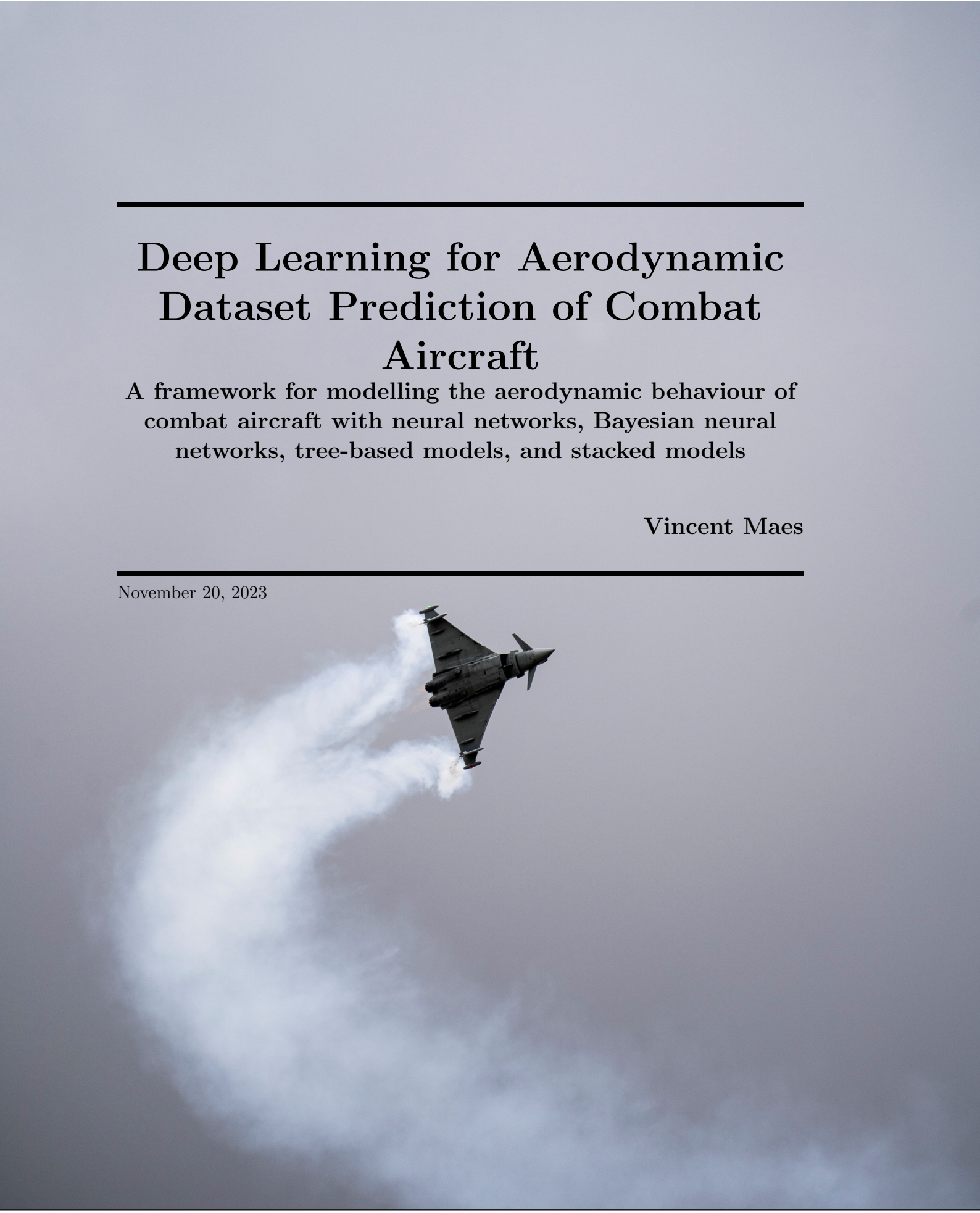
Deep Learning for Aerodynamic Dataset Prediction of Combat Aircraft

A framework for modelling the aerodynamic behaviour of combat aircraft with neural networks, Bayesian neural networks, tree-based models, and stacked models

Vincent Maes

November 20, 2023



Aerodynamics

Faculty of Aerospace Engineering

TU Delft

Delft University of Technology

Deep Learning for Aerodynamic Dataset Prediction of Combat Aircraft

A framework for modelling the aerodynamic behaviour of combat aircraft with neural networks, Bayesian neural networks, tree-based models, and stacked models

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering at Delft University of Technology

Vincent Maes

November 20, 2023

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology

DELFT UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF AERODYNAMICS

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance the thesis entitled **"Deep Learning for Aerodynamic Dataset Prediction of Combat Aircraft"** by **Vincent Maes** in fulfillment of the requirements for the degree of **Master of Science**.

Dated: November 20, 2023

Supervisors:

Dr. Anh Khoa Doan

Dr. Steven Hulshoff

Dr. Carmine Varriale

Rebecca Zahn

Preface

After performing my internship in the field of machine learning, and optimisation, I was determined to expand my knowledge of machine learning during my thesis. This thesis was proposed by Airbus Defence and Space in Manching, Germany. Fuelled by the excitement of combat aircraft after watching Top Gun Maverick, I don't think I could have dreamed of a more interesting topic than machine learning on the Eurofighter Typhoon.

Firstly, I want to thank my supervisor Dr. Anh Khoa Doan at Delft University of Technology for his expertise and for providing a comfortable learning environment. Furthermore, I want to thank my supervisor at Airbus, Rebecca Zahn for hiring me for the position and providing me with the freedom to research a wide variety of topics.

While reading this thesis, little prior knowledge is assumed in machine learning or combat aircraft aerodynamics. This is reflected in the page length, as I wanted to provide adequate reasoning and justification for my work.

I attribute this master thesis to my parents, Kathleen and Joris. Their support, encouragement, and love allowed me to strive during my studies and personal life. Also, my brother Ottavio, with whom I enjoyed many great moments during the thesis, his friendship will never be forgotten. Finally, my time in Ingolstadt would have never been so enjoyable without Michal and Martino, and I would like to thank them for their friendship.

