Integrated bio-inspired Design by AI:

Using cell structure patterns to train an AI model to explore topology design ideas

Namrata Baruah | 5326664

MASTER THESIS – REPORT

BUILDING TECHNOLOGY MASTER TRACK

Faculty of Architecture and the Built Environment

First Mentor: Dr. Michela Turrin, Design Informatics

Second Mentor: Dr. Charalampos Andriotis, Structural Design & Mechanics

External Supervisor: Dr. Alberto Pugnale and Gabriele Mirra, University of Melbourne

Student: Namrata Baruah | 5326664

Delft University of Technology
Faculty of Architecture
and the Built Environment

ABSTRACT

The construction industry around the world is growing exponentially. In building design, the structural design is usually incorporated in the latter stages of the design, and the computational tools currently available focus more on converging and optimizing a single solution rather than providing an opportunity for the designer to explore design ideas. Incorporating structural design in the initial design phases can lead to more efficient and cost-effective structures, but in order to do so practical tools should be available to the designer to deepen the design space and allow them to efficiently explore it. Generative AI models could provide a solution to the problem and expand the design space by learning from the data provided.

This thesis explores such a solution by means of training an AI model (variational Autoencoder) on an artificial dataset of 2D lattice patterns. To do that, a dataset of 6 unique 2D lattice patterns is created and used to train a simple VAE model. The methodology used in this thesis is to validate if an AI can aid in the exploration of design ideas by providing a larger design space with newly generated data that contain learned features from the dataset rather than through constraints set by the designers.

The results show that the VAE model can learn features and provide a greater diversity of design than the original dataset through newly generated designs. The output of the VAE model in this thesis is then explored for possible integrations into the design process for the early stages of design. For this, the generated patterns that are distinct and unique are identified and applied to a shell structure to explore topology design ideas.

This thesis explores the possibilities of an AI–design integration, but for the methodology to be practically incorporated into the design process, further development of the AI model is required as well as exploring other advanced generative AI models would be beneficial.

Keywords: Generative AI, Variational Autoencoder, Natural Patterns, Topology Exploration, AI Integration in Design, Deep Learning, Python programming

ACKNOWLEDGMENT

I have had invaluable support throughout this thesis. It was a very new and challenging topic for me and I am indebted to the people who have encouraged me and helped me throughout.

I would firstly like to thank my first mentor Dr.Ir. Michela Turrin, for the clear explanations, encouragement, references and always guiding me in the right direction and helping me see the bigger picture during the course of this thesis. I thoroughly enjoyed our discussions regarding the thesis and other topics too. Secondly, I would like to thank Dr. Charalampos Andriotis, for the valuable advice and insight into AI models and also for the encouragement and guidance during this thesis. The practical suggestions during our discussions were extremely helpful in understanding and progressing on this thesis topic. Thirdly, I would like to thank my external advisors, Dr. Alberto Pugnale and Gabriele Mirra, for the encouragement and patience, for helping me understand the topic better and showing me the right tools for solving the problems in the code. It was an incredible learning experience and thank you for sharing your experience and insights into the topic. This thesis would not have been possible without that.

I would also like to thank my friends, who have been a constant support throughout the process. Their inspiring discussions and work really encouraged me to do my best.

Lastly, but definitely not the least, I would like to thank my parents. My mother for always being an inspiration for me and giving me the courage to follow my ambitions. My father for making me believe in his motto “when there is a will there is a way”. It would not have been possible without them.

ABBREVIATIONS

AI:	Artificial Intelligence
BCE loss:	Binary Cross-Entropy Loss
CNN:	Convolutional Neural Network
FEM:	Finite Element Method
GAN:	Generative Adversarial Network
KLD:	Kullback-Leibler divergence
MNIST:	Modified National Institute of Standards and Technology database
MSE Loss:	Mean Squared Error Loss
NN:	Neural Network
VAE:	Variational Auto Encoder

CONTENT

I INTRODUCTION

“The loftiest and most difficult problems arise in architecture from the need to realize a synthesis between opposing sets of factors: harmony of form and the requirements technology, heat of inspiration and the coolness of scientific reason, freedom of imagination and the iron laws of economy.”

— Pier Luigi Nervi in Structures, 1956

1.1 Background

In building design, including the disciplines of architecture and structural engineering, the design process is conventionally divided into four phases: Conceptual Design, Schematic Design, Design Development, and Construction Documents (Architects. 2007). The major decisions regarding the building geometry, structure, massing is made during the Conceptual Design. These design decisions account for 75% of the final product costs. (Hsu 2000). Integrating the structure into the Conceptual Design phase can lead to several advantages, including reduced construction cost, architectural elegance, and is inherently safe. Within the domain of Structural design, topology exploration is convenient for exploring and validating ideas in the initial stages of design as it only requires an initial domain – massing studies- to act upon (M.P. Bendsøe 2004). Precedents have shown how inspirations from nature can suggest valid design directions – beyond the mere formal similarity toward functional principles (Mizobuti, C.M and Junior 2020). In the vast examples from nature, this thesis focuses on forms of cellular solids. The properties of cellular solids depend directly on the shape and structure of the cells, exploring the shape and topology of the cell walls might prove to be interesting for application in topological design explorations.

1.1.1 Computational tools for Conceptual Structural Design

Today's design practices make widespread use of computational tools throughout the design process. The designers must have the correct tools to aid in their design process. Until recently, the computational tools have mainly been used for analytical purposes in structural design. Now, their role is becoming more versatile and is being used in the generation of design concepts too (Kicinger 2005). To aid the designers in the conceptual design phase, the computational tools must allow the exploration of a variety of solutions.

In the current scenario, most of the computational tools focus on optimization rather than exploration. Optimization aims to minimize or maximize an objective value by the variation of design variables, while at the same time satisfying certain constraints. (Boonstra 2018)

To facilitate exploration, it is imperative that the designer has access to a larger design space of the design idea not constrained by parameters set by the designers. Design space is an expression of the design idea (Mirra and Pugnale 2021). It is a closed system that can generate all possible solutions to a design problem. It is bounded by selecting a set of design variables that limit the search for suitable solutions.

In the field of structural optimization, the solutions depend on the representation of the design space. Expanding the design space can lead to more freedom in the range of possible solutions. Designers are invaluable in the process of design since their experience can reduce the problem. But to aid an individual designer in overseeing the complete design problem – i.e., the complex relationships between disciplines - multi-objective building optimization methods can be useful to handle larger design spaces. (Boonstra 2018). There has been extensive research carried out in that aspect, focused mainly on two approaches: ' Super-structure' and 'Super-structure free.'

A) Superstructure Approach (Boonstra 2018):

In the super-structure approach, all the design variables are fixed by the designer who makes the representation. In the case where the topology of the design is specified in detail and predefined in advance; the representation was defined as parameterization in (Kicinger_2005).

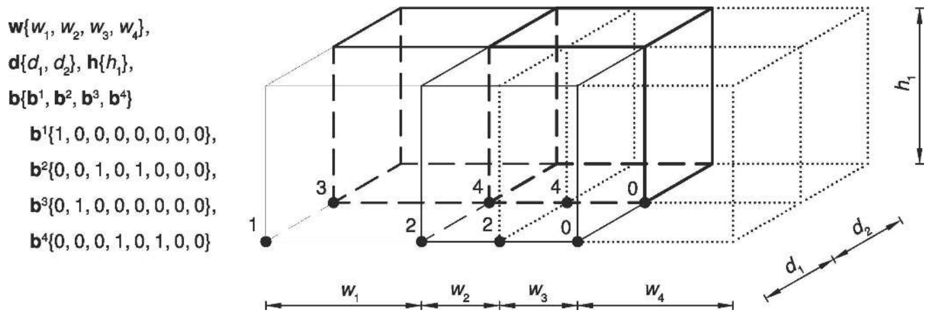

$\mathbf{w}\{w_1, w_2, w_3, w_4\}$,
$\mathbf{d}\{d_1, d_2\}$, $\mathbf{h}\{h_1\}$,
$\mathbf{b}\{\mathbf{b}^1, \mathbf{b}^2, \mathbf{b}^3, \mathbf{b}^4\}$
$\mathbf{b}^1\{1, 0, 0, 0, 0, 0, 0, 0\}$,
$\mathbf{b}^2\{0, 0, 1, 0, 1, 0, 0, 0\}$,
$\mathbf{b}^3\{0, 1, 0, 0, 0, 0, 0, 0\}$,
$\mathbf{b}^4\{0, 0, 0, 1, 0, 1, 0, 0\}$
1
2
2
0
3
4
4
0
h_1
d_1
d_2
w_1
w_2
w_3
w_4

Figure 1 Example of a Superstructure-based spatial representation. Supercube representation of a building spatial design, spaces 2 and 4 are described by two cells each, the two right cells are not used to describe a room. (Boonstra 2018)

B) Superstructure free Approach (Boonstra 2018):

In the Super-structure free approach, new design variables may originate or disappear - utilizing rules or geometric operators. In this scenario the topology of the design is changeable; this representation was called generative in (Kicinger_2005).

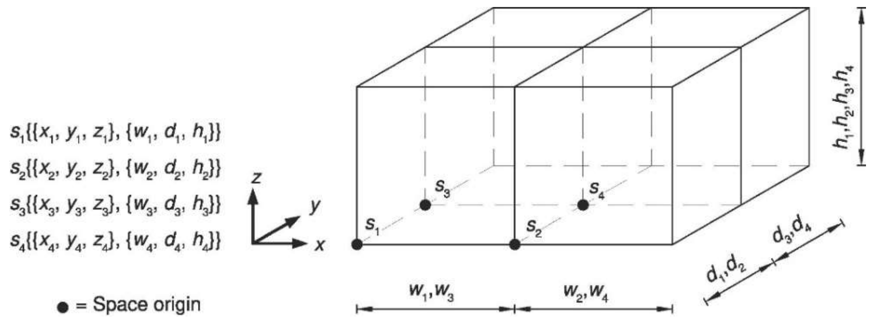

$s_1\{\{x_1, y_1, z_1\}, \{w_1, d_1, h_1\}\}$
$s_2\{\{x_2, y_2, z_2\}, \{w_2, d_2, h_2\}\}$
$s_3\{\{x_3, y_3, z_3\}, \{w_3, d_3, h_3\}\}$
$s_4\{\{x_4, y_4, z_4\}, \{w_4, d_4, h_4\}\}$
● = Space origin
z
y
x
s_1
s_2
s_3
s_4
h_1, h_2, h_3, h_4
d_1, d_2
d_3, d_4
w_1, w_3
w_2, w_4

Figure 2 Example of a Superstructure Free representation. A movable and sizeable (MS) representation for spaces is introduced for the super-structure-free design space representation. For this, a building is described with a vector that lists all the spaces. (Boonstra 2018)

1.1.2 Natural Forms as inspiration for Structural Design

Man has been drawn to nature for its diversity of systems. The vast number of substances formed are from the permutation and combination of a relatively small number of chemical elements (Pearce 1978). The formative process in nature is characteristically governed by the least energy responses. That means that the form of a natural object is created as the best response to the acting forces (intrinsic and extrinsic) while using the minimum energy. If building systems can be considered analogous to natural objects, which is highly efficient in responses to forces, it may offer a real possibility of generating new design ideas. (Pearce 1978)

Many materials have a cellular structure: an assembly of prismatic or polyhedral cells with solid edges and faces packed together to fill space (Gibsob and F.Ashby 1997). Their cellular structure gives them unique properties that are exploited in a variety of applications. In compression, cellular solids can withstand large strains at nearly constant stress. This aspect of cellular solids can be explored for the design of compression structures (e.g., Shells). Learning from the topology of the cells can help improve the topology of the structures.

This thesis explores the opportunity of utilizing precedents from nature in the conceptual design phase. As such complex systems are difficult to be integrated into later phases of design, extracting basic principles and investigating their incorporation and evolution through the initial phases of design can lead to interesting solutions.

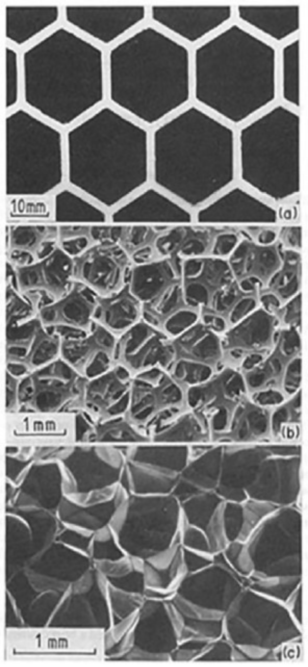


Figure 3 Examples of Cellular solids: a) a two-dimensional honeycomb. b) a three-dimensional foam with open cells. c) a three-dimensional foam with closed cells (Gibsob and F.Ashby 1997)

1.1.3 AI

“Artificial intelligence is that activity devoted to making machines intelligent, and intelligence is that quality that enables an entity to function appropriately and with foresight in its environment.” (Nilsson 2010)

In the recent years, AI has become embedded in people’s lives. From smartphones, Siri and Alexa to detecting and differentiating neurological conditions of patients in MRI (Vogelsanger and Federau 2021). Out of the different AI models, the generative models are promising for use in the design field; as they can create new content by detecting underlying patterns related to the input and produce similar content. In design, rudimental tasks that involve repetition can be automated and even enhanced with the help of AI, letting the designers focus more on the ideas and creativity. It can become a tool to aid designers in enhancing their creativity (Ahmed 2019) (Basu 2019).

AI is the broad umbrella term that includes Machine learning and Deep learning. It is explained further in the Literature Research section of this report.

1.2 Problem Statement

Computational design has shown high potential in numeric performance-driven approaches. However, designers do not exclusively use numeric performance to develop design ideas. The development of a design idea depends also on the designer’s knowledge and experience, including reasoning from inspirations. Such an approach tends to limit the search for optimal solutions and converge on a design solution rather quickly.

In both the cases of Superstructure and superstructure free approach, the bounds of the design space are dependent on the designer. In the superstructure approach, the drawback is that a better solution, outside the design space representation, will never be found as the variables are pre-defined. In the newer superstructure-free approach, the rules limit the number of solutions explored in the design space which can lead to some optimal solutions being left out.

The Computational tools must allow the freedom of exploration within the design environment. There is usually no correct answer in architectural design, and such tools must allow for a variety of design options while encouraging the user towards the ones with better performance.

Nature has evolved complex and apt topologies to solve all sorts of problems. The investigation of such a framework can help provide important physical insight as well as valid design directions. There is potential in exploring such complex structures in nature.

This thesis aims to explore design ideas through application of AI-generated patterns in shell structures. Natural patterns of cellular solids will be used to train the AI model. The current research available on these fields have examples of using AI for identifying and segmenting white blood cells (Xin Zheng 2018), Using AI to generate shells structures after being trained on a dataset of human-defined cell structures (Mirra and Pugnale 2021). But there is a gap in combining structural topology exploration with natural forms and this thesis explores the possibility of using AI for the same. AI can help open up ideas and expand the palette of architecture.

1.3 Research Questions

To explore possibilities for the analyzed problem and sub-problems, the main question of the project is:

Main Question: *How can AI extract useful information from a dataset of cellular solid structure patterns and reuse it to generate new patterns for structural design?*

To answer the main question, the following sub-questions are formulated:
Sub Question 1: What are the selection criteria of the cellular solid patterns for creating the dataset?
Sub Question 2: How to artificially create a dataset?

Sub Question 3: How to train a generative AI model on the custom dataset?

Sub Question 4: How can the AI-generated patterns be used to explore topology optimization design ideas? (Application)

1.4 Aims and Limitations

1.4.1 Aims:

1. The general aim of the thesis is to investigate the application of AI as a tool for generating design ideas. The workflow of the thesis can be used to explore different domains for training an AI.

2. To select patterns from nature that show perforations and can be modeled in 3D using a grasshopper script.

3. To train an AI with natural cellular solid perforation patterns and generate new patterns from the trained model.

4. The thesis aims to compare the AI-generated patterns (visually and also through structural performance using FEM) and verify that the AI-generated designs spaces result in a greater variety of examples while ensuring a still good structural performance.

1.4.2 Limitations:

1. For the scope of this thesis only cellular solids are considered for the dataset. The format of the data that has been used for the dataset are 2D images, hence focus was mainly on 2D cellular solid structures whose topologies are distinct and influence their structural performance.

2. Since the data available on the natural cellular solids is not vast in terms of recreating them artificially, only certain categories of them will be selected. To ensure that there is enough data, data-augmentation will be carried out.

3. The generative AI model used in this thesis is a Variational Auto Encoder (VAE). Convolutional neural networks (CNNs) are used in creating the hidden layers for the VAE since the data is in the form of 2D images. There are other formats of data that can be used to create a dataset and also other generative AI models but for the scope of this thesis only the ones mentioned are used.

4. Shell structures are selected for the evaluation of the generated patterns. In the thesis, the patterns are morphed onto a dynamically relaxed mesh, to accommodate easier customization of topology design ideas for a given form. Other structural designs are not investigated.

5. The comparative aspects of the generated patterns are focused more than the 'allowability' of the results.

6. Optimization is not considered as the application of this thesis focusses on design exploration during the initial design phases.

1.5 Approach and Methodology

The research has 5-stages according to different assessments described below:

Stage 1: Build background knowledge

In the first stage, the emphasis is on learning the concepts of Python, AI, Deep Learning, TensorFlow and Keras, Autoencoders, Variational Auto Encoders, in that order, and other necessary topics. The primary source of learning is online videos and tutorials. This stage is necessary for learning the skills, techniques, and technical know-how for later training an AI model.

Stage 2: Analysis of Precedents and Previous Researches

In the second stage, existing literature on AI in design was studied, focusing on structural design and AI. Dr. Alberto and Ph.D. candidate Gabriele Mirra's paper on 'Comparison between human-defined and AI-generated design spaces for the optimization of shell structures' influenced filtering the focus of the literature search. With their help, an approach to build on their existing research was explored. There is literature available on AI learning from human-defined design space, but there is a need of exploring nature-defined design spaces e.g., natural patterns, and forms. The properties of cellular solids depend directly on the shape and structure of the cells, exploring the shape and topology of the cell walls might prove to be interesting for application in topological optimization. Thus, for this thesis, natural cell structures were selected as the study for creating the dataset for the AI model to train on.

Stage 3: Creating Dataset and Benchmark Testing

In the third stage, the domain of natural cell structures is explored for use as datasets. The selection of the data and its representation play an important role in the performance of the AI model. The task would be to explore the geometry and patterns and decide on the representation of the data for making the dataset and choosing the design variables to construct the design space. Thus, to have better control over the data, the dataset will be artificially created by modeling the patterns of the cellular solids with a grasshopper script. Testing of the AI model will happen simultaneously. The model will be written in Python, using TensorFlow and Keras library and Convolutional Neural Networks. It will be tested with a benchmark pre-existing dataset to assess its performance. This step is important to ensure that the model works correctly before introducing the new dataset. The desired outcome of this stage will be to have a completed dataset and a working AI model to train.

Stage 4: Training and Integration

In this stage, the model will be trained with the new dataset created in Stage 3. During training, the VAE (AI model) will extract implicit design variables from the dataset and generate the design space. As the design space is defined by AI-selected variables, it can make the outcome less predictable and more diverse. The AI-generated patterns, after training, will be used as a UV map for an arbitrary shell for topology optimization. The objective is to compare the structural performance of the generated patterns and verify that the AI-generated designs spaces result in a greater variety of examples while ensuring a still good structural performance. The comparative aspects are focused.

Stage 5: Results and Conclusion

The last stage will focus on documenting the results and concluding the findings. The final report with the reflection and conclusion will be completed.

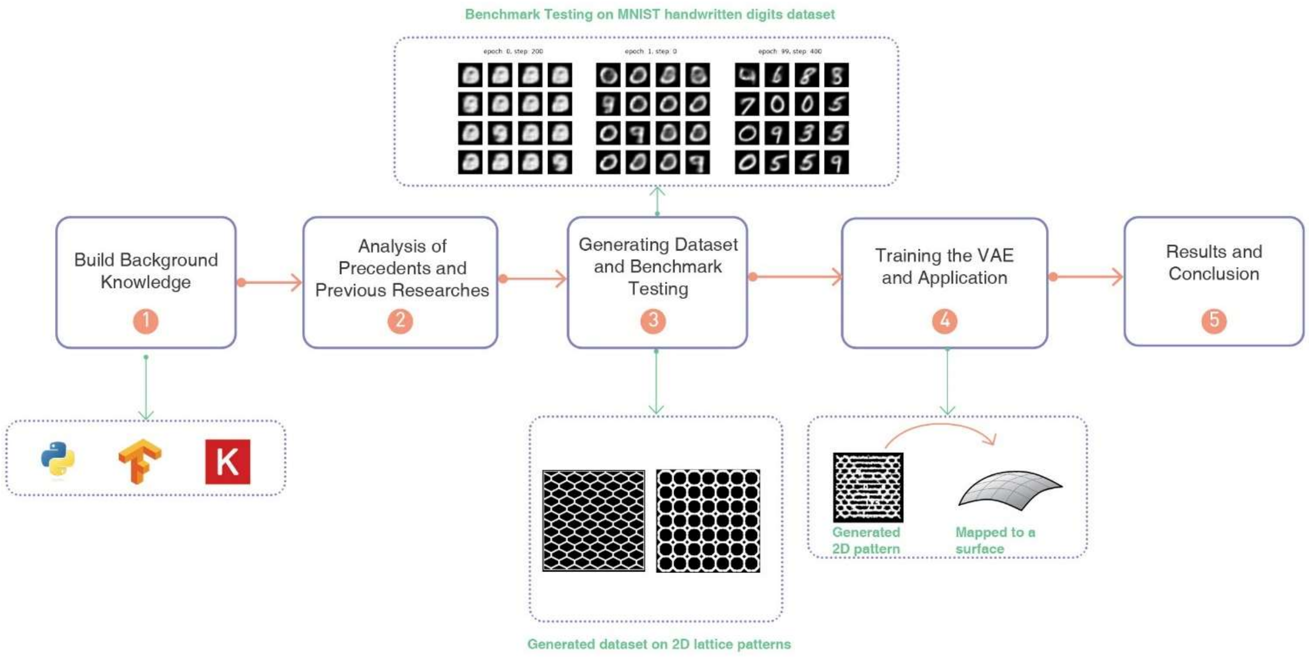

Benchmark Testing on MNIST handwritten digits dataset
epoch: 0, step: 200
epoch: 1, step: 0
epoch: 99, step: 400
Build Background Knowledge 1
Analysis of Precedents and Previous Researches 2
Generating Dataset and Benchmark Testing 3
Training the VAE and Application 4
Results and Conclusion 5
K
Generated 2D pattern
Mapped to a surface
Generated dataset on 2D lattice patterns

Figure 4 Diagrammatic representation of the Research Approach

1.6 Planning and Organization

The Timeline is added as an appendix at the end of the report.

II LITERATURE RESEARCH

2.1 Artificial Intelligence

Artificial intelligence (AI) is a wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence. Machine learning is a subset of Artificial intelligence and deep learning is a subset of machine learning. The following sections are referenced from the book Deep Learning (J. D. Kelleher 2019) . It briefly goes through the sections of deep learning and Neural Networks.

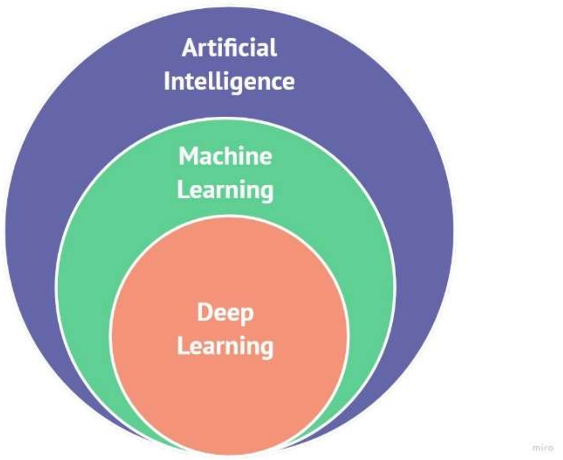


Figure 5 Relationship between Artificial Intelligence, Machine Learning, and Deep Learning

2.1.1 Deep learning (J. D. Kelleher 2019)

Machine learning is teaching computers to recognize patterns in data. Deep learning is a machine learning technique that learns features and tasks directly from data. Inputs are run through "neural networks" which have hidden layers.

It is changing the world as we know it, and many AI breakthroughs we hear about in social media are because of Deep learning. For example, DeepMind's AlphaGo; the go board game; was the first computer program to beat a professional Go player. The Go board game boasts to have more possible moves in it than there are atoms in the universe. This was possible because of deep learning, which increases the computational power extraordinarily.

Deep learning involves the training of neural networks, inspired by the neurons of the human brain. They carry out the following tasks:

i) Take the data as input

ii) Train themselves to understand patterns in the data. They extract the implicit parameters from the data to learn.

iii) Output useful information.

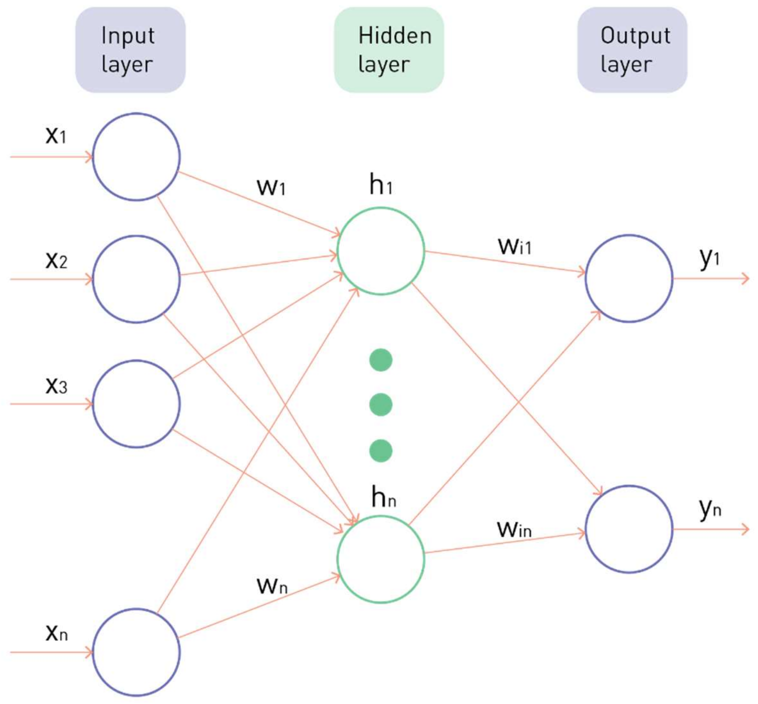


Figure 6 Schematic representation of a Neural Network

The neural networks consist of

x_i: neurons,

w_i: weights of the channels;

h_i: The biases of the neurons in the hidden layer;

y_i: output;

σ: a non-linear function that decides if the particular neuron can contribute to the next layer. It's called the activation function.

