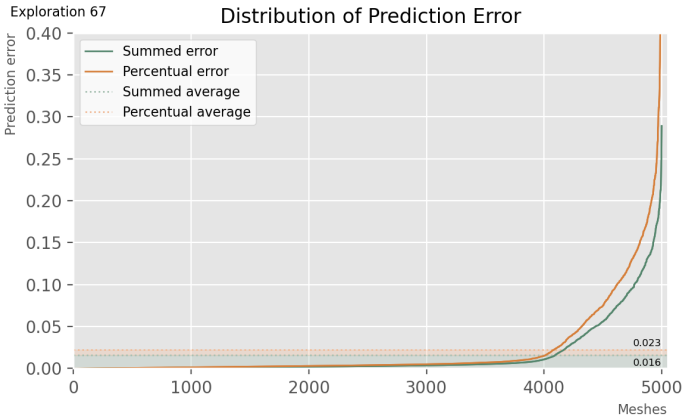
*** TRAINED FOR STRUCTURAL PERFORMANCE, ON EDGE-VERTEX MATRIX**

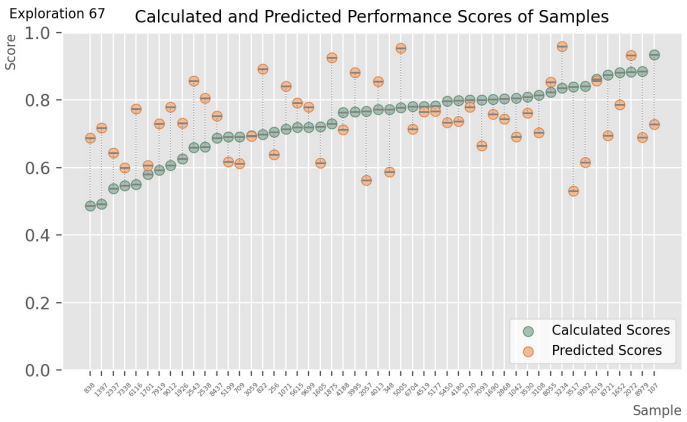


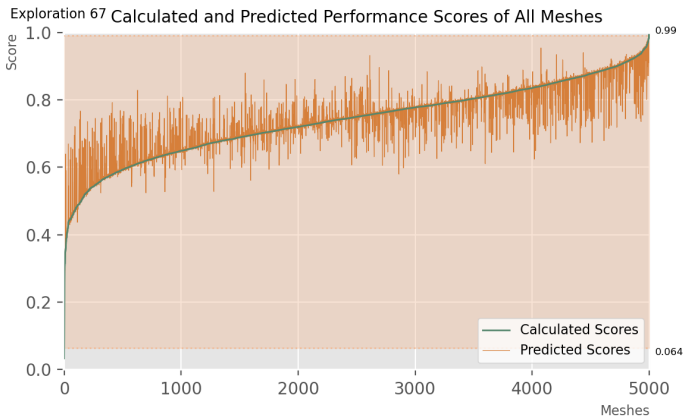
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





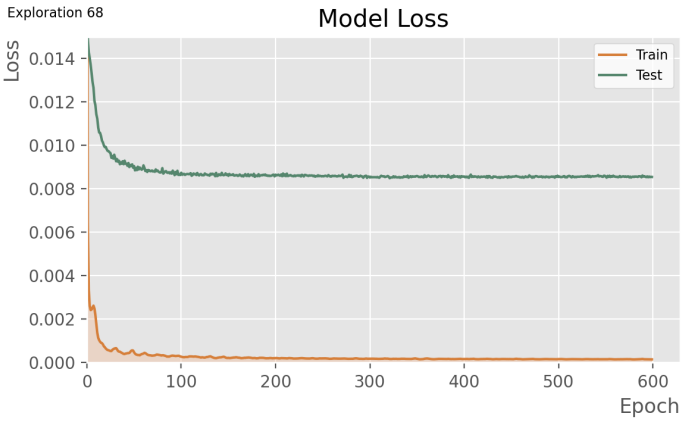


SURROGATE MODEL: 68

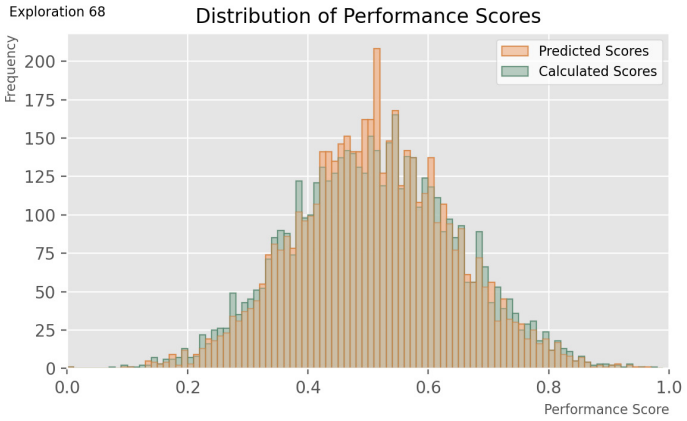
The attributes of this model are:

*** TRAINED FOR MATERIAL USE PERFORMANCE, ON EDGE-VERTEX MATRIX**

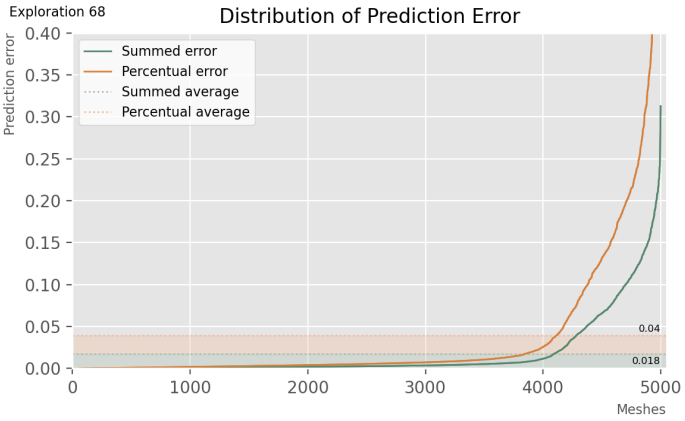
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

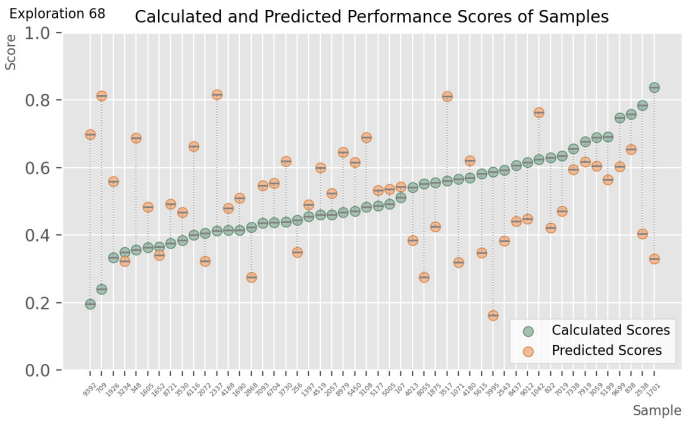


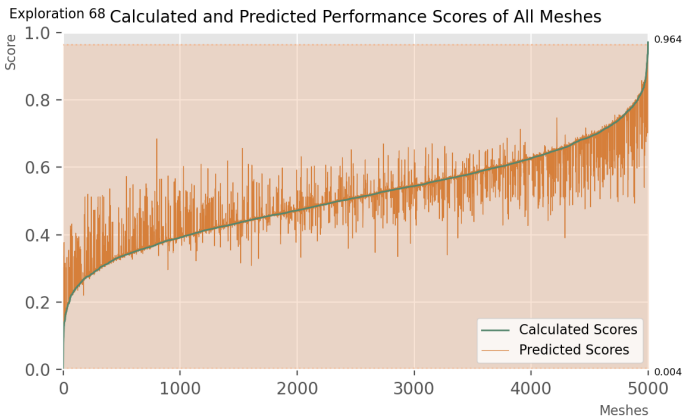
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





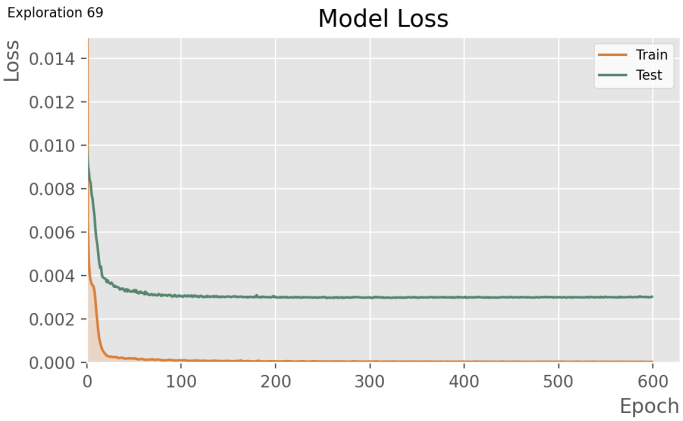


SURROGATE MODEL: 69

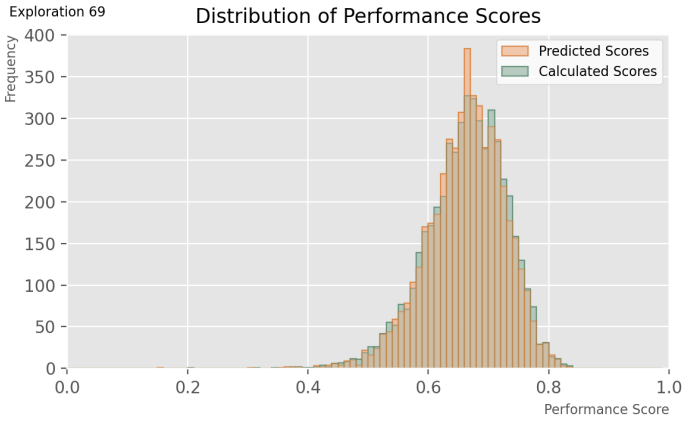
The attributes of this model are:

*** TRAINED FOR STRUCTURAL + MATERIAL USE PERFORMANCE, ON EDGE-VERTEX MATRIX**

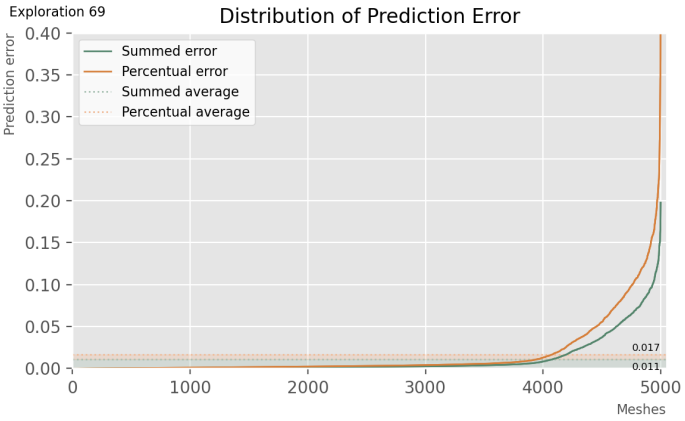
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

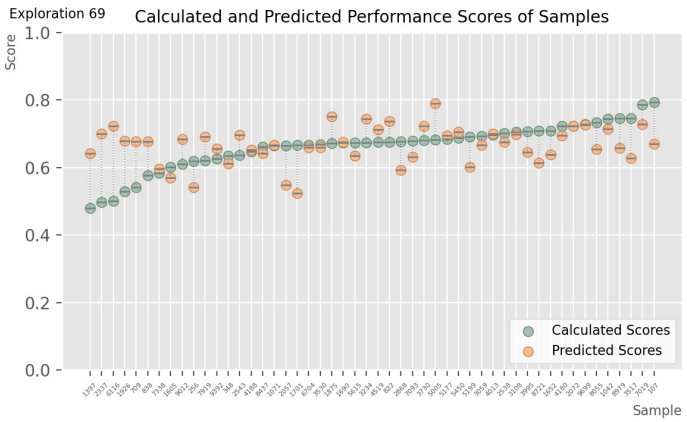


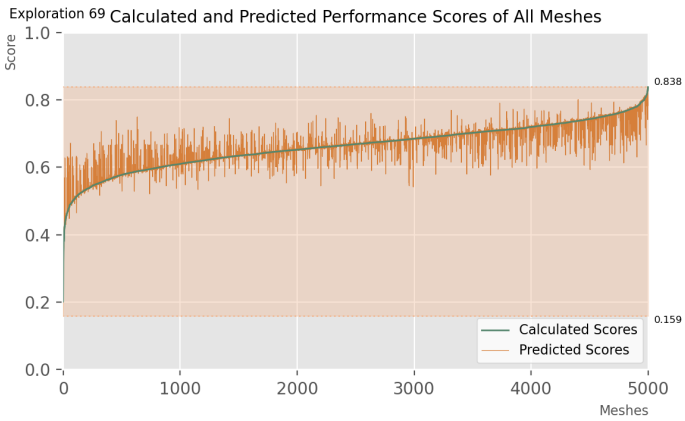
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





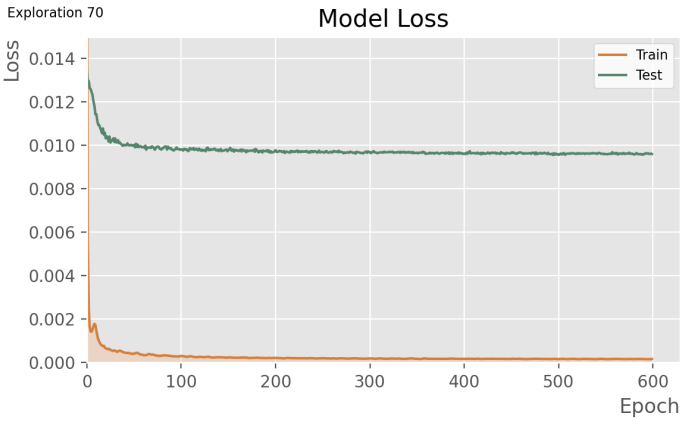


SURROGATE MODEL: 70

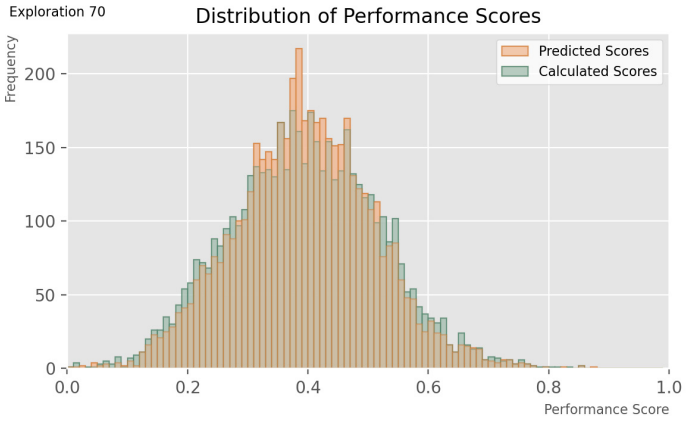
The attributes of this model are:

*** TRAINED FOR STOCK SIMILARITY PERFORMANCE, ON EDGE-VERTEX MATRIX**

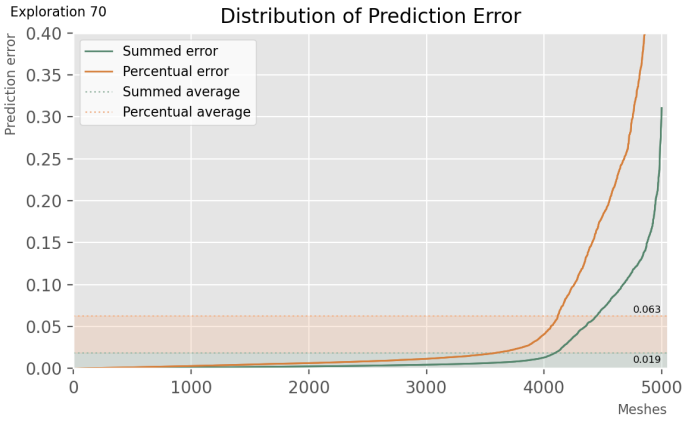
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

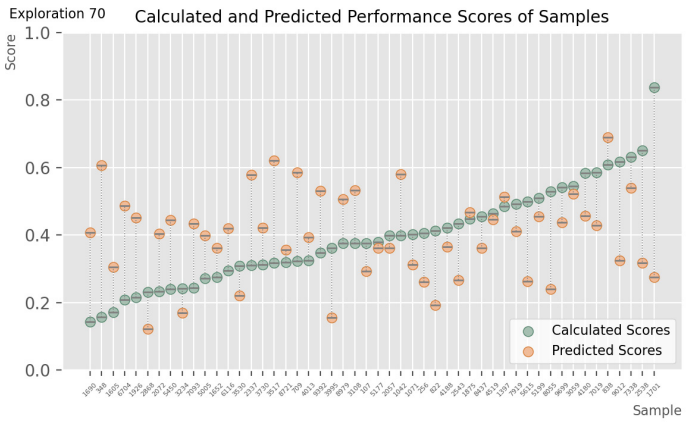


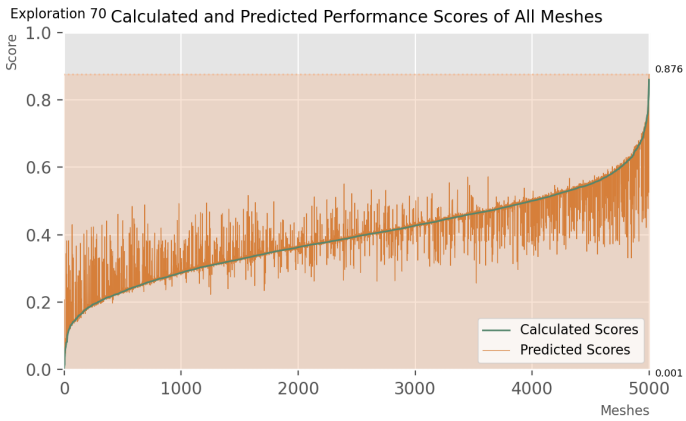
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





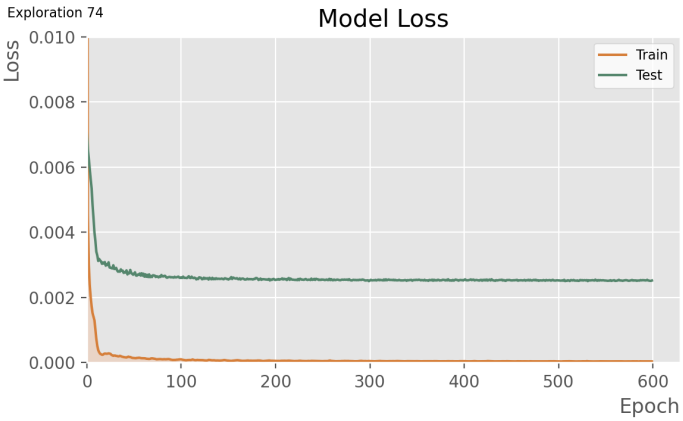


SURROGATE MODEL: 74

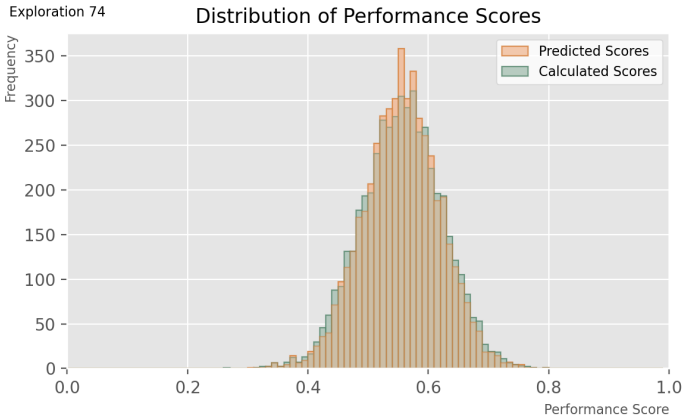
The attributes of this model are:

*** TRAINED FOR OVERALL PERFORMANCE, ON EDGE-VERTEX MATRIX**

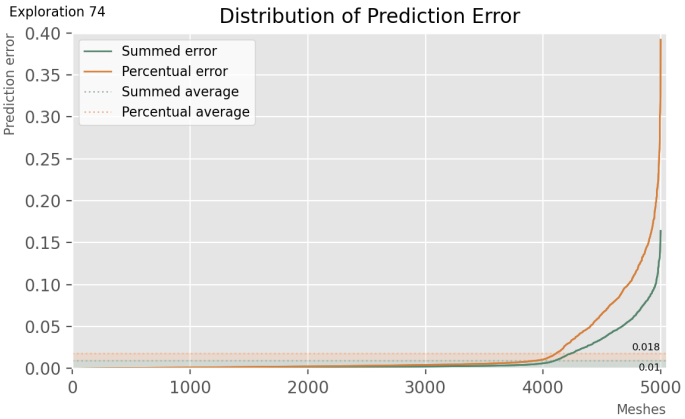
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

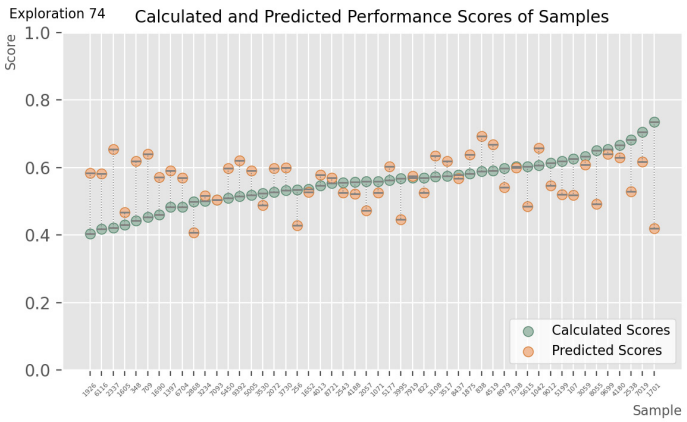


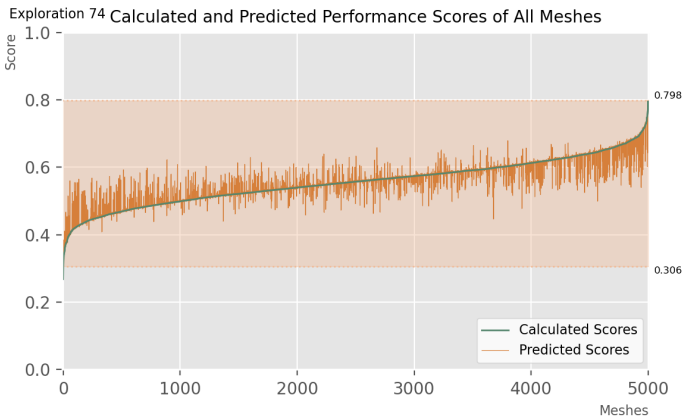
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





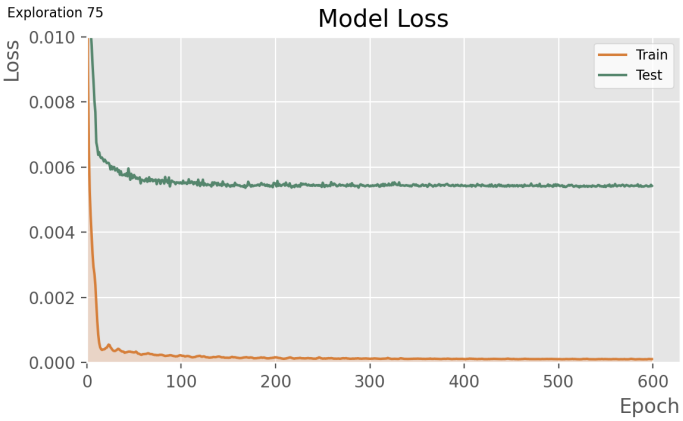


SURROGATE MODEL: 75

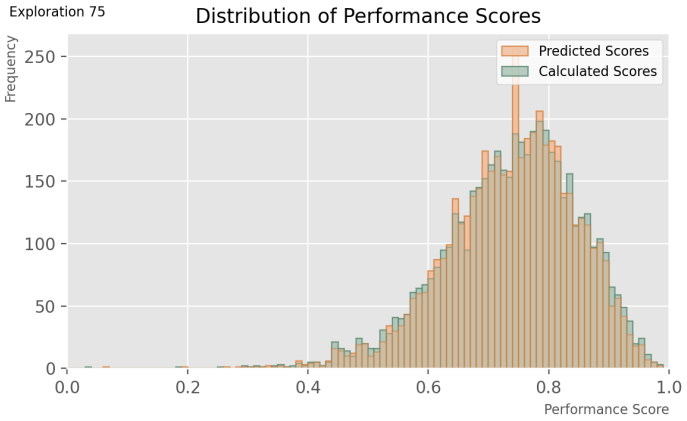
The attributes of this model are:

Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

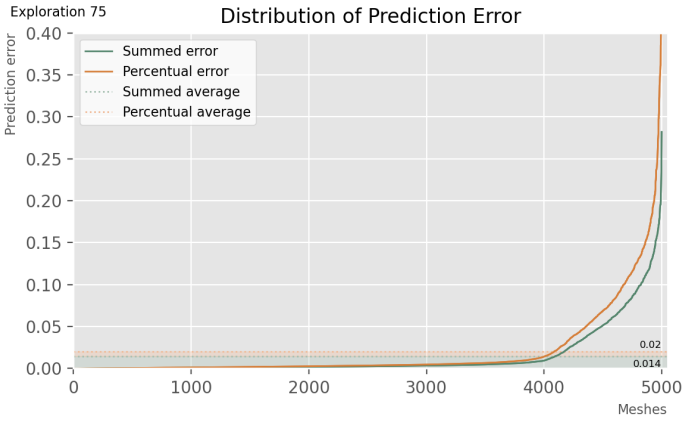
*** TRAINED FOR OVERALL PERFORMANCE, ON HALF ADJACENCY MATRIX**

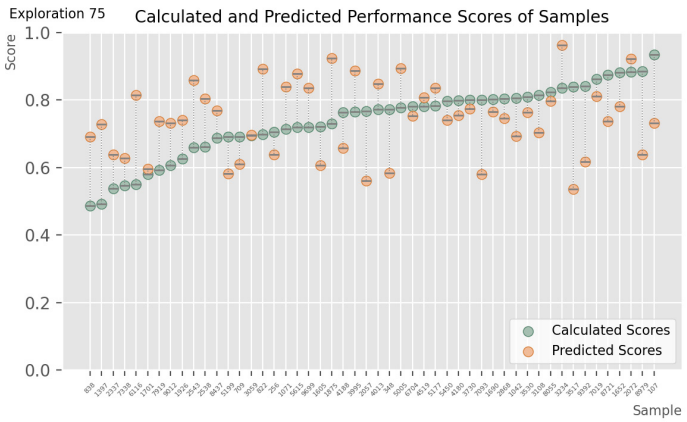


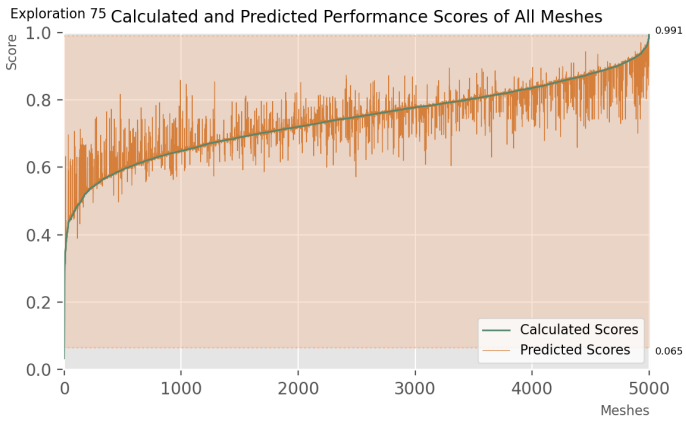
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





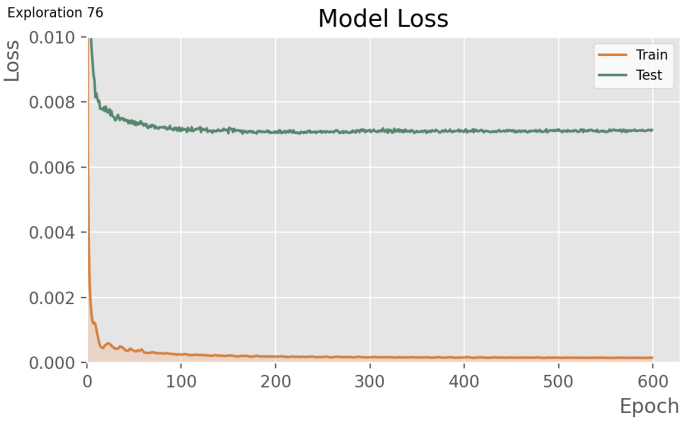


SURROGATE MODEL: 76

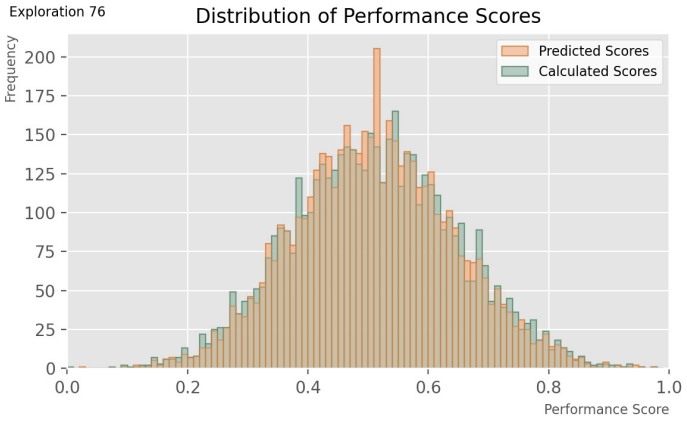
The attributes of this model are:

Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

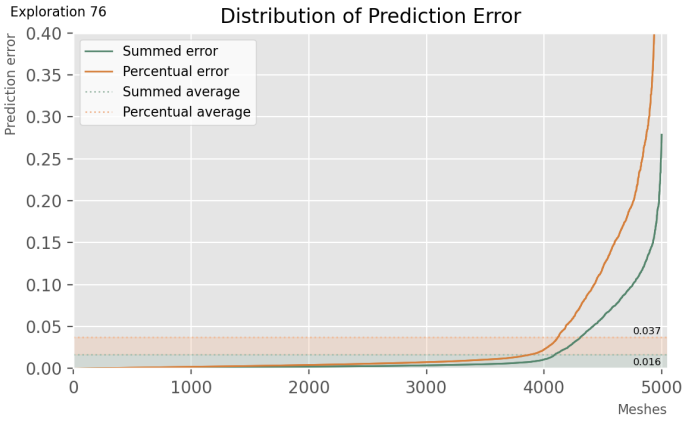
*** TRAINED FOR STRUCTURAL PERFORMANCE, ON HALF ADJACENCY MATRIX**

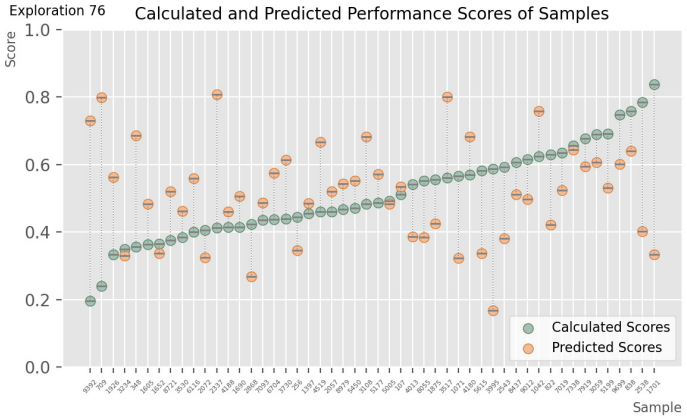


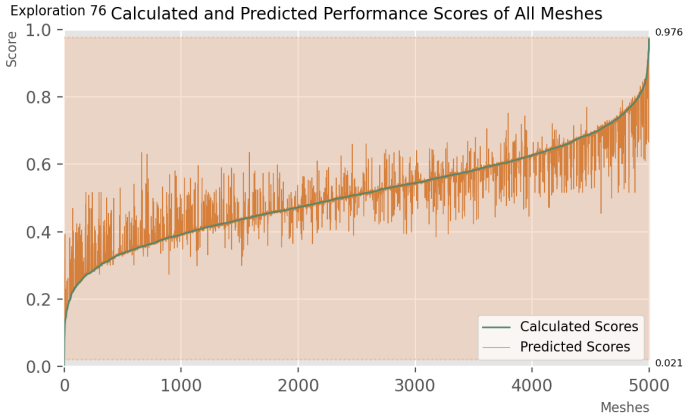
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





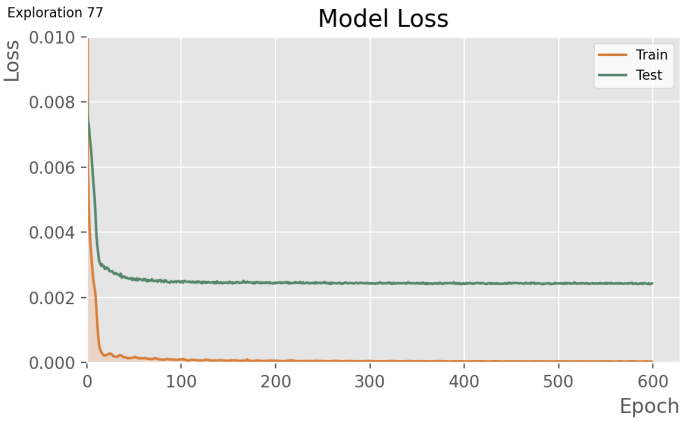


SURROGATE MODEL: 77

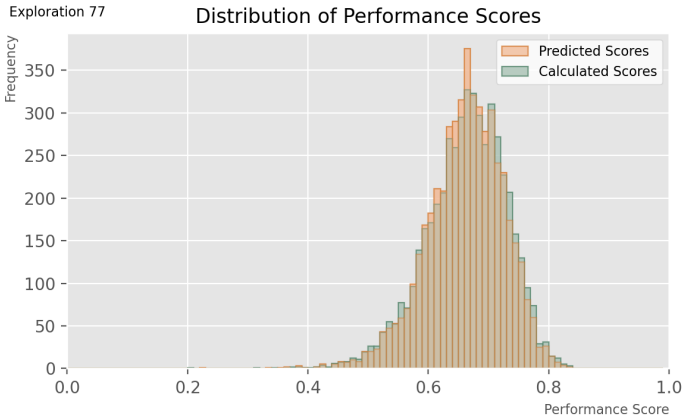
The attributes of this model are:

*** TRAINED FOR STRUCTURAL + MATERIAL USE PERFORMANCE, ON HALF ADJACENCY MATRIX**

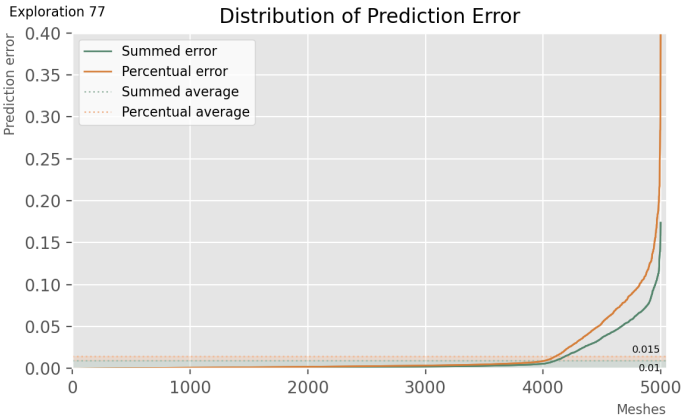
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

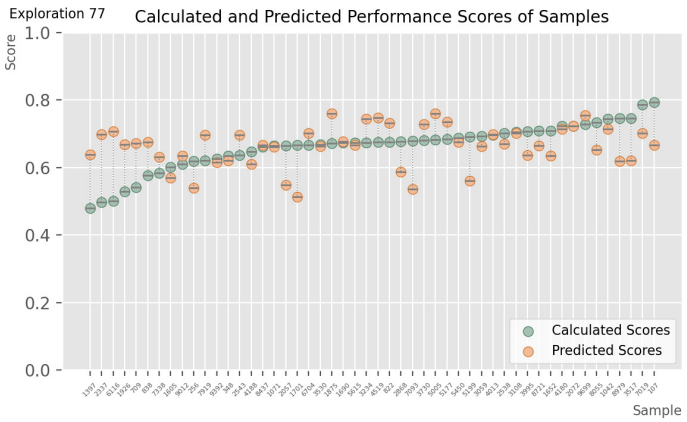


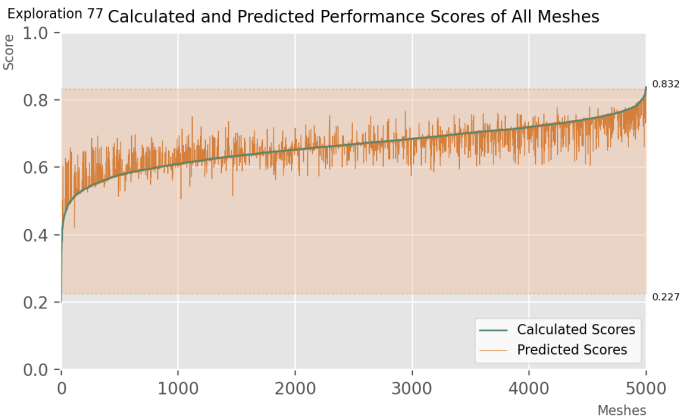
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





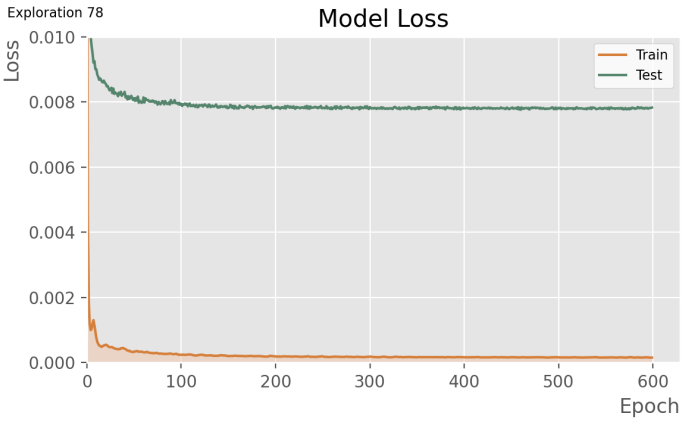


SURROGATE MODEL: 78

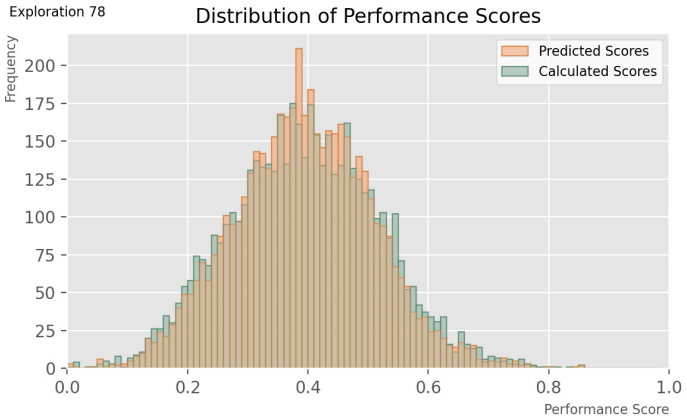
The attributes of this model are:

Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.00001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

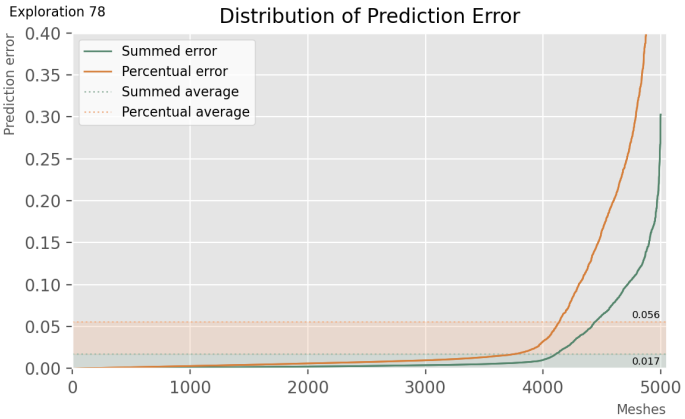
*** TRAINED FOR STOCK SIMILARITY PERFORMANCE, ON HALF ADJACENCY MATRIX**

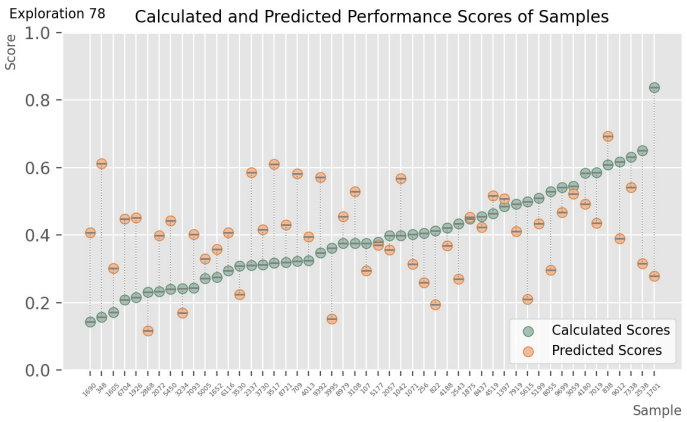


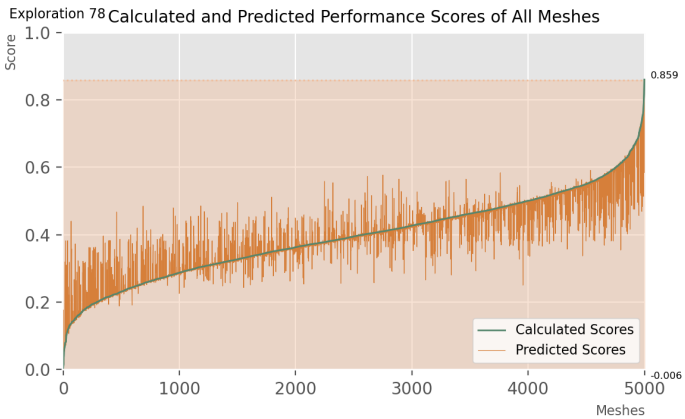
***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.





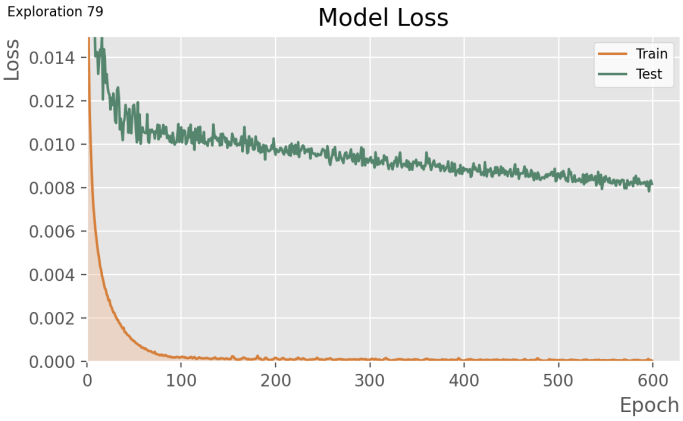


SURROGATE MODEL: 79

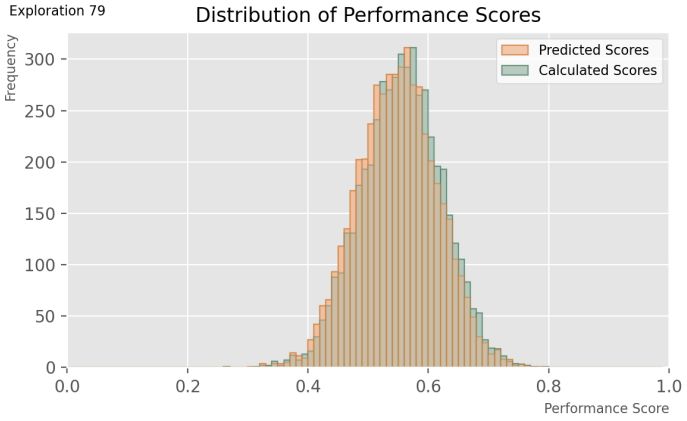
The attributes of this model are:

*** TRAINED FOR OVERALL PERFORMANCE, ON ENCODED (VAE) DATA**

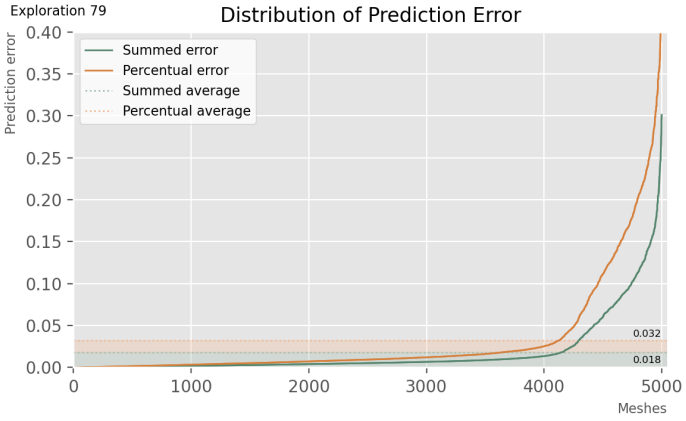
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.0001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

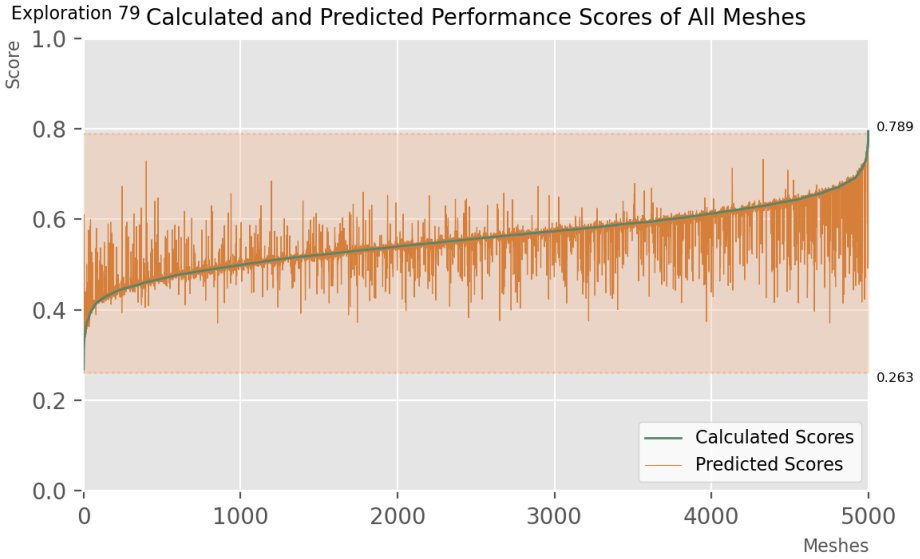


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.



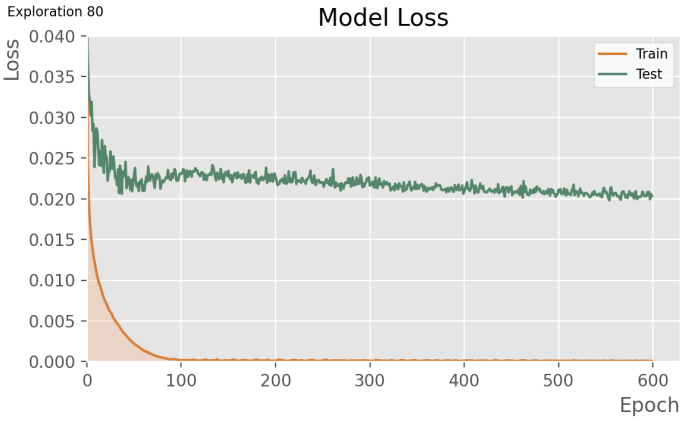


SURROGATE MODEL: 80

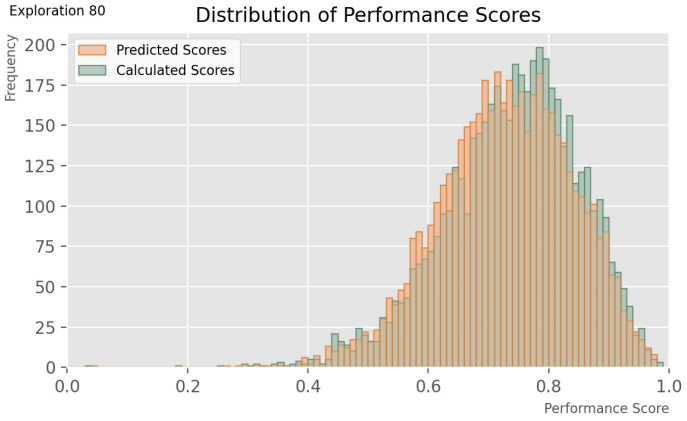
The attributes of this model are:

*** TRAINED FOR STRUCTURAL PERFORMANCE, ON ENCODED (VAE) DATA**

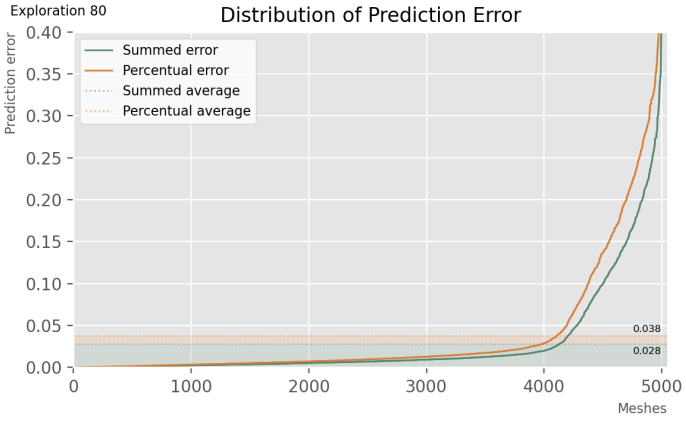
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.0001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

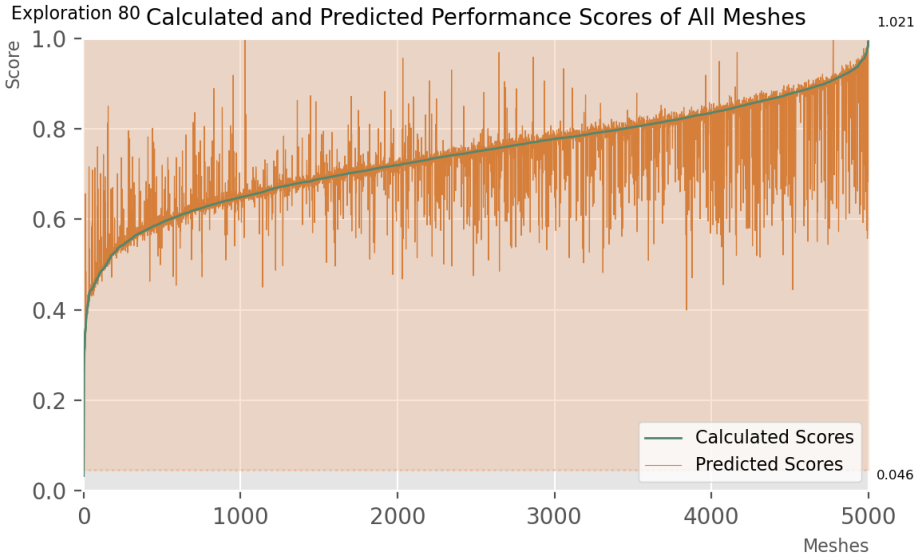


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.



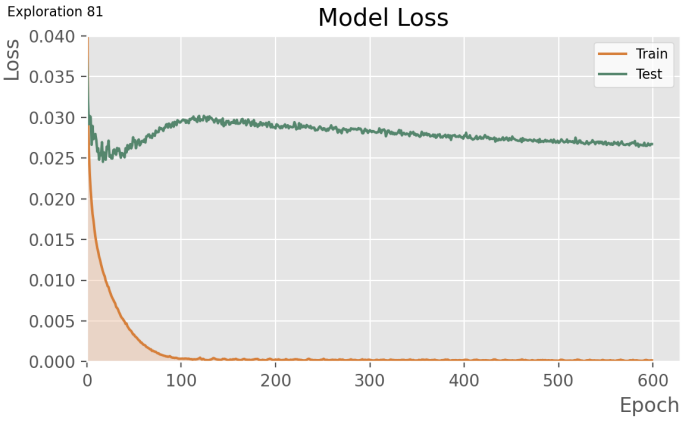


SURROGATE MODEL: 81

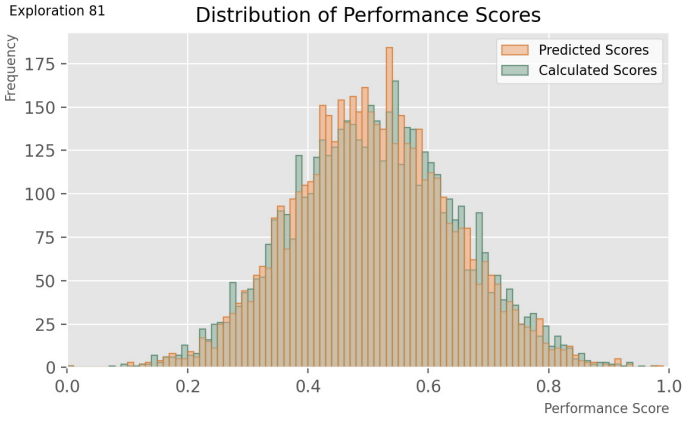
The attributes of this model are:

*** TRAINED FOR MATERIAL USE PERFORMANCE, ON ENCODED (VAE) DATA**

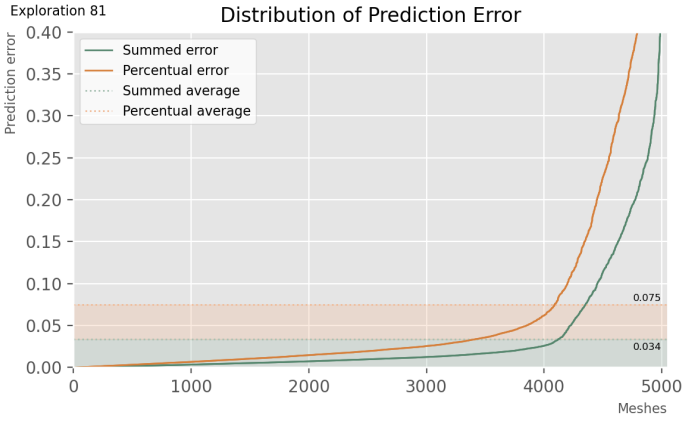
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.0001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

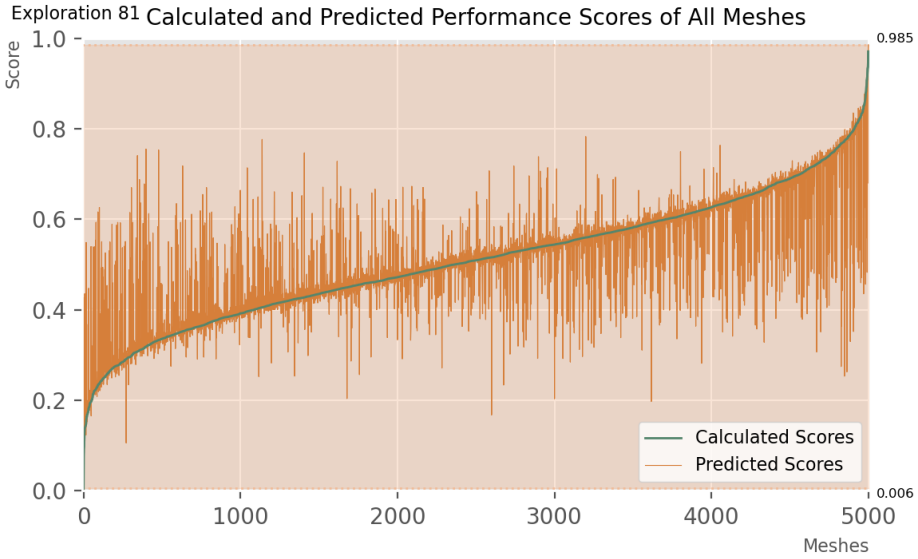


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.



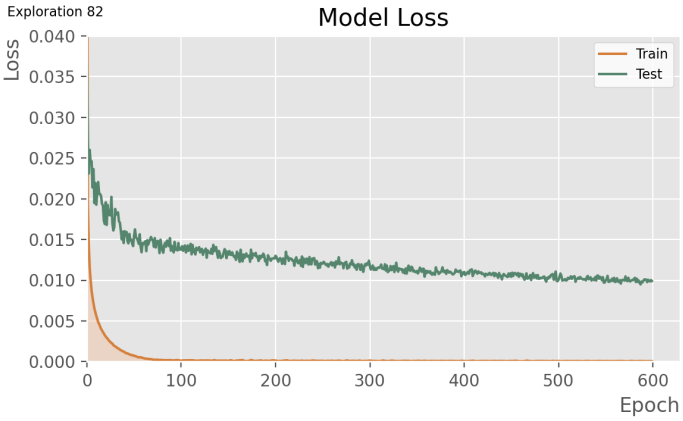


SURROGATE MODEL: 82

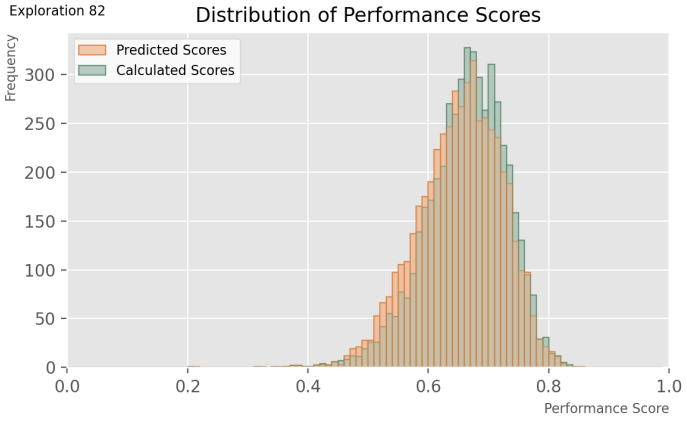
The attributes of this model are:

*** TRAINED FOR STRUCTURAL + MATERIAL USE PERFORMANCE, ON ENCODED (VAE) DATA**

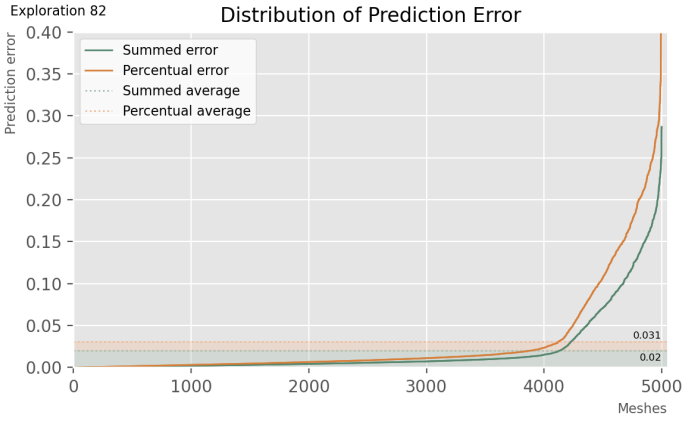
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.0001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

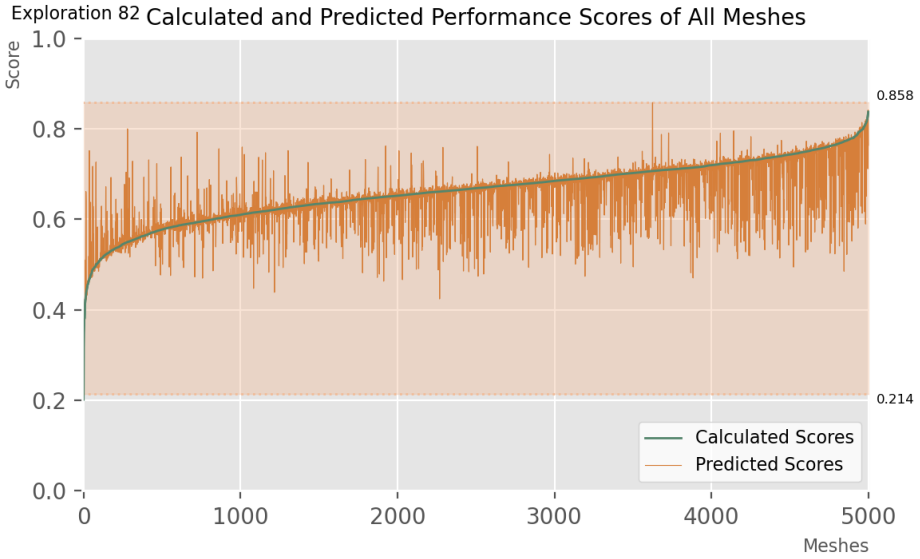


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.