Vincent Maes
Manching, November 2023

Abstract

The aerodynamic model of a combat aircraft is essential for its success and competitiveness compared to other combat aircraft. This thesis aims to research the most optimal machine learning model to create an aerodynamic model of a combat aircraft. The very large but still sparse, highly nonlinear dataset forms a challenge for using specific machine learning models. Tree-based models, artificial neural networks (ANNs), and Bayesian neural networks (BNNs) have been identified to be individually capable of modelling the aerodynamics of combat aircraft. For ANNs, additional research was performed into genetic algorithms, as a robust hyperparameter optimisation method. Transfer learning, which reduced the training time by around 14%. Finally, adding gradient information of the training data as an additional input, reduced the mean squared error (MSE) by 7%. Scalable BNNs, which used Bayes-by-backprop, were developed to handle the large dataset of over 300,000 data points. The uncertainties coming from the BNN were overconfident in the results compared to the tolerances of the dataset. The uncertainties showed only a weak correlation with the MSE of a given prediction. Finally, to leverage each of the model's advantages, a stacked model was created which improves the predictive performance on average by 27% compared to the best base model in terms of the MSE on the test dataset. With the stacked model implemented, each of the aerodynamic coefficients could be predicted with over 97% of the predictions of the test data within the tolerance. This is an average increase of 0.5% for each of the aerodynamic coefficients compared to the best base models. Due to the strict regulations related to this aerodynamic model, the machine learning model that is created cannot replace the aerodynamic model at this time. However, the implementation of this machine learning model allows engineers to design the aerodynamic model faster, and with greater precision.

Table of Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

ADAM	Adaptive Moment Estimation
AI	Artificial Intelligence
AMK	Aerodynamic Modification Kit
ANN	Artificial Neural Network
AOA	Angle of Attack
BNN	Bayesian Neural Networks
BP	Backpropagation
CAT	CatBoost
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
CRM	Common Research Model
DLR	Deutsches Zentrum für Luft- und Raumfahrt
DT	Decision Trees
ELBO	Evidence Lower Bound
FCAS	Future Air Combat System
FCS	Flight Control System
GAN	Generative Adversarial Network
GP	Gaussian Process
GPU	Graphics Processing Unit
LA	Laplace Approximation
LGB	LightGBM
LM	Levenberg-Marquardt
MAP	Maximum A Posteriori
MAPE	Mean Absolute Percentage Error
MCMC	Markov Chain Monte Carlo

MLP	Maximum Likelihood Estimation
MLP	Multilayer Perceptron
MSE	Mean Square Error
PINN	Physics-Informed Neural Networks
PWT	Percentage Within Tolerance
RBF	Radial Basis Function
RBFNN	Radial Basis Function Neural Network
ReLU	Rectified Linear Unit
RF	Random Forest
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SHARC	Subsonic High Alpha Research Concept
STD	Standard Deviation
SVI	Stochastic Variational Inference
SVR	Support Vector Regression
TL	Transfer Learning
UAV	Unmanned Aerial Vehicle
XGB	XGBoost

Coefficients

C_l	Aircraft Rolling Moment Coefficient
C_m	Aircraft Pitching Moment Coefficient
C_n	Aircraft Yawing Moment Coefficient
C_x	Aircraft Side Force Coefficient in x Direction
C_y	Aircraft Side Force Coefficient in y Direction
C_z	Aircraft Side Force Coefficient in z Direction

Latin Symbols

α	Aircraft Angle of Attack	deg
β	Aircraft Angle of Sideslip	deg
δ_1	Aircraft Trailing Edge Flap Deflection (left)	deg
δ_3	Aircraft Trailing Edge Flap Deflection (right)	deg
ϵ	Aircraft Leading Edge Flap Deflection	deg
η	Aircraft Foreplane Deflection	deg
μ	Mean	
ρ	Air Density	kgm^{-3}
σ	Standard Deviation	

ζ	Aircraft Rudder Deflection	deg

Miscellaneous

b	Aircraft Wingspan	m
c	Aircraft Reference Chord	m
E	Entropy	JK^{-1}
F_x	Aircraft Side Force in x-Direction	N
F_y	Aircraft Side Force in y-Direction	N
F_z	Aircraft Side Force in z-Direction	N
L	Total Rolling Moment	Nm
M	Mach Number	
M	Total Pitching Moment	Nm
N	Total Yawing Moment	Nm
q	Dynamic Pressure	Pa
S	Aircraft Reference Surface Area	m
V	Airspeed	ms^{-1}

Chapter 1

Introduction

In this chapter, the introduction of the research performed in this thesis is provided in section 1.1. Afterwards, the outlook of the thesis is provided in section 1.2.

1.1 Introduction

The aerodynamics of a combat aircraft is central to ensuring it is competitive and successful compared to other rival aircraft. An aerodynamically well-designed aircraft can have benefits such as increased top speed, manoeuvring, controllability, stability, and efficiency. Therefore, the aerodynamic design of an aircraft is of key importance. The latest European fighter aircraft, namely the Saab JAS 39 Gripen, Dassault Rafale, and Eurofighter Typhoon all have close coupled canards with delta-wing configurations and were all released between 1986 and 1994. The canard delta-wing configuration makes them very manoeuvrable at the cost of longitudinal stability in subsonic conditions [49]. This requires them to have a flight control system (FCS) which can keep the aircraft stable and manoeuvrable, as the pilot would not be capable of controlling it on their own. The FCS takes the pilots' inputs and transforms them into control surface deflections that produce the desired state of the aircraft. This is in contrast to non-flight control system aircraft, where the pilot's inputs are directly linked to the control surfaces. In this manner, the pilot needs to provide an input to the control stick to for example, change the direction of the aircraft, and the FCS decides which control surfaces need to be actuated, not only to perform the manoeuvre but also to maintain the stability and controllability of the aircraft. This requires the FCS to have information on the full aerodynamic behaviour of the aircraft which includes the pitching, rolling and yawing moment coefficients, as well as the side forces in x, y, and z direction. An aircraft has

many different operating conditions, such as its Mach number, angle of attack, sideslip angle, configurations, such as with full or empty armament, and control surface deflections, such as flaps, ailerons, rudder, slats, etc. Therefore, the aerodynamic model is complex and has many interacting factors.

To create such a complex aerodynamic model, aerodynamic data must be gathered on the desired aircraft, which is collected through a combination of wind tunnel tests, flight tests and high-accuracy computational fluid dynamics (CFD) simulations. Currently, the aerodynamic model of the aircraft is made using different interpolation techniques between the collected data points. This method requires around 10 people, working full-time for half a year, meaning a significant amount of resources is required to create such an aerodynamic model. With the onset of the Future Combat Air System (FCAS), a more scalable and future-proof method is being explored by allowing the aerodynamic model to be constructed, or aided by a machine learning model. This could greatly improve the predictive performance in the regions where little to no data has been gathered during wind tunnel tests, CFD and flight tests, but also greatly reduce the time needed to create the aerodynamic model. Since often a new aircraft configuration is produced, for example with a new armament, or upgraded design, the aerodynamic characteristics are changed, and the aerodynamic model needs to be updated. Machine learning models have the capability of being retrained quickly when a new aircraft configuration is presented, and could reduce the time spent on making an aerodynamic model significantly. There are many different machine learning models which could be considered in fulfilling the task described above.