The learning process of a Neural network can be broken into two processes:

1. Forward propagation: Propagation of information from the input layer to the output layer.
2. Back Propagation: The reverse of forward propagation. It is the reason why Neural Networks (NN) are so powerful. It allows the NNs to go backward and adjust the initial weights and biases that were randomly assigned in the beginning to get closer to the predicted output and minimize loss.

Terms used in Neural Networks

1. Activation functions:

Introduces non-linearity in the network. It also decides if a neuron can contribute to the next layer. There are different functions available and the uses depend on the AI model. But for this thesis, the ReLU Function (Rectified Linear Unit) will be used.

$R(z) = \max(0, z)$

This function is useful because it uses sparse activation, which means that at a time only 50% of the neurons will activate which helps in the randomness and learning of the NN.

2. Loss functions:

Loss functions quantify the deviation of the predicted output by the neural networks to the expected output.

3. Optimizers:

During training, we adjust the parameters to minimize the loss function and make out the model as optimized as possible. Optimizers tie together the loss function and model parameters by updating the network based on the output of the loss function.

4. Parameters and Hyperparameters:

Model parameters: These are variables internal to the neural network. The values are estimated right from the data. E.g., Weights and biases.

Model hyperparameters: These are variable configurations external to the Neural Networks. The value cannot be estimated right from the data. Usually, they are manually specified in the model. E.g., Learning rate, activation function, etc.

5. Epochs, batches, batch size, and iterations.

These terms are required if the dataset is large, which is usually the case.

Epochs: When the entire dataset is passed forward and backward through the neural network once.

Batch & batch size: large datasets are divided into smaller batches and those batches are fed to the neural network.

Iterations: Number of batches required to complete one Epoch.

2.1.2 Convolutional Neural network (CNN) (J. D. Kelleher 2019)

This Neural network is inspired by the organization of the neurons in the visual cortex of the human brain. It is useful for processing data like images, audio, and video.

The hidden layers in CNN are:

i) Convolutional layers

ii) Pooling layers

iii) Fully connected layers

iv) Normalization Layers

In place of activation functions, convolutional and pooling layers are used.

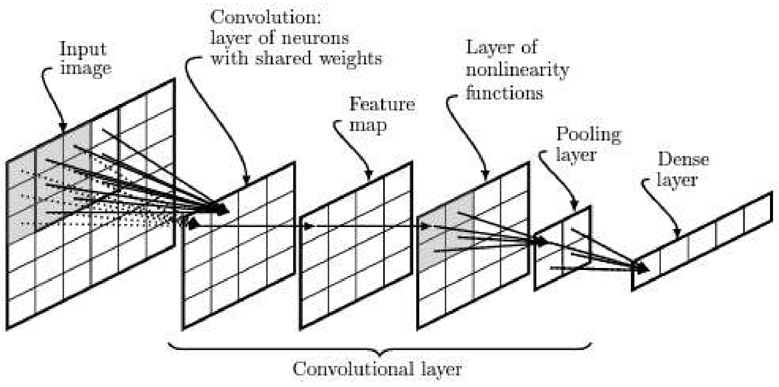

Input image
Convolution: layer of neurons with shared weights
Feature map
Layer of nonlinearity functions
Pooling layer
Dense layer
Convolutional layer

Figure 7 Illustrations of the different stages of processing in a convolutional layer. Note in this figure the Image and Feature Map are data structures; the other stages represent operations on data. (J. D. Kelleher 2019)

Pooling reduces the number of neurons necessary in subsequent layers.

The steps involved in the CNN are: (Dsouza 2020)

Step 1: Take an input image which is a 2D matrix of pixels with typically 3 color channels RGB.

Step 2: Use a convolutional layer with multiple filters to create a 2D feature matrix as an output for each filter

Step 3: Pool the results to produce a down sample feature matrix for each filter in the Convolutional Layer

Step 4: Repeat Steps 2 &3

Step 5: Add a few fully connected hidden layers to help classify the Image.

Step 6: Produce a classification prediction in the output layer

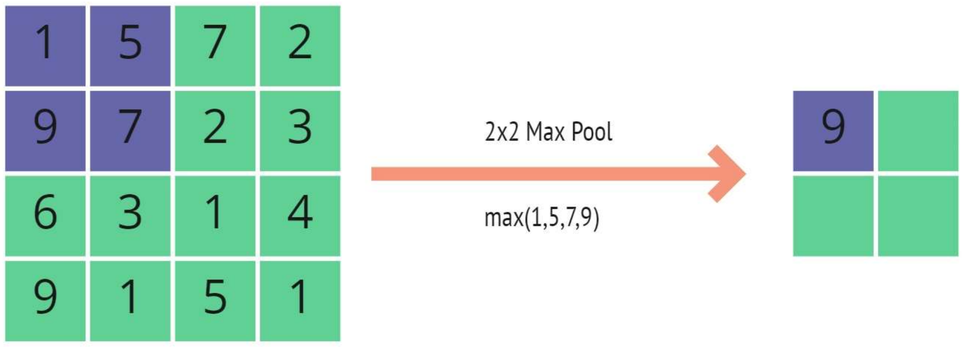


Figure 8 Representation of max pooling: Selecting a max value from a selective region

Creating a deep Learning Model

i) Gathering Data: Picking the right data is the key as bad data can lead to a bad model.

ii) Pre-processing the Data: These are done in various steps of A: Splitting the datasets into subsets. B: Formatting. C: Fixing the problems of missing data. D: Sampling. E: Feature Scaling.

iii) Training Data: This includes the steps of feeding the data, forward propagation, loss functions, and backpropagation.

iv) Evaluation: Test the trained model on a validation set.

v) Optimization: This includes the following steps:

a) Hyperparameter Tuning: Increasing epochs and adjusting learning rates.

b) Addressing overfitting: by getting more data or reducing model size and regularization. Data Augmentation: Artificially increasing the dataset. Dropout: randomly dropping out neurons in the network.

2.1.3 Variational Auto Encoders

The Variational Autoencoder is a kind of deep generative model (Kingma and Welling 2019). A VAE is an autoencoder whose encodings distribution is regularized during the training to ensure that its latent space has good properties allowing us to *generate* some *new data*. Moreover, the term "variational" comes from the close relationship there is between the regularization and the variational inference method in statistics. (Rocca 2019)

Dimensionality Reduction

In machine learning, the features of the input data are reduced for better representation and faster computation. The process can be referred to as "**encoder**" which produces 'new features' representation of old features. The "**decoder**" reverses this process. Dimensionality reduction can also be referred to as data compression of the input data from the initial space to the "**latent space**".

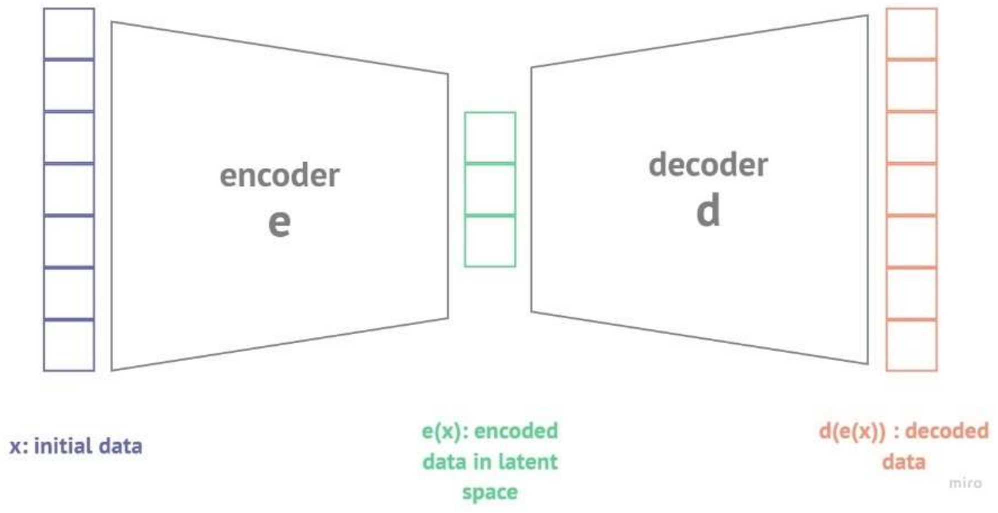


Figure 9 Illustration of the dimensionality reduction principle with encoder and decoder.

The main purpose is to find the best encoder and decoder pair that keeps the maximum information when encoding and has the minimum reconstruction error when decoding.

Autoencoders

Autoencoder sets neural networks as encoders and decoders and uses gradient descent to minimize the reconstruction error.

But with autoencoders, there is no real way of producing new content.

Variational Autoencoders

To be able to generate new data, the latent space needs to be regular enough to facilitate exploration. "A variational autoencoder is an autoencoder whose training is regularized to avoid overfitting and ensure that the latent space has good properties that enable the generative process." (Rocca 2019)

The VAE instead of encoding the input data as a single point does it as a probability distribution over the latent space; such as the mean and variance of a Gaussian. This approach produces a continuous, structured latent space, which is useful for image generation (Kingma and Welling 2019). The loss function of a VAE is composed of a 'reconstruction term' on the final layer and a 'regularization term' on the latent layer. That regularization term is expressed as Kulback-Leibler divergence.

The latent space should have two properties: continuity (two close points in latent space should be similar) and completeness (any point sampled from the latent space should be meaningful). With the regularization term, we ensure that the encoded data are not too far apart in the latent space and encourage some overlap, this creates a 'gradient' over the information encoded in the latent space.

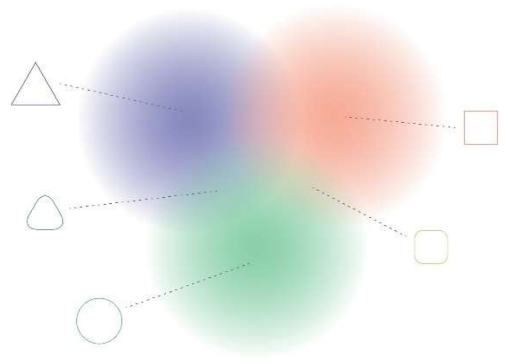


Figure 10 Regularization tends to create a "gradient" over the information encoded in the latent space. (Rocca 2019)

Example 1: (Mirra and Pugnale 2021)

The authors of this paper are the external advisors for this Thesis. The research of this thesis seeks to build on their existing research.

The paper presents a comparison between human-defined and AI-generated design spaces. The domain of shell structures was used to construct the dataset through an explicit definition of their design variables. The human-defined design space was constructed using a superstructure approach. The variables chosen were: A) The number of openings. B) The opening rotation. C) The opening position. D) the opening Width. E) the curvature of the support edges. A 2D depth map was used as the data format.

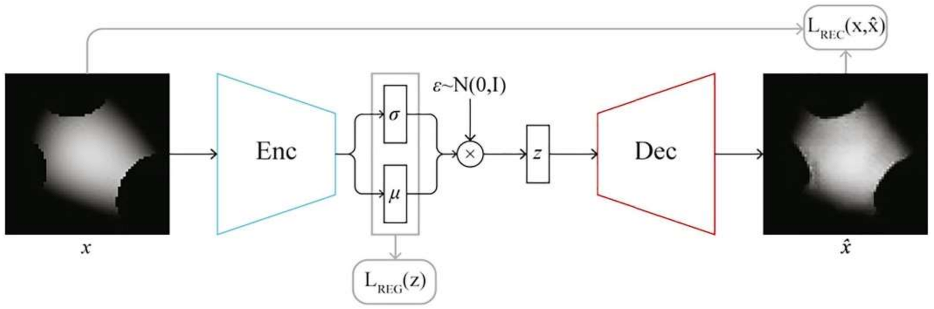


Figure 11 Diagram showing the main components of the VAE model architecture. (Mirra and Pugnale 2021)

The dataset was then used by the VAE to create a generated data space. The model was trained for 5000 epochs. The generative capabilities of the model were tested by using the decoder to generate depth maps from new samples of the Latent space.

The resulting designs were tested in two applications. First, finding optimal design solutions for a target triangular footprint. And the second, when the target footprint was a square.

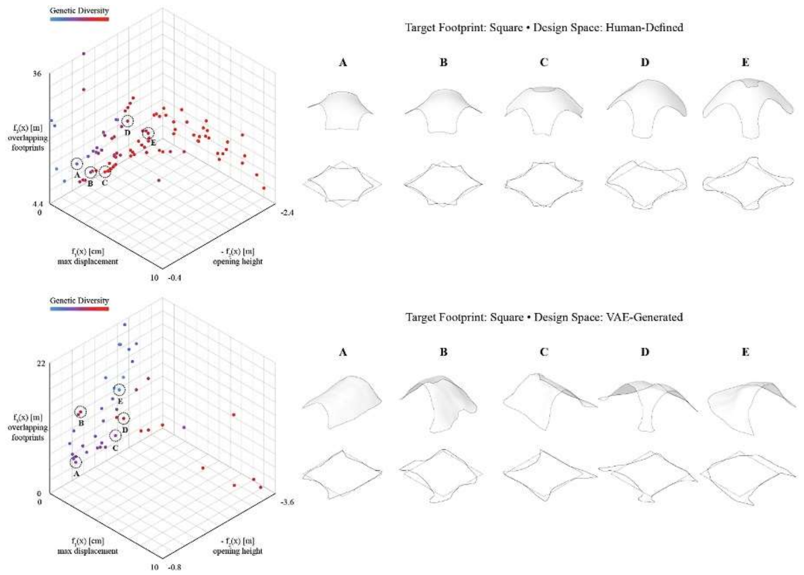


Figure 12 Results of the second application. On the left: representation of Pareto-optimal solutions in the objective space; on the right: axonometric view of the selected designs and their footprints (Mirra and Pugnale 2021)

Learning: This paper presented that the AI-generated design space is flexible and can produce more design variations than those present in the training dataset. This leads to more unpredictable results than ones from human-defined design space, helping designers explore a larger variety of solutions. However, one drawback was noticed that the AI-generated forms were less balanced and had irregular edges and areas.

Example 2: (Renaud and Caitlin T. 2021)

The paper investigates the integration of conditional variational autoencoders in a qualitative exploration of performance-driven design ideas. The research establishes a workflow for systematically exploring the design space. A conditional VAE was used for training instead of a standard one to train on a dataset of performance-driven samples. The performance condition on the VAE can be used after the model is trained to control the decoder's output and also to help it encode and decode more easily.

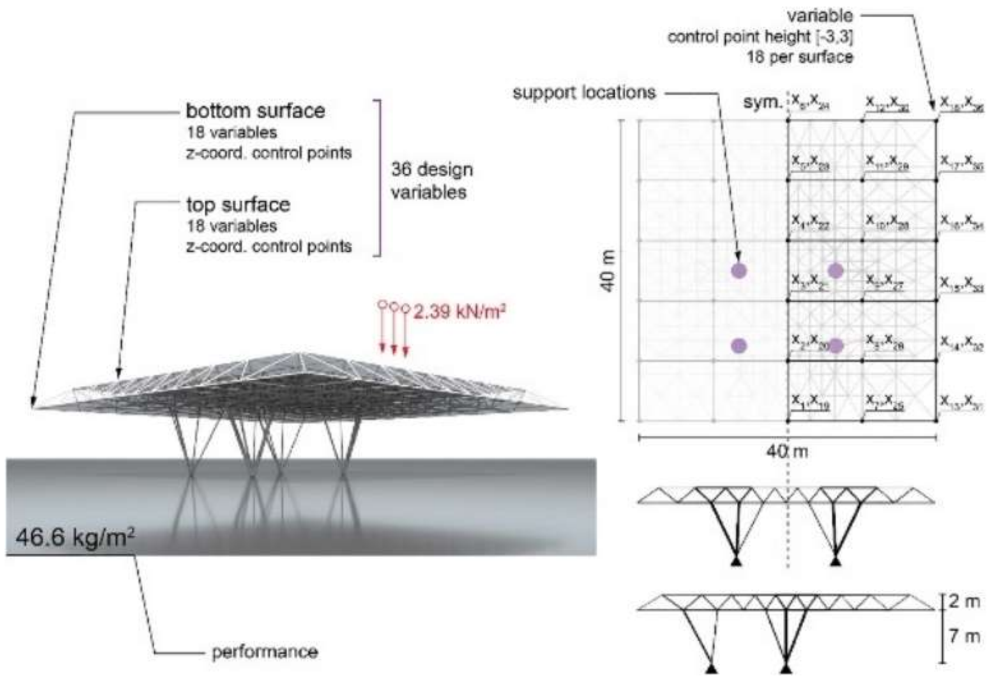


Figure 13 Long span roof example: summary of initial geometry, design variables, and performance measure. Design variables 1 through 18 control the bottom surface and variables 19 through 36 control the top surface. (Renaud and Caitlin T. 2021)

The paper shows the possibility of a latent space visualization that shows both the geometric

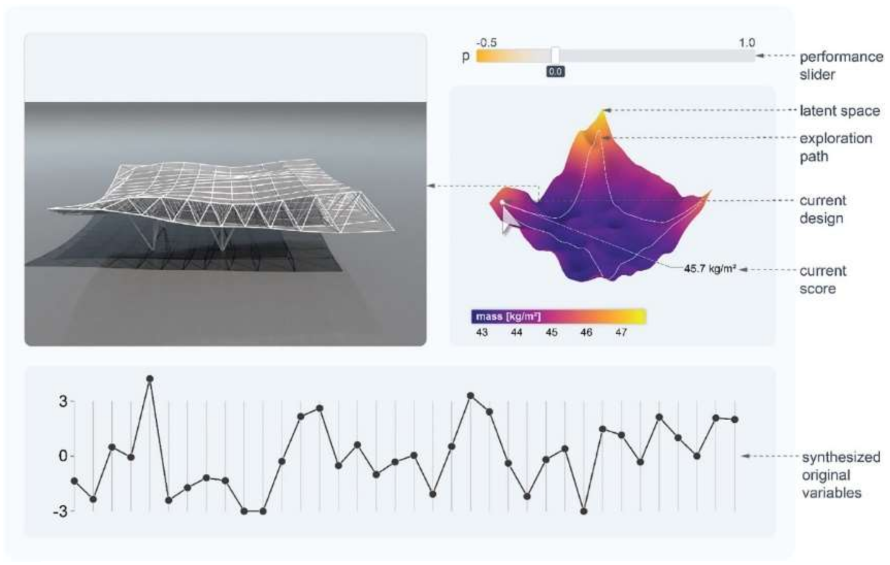


Figure 14 Prototype interface for latent space exploration. (Renaud and Caitlin T. 2021)

diversity and the performance it contains. Fig 14 shows the potential exploration path a designer can take through the latent space. The result shows that this yields diverse high-performing ideas from which the designer can choose.

Learning: This paper presented the workflow for allowing designers to intuitively explore large design spaces. The computer acts as a collaborator in the design process rather than only focusing on quantitative rules. The exploration is focused more on better-informed design options than finding the optimal solution. It would be interesting to find how this workflow can be integrated into existing multi-objective optimization techniques.

2.2 Cellular Solids

Despite an estimated 7 million animal species living on earth,1 there is remarkable repetition in the structures observed among the diversity of biological materials. This is because many different organisms have developed similar solutions to natural challenges (e.g., ambient environmental conditions, predation). As a result, the vast body of research on biological materials often presents similar solutions, since the number of materials available in nature is fairly limited and therefore resourceful combinations of them have to be developed to address specific environmental constraints.

“A cellular solid is made up of an interconnected network of solid struts or plates which form the edges and faces of cells.” (Gibsob and F.Ashby 1997)

There are three typical structures of cellular solids shown in Figure 3 Examples of Cellular solids: a) a two-dimensional honeycomb. b) a three-dimensional foam with open cells. c) a three-dimensional foam with closed cells[Figure 3]. The first is the two-dimensional array of hexagonal cells, which are referred to as *honeycombs.* The three-dimensional cellular materials are called *foams*. For open-celled foams, the solid of which the foam is made is contained only in the cell edges. For closed-cell, the faces of the cell are solid too and are sealed off from their neighbors. (Gibsob and F.Ashby 1997)

2.2.1 Structure (Gibsob and F.Ashby 1997)

Topological laws govern the connectivity of edges and faces of the cell and impose constraints on the dispersion of Cell sizes which help better understand the cell geometry.

Honeycombs:

The honeycombs which have hexagonal cells have edge connectivity of *three*. For honeycombs with square or triangular cells the edge connectivity of *four* or *six*. But they are less efficient as they use more solid to enclose the cells.

Foams:

For foams, the surface tension draws solid into the cell edges and faces, leaving a thin skin framed by thicker edges. The connectivity of the edges and faces of closed cells is harder to establish. If the surface tension is the dominant force that shapes the structure, then four edges meet at 109.4 degrees at each vertex, and three faces meet at 120 degrees at each edge. The foams shown in [Figure 15] are all like that.

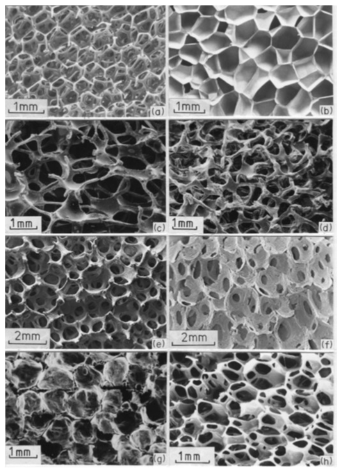


Figure 15 man-made foams: a) open-cell Polyurethane, b) closed-cell polyurethane, c) nickel, d) copper, e) zirconia, f) mullite, g) glass, h) a polyether foam with both open and closed cells. (Gibsob and F.Ashby 1997)

The properties of the cellular solids depend on the way the solid is distributed in the cell edges and faces.

2.2.2. Shape, Size, and Topology (Gibsob and F.Ashby 1997)

A) Cell Shape:

The polygons found in two-dimensional cells are: a) equilateral triangle, b) isosceles triangle, c) square, d) parallelogram, e) regular hexagon, f) irregular hexagon [Figure 17]. any triangle, quadrilateral, or hexagon with a center of symmetry can fill the plane. In three dimensions, a greater variety of cells is possible, the most known shapes are a) tetrahedron, b) triangular prism, c) rectangular prism, d) hexagonal prism, e) octahedron, f) rhombic dodecahedron, g) pentagonal dodecahedron, h) tetrakaidecahedron, i) icosahedron.

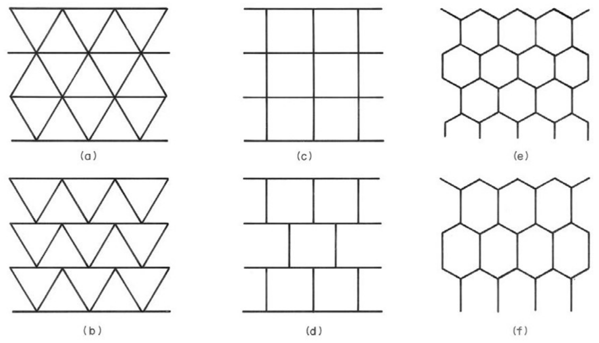


Figure 16 Packing of 2D cells to fill a plane. (Gibsob and F.Ashby 1997)

B) Faces, edges, and vertices.
From a geometric point of view, the cell can be considered as vertices, joined by edges, which surround faces. The number of vertices V, of edges E, of faces F and cells C, are related by Euler's Law (Euler 1746) which states that:

$$F - E + V = 1 \qquad (2D)$$
$$-C + F - E + V = 1 \qquad (3D)$$

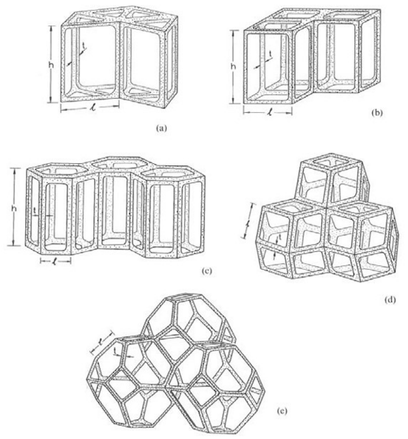


Figure 17 The Packing of Polyhedra to fill a space: a) triangular prisms, b) Rectangular Prisms, c) hexagonal prisms, d) rhombic dodecahedra, e) tetrakaidekahedral (Gibsob and F.Ashby 1997)

Example 1: (Pearce 1978)

This book attempts at providing a basis for the modularity of building components, while taking inspiration from structures in nature. Closest packing is a structural arrangement with inherent geometric stability which is found naturally in a 3-D arrangement of polyhedral cells in Biology. The book explores the possibility of stacking modular components and designing triangulated structures derived from closed-packed cells. Since the book studied the geometry and arrangement of the cellular solids, it was used as literature research to understand the form of the natural solids and possibly the grammar present in modeling them

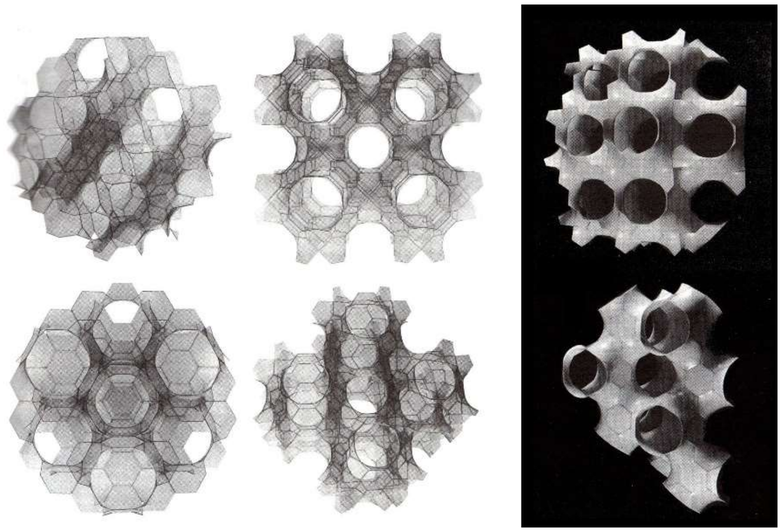


Figure 18 Periodic lattice assemblies (Pearce 1978)

Example 2: (Naboni 2017)

The paper investigated cellular solids as a tectonic system in architecture that is based on a complex form developed for material scarcity. The work explored the application of biological bone microstructure to larger-scale architecture, using computational design and additive manufacturing. Among the cellular solid models, that of a lattice cell was chosen. Cellular lattice structures are composed of an interconnected network of struts, either pin-joined or rigidly bonded at connections. For E.g., in a built environment is a Space truss. A comparative analysis of typical 3D cells was carried out [Figure 19] with an evaluation of printability, relative density, and visual permeability.