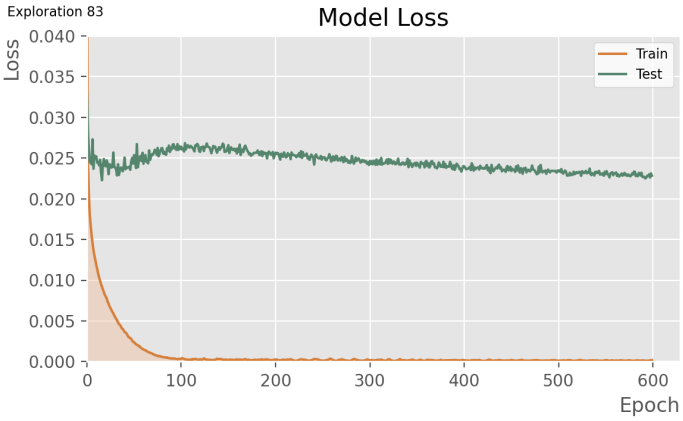


SURROGATE MODEL: 83

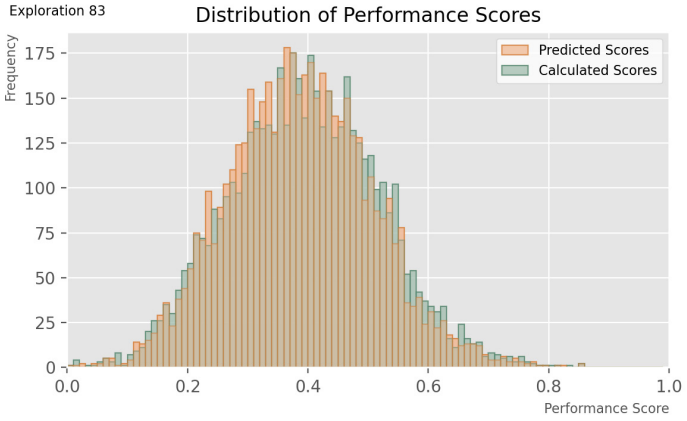
The attributes of this model are:

*** TRAINED FOR STOCK SIMILARITY PERFORMANCE, ON ENCODED (VAE) DATA**

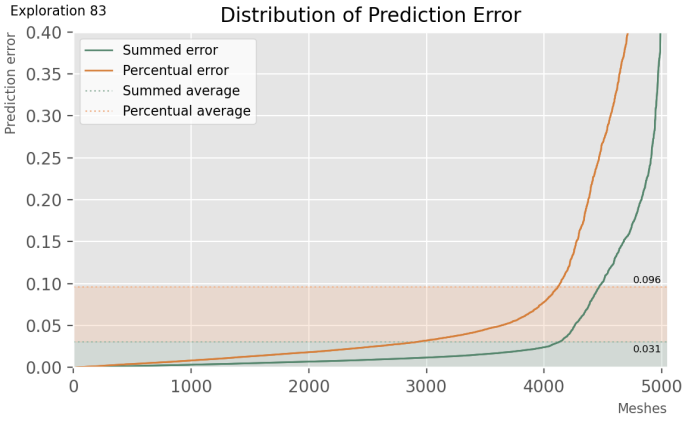
Batch size:	32
Epoch at which training is stopped:	600
Learning rate:	0.0001
Loss types:	Mean squared error
Layer activation type:	Relu
Metrics:	Accuracy
Optimizer type:	Adam
Clipnorm:	1
Addition of normalization layers:	No additional normalization layers
Addition of dropout layers:	No dropout layers
Architecture:	'256-128-64'

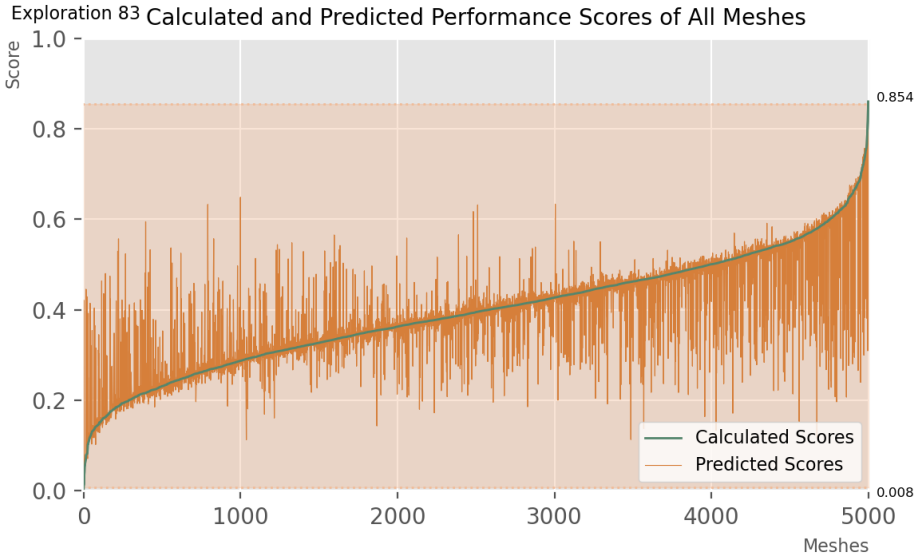


***NOTE:** For the above graph, limits of the 'epochs' axis were adapted to fit data.



***NOTE:** For the above graph, limits of the 'frequency' axis were adapted to fit data.

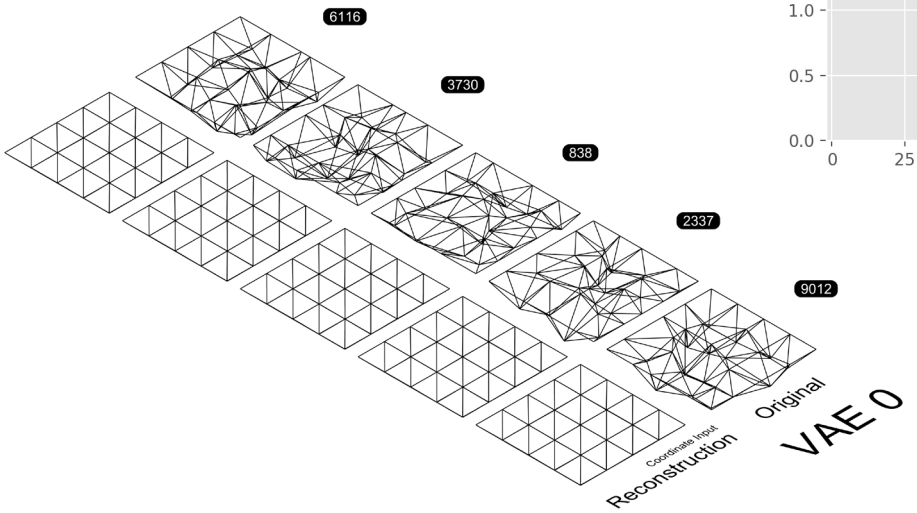




Appendix V

VAE model results

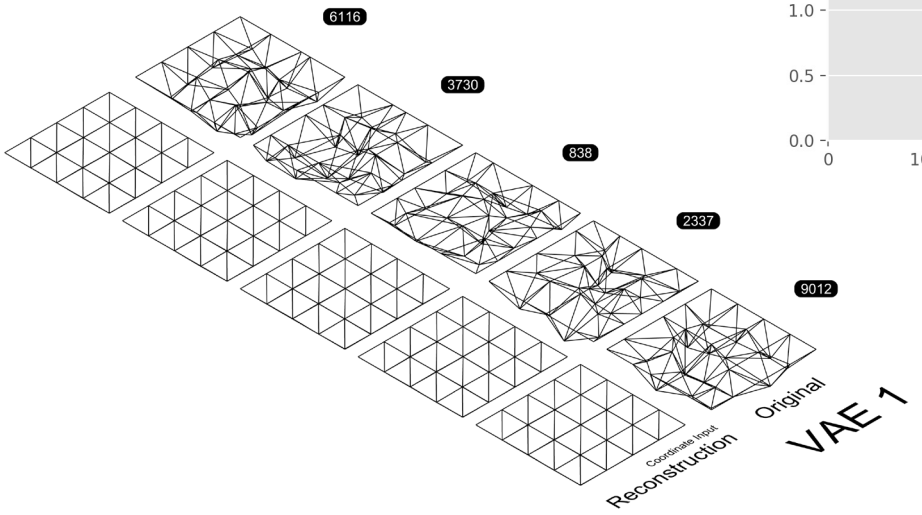
VAE MODEL: EXPLORATION 0

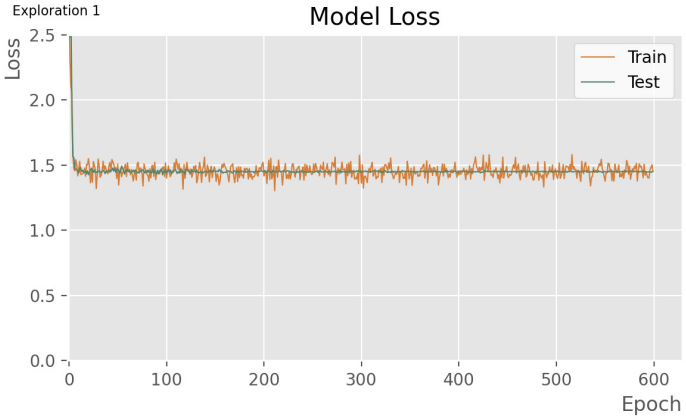


The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	1
Architecture:	'82-41'

VAE MODEL: EXPLORATION 1

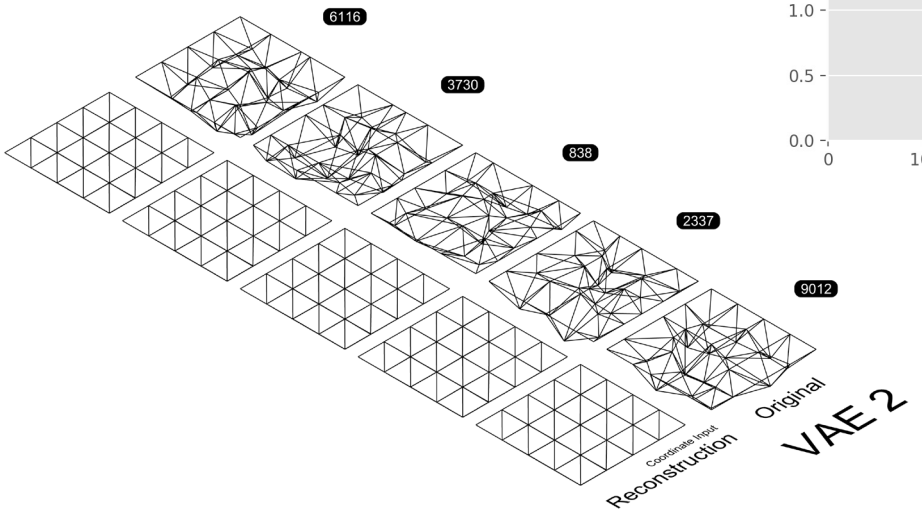


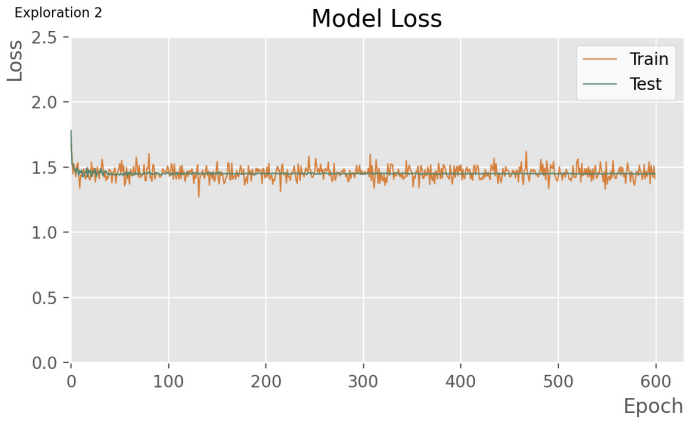


The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	1
Architecture:	'41'

VAE MODEL: EXPLORATION 2

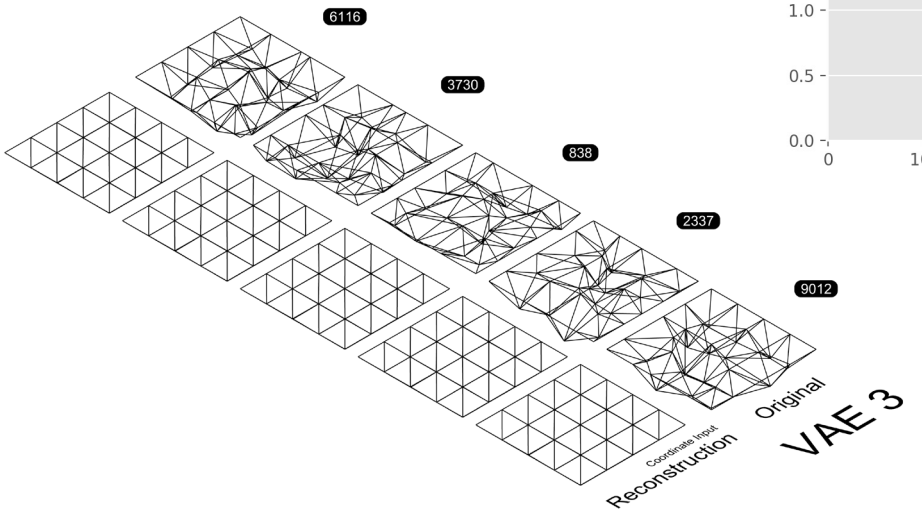


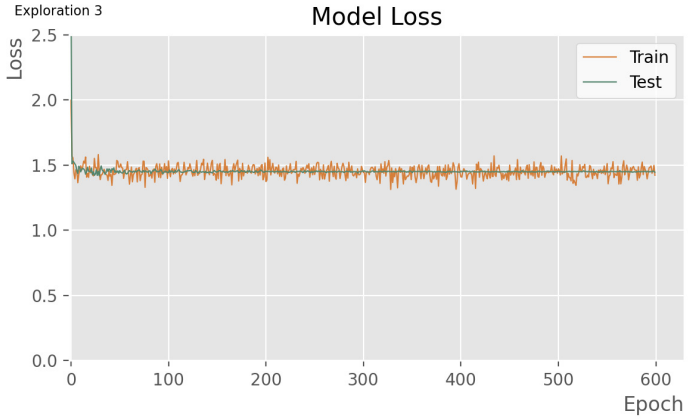


The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	1
Architecture:	**'82-41-21'**

VAE MODEL: EXPLORATION 3

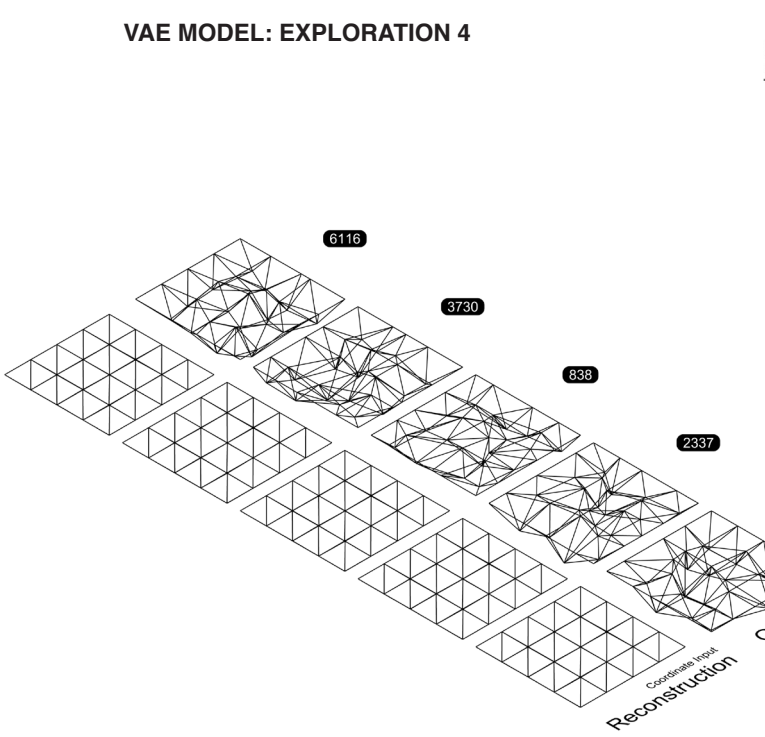




The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	1
Architecture:	**'64-32'**

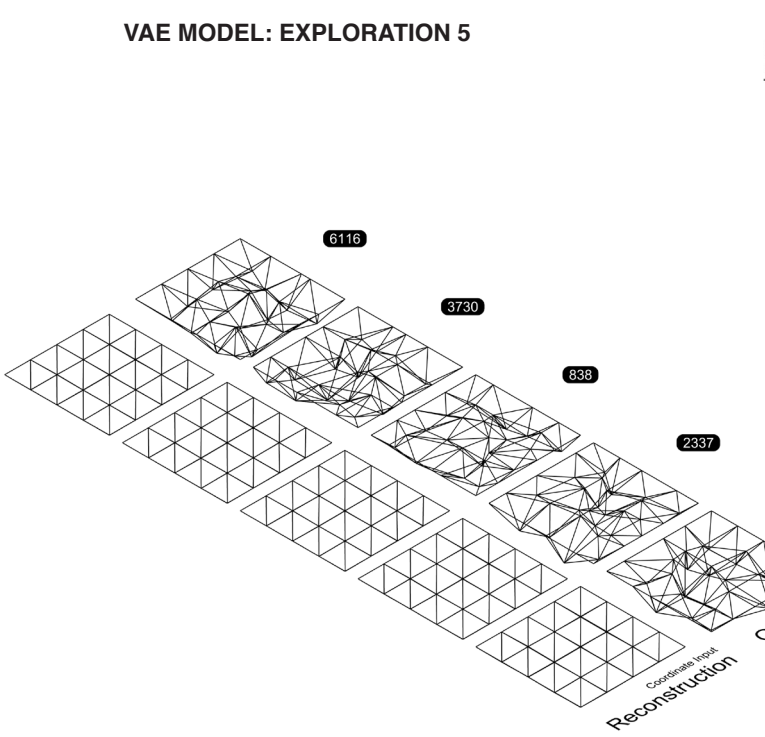
VAE MODEL: EXPLORATION 4



The attributes of this model are:

Latent Dimension	**2**
Multiplication of KL_loss	**1**
Architecture:	**‘82-41’**

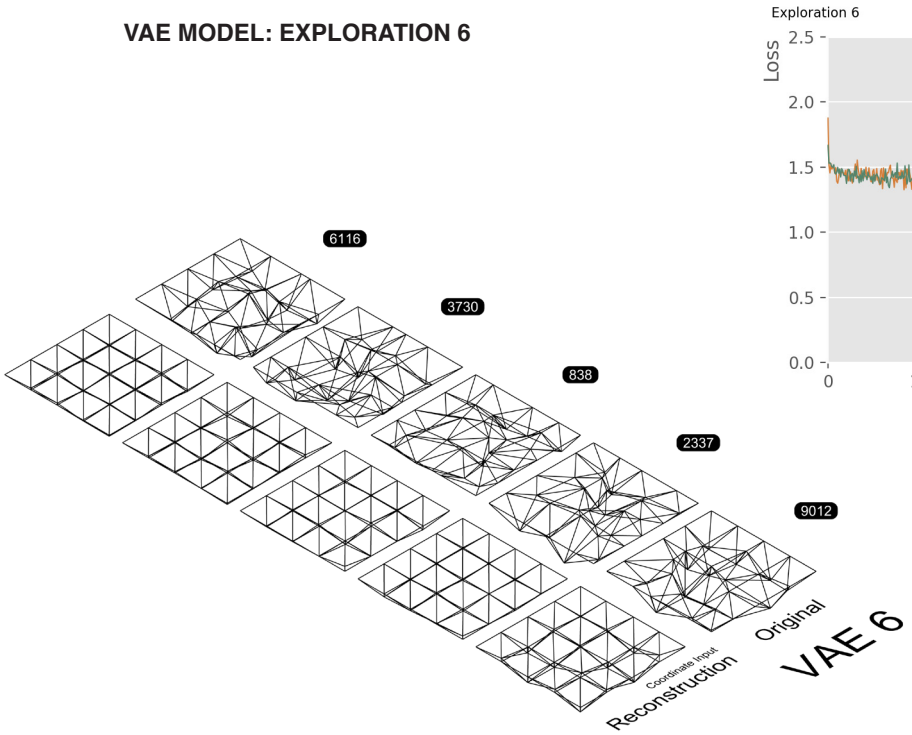
VAE MODEL: EXPLORATION 5



The attributes of this model are:

Latent Dimension	**13**
Multiplication of KL_loss	**1**
Architecture:	**‘82-41’**

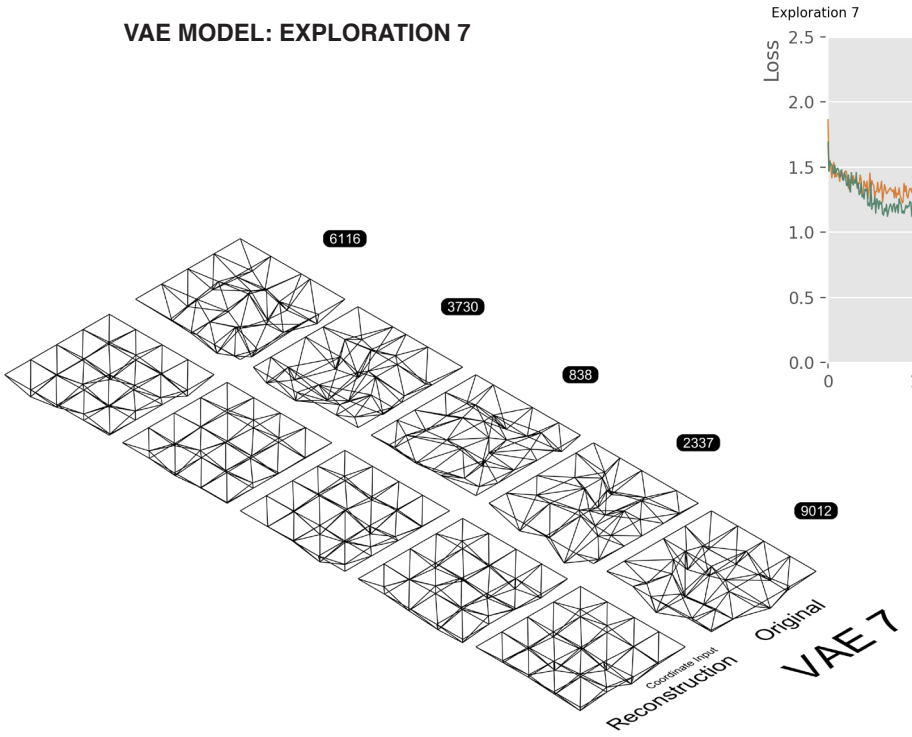
VAE MODEL: EXPLORATION 6



The attributes of this model are:

Latent Dimension	**19**
Multiplication of KL_loss	**1**
Architecture:	**‘82-41’**

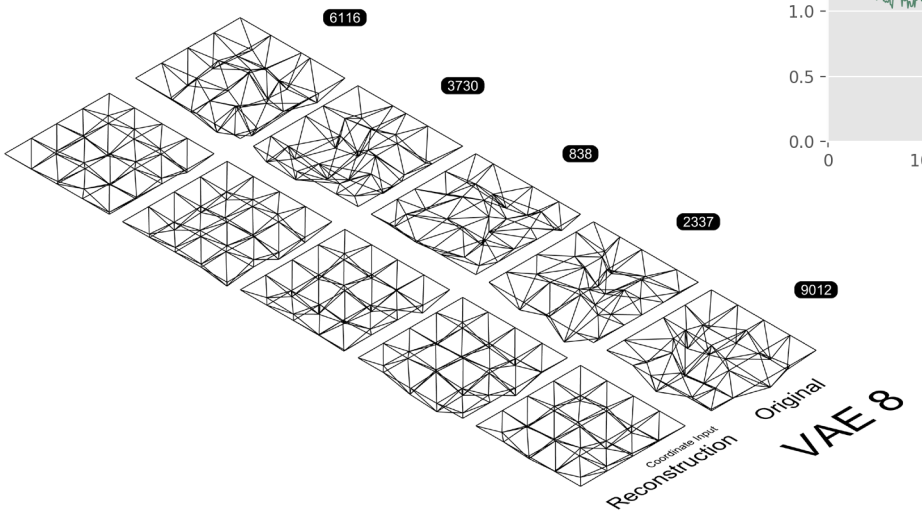
VAE MODEL: EXPLORATION 7

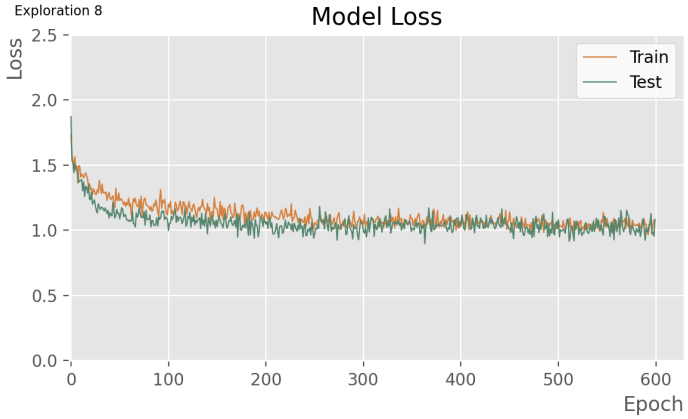


The attributes of this model are:

Latent Dimension	**30**
Multiplication of KL_loss	**1**
Architecture:	**‘82-41’**

VAE MODEL: EXPLORATION 8

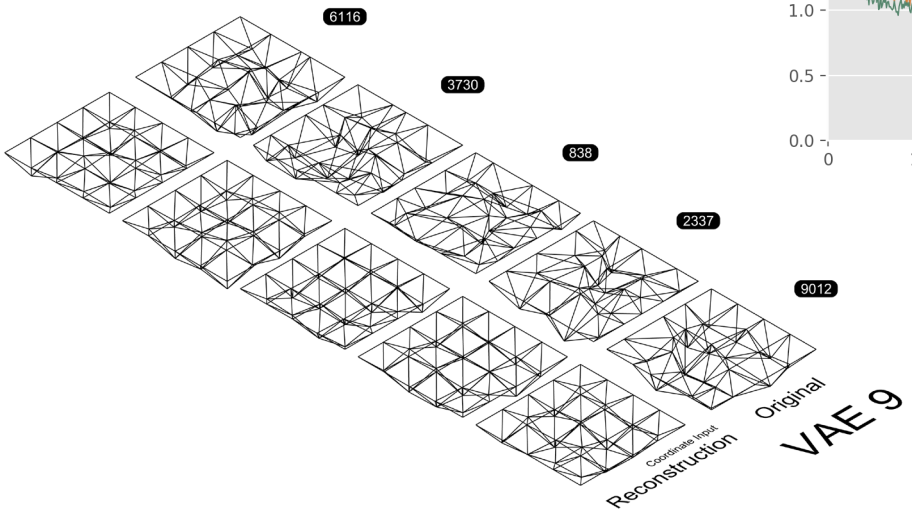


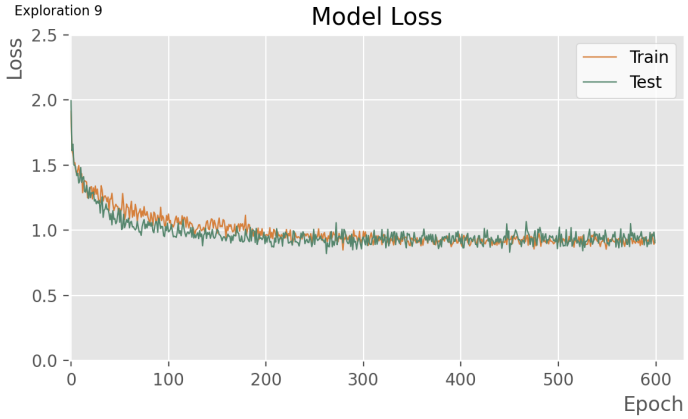


The attributes of this model are:

Latent Dimension	**40**
Multiplication of KL_loss	**1**
Architecture:	**‘82-41’**

VAE MODEL: EXPLORATION 9

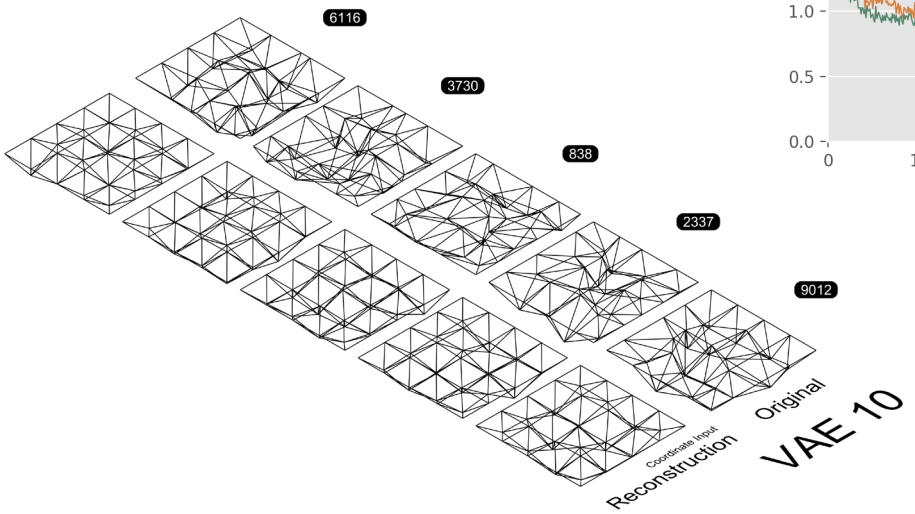


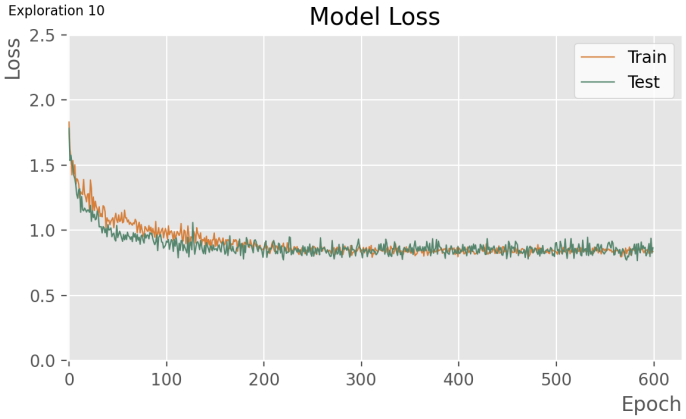


The attributes of this model are:

Latent Dimension	**50**
Multiplication of KL_loss	**1**
Architecture:	**‘82-41’**

VAE MODEL: EXPLORATION 10

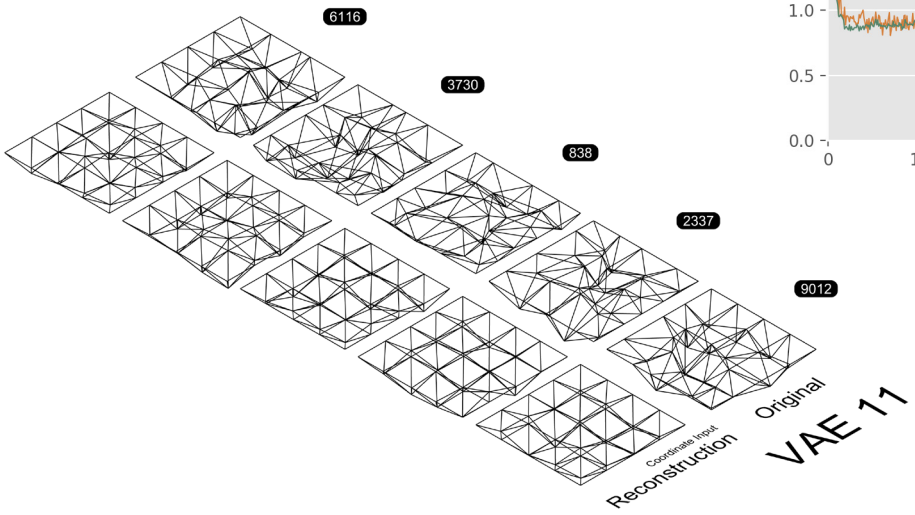




The attributes of this model are:

Latent Dimension	**60**
Multiplication of KL_loss	**1**
Architecture:	‘82-41’

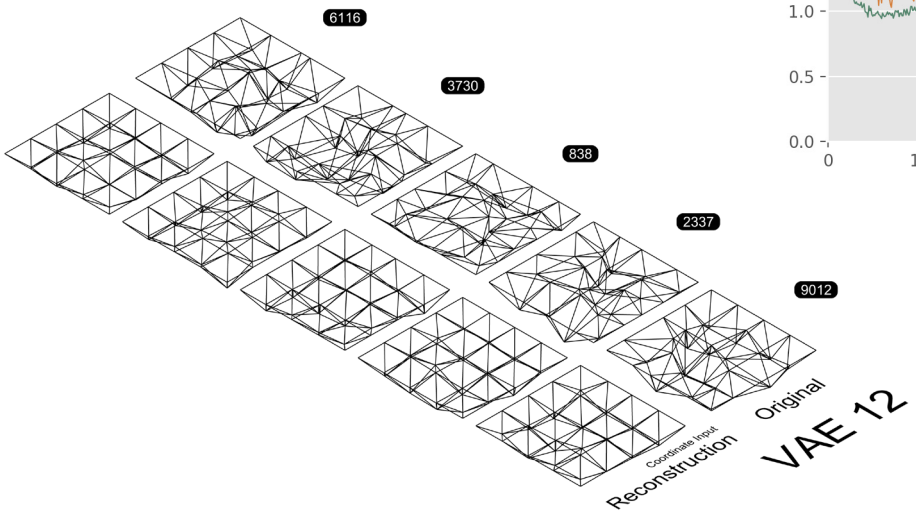
VAE MODEL: EXPLORATION 11

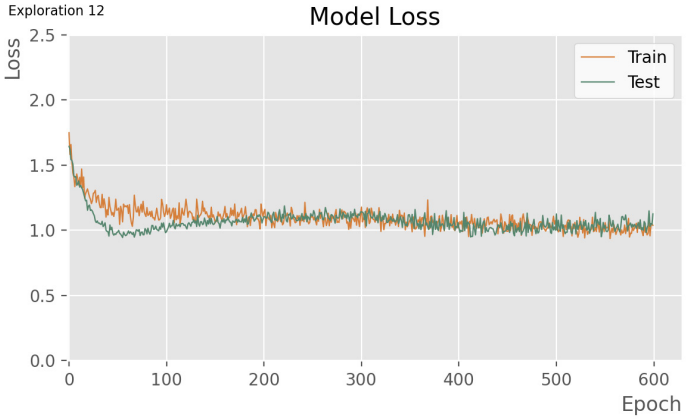


The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	**0.01**
Architecture:	‘82-41’

VAE MODEL: EXPLORATION 12

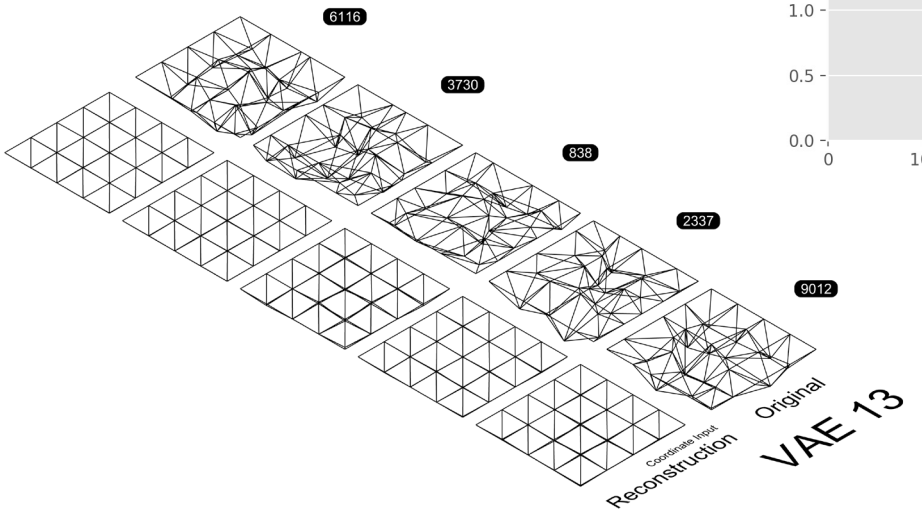


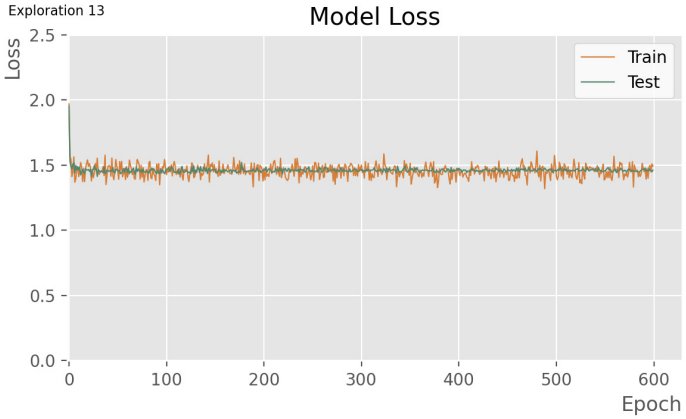


The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	**0.1**
Architecture:	'82-41'

VAE MODEL: EXPLORATION 13

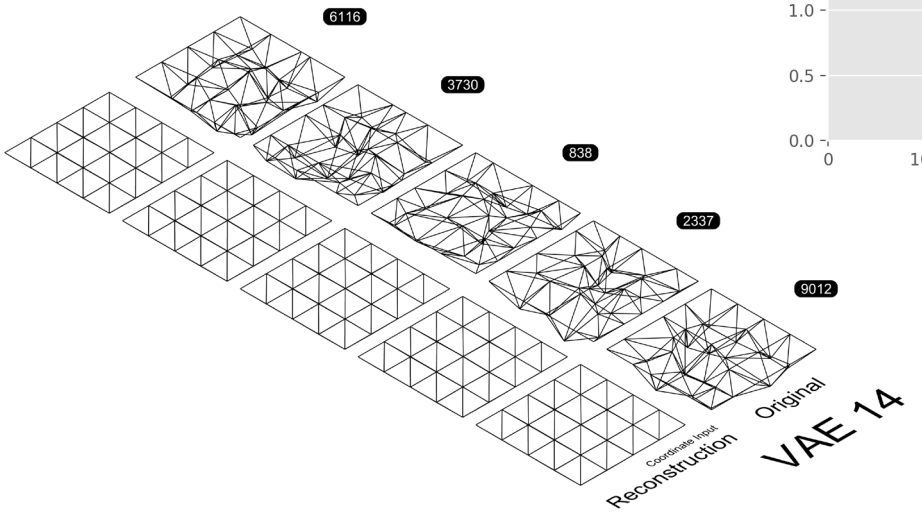


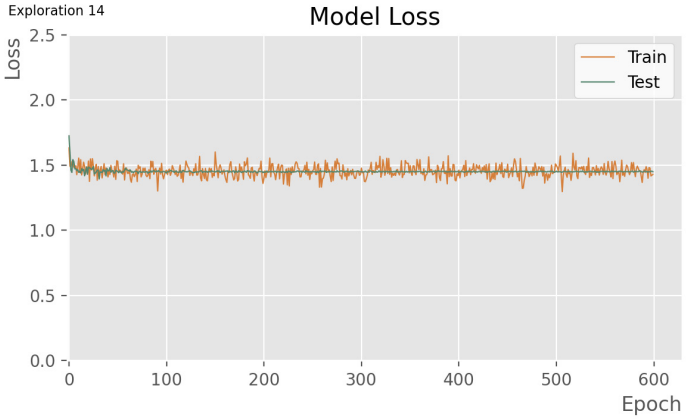


The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	**0.5**
Architecture:	'82-41'

VAE MODEL: EXPLORATION 14

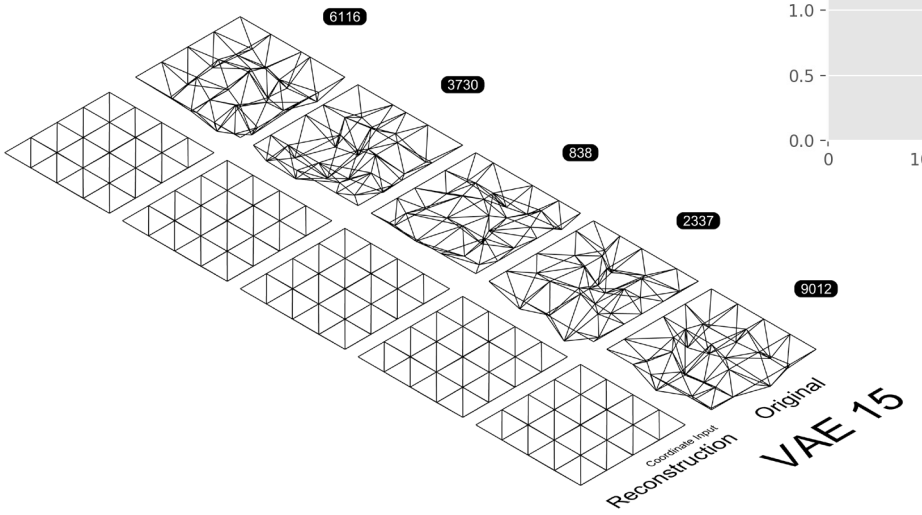


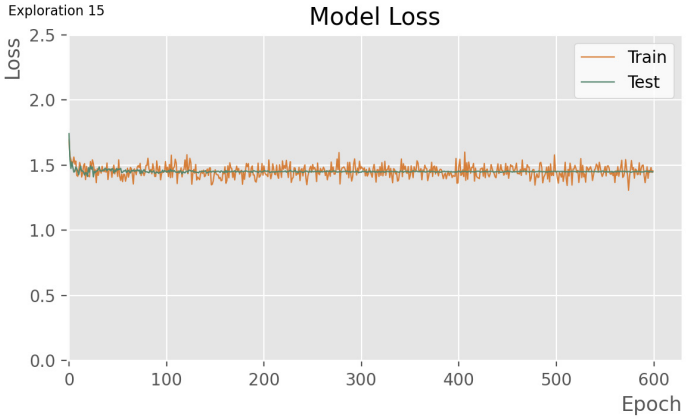


The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	**5**
Architecture:	'82-41'

VAE MODEL: EXPLORATION 15

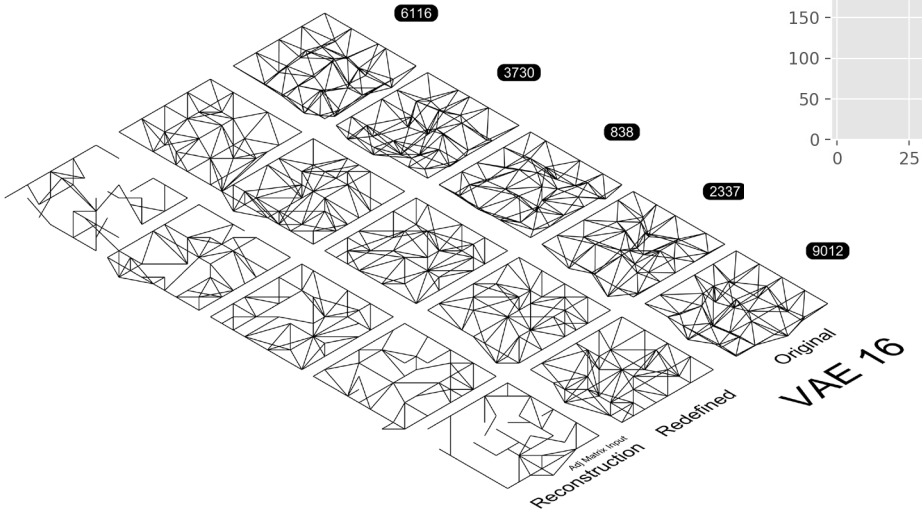


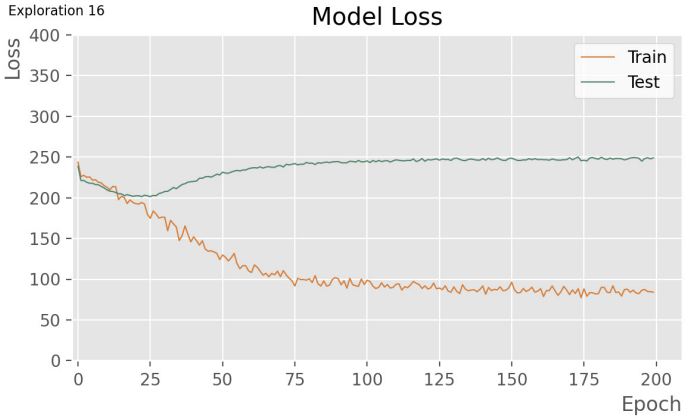


The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	**10**
Architecture:	'82-41'

VAE MODEL: EXPLORATION 16

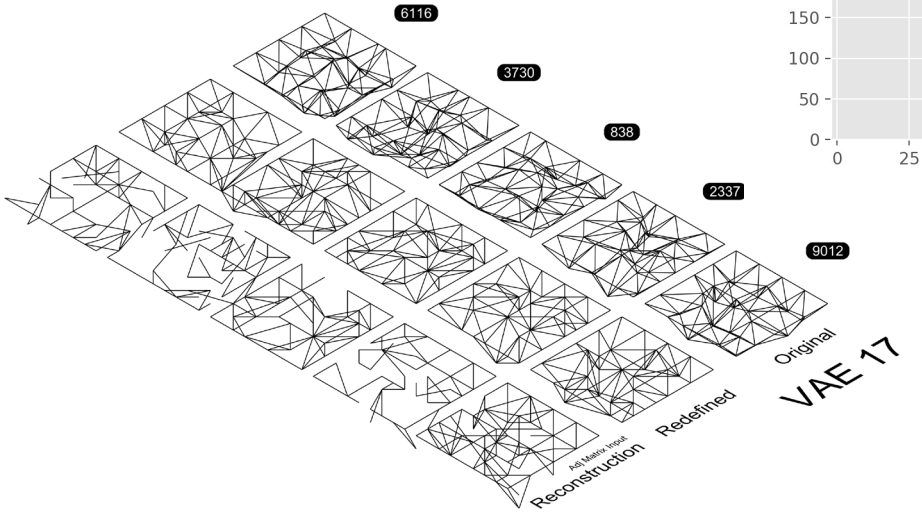




The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	1
Architecture:	'972-243'

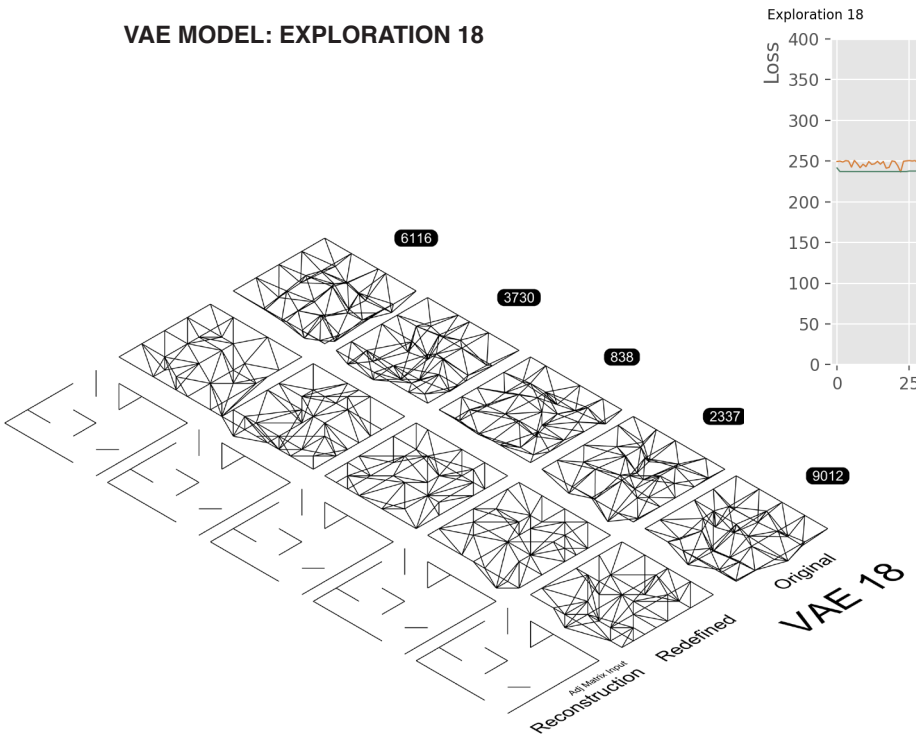
VAE MODEL: EXPLORATION 17



The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	1
Architecture:	**'1944-972-243'**

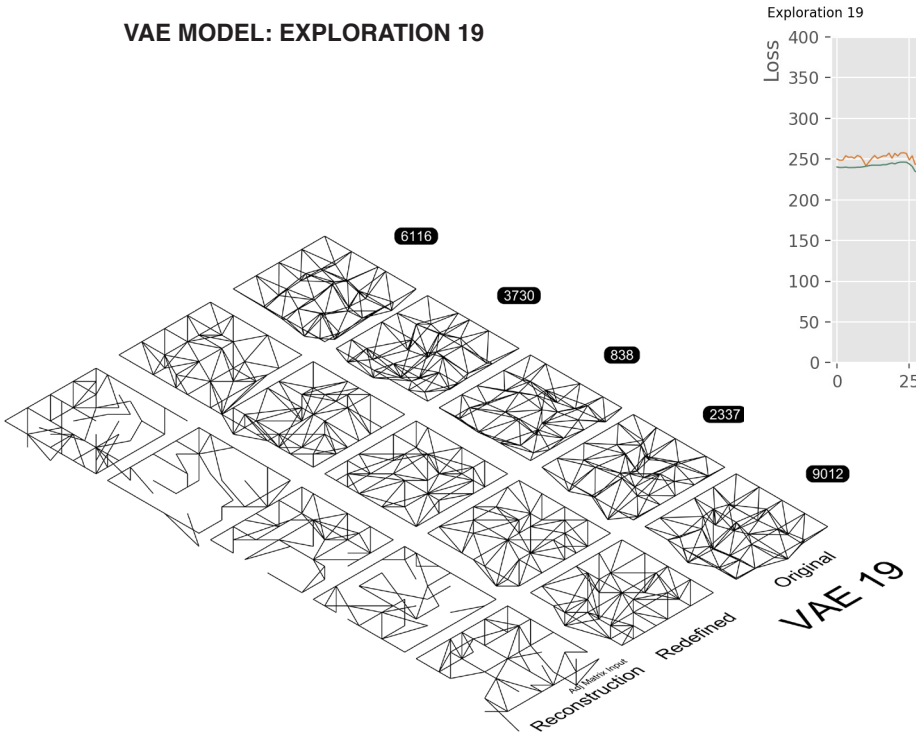
VAE MODEL: EXPLORATION 18



The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	1
Architecture:	**'1944-972-486-24**

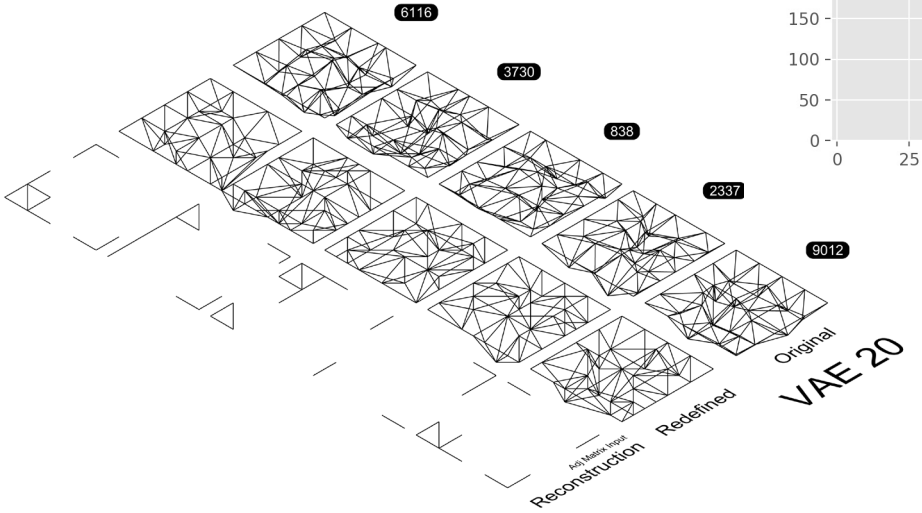
VAE MODEL: EXPLORATION 19



The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	1
Architecture:	**'3888-1944-243'**

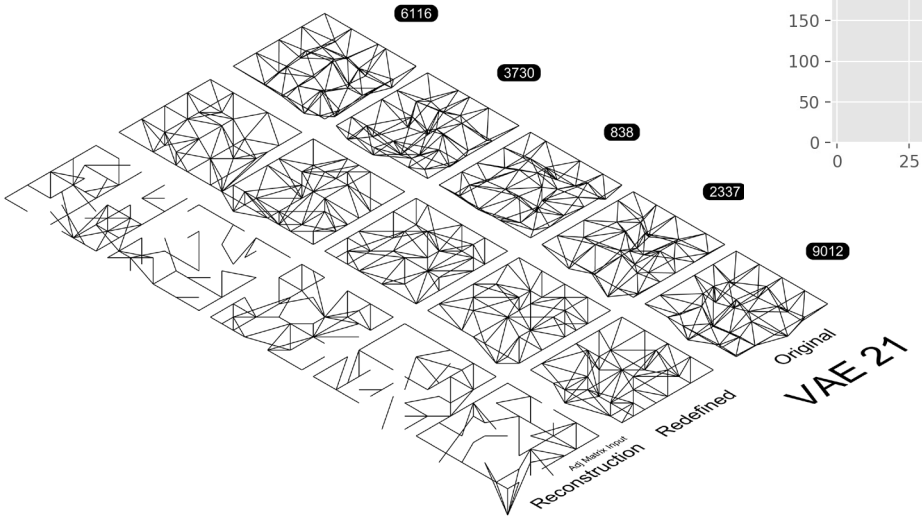
VAE MODEL: EXPLORATION 20

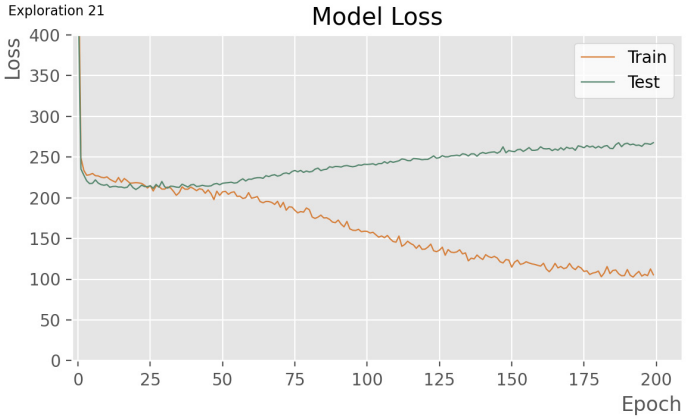


The attributes of this model are:

Latent Dimension	**2**
Multiplication of KL_loss	**1**
Architecture:	**‘972-243’**

VAE MODEL: EXPLORATION 21

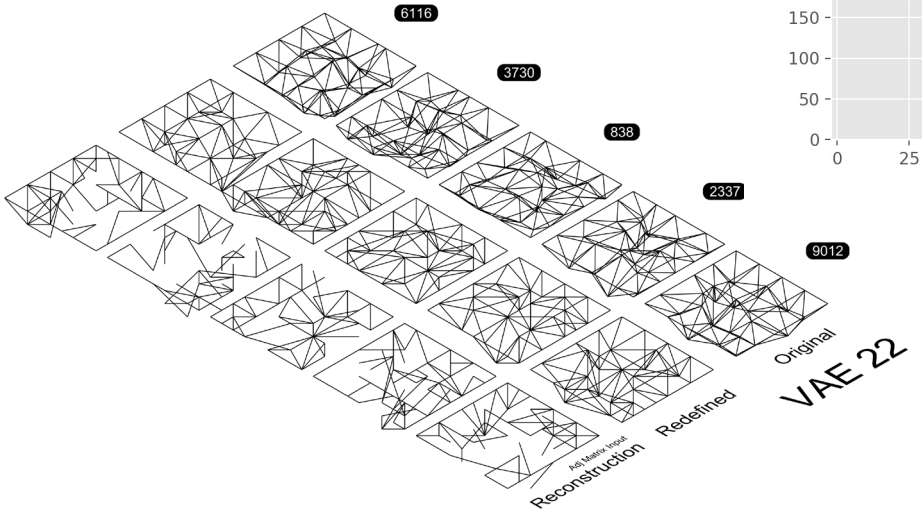


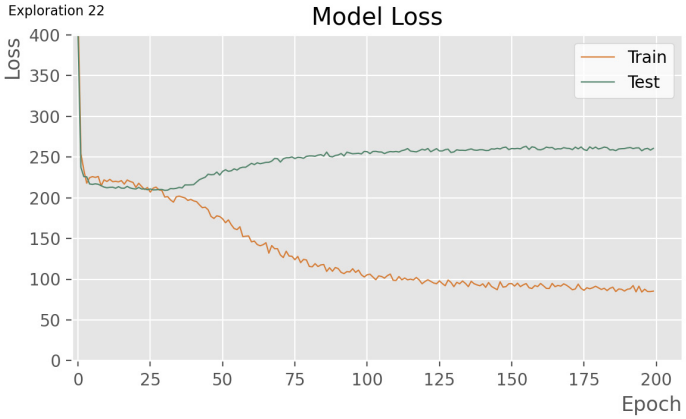


The attributes of this model are:

Latent Dimension	**7**
Multiplication of KL_loss	**1**
Architecture:	**‘972-243’**