Machine learning has exploded in popularity in recent years [115], although it is not a new field. Artificial intelligence and machine learning date back to the 1950s, and many examples exist in the industry of aerospace and combat aircraft using machine learning. The application that is presented in the thesis is new, as no previous research has been performed on this scale of dataset. This means that the thesis should not only aim to answer whether it is feasible to apply machine learning for the aerodynamic dataset prediction but also, if feasible, how well it performs compared to the traditional method. Furthermore, many different machine learning models exist, and determining which model would be the best fit for predicting the aerodynamic dataset will also be a key area of research in this thesis.

1.2 Thesis Outlook

Since the research that is aimed to be performed in this thesis relies heavily on the specific application of the aerodynamic dataset of a combat aircraft, a deeper look is taken at the aerodynamic dataset and model in chapter 2. Afterwards, in chapter 3, an in-depth literature study is performed which would allow one to formulate the research questions. The literature study paves the way to applying three types of machine learning models. Firstly, tree-based models are applied in chapter 4, after which artificial neural networks are applied in chapter 5, and finally in chapter 6, Bayesian neural networks are applied. These models have been carefully selected in the literature study based on their relevant research, and potential to predict the aerodynamic dataset. In chapter 7, a comparison is made between the different models which allows making a conclusion about which model performs best, and an ensemble model approach is presented, which aims to capture the best elements from each of the models, in a further improved model. Finally, in chapter 8, the conclusion presents the answers to the research questions, concludes this thesis, and provides directions for future work.

Chapter 2

Aerodynamic Dataset of Combat Aircraft

This chapter aims to provide the necessary background towards the research that is performed. Each combat aircraft with a flight control system requires an aerodynamic model of the aircraft it is controlling. The method for creating such a model and the rationale for having such an aerodynamic model is provided in section 2.1. Following this, the aerodynamic datasets used during the thesis to test the hypotheses that are posted, are provided in section 2.2. This section goes into greater detail on the type of data that is collected, and also the different preprocessing steps that are required before one can use the data to build a machine learning model. Finally, in section 2.3 the aerodynamics of the type of combat aircraft that is considered is further elaborated. This is to provide the reader with a better understanding of the complex aerodynamics of the aircraft, and which elements of the dataset and aerodynamic model make this a challenging task worth investigating.

2.1 Aerodynamic Model of the Eurofighter Typhoon

In this section, a closer look is taken at the data that is gathered on the Eurofighter Typhoon to aerodynamically describe it, and how this data is used to safely control the aircraft. This section is split up into two parts. Firstly, the current aerodynamic model of the Eurofighter is presented and explained in subsection 2.1.1. Afterwards, the desired aerodynamic model is explained in subsection 2.1.2.

2.1.1 Current Aerodynamic Model

In Figure 2.1 an image of the Eurofighter Typhoon is provided. The Eurofighter Typhoon is a multirole fighter aircraft designed by the British, Spanish, Italians, and Germans. The Eurofighter Typhoon or just Eurofighter entered service in 2003, and since then more than 500 aircraft have been built. The introduction of an aerodynamic modification kit (AMK) in 2015, which was developed as a part of the Eurofighter Enhanced Manoeuvrability program, has the effect of increasing the maximum lift and the manoeuvrability in high-speed turns significantly, the effects of which are described well in [49]. Combined with continuous development and the addition of new weaponry such as tanks and payloads to the Eurofighter, continuous updates are made towards the aerodynamic model of the aircraft.

The aerodynamic model forms a key component of the Eurofighter's development as the canard and delta wing configuration is known to be unstable in the subsonic operating window. Therefore, a flight control system is integrated, for which the aerodynamic model of the aircraft is one of its inputs. This flight control system is a system on the aircraft which performs two primary functions:

- Stabilization of the unstable aircraft
- Limits the flight envelope such that the pilot's input does not, for example, overload the airframe, stall the aircraft, etc.

This aerodynamic model describes the aircraft behaviour in terms of pitching C_m, rolling C_l and yawing C_n moment coefficients, as well as the side force coefficients in x, y, and z directions, C_x, C_y, C_z respectively. This output is provided for each possible combination of control surface deflections, operating conditions and aircraft configuration. Without this, the flight control system would not be capable of predicting what the aircraft state will be when the pilot provides a control input, and not be able to know what control surfaces need to be actuated for it to continue stable flight. Besides the flight control system, the aerodynamic model is also used to estimate the aerodynamic loads of the aircraft and thus plays a key role in the structural design of the aircraft. Furthermore, during flight simulation for training pilots, or mission analysis, the aerodynamic model is an input towards the simulation model of the aircraft. The Eurofighter is a combat aircraft and should be capable of operating at the very limits of its capabilities while still guaranteeing safe flight, and this requires a very detailed aerodynamic model.

The equations for the aerodynamic coefficients, which were discussed above, are shown below. The moment coefficient can be seen in Equation 2.1, where M is the total pitching moment around the centre of gravity, while q $(=\frac{1}{2}\rho V^2)$ is the dynamic pressure of the air, S the reference surface area, and c is the chord of the aircraft wing. For the rolling

moment coefficient, seen in Equation 2.2, L is the total rolling moment, while for the yawing moment coefficient, seen in Equation 2.3, N is the total yawing moment, with the reference length being the wingspan b. The side force coefficients are simply the total force F in each direction, divided by the dynamic pressure and wing surface area. The two most important side force coefficients are the C_x, shown in Equation 2.4, which is very highly correlated towards the drag coefficient, and C_z, shown in Equation 2.6, which is highly correlated with the lift coefficient. The side force coefficient C_y, shown in Equation 2.5, has a small influence on the aircraft behaviour, although not negligible, it is not critical for the aircraft behaviour.

$$C_m = \frac{M}{qSc} \quad (2.1) \qquad C_l = \frac{L}{qSb} \quad (2.2) \qquad C_n = \frac{N}{qSb} \quad (2.3)$$

$$C_x = \frac{F_x}{qS} \quad (2.4) \qquad C_y = \frac{F_y}{qS} \quad (2.5) \qquad C_z = \frac{F_z}{qS} \quad (2.6)$$

For the input values of the model, one can see from Figure 2.1 that there are two flow direction parameters, the angle of attack α and sideslip angle β, and one additional flow parameter the Mach number. For the control surface deflections, there is firstly the foreplane (or canard) deflection, secondly the leading edge flap deflection ε, thirdly the rudder deflection ζ, and finally the main flap deflection δ. The flap deflection is split up into a left and right flap deflection δ_1, and δ_3 since the aircraft is capable of operating both independently to perform manoeuvres. These inputs form independent input features towards the model.