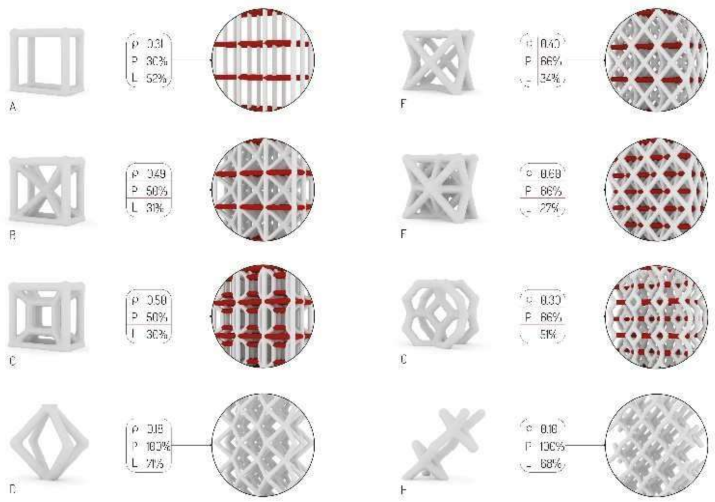


Figure 19 The image shows eight different unit cells typologies for the Cellular Lattice Structure and their observed characteristics; the first column shows unit cell types: A - orthogonal grid, B - star, C - tesseract, D - octahedron, E - cross, F - octet, G – vintiles and H – Diamond.

Example 3: (Xin Zheng 2018)

In this paper, a self-supervised learning approach was presented for the segmentation of white blood cells (WBC). The first module extracts the foreground region of the cell through unsupervised learning using AI. The second module consists of using the results of the first module and actively training the model in which each input sample contains a feature vector and label. The two modules combined make the self-supervised model. The approach of separating the background and foreground of the image and creating the dataset of just the topology of the WBC is interesting and can be utilized in this thesis while creating the dataset.

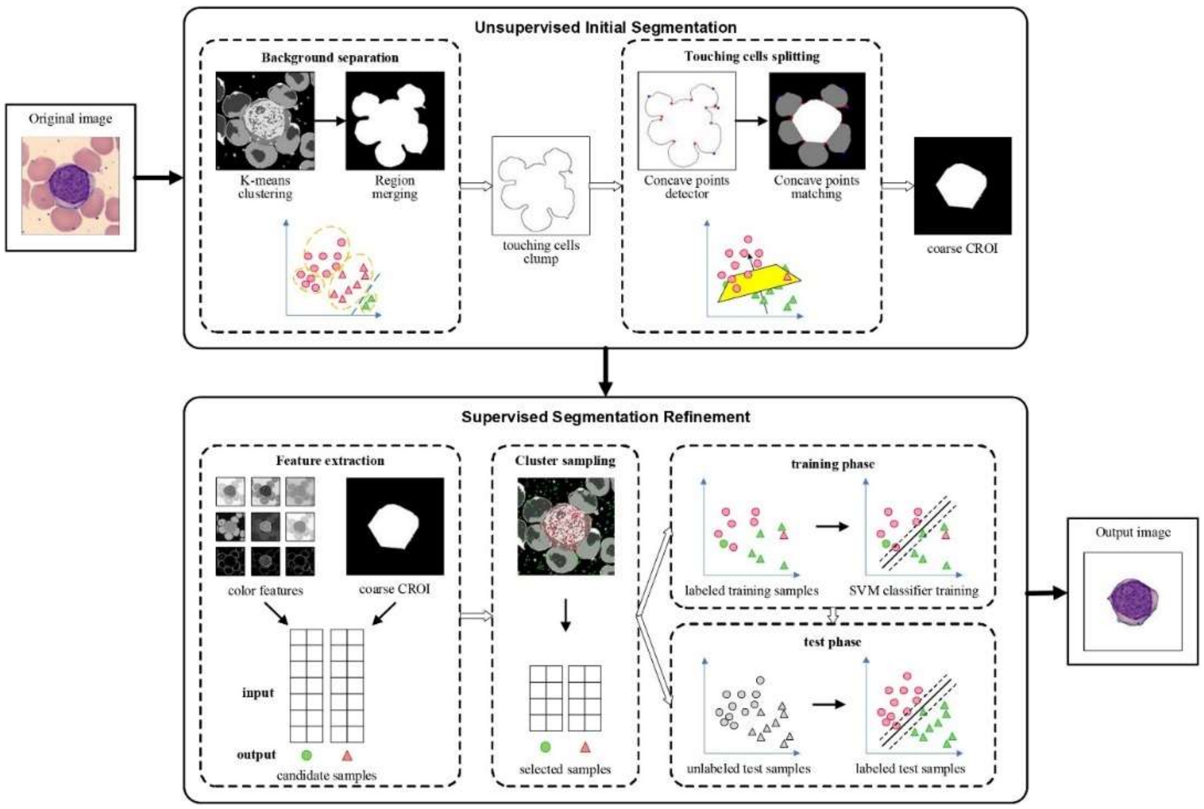


Figure 20 Overview of the Self-supervised learning Approach (Xin Zheng 2018)

2.3 Topology

Topology Optimization of solid structure involves the determinations of features such as the number and location of holes and the connectivity of the domain. (Bensoe 1995). In other words, it is a mathematical method that spatially optimizes the distribution of material within a defined domain. It has been used in the field of civil and mechanical engineering to minimize the amount of used material and the strain energy of structures while maintaining mechanical strength. (Bendsoe 2002). But optimization means that the best design is selected out of the available options for a given set of parameters. This leaves little room for exploration, as the intent is to find the one best solution, rather than explore many feasible solutions.

Shell structures are form-passive structures that resist loads through internal membrane stresses. They are described by three dimensional curved surfaces with one dimension extremely smaller than the other two (Adriaenssens, et al. 2014). In topology exploration of shell structures, the patterns integrate the load bearing systems to be fabricated and assembled (Tam 2021). The topology of a pattern used for a shell structure matters because it sets the boundaries for the available design space, within the general design space of the shell structure. This geometric space, for a given topology may not contain efficient or even feasible designs. Hence, rather than for optimizations, designers should have access to explore topology findings during the conceptual design phases (Tam 2021).

Example: (Oval 2019)

This paper explores the topology findings of patterns for shell structures such as beam grids for grid shells or voussoir tessellations for vaults among others. The geometry of a pattern is explored to achieve diverse design criteria; however, the predefined topology of the pattern depends on the experience of the designer.

The design space of the patterns is created based on: a) The pattern singularity design space, b) The pattern density design space, c) the pattern connectivity space, d) the pattern geometry space.

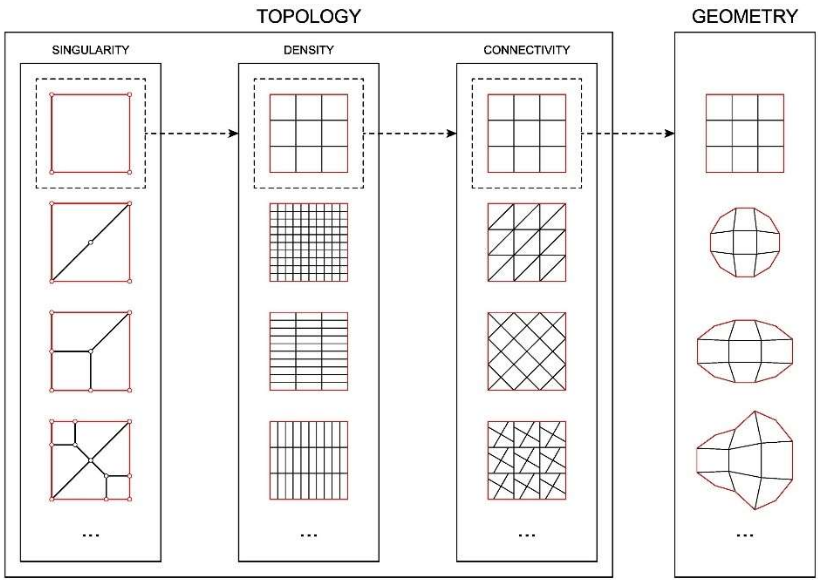


Figure 21 Design space structure of a pattern's singularities, density, connectivity, and geometry, where each design space is defined by the design choices in the upstream spaces. (Oval 2019)

Based on these parameters, modification can be added to the actual pattern. Below is the result of quad-mesh pattern exploration.

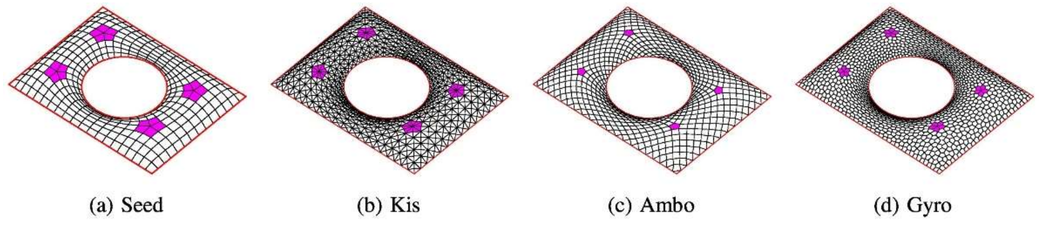


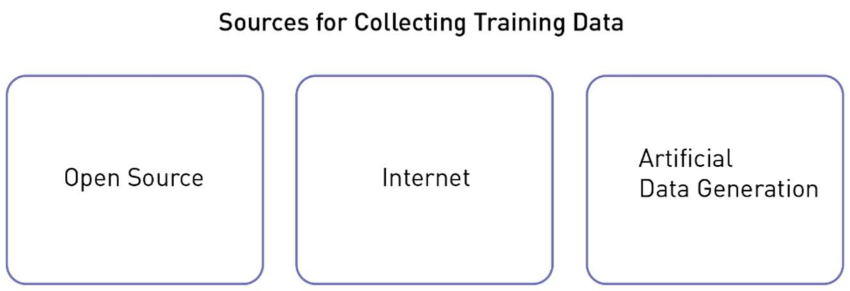
Figure 22 Exploring quad-based pattern symmetries with equivalent vertex or face singularities highlighted in pink by applying different Conway operators on a seed quad mesh. (Oval 2019)

The patterns in this paper are designer generated, which limits the exploration of a design space. But the process can be utilized in this thesis for the exploration of AI-generated patterns on shell structures.

III DATASET GENERATION

Oxford dictionary defines dataset as “a collection of data that is treated as a single unit by a computer”. The first step to training an AI model is to have a dataset of trainable data available. The algorithm uses that data to train and find underling patterns and similarities to learn from. This means that the data needs to made uniform and understandable for the AI model. The following section explains how that is carried out.

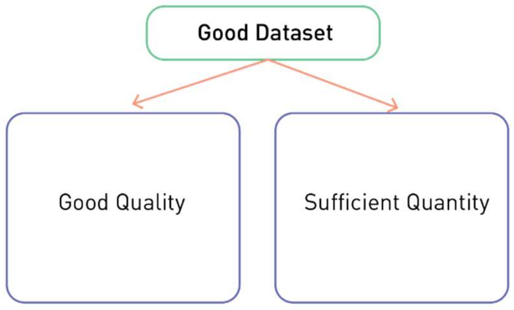
There are different sources for collecting data; open source, internet and artificial data generation. The free available datasets on the internet are useful for testing the working of the AI model, but for specific projects it is more useful to generate the data. It provides more control and directly corelates with the project goals.


Sources for Collecting Training Data
Open Source
Internet
Artificial Data Generation

Features of a good Dataset:

Good quality of the dataset is essential for getting the desired training and output from the trained AI model. This needs to be considered more carefully when generating a custom dataset. While the characteristics of a good quality dataset may vary depending on the goals of the model, some common ones are – Existence of similarity between some samples (uniformity), some recurring features in all the samples (continual).

Sufficient quantity ensures that there is enough data for the AI model to train correctly. If there is less data available it might lead to overfitting which is the model memorizing the data rather than learning. This prevents the model from producing enough high-quality results.



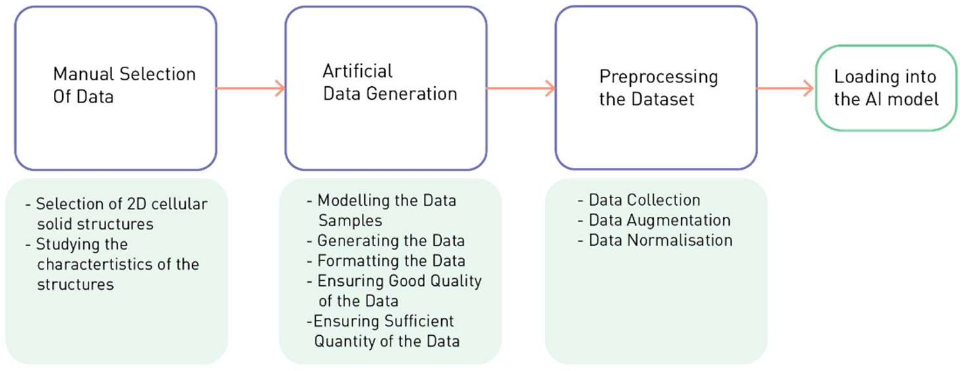

Manual Selection Of Data
Artificial Data Generation
Preprocessing the Dataset
Loading into the AI model
- Selection of 2D cellular solid structures
- Studying the charactertistics of the structures
- Modelling the Data Samples
- Generating the Data
- Formatting the Data
- Ensuring Good Quality of the Data
-Ensuring Sufficient Quantity of the Data
- Data Collection
- Data Augmentation
- Data Normalisation

Figure 23 Summary of Section III

This thesis uses an artificially generated dataset from cellular solid structures. The flowchart explains the content of this section which explains the process of selecting the data, generating the dataset samples and, formatting the dataset.

3.1 Manual Selection of the Data

There are visible regularities of form found in nature. These regularities in mathematics find explanation in fractals, topology, logarithmic spirals and other mathematical patterns. The Cambridge dictionary defines patterns as "any regularly repeated arrangement, especially a design made from repeated lines, shapes, or colors on a surface". These patterns are observed in the macro as well as the micro scale in nature. In micro scale, for example, we can observe patterns in cellular solids. Their study offer insight into advantages in structural design such as improved performance while maintaining structural integrity and decreasing weight (Schaedler 2016). From studying their deformation mechanism, cellular solids are classified into two main groups: bending- dominated and stretching-dominated (M. Benedetti 2021). The stiffness of a body is its resistance to deformation in response to applied force (Baumgart 2000). The stiffness of the cellular solid structures depends on a) the mechanical properties of the structures, b) the topology and cell geometry, c) the relative density (Arredondo-Soto M 2021). The stiffness of such shapes can be tailored by modifying the shape of the patterns, while some shapes are better suited for increasing stiffness, others are more flexible. Cellular structures have been studied extensively in shape morphing applications as it provides stiffness to not deform under out-of-plane loads as well as flexibility for morphing with actuation. For this application, 2D cellular structures have been suggested for general morphing skin application, for example, 2D lattices (Arredondo-Soto M 2021).

Lattice materials are a subclass of cellular solids which are generated by tessellating a unit cell either in a plane or space. If a unit cell is tessellated throughout a plane it is known as a 2D lattice (Tekõglu 2017). They are open cell cellular solids where the shapes are formed by a network of interconnected slender struts in place of surfaces which form the edges and faces of the cells. Lattice structures have high stiffness-weight ratio because of which they have been extensively used in the lightweight applications and additive manufacturing to reduce the mass while maintain or even increasing the stiffness levels. Some examples of such research can be read in the following books and papers: (Nguyen, Park and Rosen 2012), (Kantareddy, et al. August 2016), (Cheng 2017), (Seharing 2020). In theory, 2D lattice material possess large in plane passive stiffness and strength (Tekõglu 2017) . In the papers mentioned above, most of the work is focused on optimization of the configuration of the unit cell of the lattice to get a higher stiffness and structural stability with mass reduction.

3.1.1 Characteristics of the Patterns

For simplicity lattice structures that have been selected for the generating the dataset would be referred to as lattice patterns in this paper. As previously mentioned 2D lattices have been studied for skin morphing applications, and for this thesis, the application that is explored is morphing the generated pattern to a shell structure, hence for creating the dataset only 2D lattice patterns have been chosen. As topology and cell geometry are properties that effects their stiffness, studying lattices with different variety of configurations would create a plausible dataset for training the AI. As the patterns of the lattices are informed of their mechanical properties, the intension is that the generated patterns from the model would also have some mechanical properties translated. Certain topologies are preferred depending on the stiffness objective of the model, to maximize stiffness a stretch dominated lattice is preferred, whereas if flexibility is needed, a bending dominated lattice is preferred (Arredondo-Soto M 2021).

Data samples of lattice patterns of cell structures that are both increasing stiffness and reducing stiffness (flexible) are used to train an AI model to generate new patterns that might have properties of both. Six lattice patterns that are most commonly studied were selected from reviewing papers ((Arredondo-Soto M 2021), (Naboni 2017)) on their properties and they are categorized on the basis of their stiffness.

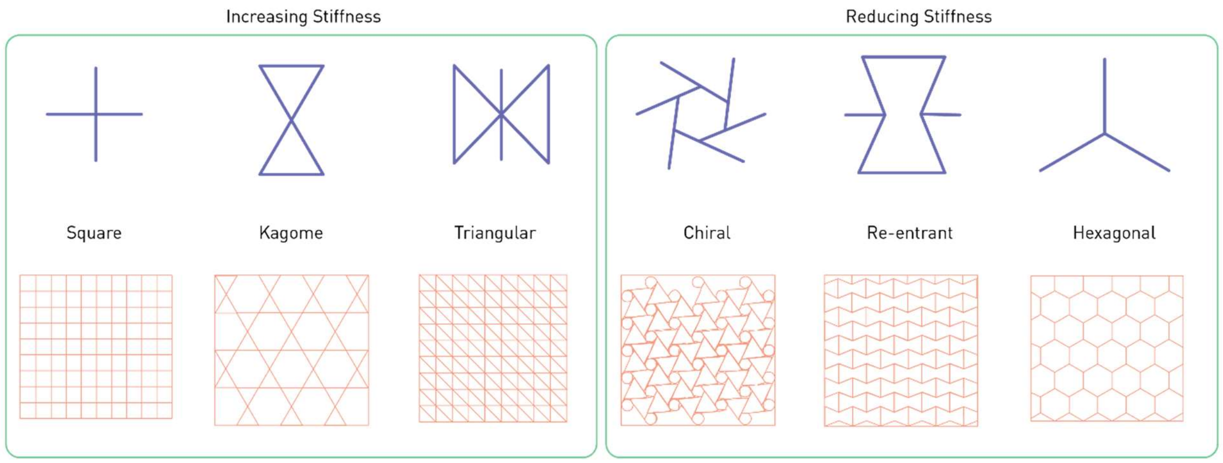


Figure 24 2D lattice patterns for increasing stiffness are: Square (Mihai, Alayyash and Wyatt 2017), Kagome (Lipperman, Fuchs and Ryvkin 2008), Triangular (Lubombo and Huneault 2018). 2D lattice patterns for reducing stiffness are: Chiral, Re-entrant, Hexagonal.

For the scope of this thesis, the focus will be on generating these patterns artificially and formatting the output such that it can be used as an input for the VAE. The detailed mechanical properties of these lattice patterns can be found in the paper: (Arredondo-Soto M 2021) . The selected lattices are shown above, which are Square, Kagome, triangular, Chiral, Re-entrant and Hexagonal.

3.2 Model Set-up for the Patterns

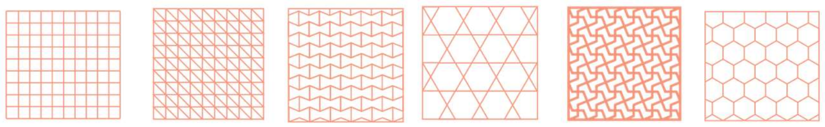
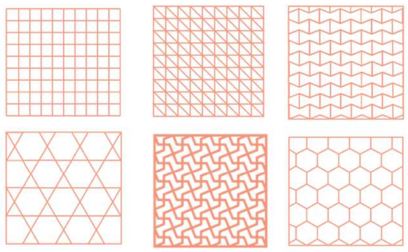
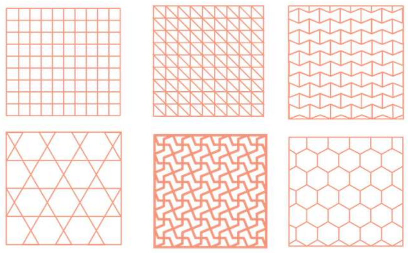
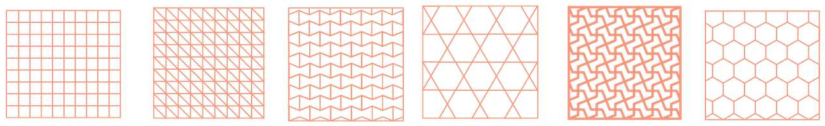
There are different formats of data samples that can be used to create a dataset for training an AI model. In generating the dataset from lattice patterns, images would be appropriate in translating the characteristics of the patterns and also easier to artificially create from a model of the patterns. For this thesis, the format of data that will be used are images in greyscale. This is similar to the MNIST handwritten digits dataset and fashion dataset (Yann, corinna and Christopher J.C. 1998). The testing of the VAE model was done on MNIST handwritten digits dataset so it would be convenient to keep a similar architecture for a similar format of the dataset.

For a good dataset the data needs to be continuous and also uniform. Generating a dataset from a parametric model ensures that those criteria are fulfilled. The parametric approach involves defining a set of variables which can control the features of the data samples. This ensures that the generated data have some similarities in their features as well as that the recurrence of underlying features. The variables chosen to construct the dataset were 1) Type of the 2D lattice selected, 2) the U and V divisions of the patterns, 3) the Thickness of the patterns, 4) Transformation of the angle of the joints in the patterns that would not affect its shape.

A parametric model created in Grasshopper (Rhinoceros) was used to create the 3D model for the patterns of the lattices. The first part of the parametric model was used to create the models bounded by the range of variables that were selected. These are described in the Table 1 below. The bounds of the variables were selected to control the total number of variations of the patterns for each unique pattern. To ensure that the data samples were consistent and not biased, the number of iterations for each pattern were kept the same which was 198 for hexagonal, chiral and re-entrant patterns, 126 for square and triangular, 176 for Kagome. The intension was to keep the total iterations of the patterns between 100-200, and increase the population of the data with data augmentation later.

The process of generating the lattice patterns for hexagonal, square, re-entrant and triangular are similar and is explained in details with the example of the square lattice. The parametric process in grasshopper of all the patterns are available in the appendices. The Kagome and the square chiral patterns are also described in details in the text.

Table 1 Variation Domain of the design variables for generating the dataset

Variables	Application			
1) Types of patterns	 Square, Triangular, Re-entrant, Kagome, Square-Chiral, Hexagonal			
	Patterns Applied to	Variable		Domain
2) Size of the Patterns		Square	U	[5, 10]
			V	[5, 10]
		Triangular	U	[5, 10]
			V	[5, 10]
		Re-entrant	U	[10,12]
			V	[10,12]
		Kagome	U	[5, 8]
			V	[6, 9]
		Square-Chiral	U	[8,10]
			V	[8,10]
		Hexagonal	U	[8,10]
				[8,10]
3) Thickness		Square	(t)	[50,60]mm
		Triangular	(t)	[50,60]mm
		Re-entrant	(t)	[50,60]mm
		Kagome	(t)	[50,60]mm
		Square-Chiral	(t)	[50,60]mm
		Hexagonal	(t)	[50,60]mm
4) Parameter for Transformation		Re-entrant	(a)	[0.6,0.7]
		Square-Chiral	(a)	[0.5,0.6]
		Hexagonal	(a)	[0.2,0.3]
Total number of Iterations	 Square 126, Triangular 126, Re-entrant 198, Kagome 176, Square-Chiral 198, Hexagonal 198			

3.2.1 Square Lattice Patterns

The first part of the parametric model is to generate the iterations of the data for a specific pattern. Generating the models for the square, hexagonal, re-entrant and triangular lattices patterns use the grasshopper plugin called lunchbox, which had pre-existing components for panels of those geometry. To create the patterns, a surface is taken as the base. The surface is then sub divided into the panels using the grasshopper plugin according to the required pattern. After getting the curves of the pattern, it then given an offset which will create its thickness. The offset distance can be controlled so the thickness of the pattern can be altered. After this step the offsets are used as boundary curves to create a surface for the pattern. The last step in the modelling process is creating a mesh and custom previewing it with a white swatch in a black background of rhinoceros. Figure 25 shows a detailed workflow for generating a square lattice pattern in grasshopper. The design variables for square patterns are the number of divisions or the size of the patterns (the higher the division, the smaller the patterns) and the thickness (which is the offset distance of the curves).