VAE MODEL: EXPLORATION 22

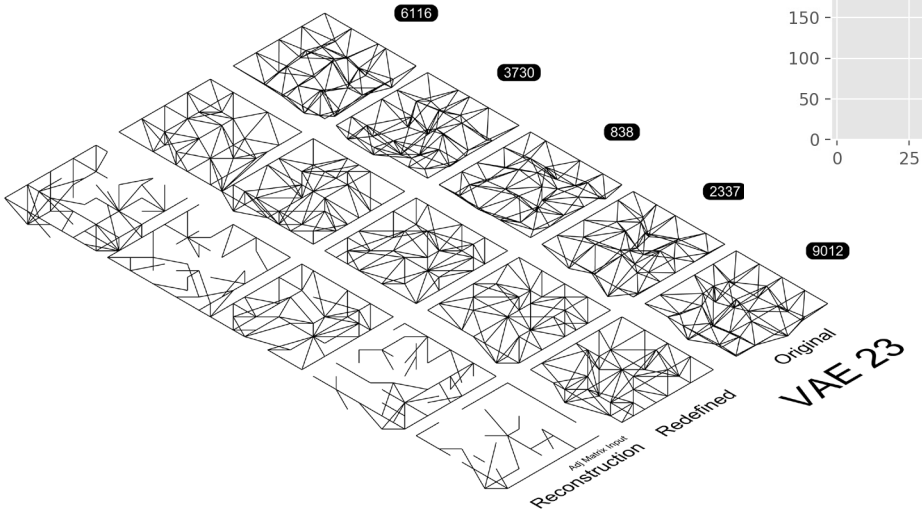


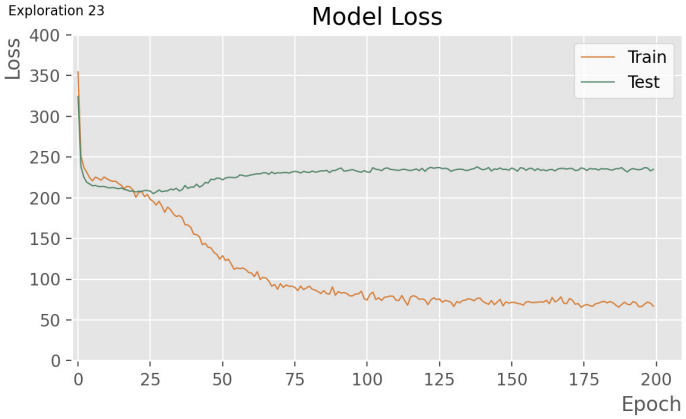


The attributes of this model are:

Latent Dimension	**13**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 23

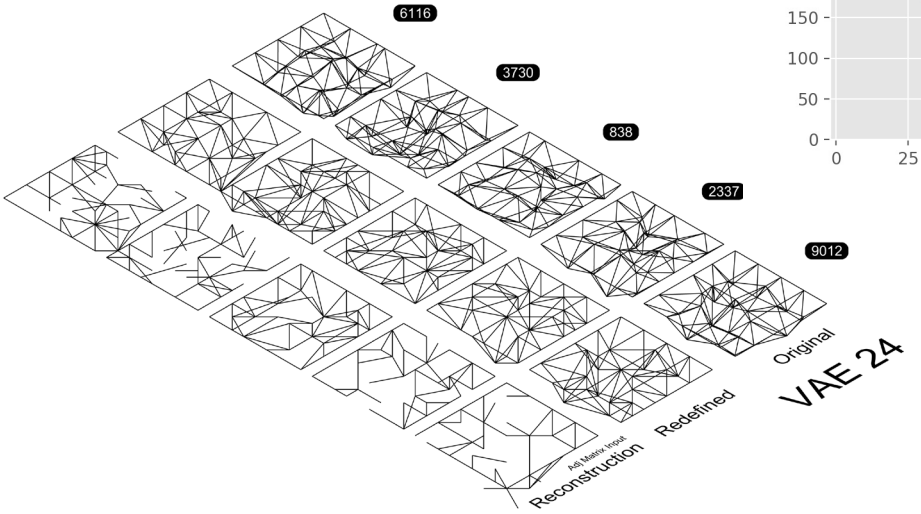


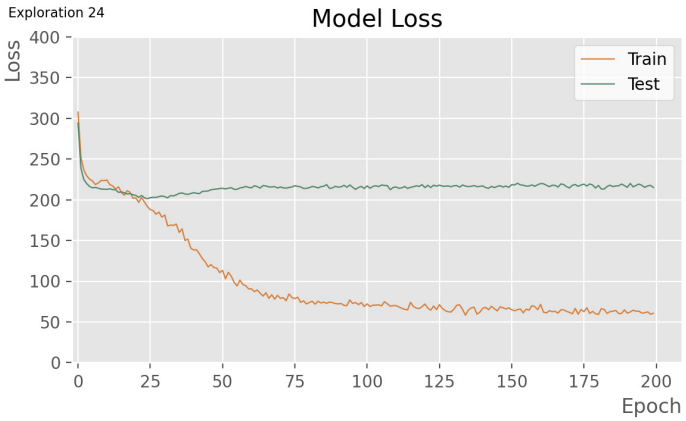


The attributes of this model are:

Latent Dimension	**30**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 24

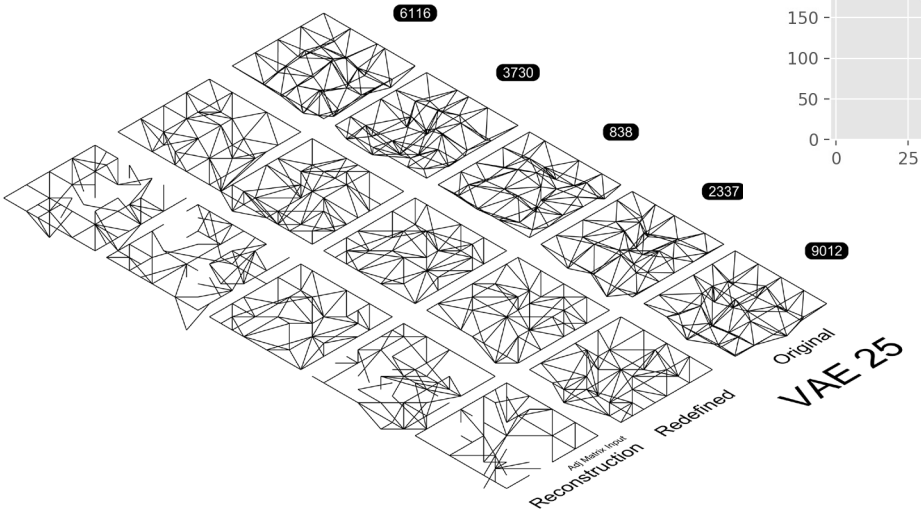




The attributes of this model are:

Latent Dimension	**40**
Multiplication of KL_loss	**1**
Architecture:	**‘972-243’**

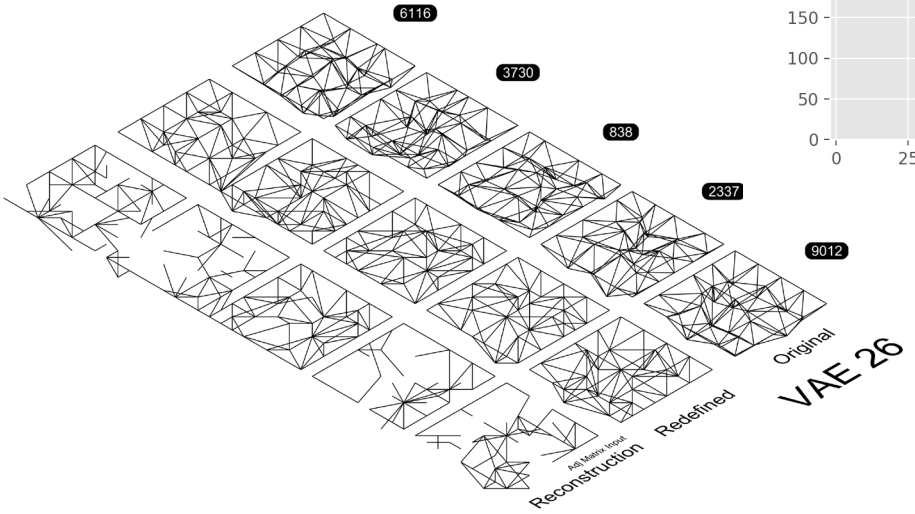
VAE MODEL: EXPLORATION 25



The attributes of this model are:

Latent Dimension	**50**
Multiplication of KL_loss	**1**
Architecture:	**‘972-243’**

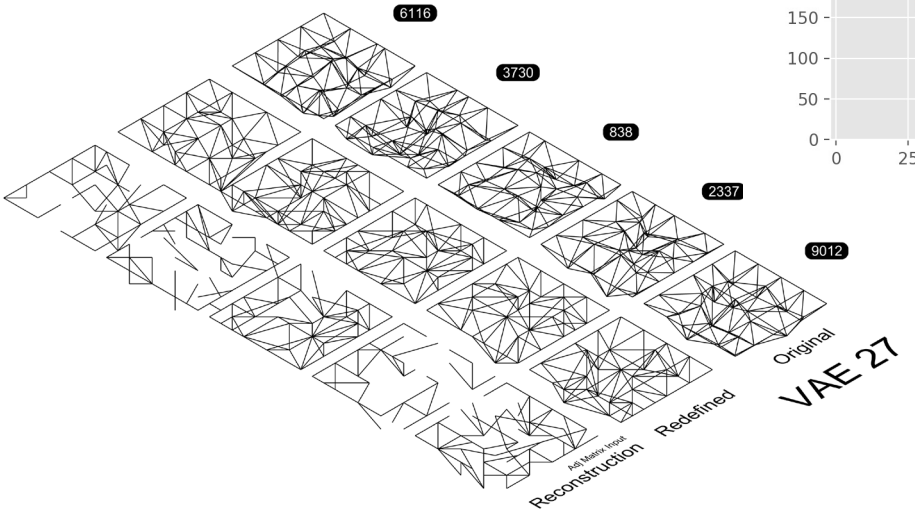
VAE MODEL: EXPLORATION 26



The attributes of this model are:

Latent Dimension	**60**
Multiplication of KL_loss	**1**
Architecture:	‘972-243’

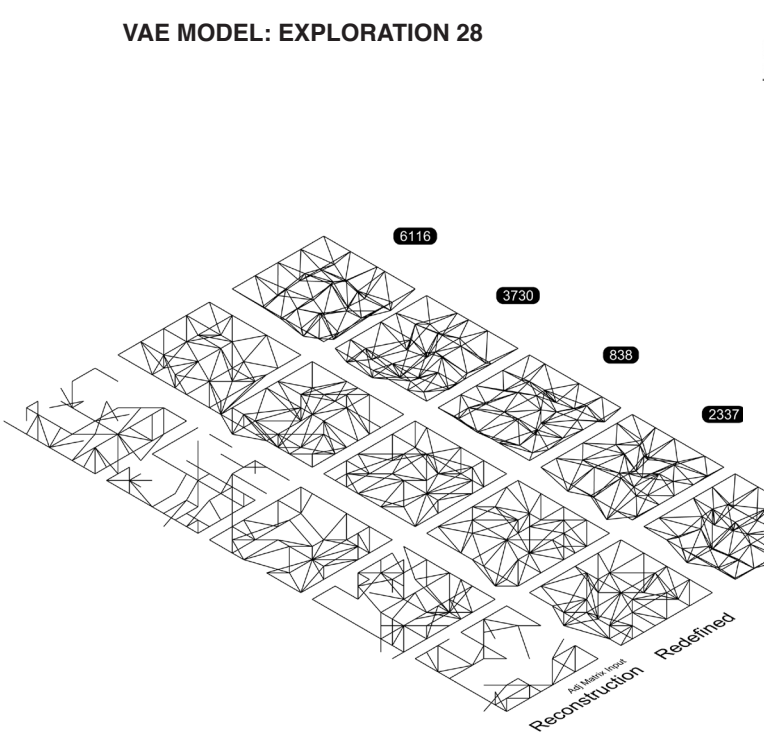
VAE MODEL: EXPLORATION 27



The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**0.01**
Architecture:	‘972-243’

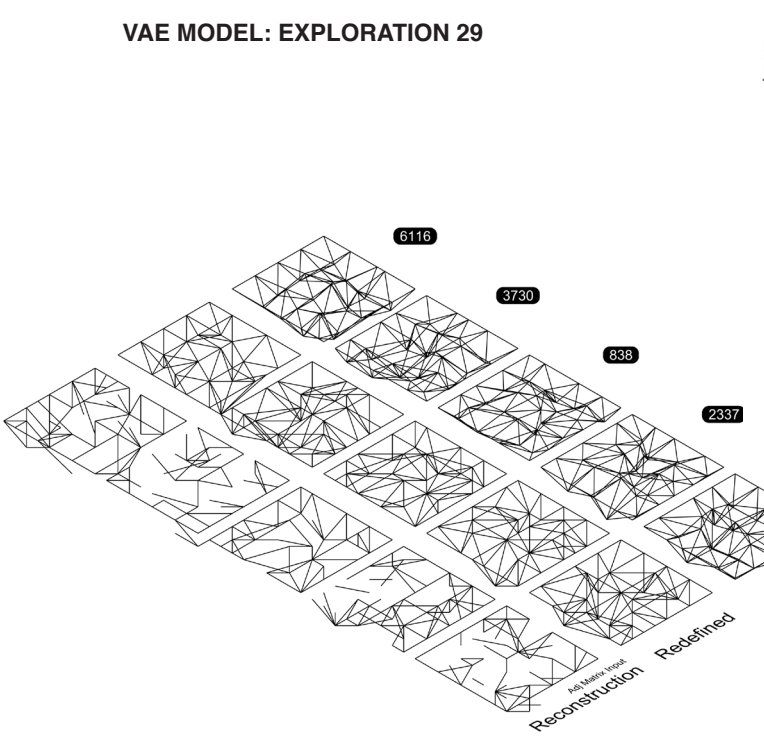
VAE MODEL: EXPLORATION 28



The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**0.1**
Architecture:	'972-243'

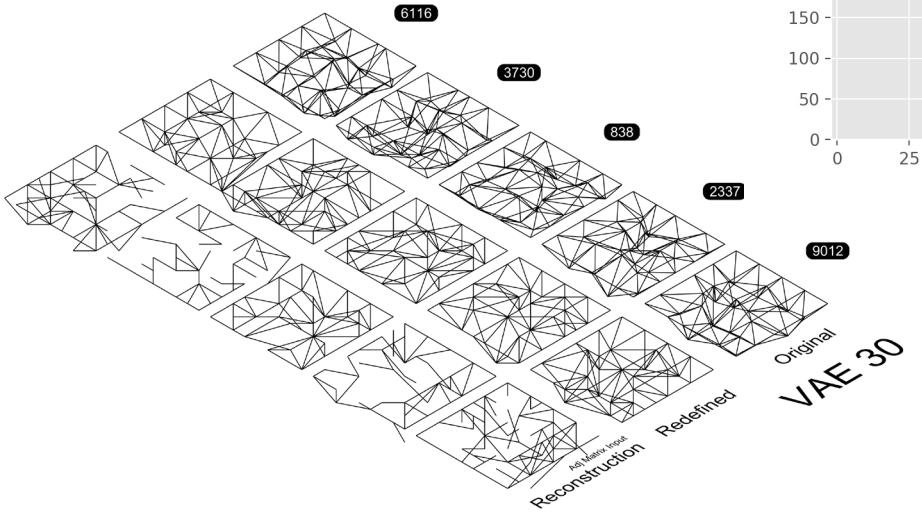
VAE MODEL: EXPLORATION 29



The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**0.5**
Architecture:	'972-243'

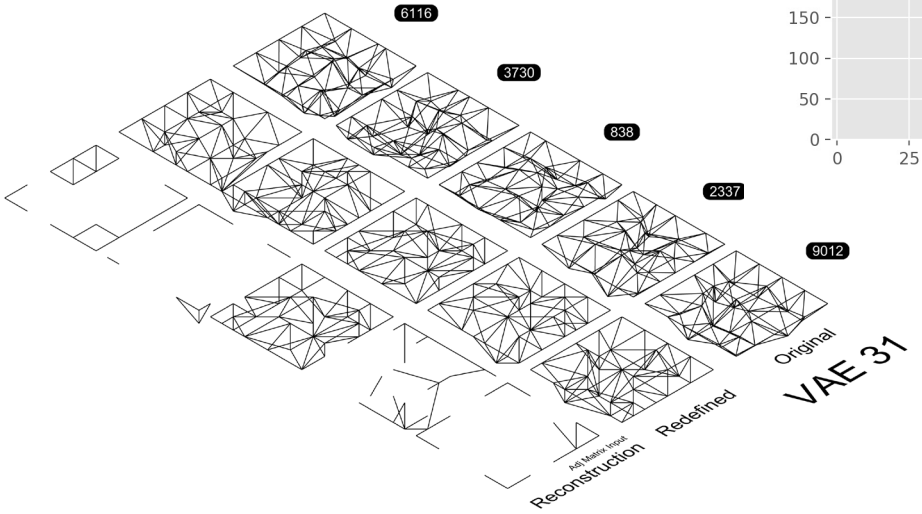
VAE MODEL: EXPLORATION 30



The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**5**
Architecture:	'972-243'

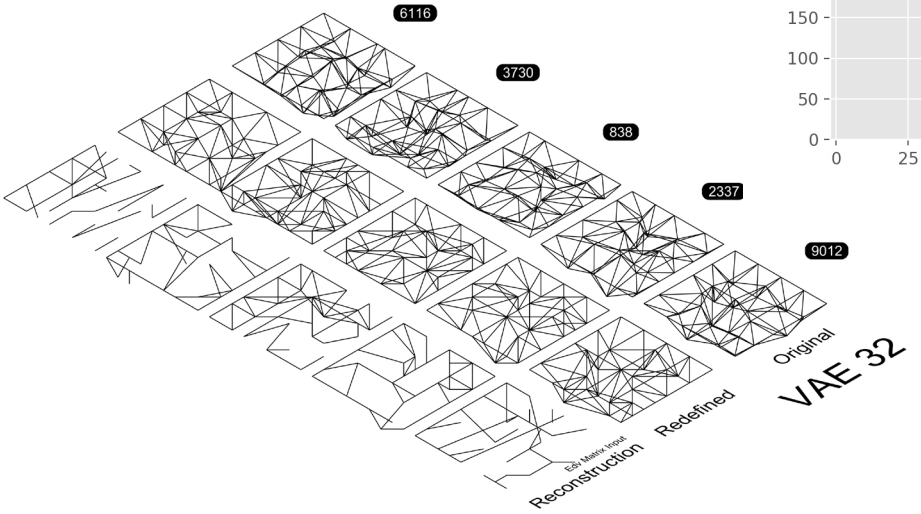
VAE MODEL: EXPLORATION 31

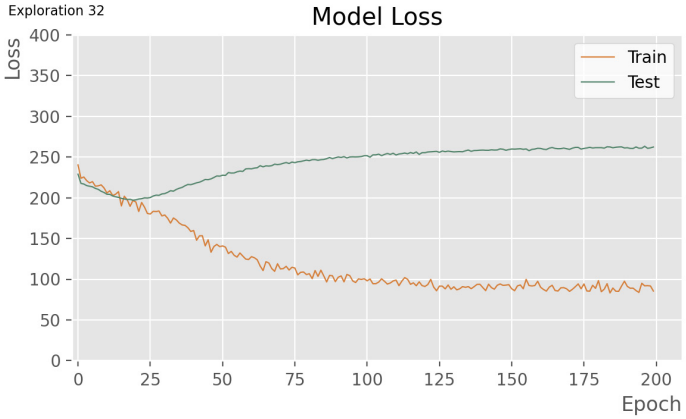


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**10**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 32

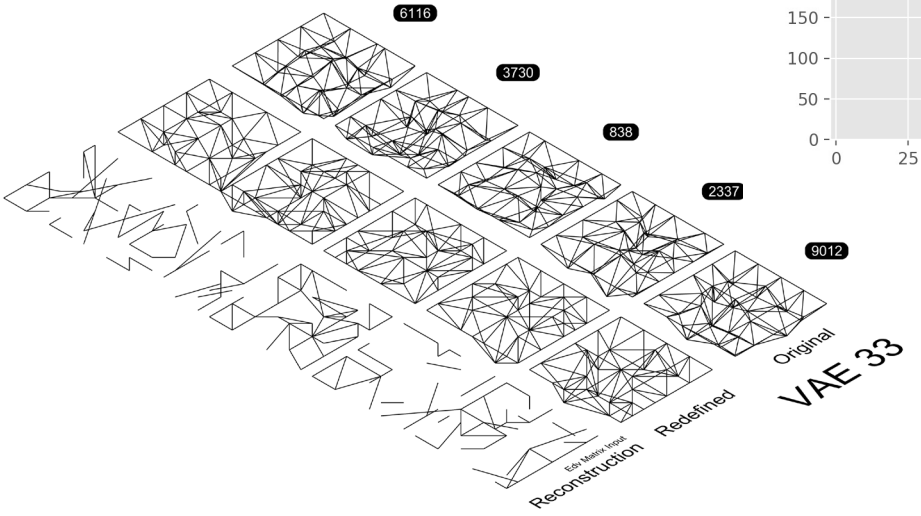


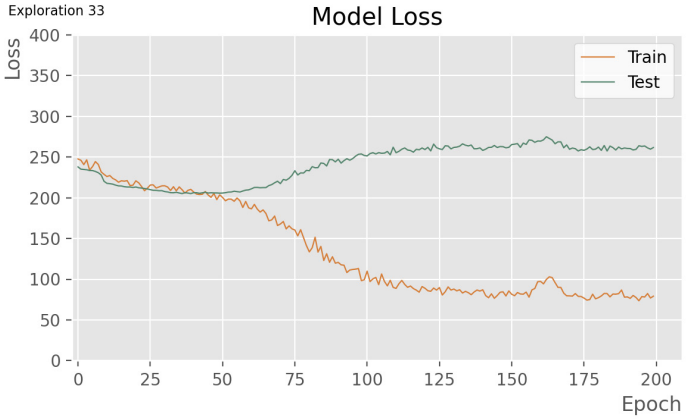


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 33

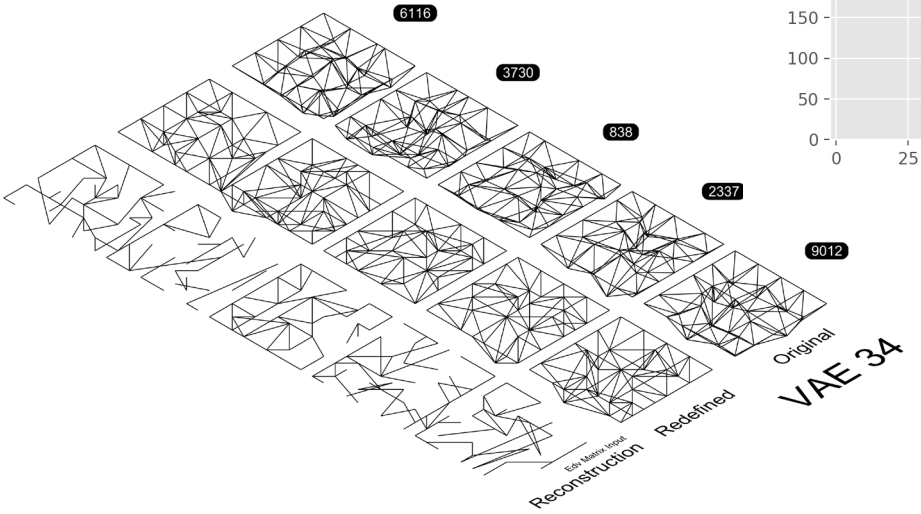


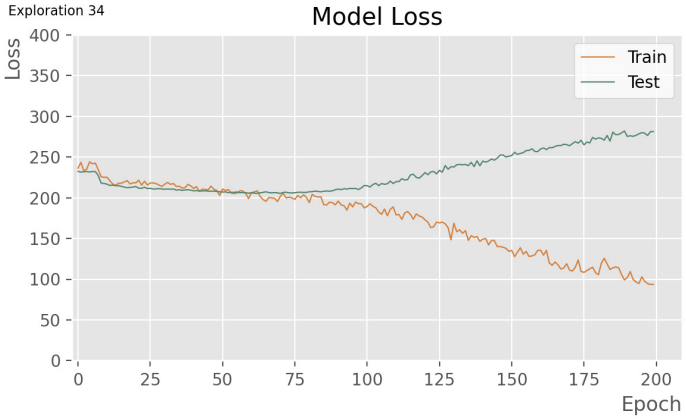


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	**'1944-972-243'**

VAE MODEL: EXPLORATION 34

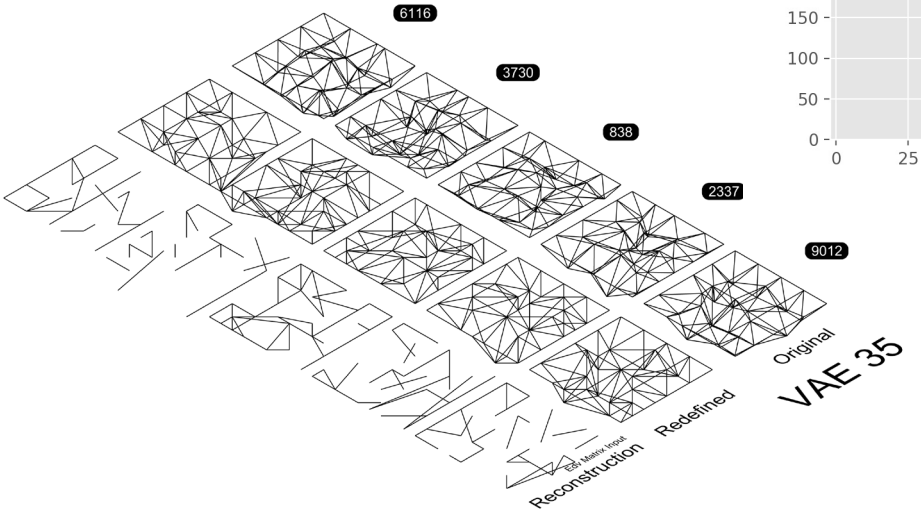


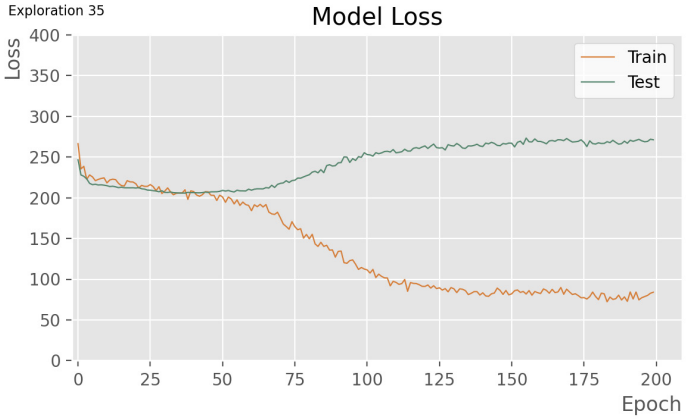


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	**'1944-972-486-24**

VAE MODEL: EXPLORATION 35

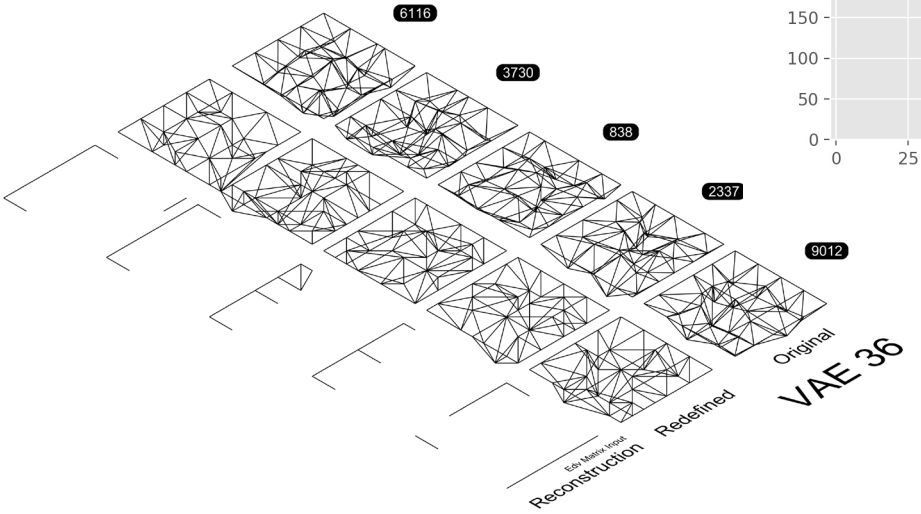




The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	**'3888-1944-243'**

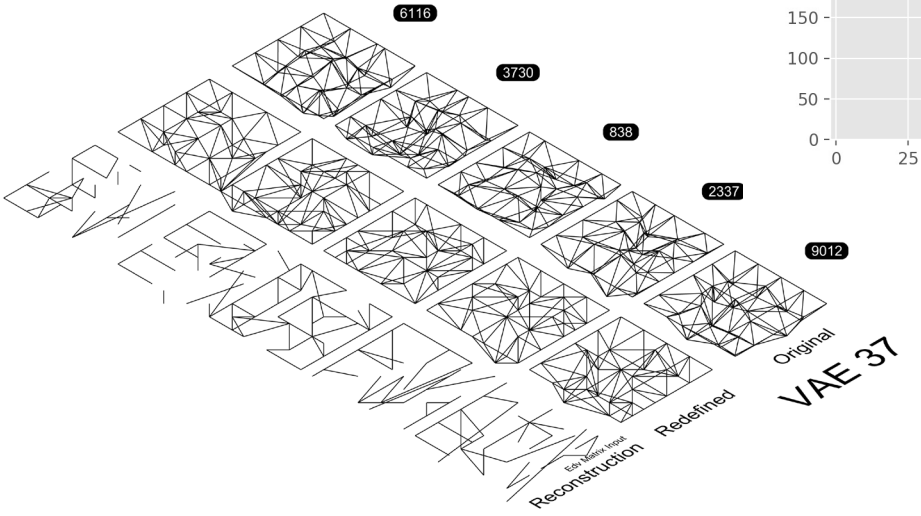
VAE MODEL: EXPLORATION 36

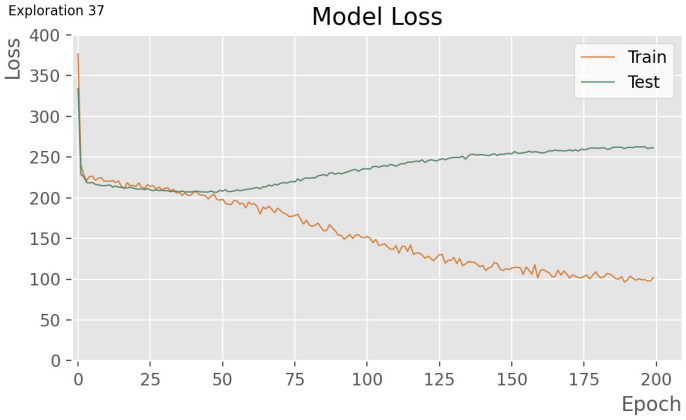


The attributes of this model are:

Latent Dimension	**2**
Multiplication of KL_loss	**1**
Architecture:	**‘972-243’**

VAE MODEL: EXPLORATION 37

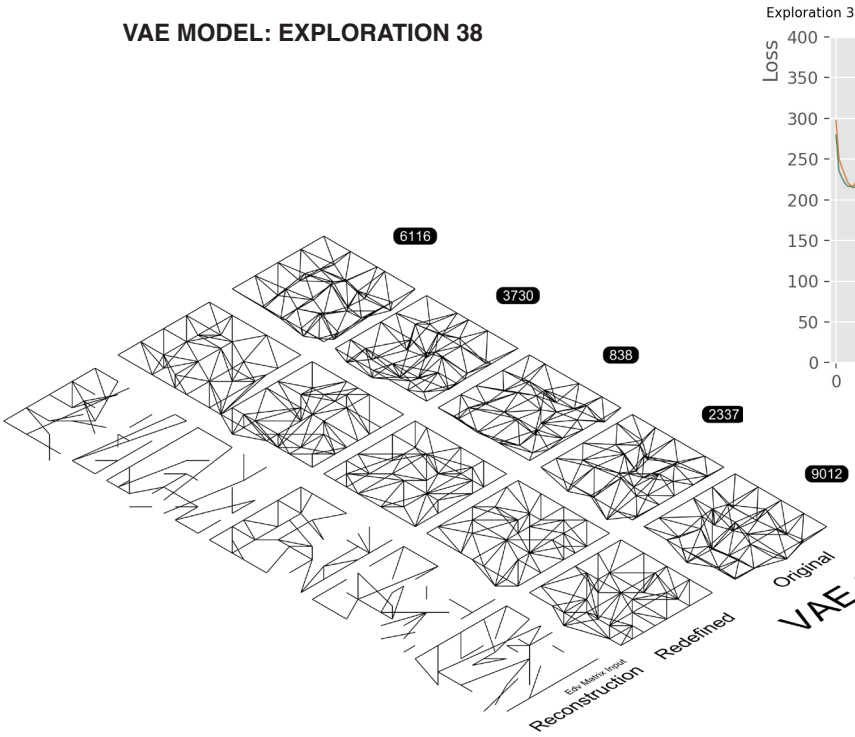




The attributes of this model are:

Latent Dimension	**7**
Multiplication of KL_loss	**1**
Architecture:	**‘972-243’**

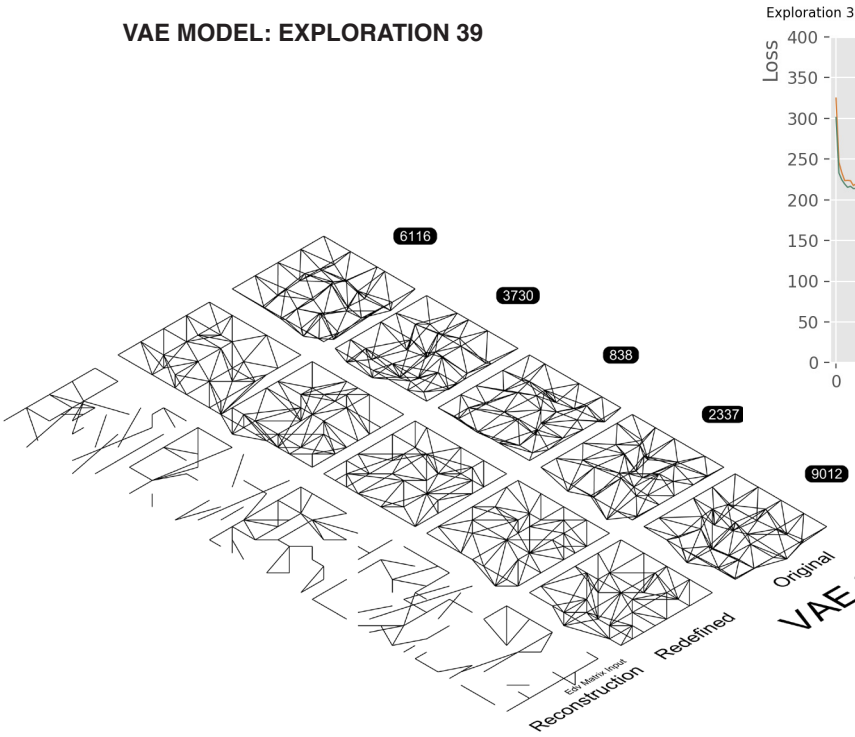
VAE MODEL: EXPLORATION 38



The attributes of this model are:

Latent Dimension	**19**
Multiplication of KL_loss	1
Architecture:	'972-243'

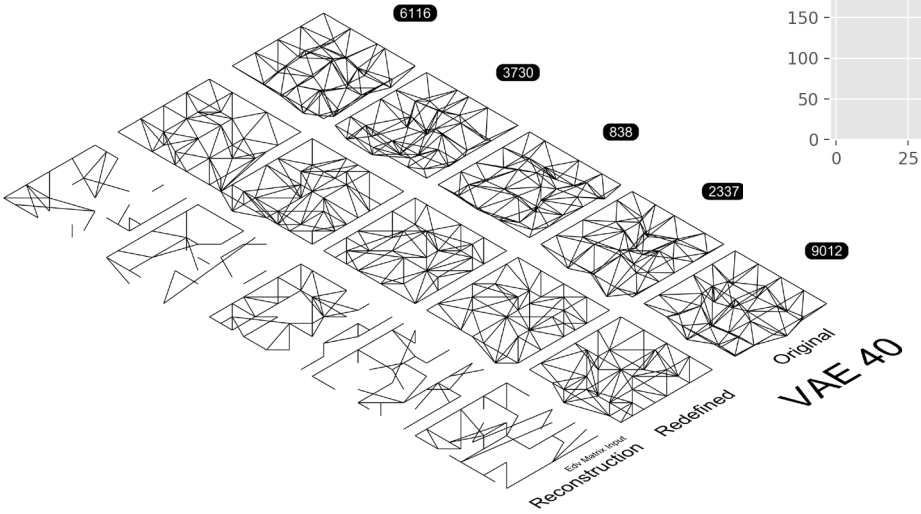
VAE MODEL: EXPLORATION 39



The attributes of this model are:

Latent Dimension	**30**
Multiplication of KL_loss	1
Architecture:	'972-243'

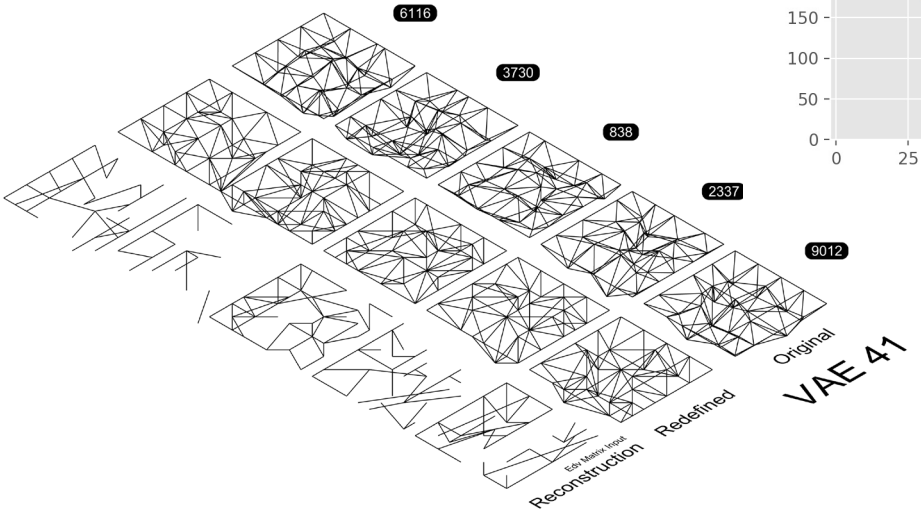
VAE MODEL: EXPLORATION 40

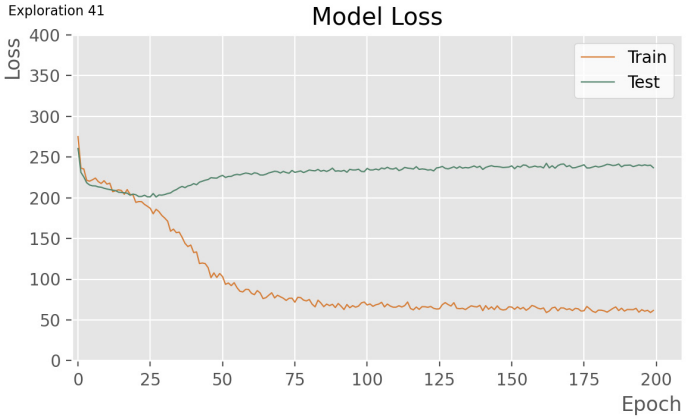


The attributes of this model are:

Latent Dimension	**40**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 41

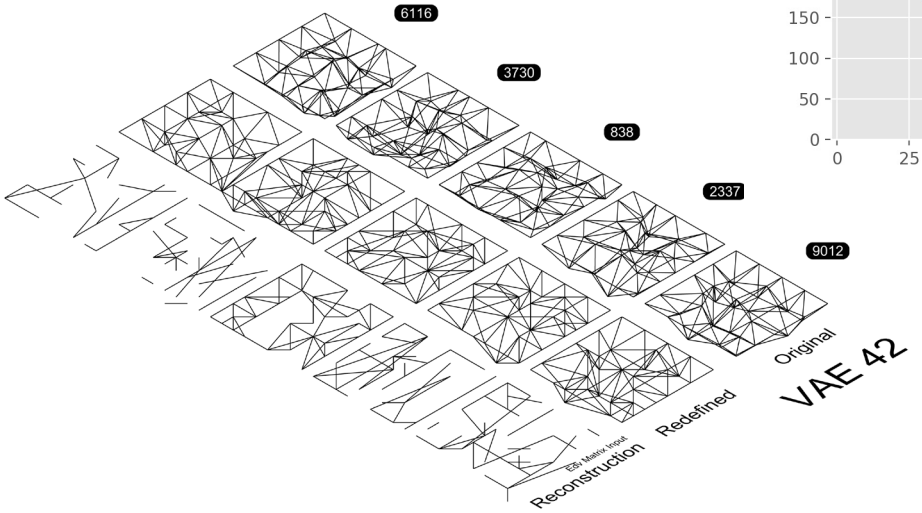




The attributes of this model are:

Latent Dimension	**50**
Multiplication of KL_loss	1
Architecture:	'972-243'

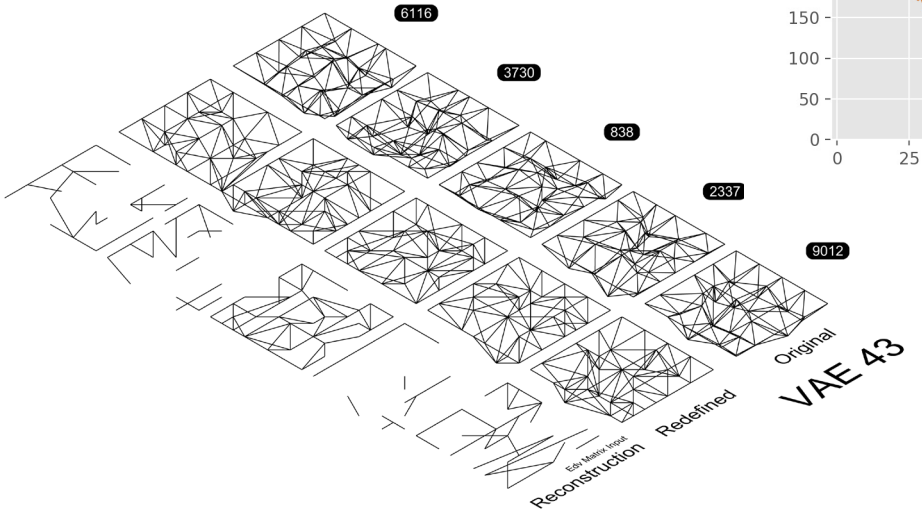
VAE MODEL: EXPLORATION 42



The attributes of this model are:

Latent Dimension	**60**
Multiplication of KL_loss	1
Architecture:	'972-243'

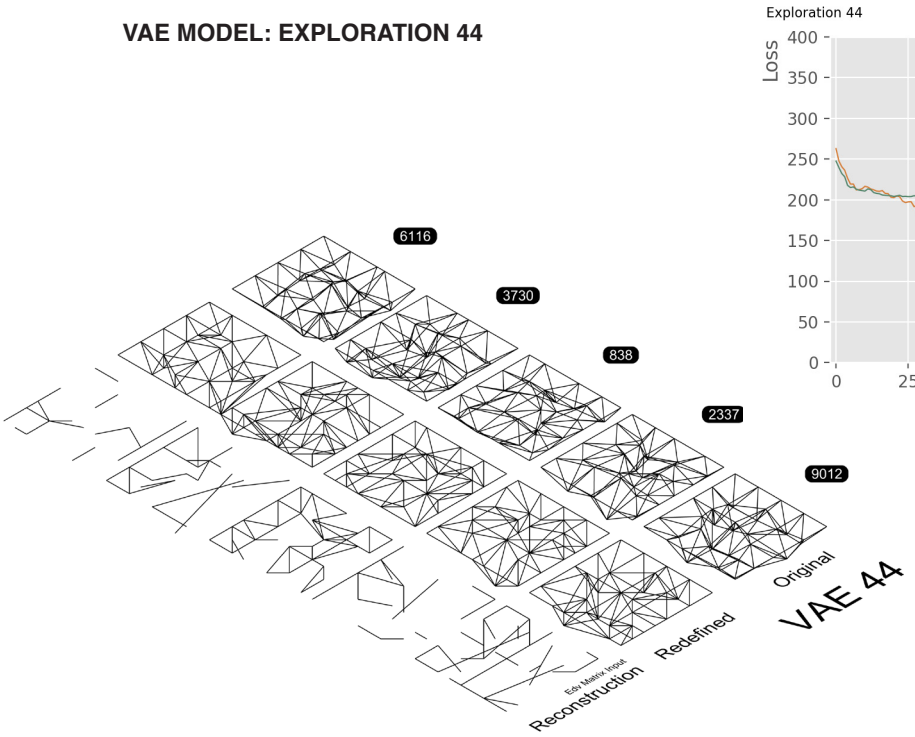
VAE MODEL: EXPLORATION 43



The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.01**
Architecture:	'972-243'

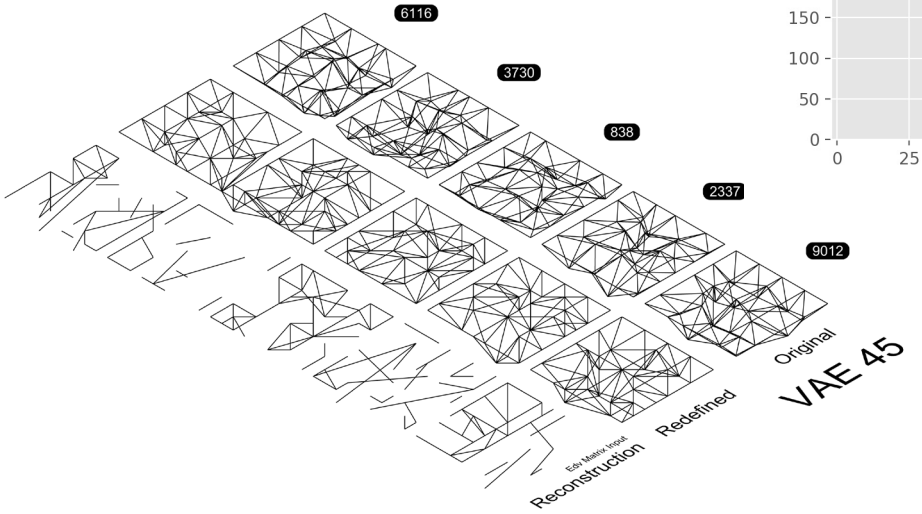
VAE MODEL: EXPLORATION 44



The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.1**
Architecture:	'972-243'

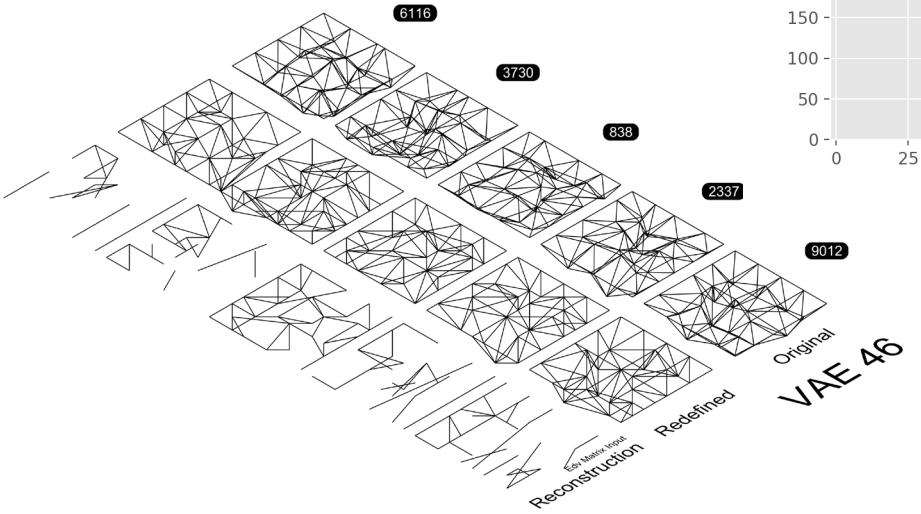
VAE MODEL: EXPLORATION 45

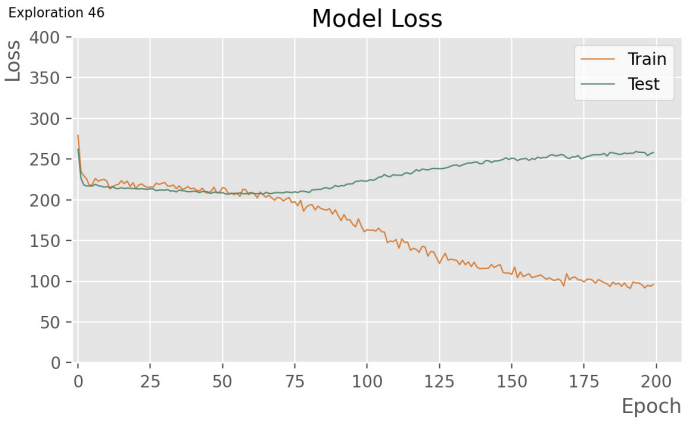


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.5**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 46

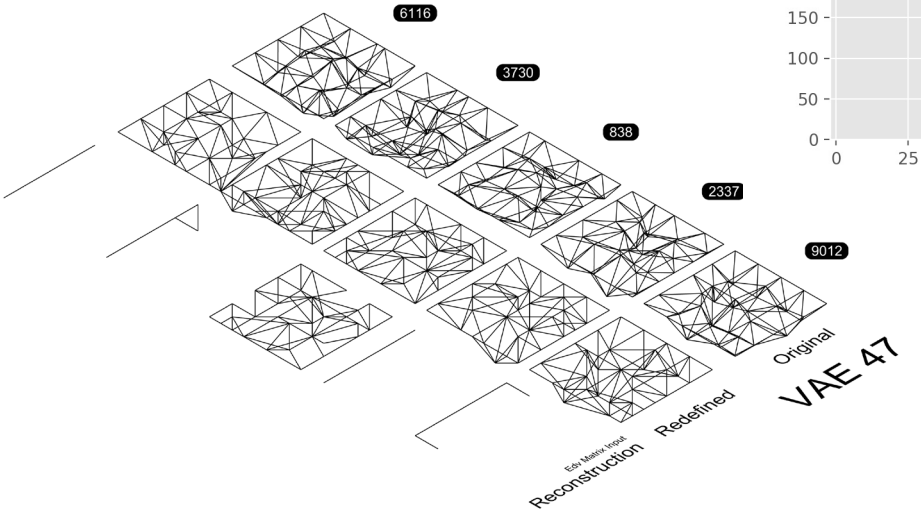




The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	5
Architecture:	'972-243'

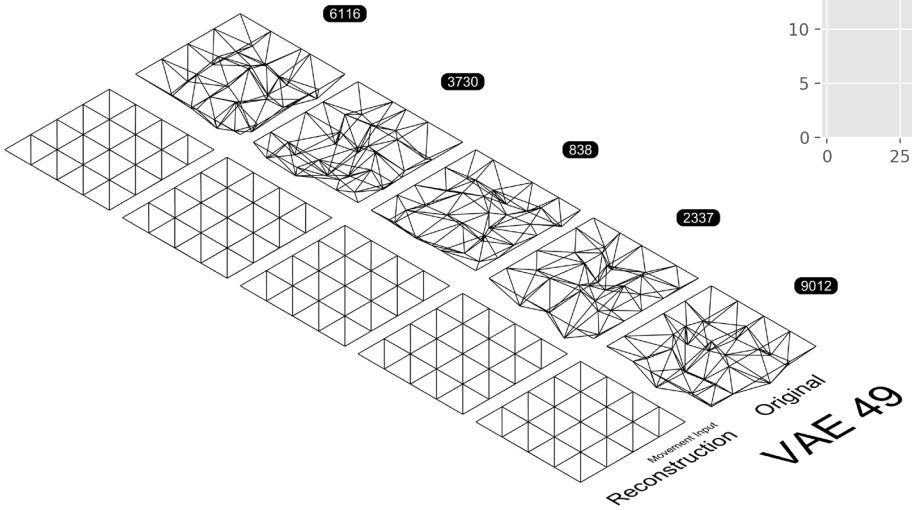
VAE MODEL: EXPLORATION 47



The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	10
Architecture:	'972-243'

VAE MODEL: EXPLORATION 49

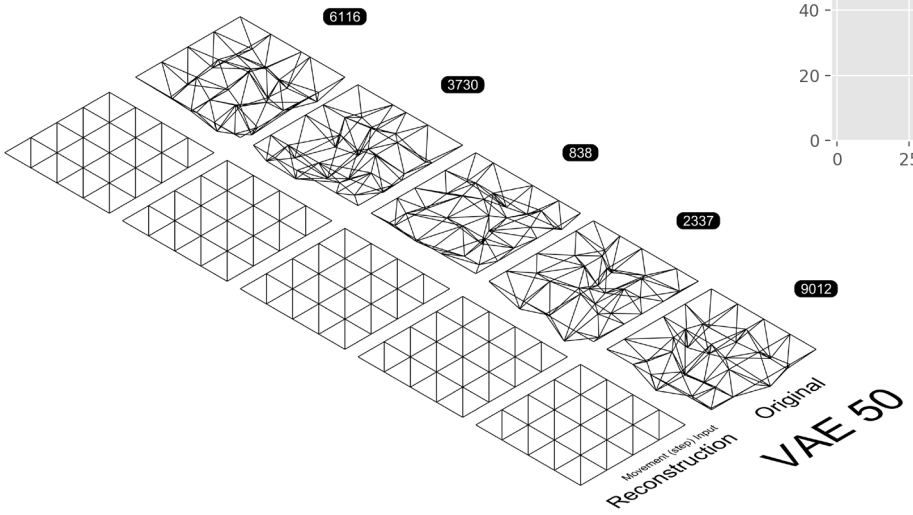


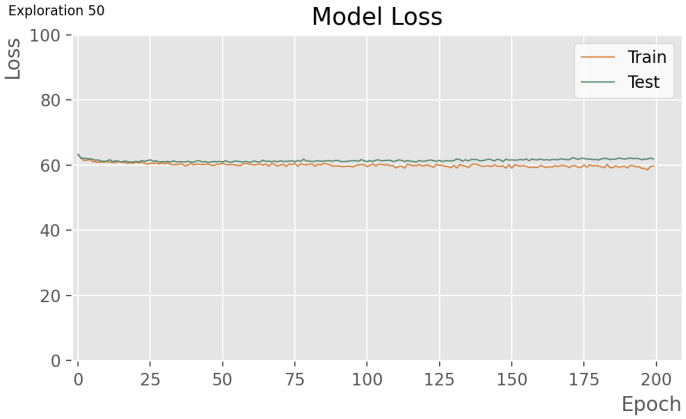
The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	10
Architecture:	'82-41'

*with "movement" input

VAE MODEL: EXPLORATION 50



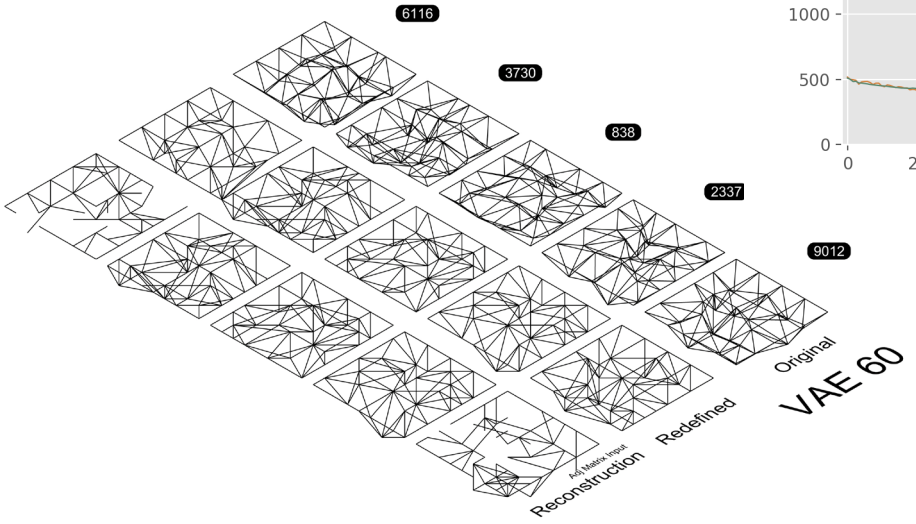


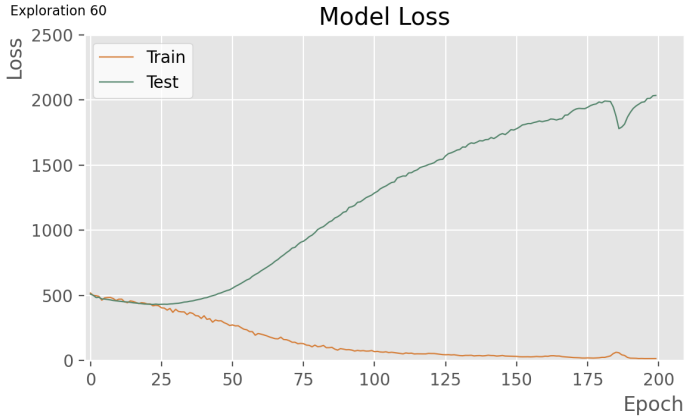
The attributes of this model are:

Latent Dimension	7
Multiplication of KL_loss	10
Architecture:	'123-41'