Currently, the aerodynamic model is created by interpolating between the available wind tunnel, computational fluid dynamic (CFD), and flight test data. With the CFD and flight test data being a small portion of the overall dataset. This is done by a team of engineers who manually verify the data and perform the interpolation between the control surface deflections and operating conditions which are available. Furthermore, an extrapolation is made towards the boundaries of the domain where no data could be gathered. The current method provides fast predictions. However, it has the downside of requiring a lot of manual work to set up the model and perform the quality assurance of the model. Furthermore, interpolation can be considered the limit of regression when the noise of the data that is being interpolated goes to zero [121]. However, this assumption of zero noise cannot be made for the wind tunnel data, which is used currently for the aerodynamic model. This is because the wind tunnel data has many sources of errors, both systematic and random. Therefore, tolerances are implemented around the model which ensures that the uncertainty in the aerodynamic data is provided towards the users of the aerodynamic model. Going from an interpolative model towards a regression-type model is desirable to be more robust against the noise of the wind tunnel data.

Figure 2.1: Image of a Eurofighter Typhoon in flight [2] with annotations of the control surface deflections and flow variables.

The process of making the aerodynamic model using the method above is very tedious and costly, as a lot of working hours are involved in the process. To give a perspective, building an aerodynamic model can take between 3 and 6 months, with around 10 people working full-time on the task. Furthermore, one can imagine that when a new payload is added to the external holders of the aircraft, the aerodynamic behaviour of the aircraft is changed, and the model needs to be adjusted. One can also think of cases where one of the payloads is released, and an asymmetric loading configuration is obtained, which also severely alters the aerodynamic behaviour of the aircraft. Therefore, the ultimate goal would be to have a model which is capable of being updated very quickly, without much time having to be spent by the engineers in updating it for each new aircraft configuration, or operational condition that is presented. With the rise in popularity of artificial intelligence and the flexibility with which these models can be applied and retrained, they form an obvious candidate to be explored for this purpose.

2.1.2 Desired Aerodynamic Model

The aerodynamic model that should be created by using a machine learning model is shown in Figure 2.2. The inputs of the model are all the flow variables and control surface deflections shown in Figure 2.2. The outputs are the pitching moment (C_m), rolling moment (C_l), yawing moment (C_n) coefficients, and side force coefficients in x, y, and z-direction (C_x, C_y, C_z). This model will be created for a constant store group configuration, which indicates which armament and payload the Eurofighter is carrying.

Furthermore, the model does not necessarily need to predict all of the aerodynamic coefficients simultaneously, but can also predict each of the output variables separately, which is something that the research of this thesis should show the potential benefit of.

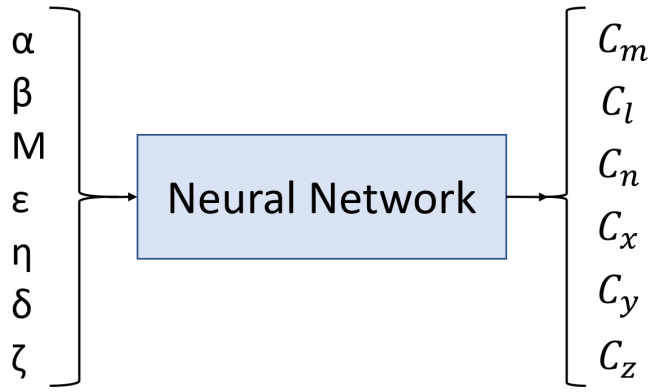


Figure 2.2: Inputs and outputs of the machine learning model that is desired to be created.

The desired aerodynamic model captures all the aerodynamic effects of the Eurofighter's behaviour. The aerodynamics of a combat aircraft is highly nonlinear and is described in more detail in section 2.3. Due to its non-linearity, making an aerodynamic model is a challenging task. However, to provide a better measure for the aerodynamic model than simply, "capture all the aerodynamic effects" the tolerances are introduced. The aerodynamic model has tolerances which the model needs to fall within. These tolerances are carefully chosen such that the parameters of the flight control system can still control the aircraft when there is a deviation of the aerodynamic model within the tolerances. The need for tolerances is due to the fact that the data coming into the model has certain uncertainties. For example, the wind tunnel data has measurement errors, and the repeatability of a data point is not exact. The tolerances are such that if a repeat of a wind tunnel polar is made, the new polar falls within the tolerance band. One can imagine that when the Mach number increases, the wind tunnel error becomes larger. Furthermore, at higher angles of attack, the aerodynamics become even more unsteady and the wind tunnel gives more sources of error which might result in larger tolerances at these angles of attack. Therefore, to summarise, the tolerances are different for each output variable (C_m, C_l, C_n, C_x, C_y, C_z), and vary with Mach number and angle of attack. Due to confidentiality, the values of the tolerances cannot be provided.

2.2 The Aerodynamic Dataset Used in the Thesis

The data which will be used during the thesis consists entirely of wind tunnel data. Although more data is available on the Eurofighter, such as CFD test and flight test data, the thesis forms the initial attempt at creating an aerodynamic model as has been described in subsection 2.1.1, and therefore only the wind tunnel data is used to create a model at this stage. The wind tunnel data which is gathered consists of the angle of attack (α) sweeps and the angle of sideslip (β) sweeps while keeping the other input parameters constant. However, sometimes variations in the wind tunnel cause the angle of attack, sideslip angle, or Mach number to vary slightly. A single angle of attack sweep, or angle of sideslip sweep is called a polar. This data is in the form of a table, an example of which is given in Table 2.2. Where the angle of attack is swept from -5 to 20 degrees with a step size of 0.1 degrees. While the other input parameters which are seen, are held constant, with the outputs C_m and C_l being shown in the last two columns. In the data, there are mostly α sweeps as these are the most important for longitudinal stability. While there are fewer β polars, which are most important for lateral stability. The ratio of α to β polars differs between datasets, but usually, they are distributed 90-10%.

Table 2.1: Table showing the data of an example angle of attack sweep with fictive pitching and rolling moment coefficient.