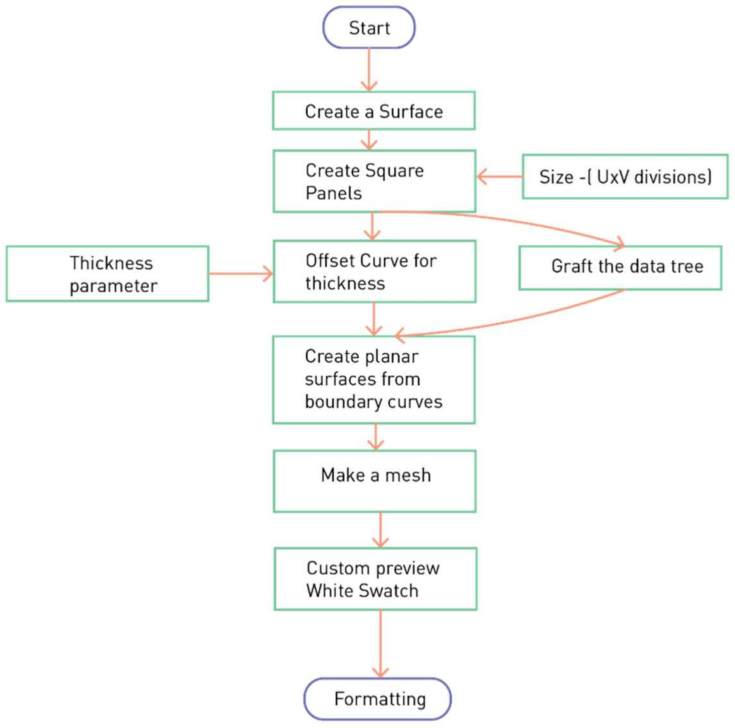


Figure 25 Workflow for Generating Square lattice Patterns

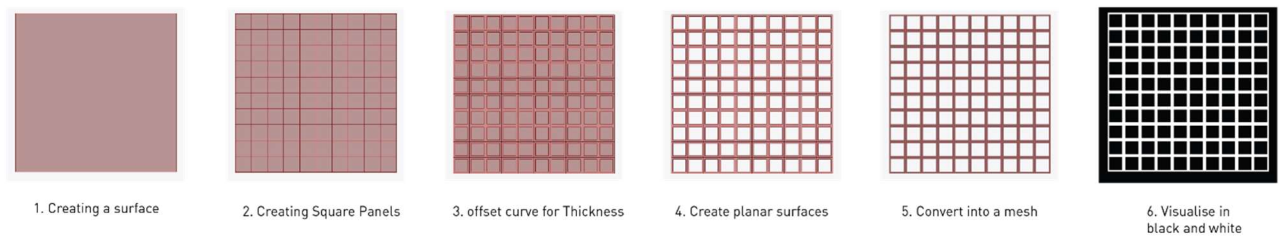

1. Creating a surface
2. Creating Square Panels
3. offset curve for Thickness
4. Create planar surfaces
5. Convert into a mesh
6. Visualise in black and white

Figure 26 Generating a Square lattice pattern

Figure 26 shows the sequence of generating the square lattice patterns, after this step the output is then formatted to be of the correct resolution and all the iterations of the data are recorded to create the data samples for the dataset

3.2.2 Square – Chiral Lattice patterns:

In the paper by (Körner, Liebold-Ribeiro and Yvonne. (2015).) they explored square chiral patterns, the quadratic mode 9 of the periodic boundary conditions lattice was selected as the basis for the pattern of the chiral for this thesis. As the details of the parameterization are not extensively available for that pattern, a process to generate it while studying the connections was developed. The square chiral can be generated from a square module, and then arrayed to create the pattern. The first step is to create a square with the length and width as inputs. The square is then scaled down with reference to its center point. The geometry of the chiral is such that the line segments from the corner of the larger square connect to first, the mid-point of the corresponding rotated edge of the smaller square and then finally it extends to the center point of the inner square. The rotation of the inner square provides different configuration of the pattern. After the module of the square-chiral pattern is completed, it is arrayed to create the pattern. The next step is to add the thickness which is done by first extracting the surfaces and the boundary curves from the array and then offsetting them the required distance to generate the thickness. Figure 27 shows the workflow for generating the pattern in grasshopper in more details.

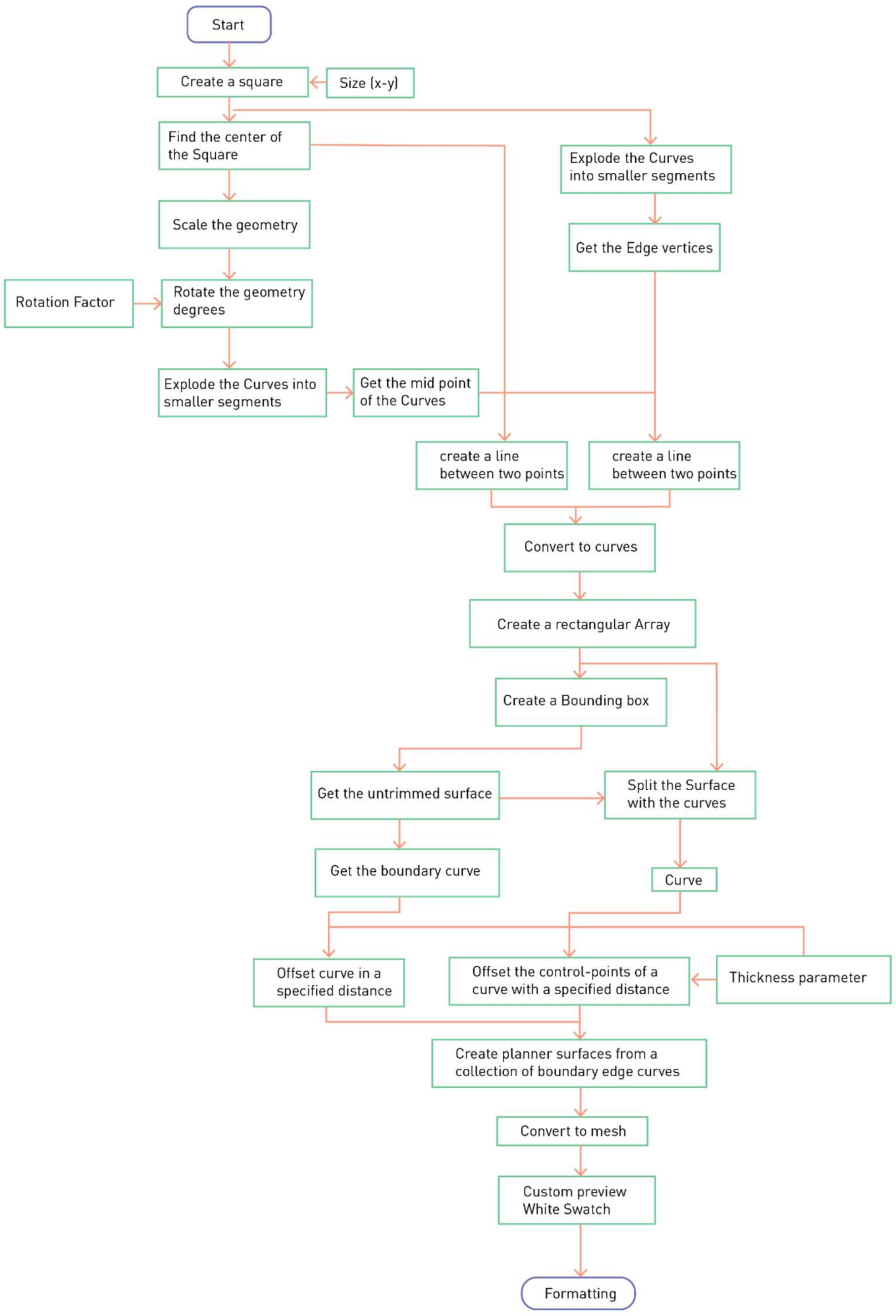

Start
Create a square
Size (x-y)
Find the center of the Square
Explode the Curves into smaller segments
Scale the geometry
Get the Edge vertices
Rotation Factor
Rotate the geometry degrees
Explode the Curves into smaller segments
Get the mid point of the Curves
create a line between two points
create a line between two points
Convert to curves
Create a rectangular Array
Create a Bounding box
Get the untrimmed surface
Split the Surface with the curves
Get the boundary curve
Curve
Offset curve in a specified distance
Offset the control-points of a curve with a specified distance
Thickness parameter
Create planner surfaces from a collection of boundary edge curves
Convert to mesh
Custom preview White Swatch
Formatting

Figure 27 Workflow for generating square-chiral lattice patterns

3.2.3 Kagome Lattice patterns

Kagome is a 2D lattice of corner sharing triangles, and in macro scale it is also a popular type of Japanese weaving pattern. The patterns have been studied for creating the interesting lattice structures (Ayres (2018).). The geometry of the pattern generated in the parametric model is the same way the pattern in made during weaving. The first step is creating diagonals on the surface of the plane. The geometry is then divided by vertical line segments which pass through the intersection of the diagonals, this is to get the height of the triangles that make the corners of the Kagome pattern. Points on the line segments at 0.25 and 0.75 are identified (the height of the triangles will be 0.25 units). The points are then horizontally connected to create the base of the triangles at the corners. After the base geometry of the Kagome pattern is completed, the curves are then given an offset for thickness and connected to a custom preview with a white swatch.

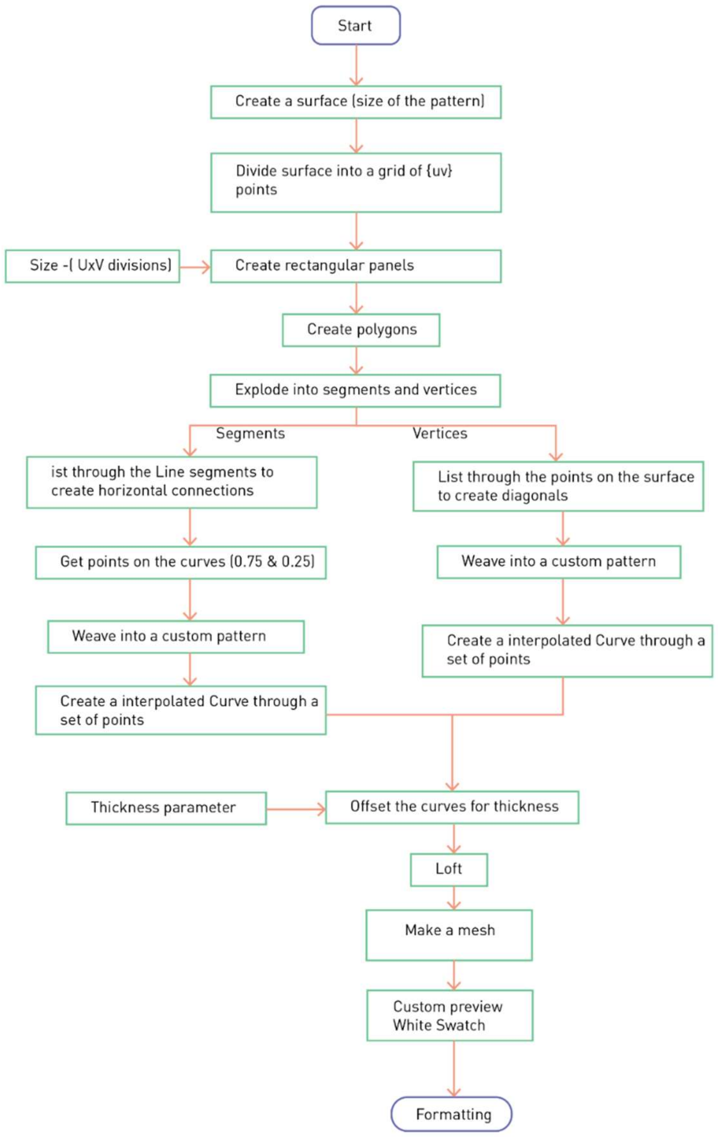

Start
Create a surface (size of the pattern)
Divide surface into a grid of {uv} points
Size -(UxV divisions)
Create rectangular panels
Create polygons
Explode into segments and vertices
Segments
Vertices
ist through the Line segments to create horizontal connections
List through the points on the surface to create diagonals
Get points on the curves (0.75 & 0.25)
Weave into a custom pattern
Weave into a custom pattern
Create a interpolated Curve through a set of points
Create a interpolated Curve through a set of points
Thickness parameter
Offset the curves for thickness
Loft
Make a mesh
Custom preview White Swatch
Formatting

Figure 28 Workflow for generating Kagome lattice patterns

3.2.4 Formatting the Data Samples

After completing the models for the lattice patterns, the patterns are custom previewed with a white swatch on a black background to create a black and white preview. Grey scale images have less information in their pixels than colored images. This ensures that the VAE model has less information to process, similar to an MNIST handwritten numbers (Yann, corinna and Christopher J.C. 1998) dataset which is also in greyscale. The next step is to format the image exports to required resolution; the higher the pixels, the more complex the VAE architecture needs to be to encode and decode. The handwritten dataset has a resolution of 28 x 28, since the patterns are more complex than images a resolution of 64x64 was chosen.

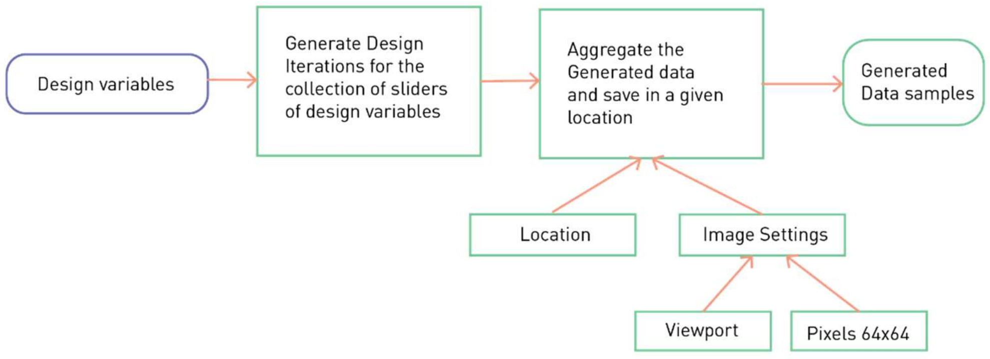

Design variables
Generate Design Iterations for the collection of sliders of design variables
Aggregate the Generated data and save in a given location
Generated Data samples
Location
Image Settings
Viewport
Pixels 64x64

Figure 29 Flowchart for the process of generating the formatted data samples images of patterns)

Figure 29 Shows the process of exporting images all the iteration of the lattice pattern models in the correct resolution for use in the dataset. Since only the viewport and the resolution can be used as an input, the placement of the pattern within the 64x64 resolution needs to be manually adjusted and can be done with trial runs during the export [Figure 30]. Data samples are the exported iterations of the lattice patterns which will constitute the dataset.

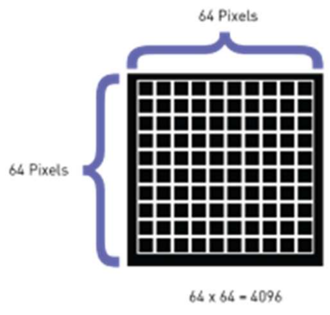


Figure 30 Size of the formatted Lattice pattern images

Figure 31 Shows an iteration of all the lattice patterns exported with the specified resolution. As seen in the picture, some patterns are denser than the others, but since the data samples will be augmented to scale up and crop the final dataset will have uniform samples of both sparse and dense patterns.

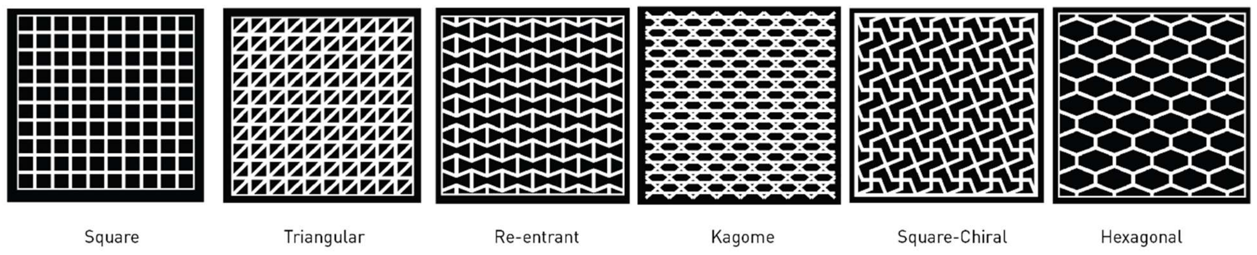


Figure 31 examples of all the Lattice pattern exported as 64x64 resolution greyscale images

All the patterns have iterations of combination of the different design variables and an example of that can be seen in Figure 32 . For the same pattern, through different values for angles, thickness and size, similarity and continuity can be achieved within the data samples.

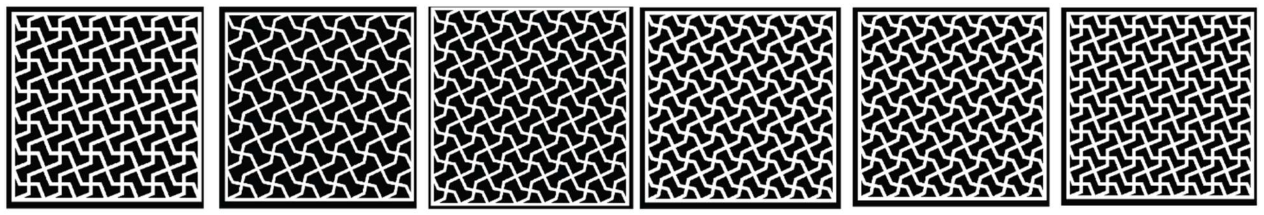


Figure 32 A few Iterations of the Square-Chiral lattice pattern

After completing the exports of all the data samples and their iterations, the images are uploaded into the google drive. Google Collab is used as the coding platform for the VAE model and since it’s a remote computing platform, the files are accessible through google drive.

3.3 Preprocessing the Dataset

The MNIST handwritten digits dataset has about 70,000 greyscale images of handwritten digits which are 28x28 pixels. Having a large dataset ensures that the AI can learn faster as it has more information to learn from. The challenge with generating an artificial dataset is that the quantity might not be enough to properly train an AI model. The total number of data samples that were generated from the lattice patterns were 1022, which even though is a large number, studying other AI models it was inferred that around 4000 or more would be better suited to train the VAE model. This can be done by data augmentation. After data augmentation the dataset needs to be reshaped and normalized and then split into training and validation datasets. These steps are explained in this section.

3.3.1 Data Augmentation

Deep convolutional Neural networks run well when it has access to big data, in order for it to train properly and not overfit. However, large amounts of data are not available readily for custom projects or in cases where the data is artificially generated (as in the case of this thesis). Data Augmentation provides a solution to the problem of limited data (Shorten 2019). There are different data augmentation algorithms available on image manipulations, like cropping, flipping, color space transformation etc. For this thesis the geometric transformations that were chosen were cropping the images in the factor 0.5 and 0.25 and then flipping the images horizontally and vertically.

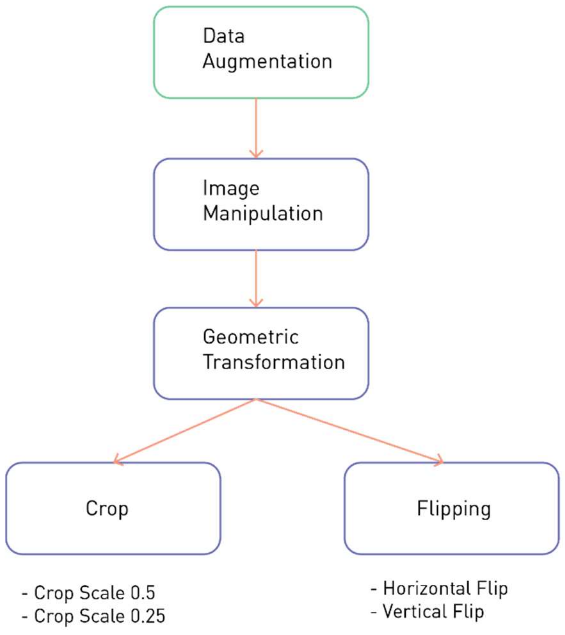


Figure 33 Image data augmentation techniques

Figure 33 Shows the image data augmentation techniques used in this thesis. There are keras functions as well as TensorFlow methods that can be used for data augmentation but they are compatible with predefined Sequential models in their library. To get better control over the augmentation, OpenCV library was used. The dataset is loaded from the google drive, which is shown in the following code snippet:

```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)

dataset_path = '/content/gdrive/MyDrive/Thesis/Dataset2'
```

First all the image paths from the google drive folder containing the dataset was accessed and called into an open list images_dataset []. As shown in the following code snippet:

```
from google.colab.patches import cv2_imshow
images_dataset = []
images_path = glob.glob("/content/gdrive/MyDrive/Thesis/Dataset2/*.png")
for img_path in images_path:
  img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
```

The augmentations were carried out in a loop. All the image augmentations were then appended into the images_dataset [] so that it could be used as the new dataset with more data from the data augmentation. A snippet of the code to scale the images by a factor of 0.5 is shown below. Adapted from (Woshicver 2022):

```
scale = 0.5
  height, width = int(img.shape[0]*scale), int(img.shape[1]*scale)
  x = random.randint(0, img.shape[1] - int(width))
  y = random.randint(0, img.shape[0] - int(height))
  cropped = img[y:y+height, x:x+width]
  resized = cv2.resize(cropped, (img.shape[1], img.shape[0]))
  images_dataset.append(resized)
```

The total number of data samples in the new augmented dataset was 4089. This can be increased further with more augmentations, but due to time constrain this was selected as the final dataset.

3.3.2 Data Reshaping and Normalization

To be used in Keras, the dataset needs to be reshaped to a 4-dimensional NumPy array from a 3-dimensional array. After the reshaping the data needs to be normalized. Normalization is changing the range of pixel intensity values (SelectStar 2020). It is required to normalize the data for training in neural networks, this is done by dividing the images by the max RGB values= 255. The reshaping and the normalization are defined as a function and is shown in the code snippet below. Adapted from (Dutta 2021):

```
def map_image(image):
  '''preprocesses the images'''
  image = tf.cast(image, dtype=tf.float32)
  image = tf.image.resize(image, (IMAGE_SIZE, IMAGE_SIZE))
  image = image / 255.0
  image = tf.reshape(image, shape=(IMAGE_SIZE, IMAGE_SIZE, 1,))
  return image
```

3.3.3 Splitting the Data

In training a VAE model, the dataset is split into training dataset and the validation dataset. The ratio is 80:20 which is the most commonly used. Training datasets are essentially the data from which the VAE model learns from. It used it to start seeing and learning the underlying patterns of the data and use it to make predictions. The validation dataset is used to validate the progress of the training and adjust and optimize. It also helps check if the model is overfitting.

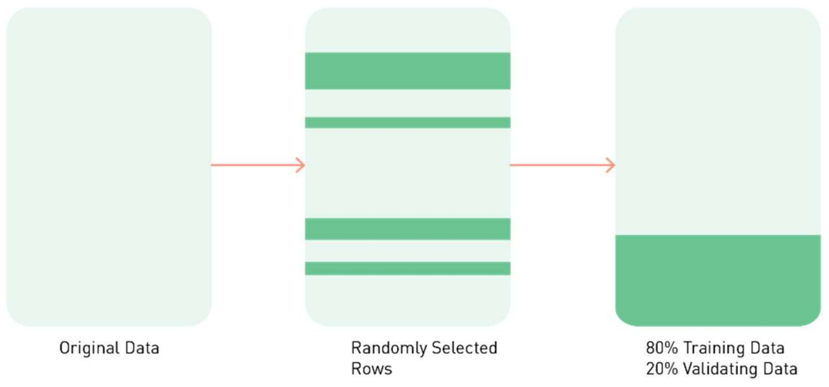


Figure 34 Training data vs validation data

The code for splitting the dataset is shown below (Dutta 2021):

```
# split the paths list into to training (80%) and validation sets(20%).
paths_len = images_dataset.shape[0]
train_paths_len = int(paths_len * 0.8)

train_paths = images_dataset[:train_paths_len]
val_paths = images_dataset[train_paths_len:]

# load the training image paths into tensors, create batches and shuffle
training_dataset = tf.data.Dataset.from_tensor_slices((train_paths))
training_dataset = training_dataset.map(map_image)
training_dataset = training_dataset.shuffle(1000).batch(BATCH_SIZE)
#training_dataset = training_dataset.map(augment)
# load the validation image paths into tensors and create batches
validation_dataset = tf.data.Dataset.from_tensor_slices((val_paths))
validation_dataset = validation_dataset.map(map_image)
validation_dataset = validation_dataset.batch(BATCH_SIZE)
```

IV TRAINING THE VAE ON THE DATASET

For Autoencoders, the decoder samples from the latent variables. In Variational autoencoders (VAEs) the decoder takes the sampling from a gaussian distribution. The samples from the input dataset are used as a mean and variance of the gaussian distribution with some added noise which the decoder uses to sample from and that is the reason why new images can be generated from this architecture. The architecture of the VAE model used in this thesis as well as the steps involved in creating a VAE is explained in detail in the following section.

4.1 Latent Space Sampling

For VAEs there is an additional sampling step in the latent space between the encoder and the decoder. This section is informed from the book by (Cheong 2020) where the gaussian distribution is explained in detail. This section will provide concise relevant information from the book that is important to get an overview of the VAE. The encoder provides the mean and the standard deviation for the gaussian distribution. Neural networks are trained using back propagation, but sampling is not back-propagatable and hence not trainable. To avoid this problem, a reparameterization trick is used where a gaussian distribution is sampled and then the standard deviation from the encoder is multiplied and the mean is added to it.