*with "step movement" input

VAE MODEL: EXPLORATION 60

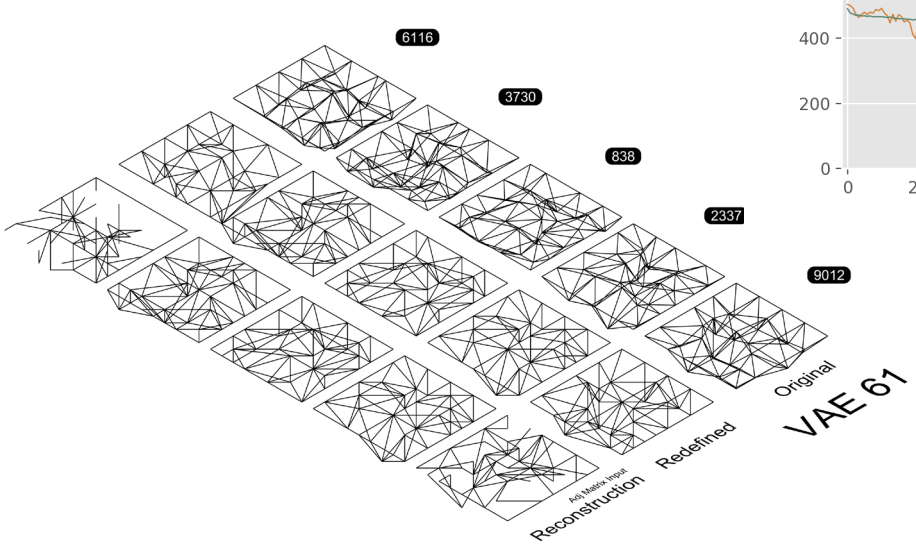


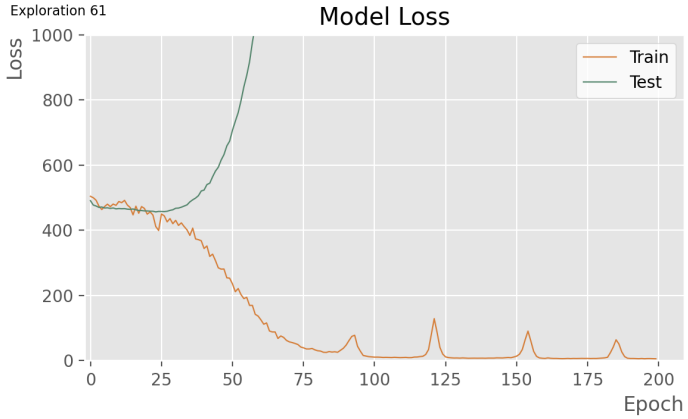


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 61

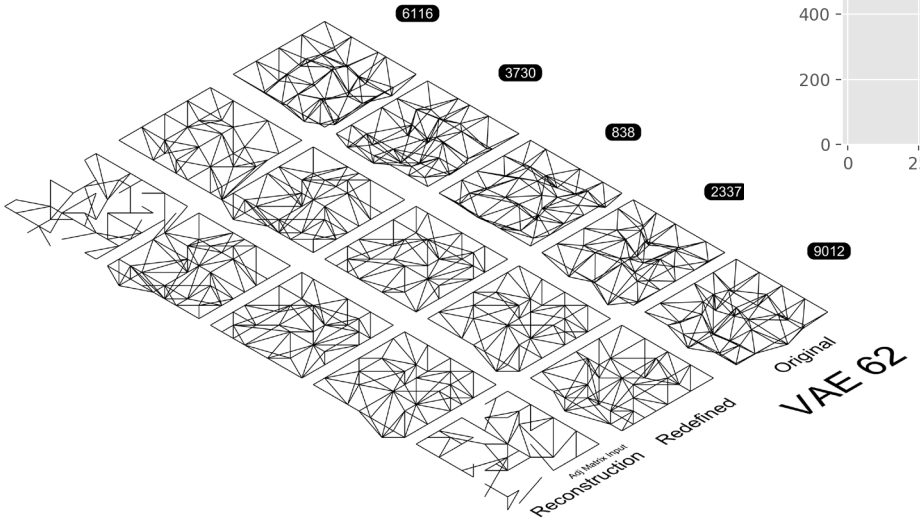




The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	**'3267-1089-121'**

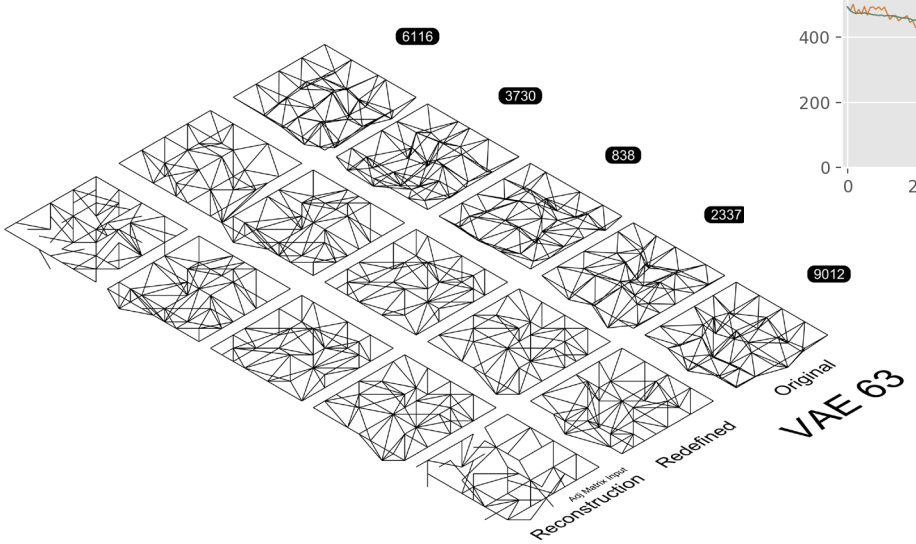
VAE MODEL: EXPLORATION 62

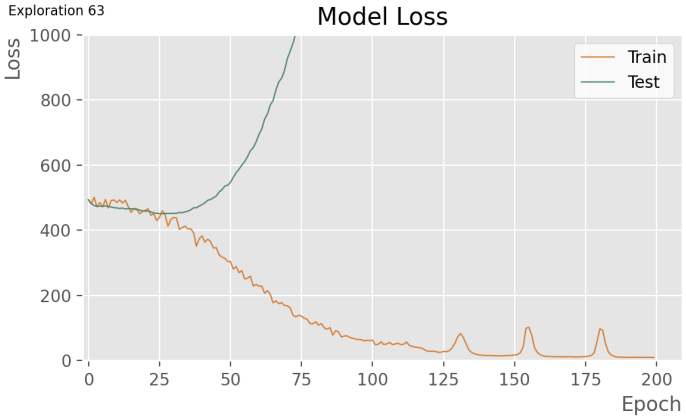


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	**'3267-1089-363-1**

VAE MODEL: EXPLORATION 63

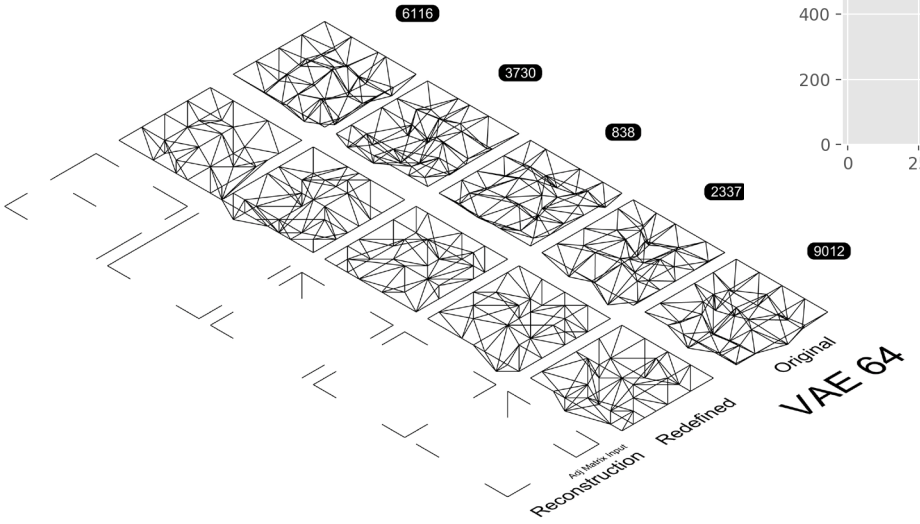




The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	**'1089-363-121'**

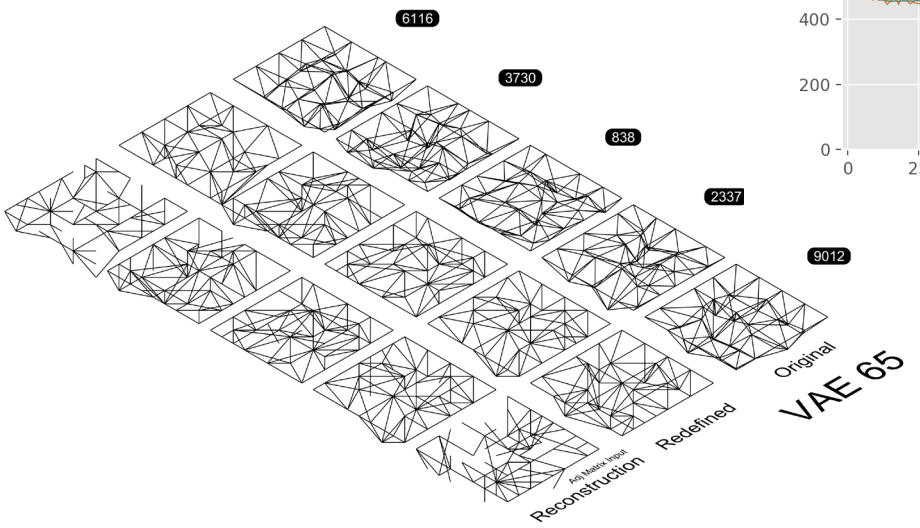
VAE MODEL: EXPLORATION 64

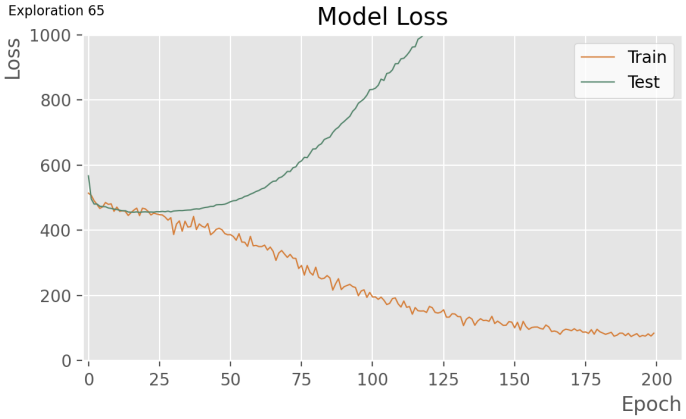


The attributes of this model are:

Latent Dimension	**2**
Multiplication of KL_loss	1
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 65

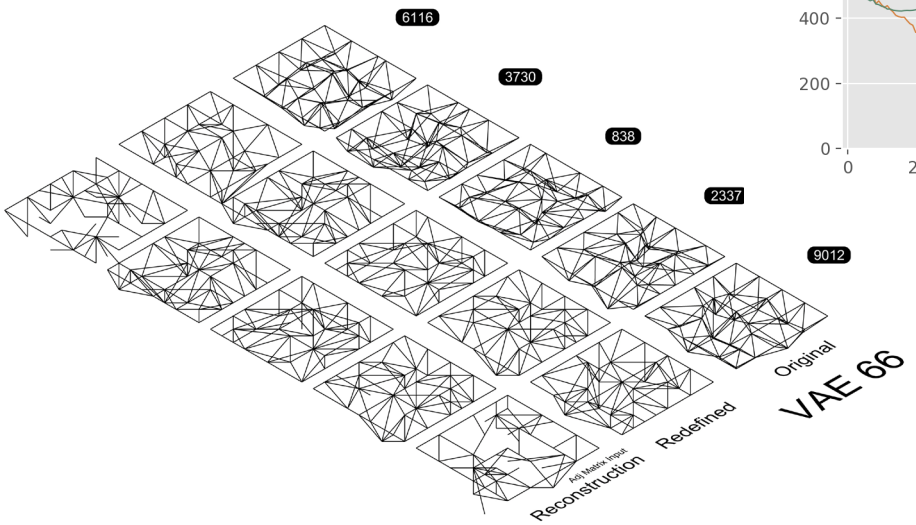


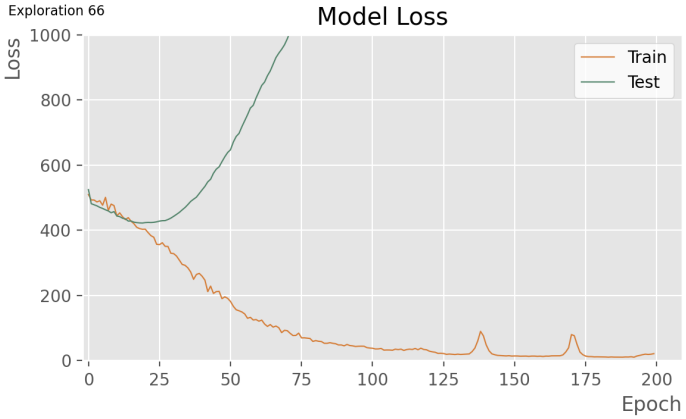


The attributes of this model are:

Latent Dimension	**7**
Multiplication of KL_loss	1
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 66

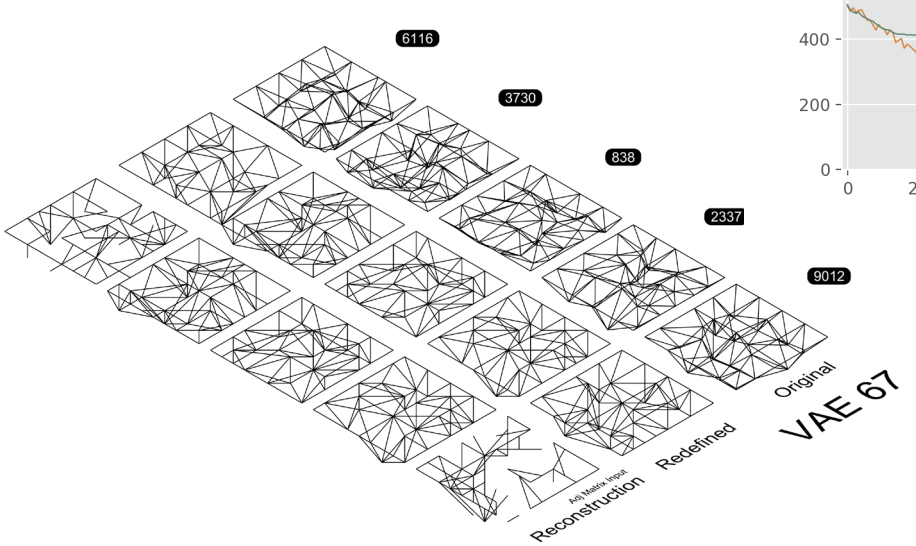


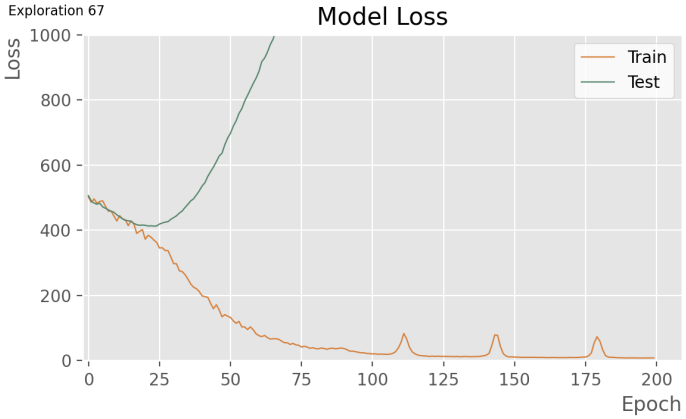


The attributes of this model are:

Latent Dimension	**19**
Multiplication of KL_loss	1
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 67

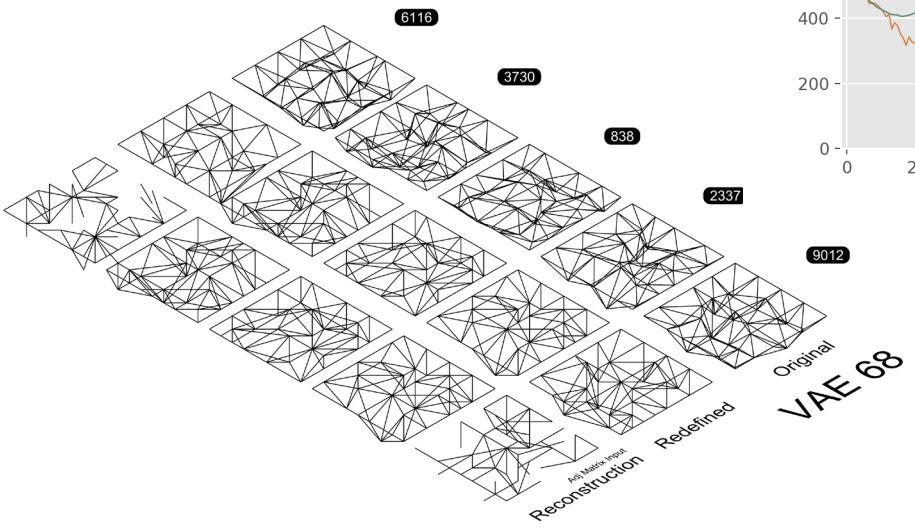


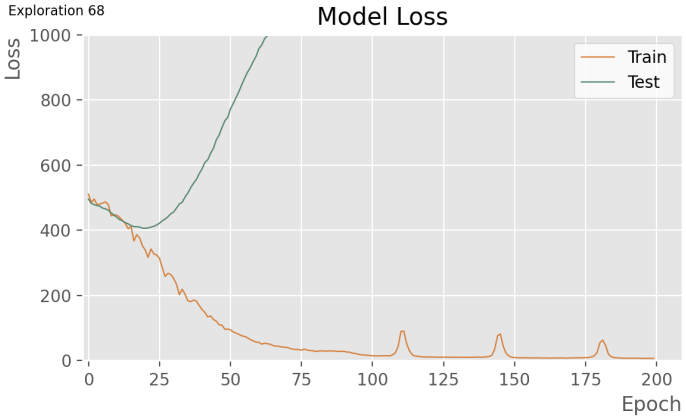


The attributes of this model are:

Latent Dimension	**30**
Multiplication of KL_loss	1
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 68

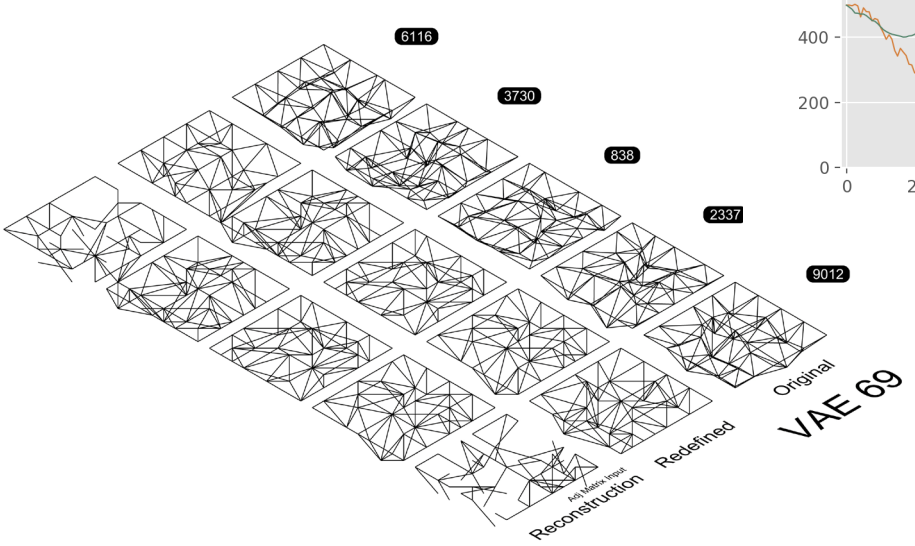


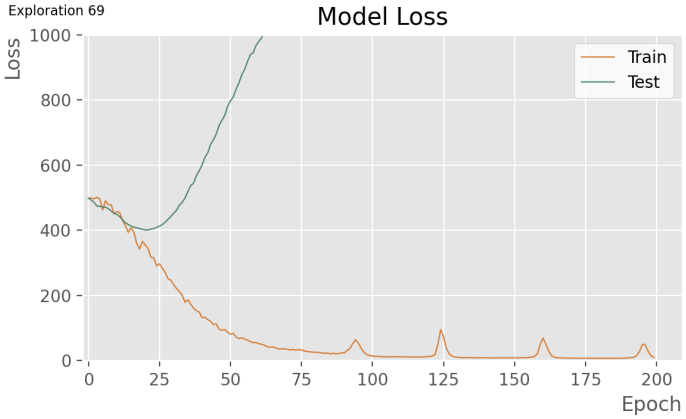


The attributes of this model are:

Latent Dimension	**40**
Multiplication of KL_loss	1
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 69

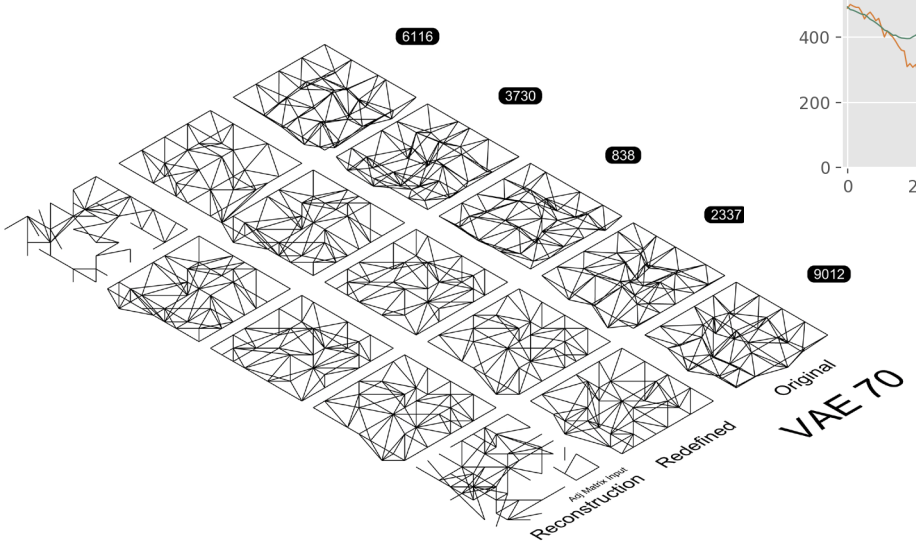


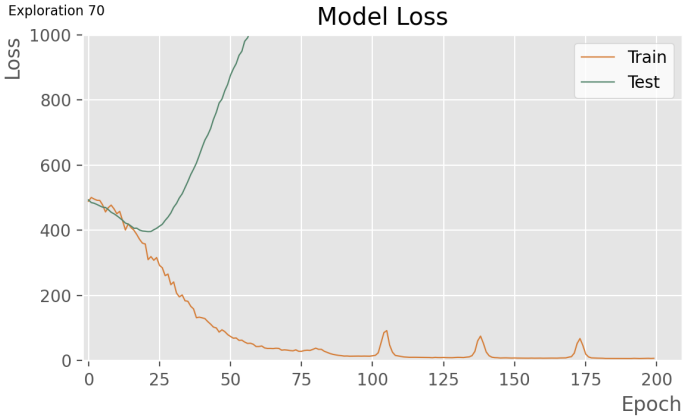


The attributes of this model are:

Latent Dimension	**50**
Multiplication of KL_loss	1
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 70

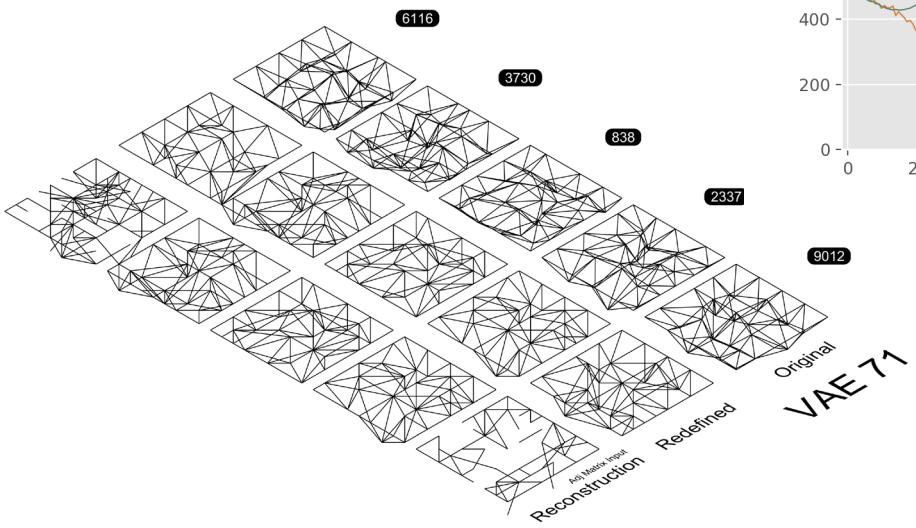




The attributes of this model are:

Latent Dimension	**60**
Multiplication of KL_loss	1
Architecture:	'1089-121'

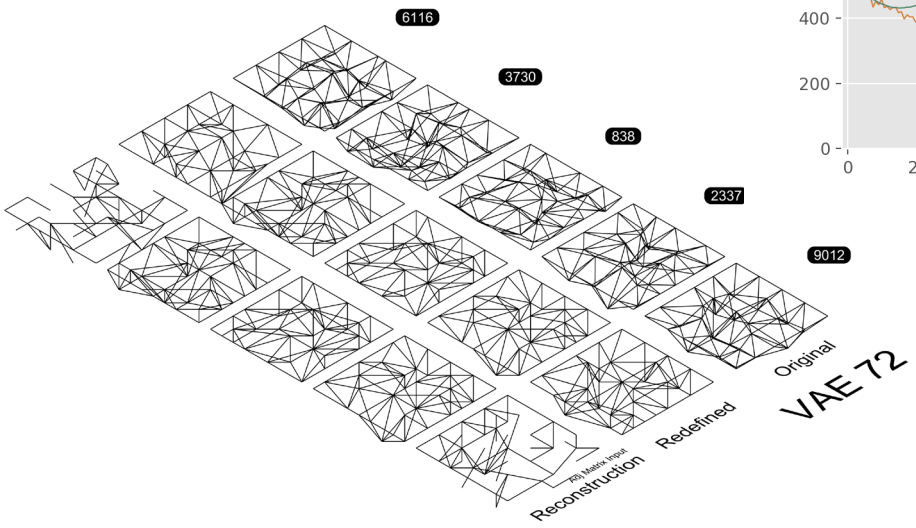
VAE MODEL: EXPLORATION 71

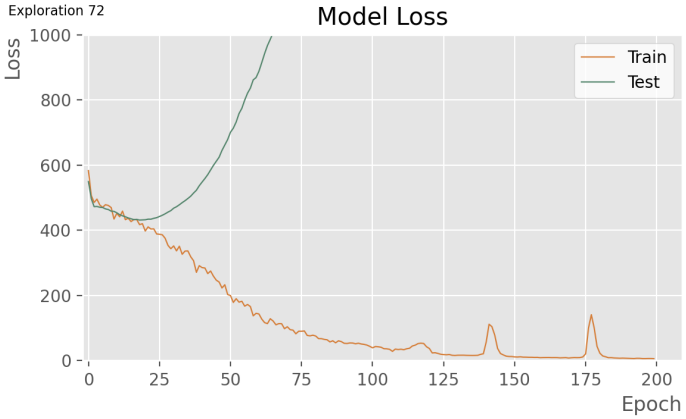


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.01**
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 72

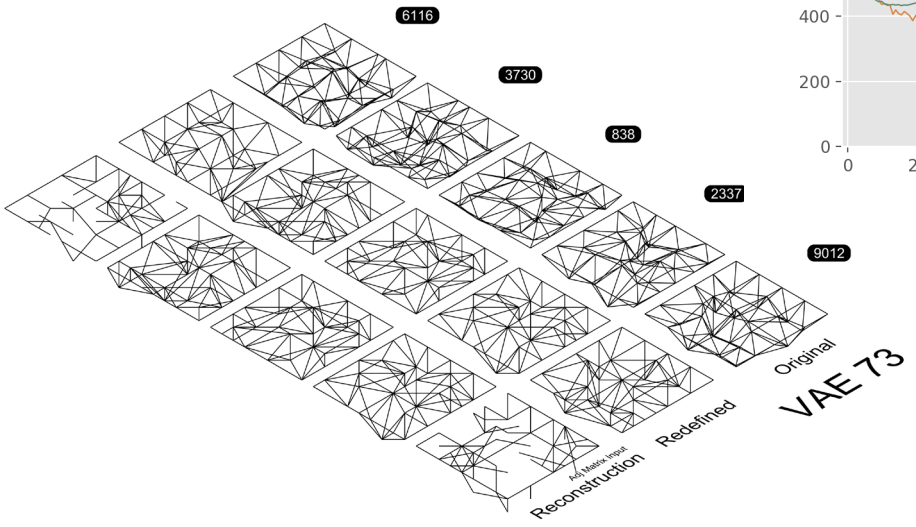


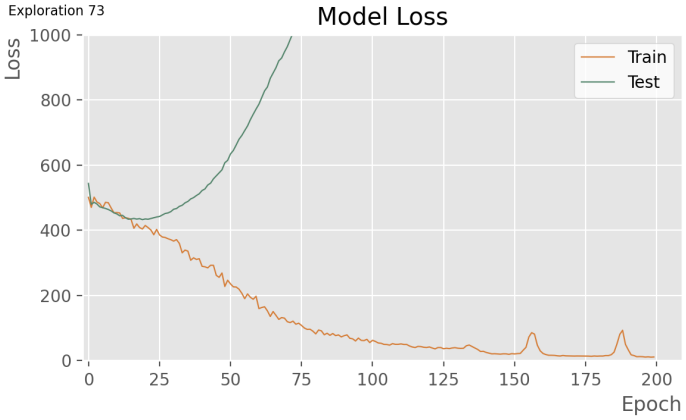


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.1**
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 73

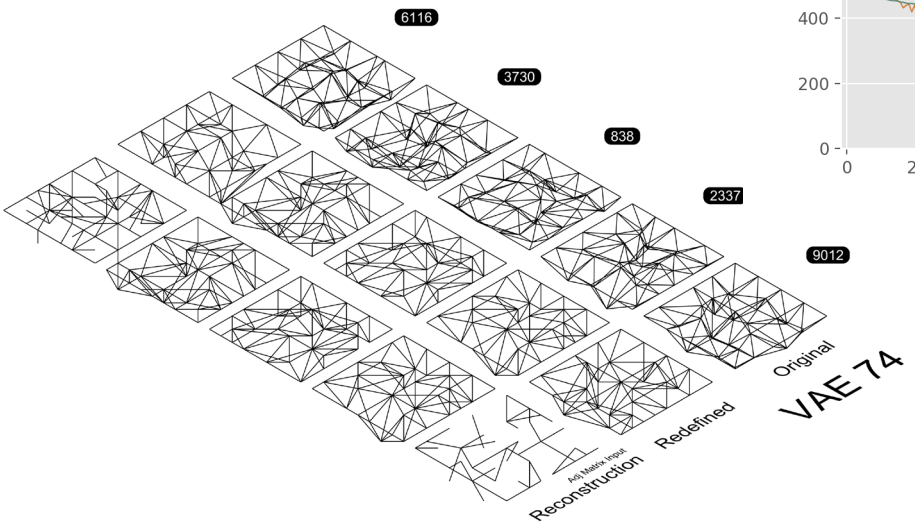


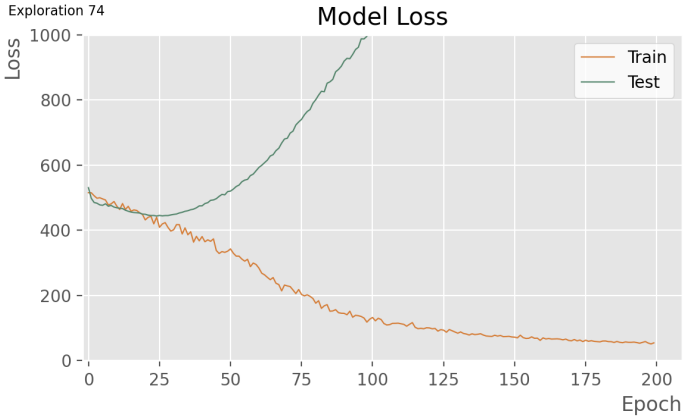


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.5**
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 74

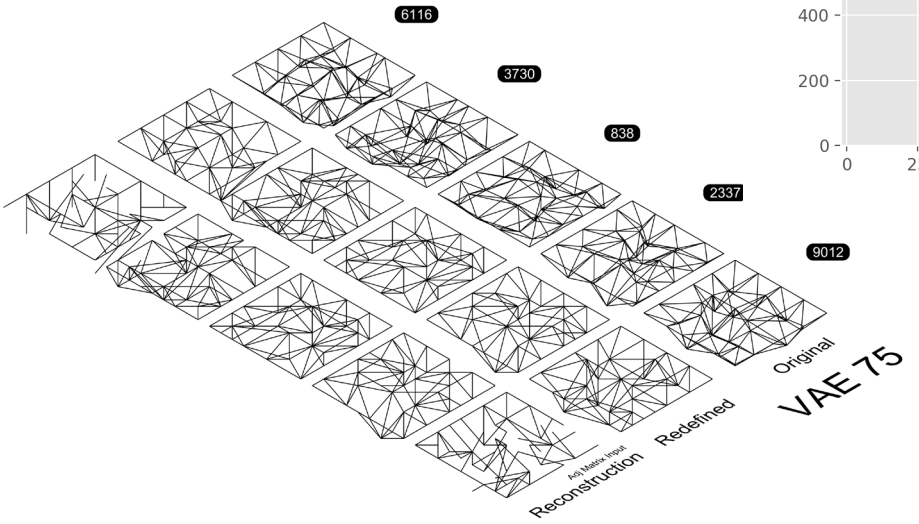


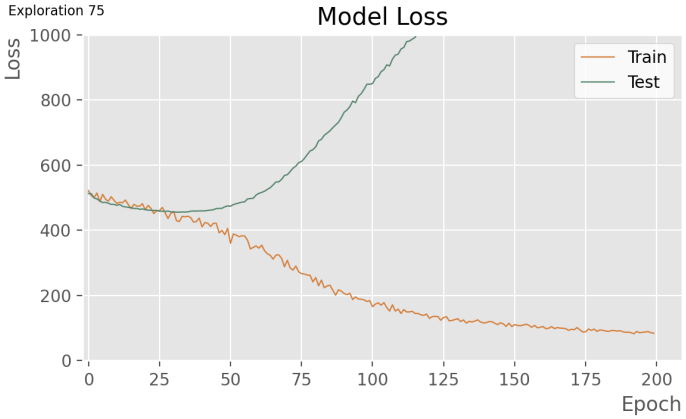


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**5**
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 75

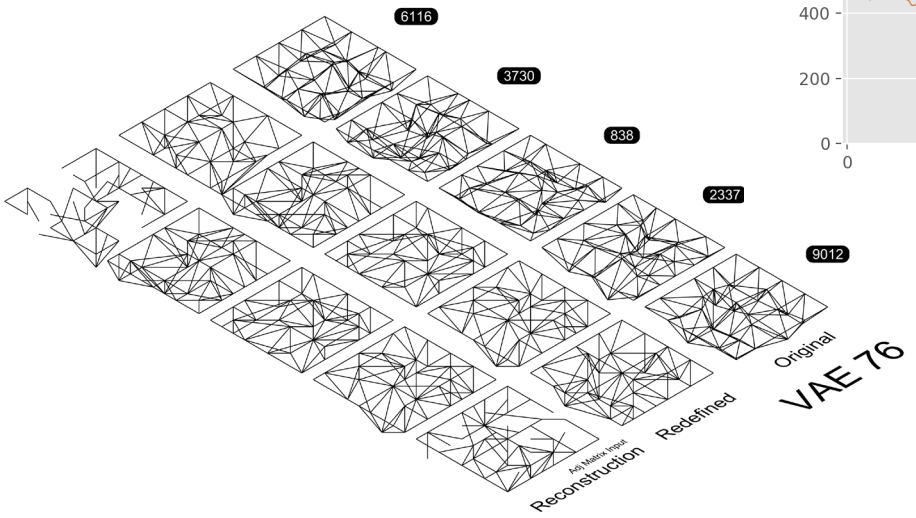


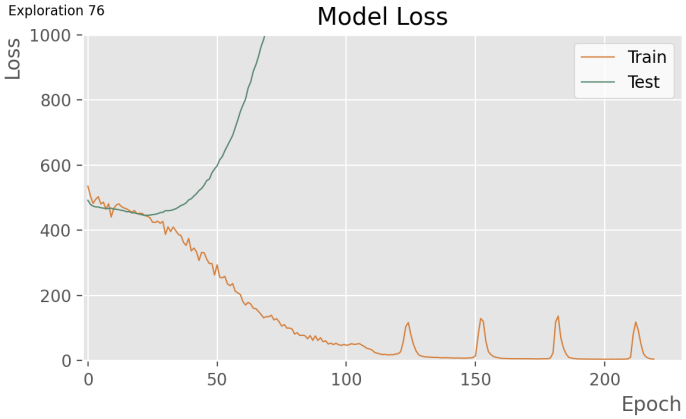


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**10**
Architecture:	'1089-121'

VAE MODEL: EXPLORATION 76



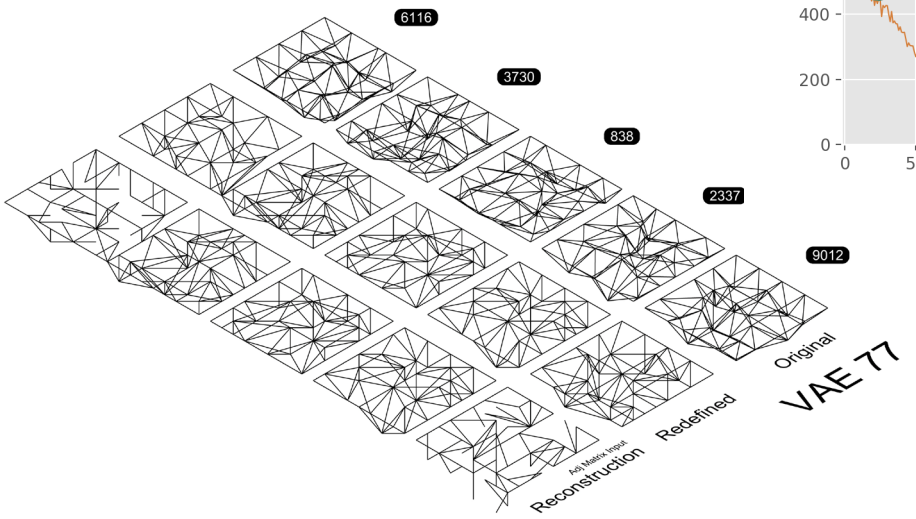


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.3**
Architecture:	**'1089-363-121'**

*with dropout layers

VAE MODEL: EXPLORATION 77

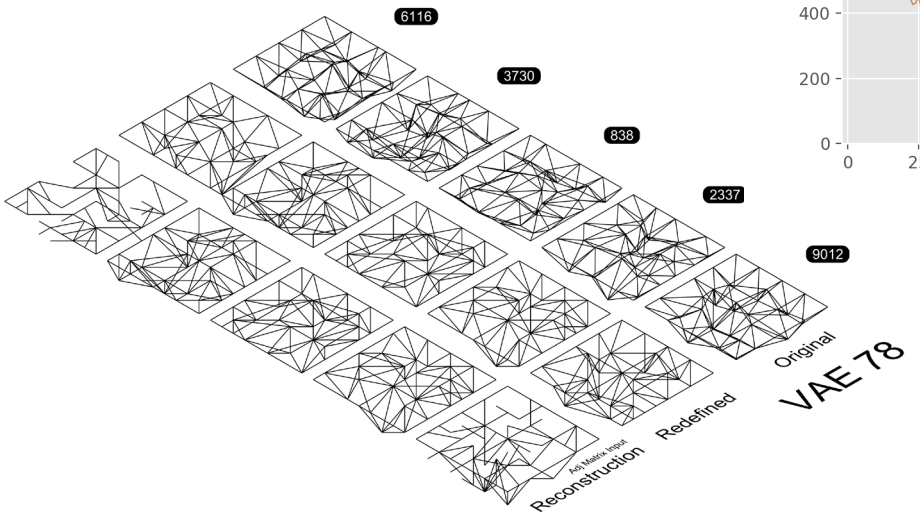


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.3**
Architecture:	**'1089-363-121'**

*with 400 epochs

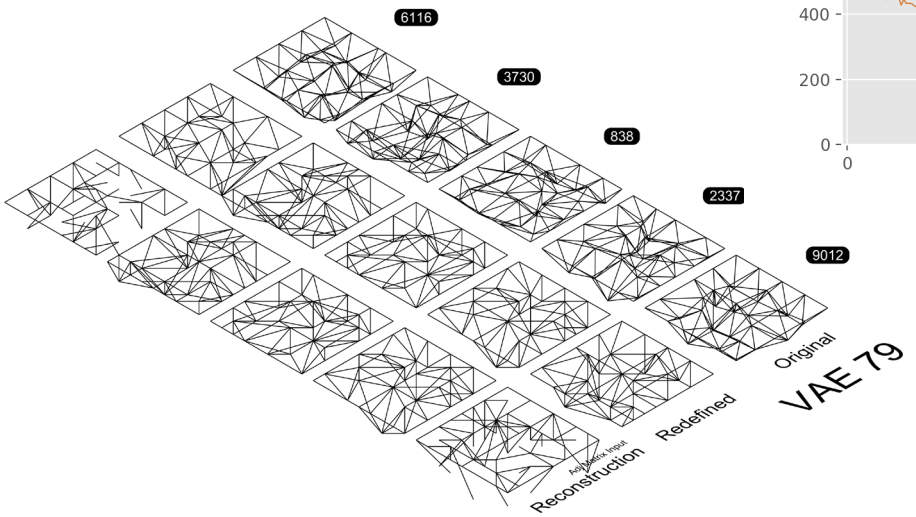
VAE MODEL: EXPLORATION 78

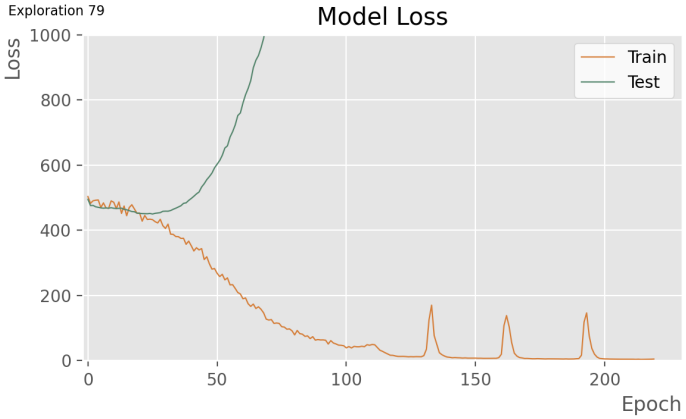


The attributes of this model are:

Latent Dimension	**15**
Multiplication of KL_loss	**0.3**
Architecture:	**'1089-363-121'**

VAE MODEL: EXPLORATION 79

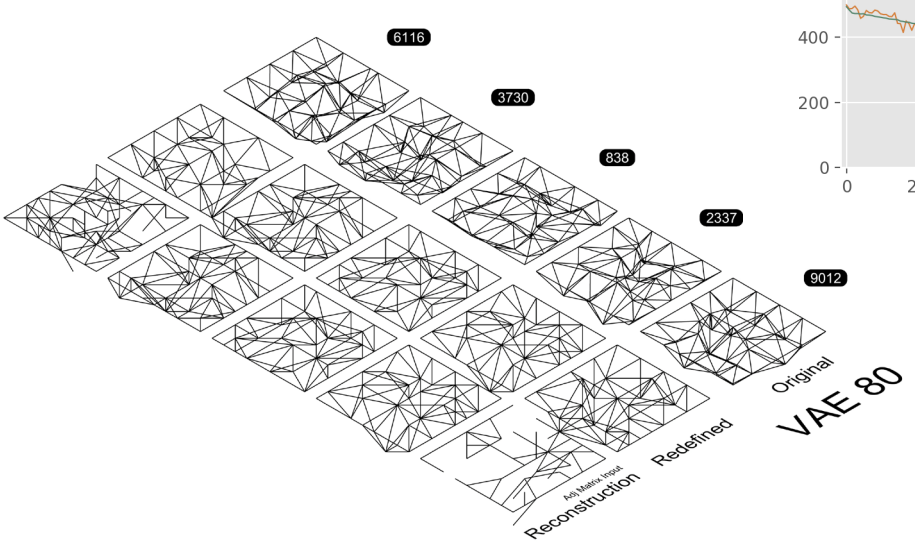


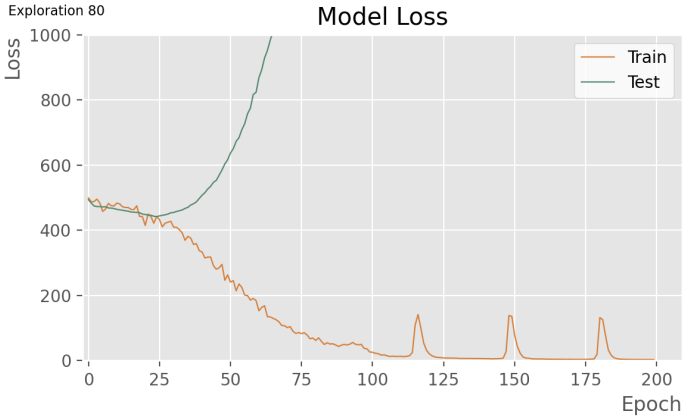


The attributes of this model are:

Latent Dimension	**11**
Multiplication of KL_loss	**0.3**
Architecture:	**'1089-363-121'**

VAE MODEL: EXPLORATION 80

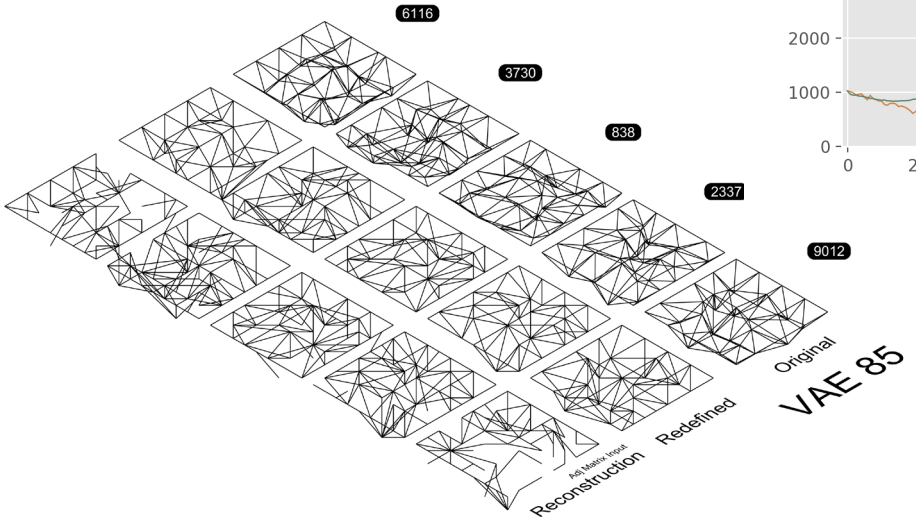


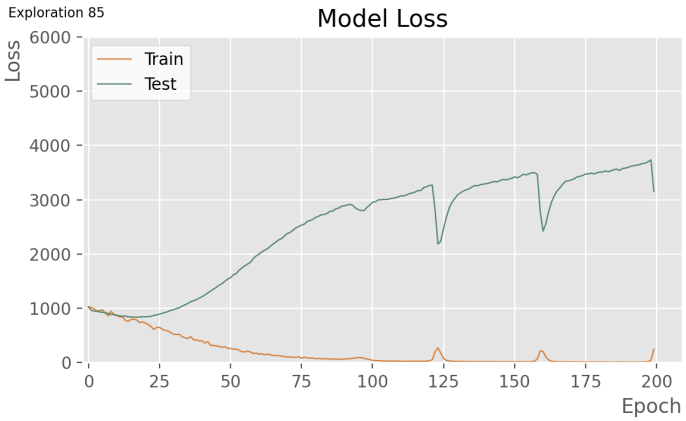


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.3**
Architecture:	**'1089-363-121'**

VAE MODEL: EXPLORATION 85

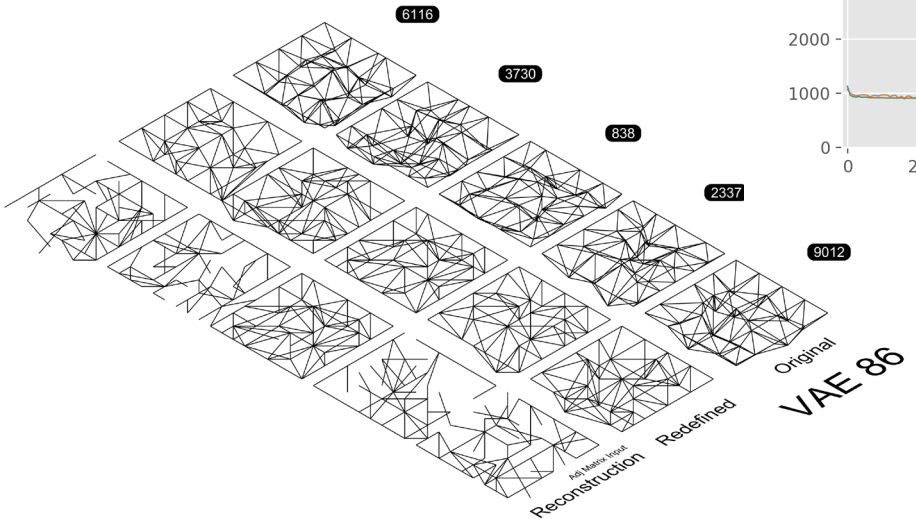




The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	1
Architecture:	'972-243'

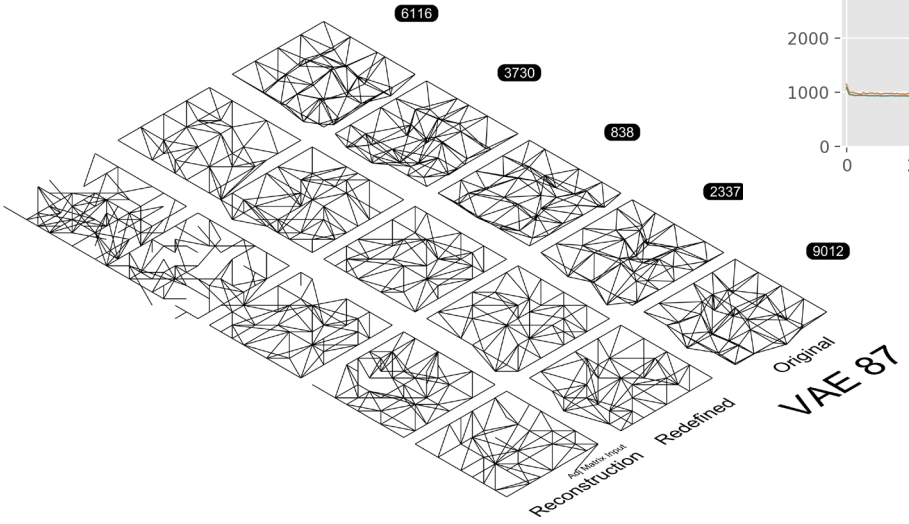
VAE MODEL: EXPLORATION 86

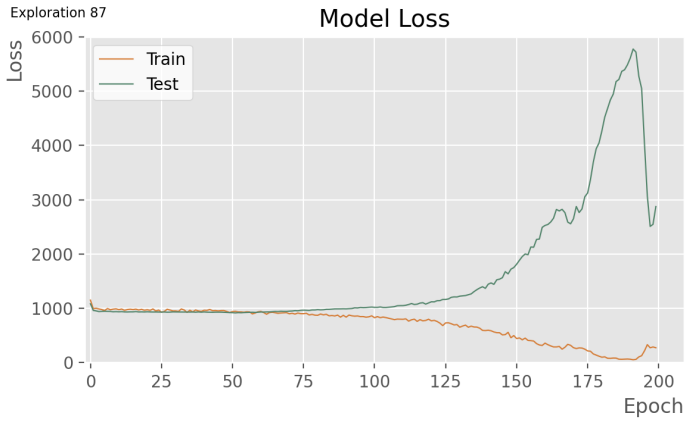


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	1
Architecture:	**'1944-972-243'**

VAE MODEL: EXPLORATION 87

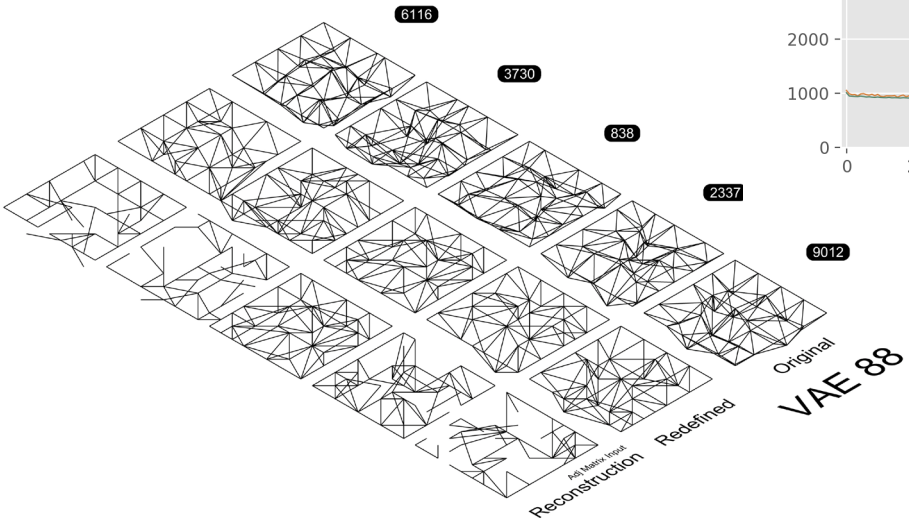




The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	1
Architecture:	**'1944-972-486-24**

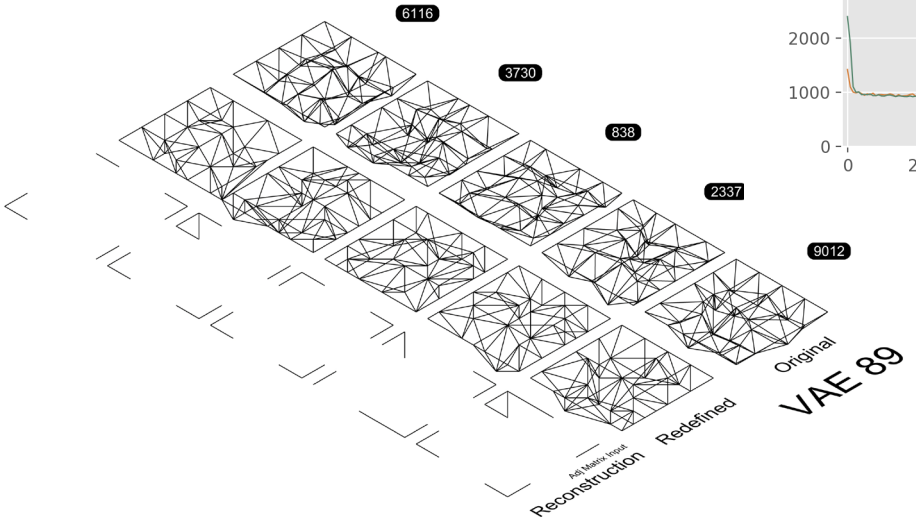
VAE MODEL: EXPLORATION 88

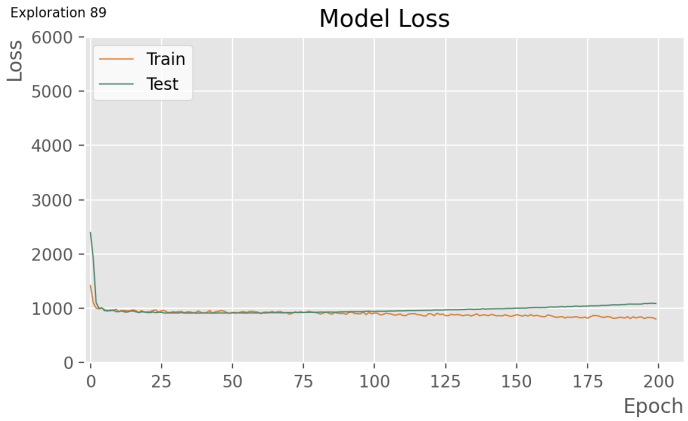


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	1
Architecture:	**'3888-1944-243'**

VAE MODEL: EXPLORATION 89

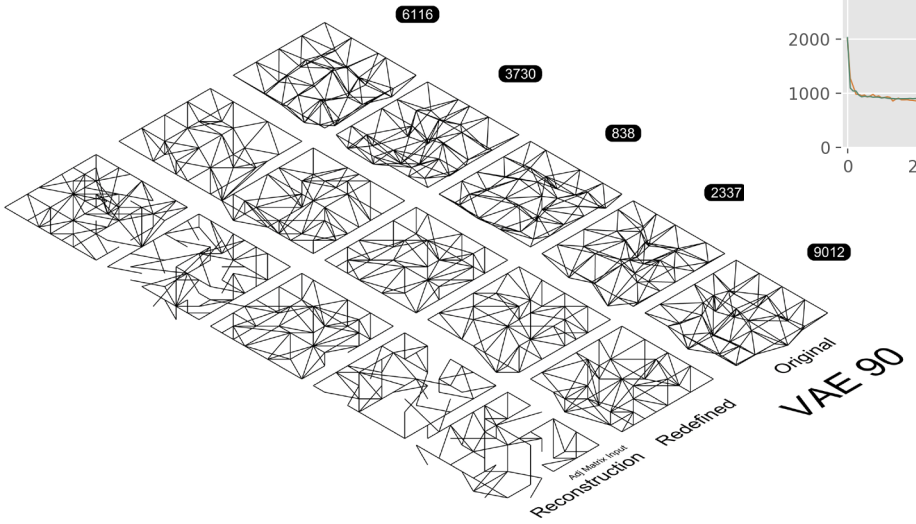


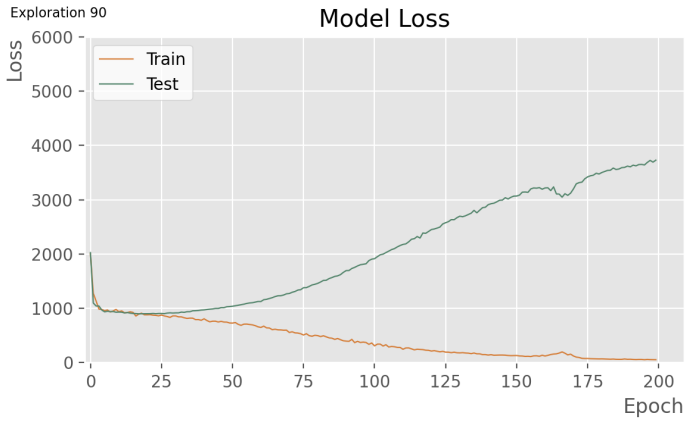


The attributes of this model are:

Latent Dimension	**2**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 90

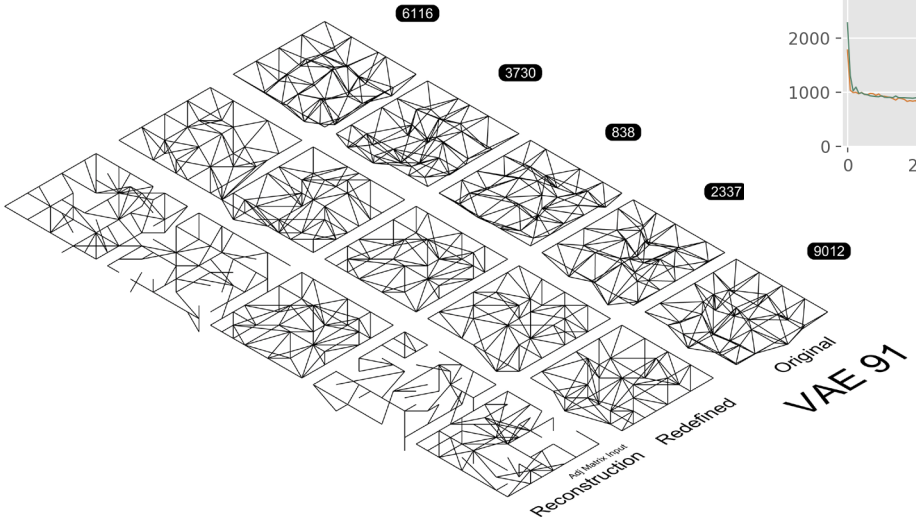


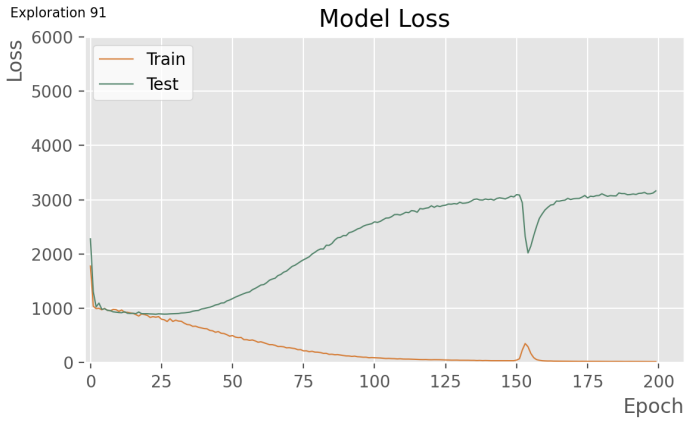


The attributes of this model are:

Latent Dimension	**7**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 91

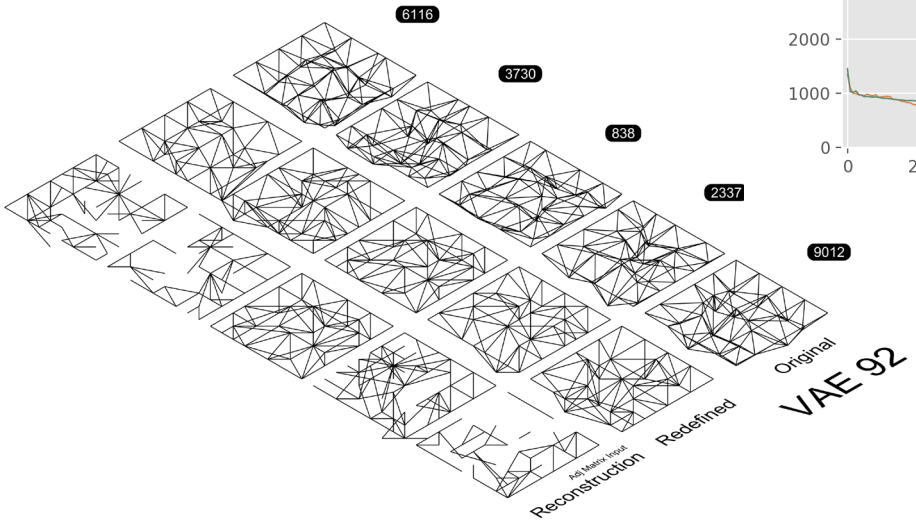


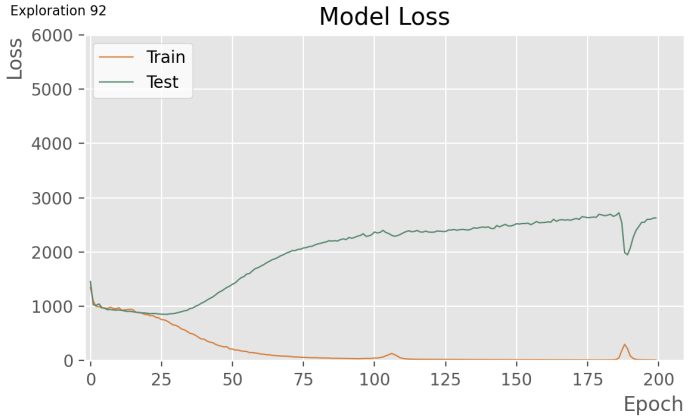


The attributes of this model are:

Latent Dimension	**13**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 92

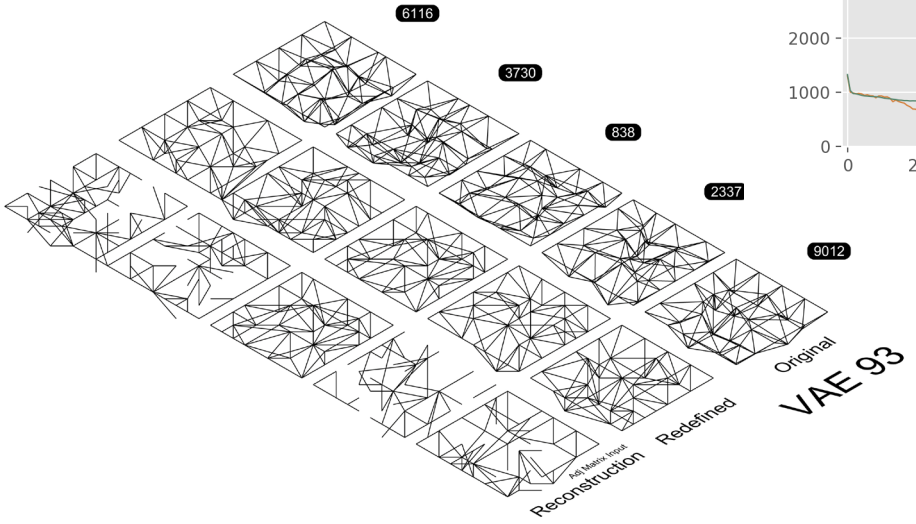


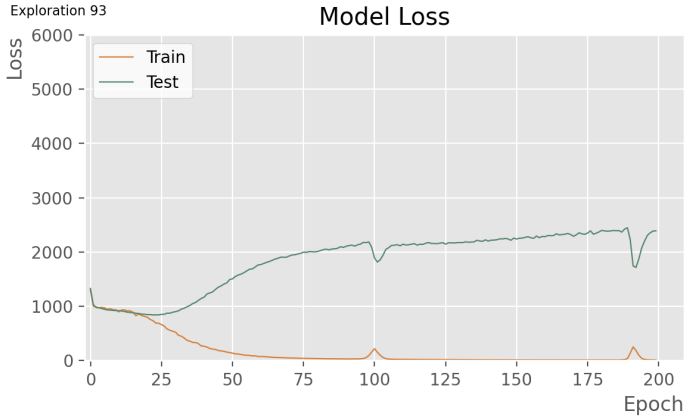


The attributes of this model are:

Latent Dimension	**30**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 93

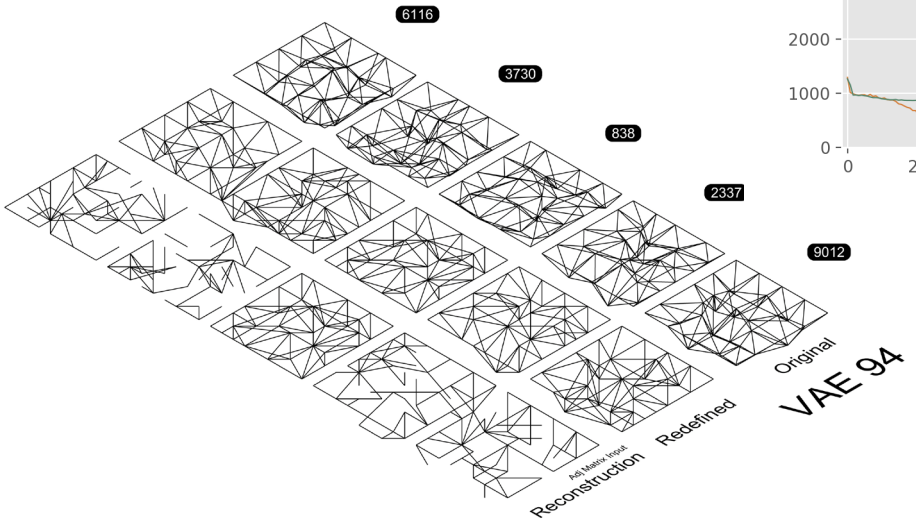


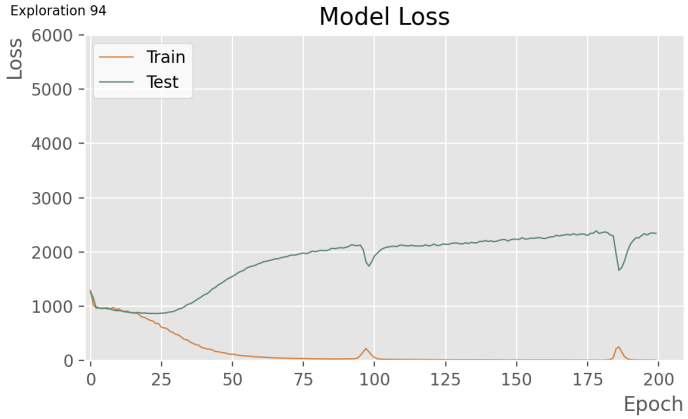


The attributes of this model are:

Latent Dimension	**40**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 94

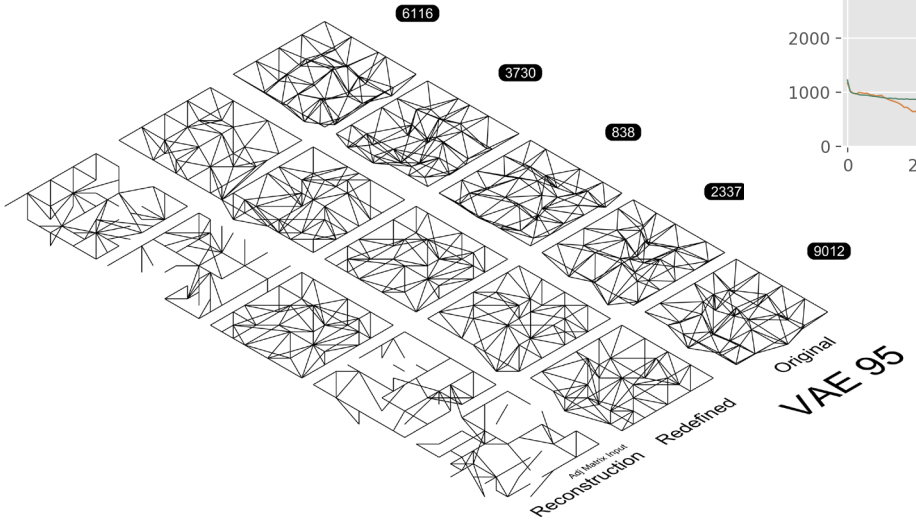


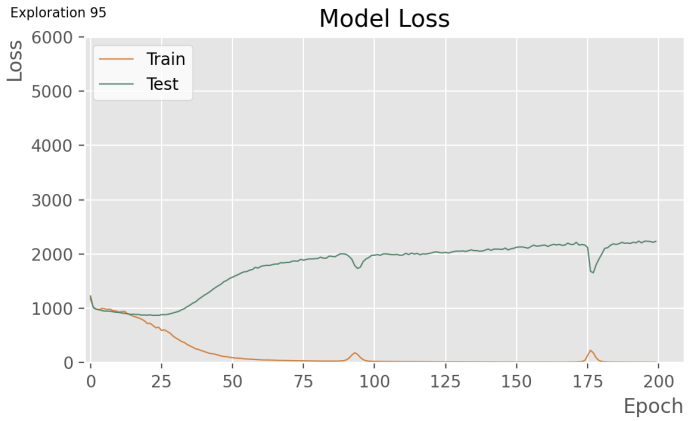


The attributes of this model are:

Latent Dimension	**50**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 95

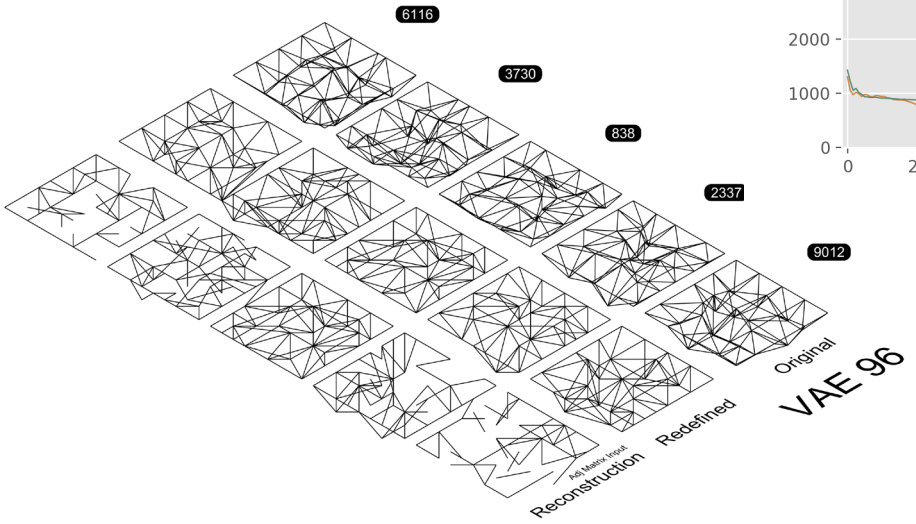




The attributes of this model are:

Latent Dimension	**60**
Multiplication of KL_loss	1
Architecture:	'972-243'

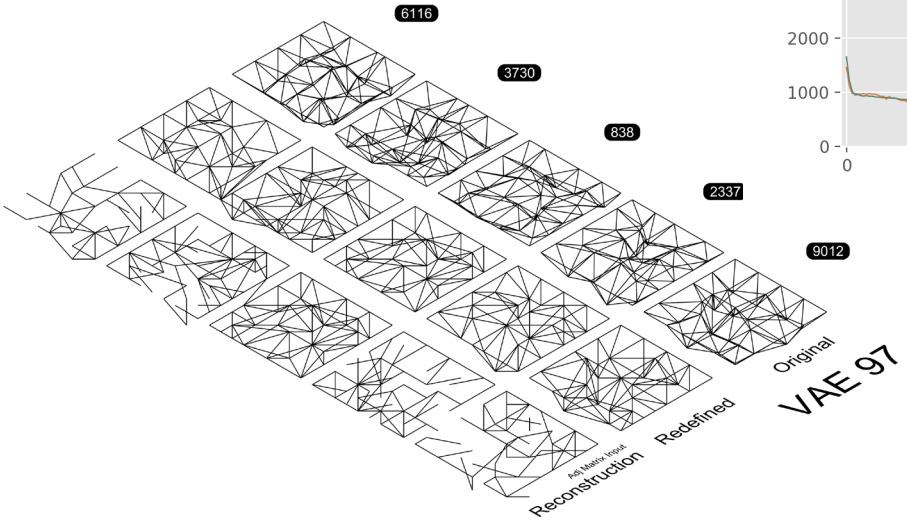
VAE MODEL: EXPLORATION 96



The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**0.01**
Architecture:	'972-243'

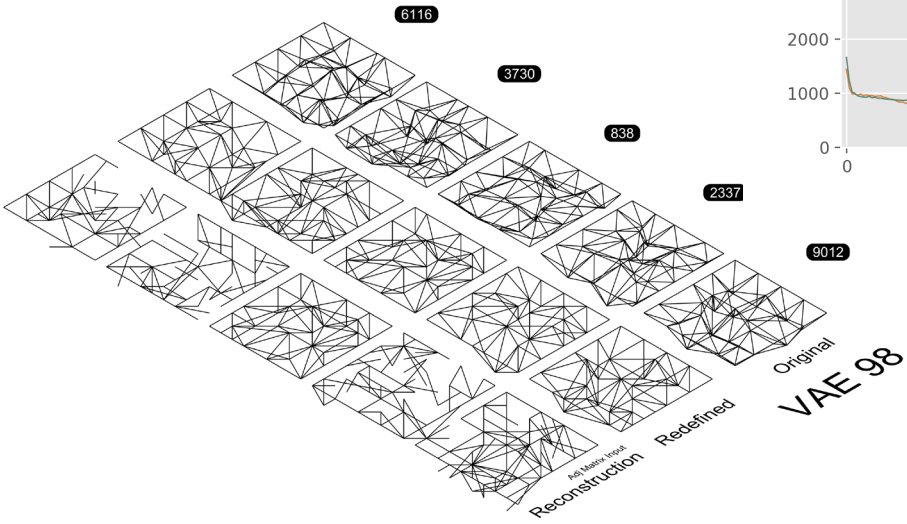
VAE MODEL: EXPLORATION 97

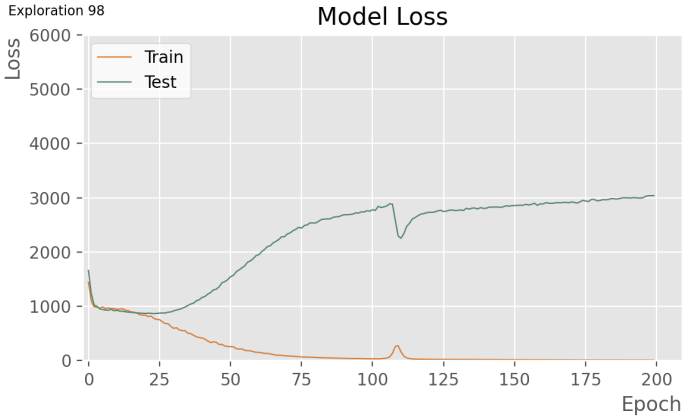


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**0.1**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 98

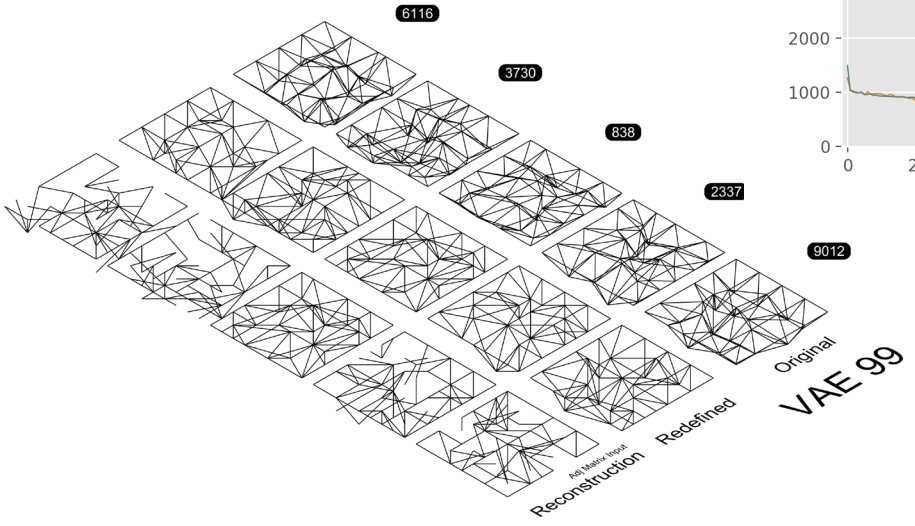


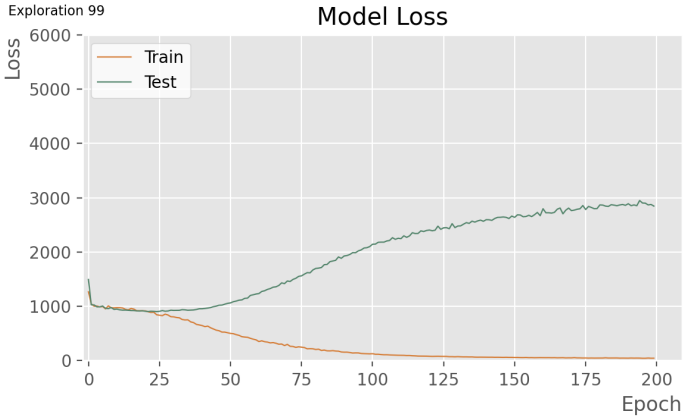


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**0.5**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 99

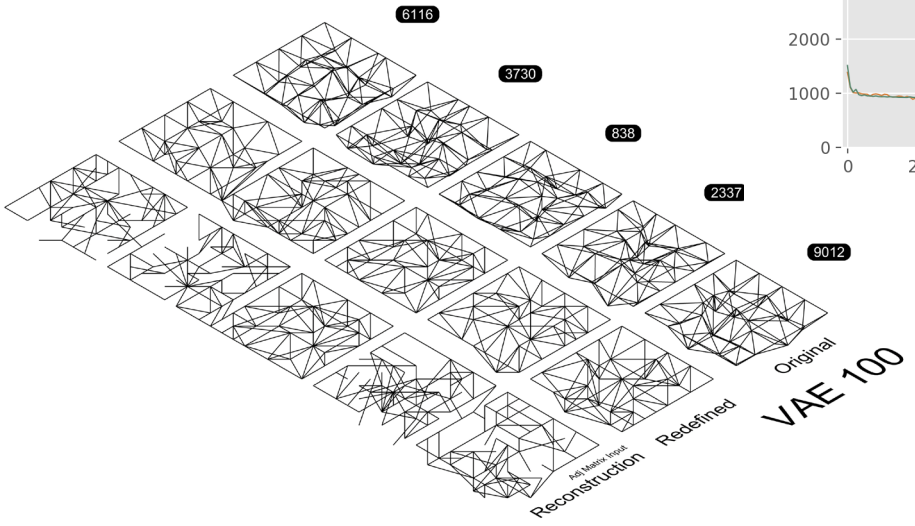


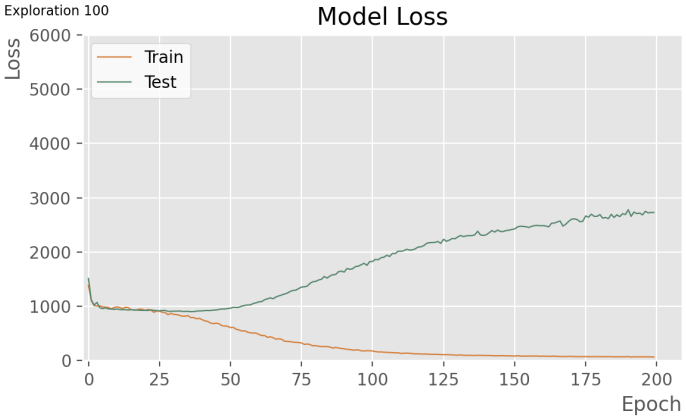


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**5**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 100

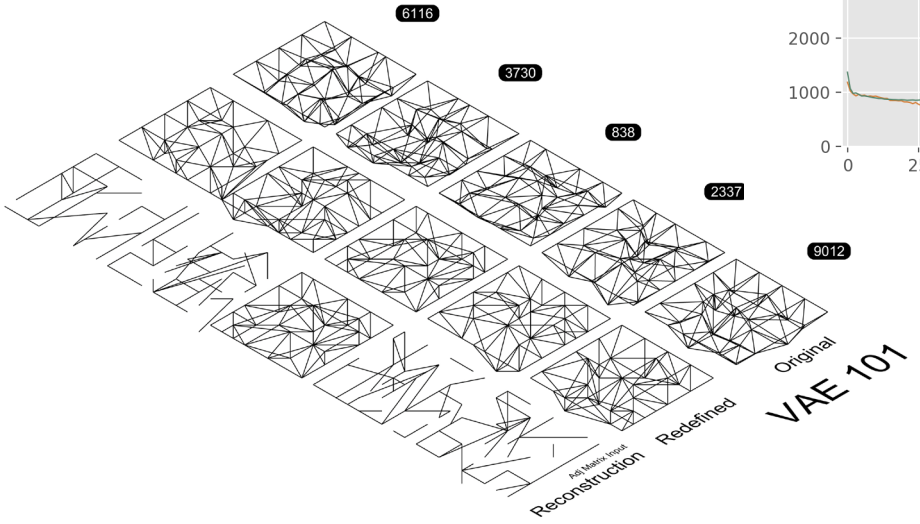


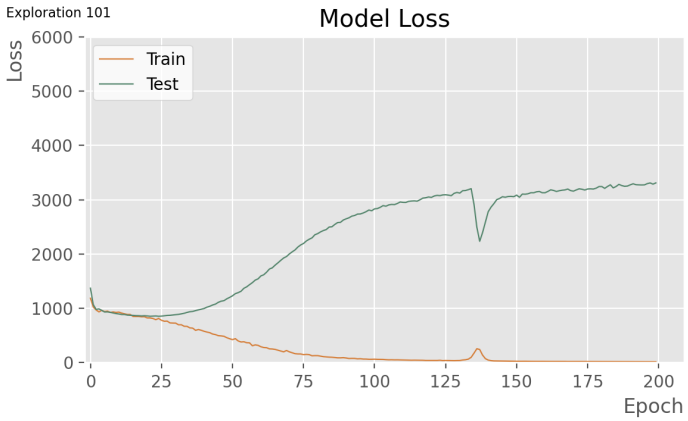


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**10**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 101

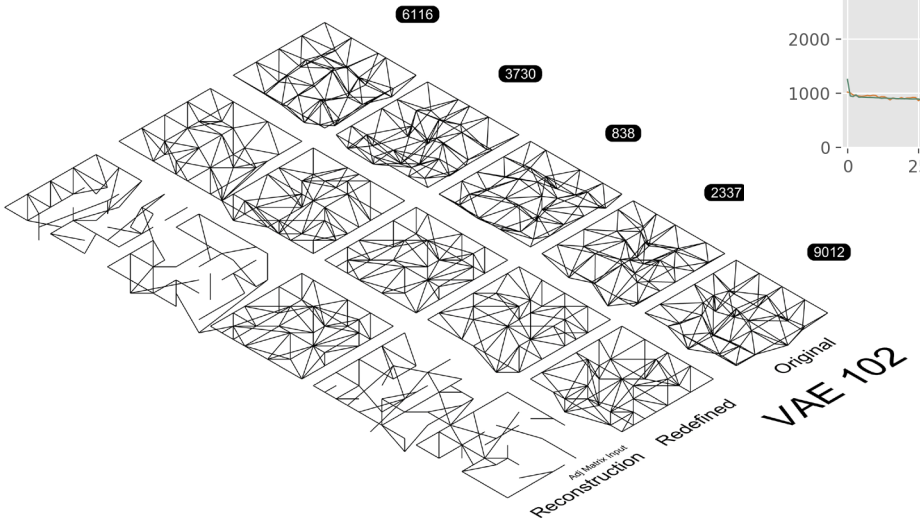




The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	'972-243'

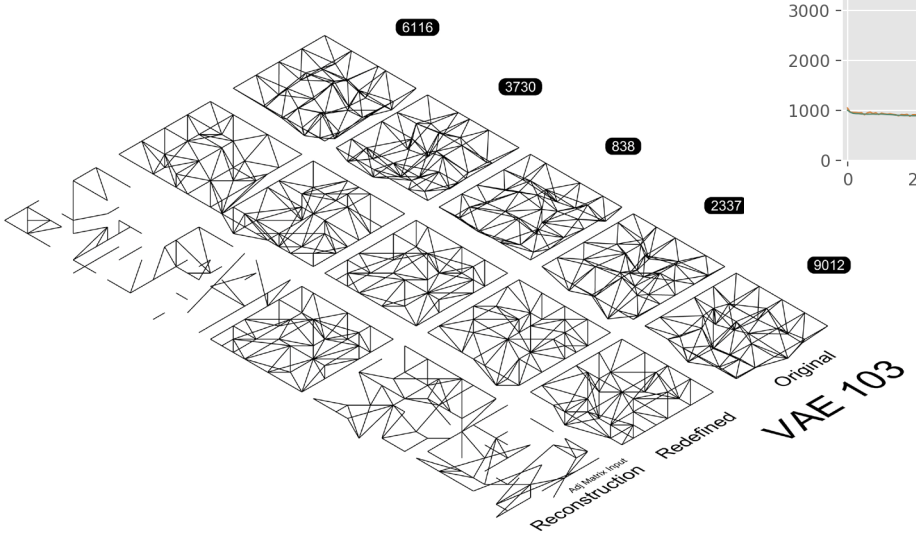
VAE MODEL: EXPLORATION 102

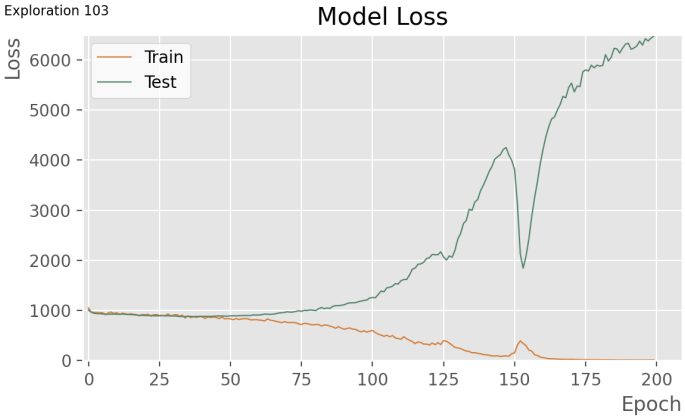


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	**'1944-972-243'**

VAE MODEL: EXPLORATION 103

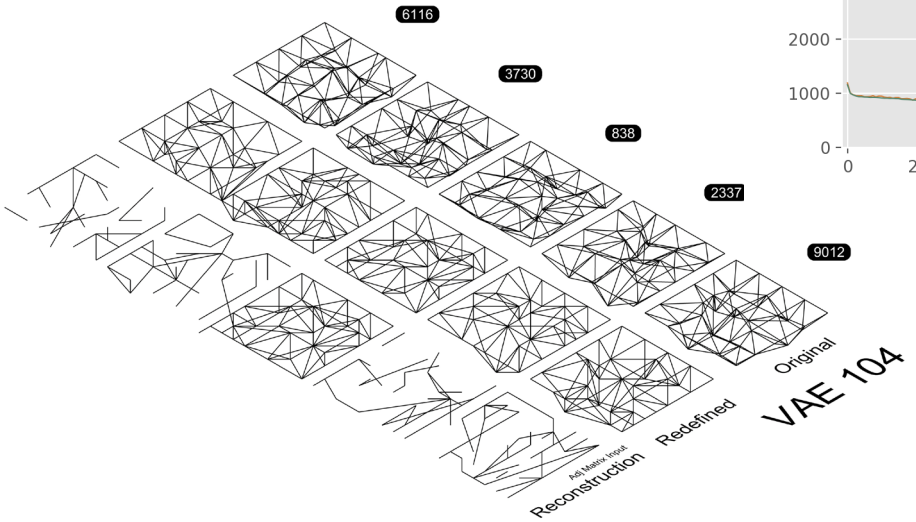


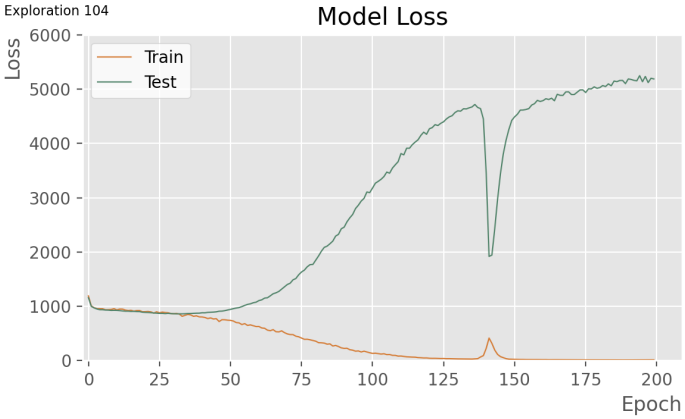


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	**'1944-972-486-24**

VAE MODEL: EXPLORATION 104

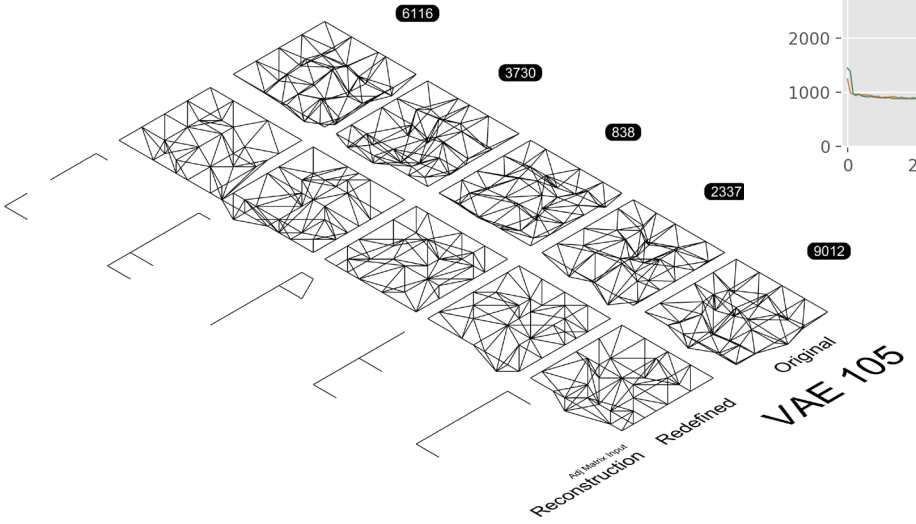


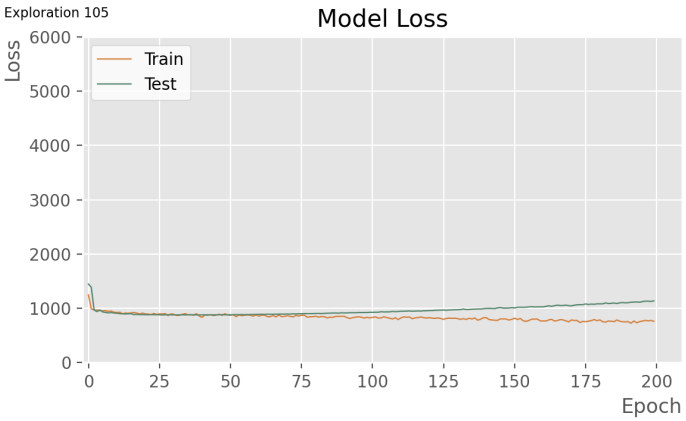


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	1
Architecture:	**'972-486-243'**

VAE MODEL: EXPLORATION 105

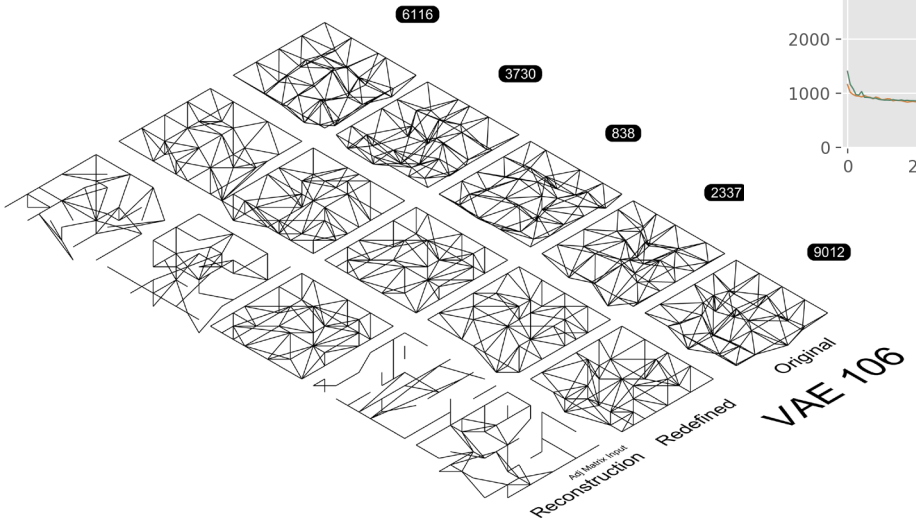


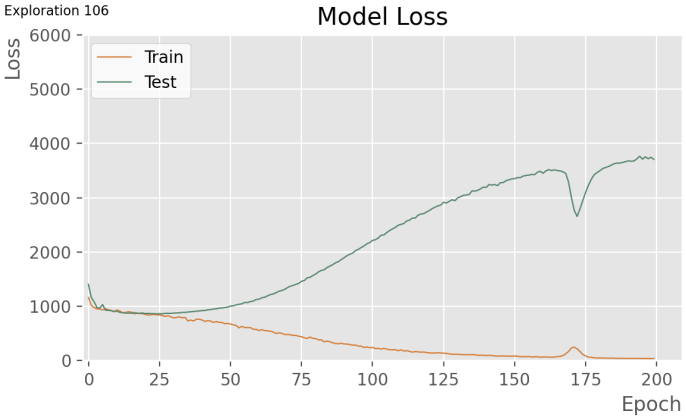


The attributes of this model are:

Latent Dimension	**2**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 106

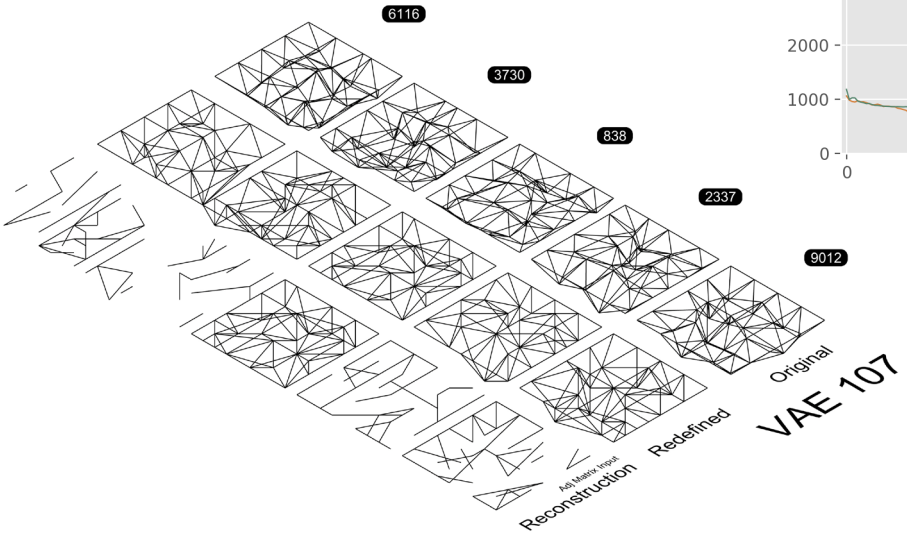


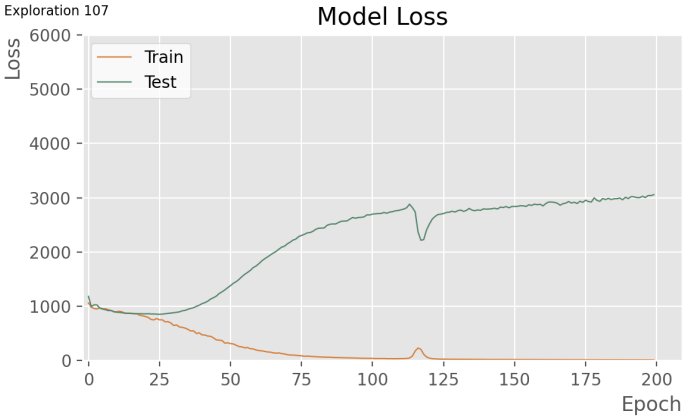


The attributes of this model are:

Latent Dimension	**7**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 107

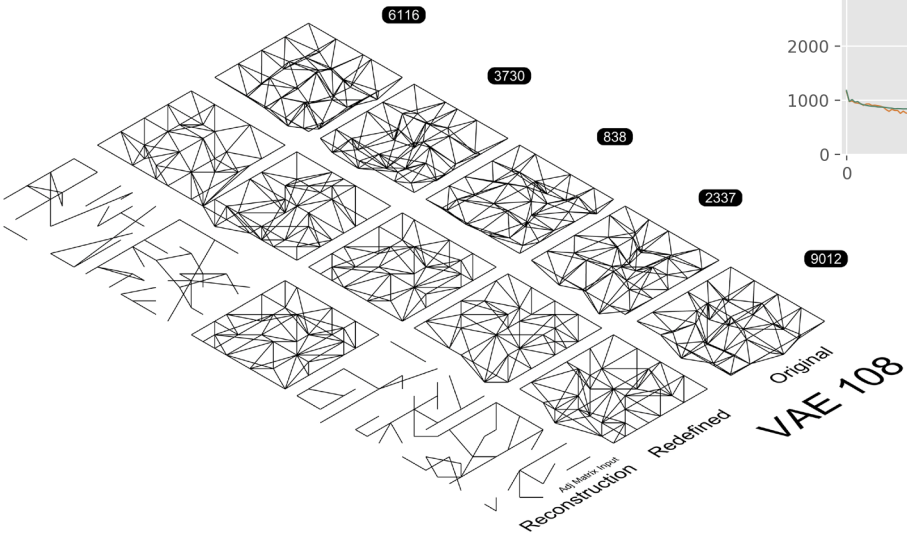


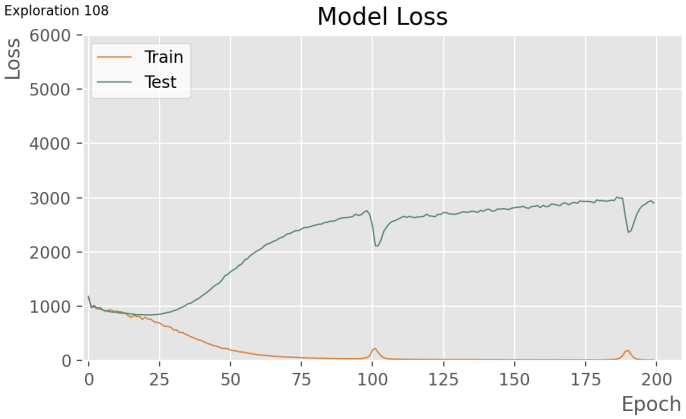


The attributes of this model are:

Latent Dimension	**19**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 108

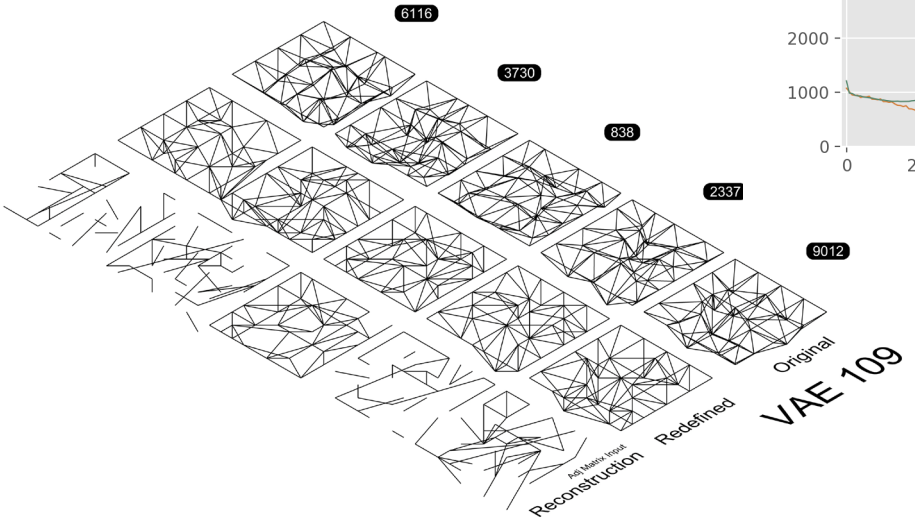




The attributes of this model are:

Latent Dimension	**30**
Multiplication of KL_loss	1
Architecture:	'972-243'

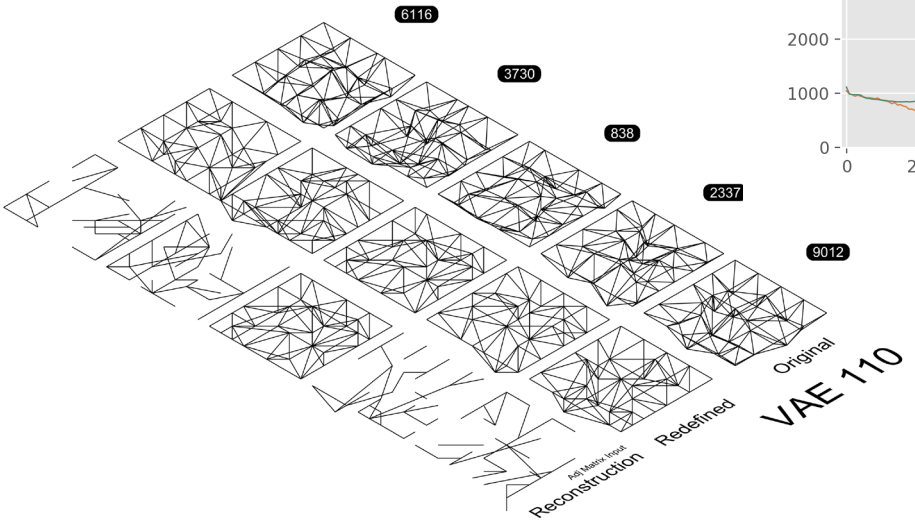
VAE MODEL: EXPLORATION 109



The attributes of this model are:

Latent Dimension	**40**
Multiplication of KL_loss	1
Architecture:	'972-243'

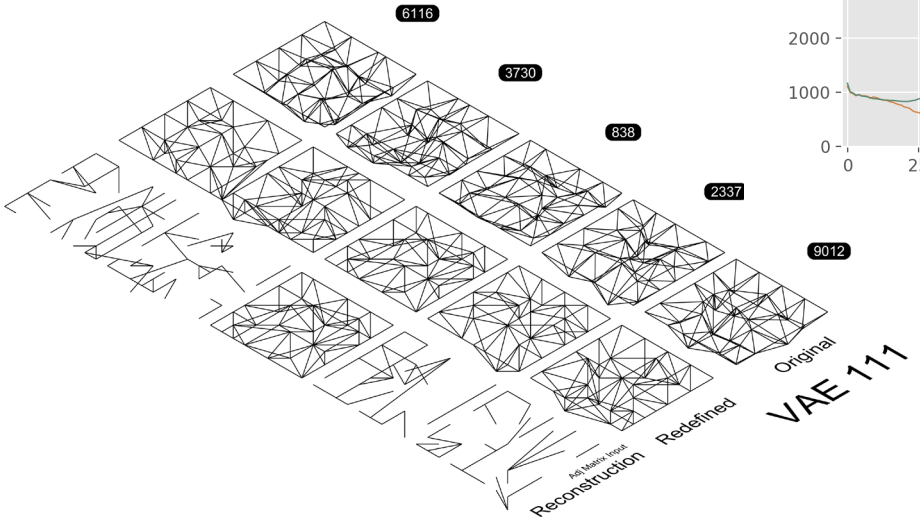
VAE MODEL: EXPLORATION 110

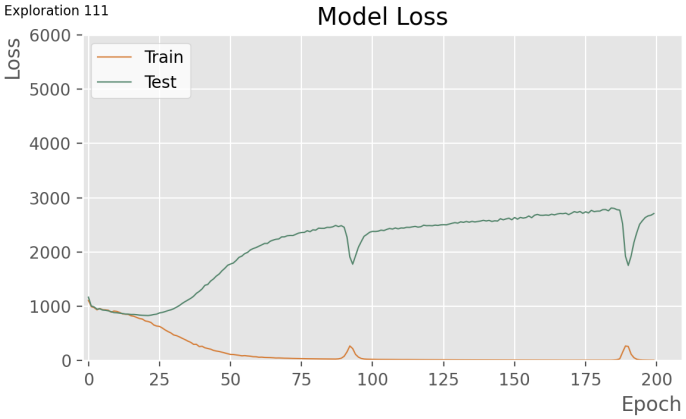


The attributes of this model are:

Latent Dimension	**50**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 111

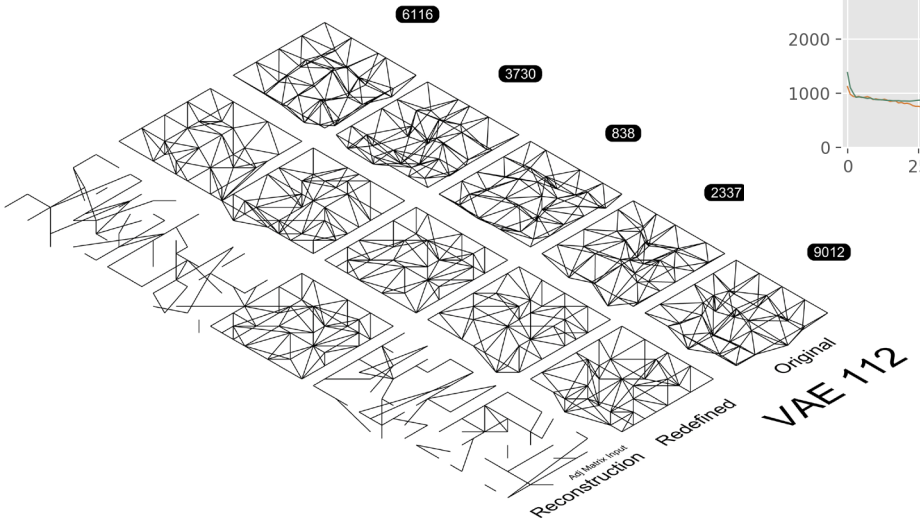


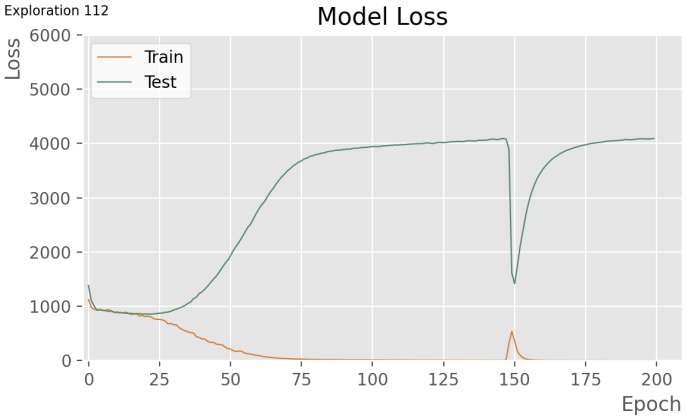


The attributes of this model are:

Latent Dimension	**60**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 112

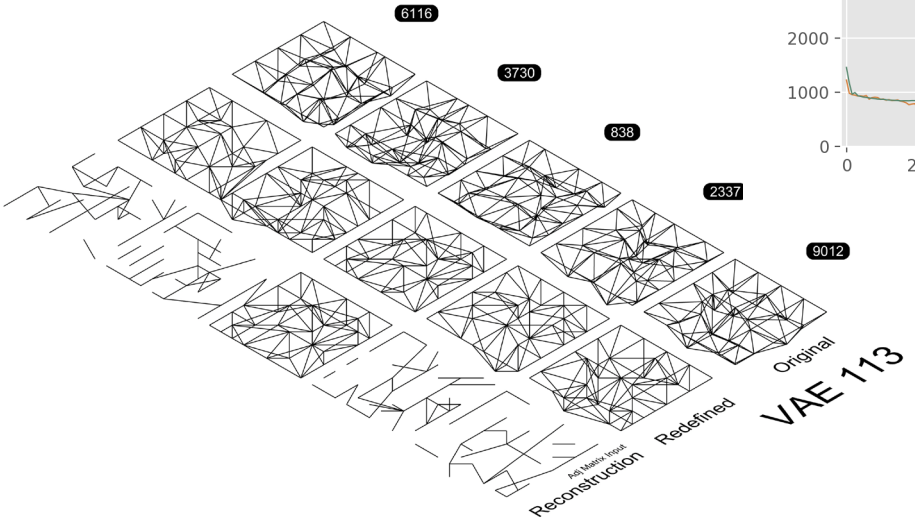


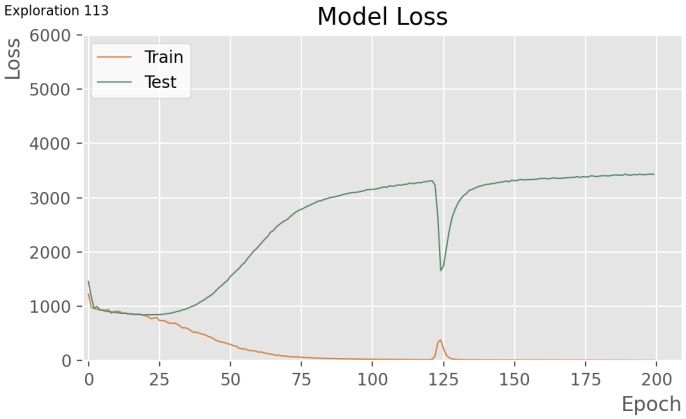


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.01**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 113

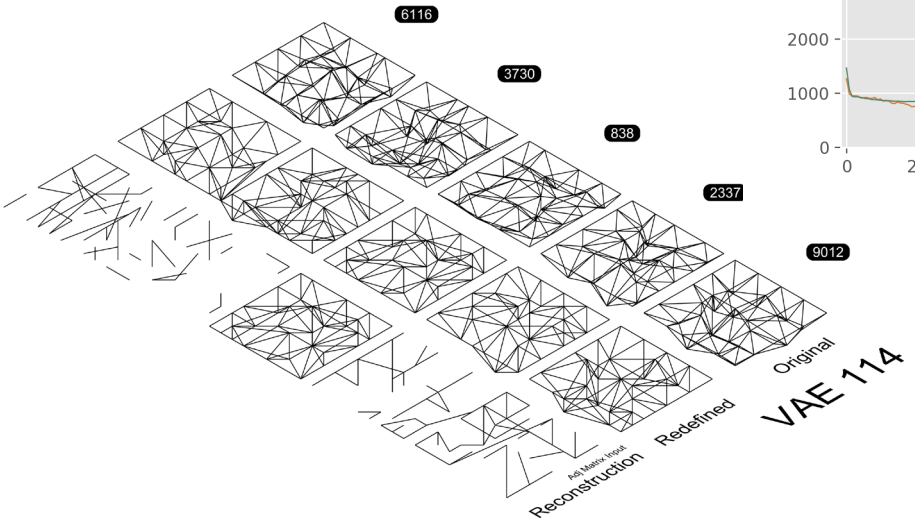


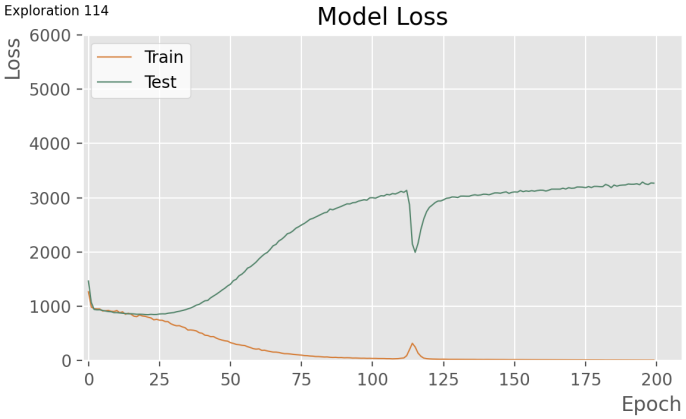


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.1**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 114

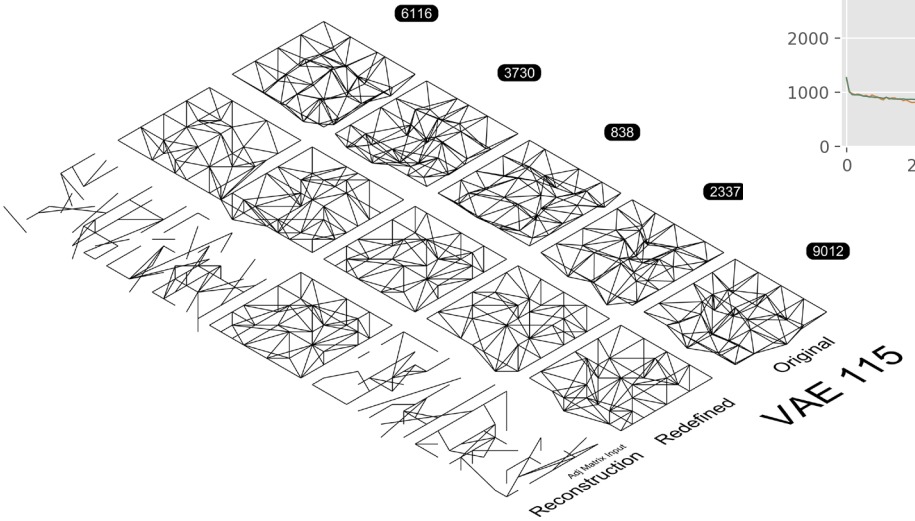


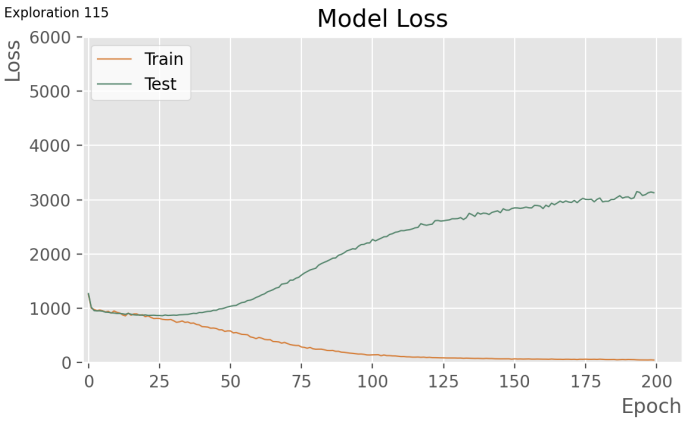


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**0.5**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 115

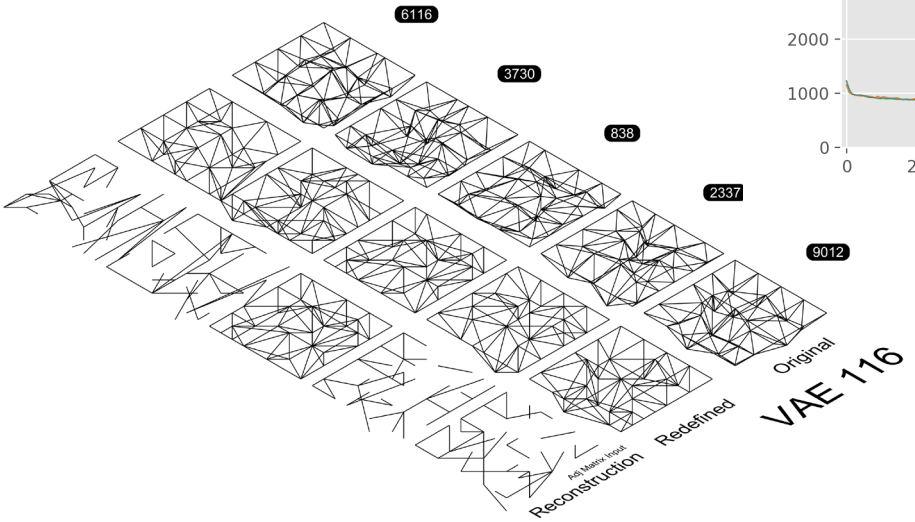


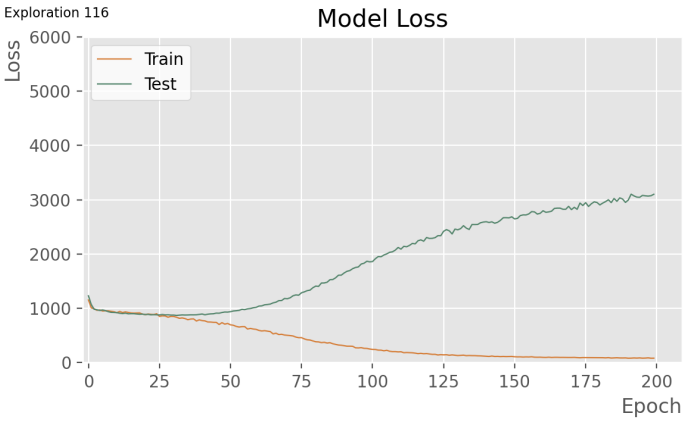


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**5**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 116