$\boldsymbol{\alpha}$ [deg]	$\boldsymbol{\beta}$ [deg]	M [-]	$\boldsymbol{\eta}$ [deg]	$\boldsymbol{\epsilon}$ [deg]	$\boldsymbol{\delta}$ [deg]	$\boldsymbol{\zeta}$ [deg]	$\boldsymbol{C_m}$ [-]	$\boldsymbol{C_l}$ [-]
-5	-5	0.7	0	0	-10	0	-0.1	0.5
-4.9	-5	0.7	0	0	-10	0	-0.05	0.55
-4.8	-5	0.7	0	0	-10	0	-0.2	0.45
...	...	...	...	...	...	...	...	...
19.8	-5	0.7	0	0	-10	0	0.1	0.6
19.9	-5	0.7	0	0	-10	0	0.2	0.7
20	-5	0.7	0	0	-10	0	0.3	0.8

Since the angle of attack, and sideslip angle are varied in much smaller intervals than the other parameters, the input data is skewed heavily in terms of their prominence. The combat aircraft wind tunnel data is gathered at discrete locations. This means that in between these increments, an interpolation is required. One of the main goals of the model, which will be developed during the thesis, is to be able to replace the interpolation with a general model which provides a prediction on the output variable continuously over the entire flight envelope. This reduces the amount of wind tunnel data that is required greatly and also allows for better mapping of the aerodynamics. The goals of the model that will be built can be summarised below:

- Provide a prediction for C_m, C_l, C_n, C_x, C_y, C_z for each of the model inputs shown in Figure 2.2.

- The model should fall within a tolerance band of the original wind tunnel data, which will be specified by the quality assurance department.

2.2.1 Data Preprocessing

In this subsection, the preprocessing steps that need to be taken to use the wind tunnel data for a machine learning model are provided.

Splitting the Data

Splitting the data into three batches is important, as an unbiased evaluation of the machine learning model is required to be able to assess its performance. The training data set is used to train the model. Then there are two datasets used for evaluating the model and optimising the hyperparameters. The validation data set is used to optimise the hyperparameters of the model. The test data set is a separate data set which is used to evaluate the trained model [5]. The percentages in which the data is split are shown below and are based on standard machine learning practices, where the decision is made to have 10% of the dataset as the test dataset [85].

1. Train = 81%
2. Validation = 9%
3. Test = 10%

It is important to evaluate whether the model is overfitting or underfitting the data. Overfitting is when the model is trying to fit the model to the training data set perfectly, but does not learn the underlying distribution. An underfitting model, did not learn enough from the training data set, and thus will perform badly on the test data set. A good fit is when the model learns the underlying distribution in a way that is generalisable towards new data that the model has not seen yet. The metrics that can be used to evaluate the model performance are provided in subsection 2.2.3.

The splitting of data into train, validation, and test data is always performed polar-wise, and not point-wise. This difference has a big impact on the training of the model, and there is a key reason for making this distinction. If the model would be split point-wise into test and train, it would look like the situation in Figure 2.3, where a single polar is displayed of the dataset that has been split pointwise. This results in testing points being located directly next to train points, and therefore the model performance on the test dataset, is nearly identical to the performance on the training data, as their proximity is so close. This does not provide a good measure of the models' generalizability, as the

model has seen training data that is very similar to the testing data. Therefore, it is always important to split the aerodynamic dataset, polar by polar, such that in the test data there is a polar with input parameters that the model has never seen before, and a good evaluation can be made.

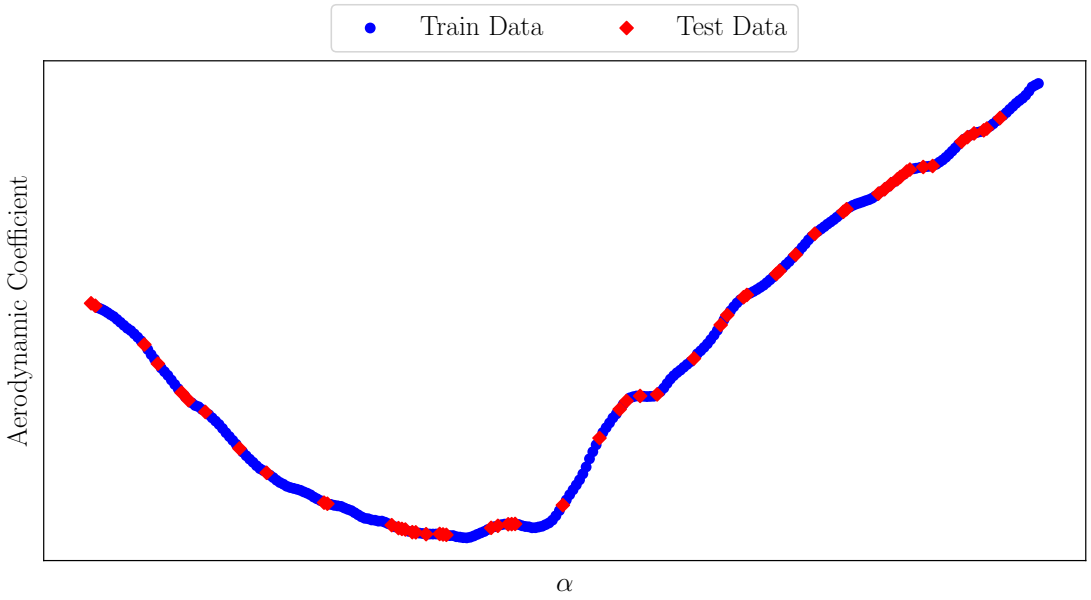


Figure 2.3: Point-wise train and test split of an example wind tunnel polar.

Duplicate Entries

In this subsubsection, a short comparison is made on whether removing duplicate data points from the training dataset is beneficial for training the machine learning models.

In the dataset for a specific configuration of a combat aircraft, there exist duplicate entries. This means that in one set of input parameters, which are the control surface deflections, and flow parameters, there exist multiple polars which provide information on the aerodynamic coefficients of the aircraft. An example of this can be seen in Figure 2.4, where for a particular set of parameters, three polars are available for training the data. Duplicate data polars are present in the dataset since repeats are performed in the wind tunnel to make sure that for example, the wind tunnel provides similar results at the beginning of the testing campaign, as at the end. There are a few reasons why duplicate entries can form a problem for evaluating the dataset. Firstly, when the random split is made between training and testing polars, there might be some polars that go into the training dataset, for which a very similar polar is present in the test dataset. This gives a false impression of the performance of the machine learning model on the test dataset,

as the model is essentially predicting a training polar, and not a polar that the model has never seen. Furthermore, one can see that all the polars are within the tolerances, but they are not identical. When for the same input parameters the machine learning model receives two or more different target values, the training of the model could be negatively influenced. Furthermore, if the same polar is seen multiple times during the training, the training process might place too much importance on one data polar, which is not desirable.

The duplicate polars can be handled in multiple ways. Firstly, one can remove the duplicate polars. However, the question then becomes, which polars should be removed and which should be kept. Therefore, the choice is made to average the data polars. Meaning that the three polars that are present in Figure 2.4, are replaced by a single polar seen in black in the figure. One can see from Figure 2.4 that in certain locations the mean value flattens out certain bumps in the dataset that are present in the individual polars. It is assumed that the machine learning model, would not have been capable of predicting these bumps anyway if all polars were fed into the model. The largest trends in the aerodynamic coefficients that all polars exhibit will be present in the mean value, and these are aimed to be learned by the model.