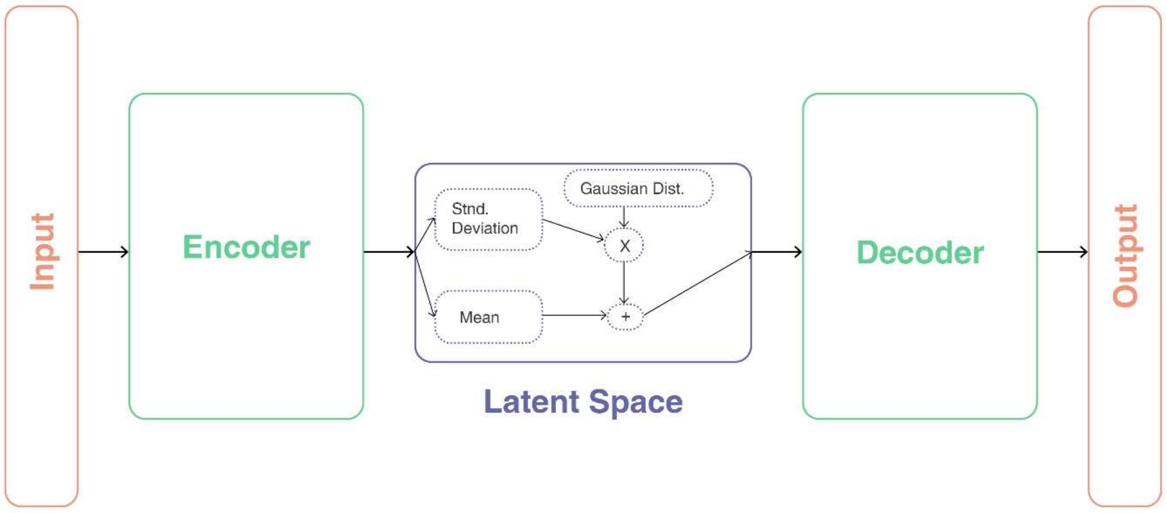


Figure 35 Gaussian Sampling in a VAE

For the VAE the encoder now needs to output the mean and the standard deviation to be used in the sampling layer, this creates a change in its architecture than an autoencoder's encoding layer. The encoder has two convolutional layers with batch normalization then the output is flattened and mu and sigma values are calculated. The encoder layers are shown in Figure 36

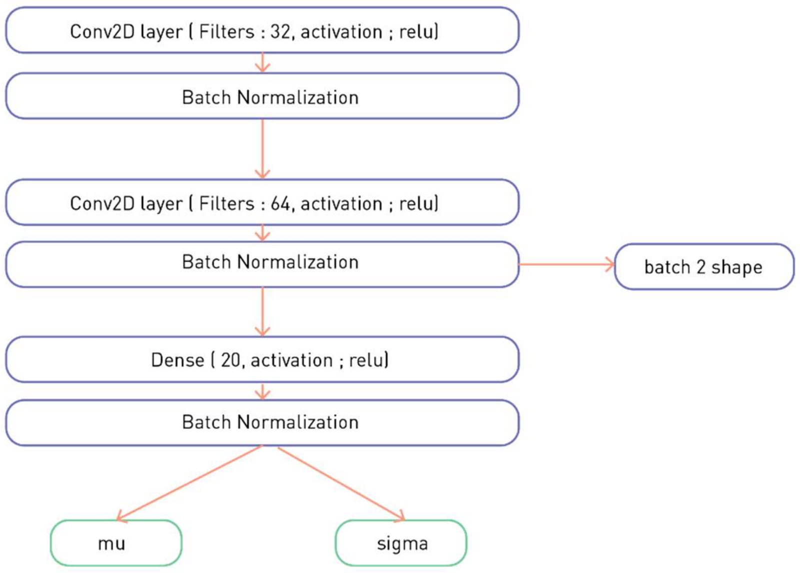

Conv2D layer (Filters : 32, activation ; relu)
Batch Normalization
Conv2D layer (Filters : 64, activation ; relu)
Batch Normalization
batch 2 shape
Dense (20, activation ; relu)
Batch Normalization
mu
sigma

Figure 36 Encoder

After the encoder is defined, a custom sampling layer code is added to the VAE architecture that uses mu and sigma as inputs, the epsilon is the random normal gaussian distribution that uses the shape of mu and sigma to give us the output of the sampling layer. The snippet of the code is shown below:

```
#Sampling

class Sampling(tf.keras.layers.Layer):

  def call(self,inputs):

    mu, sigma = inputs

    batch = tf.shape(mu)[0]
    dim = tf.shape(mu)[1]

    epsilon = tf.keras.backend.random_normal(shape=(batch,dim))

    return mu + tf.exp(0.5*sigma) * epsilon
```

The encoder model now has the combination of the sampling layer and the encoder layers which creates the first part of the architecture of the VAE. We get z as the output which is the sampling layer along with the mu and sigma from the encoder. The code for the same is shown below:

```
#Encoder_model

def encoder_model(latent_dim, input_shape):

  inputs = tf.keras.layers.Input(shape=input_shape)

  mu, sigma, conv_shape = encoder_layers(inputs, latent_dim=LATENT_DIM)

  z = Sampling()((mu,sigma))

  model = tf.keras.Model(inputs, outputs=[mu,sigma,z])

  return model, conv_shape
```

The first half of the architecture is now complete. The next step is to define a loss function and then the decoder.

4.2 Loss function

To ensure that the Gaussian distribution that has been sampled aren't too far apart, a regularization term is added which is the Kullback-Leiber Divergence (KLD).

" KLD is a measurement of how different one probability distribution is to another. For two distributions P and Q, the KLD of P with respect to Q is the cross-entropy of P and Q minus the entropy of P. In information theory, entropy is a measure of information or uncertainty of a random variable:

$D_{KL} (P||Q) = H (P, Q) -H(P)$" (Cheong 2020)

The code for the KLD is shown below, it uses the mu and sigma from the encoder model output as an input.

```
#KL Divergence loss

def kl_divergence_loss(inputs,outputs,mu,sigma):

  kl_loss = 1 + sigma - tf.square(mu) - tf.math.exp(sigma)
  kl_loss = tf.reduce_mean(kl_loss) * -0.5

  return kl_loss
```

Other loss functions that are used in the VAE are the reconstruction loss. This measures the difference between the reconstructed images and the target images. For this thesis the binary cross-entropy (BCE) loss has been used, but for that reason the activation of the last layer of the decoder needs to be sigmoid. The calculation of the reconstruction loss is done with-in the training loop of the model, the loss is multiplied by the resolution of the images which is 64x64 = 4096. The code for the same is shown below (Dutta 2021):

```
for step, x_batch_train in enumerate(training_dataset):
    with tf.GradientTape() as tape:

# pass batch of training data
       reconstructed = vae(x_batch_train)

#to measure loss
       flattened_inputs = tf.reshape(x_batch_train, shape=[-1])
       flattened_outputs = tf.reshape(reconstructed, shape=[-1])
       loss_reconstruction = bce_loss(flattened_inputs, flattened_outputs)
 * 4096
```

The two losses are then added together within the training loop so that it tries to minimize both the losses after each step and update the weights and biases. The KLD loss term was added to the VAE model hence it can be called by vae.losses. The code for the same is shown below:

```
loss = loss_reconstruction + sum(vae.losses)
```

4.3 Decoder

The decoder reconstructs data from the latent space and in the process generates new data in the VAE. The conv_shape is what has been saved from the encoder, for this architecture its which means that there are those many numbers of images output from that layer. The decoder is loaded with a dense layer with the same number of neurons as conv_shape. The output is then reshaped and passed through tf.keras.layers.Conv2DTranspose layers with the reverse number of filters as the encoder layers to invert the convolutional filters.

The architecture of the decoder is shown in the picture, the final output is a single filter conv2DTranspose layer which will give the same output as the input which is a 64x64x1 image.

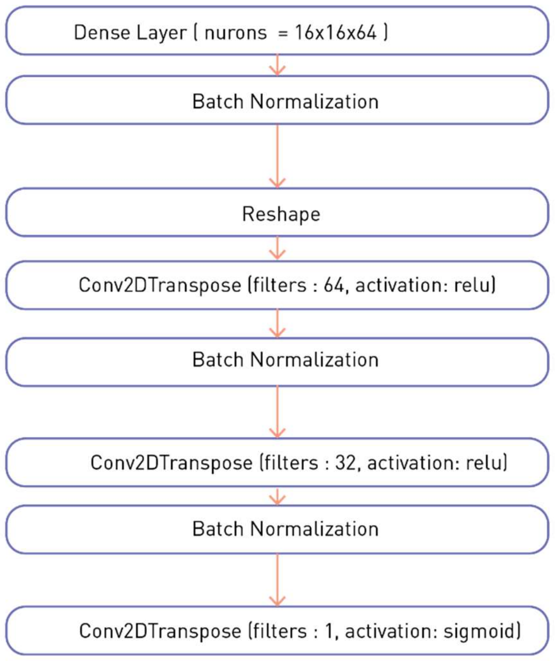

Dense Layer (nurons = 16x16x64)
Batch Normalization
Reshape
Conv2DTranspose (filters : 64, activation: relu)
Batch Normalization
Conv2DTranspose (filters : 32, activation: relu)
Batch Normalization
Conv2DTranspose (filters : 1, activation: sigmoid)

Figure 37 Decoder

After the decoder layers, the decoder model is created as a function which takes the latent dimension and the conv_shape from the encoder as it's inputs. These inputs are used to create an input layer inside the function and we use that to call the decoder layers created before as outputs. These inputs and outputs are then used to create a tf.keras.model for the decoder. The code for the same is shown below:

```
#Decoder_model

def decoder_model(latent_dim, conv_shape):

  inputs = tf.keras.layers.Input(shape=(latent_dim))

  outputs = decoder_layers(inputs, conv_shape)

  model = tf.keras.Model(inputs, outputs)

  return model
```

4.4 Generation of New Patterns

After the Encoder, Sampling and the Decoder, the next step is to train the VAE model. The hyperparameters such as epochs and learning rate of the model is controlled during this stage. The number of epochs is the number of times the entire dataset will be passed through the VAE model. The optimizer is also specified with the learning rate of the model. Learning rate is the steps in which the model learns. A larger learning rate would overshoot the results and too small will take forever to train. The standard learning rate for the Adam optimizer is 0.001. During this stage iterations of the hyperparameters, batch size, latent dimensions are carried out for the optimum result to reduce loss.

The first step is creating the VAE model from the encoder, decoder and sampling layers. The inputs are from the encoder model (mu, sigma and z) and the output is the decoder model which decodes z. The kl_divergence_loss is added to the model in this stage so that it recognizes it as a loss later on during training. The code for the VAE model is shown below (Dutta 2021):

```
# Final VAE Model

def vae_model(encoder,decoder, input_shape):

  inputs = tf.keras.layers.Input(shape=input_shape)

  mu,sigma,z = encoder(inputs)

  reconstructed = decoder(z)

  model = tf.keras.Model(inputs=inputs, outputs=reconstructed)

  loss = kl_divergence_loss(inputs, z, mu, sigma)
  model.add_loss(loss)

  return model
```

For this step a custom training model is created. In Keras, it is difficult to customize the training process with pre-existing training functions like fit() and evalulate(). With tf.Gradient.tape() we can compute the loss for each batch during a training and can add an optimizer to update the weights and biases during the training itself based on the gradients. This allows for more control over the training process. The loop measures the total loss every epoch and then updated the weights and biases after every epoch. The code for the training loop is given below (Dutta 2021):

```
for epoch in range(epochs):
  print('start of epoch %d'%(epoch,))

  for step, x_batch_train in enumerate(training_dataset):
    with tf.GradientTape() as tape:

# pass batch of training data
       reconstructed = vae(x_batch_train)

#to measure loss
       flattened_inputs = tf.reshape(x_batch_train, shape=[-1])
       flattened_outputs = tf.reshape(reconstructed, shape=[-1])
       loss_reconstruction = bce_loss(flattened_inputs, flattened_outputs)
 * 4096

 # adding KLD regularization loss
       loss = loss_reconstruction + sum(vae.losses)


 # get the gradients and update the weights
       grads = tape.gradient(loss, vae.trainable_weights)
       optimizer.apply_gradients(zip(grads, vae.trainable_weights))
```

4.4.1 Training Results

After building the VAE model, it was trained on the dataset of lattice patterns. The dataset contained 4089 samples after data augmentation. During the training process the weights and biases are updated in the model after each epoch to reduce the loss. To increase the efficiency of the model and hyperparameters such as, epoch, batch size, learning rate can be modified and checked for better loss. However, the architecture of the VAE can also be adjected to check if there are better configurations of it to reduce the loss. For example, the number of convolutional layers, latent dimensions, filters of the convolutional layers and dense layer etc. Different configurations of latent dimensions were tested as they produced different results.

The total loss was recorded and compared with the validation loss to check if the model was a good fit or if it was overfitting. The model for most cases, except for the one mentioned below was a good fit and the validation loss and the total loss converged after a certain point. After setting up the model the first iteration was the latent dimension. In different studies, depending on the complexity of the dataset, the latent dimension is adjusted. From precedents of VAEs trained with MNIST greyscale images the latent dimension usually selected was 2, however in other cases with more custom datasets, higher latent dimensions are also selected (Vogelsanger and Federau 2021). Values ranging from 2,3,8, 32,64 and 128 were selected for the latent dimension for epochs of 1000. The results showed that even though the loss was less for higher dimensions, the generated images were blurrier. Through visual comparison, it was observed that, the generated images with higher latent dimensions, even though had reduced total loss, were high in contrast but more organic and less similar to the images from the original dataset. This can be observed in the figure below where latent dimensions of 2, 3 and 8 are shown. The second observation was that reducing the learning rate to 0.0005 (No.2) in the figure caused overfitting of the model, the loss was significantly less than iteration No.1 in the figure during the initial epochs but then started increasing over time. Through these observations, since the thesis aims to generate images that can be used in design, latent dimension of 2 and learning rate of 0.001 was selected for further training of the model.

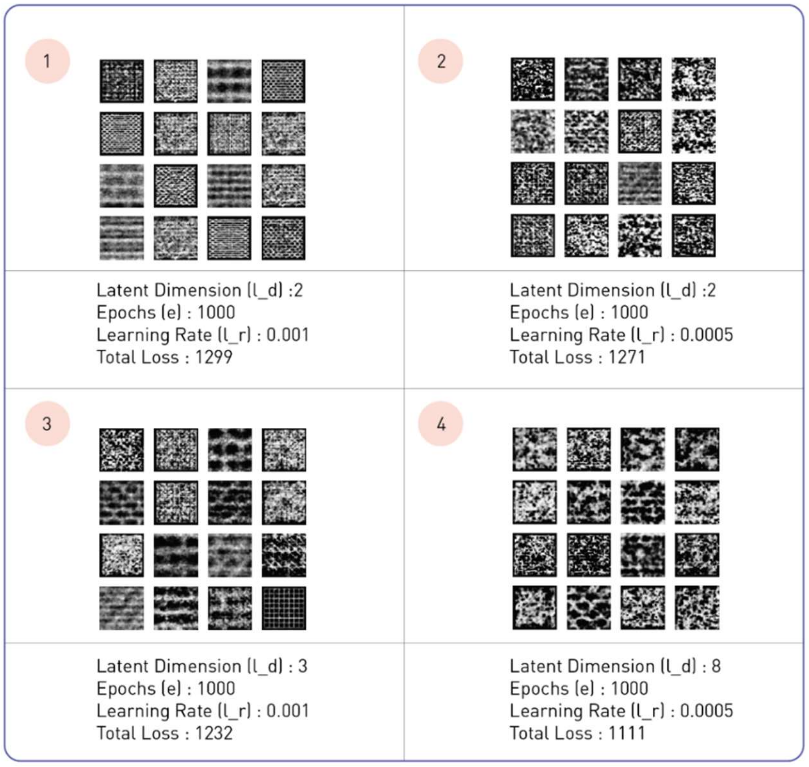

1
Latent Dimension (l_d) :2
Epochs (e) : 1000
Learning Rate (l_r) : 0.001
Total Loss : 1299
2
Latent Dimension (l_d) :2
Epochs (e) : 1000
Learning Rate (l_r) : 0.0005
Total Loss : 1271
3
Latent Dimension (l_d) : 3
Epochs (e) : 1000
Learning Rate (l_r) : 0.001
Total Loss : 1232
4
Latent Dimension (l_d) : 8
Epochs (e) : 1000
Learning Rate (l_r) : 0.0005
Total Loss : 1111

Figure 38 generated images for different iterations of Latent Dimensions. Since Latent Dimensions of higher than 8 generated extremely blurry images they are not shown in this comparison.

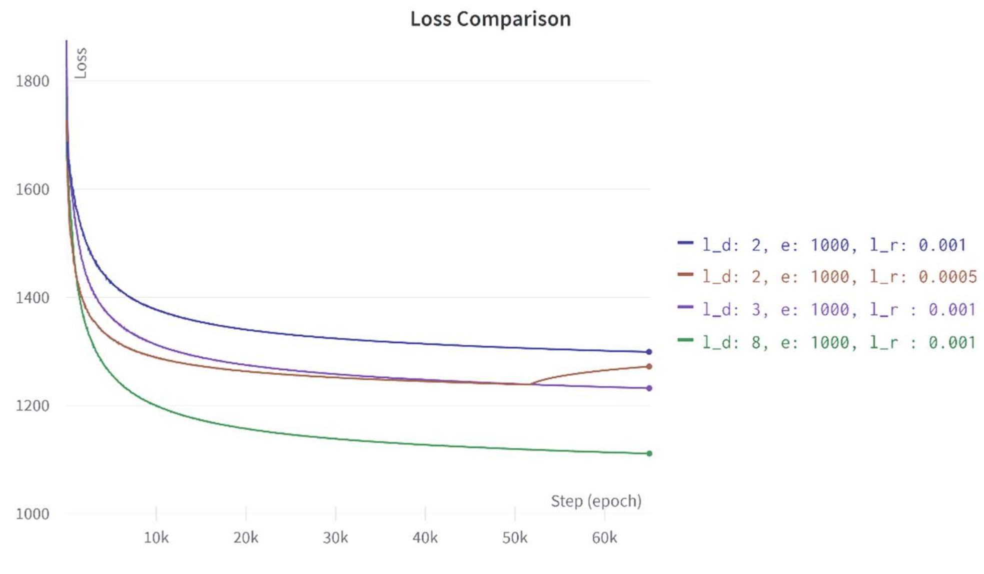


Figure 39 Loss Comparison of the different iterations of the Latent Dimension

The model was further trained for different hyperparameter iterations of latent dimension 2. The hyperparameters that were adjusted were the epoch, batch size and the learning rate was kept at 0.001 which worked best for the model. The model architecture was not adjusted during the training, except for adjusting the dense layer filters in the encoder layer, to check if there were any changes in the loss. A few of the results of the hyperparameter tuning are shown below in the figure. Option 1 had the least loss among the many iterations and hence, due to the time constraint and for the scope of this thesis it was selected as the final output of the VAE model. However, more iterations and hyperparameter tuning can be carried out to get better results. The generated images are still relatively blurry but the outline of the patterns could be identified from some of them.

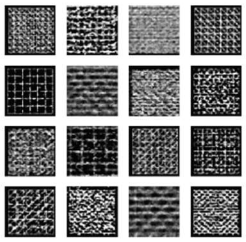
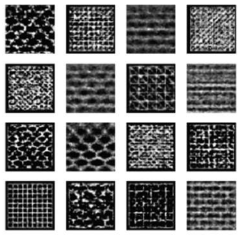
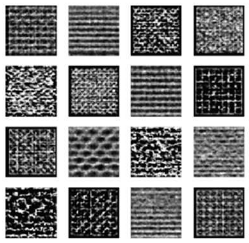
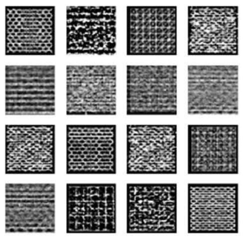
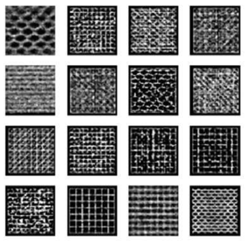
1 	2 
Latent Dimension (l_d) :2 Epochs (e) : 6000 Dense layer Filters (dl) : 60 Batch Size (b) : 64 Total Loss : 1249	Latent Dimension (l_d) :2 Epochs (e) : 4000 Dense layer Filters (dl) : 20 Batch Size (b) : 64 Total Loss : 1324
3 	4 
Latent Dimension (l_d) : 2 Epochs (e) : 4000 Dense layer Filters (dl) : 20 Batch Size (b) : 32 Total Loss : 1343	Latent Dimension (l_d) : 2 Epochs (e) : 4000 Dense layer Filters (dl) : 100 Batch Size (b) : 64 Total Loss : 1272
5 	
Latent Dimension (l_d) : 2 Epochs (e) : 4000 Dense layer Filters (dl) : 60 Batch Size (b) : 64 Total Loss : 1255	

Figure 40 Generated mages for different iterations of latent dimension 2

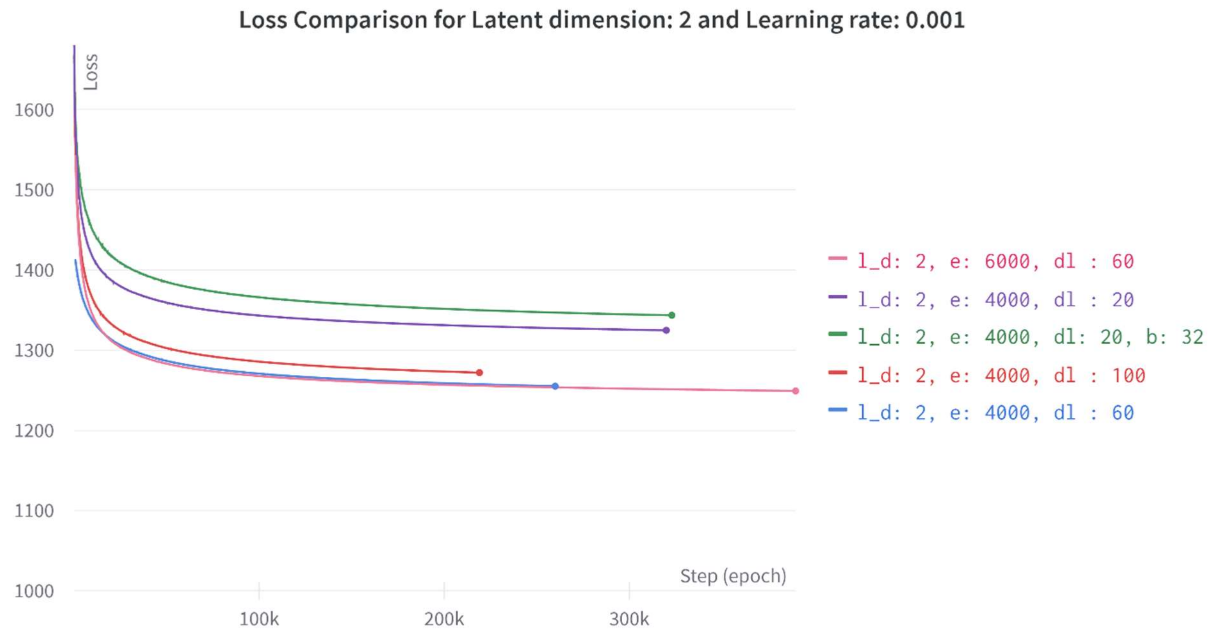


Figure 41 Loss Comparison of Different hyperparameter tuning for latent dimension 2

After the selection of the result from the training, a collage of the generated images sampled from the decoder was made. The images that were more defined than the rest were visually compared with the images from the existing dataset and the ones that appeared new were selected. In the figure below the new generated patterns are identified.

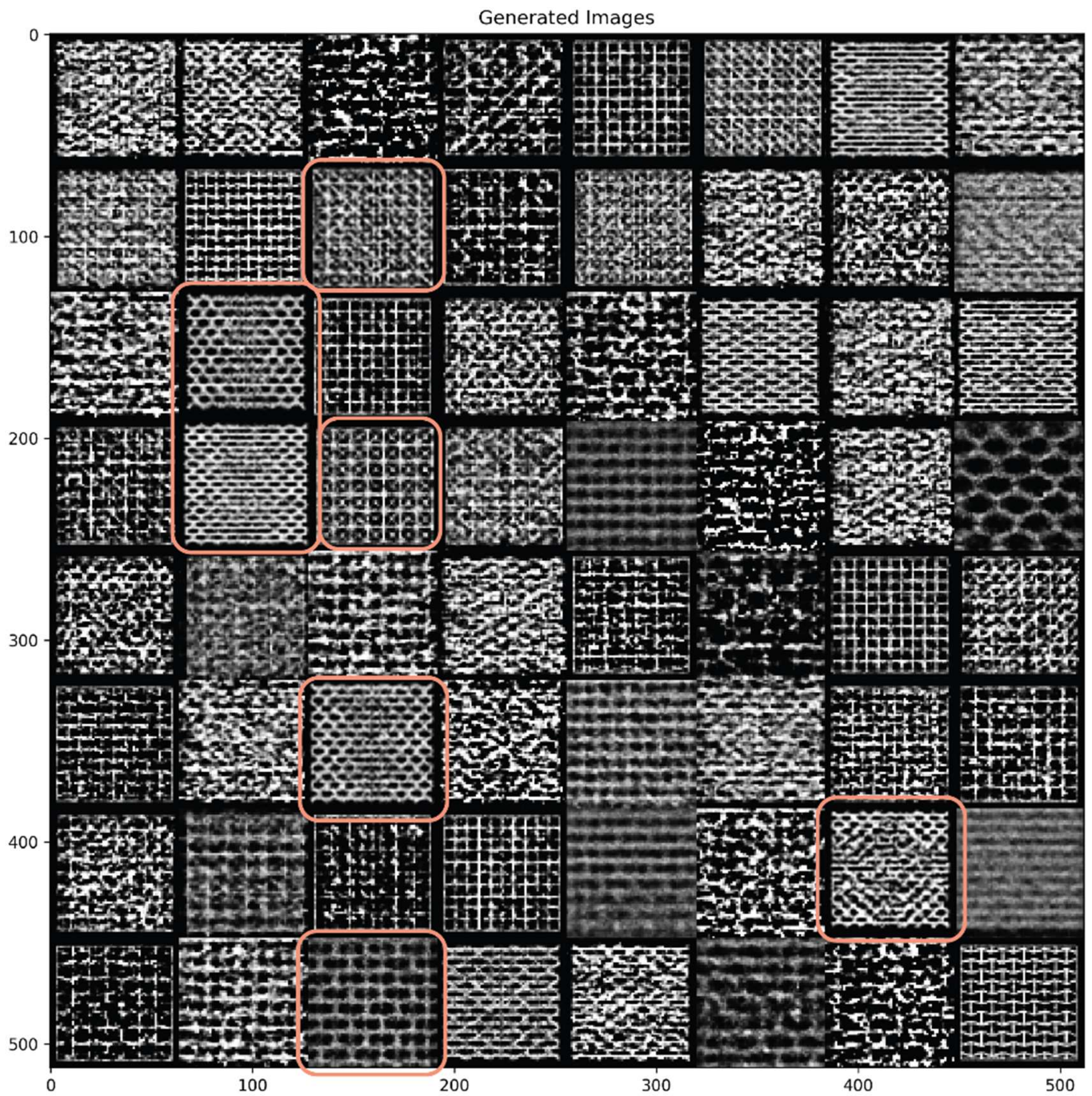


Figure 42 Sampling of Generated patterns

V USING THE VAE AS A DESIGN TOOL

The design process consists of various stages. Major decisions regarding the building geometry, structure, massing is made during the conceptual design phase which account for 75% of the total product costs (Hsu 2000). Generative AI models can provide an opportunity of exploring larger variety of designs during the conceptual design phase, which can be structurally informed to create a better design – build workflow for the future.