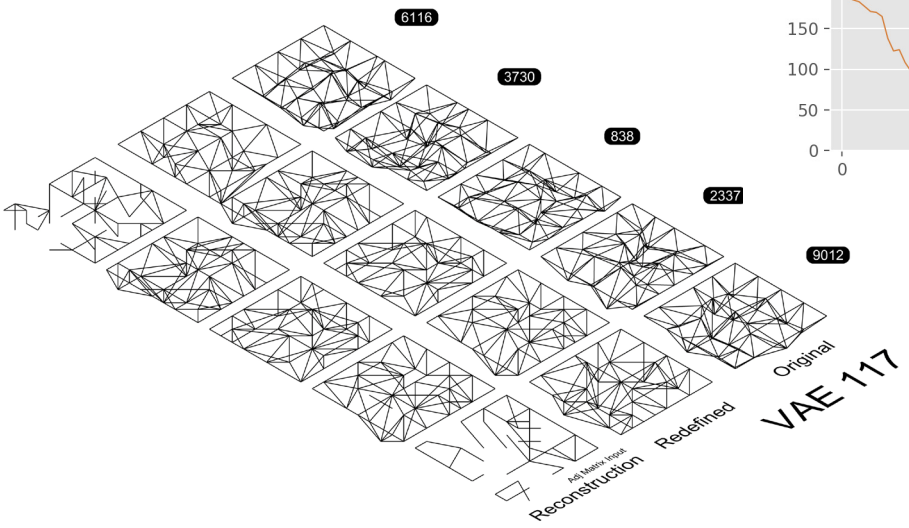


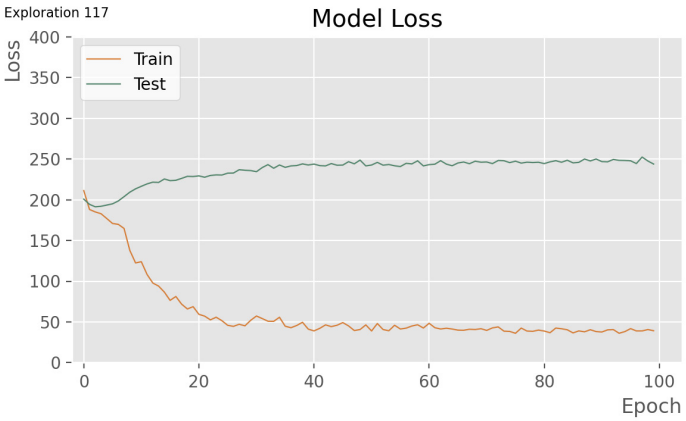


The attributes of this model are:

Latent Dimension	13
Multiplication of KL_loss	**10**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 117

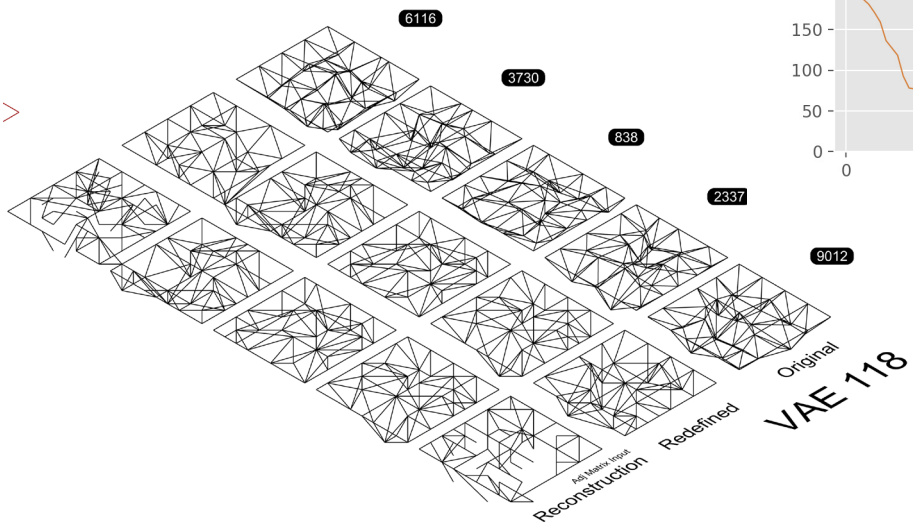


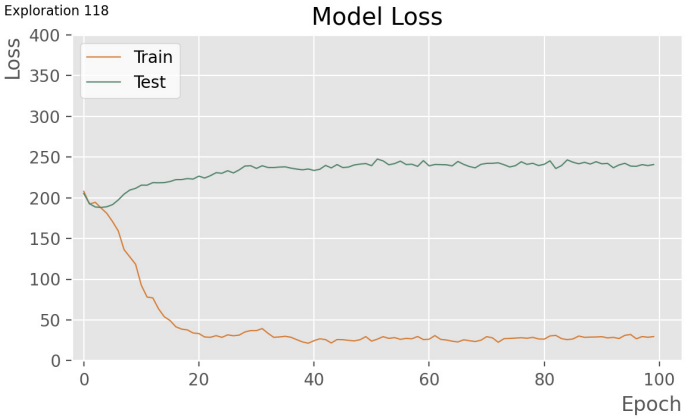


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	1
Architecture:	32-64 (filters)

VAE MODEL: EXPLORATION 118

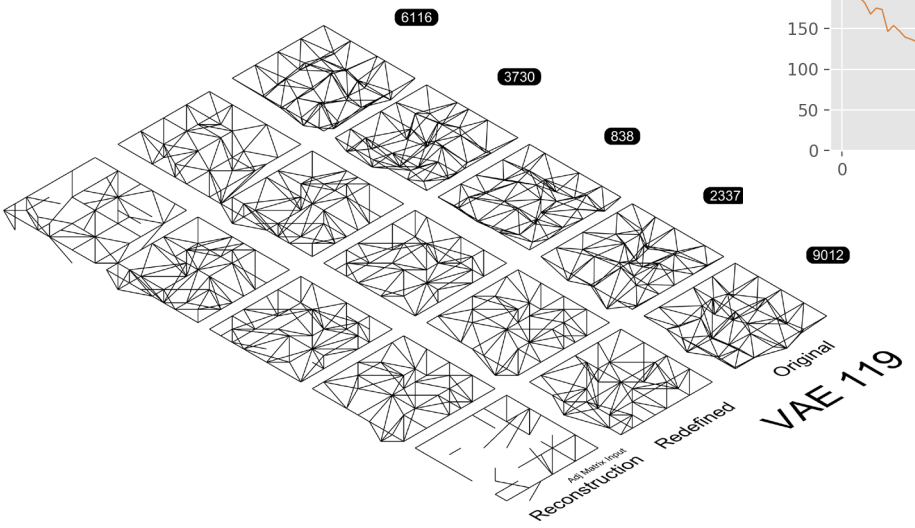




The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	1
Architecture:	**64-128 (filters)**

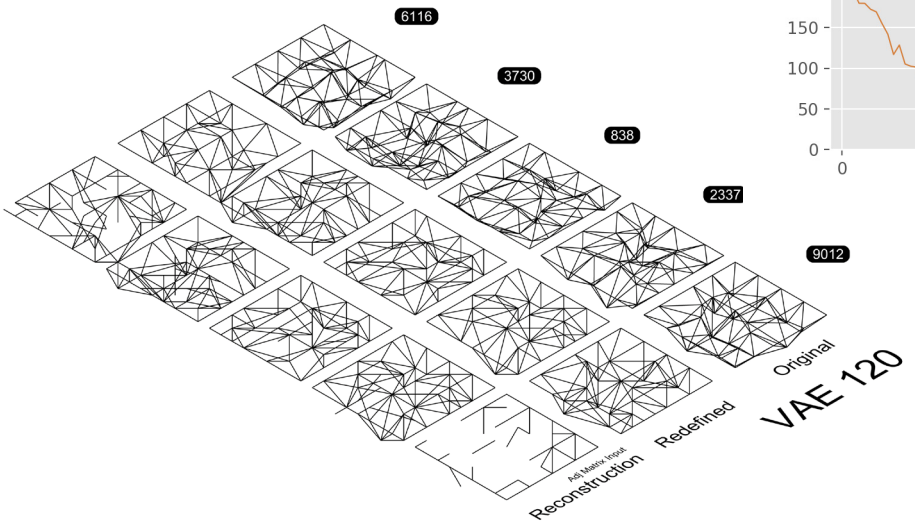
VAE MODEL: EXPLORATION 119

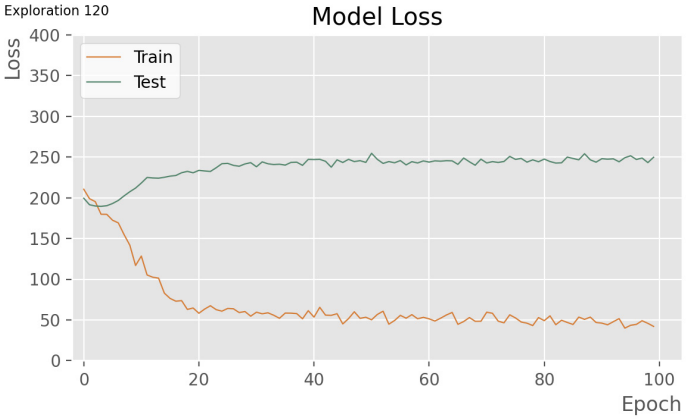


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	1
Architecture:	**16-32 (filters)**

VAE MODEL: EXPLORATION 120

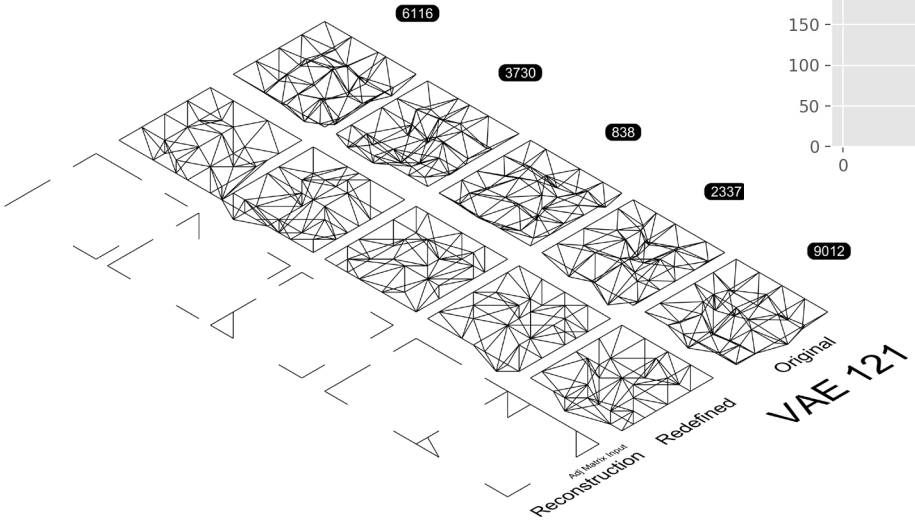


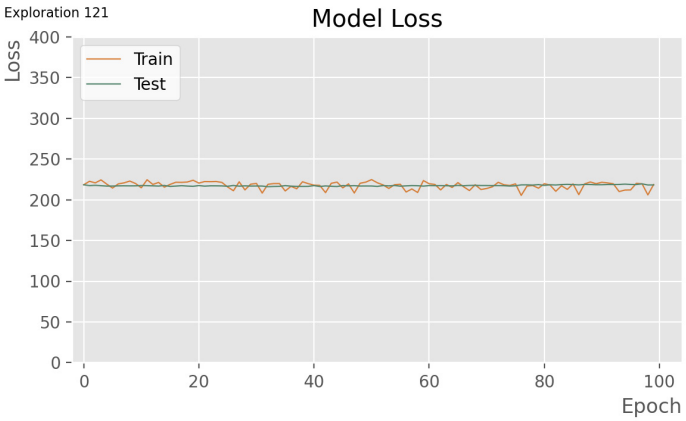


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	1
Architecture:	**16-32-64 (filters)**

VAE MODEL: EXPLORATION 121

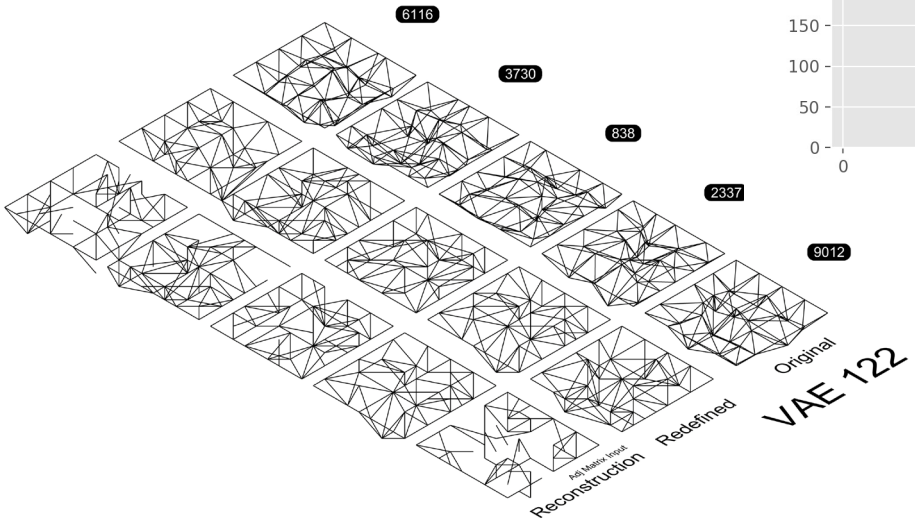


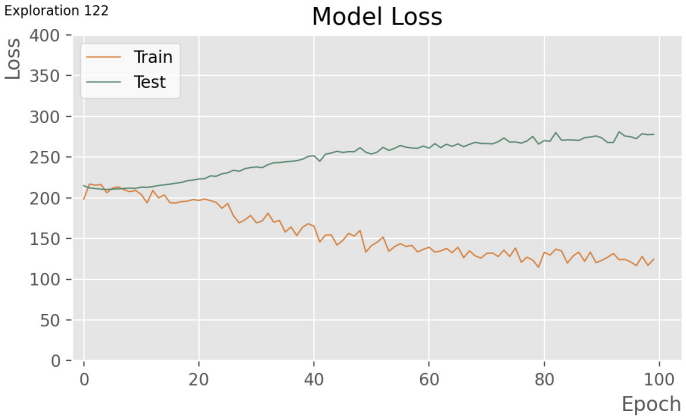


The attributes of this model are:

Latent Dimension	**2**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 122

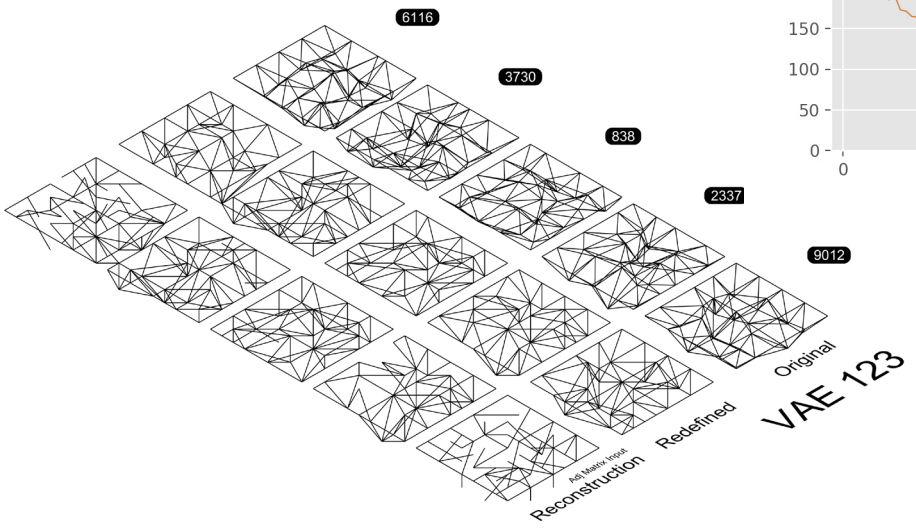




The attributes of this model are:

Latent Dimension	**7**
Multiplication of KL_loss	1
Architecture:	'972-243'

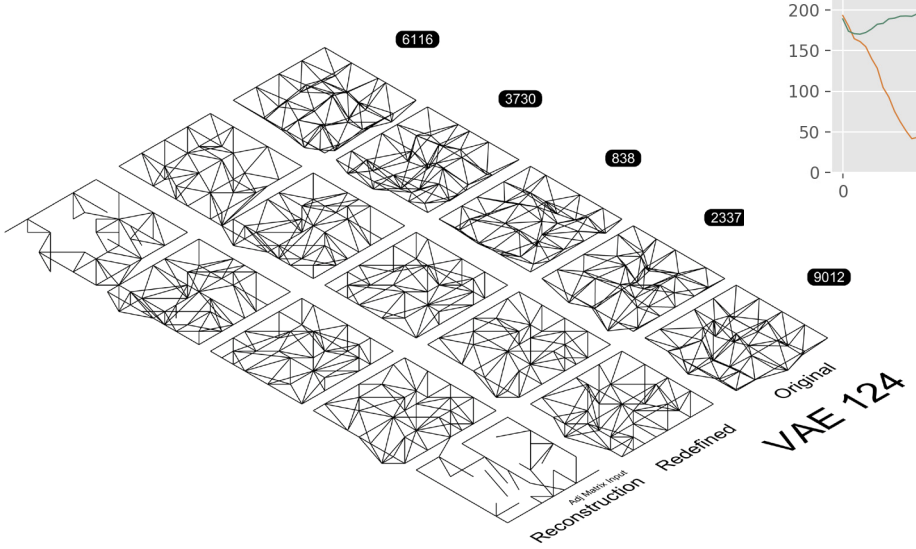
VAE MODEL: EXPLORATION 123

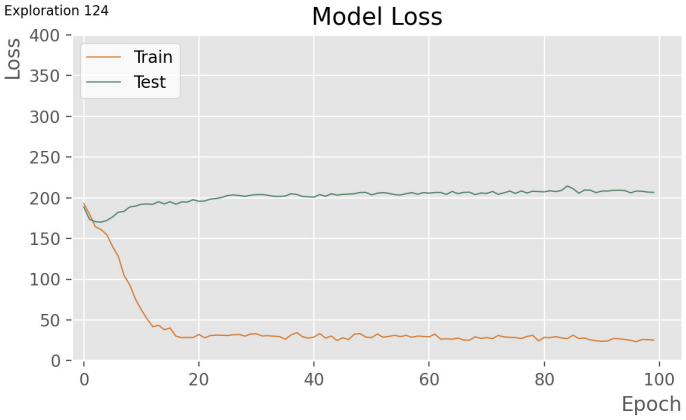


The attributes of this model are:

Latent Dimension	**13**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 124

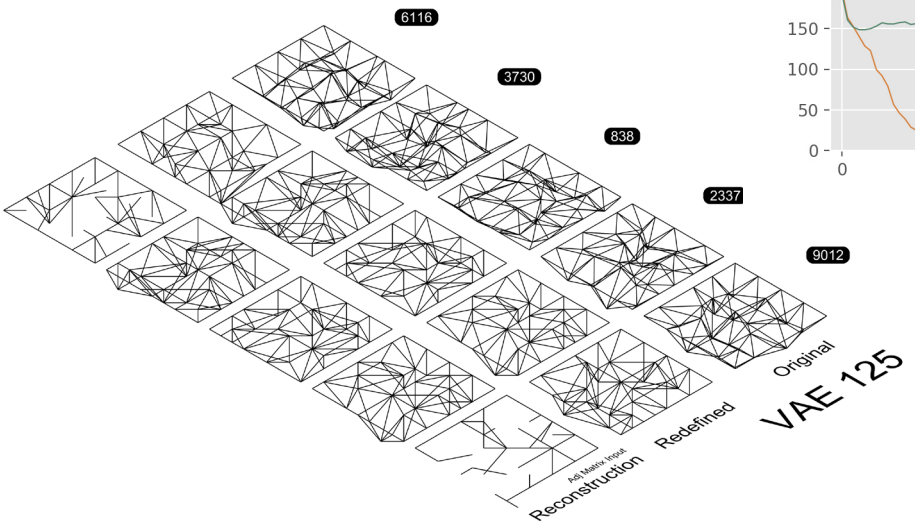


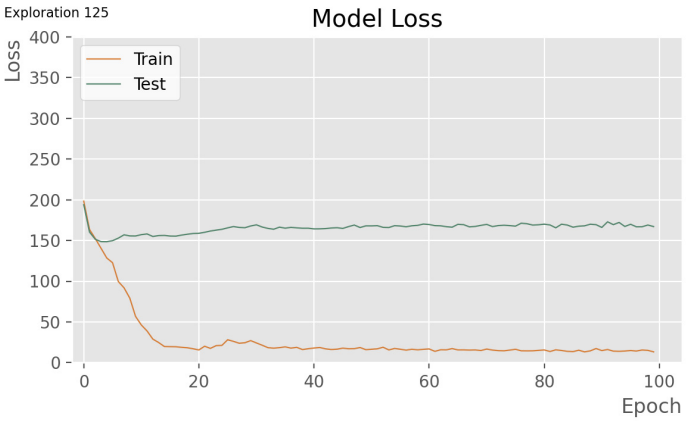


The attributes of this model are:

Latent Dimension	**30**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 125

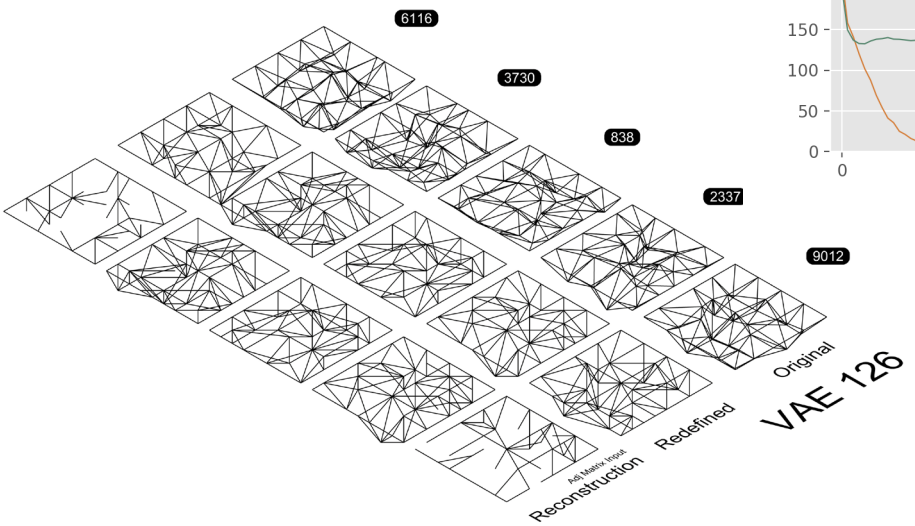


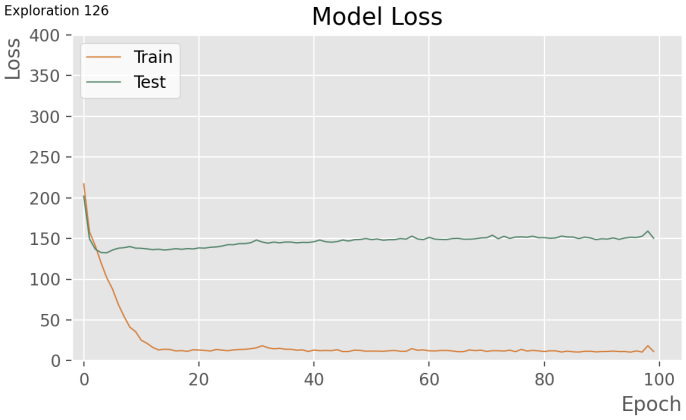


The attributes of this model are:

Latent Dimension	**40**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 126

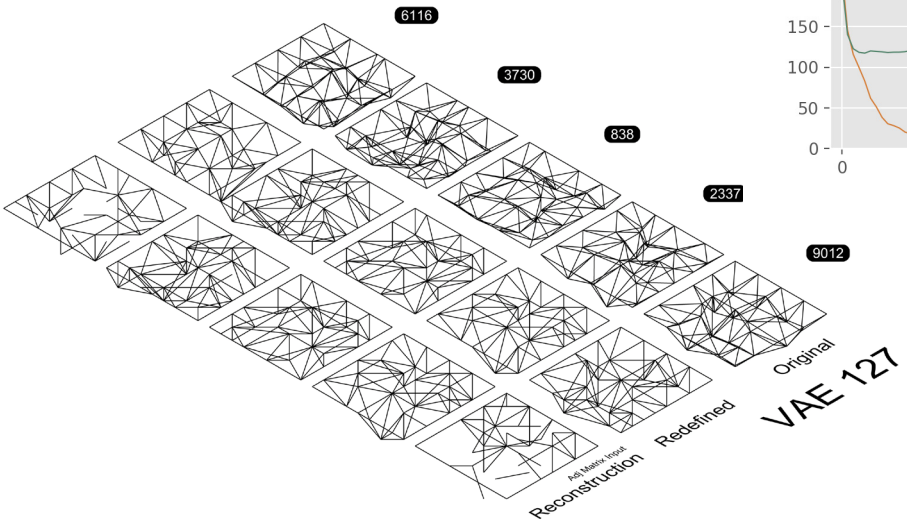


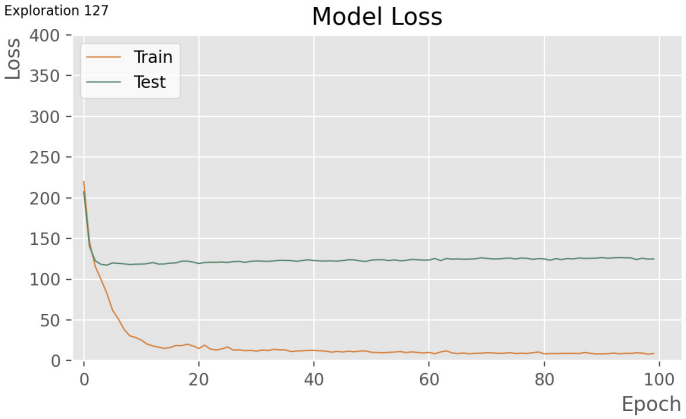


The attributes of this model are:

Latent Dimension	**50**
Multiplication of KL_loss	1
Architecture:	'972-243'

VAE MODEL: EXPLORATION 127

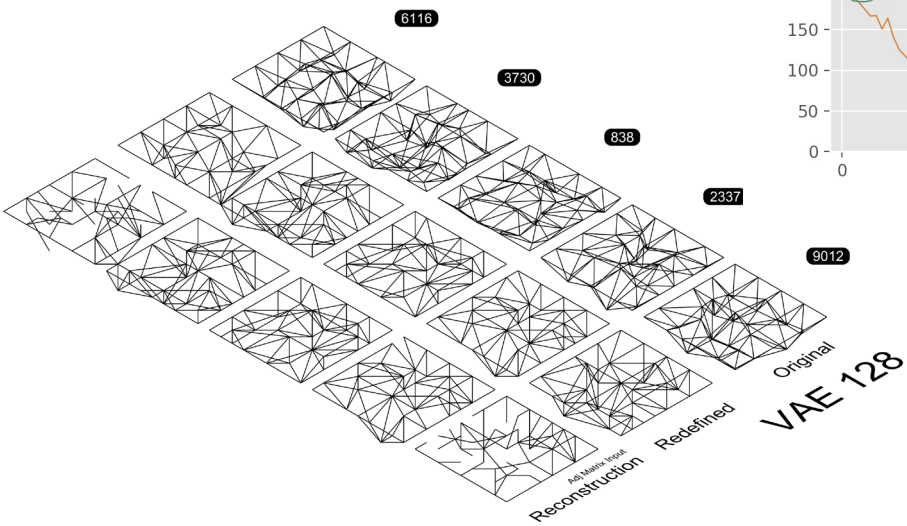




The attributes of this model are:

Latent Dimension	**60**
Multiplication of KL_loss	1
Architecture:	'972-243'

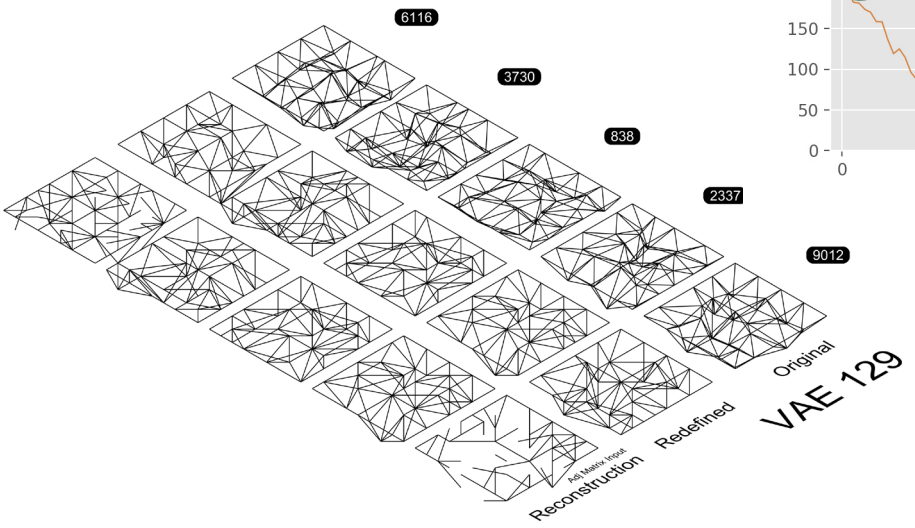
VAE MODEL: EXPLORATION 128



The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**0.01**
Architecture:	'972-243'

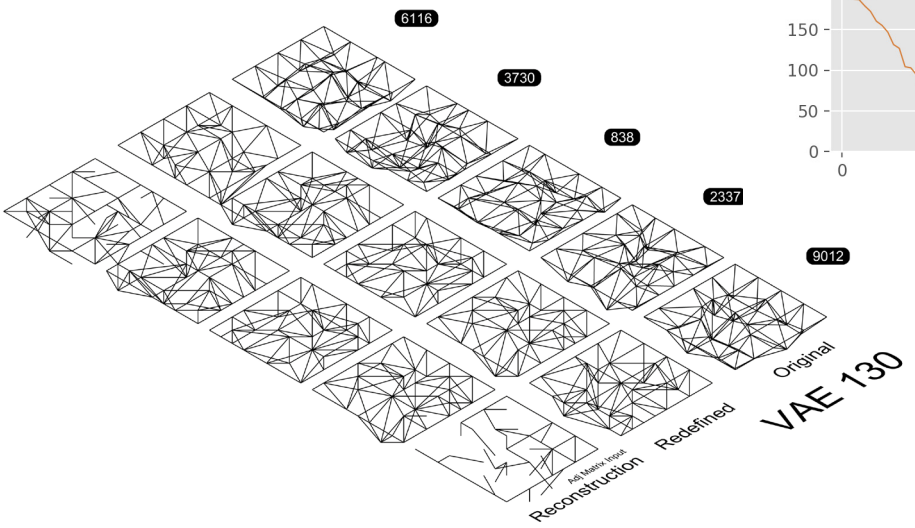
VAE MODEL: EXPLORATION 129

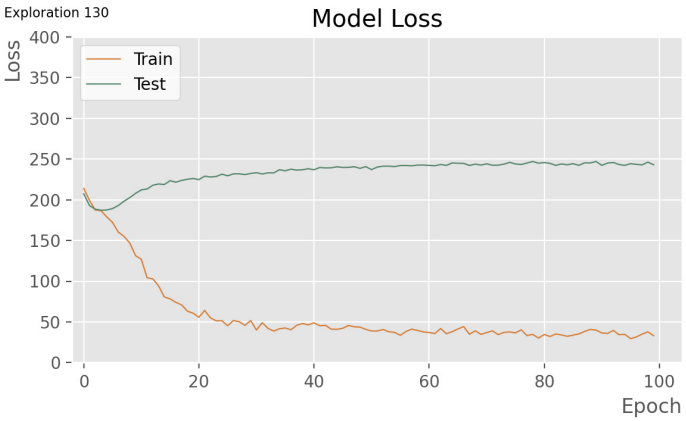


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**0.1**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 130

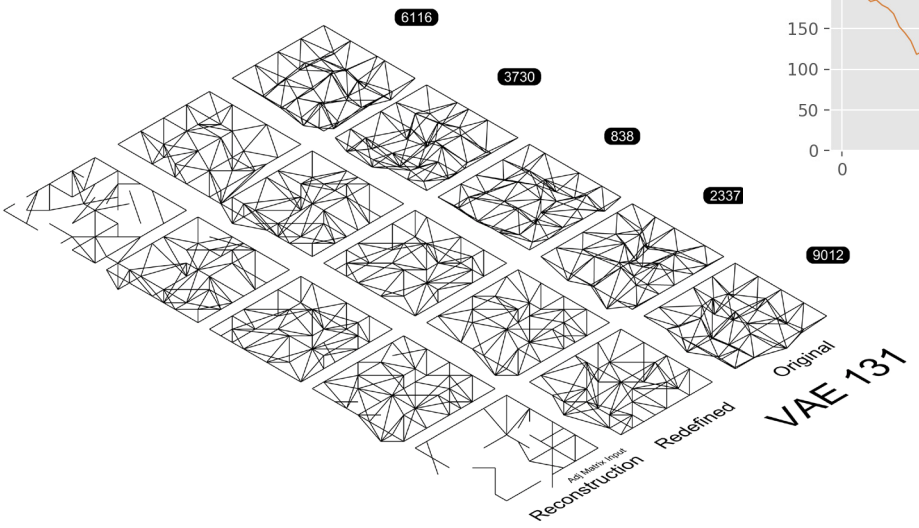




The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**0.5**
Architecture:	'972-243'

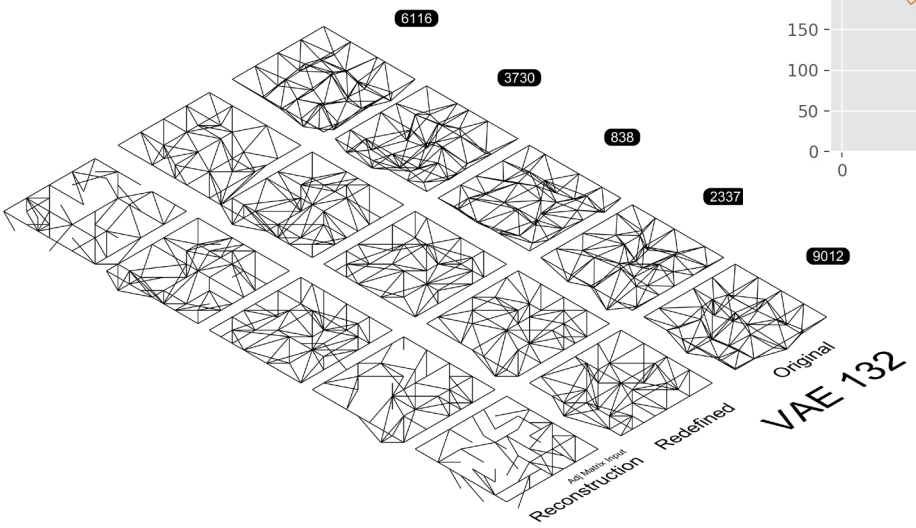
VAE MODEL: EXPLORATION 131



The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**5**
Architecture:	'972-243'

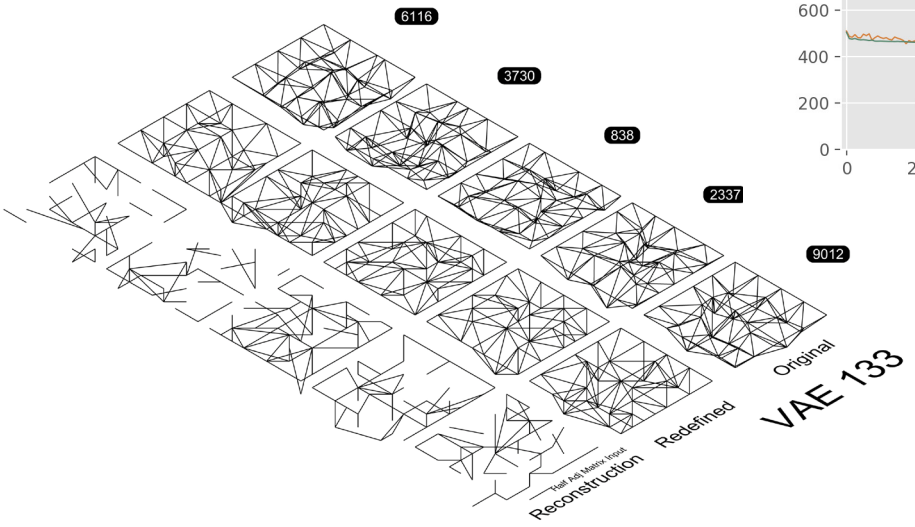
VAE MODEL: EXPLORATION 132

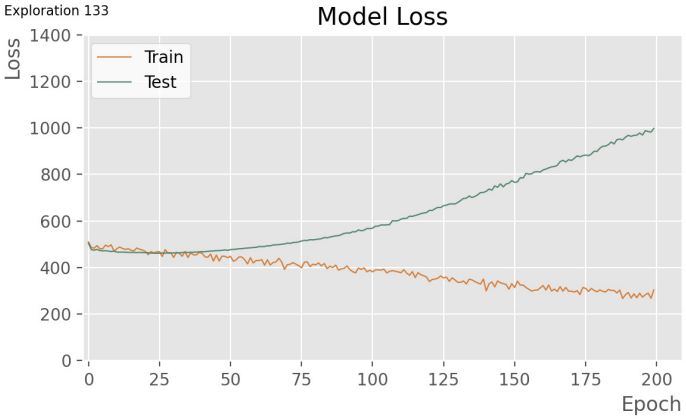


The attributes of this model are:

Latent Dimension	20
Multiplication of KL_loss	**10**
Architecture:	'972-243'

VAE MODEL: EXPLORATION 133

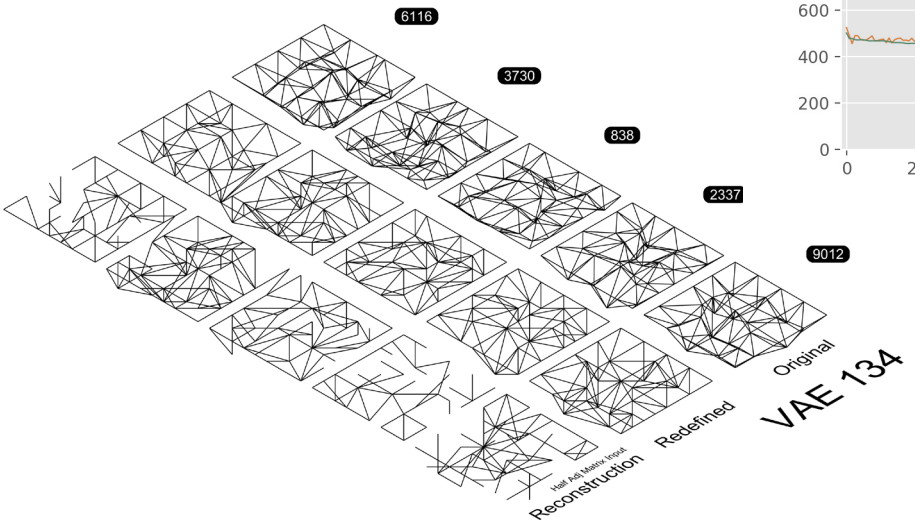


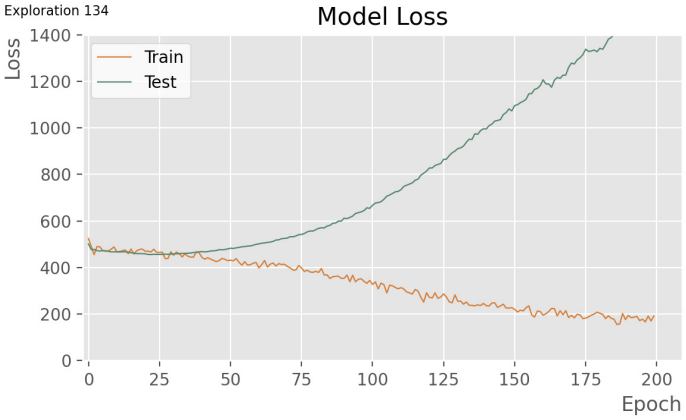


The attributes of this model are:

Latent Dimension	**3**
Multiplication of KL_loss	0.3
Architecture:	'1089-363-121'

VAE MODEL: EXPLORATION 134

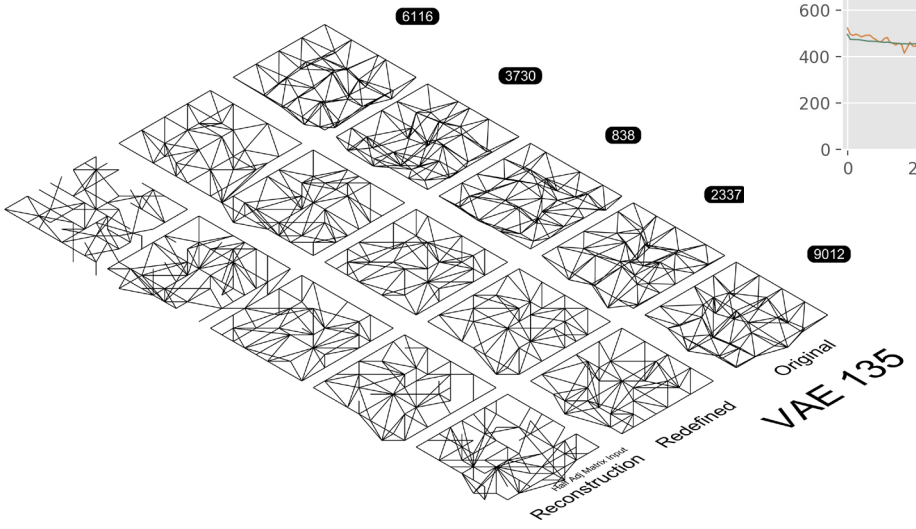


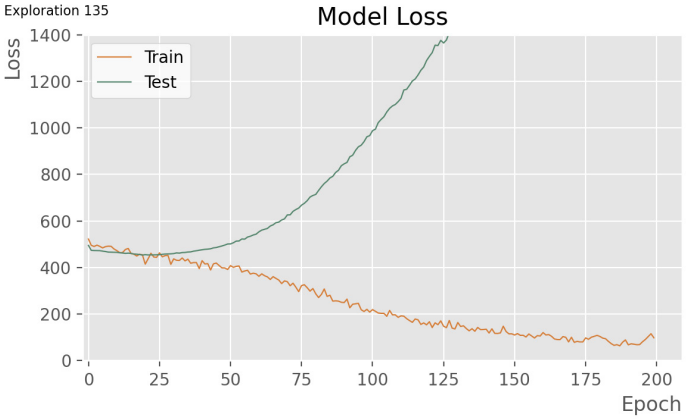


The attributes of this model are:

Latent Dimension	**4**
Multiplication of KL_loss	0.3
Architecture:	'1089-363-121'

VAE MODEL: EXPLORATION 135

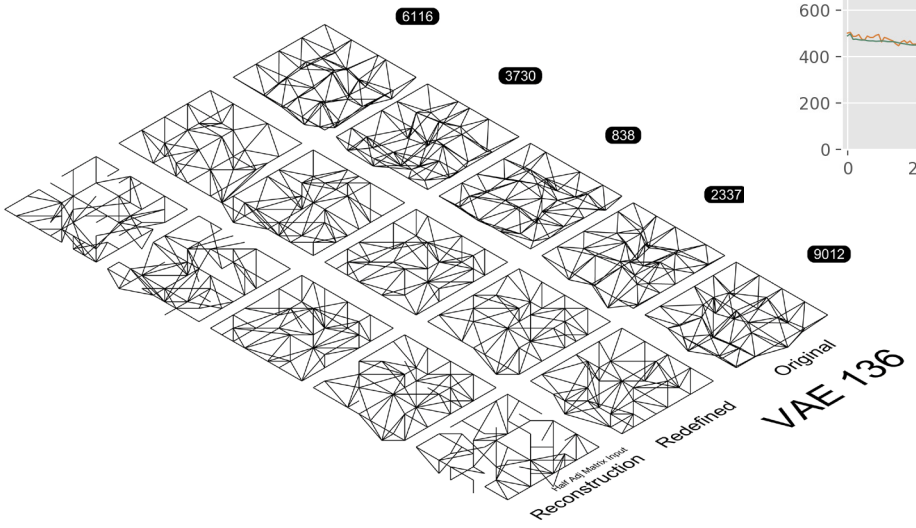


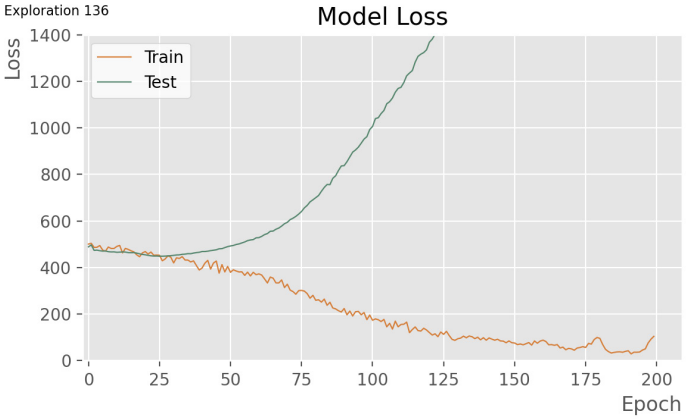


The attributes of this model are:

Latent Dimension	**5**
Multiplication of KL_loss	0.3
Architecture:	'1089-363-121'

VAE MODEL: EXPLORATION 136

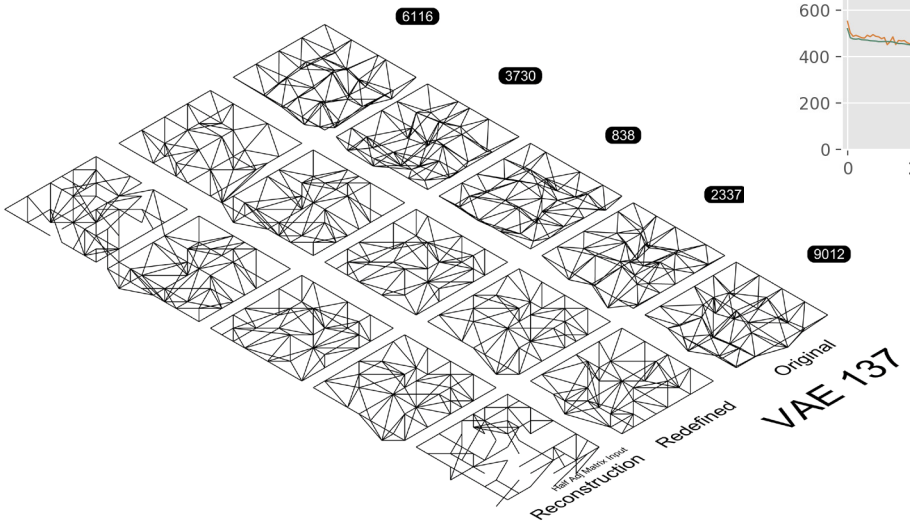


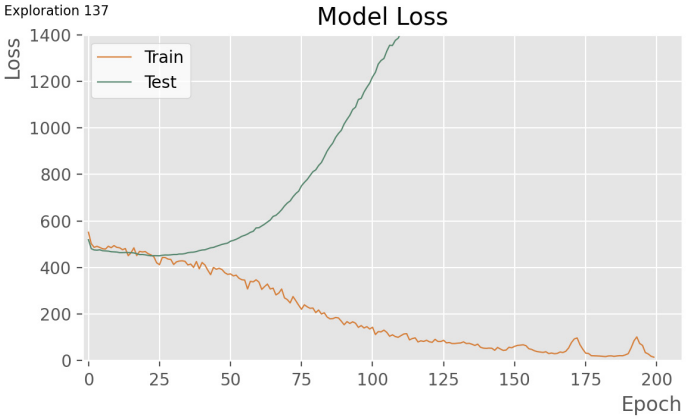


The attributes of this model are:

Latent Dimension	**6**
Multiplication of KL_loss	0.3
Architecture:	'1089-363-121'

VAE MODEL: EXPLORATION 137

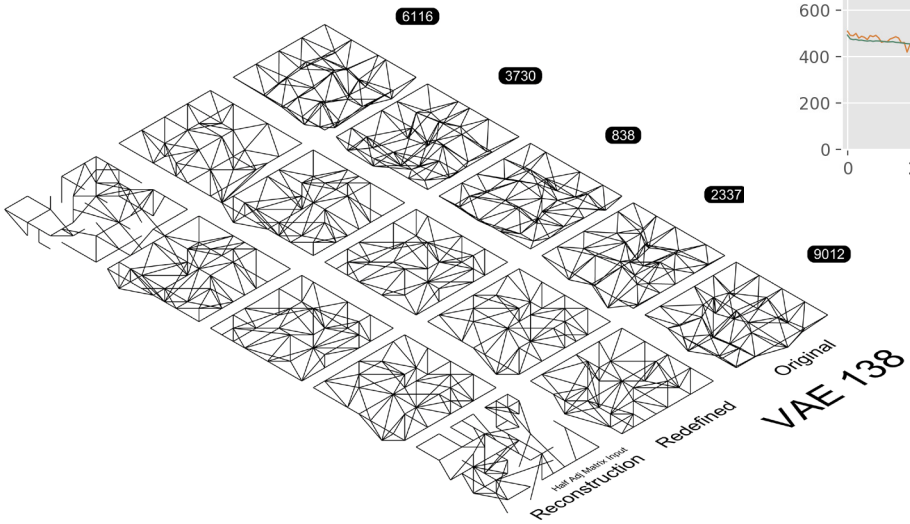


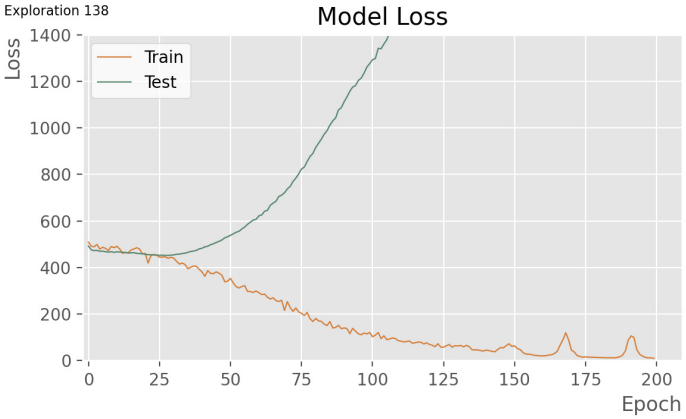


The attributes of this model are:

Latent Dimension	**7**
Multiplication of KL_loss	0.3
Architecture:	'1089-363-121'

VAE MODEL: EXPLORATION 138

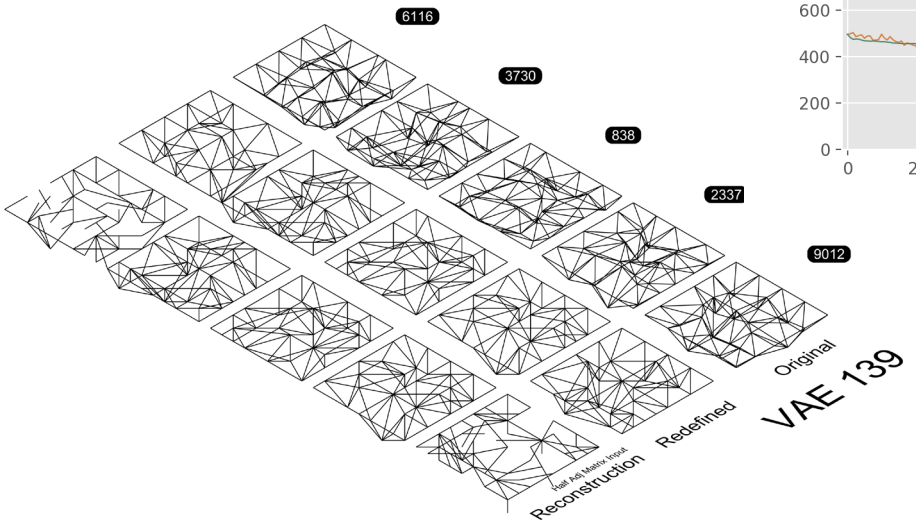


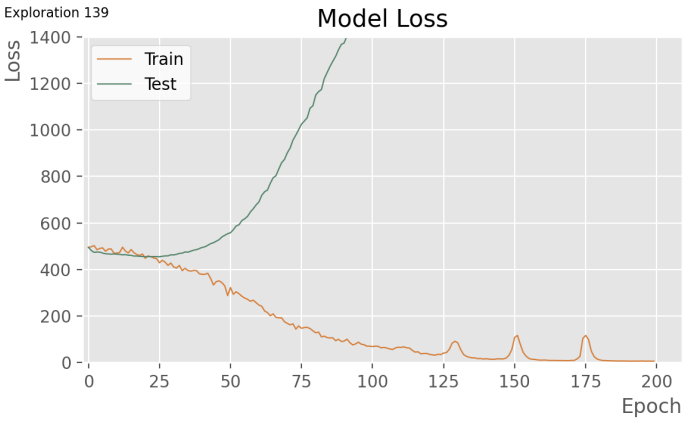


The attributes of this model are:

Latent Dimension	**8**
Multiplication of KL_loss	0.3
Architecture:	'1089-363-121'

VAE MODEL: EXPLORATION 139





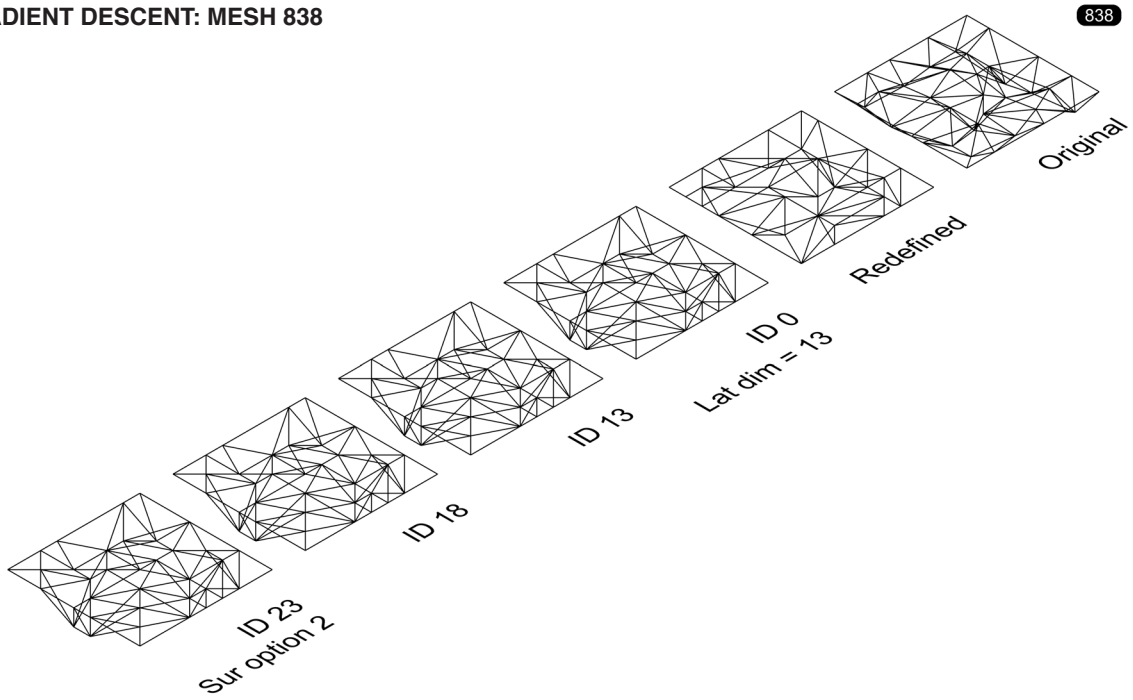
The attributes of this model are:

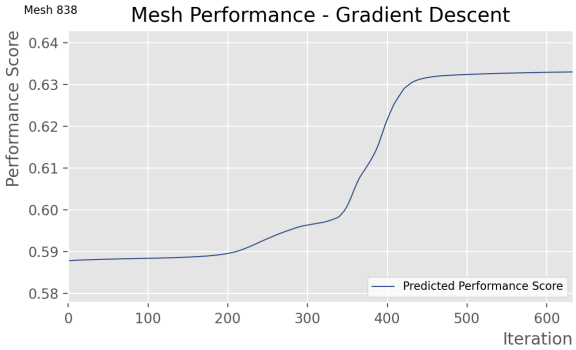
Latent Dimension	**9**
Multiplication of KL_loss	0.3
Architecture:	'1089-363-121'

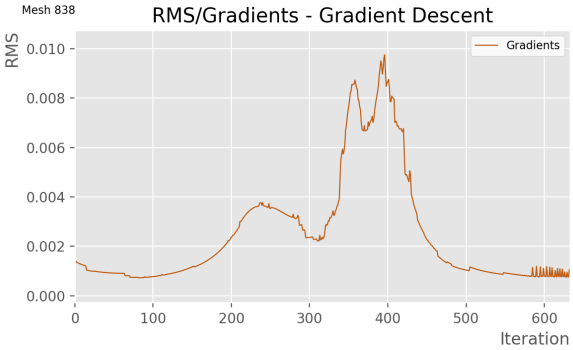
Appendix VI

Gradient descent results

GRADIENT DESCENT: MESH 838

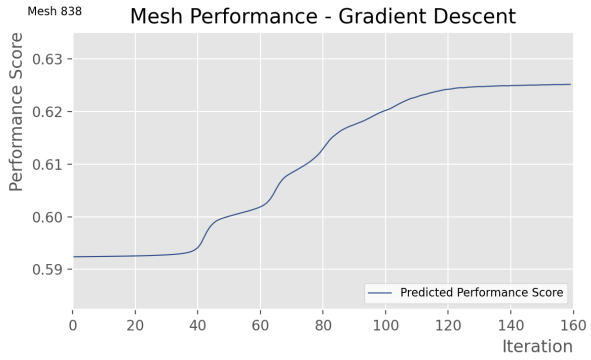


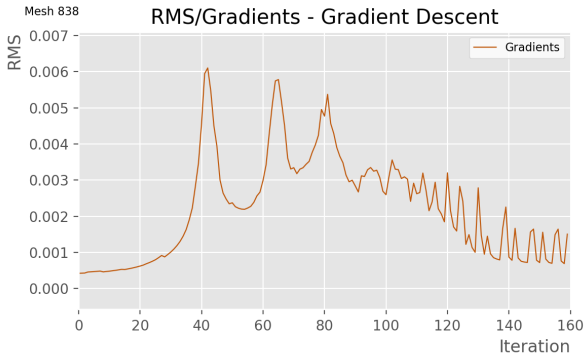




Model ID: 0
Iterations: 633

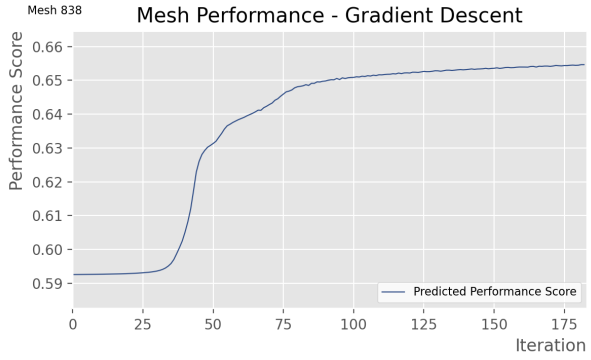
Predicted Performance was: 0.5878
Predicted Performance is now: 0.63294
Predicted performance was increased by: 0.04516 (7.68%)

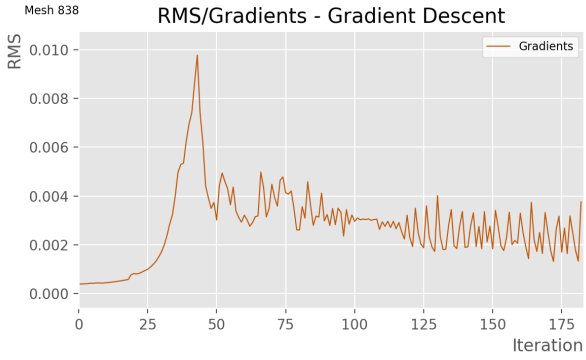




Model ID: 13
Iterations: 160

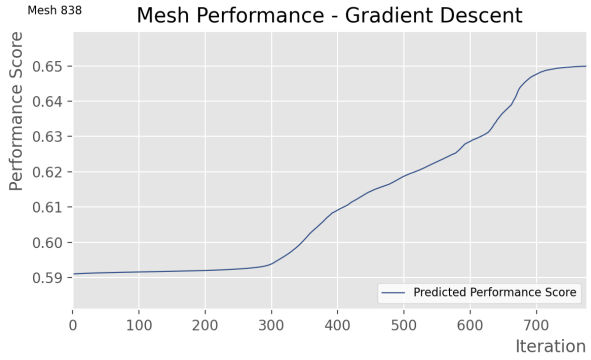
Predicted Performance was: 0.59239
Predicted Performance is now: 0.62506
Predicted performance was increased by: 0.03276 (5.53%)