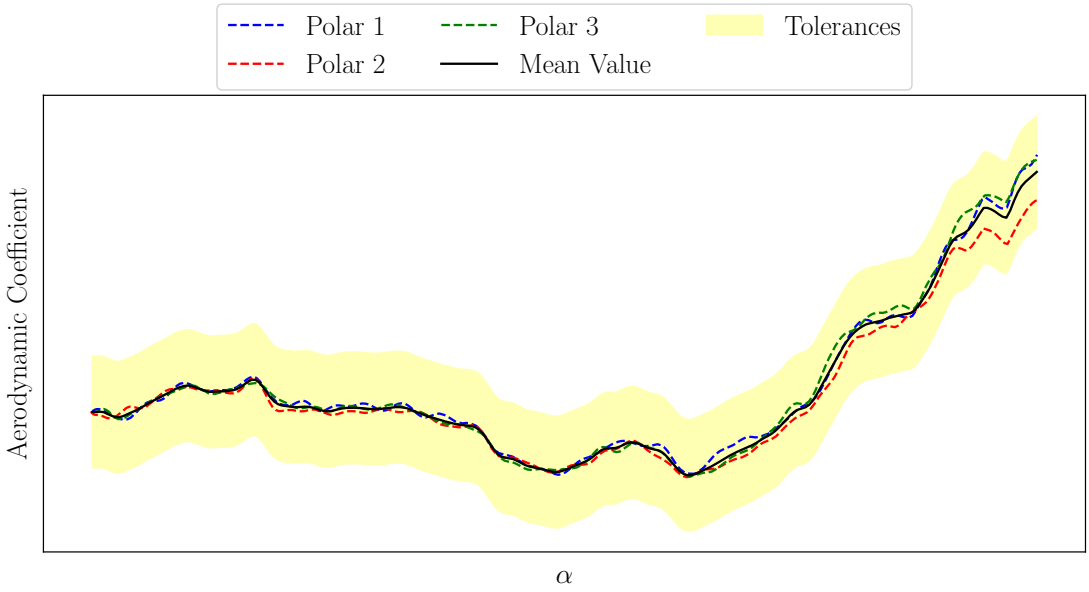


Figure 2.4: Comparison of training polars with identical input parameters but non-identical values for the aerodynamic coefficient.

Normalisation

Normalisation is a method used in machine learning to make sure all the input features are on a similar scale. This has the benefit that it stabilises the gradient descent step used in the convergence of the model [27]. If the gradient descent step is more stable, a larger learning rate can be used, and the network can be trained faster. However, if one thinks about the scale of different input features, the Mach number for example is between 0 and 2, while the flap deflections are for example between +40 and -40 degrees. Normalising ensures that one of the features does not on average have a significantly larger value than another feature. This is important so the model does not put extra importance on the feature with a larger value. There are a few different options on how to normalise or standardise the input data. In this subsection, focus is given to two methods: Minimum-maximum normalisation and Z-score normalisation.

Minimum-maximum normalisation re-scales the input features in between a value of 0 and 1 [5]. Z-score normalisation can also be called standardization, where the input features are scaled to unit variance (σ^2) and mean (μ) of zero. This can be achieved using Equation 2.7. In the thesis, z-score normalisation is chosen as it is less sensitive to outliers. In some of the input parameters, there are only a very limited number of data points for the largest deflections, and most are concentrated at the zero deflection of the control surfaces. Min-max scaling would result in these important measurements having a small difference between them in their input parameters, while for the z-score normalisation, these would be handled better.

$$\bar{x} = \frac{x - \mu}{\sigma} \tag{2.7}$$

Shuffling the Input Data

Another aspect of preprocessing a dataset is whether to use shuffling when loading data into the model. Shuffling the data will give a random order to the data points and can be useful when there is a specific order to the data such as always having increasing angles of attack, the model might start to learn the order of the data set. Therefore, shuffling the data can reduce the overfitting of the model. Furthermore, it can help the optimiser avoid getting stuck in a local minimum when it receives data from only a specific subset of the entire data set. Furthermore, it makes sure that each batch during the batch learning of the model is a good representation of the entire dataset. Shuffling the data means that after splitting the polars into the train, test and validation sets, the data within the training set is shuffled such that the tabular data is no longer ordered with all the points from polar 1, followed by polar 2, etc. Instead, one might have a point from polar 50 at $\alpha = 5$ deg, followed by a point from polar 21 at $\alpha = -10.2$ deg for example.

2.2.2 Datasets Used in Thesis

In this subsection, the actual datasets that will be used during the thesis are discussed, and shown. Furthermore, not every machine learning model that is trained during the thesis uses the full dataset due to time restrictions. In this subsection, the different datasets are labelled such that they can be referred to in their respective sections.

Firstly, the input parameters are provided below which are the ones that could be seen in Figure 2.1.

- Angle of Attack: α
- Angle of Sideslip: β
- Mach number: M
- Foreplane Deflection Angle: η
- Leading Edge Slats Deflection Angle: ϵ
- Trailing Edge Flap Deflection Angle (left): δ_1
- Trailing Edge Flap Deflection Angle (right): δ_3
- Rudder Deflection Angle: ζ

The output parameters are provided below, which are the ones that are discussed in section 2.1.

- Pitching Moment Coefficient: C_m
- Rolling Moment Coefficient: C_l
- Yawing Moment Coefficient: C_n
- Side Force Coefficient in x-direction: C_x
- Side Force Coefficient in y-direction: C_y
- Side Force Coefficient in z-direction: C_z

There are additional input parameters that define the combat aircraft. These parameters are present in the aerodynamic database, but not within the aerodynamic model. This is because a single aerodynamic model is created for each of these parameters. These include the wing specification, which is a categorical value, which refers to a wing with

for example a thicker trailing edge, or sharper leading edge. These are also present in the canard, fin and body. Different customers of the combat aircraft have for example requested a slight modification for which a new wind tunnel testing campaign is held. The most important of these categorical parameters has not been mentioned yet and is called the store code parameter. A store code uniquely defines the combat aircraft payload it is carrying. Therefore, each different combination of weapons, fuel tanks, or empty payload attachment points is summarised in a store configuration, and each different store configuration can be considered to have a different aircraft aerodynamic behaviour. The differences between different store code models depend on how many different payloads are under the wing. However, due to the nonlinear nature of the aerodynamics of the aircraft, the addition or removal of a payload can have an unexpected change in the aerodynamics. The store configuration is one of the key reasons that a new wind tunnel campaign is held, as a new combination of weapons and payloads results in an aircraft that has never been aerodynamically modelled, meaning the flight control system might not be able to control it, and the aircraft is not airworthy. Therefore, when a new weapon is launched on the market, which has been a significant number since its initial launch in 2003, a new or updated aerodynamic model is created. This means that for the thesis, a selection from a wide range of aerodynamic datasets can be chosen, with different, wing, fins, canard, and store codes, so the question arises, which one should be chosen?