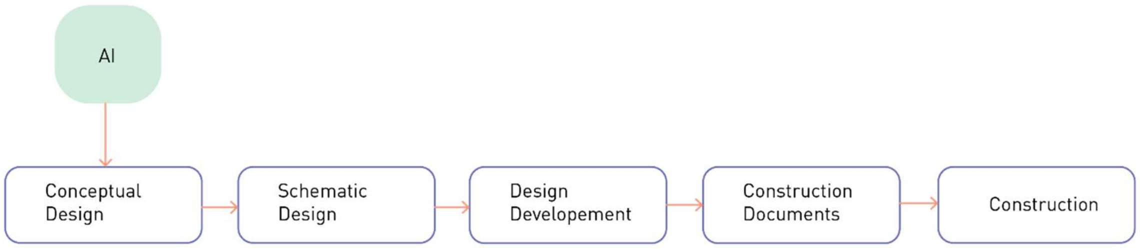


Figure 43 Design Workflow with AI

The exercise of this thesis was to explore a way of generating new patterns through AI and incorporating that into the conceptual design phase as a tool for exploration of topological design ideas. For the application of the patterns, topological exploration of shell structures was used.

From the previous section, the most distinct patterns from the new generated patterns were identified. Out of the selected patterns, only few could be extracted for use as the rest were too blurry. The patterns are then edited to make them sharper by increasing the contrast and the brightness.

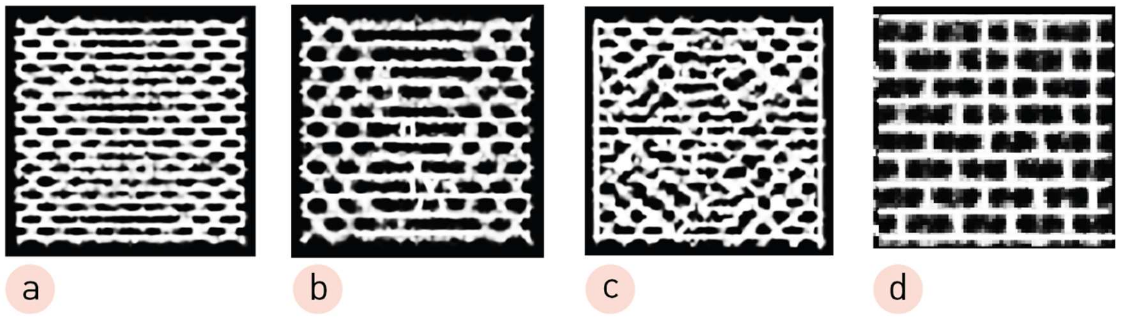


Figure 44 Generated Images selected for application in the Shell structure

5.1 Application in Topology Exploration

After training of the VAE, its output is 2D images in the same format of 64x64 pixel resolution as the dataset. These images are of patterns that did not exist before but are generated by the AI. Since previously explained, the 2D lattice patterns are widely studied for their stiffness and skin morphing properties (Arredondo-Soto M 2021), the new generated patterns could also have some of the underlying properties of the original data translated. For use of these patterns in the conceptual design phases on shell structures a basic workflow of the same has been developed. This workflow has 4 stages.

The first stage is the extraction of the geometry from the 2D image pattern into a data structure which can be controlled in the parametric model. This is done by removing the darker points from the surface and only keeping the high contrast points of the pattern. The drawback of this process is that if there are low contrast areas, it removes those parts of the pattern too, so if the generated image has a high variance in its brightness levels, it would be efficient to edit the contrast and then import into the parametric model.

The second step is to create the form of the shell on which the pattern will be morphed. The plane of the pattern and the footprint of the shell are the same size and geometry. The form of the shell is generated through mesh relaxation for this application example, but other approaches can be taken too.

The pattern is morphed on the shell form rather than the pattern themselves being relaxed to create the shell is because, having the form as a separate part of the process allows for greater diversity in the design. And the process is simpler to morph the pattern on a form, and it does not affect the structural analysis as the FEM analysis in grasshopper takes the morphed pattern as the final form of the shell for its calculation. Workflow for Stages 1 and 2 are shown in Figure 46.

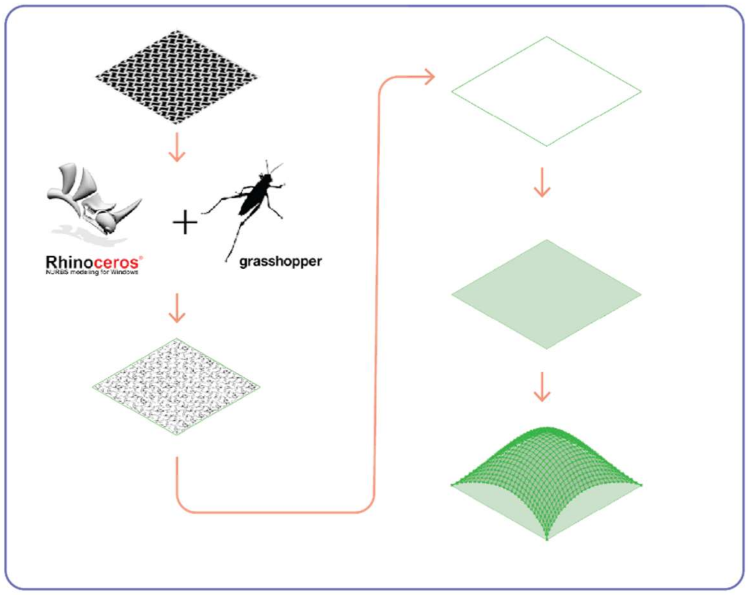


Figure 45 Stage 1 and Stage 2 of the application process

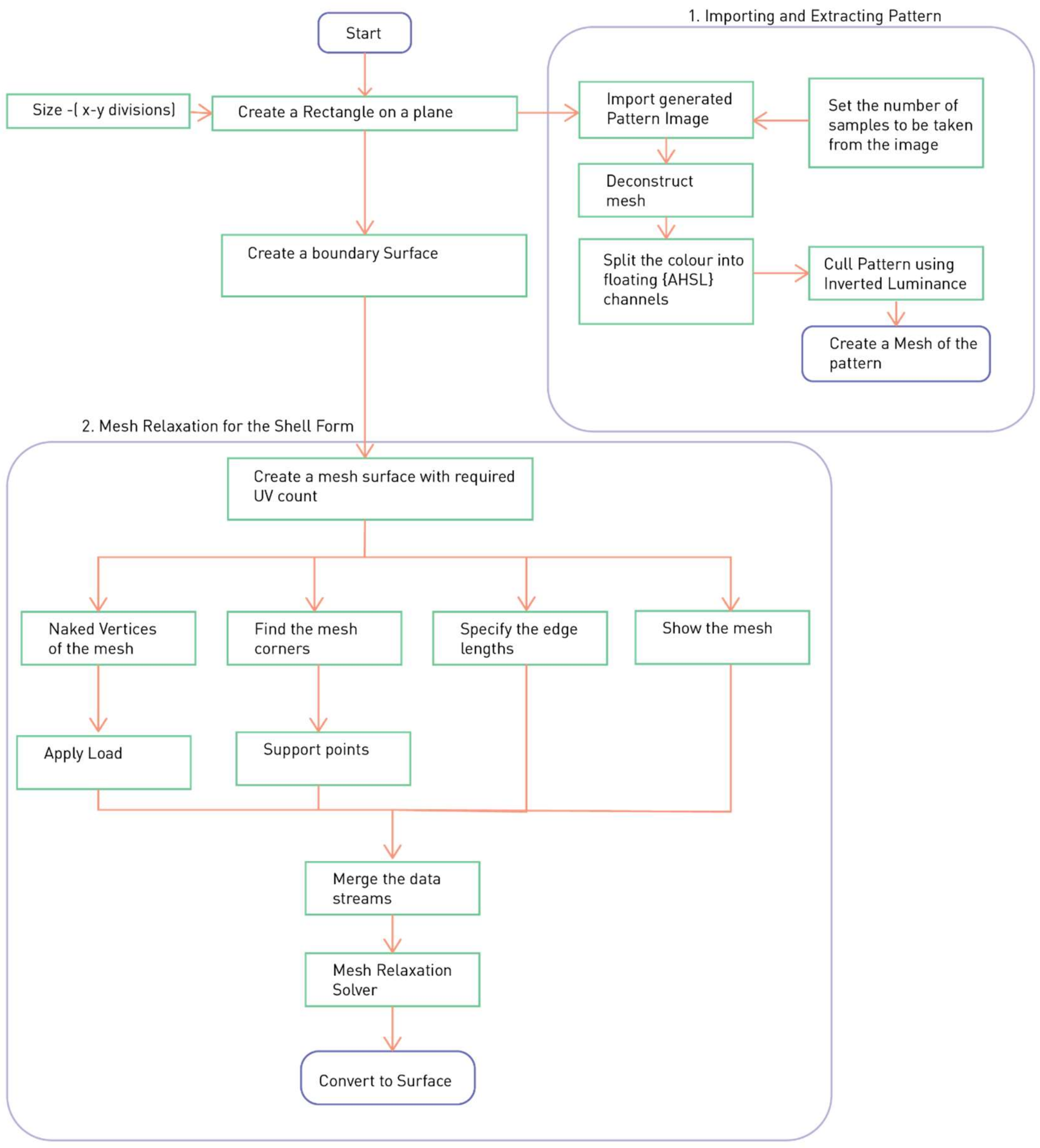

Start
Size -(x-y divisions)
Create a Rectangle on a plane
Create a boundary Surface
1. Importing and Extracting Pattern
Import generated Pattern Image
Set the number of samples to be taken from the image
Deconstruct mesh
Split the colour into floating {AHSL} channels
Cull Pattern using Inverted Luminance
Create a Mesh of the pattern
2. Mesh Relaxation for the Shell Form
Create a mesh surface with required UV count
Naked Vertices of the mesh
Find the mesh corners
Specify the edge lengths
Show the mesh
Apply Load
Support points
Merge the data streams
Mesh Relaxation Solver
Convert to Surface

Figure 46 Stage 1 and 2 of the application

The third stage is morphing the pattern mesh from stage 1 on the form created in stage 2 and the fourth stage is the FEM analysis. In the parametric model in Grasshopper, this is done with Karamba. The morphed surface with the pattern is considered a shell in the input for the analysis, and loads conditions and support conditions can be defined. The material of the shell as well as the cross-section of the elements can also be defined which makes it customizable to go through different design iterations. Since this application is for conceptual design phase, the focus was more on creating a workflow that helps in exploring and analyzing different options rather than optimizing a single solution. Figure 47 shows the detailed workflow for stages 3 and 4 in the application process.

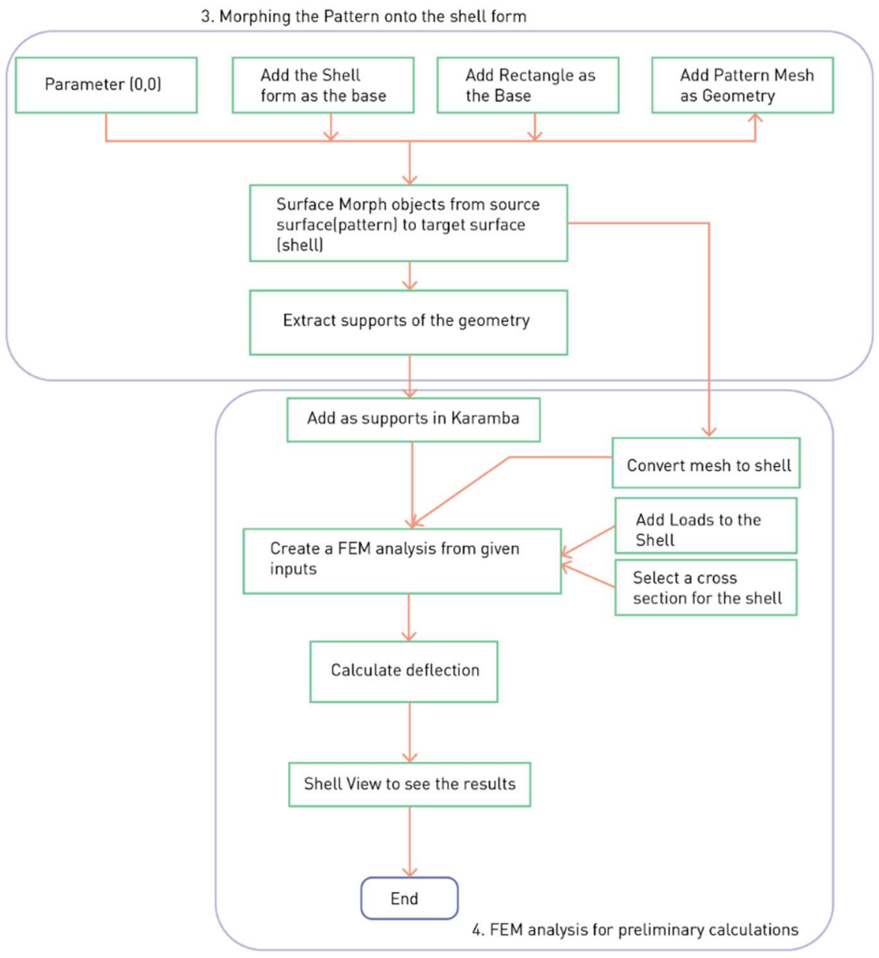

3. Morphing the Pattern onto the shell form
Parameter (0,0)
Add the Shell form as the base
Add Rectangle as the Base
Add Pattern Mesh as Geometry
Surface Morph objects from source surface(pattern) to target surface (shell)
Extract supports of the geometry
Add as supports in Karamba
Convert mesh to shell
Add Loads to the Shell
Select a cross section for the shell
Create a FEM analysis from given inputs
Calculate deflection
Shell View to see the results
End
4. FEM analysis for preliminary calculations

Figure 47 Stage 3 and 4 of the application

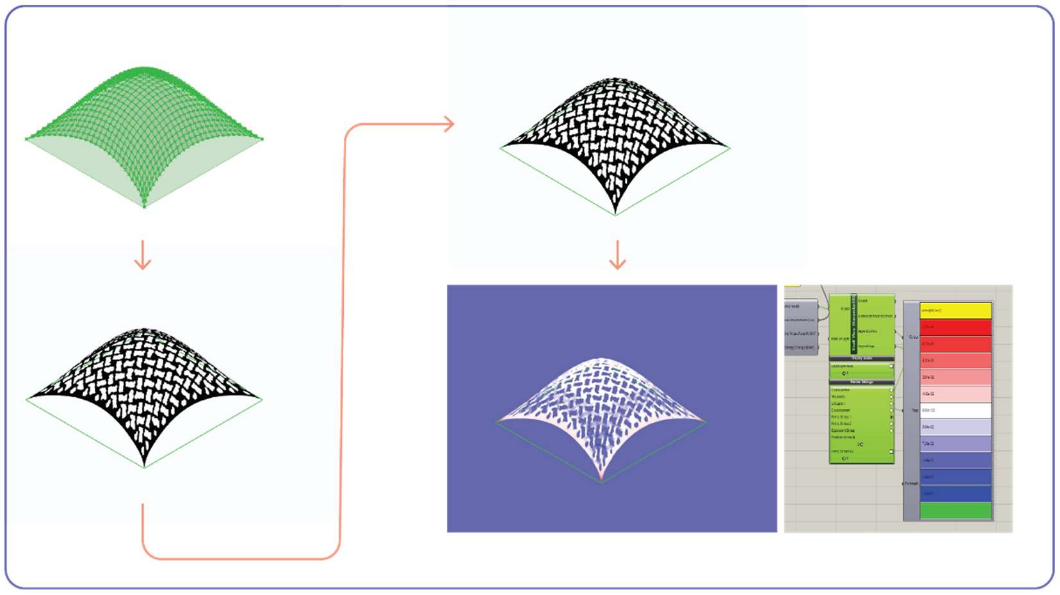


Figure 48 Stage 3 and Stage 4 of the application process

The next step is integrating the generated patterns into the process. In that process it was observed that some of the details of the patterns do not get translated into the parametric model (observed in option d). This might be because of the lower resolution of the generated images which the inverted luminance component fails to identify points in different luminance with higher precision. The following figure shows the final shell structure with the topology of the new generated patterns.

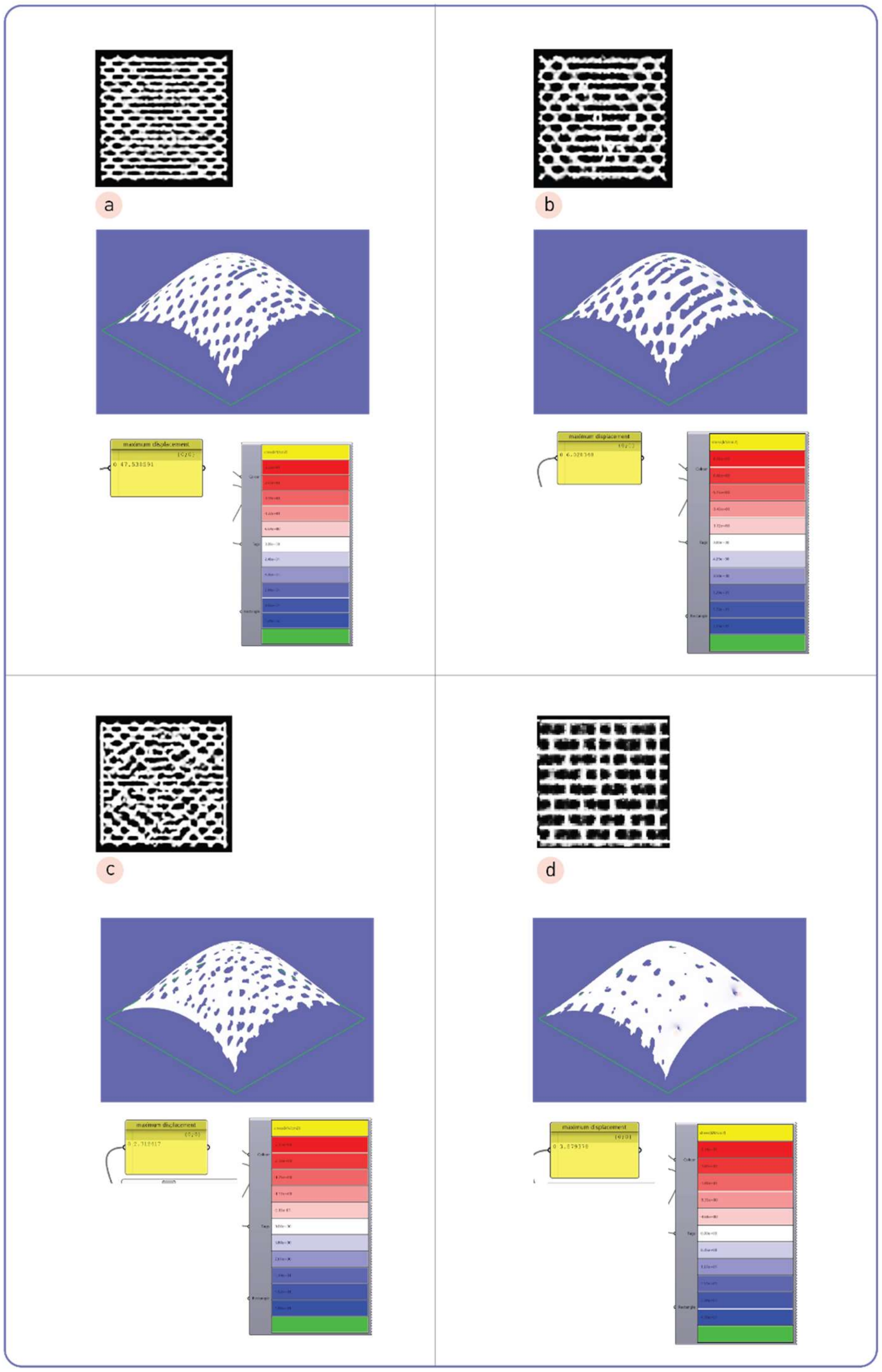


Figure 49 New topology design ideas from generated patterns.

Future Development of Application:

We can observe that, integrating the VAE into the design process can yield more variety of options which are informed by the design options that were used during the training. Though this process can be used to explore more design options, this is not the ideal realization of the workflow, there are a few steps that were missing and are important for future applications. The importance of VAE exists in its capability to compress information into a lower dimension. Though it is not within the scope of this thesis, but in terms of future application and integration, the most powerful way a VAE can be utilized is through the exploration of the latent space. The latent space contains all the learned features of the input data that is stored by the VAE in a reduced dimension. For example, in this thesis, the original dataset of images is with dimensions 64x64x1, and the latent space is 2x1, the compressed datapoint of each data sample is a vector with 2 dimensions. New data can be generated through the linear interpolation of the latent space, using the model decoder to reconstruct the latent space representation into a 2D image of the same dimensions as the original images. An example of using the latent space for design exploration are in the papers: (Renaud and Caitlin T. 2021), (Mirra and Pugnale 2021). Exploring the latent space can provide insight into understanding the patterns of the data and similarities within the data which can be extremely useful in the design process, as design data samples that have similar features (for example: performance) would be closer together along with new generated options with similar features, this can then aid the designer in exploring the latent space in the direction of the desired results.

VI DISCUSSION AND CONCLUSION

6.1 Discussion

This section presents the discussion for the dataset creation, VAE architecture and training, and the application process carried out in this thesis. There are four sections discussed which are defined as: Dataset creation, the VAE architecture and training, Generation of new data, and Application of AI in design.

1. Dataset creation: The dataset for training the VAE model was artificially created using 6 different typologies of 2D lattice patterns. The patterns were modelled in grasshopper and then the iterations were exported as 2D images in greyscale. The resolution of the data samples was set to be 64x64 as the information in the data samples (patterns) were not extremely complex and increasing the resolution would make the feature extraction more complex and would result in a more complex VAE architecture. The size of the dataset was relatively small in comparison to other open-source datasets which have larger quantities of data samples which might have led to the model not performing that well during training. Also, the scale of some of the patterns were too small, even though they well scaled during data augmentation, because of which during training there were generated images with a lot of noise as it couldn't differentiate appropriately the features of the data samples. The format of the data samples used to create the dataset could also be changed to a format other than 2D images, for example depth maps or points but since the images could translate the topology of the patterns, they were selected. There was no error reported during import of the dataset into the VAE model, so the methodology used in this thesis worked. However, there can be improvements in the artificial generation of the data and also increasing the size of the dataset.
2. VAE Architecture and training: The VAE has two convolutional layers, one with 32 filters and the second one with 64 filters in the encoder. This was observed to be the most efficient in terms of utilizing the computational power and also the most used in other examples of datasets with the same format. However, the number and the filters of the convolutional layers could be increased or adjusted depending on the complexity of the data samples. Due to the time limitation, only a few hyperparameter iterations were tested, but there can be further iterations tested, which might lead to a better loss.

The platform used to write the code was google collab notebooks, which is an online coding platform. It was convenient to use it as it did not require any installations and all the libraries and their most recent versions could be imported into the notebook. However, the drawbacks were that the dataset needed to be located in the google drive for it to be accessible through google collab and the maximum duration of a run is 12 hours. Because of which, for longer epochs and complex architecture it might not be convenient, and for this thesis, the architecture of the VAE had to be made as simple as possible to be computed without hindering the actual training of the model. The loss function used was binary cross entropy because of which the last activation of the decoder layer had to be sigmoid, but other loss functions for example mse, could be explored. Another observation during training was that, the reconstructed images sampled for the exact input coordinates were very noisy, but when random points were sampled from the decoder, there were still some clearer patterns. This needed to be further researched, as to why that was occurring. Since, the model did run without errors and there was convergence of the total loss and the validation loss, it can still be concluded that the model was a good fit. However, it can of course be made more efficient and the training results can be improved.

3. Generation of new data: Simple VAE (model used in this thesis) among the generative models of AI produces blurrier images than other hybrid models of VAE or GAN (Vogelsanger and Federau 2021). This can be observed in the results of the training, but in context to this thesis, the hyperparameters could also be explored further. The VAE did produce some distinct patterns that were new and didn't exist before, but to be able to discover the full potential of generative models in AI, it might be interesting to test out this dataset in other hybrid models of VAE or GANs. In this thesis, since the generated data was blurry, the selected patterns were postprocessed to be made sharper, but ideally that would not be necessary with better models or clearer results.