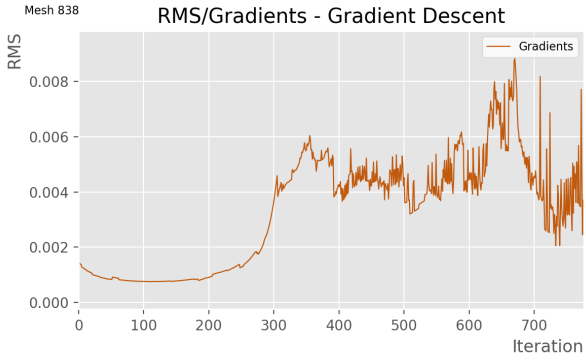




Model ID: 18
Iterations: 183

Predicted Performance was: 0.59256
Predicted Performance is now: 0.65434
Predicted performance was increased by: 0.06209 (10.48%)

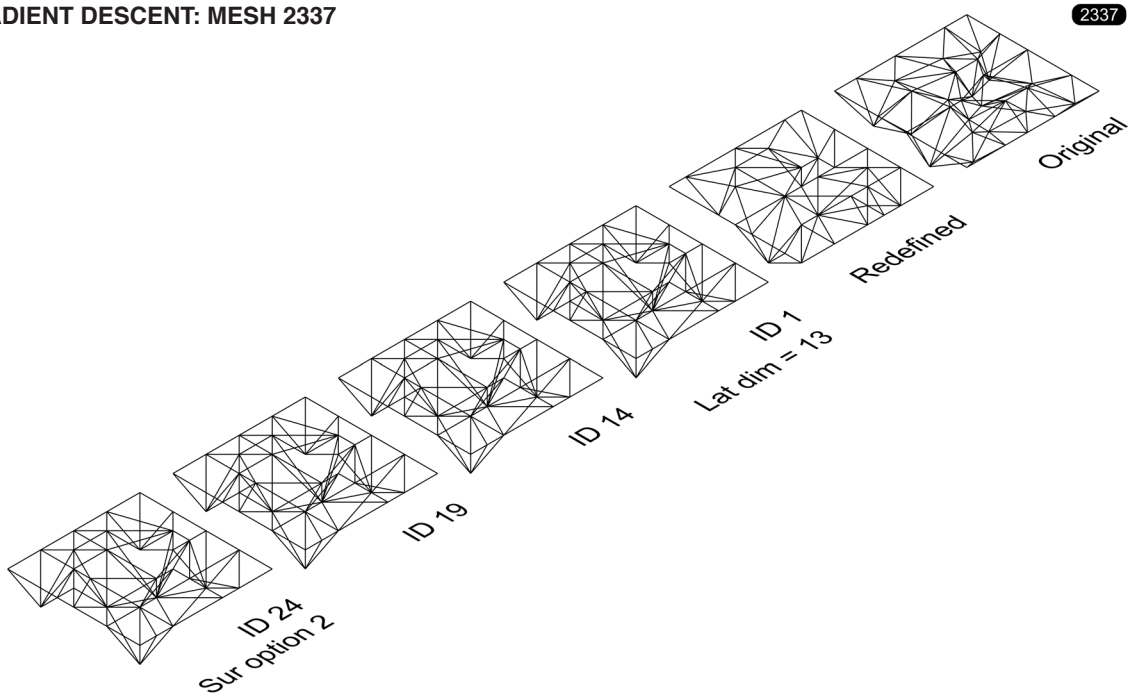


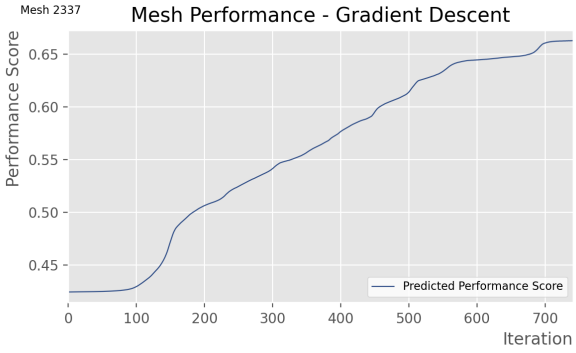


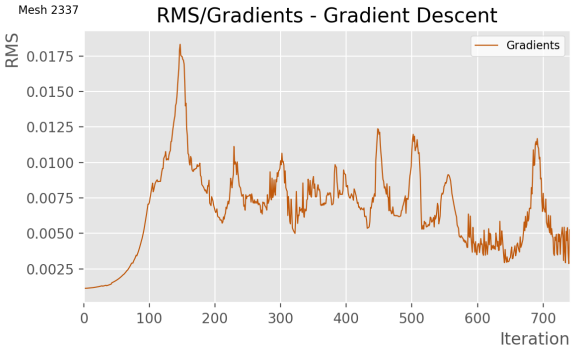
Model ID: 23
Iterations: 776

Predicted Performance was: 0.59098
Predicted Performance is now: 0.64985
Predicted performance was increased by: 0.05897 (9.98%)

GRADIENT DESCENT: MESH 2337

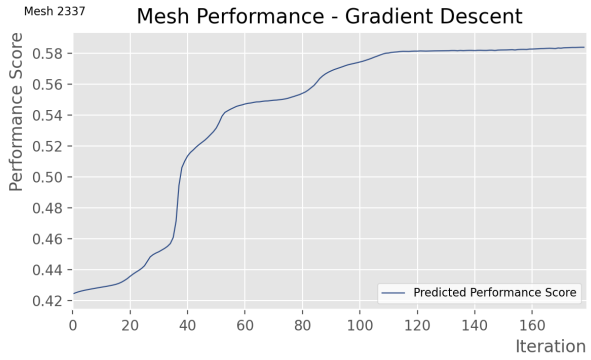


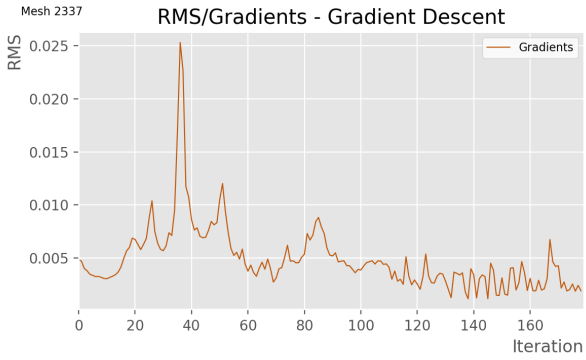




Model ID: 1
Iterations: 741

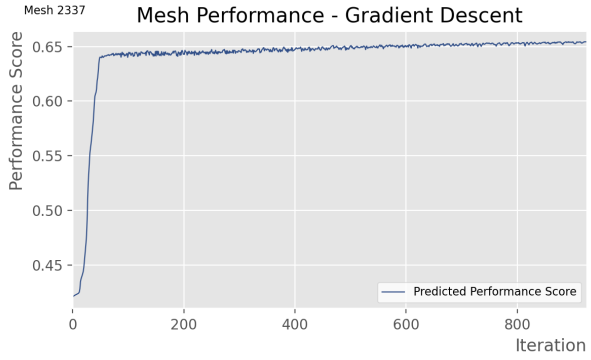
Predicted Performance was: 0.4245
Predicted Performance is now: 0.66246
Predicted performance was increased by: 0.23809 (56.09%)

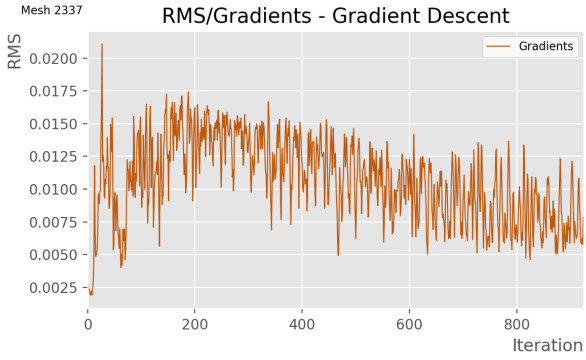




Model ID: 14
Iterations: 179

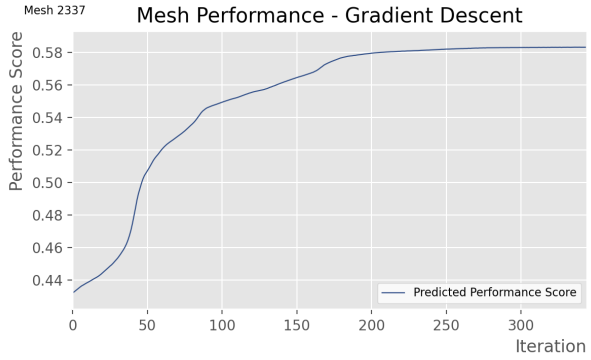
Predicted Performance was: 0.42422
Predicted Performance is now: 0.58352
Predicted performance was increased by: 0.15979 (37.67%)

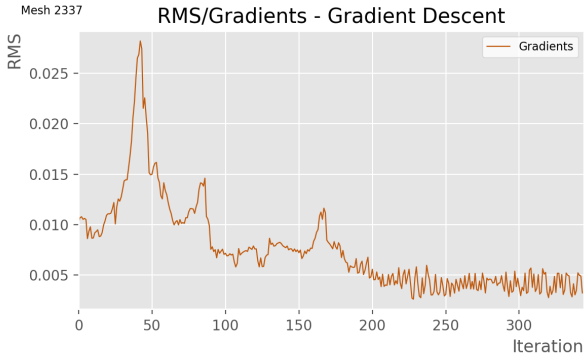




Model ID: 19
Iterations: 925

Predicted Performance was: 0.42025
Predicted Performance is now: 0.65326
Predicted performance was increased by: 0.23406 (55.69%)

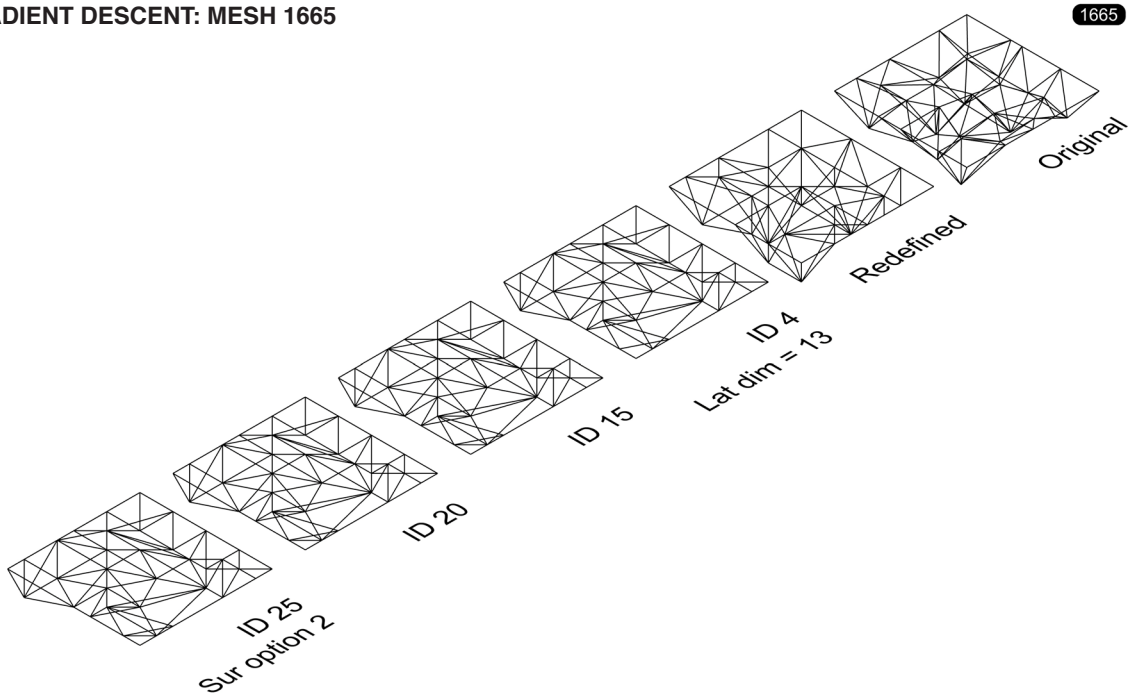


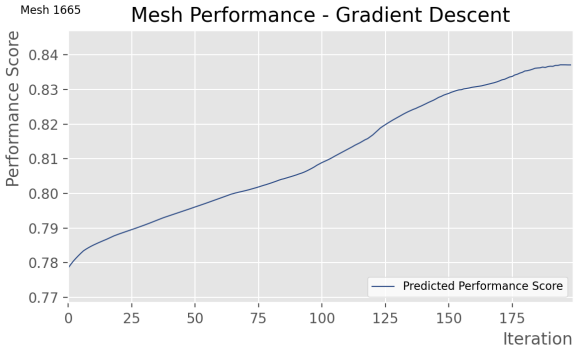


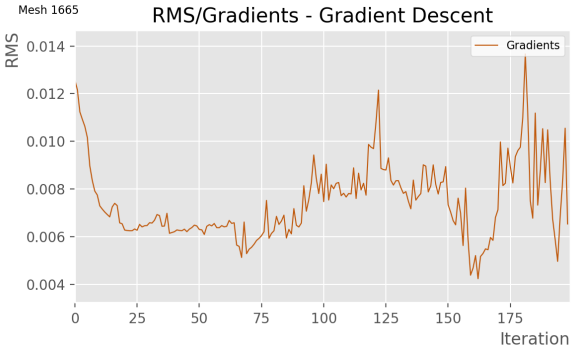
Model ID: 24
Iterations: 344

Predicted Performance was: 0.43217
Predicted Performance is now: 0.58309
Predicted performance was increased by: 0.15094 (34.92%)

GRADIENT DESCENT: MESH 1665

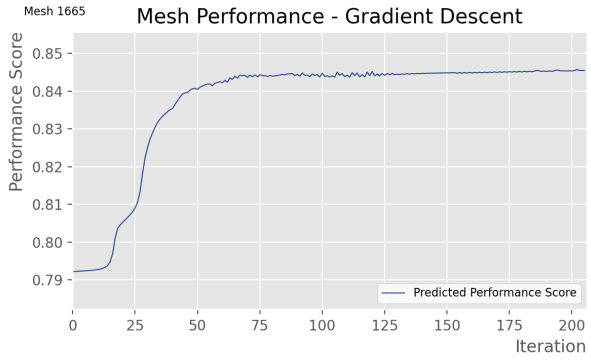


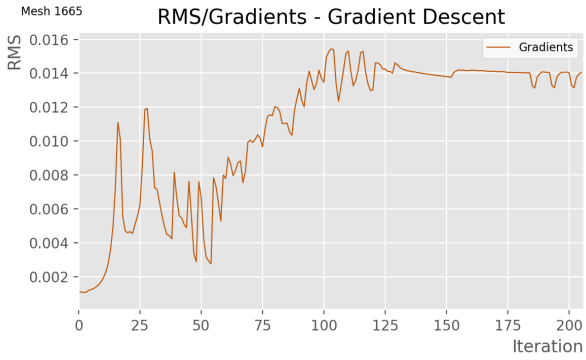




Model ID: 4
Iterations: 199

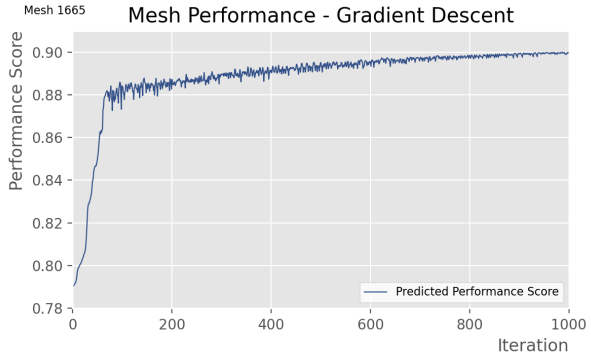
Predicted Performance was: 0.77849
Predicted Performance is now: 0.83657
Predicted performance was increased by: 0.05859 (7.53%)

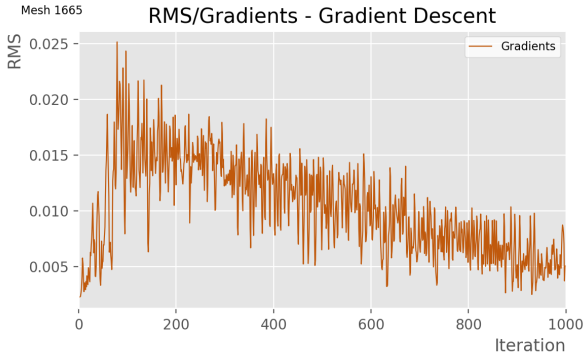




Model ID: 15
Iterations: 206

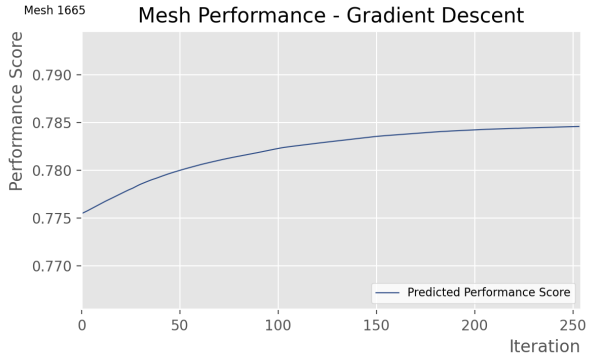
Predicted Performance was: 0.79222
Predicted Performance is now: 0.84534
Predicted performance was increased by: 0.05322 (6.72%)

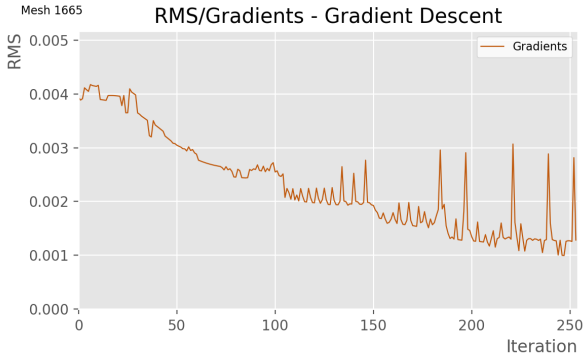




Model ID: 20
Iterations: 1000

Predicted Performance was: 0.78993
Predicted Performance is now: 0.89959
Predicted performance was increased by: 0.1097 (13.89%)

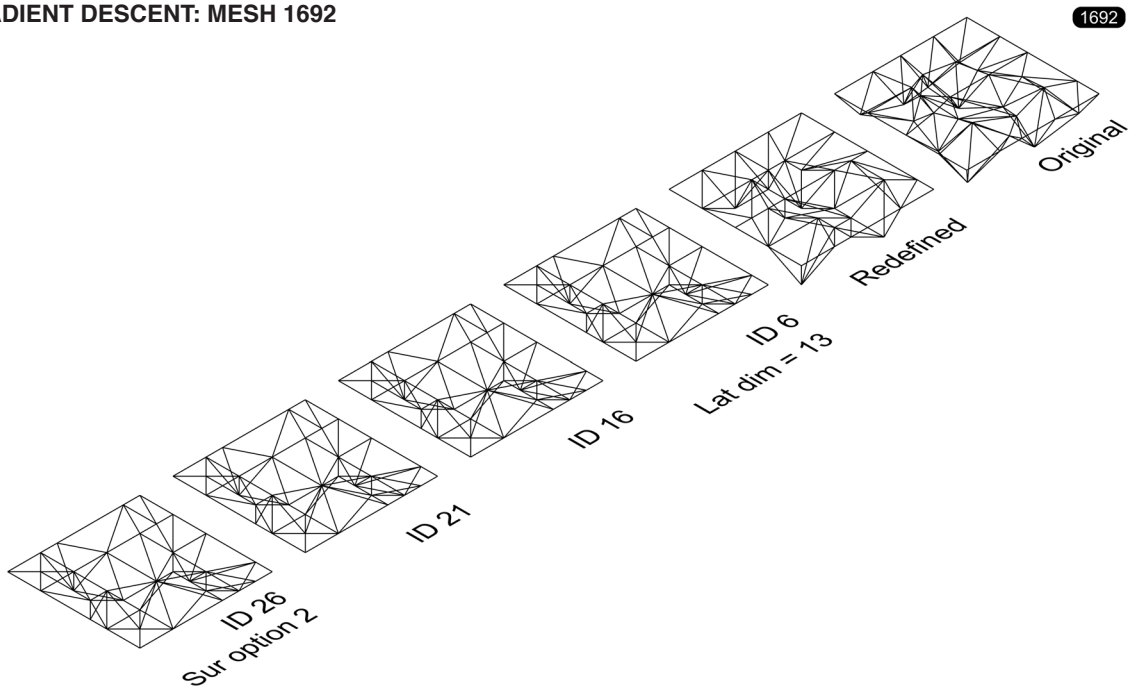


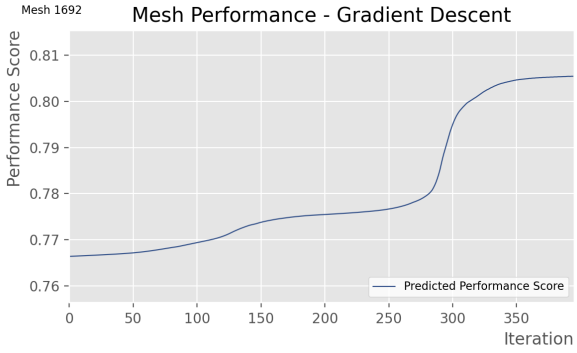


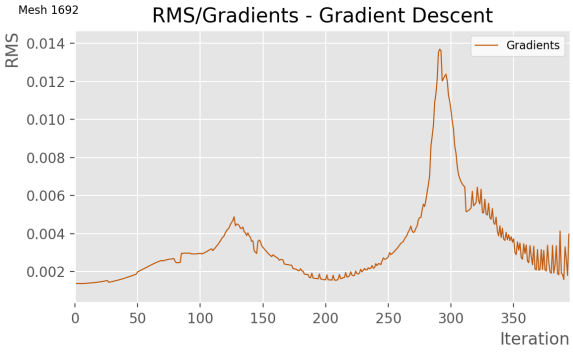
Model ID: 25
Iterations: 254

Predicted Performance was: 0.77548
Predicted Performance is now: 0.78454
Predicted performance was increased by: 0.00911 (1.17%)

GRADIENT DESCENT: MESH 1692

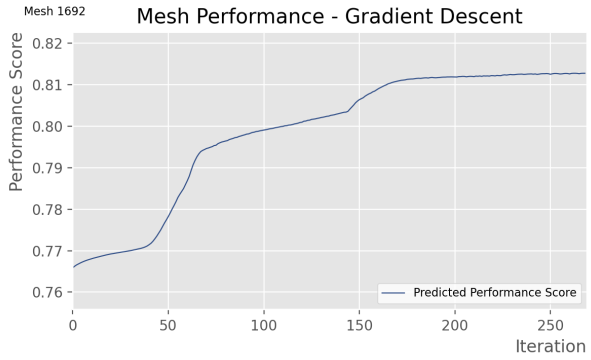


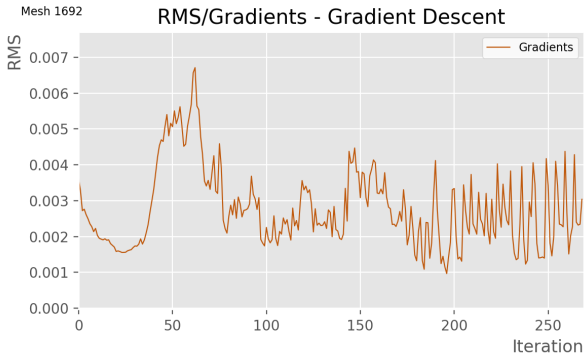




Model ID: 6
Iterations: 395

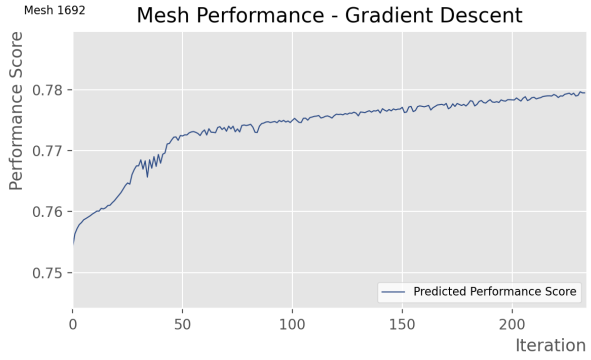
Predicted Performance was: 0.76639
Predicted Performance is now: 0.80533
Predicted performance was increased by: 0.03902 (5.09%)

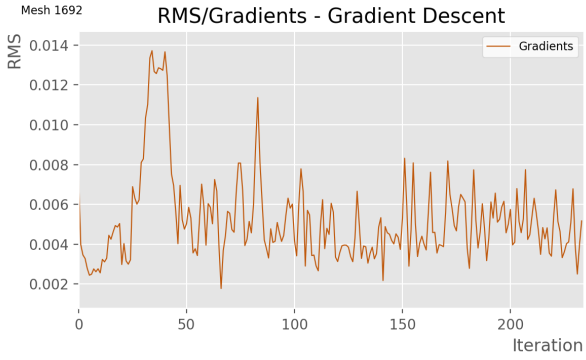




Model ID: 16
Iterations: 269

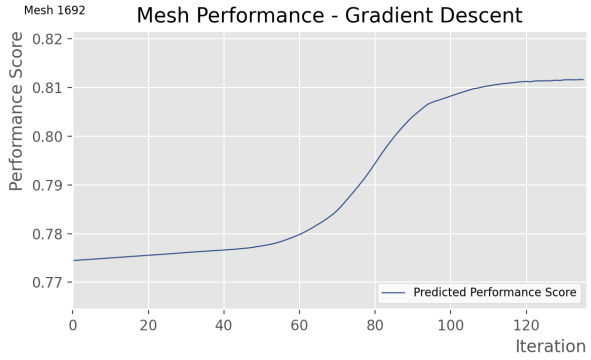
Predicted Performance was: 0.76584
Predicted Performance is now: 0.81272
Predicted performance was increased by: 0.04694 (6.13%)

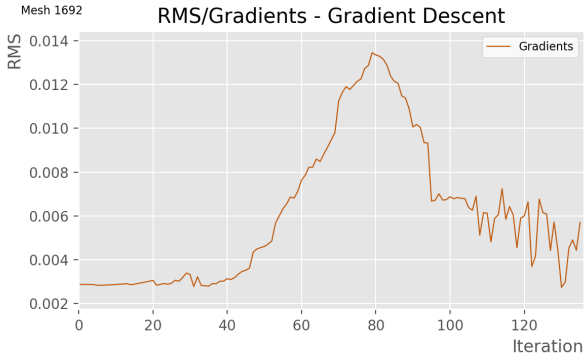




Model ID: 21
Iterations: 234

Predicted Performance was: 0.75413
Predicted Performance is now: 0.77925
Predicted performance was increased by: 0.02535 (3.36%)

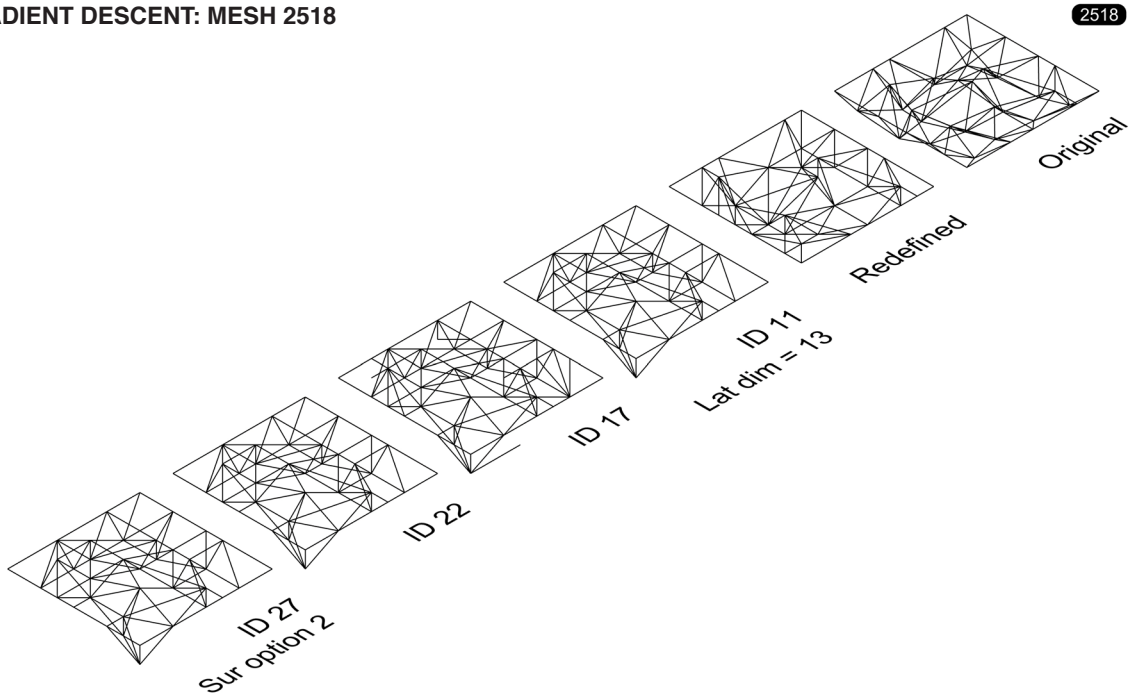


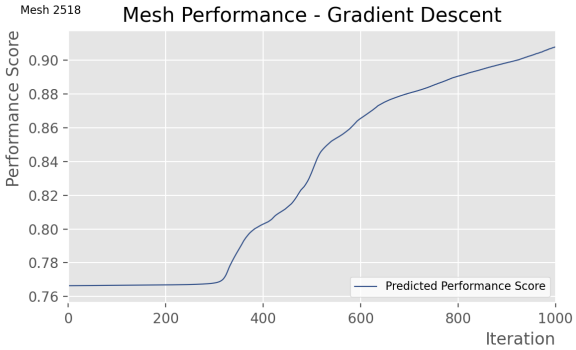


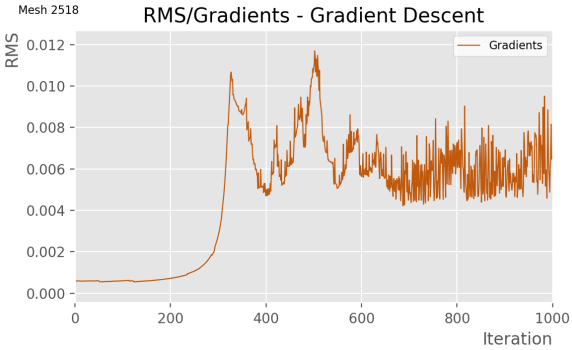
Model ID: 26
Iterations: 136

Predicted Performance was: 0.77446
Predicted Performance is now: 0.81139
Predicted performance was increased by: 0.03719 (4.8%)

GRADIENT DESCENT: MESH 2518

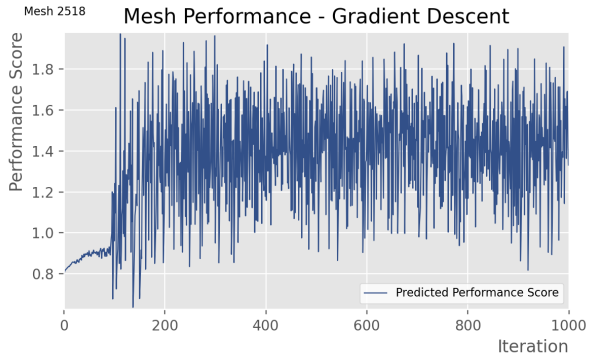


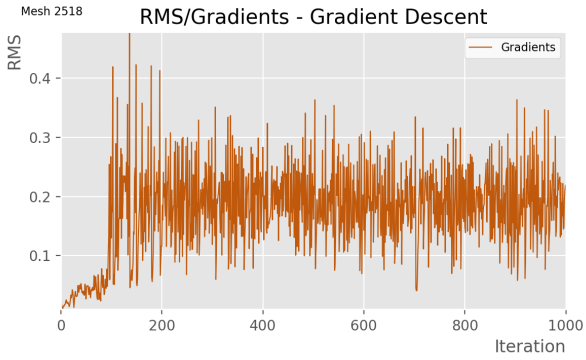




Model ID: 11
Iterations: 1000

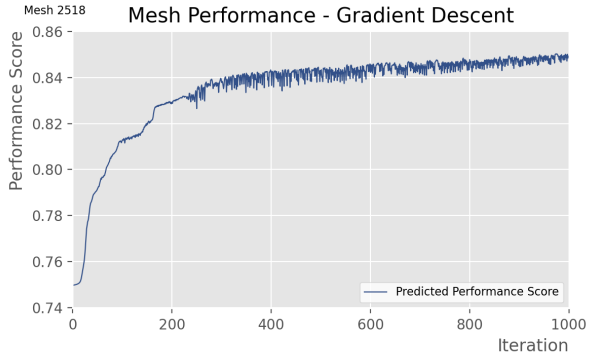
Predicted Performance was: 0.76638
Predicted Performance is now: 0.90697
Predicted performance was increased by: 0.14154 (18.47%)

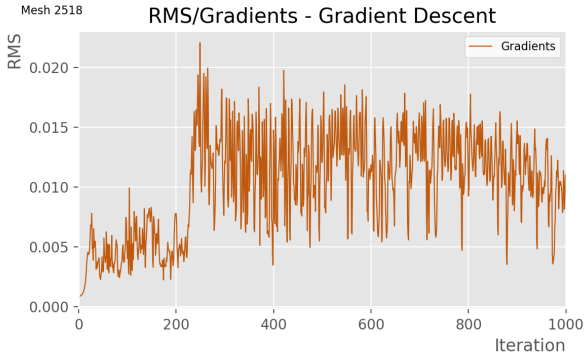




Model ID: 17
Iterations: 1000

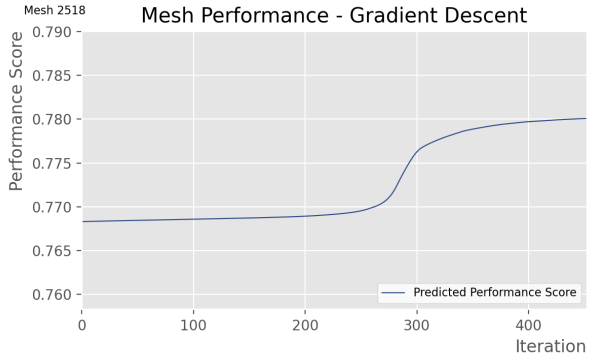
Predicted Performance was: 0.78553
Predicted Performance is now: 1.90782
Predicted performance was increased by: 0.54527 (69.41%)

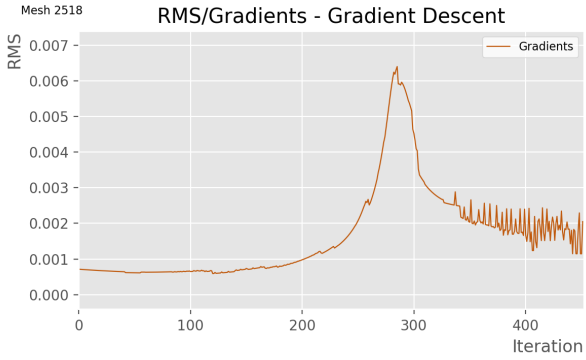




Model ID: 22
Iterations: 1000

Predicted Performance was: 0.74965
Predicted Performance is now: 0.84755
Predicted performance was increased by: 0.09953 (13.28%)



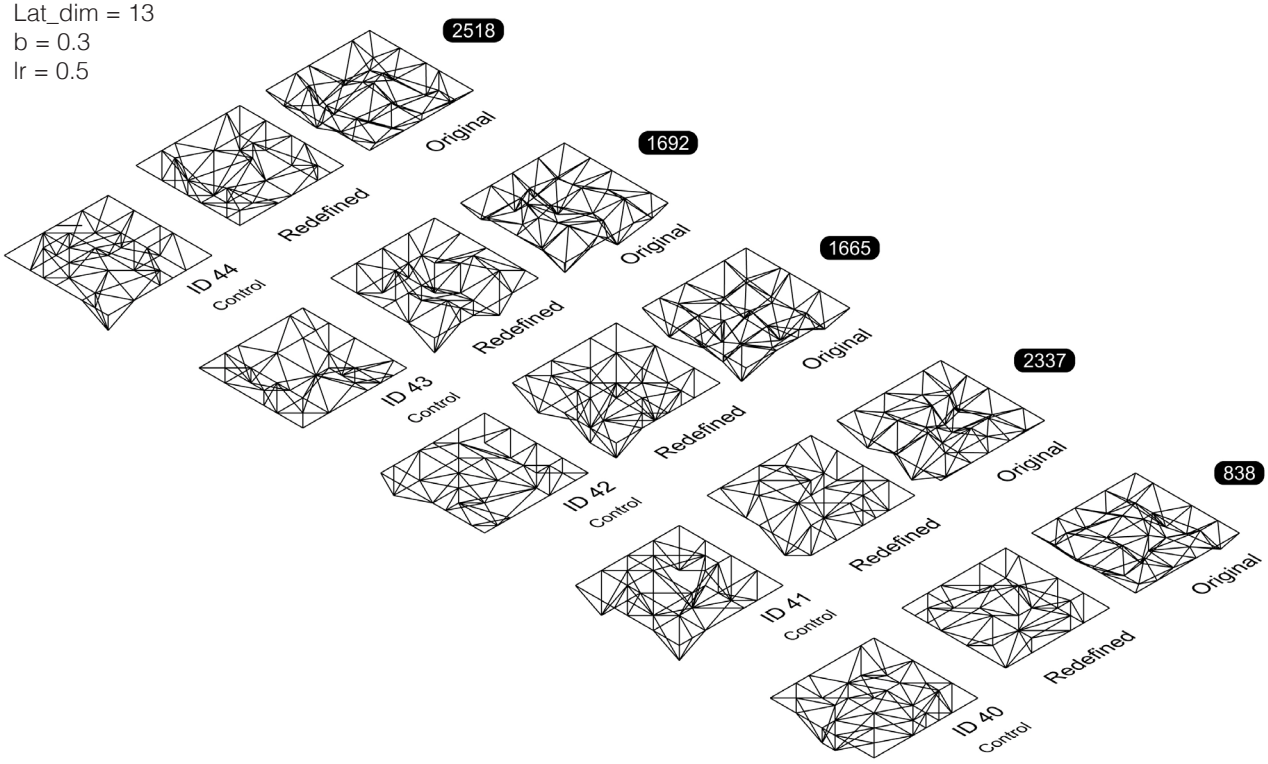


Model ID: 27
Iterations: 452

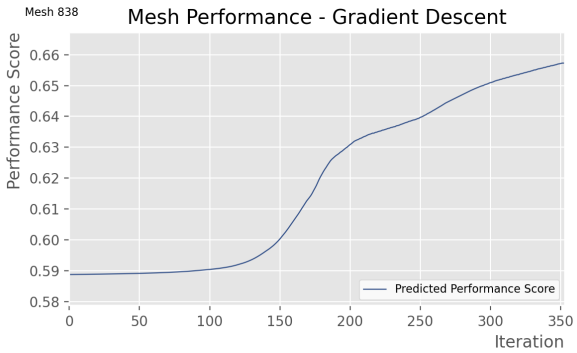
Predicted Performance was: 0.76831
Predicted Performance is now: 0.78003
Predicted performance was increased by: 0.01177 (1.53%)

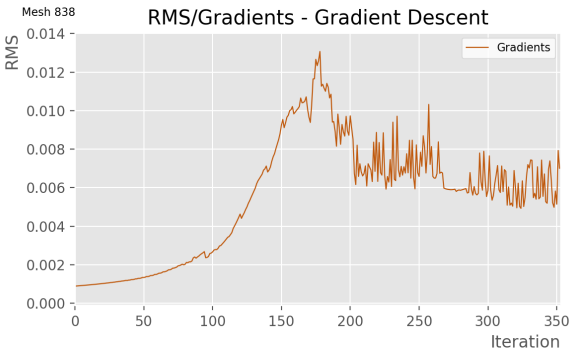
FURHTER GRADIENT DESCENT EXPLORATION: CONTROL GROUP

Lat_dim = 13
b = 0.3
lr = 0.5

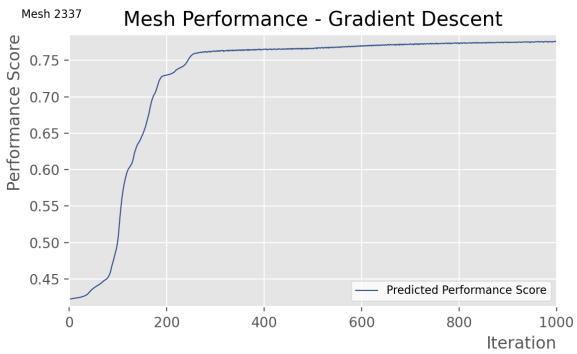


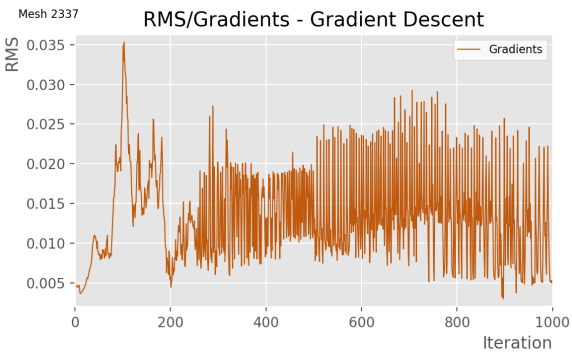
Mesh 838 - control



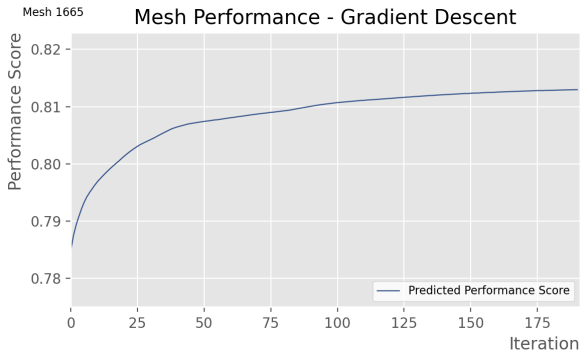


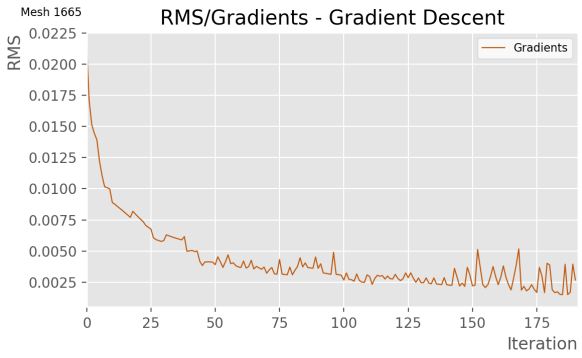
Mesh 2337 - control



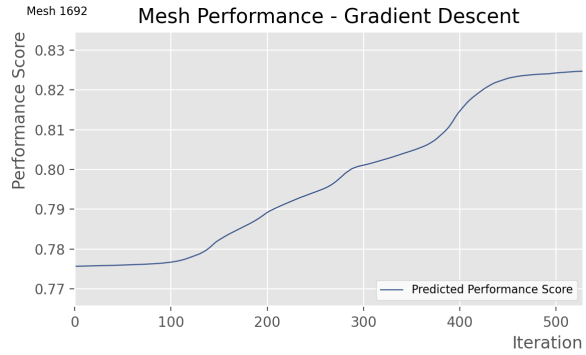


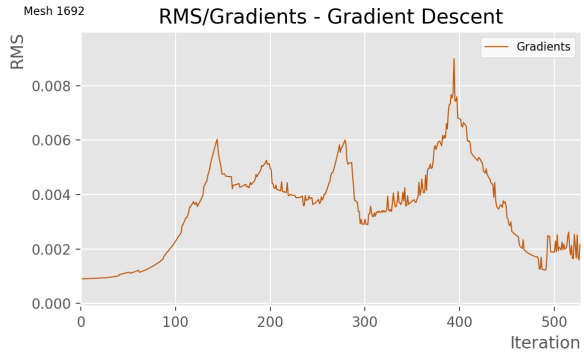
Mesh 1665 - control



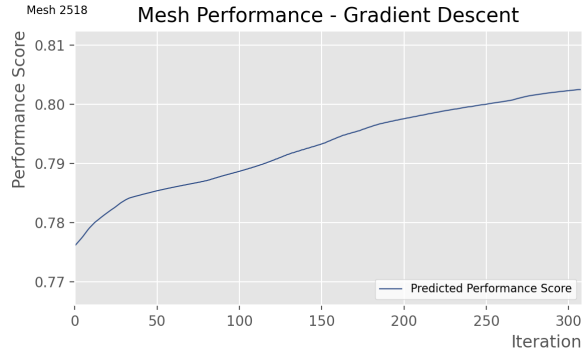


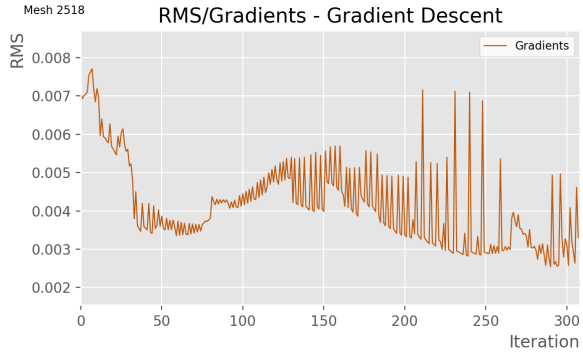
Mesh 1692 - control





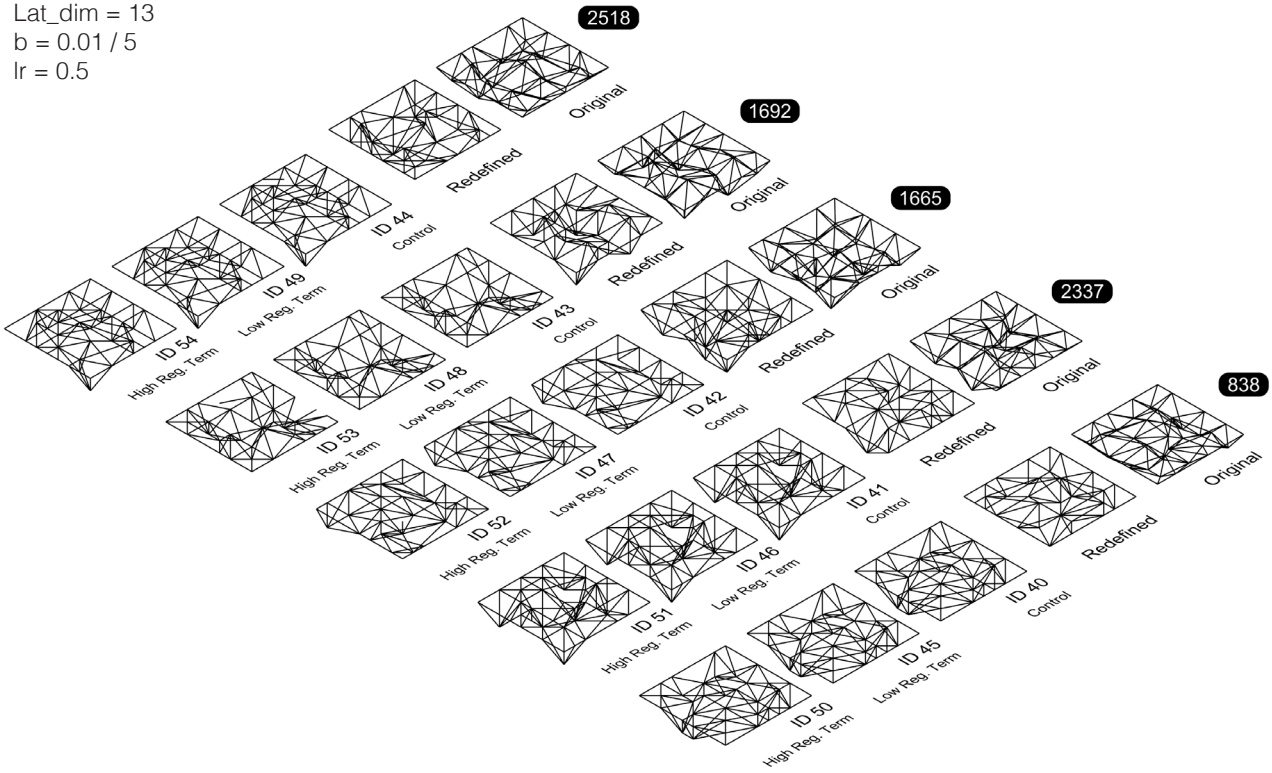
Mesh 2518 - control



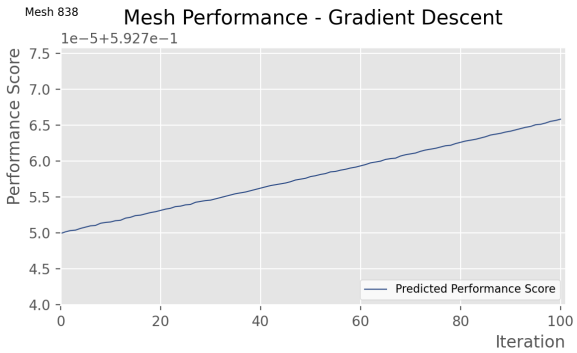


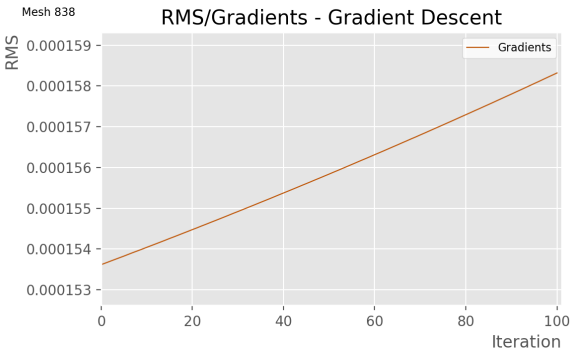
FURHTER GRADIENT DESCENT EXPLORATION: REGULARISATION TERM B

Lat_dim = 13
b = 0.01 / 5
lr = 0.5

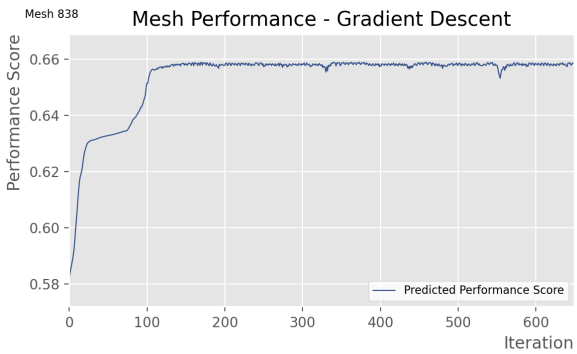


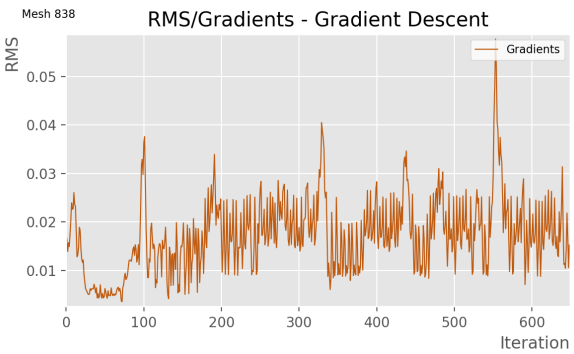
Mesh 838 - low b (0.01)



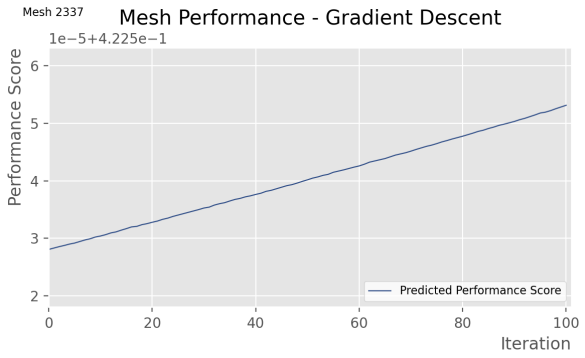


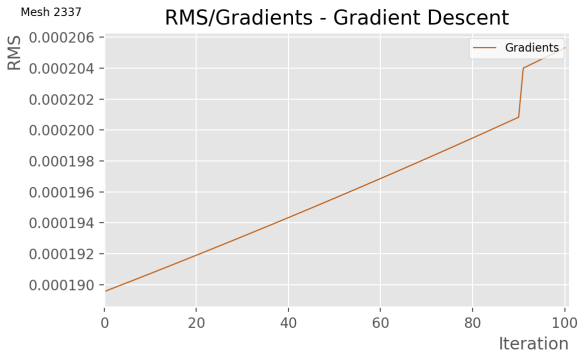
Mesh 838 - high b (5)



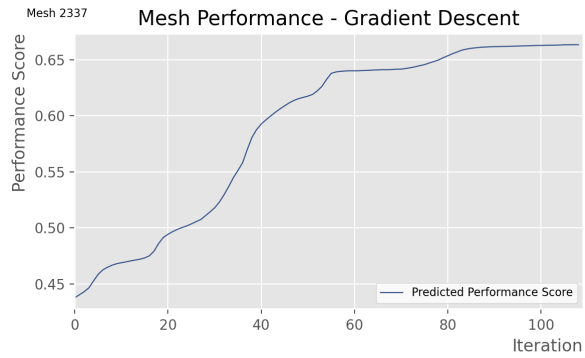


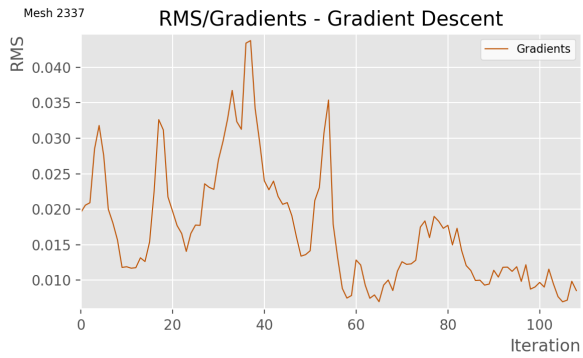
Mesh 2337 - low b (0.01)



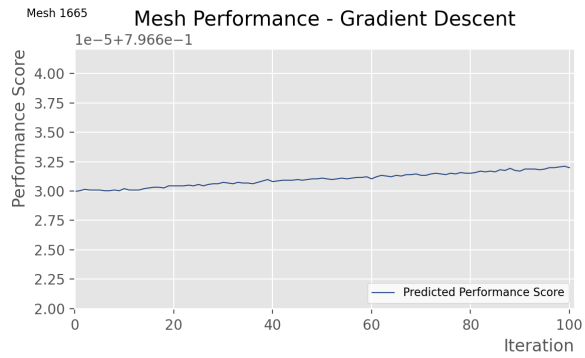


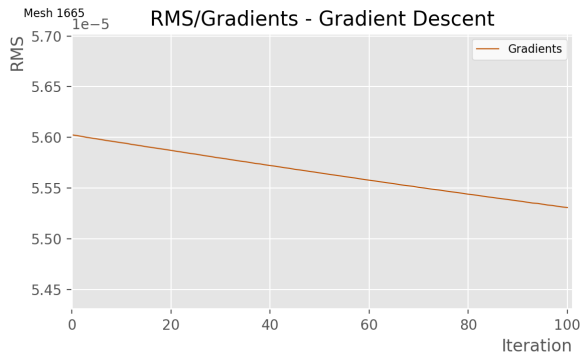
Mesh 2337 - high b (5)



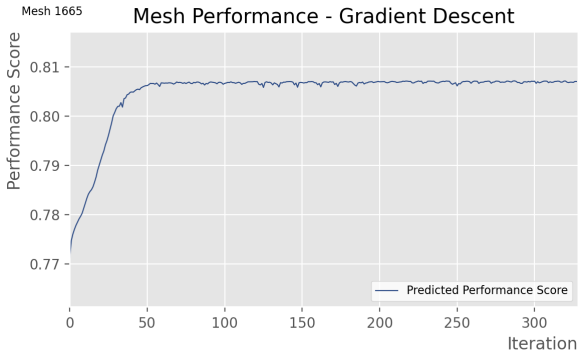


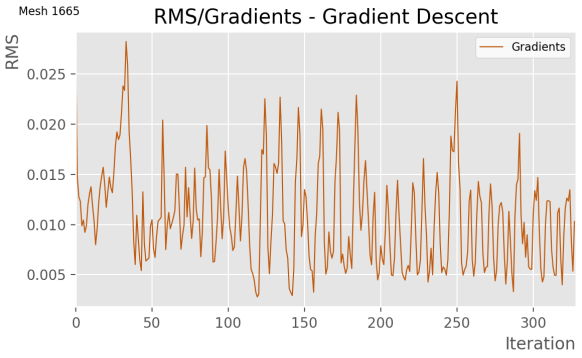
Mesh 1665 - low b (0.01)



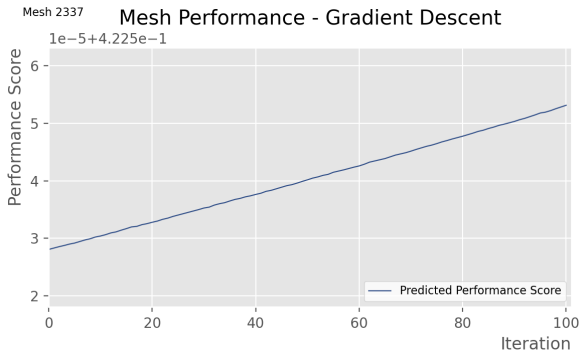


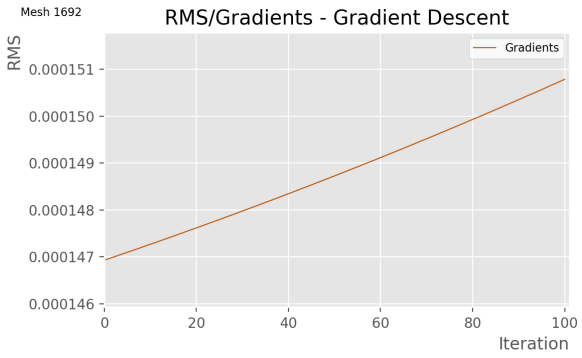
Mesh 1665 - high b (5)



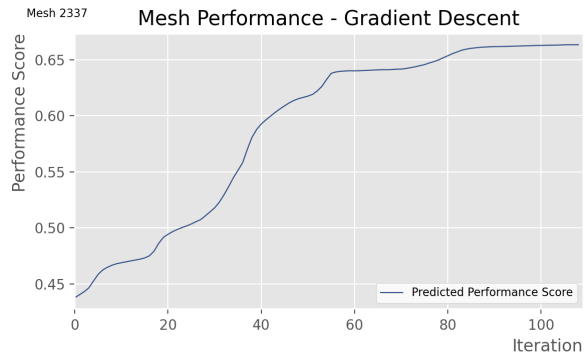


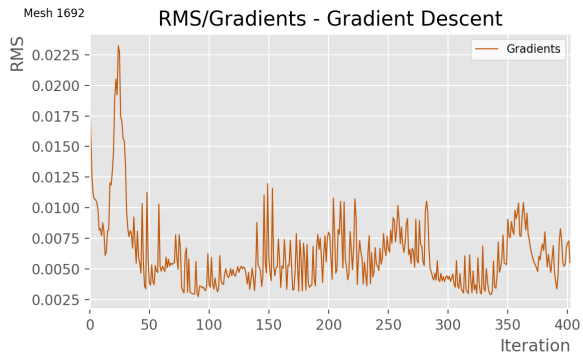
Mesh 1692 - low b (0.01)