The strategy for choosing the primary dataset was based on the following rationale. Since this is the first attempt at aerodynamically modelling the complete aerodynamic dataset with a machine learning model, the feasibility is not entirely sure. Therefore, one must provide the machine learning model with a dataset that has the highest chance of success. Therefore, the dataset with the highest number of data points is searched. As was discussed in section 2.1, there exists an aerodynamic modification kit (AMK) for the combat aircraft which is studied in this thesis. Therefore, the extra requirement was to use AMK data, which ensures the novelty of the research being performed in this thesis. The dataset that is found is the AMK aircraft, with baseline store configuration, meaning that the amount of underwing objects is limited.

The second dataset that is used can provide comparisons based on having a low or high number of payloads underneath the wing. Therefore, the second dataset consists of a combat aircraft with the same wing, body, fin, and AMK but with a full weapons load-out. The exact number or specific payloads that are part of this dataset cannot be provided due to confidentiality. However, two images that compare an empty configuration as was discussed for dataset 1 and a full store configuration can be shown. This is done in Figure 2.5 for dataset 1, and Figure 2.6 for dataset 2.

The final dataset is equal to the first dataset, besides the fact that it has only five input features. The flap deflections and rudder deflection are neglected to reduce the number of data points and input features to provide a faster method of evaluating a

Figure 2.5: Example figure of an empty store configuration of the Eurofighter, comparable to the baseline configuration.

Figure 2.6: Example figure of a full armament Eurofighter store configuration. Image Credit: Creative Commons

model, reducing the computational cost of training a model. This dataset can then be used to quickly gain an understanding of the underlying effects of each of the model hyperparameters or to form an initial test of a new model implementation.

The violin plots in Figure 2.7 show the distribution of the different parameters in the first and second datasets. From this, one can see that the angle of attack is a continuous variable, also the sideslip angle is continuous in some cases however, it is dominated by the discrete values from the angle of attack sweeps. Finally, the left and right flap deflection might seem equal at first sight, however, they are very slightly offset because there are some polars included for settings where the left and right flap do not have the same value. For the third dataset, the last three inputs of dataset 1 are removed, and only the first five inputs remain.

In Figure 2.8 the violin plot for the full armament dataset is provided. One can see that in this configuration, no rudder deflections ζ are measured, and thus the last columns are only 0. Furthermore, one can see that there are more asymmetric flap settings which are measured in this configuration due to there being a larger offset in the left and right flap deflections.

The total number of polars and points for each of the datasets is shown in Table 2.2. One can see that between datasets 1 and 3, there is a significant difference in the total number of polars that are tested. This is because the baseline has the most amount of data points out of any dataset and therefore every other dataset will have fewer points. In dataset 3, one can see that removing the last 3 input parameters and constraining them to only have the baseline value of no deflection, results in a reduction of 1000 polars compared to dataset 1, which should help with reducing the computational cost.

A final check is performed on dataset 1, on whether the input parameters are uncorrelated

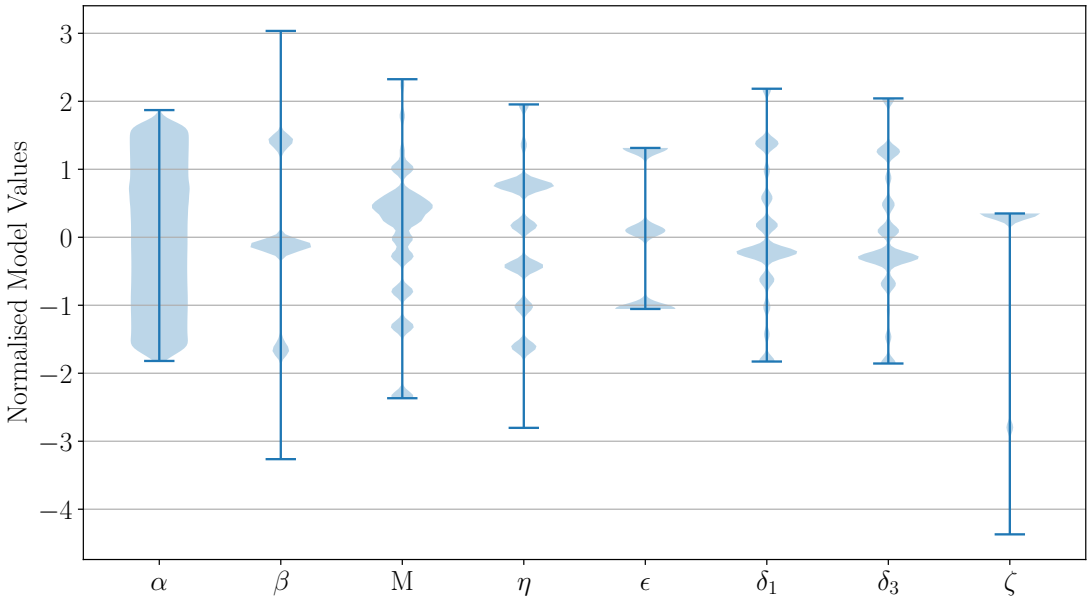


Figure 2.7: Violin plot of the wind tunnel dataset 1 and 3, using an empty armament.