4. Application of AI in design: The application process explained in this thesis involved a workflow with the parametric modelling software – grasshopper in rhinoceros, since it is a commonly used platform. The algorithm in grasshopper culls the pattern based on contrast and pixels. In this thesis, the resolution of the data samples was relatively small (64x64) which led to the geometry in the parametric platform to have rough edges and not a smooth geometry. This can be improved if the resolution is increased. The results of the application are more for comparison and presenting what is practically possible. But this is just a preliminary exploration of the possibilities of generative AI models in design, and other applications are also possible and should be explored. The latent space interpolation and data generation can become a powerful tool in the design process.

6.2 Conclusion

The use of AI in design is still in its early stages and this thesis explored a workflow of possible integration of a generative AI model (VAE) into the design process. To explore the possibilities a series of questions were formulated that guided the research process. They can be answered as follows:

Main Question: How can AI extract useful information from a dataset of cellular solid structure patterns and reuse it to generate new patterns for structural design?

The AI model used in this thesis that can generate new patterns is the Variational Autoencoder (VAE). The VAE learns by training on a dataset made up of a collection of data samples. These data samples are either artificially generated or sourced. For the VAE model to train on cellular solid patterns, the dataset had to be generated as images of cellular solids that fit the format to train a VAE model in sufficient quantity and quality is not available. 2D lattice patterns were chosen to create the data samples for the dataset. The data is then augmented to increase the population of the dataset. After the dataset is formatted, normalized and reshaped, it is fed into the VAE model. Inside the model the data is first compressed by the encoder and then it is sent to the latent space where noise is added (gaussian distribution) to the original data, so that when the decoder decodes the latent space it can sample data that didn't exist before, hence generating new data. This process gets more efficient with the training of the VAE which minimizes the loss. And at the end of the process the VAE can generate new data. The steps are described in further detail in the sub questions

Sub Question 1: What are the selection criteria of the cellular solid patterns for creating the dataset?

The selection criteria were the diversity of the patterns and if their mechanical properties were dependent on their topology. The patterns needed to be in 2D as the dataset was made of images. The 2D cellular solid patterns that had distinct mechanical properties, were diverse enough and also which were researched enough to be able to get the required information to artificially make the dataset were 2D lattice patterns. Out of the existing 2D lattice patterns, the most popularly researched patterns in terms of their stiffness or flexibility were chosen.

Sub Question 2: How to artificially create a dataset?

The format of the data selected to be used in training the VAE model was greyscale images; similar to the MNIST handwritten number dataset, because the benchmark testing of the VAE was done on that dataset and using the same format meant least changes to the architecture of the VAE model. The next step was translating the lattice patterns to images. This was done by creating a model of the lattice patterns in a parametric design space (grasshopper, rhinoceros). The model was then used to generate iterations of the lattice patterns over selected design variables and export those iterations as images of a fixed resolution. This resolution was 64x64 pixels. After generating the data, to ensure that the AI has enough data to train on, data augmentation of the geometric features of the images are carried out. After data augmentation, the dataset is reshaped and normalized to be used within the model and then finally split into test and validation datasets.

Sub Question 3: How can the AI-generated patterns be used to explore topology optimization design ideas? (Application)

The generated patterns are just images in the same resolution as the input data. To explore topological design ideas, these patterns need to be translated into a geometry, which can be analyzed. One such method that was suggested in this thesis was to morph the pattern onto a form which can be a generated shell form from parametric modelling. Morphing of the pattern onto an existing form makes it easier to alternate between different patterns and explore a wider variety of options. Since this is done during the initial design phase having better control over customization would be helpful. The morphed patterns are then analyzed by a FEM to get the overall comparison of the results of the different patterns and how they perform.

6.3 Limitations

1) The dataset was limited to only 6 lattice patterns. For the scope of this thesis, these patterns were easier to produce in the time frame. But more organic patterns and more complex patterns can also be used to train and AI model

2) The data augmentation is only done to crop and flip the images, but other geometric transformations are also possible.

3) Google collab was used to implement the code. It is online, so the dataset needed to be uploaded in the google drive. But any other platform can be used to run the code, after checking the compatibility of the libraries used.

4) This workflow for creating the AI model is just one way of doing it. There are a lot of different approaches available. This approach was chosen is in no way tested to be superior to a different VAE architecture. Customization according to goal of the model is necessary.

5) The output of the VAE training can be improved further, but within the time and scope of this thesis, the best result out of the iterations of hyperparameters is chosen.

6) The application process is only to show an example of how a VAE can fit into the design process. It can be explored further.

7) The detailed mechanical properties of the lattice structures are not studied for this paper. The geometry is developed based on available data or through studying the pattern.

6.4 Future Development

There are a lot of opportunities for future development from this thesis. AI is still a novel field of research in design and there were a lot of aspects that due to time constraint were not explored.

1. The interpolation of the latent space to navigate through the data samples (original and generated) better, which could be interesting to explore traversing design options.
2. Exploring the use of 3D shapes to create the dataset. How would the data be fed into the AI model and which information would be extracted from it?
3. Develop the AI model from this thesis further. It needs further training and the code can be made more efficient.
4. Explore other possibilities of application of the generated patterns and analyze their mechanical properties and check if they can be compared to the original lattice patterns, if at all.
5. The optimization of the generated pattern and their corresponding shell structures could be explored.

VII REFERENCES

Adriaenssens, Sigrid, Philippe Block, Diederik Veenendaal, and Chris Williams. 2014. *Shell Structures for Architecture.* Oxon and New York: Routledge.

Ahmed, Usman. 2019. *The Top 6 Ways Artificial Intelligence Will Affect Design In The Future.* September 25. Accessed June 19, 2022. https://www.jeffbullas.com/artificial-intelligence-design/.

Architects., American Institute of. 2007. "Integrated Project Delivery: A Guide." *American Institute of Architects.* http://info.aia.org/SiteObjects/files/IPD_Guide_2007.pdf.

Arredondo-Soto M, Cuan-Urquizo E, Gómez-Espinosa A. 2021. "A Review on Tailoring Stiffness in Compliant Systems, via Removing Material: Cellular Materials and Topology Optimization." *Applied Sciences.* 11(8). 11(8):3538. https://doi.org/10.3390/app11083538.

Ayres, Phil & Martin, Alison & Zwierzycki, Mateusz. (2018). "Beyond the basket case: A principled approach to the modelling of kagome weave patterns for the fabrication of interlaced lattice structures using straight strips." *Conference: Advances in Architectural Geometry.* Gothenburg, Sweden: Chalmers University of Technology,.

Basu, Ritupriya. 2019. *Algorithms Are a Designer's New BFF – Here's Proof.* December 6. Accessed May 16, 2022. https://xd.adobe.com/ideas/principles/emerging-technology/automation-ai-wont-replace-designers/.

Baumgart, F. 2000. "Stiffness — an unknown world of mechanical science?" *Injury,Volume 31, Supplement 2,* 14-84. doi:https://doi.org/10.1016/S0020-1383(00)80040-6.

Bendsoe, Sigmund. 2002. *Topology Optimization: Theory, Methods and Applications (2nd ed).* Heidelberg: Springer, Berlin, Germany.

Bensoe, Martin P. 1995. *Optimisation of Structural topology, Shape and material.* Belin: Springer-Verlag.

Boonstra, Sjonnie & van der Blom, Koen & Hofmeyer, Hèrm & Emmerich, Michael & Schijndel, Jos & Wilde, Pieter. 2018. "Toolbox for super-structured and super-structure free multi-disciplinary building spatial design optimisation." *Advanced Engineering Informatics.* 36.

Cheng, L., Zhang, P., Biyikli, E., Bai, J., Robbins, J., & To, A. 2017. "Efficient design optimization of variable-density cellular structures for additive manufacturing: Theory and experimental validation." *Rapid Prototyping Journal* 660-677. doi:https://doi-org.tudelft.idm.oclc.org/10.1108/RPJ-04-2016-0069.

Cheong, S.Y. 2020. *Hands-On Image Generation with TensorFlow: A practical guide to generating images and videos using deep learning.* Packt Publishing. https://books.google.nl/books?id=tGcREAAAQBAJ.

Dsouza, Jason. 2020. *Deep Learning Crash course for beginners .* July 30. https://www.youtube.com/watch?v=VyWAvY2CF9c.

Dutta, Balaram. 2021. *Variational Autoencoders on Anime Faces.* March 1. Accessed March 20, 2022. https://github.com/duttab49/tfwork/blob/main/VariationalAutoencodersOnAnimeFace.py .

Euler, L. 1746. *Thoughts on the elements of bodies.* Berlin: Booksellers by royal appointment and by appointment to the Academy of Science.

Gibsob, Lorna J., and Michael F.Ashby. 1997. *Cellular Solids: Structure and Properties.* Cambridge: Cambridge University Press.

Hsu, W., & Liu, B. 2000. "Conceptual design: issues and challenges." *Computer-Aided Design* 849-850.

Kantareddy, S., B. Roh, T. Simpson, S. Joshi, C. Dickman, and E Lehtihet. August 2016. "Saving weight with metallic lattice structures: Design challenges with a real-world example." *Proceedings of the Solid Freeform Fabrication Symposium (SFF).* Austin, TX, USA: The Pennsylvania State University, University Park, PA 16802. 8–10.

Kelleher, John D. 2019. *DEEP LEARNING.* Cambridge, Massachusetts: The MIT Press.

Kelleher, John D. 2019. *Deep learning.* Cambridge: The MIT Press.

Kicinger, Rafal & Arciszewski, Tomasz & De Jong, Kenneth. . 2005. "Evolutionary computation and structural design: A survey of the state-of-the-art. ." *Computers & Structures. 83* 1943-1978.

Kingma, Diederik P., and Max Welling. 2019. "An Introduction to Variational Autoencoders." *Foundations and Trends in machine Learning* 1-18.

Körner, Carolin & Liebold-Ribeiro, and Yvonne. (2015). "A systematic approach to identify cellular auxetic materials." *Smart Materials and Structures. 24. 10.* 1088/0964. doi:1726/24/2/025013.

Lipperman, F., M.B. Fuchs, and M. Ryvkin. 2008. "Stress localization and strength optimization of frame material with periodic microstructure." *Comput. Methods Appl. Mech. Eng.* 4016–4026.

Lubombo, C., and M.A. Huneault. 2018. "Effect of infill patterns on the mechanical performance of lightweight 3D-printed cellular PLA parts." *Mater. Today Commun.* 214–228.

M. Benedetti, A. du Plessis, R.O. Ritchie, M. Dallago, S.M.J. Razavi, F. Berto. 2021. "Architected cellular materials: A review on their mechanical properties towards fatigue-tolerant design and fabrication." *Materials Science and Engineering: R: Reports, Volume 144* 100606. doi:https://doi.org/10.1016/j.mser.2021.100606.

M.P. Bendsøe, O. Sigmund. 2004. *Topology Optimization: Theory, Methods and Applications.* Heidelberg : Springer-Verlag Berlin Heidelberg.

Mahdiyar. 2020. *Rhinocerous Forums.* September 2. Accessed April 24, 2022. https://discourse.mcneel.com/t/is-there-a-way-to-export-black-and-white-image-in-grasshopper-rhino/108742.

Mihai, L.A., K. Alayyash, and H. Wyatt. 2017. "The optimal density of cellular solids in axial tension." *Comput. Methods Biomech. Biomed. Eng.* 701–713.

Mirra, Gabriele, and Alberto Pugnale. 2021. "Comparison between human-defined and AI-generated design spaces for the optimisation of shell structures." *Structures, Volume 34* 2950-2961.

Mirra, Gabriele, and Alberto Pugnale. 2021. "Comparison between human-defined and AI-generated design spaces for the optimisation of shell structures." *Structures,Volume 34* 2950-2961.

Mirra, Gabriele, and Alberto Pugnale. 2021. "Comparison between human-defined and AI-generated design spaces for the optimisation of shell structures,." *Structures* Volume 34, 2950-2961.

Mizobuti, Vinicius, Luiz C.M, and Vieira Junior. 2020. "Bioinspired architectural design based on structural topology optimization." *Frontiers of Architectural Research* 264-276. doi:https://doi.org/10.1016/j.foar.2019.12.002.

Naboni, Roberto & Kunic, Anja. (). . . 10.5151/sigradi2017-058. 2017. "Design and Additive Manufacturing of Lattice-based Cellular Solids at Building Scale." *XXI Congreso Internacional de la Sociedad Iberoamericana de Gráfica Digital.* Chile: Blucher Proceedings. 369-375.

Nguyen, J., S.I. Park, and D.W. Rosen. 2012. *Cellular structure design for lightweight components.* UK: Taylor & Francis: Abingdon,.

Nilsson, Nils J. 2010. *The Quest for Artificial Intelligence: A History of Ideas and Achievements.* Cambridge, UK: Cambridge University Press, 2010.

Oval, Robin & Rippmann, Matthias & Mesnil, Romain & Mele, Tom & Baverel, Olivier & Block, Philippe. 2019. "Feature-based Topology Finding of Patterns for Shell Structures." *Automation in Construction* -.

Pearce, Peter. 1978. *Structure in Nature is a Strategy for design.* Cambridge: The MIT Press.

Renaud, Danhaive, and Mueller Caitlin T. 2021. "Design subspace learning: Structural design space exploration using performance-conditioned generative modeling,." *Automation in Construction* Volume 127.

Rocca, Joseph. 2019. *Understanding variational Autoencoders (VAEs).* September 24. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.

Schaedler, Tobias A. and Carter, William B. 2016. "Architected Cellular Materials." *Annual Review of Materials Research,46* 187-210. doi:10.1146/annurev-matsci-070115-031624.

Seharing, Asliah & Azman, Abdul Hadi & Abdullah, Shahrum. 2020. "A review on integration of lightweight gradient lattice structures in additive manufacturing parts." *Advances in Mechanical Engineering* 168781402091695. doi:.10.1177/1687814020916951.

SelectStar. 2020. *What is MNIST? And why is it important?* June 22. Accessed May 10, 2022. https://selectstar-ai.medium.com/what-is-mnist-and-why-is-it-important-e9a269edbad5.

Shorten, C., Khoshgoftaar, T.M. 2019. "A survey on Image Data Augmentation for Deep Learning." *J Big Data 6* 60. doi:https://doi.org/10.1186/s40537-019-0197-0.

Tam, Kam-Ming Mark & Moosavi, Vahid & Mele, Tom & Block, Philippe. 2021. *Towards Trans-topological Design Exploration of Reticulated Equilibrium Shell Structures with Graph Convolution Networks.* In proceedings: In proceedings.

Tekõglu, T.N. Pronk and C. Ayas and C. 2017. "A quest for 2D lattice materials for actuation." *Journal of the Mechanics and Physics of Solids* 199-216.

Vogelsanger, Christopher, and Christian Federau. 2021. "Latent Space Analysis of VAE and Intro-VAE applied to 3-dimensional MR Brain Volumes of Multiple Sclerosis, Leukoencephalopathy, and Healthy Patients." *arXiv.* doi:10.48550/ARXIV.2101.06772.

Woshicver. 2022. *Data expansion with python + opencv.* January 15. Accessed April 24, 2022. https://pythonmana.com/2022/01/202201150747542076.html.

Xin Zheng, Yong Wang, Guoyou Wang, Jianguo Liu,. 2018. "Fast and robust segmentation of white blood cell images by self-supervised learning,." *Micron,Volume 107* 55-71.

Yann, LeCun, Cortes corinna, and Burges Christopher J.C. 1998. *THE MNIST DATABASE.* Accessed 05 15, 2022. http://yann.lecun.com/exdb/mnist/.

Yunlong Tang, Aidan Kurtz, Yaoyao Fiona Zhao. 2015. "Bidirectional Evolutionary Structural Optimization (BESO) based design method for lattice structure to be fabricated by additive manufacturing." *Computer-Aided Design* 91-101.

VIII APPENDICES

VAE Code : adapted from Generative Deep Learning with TensorFlow
by DeepLearning.AI (Authour : Laurence Moroney)
Link : https://www.coursera.org/learn/generative-deep-learning-with-tensorflow/home/info
and
(Dutta 2021)
Dutta, Balaram. 2021. Variational Autoencoders on Anime Faces. March 1. Accessed March 20, 2022. https://github.com/duttab49/tfwork/blob/main/variationalAutoencodersOnAnimeFace.py

SET UP

```
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow import keras
import datetime
from packaging import version
from keras import layers
import tensorboard
import cv2
import glob


import numpy as np
import os
import zipfile
import urllib.request
import random

import matplotlib.pyplot as plt
from IPython import display
```

```
!pip install wandb -qqq
import wandb
```

```
# Log in to your W&B account
wandb.login()
```

```
print(tf.__version__)
```

IMPORT DATASET FROM GOOGLE DRIVE

```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)

dataset_path = '/content/gdrive/MyDrive/Thesis/Dataset2'
```

```
np.random.seed(51)

BATCH_SIZE = 64
LATENT_DIM = 2
IMAGE_SIZE = 64
```

DATA PREPROCESSING

```

# open cv image augmentation


#load data

from google.colab.patches import cv2_imshow
images_dataset = []
images_path = glob.glob("/content/gdrive/MyDrive/Thesis/Dataset2/*.png")
for img_path in images_path:
  img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
  #Check import

 # random crop open cv
  scale = 0.65
  height, width = int(img.shape[0]*scale), int(img.shape[1]*scale)
  x = random.randint(0, img.shape[1] - int(width))
  y = random.randint(0, img.shape[0] - int(height))
  cropped = img[y:y+height, x:x+width]
  resized = cv2.resize(cropped, (img.shape[1], img.shape[0]))
  images_dataset.append(resized)

  scale = 0.75
  height, width = int(img.shape[0]*scale), int(img.shape[1]*scale)
  x = random.randint(0, img.shape[1] - int(width))
  y = random.randint(0, img.shape[0] - int(height))
  cropped = img[y:y+height, x:x+width]
  resized = cv2.resize(cropped, (img.shape[1], img.shape[0]))
  images_dataset.append(resized)


# Horizontal Flip
  def horizontal_flip(img, flag):
    if flag:
        return cv2.flip(img, 1)
    else:
        return img
  img = horizontal_flip(img, True)
  images_dataset.append(img)

#Vertical flip
  def vertical_flip(img, flag):
    if flag:
        return cv2.flip(img, 0)
    else:
        return img
  img = vertical_flip(img, True)
  images_dataset.append(img)

images_dataset.append(img)

print("Number of items in the list = ", len(images_dataset))
cv2_imshow(images_dataset[2])
cv2.waitKey(0)
cv2.destroyAllWindows()
print('Dimensions : ',img.shape)

images_dataset = np.expand_dims(np.array(images_dataset),-1)
print('Dimensions New : ',images_dataset.shape)
```

```
def get_dataset_slice_paths(image_dir):
  '''returns a list of paths to the image files'''
  image_file_list = os.listdir(image_dir)
  image_paths = [os.path.join(image_dir, fname) for fname in image_file_list]

  return image_paths


def map_image(image):
  '''preprocesses the images'''

  image = tf.cast(image, dtype=tf.float32)
  image = tf.image.resize(image, (IMAGE_SIZE, IMAGE_SIZE))
  image = image / 255.0
  image = tf.reshape(image, shape=(IMAGE_SIZE, IMAGE_SIZE, 1,))


  return image
```

SPLIT THE DATASET

```
# split the paths list into to training (80%) and validation sets(20%).
paths_len = images_dataset.shape[0]
train_paths_len = int(paths_len * 0.8)

train_paths = images_dataset[:train_paths_len]
val_paths = images_dataset[train_paths_len:]

# load the training image paths into tensors, create batches and shuffle
training_dataset = tf.data.Dataset.from_tensor_slices((train_paths))
training_dataset = training_dataset.map(map_image)
training_dataset = training_dataset.shuffle(1000).batch(BATCH_SIZE)
#training_dataset = training_dataset.map(augment)


# load the validation image paths into tensors and create batches
validation_dataset = tf.data.Dataset.from_tensor_slices((val_paths))
validation_dataset = validation_dataset.map(map_image)
validation_dataset = validation_dataset.batch(BATCH_SIZE)


print(f'number of batches in the training set: {len(training_dataset)}')
print(f'number of batches in the validation set: {len(validation_dataset)}')
```

```
i=0
for image in validation_dataset:
  i += 1

  print("Shape",image.shape)
```

VAE ARCHITECTURE

```
# Encoder_layers

def encoder_layers(inputs, latent_dim):
  #1
  x = tf.keras.layers.Conv2D(filters=32, kernel_size= 3, strides = 2, padding = 'same',
                             activation = 'relu', name = "encode_conv1")(inputs)
  x = tf.keras.layers.BatchNormalization()(x)

  #2
  x = tf.keras.layers.Conv2D(filters=64, kernel_size= 3, strides = 2, padding = 'same',
                             activation = 'relu', name = "encode_conv2")(x)
  batch_2 = tf.keras.layers.BatchNormalization()(x)
  #3
  x = tf.keras.layers.Flatten(name="encode_Flatten")(batch_2)
  #4
  x = tf.keras.layers.Dense(60, activation='relu', name="encode_dense")(x)
  x = tf.keras.layers.BatchNormalization()(x)
  #5
  mu = tf.keras.layers.Dense(latent_dim, name='latent_mu')(x)
  sigma = tf.keras.layers.Dense(latent_dim, name="latent_sigma")(x)

  return mu, sigma, batch_2.shape
```

```
#Sampling

class Sampling(tf.keras.layers.Layer):

  def call(self,inputs):

    mu, sigma = inputs

    batch = tf.shape(mu)[0]
    dim = tf.shape(mu)[1]

    epsilon = tf.keras.backend.random_normal(shape=(batch,dim))

    return mu + tf.exp(0.5*sigma) * epsilon
```

```
#Encoder_model

def encoder_model(latent_dim, input_shape):

  inputs = tf.keras.layers.Input(shape=input_shape)

  mu, sigma, conv_shape = encoder_layers(inputs, latent_dim=LATENT_DIM)

  z = Sampling()((mu,sigma))

  model = tf.keras.Model(inputs, outputs=[mu,sigma,z])

  return model, conv_shape
```

```
#Decoder_layers

def decoder_layers(inputs, conv_shape):

  units = conv_shape[1]*conv_shape[2]*conv_shape[3]
  #1
  x = tf.keras.layers.Dense(units, activation='relu', name="decode_dense1")(inputs)
  x = tf.keras.layers.BatchNormalization()(x)
  #2
  x = tf.keras.layers.Reshape((conv_shape[1],conv_shape[2],conv_shape[3]), name="decode_reshape")(x)
  #3
  x = tf.keras.layers.Conv2DTranspose(filters=64, kernel_size=3, strides=2, padding='same', activation='relu', name="decode_conv2d_2")(x)
  x = tf.keras.layers.BatchNormalization()(x)
  #4
  x = tf.keras.layers.Conv2DTranspose(filters=32, kernel_size=3, strides=2, padding='same', activation='relu', name="decode_conv2d_3")(x)
  x = tf.keras.layers.BatchNormalization()(x)
  #5
  x = tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=3, strides=1, padding='same', activation='sigmoid', name="decode_final")(x)



  return x
```

```
#Decoder_model

def decoder_model(latent_dim, conv_shape):

  inputs = tf.keras.layers.Input(shape=(latent_dim))

  outputs = decoder_layers(inputs, conv_shape)

  model = tf.keras.Model(inputs, outputs)

  return model
```

DEFINE THE LOSS FUNCTION

```
#KL reconstruction loss

def kl_reconstruction_loss(inputs,outputs,mu,sigma):

  kl_loss = 1 + sigma - tf.square(mu) - tf.math.exp(sigma)
  kl_loss = tf.reduce_mean(kl_loss) * -0.5

  return kl_loss
```

```
# Final VAE Model

def vae_model(encoder,decoder, input_shape):

  inputs = tf.keras.layers.Input(shape=input_shape)

  mu,sigma,z = encoder(inputs)

  reconstructed = decoder(z)

  model = tf.keras.Model(inputs=inputs, outputs=reconstructed)

  loss = kl_reconstruction_loss(inputs, z, mu, sigma)
  model.add_loss(loss)

  return model
```

```
# getting the different models

def get_models(input_shape, latent_dim):

   encoder, conv_shape = encoder_model(latent_dim=latent_dim, input_shape=input_shape)

   decoder = decoder_model(latent_dim=latent_dim, conv_shape=conv_shape)

   vae = vae_model(encoder, decoder, input_shape=input_shape)

   return encoder, decoder, vae

encoder, decoder, vae = get_models(input_shape = (64,64,1,), latent_dim=LATENT_DIM)

encoder.summary()
decoder.summary()
vae.summary()
```

DEFINE LOSS AND OPTIMIZERS

```
# Defining loss and optimizers

optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001)

loss_metric = tf.keras.metrics.Mean()

bce_loss = tf.keras.losses.BinaryCrossentropy()

val_acc_metric = keras.metrics.SparseCategoricalAccuracy()
```

```
#display images while training

def generate_and_save_images(model, epoch, step, test_input):

  # generate images from the test input
  predictions = model.predict(test_input)

  # plot the results
  fig = plt.figure(figsize=(8,8))

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0], cmap='gray')
      plt.axis('off')

  # tight_layout minimizes the overlap between 2 sub-plots
  fig.suptitle("epoch: {}, step: {}".format(epoch, step))
  plt.savefig('image_at_epoch_{:04d}_step{:04d}.png'.format(epoch, step))
  plt.show()
```

```
# Display the Latent Space

x_test = train_paths

# display a 2D plot of the digit classes in the latent space
z_test = encoder.predict(x_test, batch_size=BATCH_SIZE)
plt.figure(figsize=(6, 6))
plt.scatter(z_test[0], z_test[1], c= '#6667ab',
            alpha=.4, s=3**2)
plt.title("Projection of 2D Latent-Space ", size=20)

plt.show()
```

```
# Use TSNE only when ALtent_dim > 2

X = train_paths

X_encoded = encoder.predict(X)
X_encoded = np.array(X_encoded)
print(X_encoded.shape)
X_encoded = np.reshape(X_encoded,(-1,LATENT_DIM))
print(X_encoded.shape)


from sklearn.manifold import TSNE

m = TSNE(learning_rate=50)

tsne_features = m.fit_transform(X_encoded)
tsne_features[1:4,:]


import seaborn as sns

sns.scatterplot(x= tsne_features[:,0], y= tsne_features[:,1], data=X_encoded, legend= "auto")

plt.show()
```