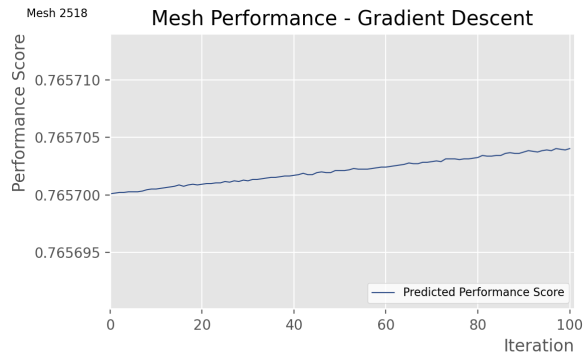


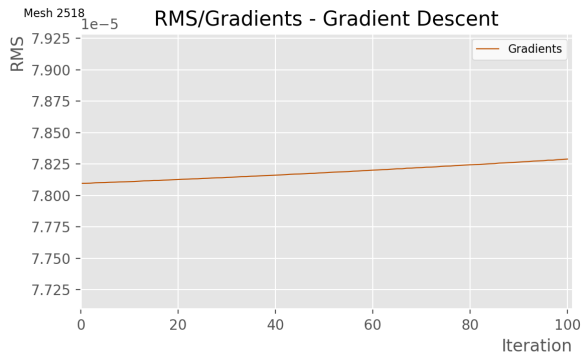
Mesh 1692 - high b (5)



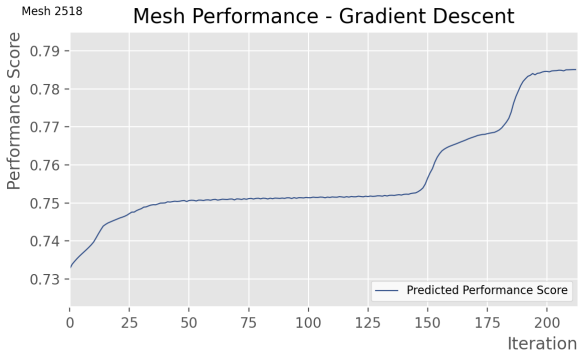


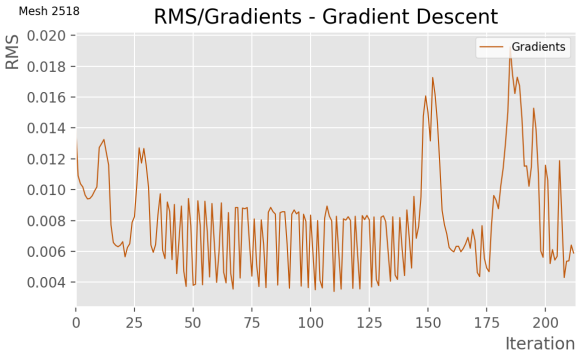
Mesh 2518 - low b (0.01)





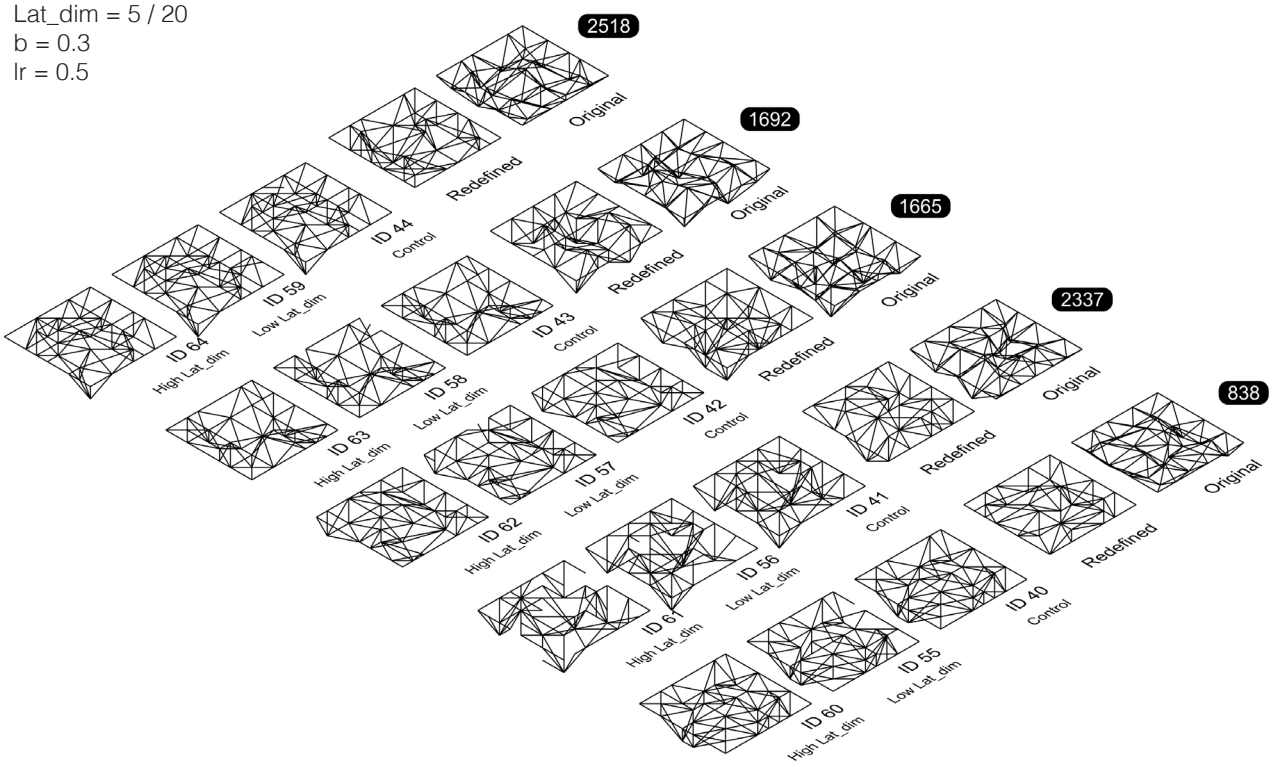
Mesh 2518 - high b (5)



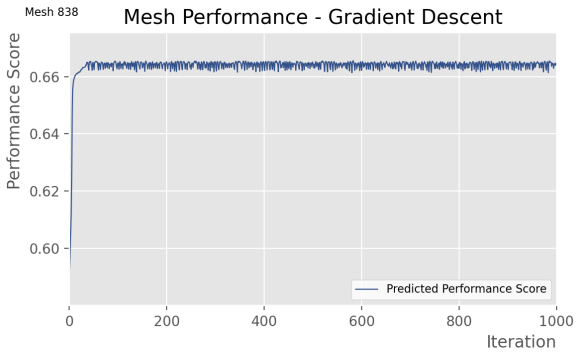


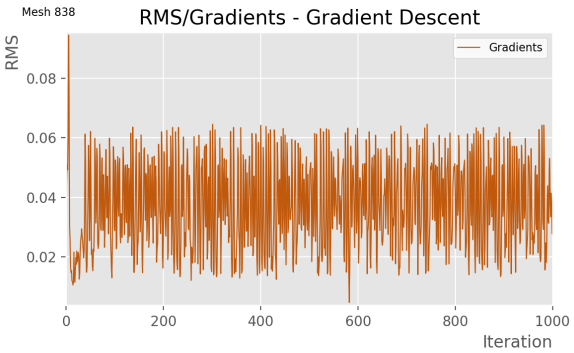
FURHTER GRADIENT DESCENT EXPLORATION: LATENT DIMENSION

Lat_dim = 5 / 20
b = 0.3
lr = 0.5

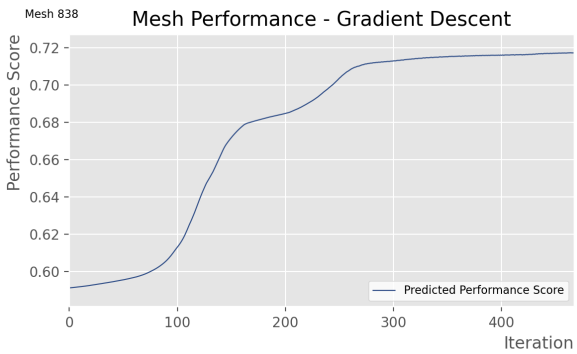


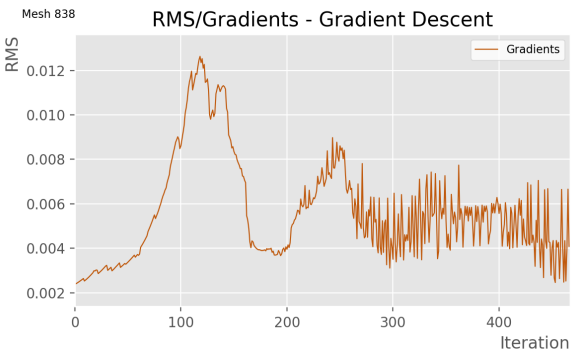
Mesh 838 - low lat_dim (5)



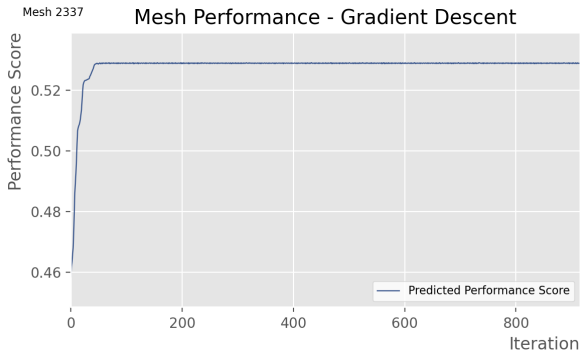


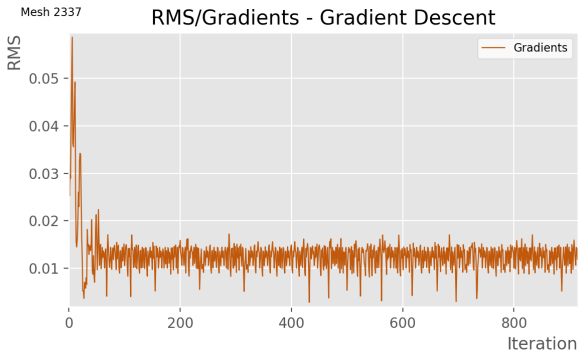
Mesh 838 - high lat_dim (20)



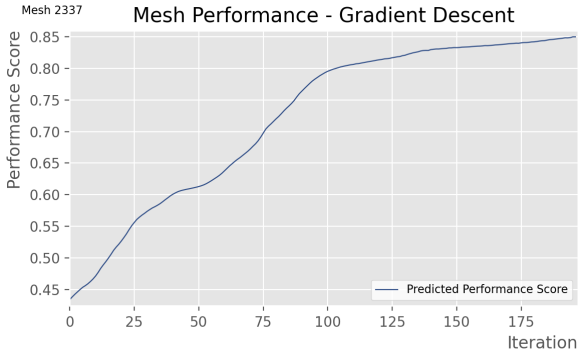


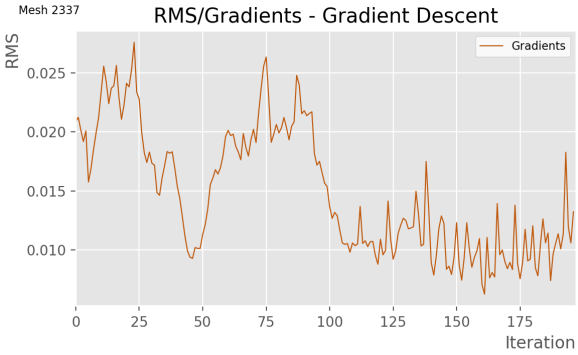
Mesh 2337 - low lat_dim (5)



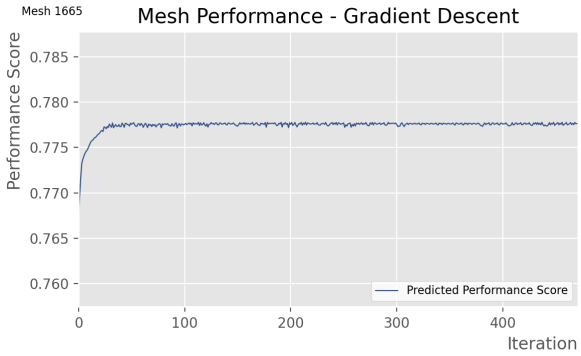


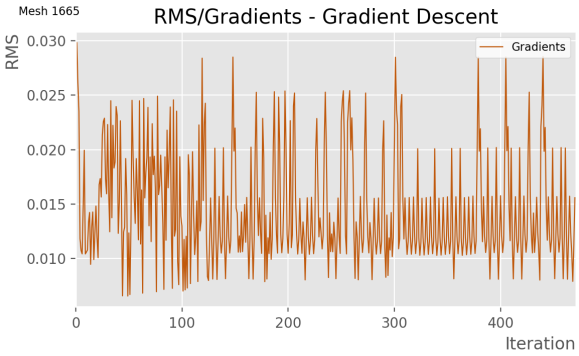
Mesh 2337 - high lat_dim (20)



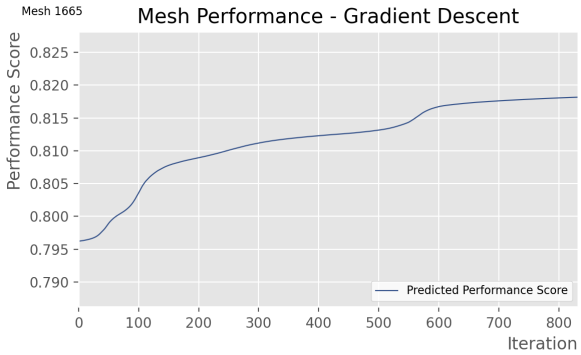


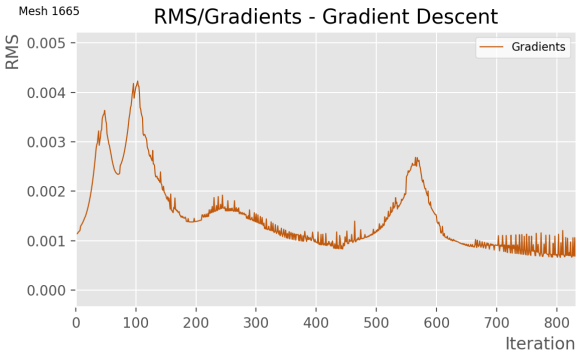
Mesh 1665 - low lat_dim (5)



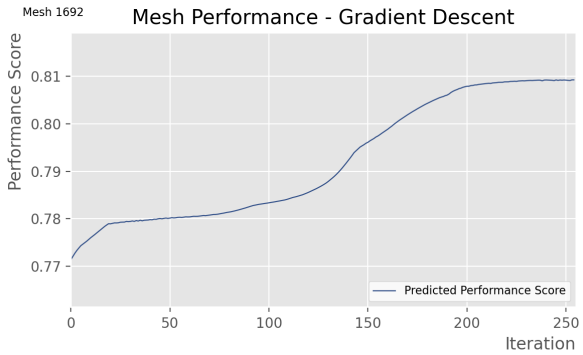


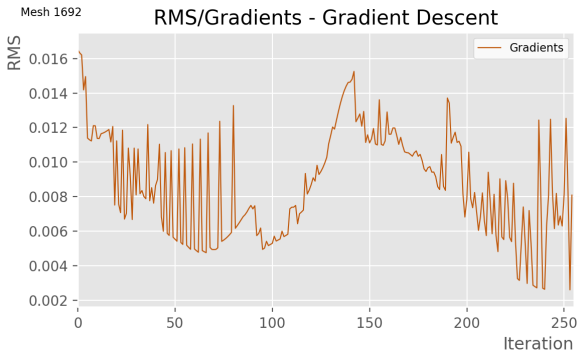
Mesh 1665 - high lat_dim (20)



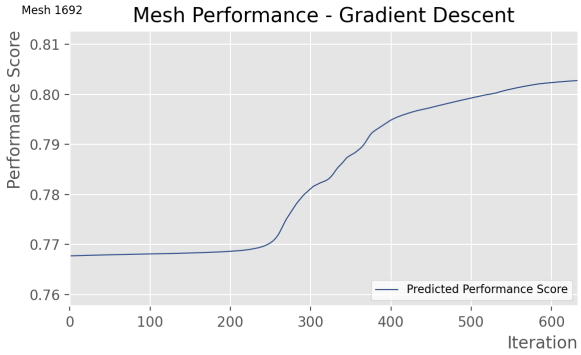


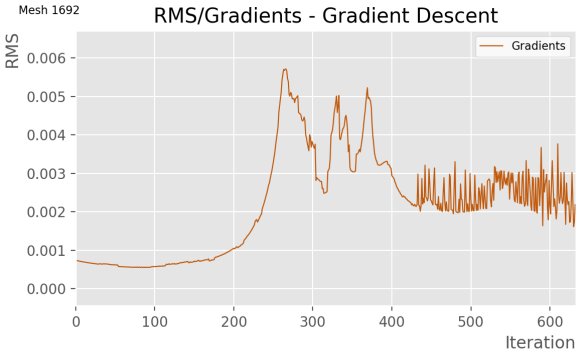
Mesh 1692 - low lat_dim (5)



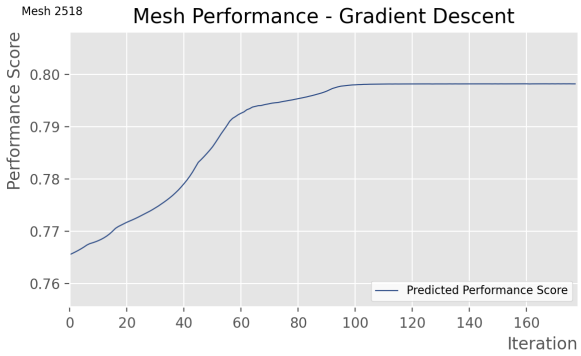


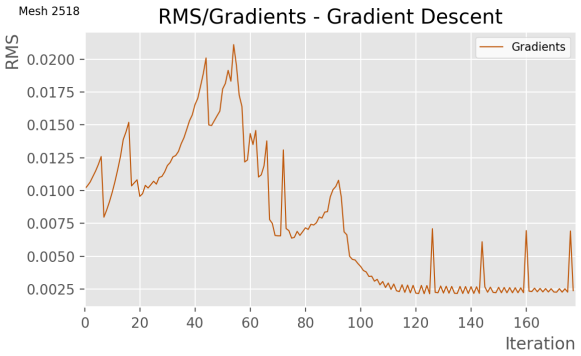
Mesh 1692 - high lat_dim (20)



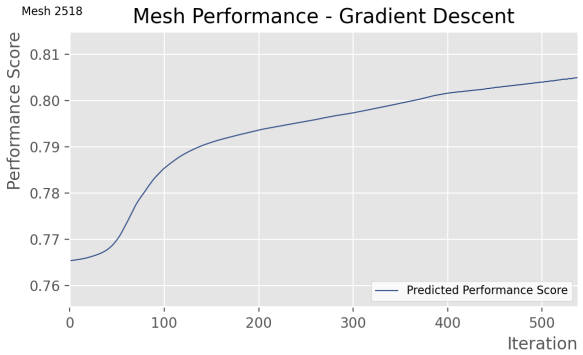


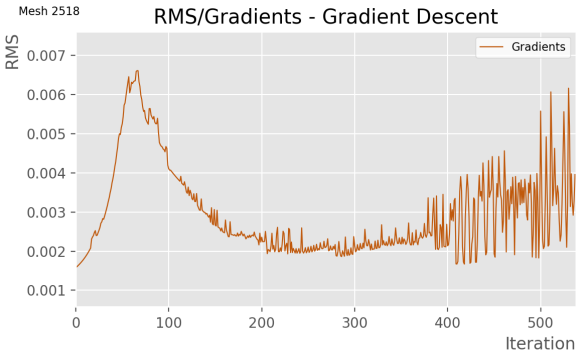
Mesh 2518 - low lat_dim (5)





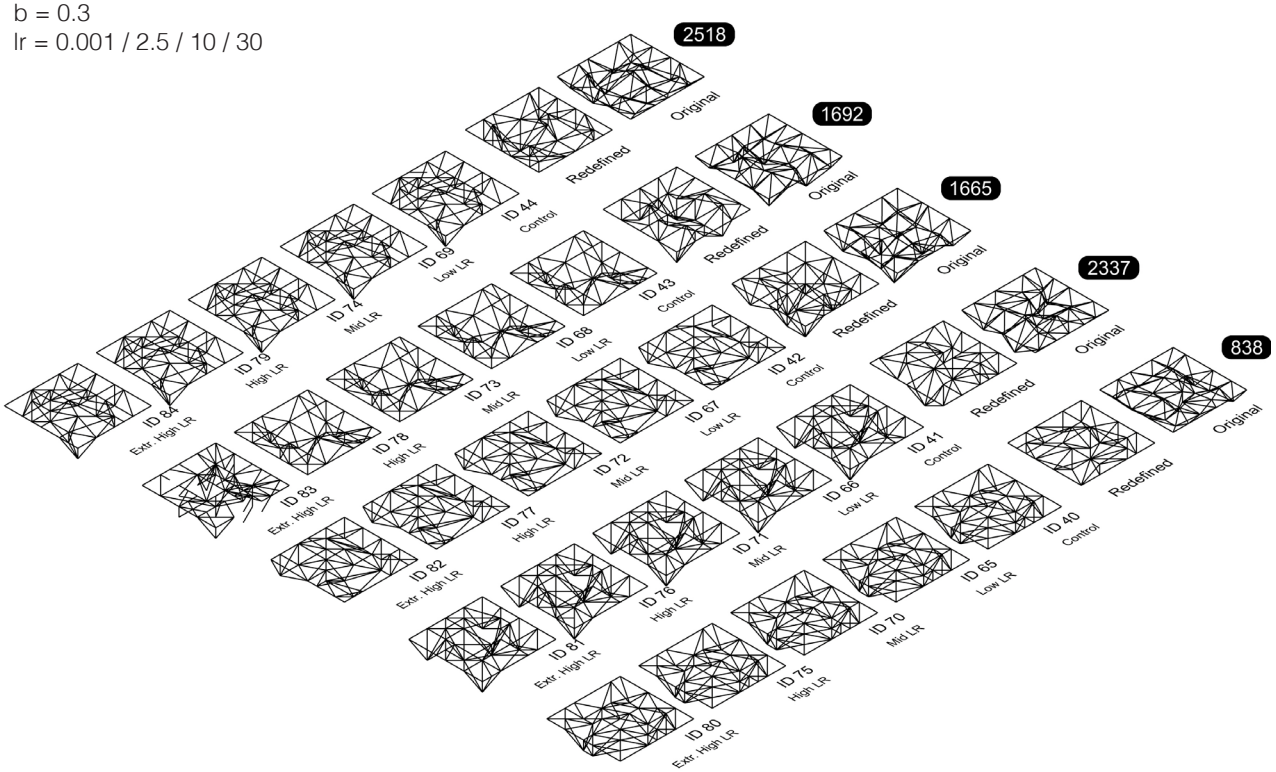
Mesh 2518 - high lat_dim (20)



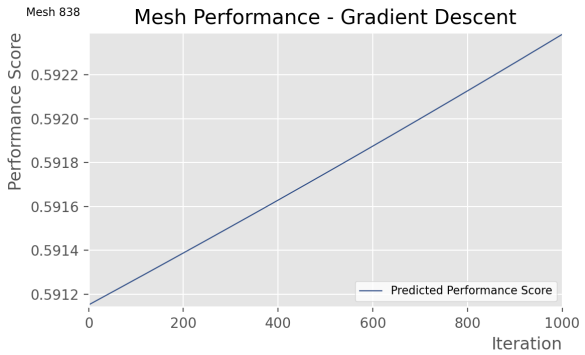


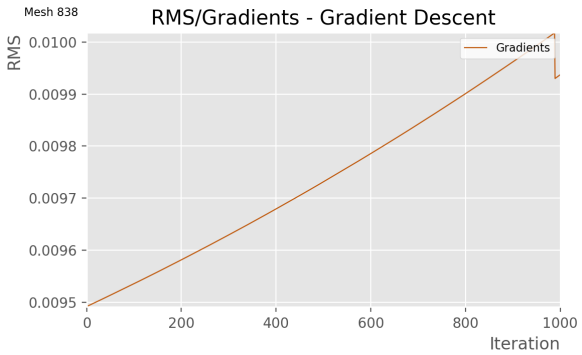
FURHTER GRADIENT DESCENT EXPLORATION: LEARNING RATE

Lat_dim =13
b = 0.3
lr = 0.001 / 2.5 / 10 / 30

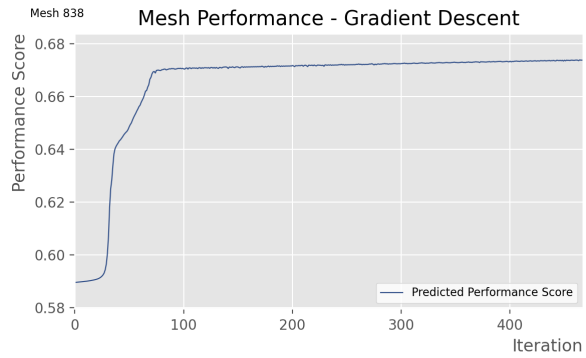


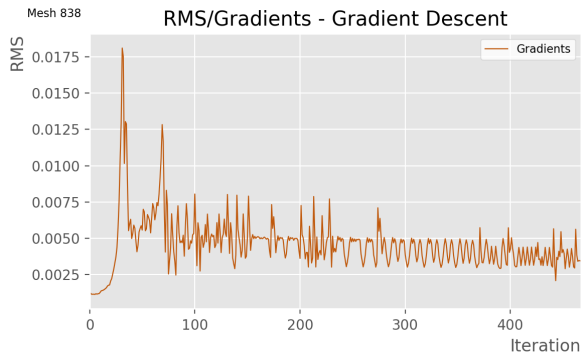
Mesh 838 - low learning rate (0.001)



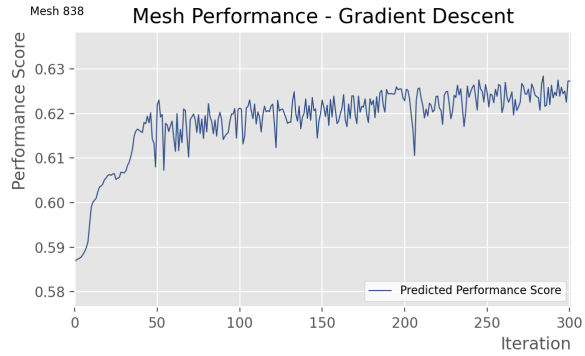


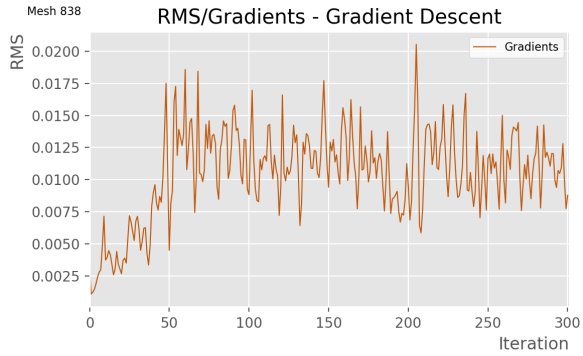
Mesh 838 - mid learning rate (2.5)



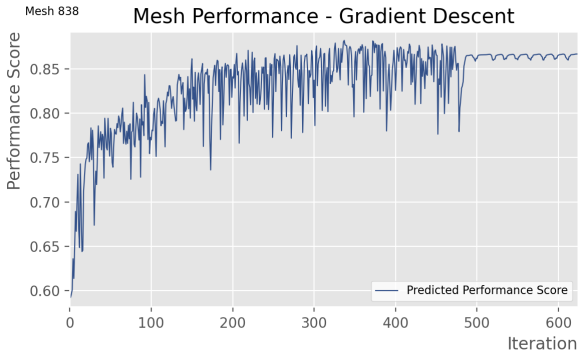


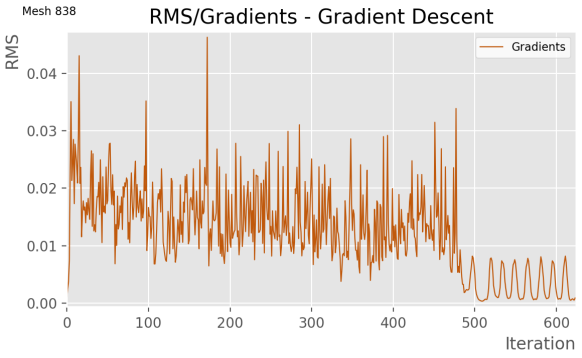
Mesh 838 - high learning rate (10)



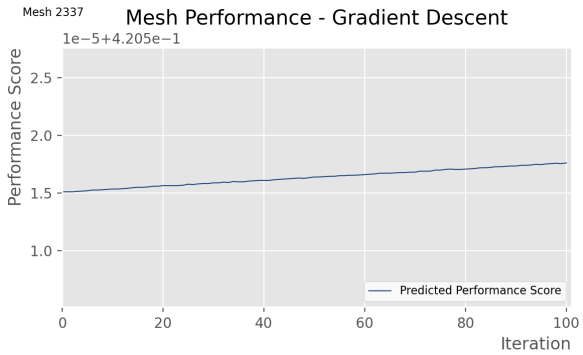


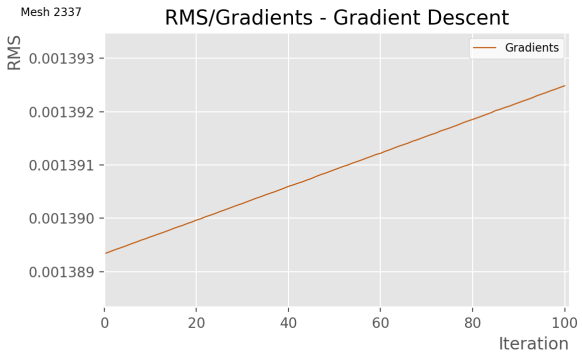
Mesh 838 - extr. high learning rate (30)



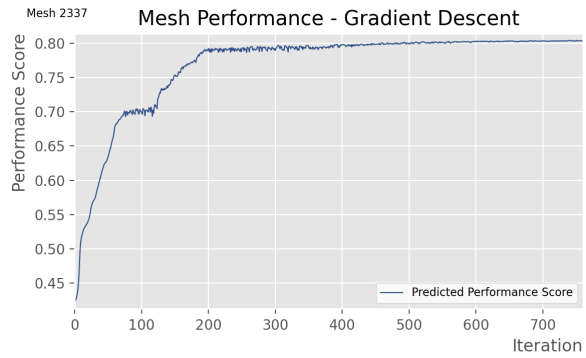


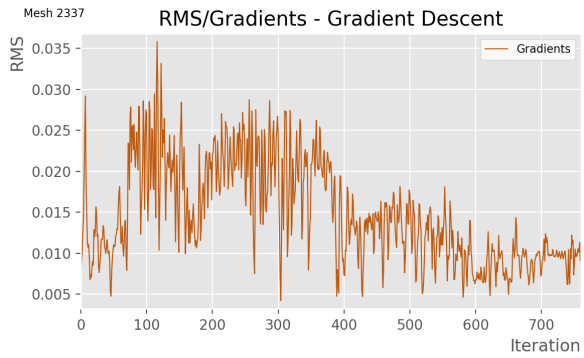
Mesh 2337 - low learning rate (0.001)



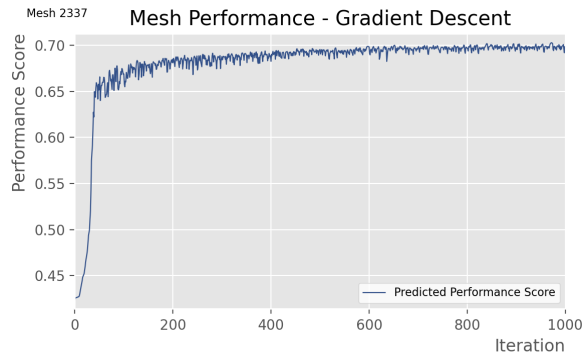


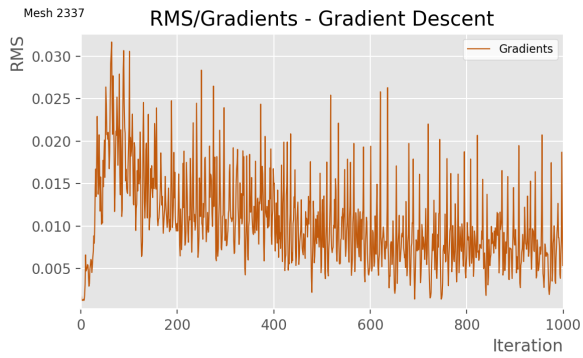
Mesh 2337 - mid learning rate (2.5)



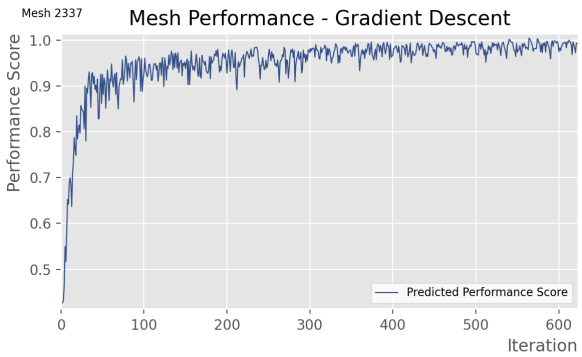


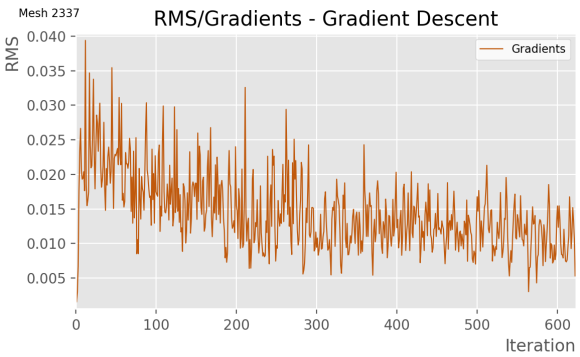
Mesh 2337 - high learning rate (10)



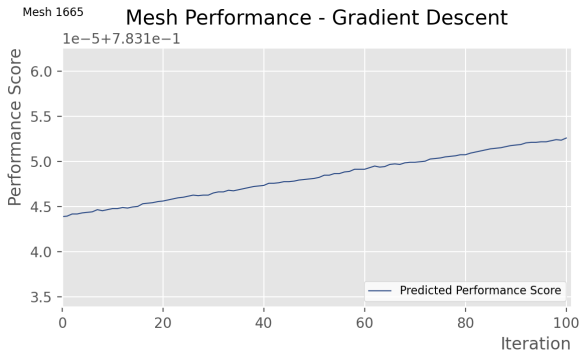


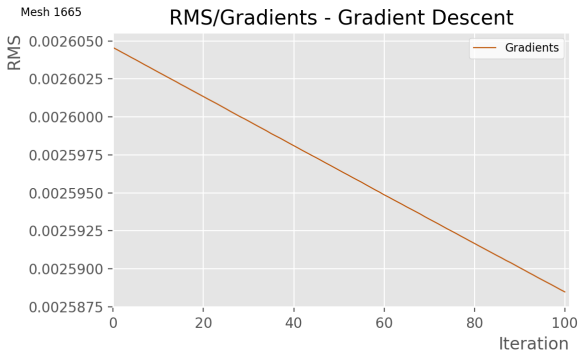
Mesh 2337 - extr. high learning rate (30)



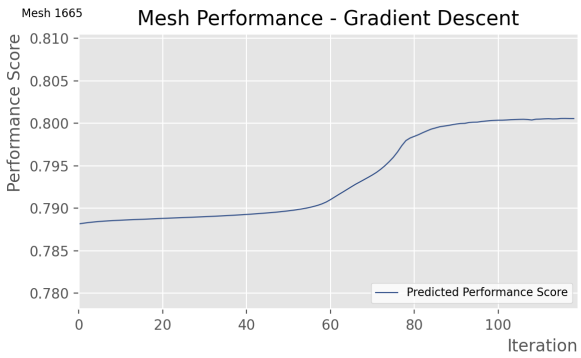


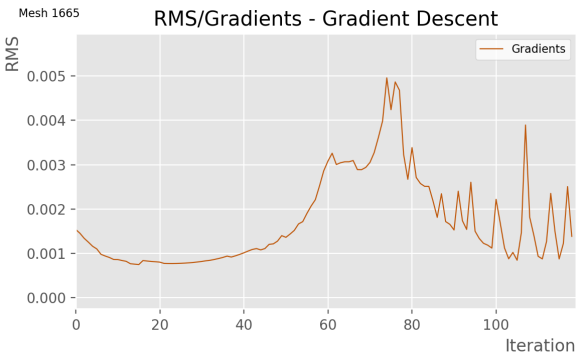
Mesh 1665 - low learning rate (0.001)



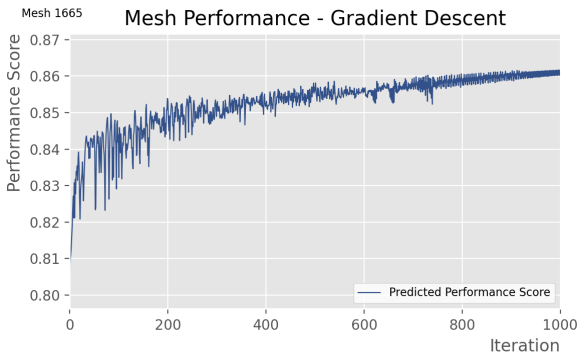


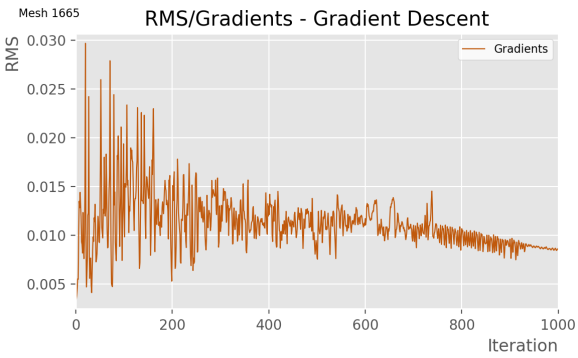
Mesh 1665 - mid learning rate (2.5)



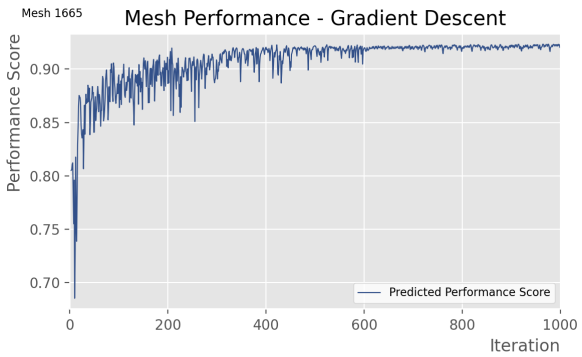


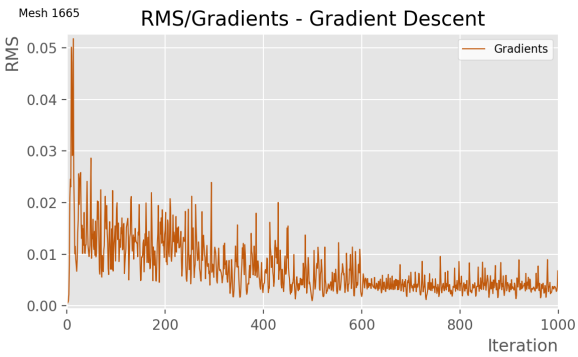
Mesh 1665 - high learning rate (10)



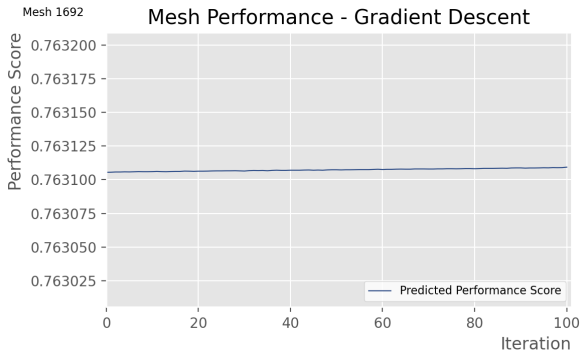


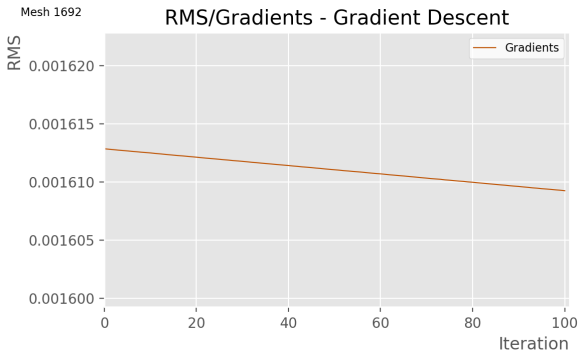
Mesh 1665 - extr. high learning rate (30)



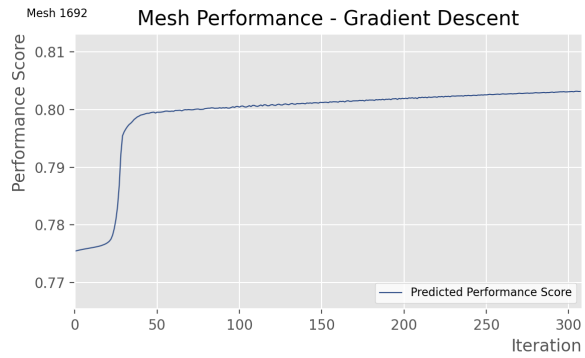


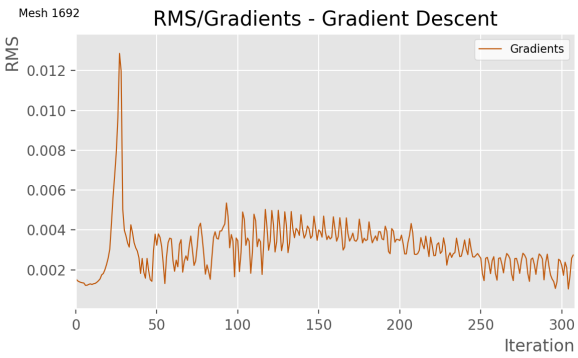
Mesh 1692 - low learning rate (0.001)



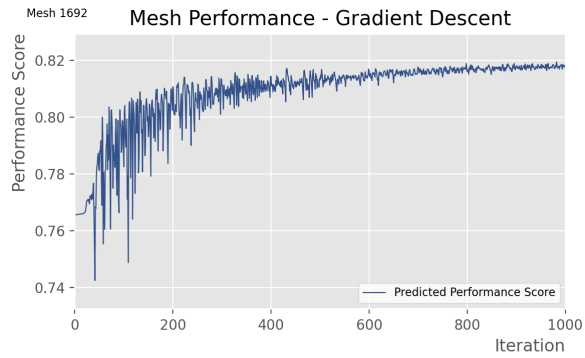


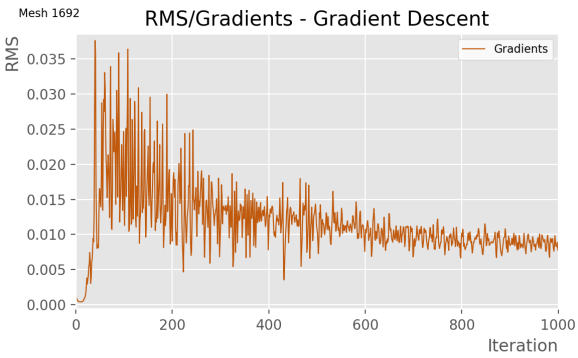
Mesh 1692 - mid learning rate (2.5)



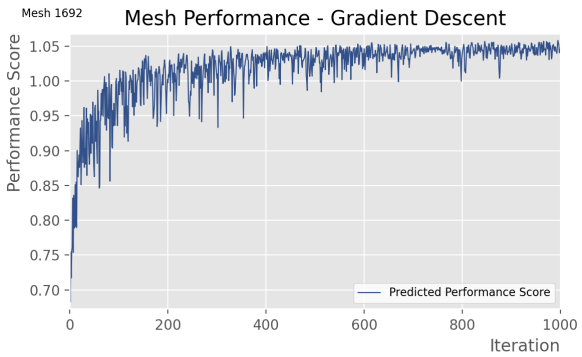


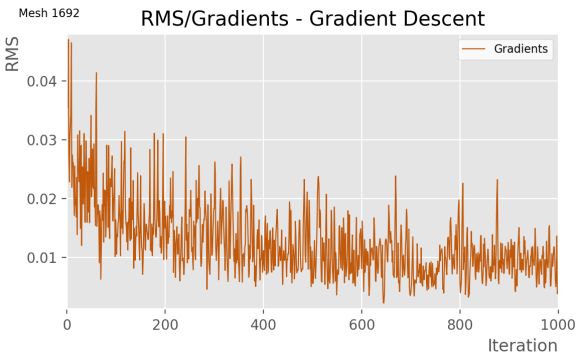
Mesh 1692 - high learning rate (10)



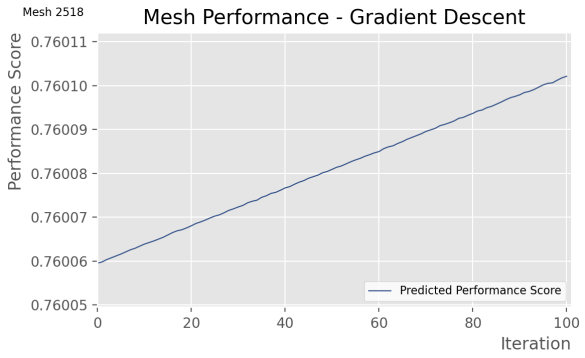


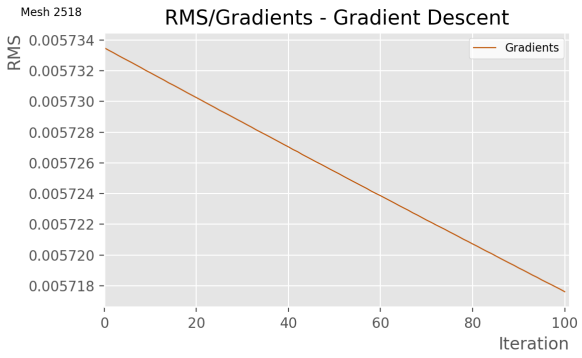
Mesh 1692 - extr. high learning rate (30)



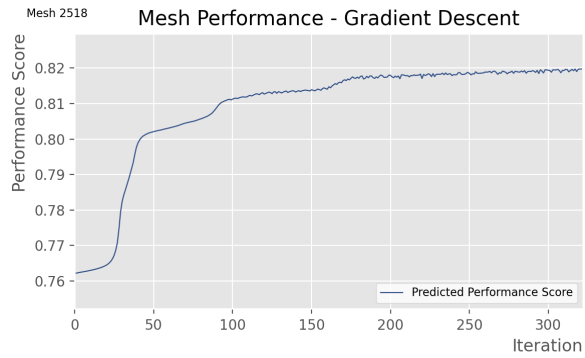


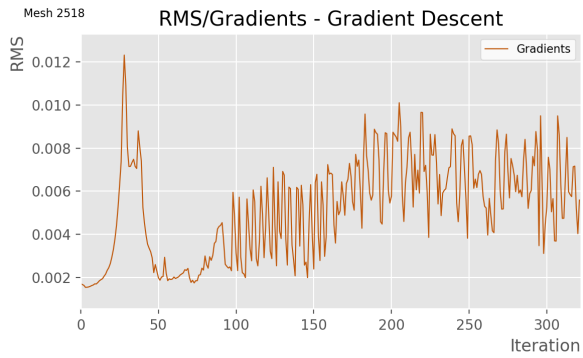
Mesh 2518 - low learning rate (0.001)



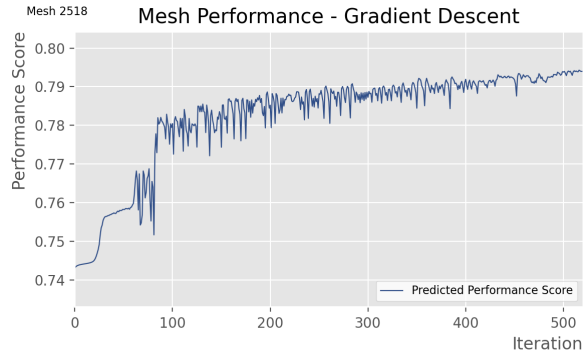


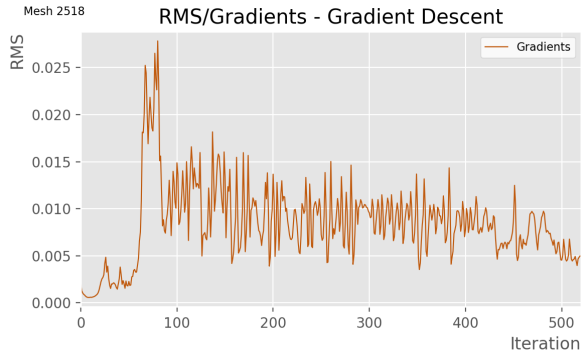
Mesh 2518 - mid learning rate (2.5)



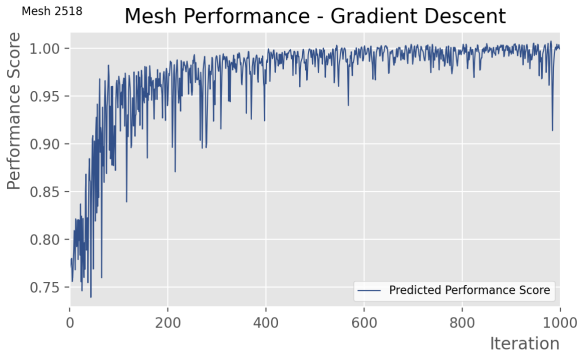


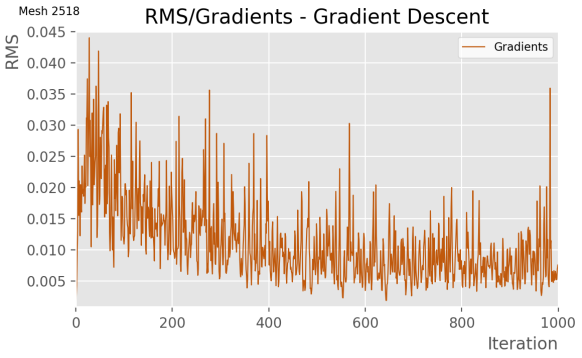
Mesh 2518 - high learning rate (10)





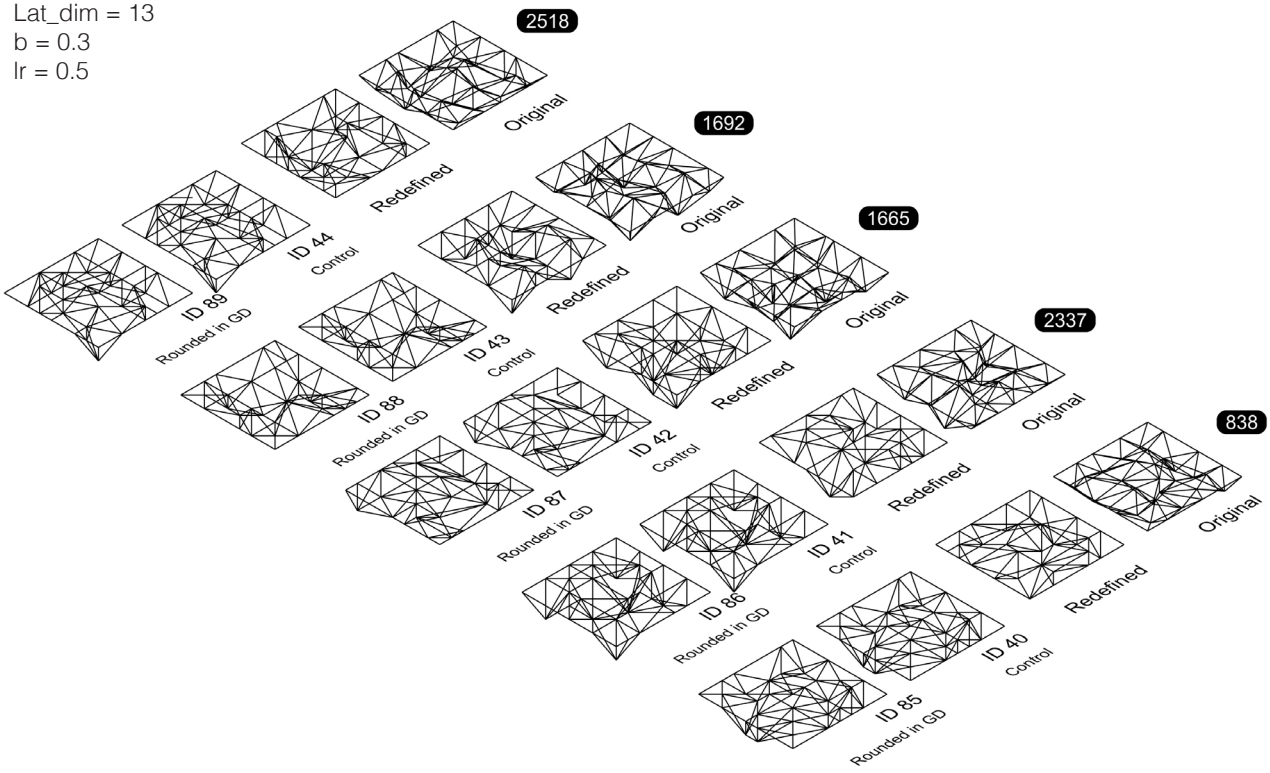
Mesh 2518 - extr. high learning rate (30)



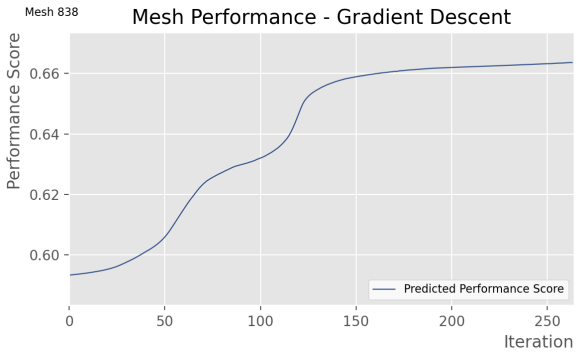


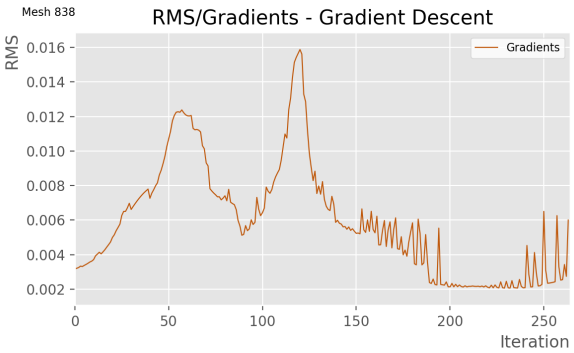
FURHTER GRADIENT DESCENT EXPLORATION: ROUNDED MATRICES

Lat_dim = 13
b = 0.3
lr = 0.5

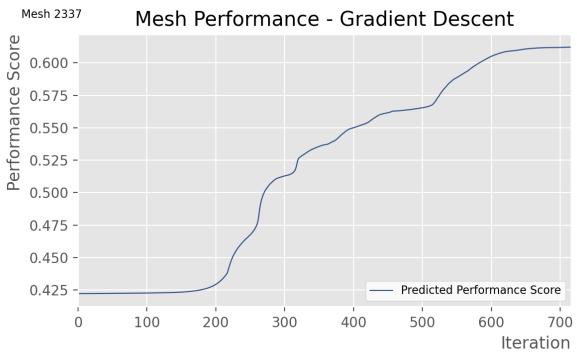


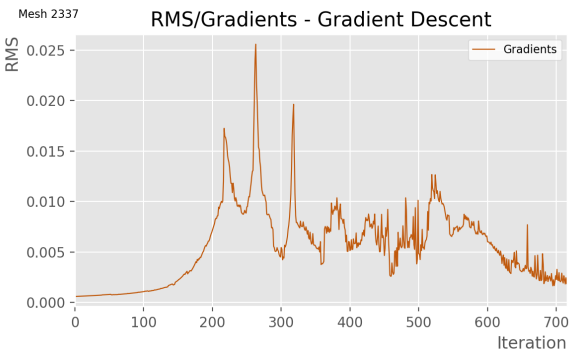
Mesh 838 - rounded matrices



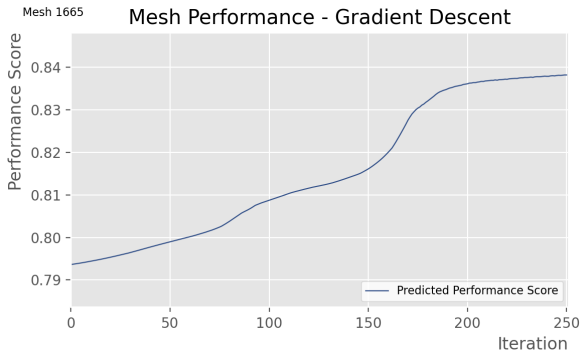


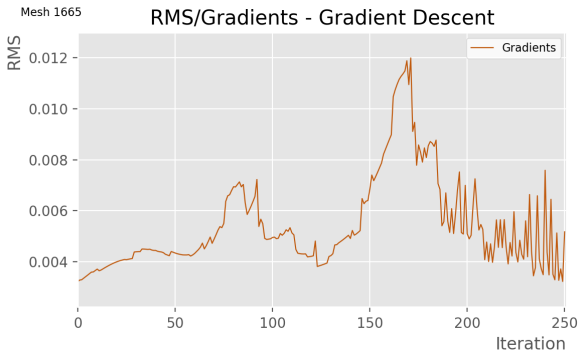
Mesh 2337 - rounded matrices



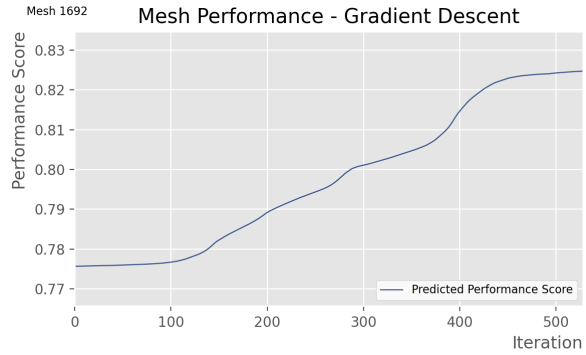


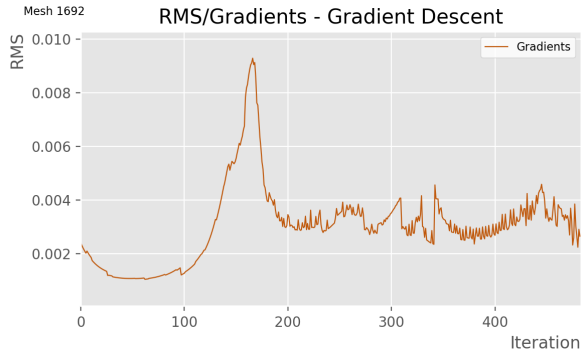
Mesh 1665 - rounded matrices





Mesh 1692 - rounded matrices





Mesh 2518 - rounded matrices