TRAINING

```
random_vector_for_generation = tf.random.normal(shape=[16, LATENT_DIM])

epochs = 6000


generate_and_save_images(decoder,0,0, random_vector_for_generation)


run = wandb.init(project="Loss2", entity="nb42" )
wandb.config ={
            "epochs": epochs,
            "batch_size": BATCH_SIZE,
            "lr": 0.001,
            "dropout": random.uniform(0.01, 0.80),
            }

#loop

for epoch in range(epochs):
  print('start of epoch %d'%(epoch,))



  for step, x_batch_train in enumerate(training_dataset):
    with tf.GradientTape() as tape:

# pass batch of training data
       reconstructed = vae(x_batch_train)

#to measure loss
       flattened_inputs = tf.reshape(x_batch_train, shape=[-1])
       flattened_outputs = tf.reshape(reconstructed, shape=[-1])
       loss_reconstruction = bce_loss(flattened_inputs, flattened_outputs) * 4096

 # adding KLD regularization loss
       loss = loss_reconstruction + sum(vae.losses)


 # get the gradients and update the weights
       grads = tape.gradient(loss, vae.trainable_weights)
       optimizer.apply_gradients(zip(grads, vae.trainable_weights))

# compute the loss metric
       total_loss = loss_metric(loss)
       reconstruction_loss = loss_metric(loss_reconstruction)
       regularisation_loss = loss_metric(sum(vae.losses))



#wandb

       wandb.log({"total_loss": loss_metric.result().numpy(),
                  "reconstruction_loss": reconstruction_loss,
                  "regularisation_loss":regularisation_loss})
```

```
# display outputs every 100 steps and Latent space
       if step % 100 == 0:
         display.clear_output(wait=False)
         generate_and_save_images(decoder, epoch, step, random_vector_for_generation)
         print('Epoch: %s step: %s mean loss = %s reconstruction loss = %s ' % (epoch, step, loss_metric.result().numpy(), reconstruction_loss))




# display a 2D plot of the digit classes in the latent space


  for step, x_batch_val in enumerate(validation_dataset):
     validation = vae(x_batch_val)
     flattened_inputs = tf.reshape(x_batch_val, shape=[-1])
     flattened_outputs = tf.reshape(validation, shape=[-1])
     val_loss = bce_loss(flattened_inputs, flattened_outputs) * 4096

     validation_loss = loss_metric(val_loss)

     print('validation loss = %s ' % (validation_loss))

     wandb.log({"Val_loss": validation_loss})



wandb.finish()
```

PLOTTING RECONSTRUCTED IMAGES

```
def display_one_row(disp_images, offset, shape=(28,28)):
  '''Displays a row of images.'''
  for idx, image in enumerate(disp_images):
    plt.subplot(3, 10, offset + idx + 1)
    plt.xticks([])
    plt.yticks([])
    image = np.reshape(image, shape)
    image = np.squeeze(image, axis =2)
    plt.imshow(image, cmap = 'gray')


def display_results(disp_input_images, disp_predicted):
  '''Displays input and predicted images.'''
  plt.figure(figsize=(15, 5))
  display_one_row(disp_input_images, 0, shape=(IMAGE_SIZE,IMAGE_SIZE,1))
  display_one_row(disp_predicted, 20, shape=(IMAGE_SIZE,IMAGE_SIZE,1))




test_dataset = validation_dataset.take(1)
output_samples = []

for input_image in tfds.as_numpy(test_dataset):
      output_samples = input_image

idxs = np.random.choice(64, size=10)

vae_predicted = vae.predict(test_dataset)
display_results(output_samples[idxs], vae_predicted[idxs])
```

```
def plot_images(rows, cols, images, title):
    '''Displays images in a grid.'''
    grid = np.zeros(shape=(rows*64, cols*64, 1))
    for row in range(rows):
        for col in range(cols):
            grid[row*64:(row+1)*64, col*64:(col+1)*64, : ] = images[row*cols + col]
    grid = np.squeeze(grid, axis=2)
    plt.figure(figsize=(12,12), dpi= 300)
    plt.imshow(grid, cmap = 'gray')
    plt.title(title)
    plt.show()

# initialize random inputs
test_vector_for_generation = tf.random.normal(shape=[64, LATENT_DIM])

# get predictions from the decoder model
predictions= decoder.predict(test_vector_for_generation)

# plot the predictions
plot_images(8,8,predictions,'Generated Images')
```

SAVE THE MODEL

```
vae.save("thesisl2.h5")
```

```
     WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or
```

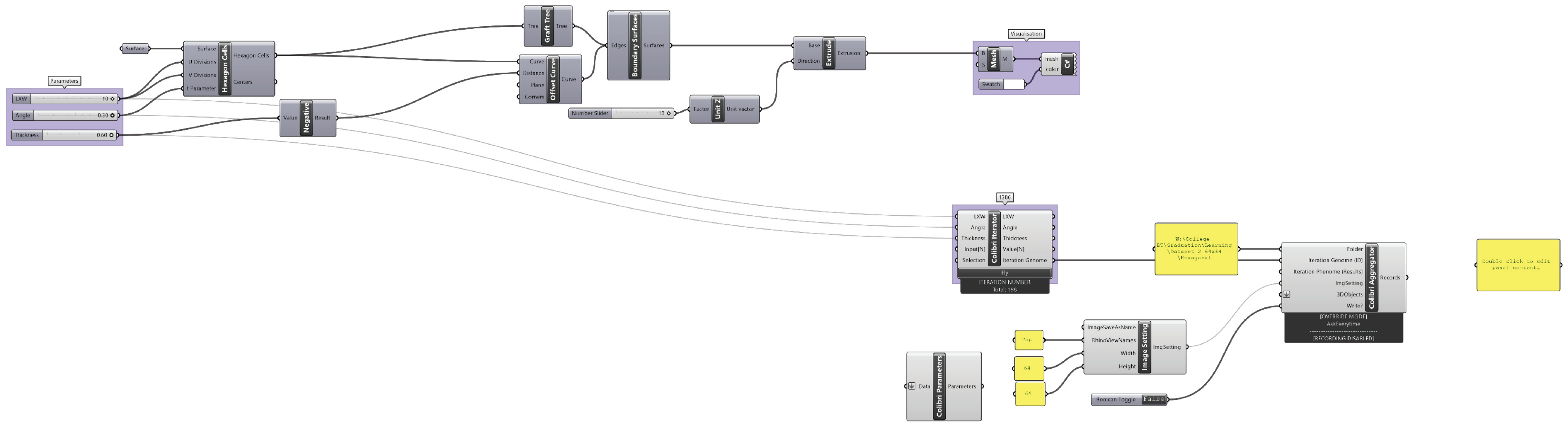
Timeline - APPENDIX I

Namrata Baruah - 5326664

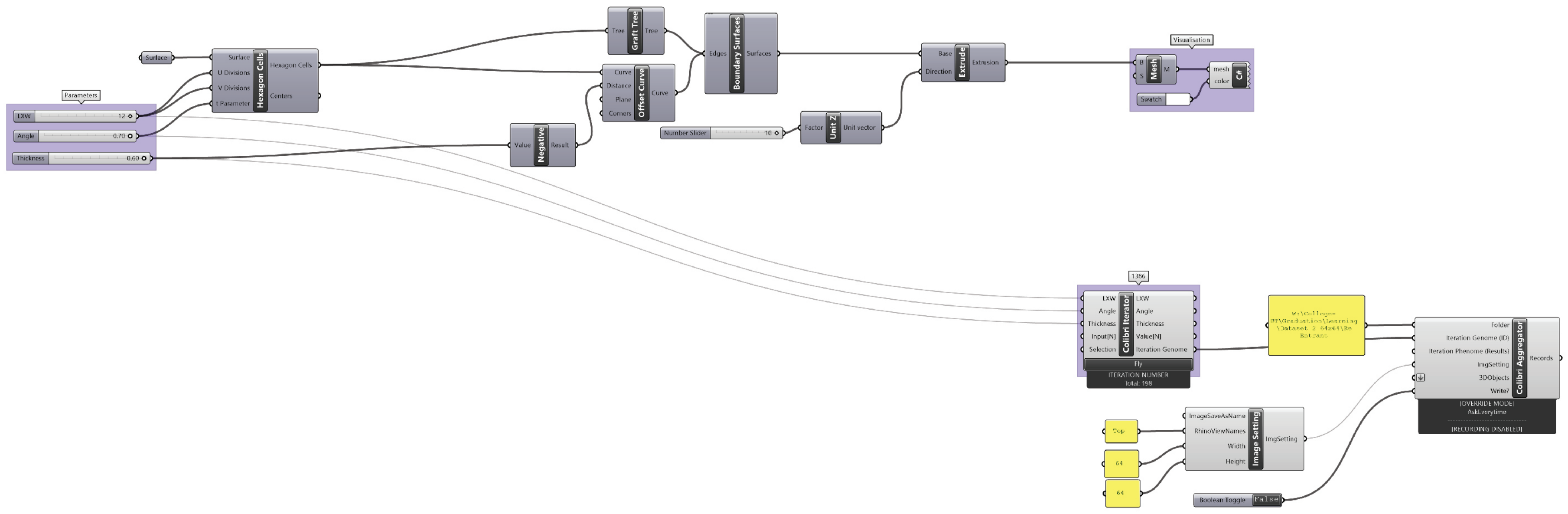
	1.03	1.04	1.05	1.06	1.07	1.08	1.09	1.10	2.01	2.02	2.03	2.04	2.05	2.06	2.07	2.08	2.09	2.10	3.01	3.02	3.03	3.04	3.05	3.06	3.07	3.08	3.09	3.10	4.01	4.02	4.03
Action	15/11/21	22/11/21	29/11/21	6/12/2021	13/12/21	20/12/21	27/12/21	3/1/2022	10/1/2022	17/1/22	24/1/22	31/1/22	7/2/2022	14/2/22	21/2/22	28/2/22	7/3/2022	14/3/22	21/3/22	28/3/22	4/4/2022	11/4/2022	18/4/22	25/4/22	2/5/2022	9/5/2022	16/5/22	23/5/22	30/5/22	6/6/2022	13/6/22
P1																														
P1 Presentation	▨																													
P2 : Literature Study and Research																														
Research on AI concepts	▨	▨	▨	▨																										
Studying Python and Deep learning		▨	▨	▨	▨	▨																								
Study the Basics of VAE					▨	▨	▨	▨																						
Literature Study				▨	▨	▨	▨	▨																						
Research on use of AI in Design						▨	▨	▨																						
Research on bio forms for creating datasets							▨	▨	▨																					
Write Research Plan									▨																					
Write report									▨	▨																				
P2 Presentation										▨																				
P3 : Creating Dataset																														
Test VAE on a benchmark dataset											▨	▨	▨																	
Explore the geometry for creating Artificial Datasets from cellular solids												▨	▨	▨	▨	▨														
Defining parameters for data augmentations													▨	▨	▨	▨														
Start testing the new dataset																▨	▨	▨	▨											
P3 Presentation																			▨											
P4 : Training and Testing																														
Train the AI model (optimise the loss function)																	▨	▨	▨	▨	▨	▨	▨	▨						
Explore the AI generated patterns																					▨	▨	▨	▨	▨					
Explore solutions for topology optimisation using the generated patterns																							▨	▨	▨	▨	▨	▨	▨		
Analyse results																								▨	▨					
Conclusion																										▨				
Write report																							▨	▨	▨	▨	▨			
P4 presentation																											▨			
P5 :Results																														
Improve Final Results																												▨	▨	▨	
Reflection on Research																														▨	▨
Recommendations																														▨	▨
P5 presentation																															▨

3.2.1 Lattice Patterns (a)

Hexagon

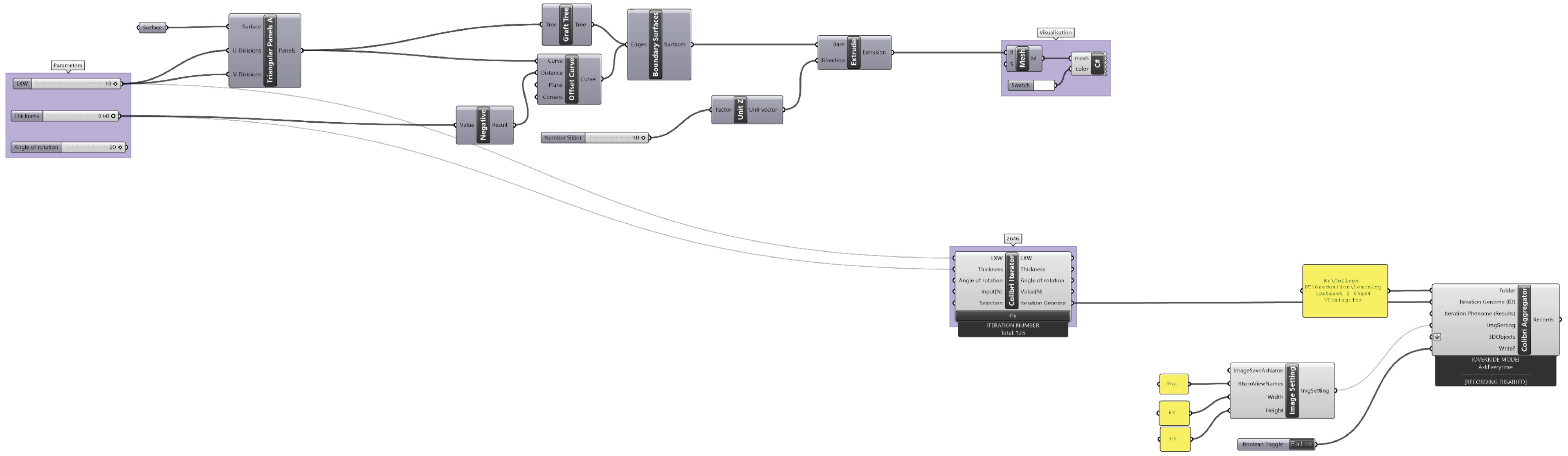


Re-Entrant

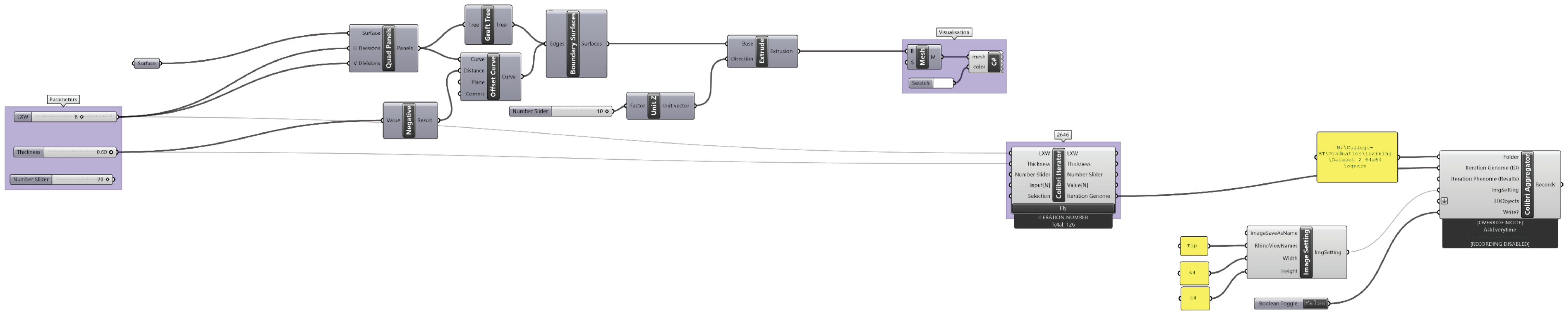


3.2.1 Lattice Patterns (b)

Triangular

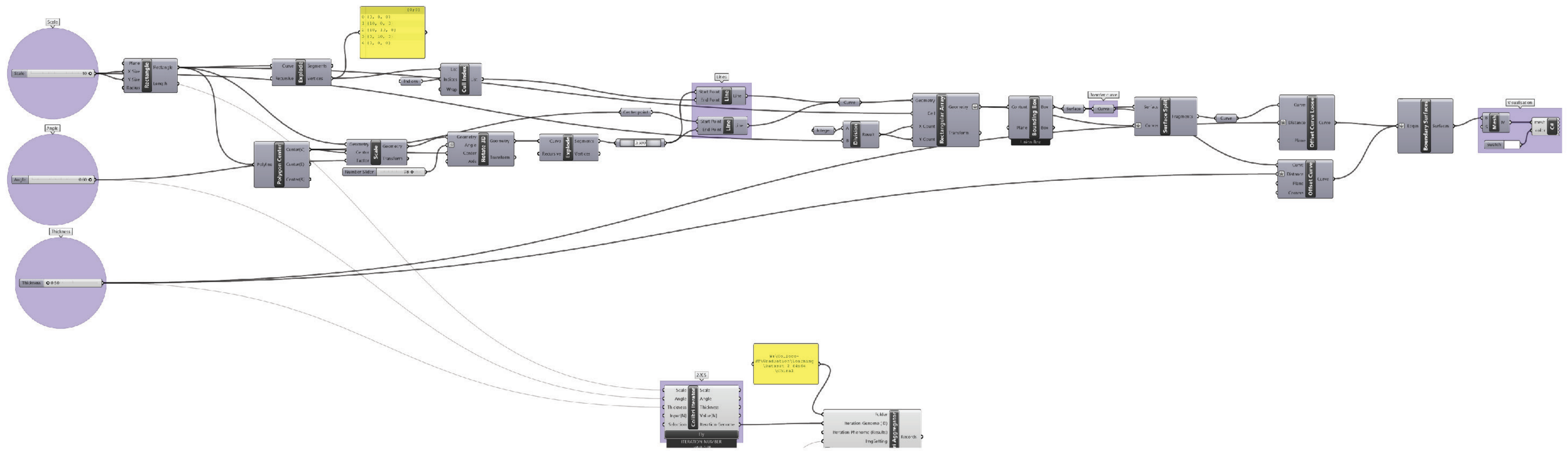


Square



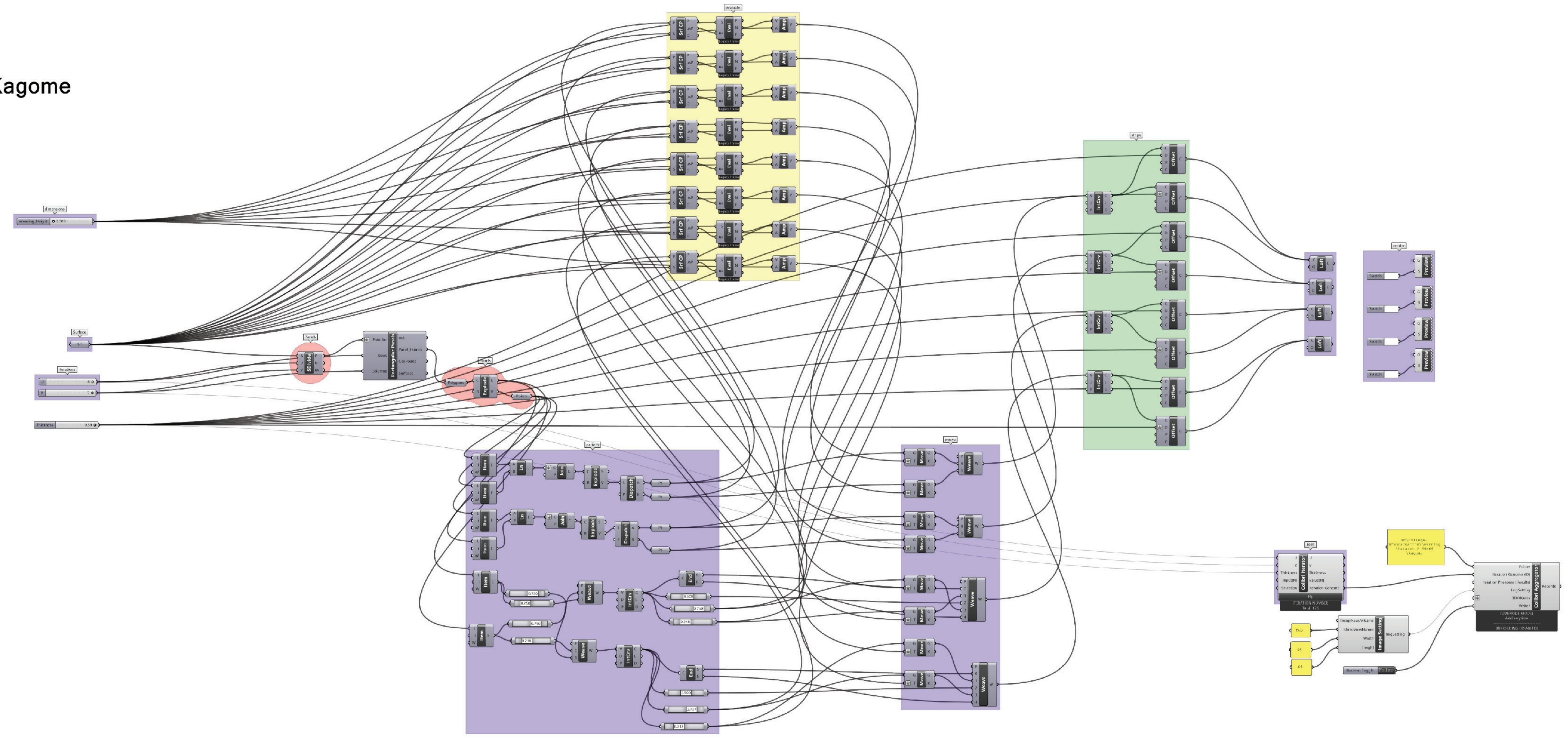
3.2.2 Square – Chiral Lattice patterns:

Chiral



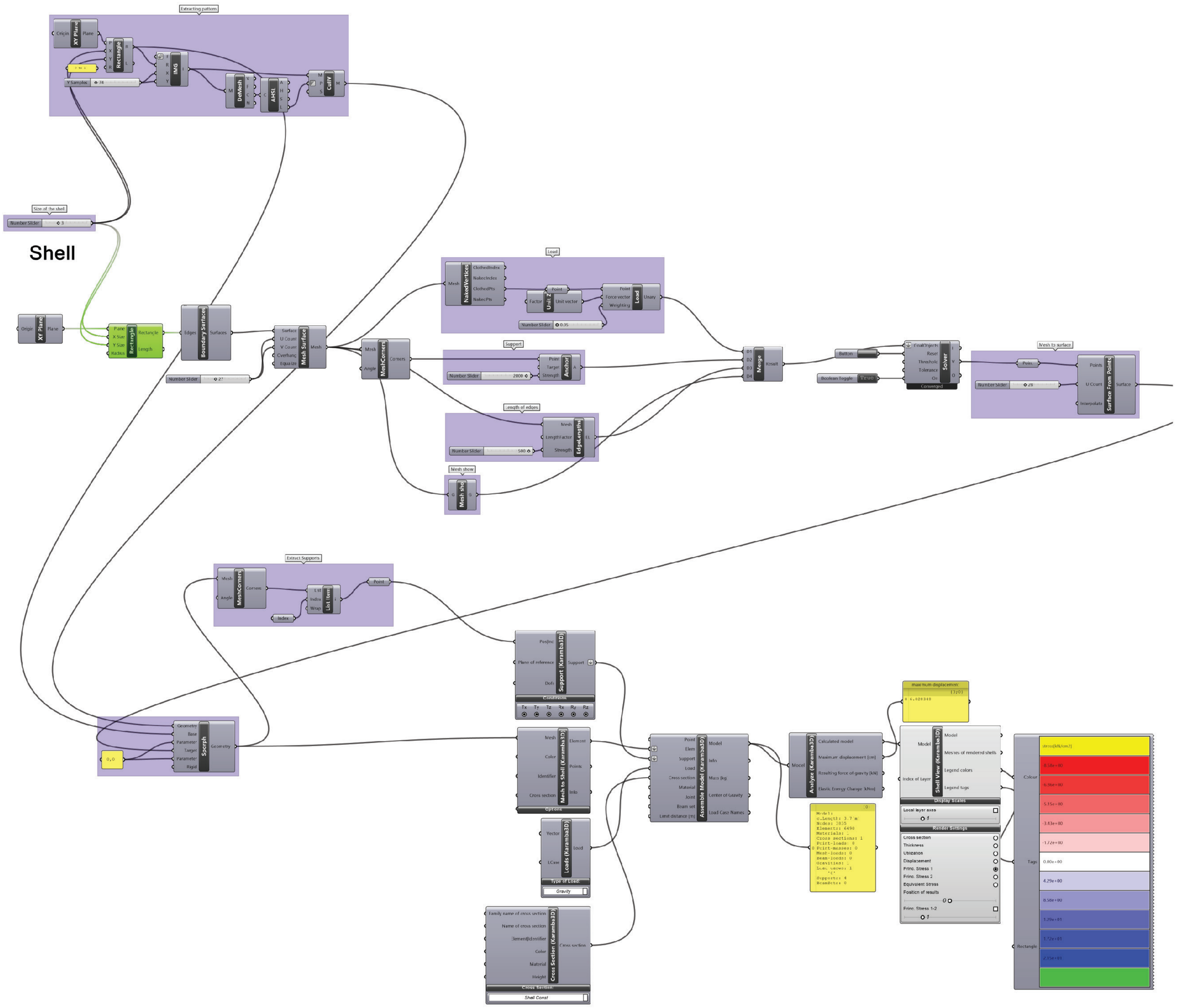
3.2.3 Kagome Lattice patterns

Kagome



5.1 Application in topology Exploration

Pattern



maximum displacement
{0;0}
0 6.028348

```
Model:
c.Length: 3.7 m
Nodes: 3835
Elements: 6490
Materials: 1
Cross sections: 1
Point-loads: 0
Point-masses: 0
Mesh-loads: 0
Beam-loads: 0
Gravities: 1
Load cases: 1
    '0'
Supports: 4
BeamSets: 0
```

stress[kN/cm2]
-8.58e+00
-6.86e+00
-5.15e+00
-3.43e+00
-1.72e+00
0.00e+00
4.29e+00
8.58e+00
1.29e+01
1.72e+01
2.15e+01