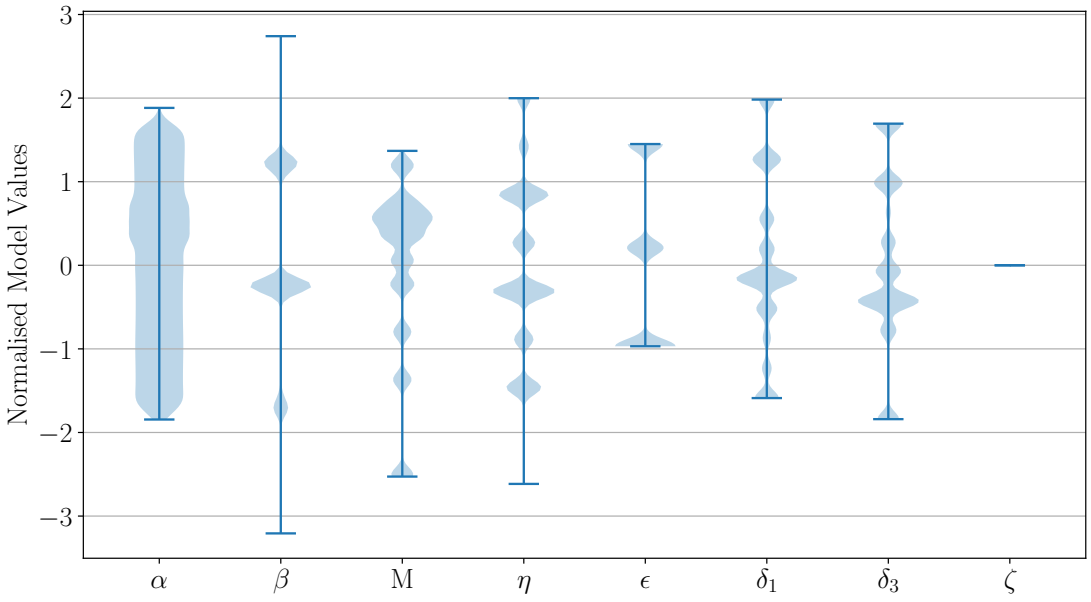


Figure 2.8: Violin plot of the wind tunnel dataset 2, using a full armament.

Table 2.2: Comparison of the number of polars and data points for each of the three datasets used during the thesis.

	Dataset 1	Dataset 2	Dataset 3
Total Polars	1533	899	520
Training Polars	1241	728	421
Testing Polars	154	90	52
Validation Polars	138	81	47
Total Points	435 140	260 250	141 823
Training Points	352 905	210 687	115 310
Testing Points	43 730	26 148	13 692
Validation Points	38 505	23 415	12 821

with each other. This is important as one does not want to create a machine learning model with highly correlated input features. In Figure 2.9, the correlation map is shown for dataset 1, which is the Pearson correlation coefficient calculated for the dataset [62]. Firstly, one can see that none of the features are strongly correlated with each other, meaning they all have a correlation value below 0.7. Nevertheless, one can see three moderately correlated values, the first is between η and ϵ, meaning that when the foreplane is deflected in the dataset, the leading edge slats are also deflected. This correlation can be largely attributed towards the testing matrix, where the foreplane and front slats are deflected for increasing angle of attack. This is because these deflections become critical for the combat aircraft's behaviour at high angles of attack. One can think of a case where the aircraft is in a steady state high angle of attack. A critical requirement is that in any flight condition, the aircraft needs to be able to lower the angle of attack and make sure to not increase the angle of attack further. At high angles of attack, the front canard generates a lot of lift, and since it is located at a greater distance from the centre of gravity it generates a large nose-up pitching moment. At high angles of attack one does not want to increase the angle of attack further, but rather be able to quickly reduce it. In this condition, the front canard angle is changed to the condition of less lift, which is denoted as an increase in the canard angle. This means that there is a nose-down pitching moment. The slat is then also moved to an angle of lower lift of the main wing, and the centre of lift for the main wing shifts rearward, also creating a nose-down pitching moment. This means the aircraft can manoeuvre very rapidly. This is the reason why there is a correlation of 0.53 in the dataset between canard η, and front slat ϵ angle.

The other strong correlation is the one between the left and right flap deflection (δ_1, δ_3), which is very logical since in most cases the flaps are deflecting simultaneously, with only in a minority of the cases the flaps are deflected separately. Finally, there are some weaker but recognisable negative correlations between the foreplane deflection η and the left and right flap deflections δ_1, δ_3. This means that when the foreplane is deflecting less (creating less lift), the flaps are deflecting more downwards to increase the lift of the

main wing. This ties again towards the high angle of attack case when the nose needs to be brought down. At high angles of attack, the canard is deflected towards a state of lower lift. This is correlated with a more negative flap angle (wing generating more lift), which also creates a nose-down pitching moment.

In summary, the correlation that can be seen from the input data related to the foreplane deflection η is related to the testing plan of the aerodynamic dataset. When the angle of attack is increased, the foreplane deflection cannot be too negative (positive lift) as this would tend to increase the angle of attack even further. For stability and mostly controllability, methods for creating a nose-down pitching moment are searched. These are increasing the foreplane angle (reducing lift), increasing the front slat angle (reducing lift), and increasing the negative flap angle (more lift). However, none of these correlations form a problem for the machine learning models, and each input feature is valid to be used in the model.

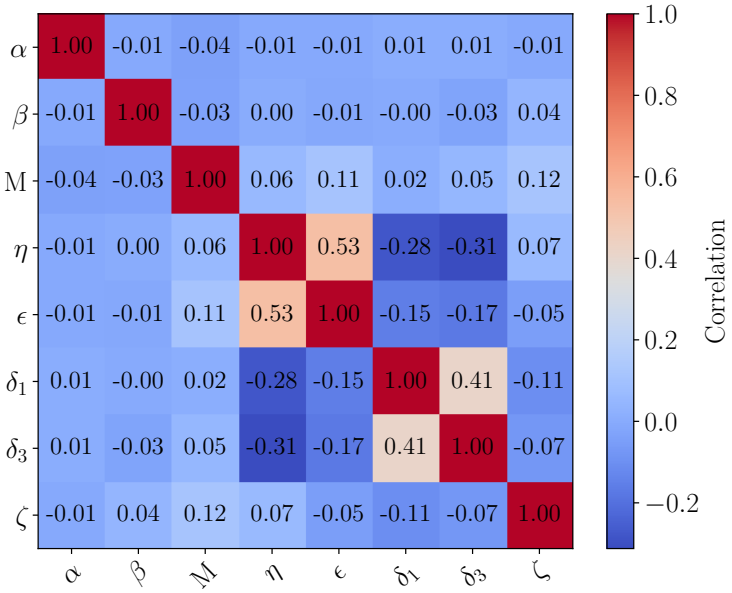


Figure 2.9: Correlation analysis of dataset 1 with empty store configuration and 8 input features.

2.2.3 Evaluation Metrics

In this subsection, a closer look is taken towards the evaluation metrics that can be used to evaluate the performance of the machine learning models [45].

Since the models which will be treated in this thesis are regression models. Some of the most common metrics are the mean squared error (MSE), and root mean squared error (RMSE), with the equation for the MSE shown in Equation 2.8. The advantage of the RMSE is that the error is on the same scale as the output variable. In Equation 2.8, the average is taken between the difference of the predicted value y_i and the actual value $\hat{y}_i$.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.8}$$

Another popular metric for gaining a measure of the error of a model is the mean absolute error (MAE), which penalises larger deviations from the true value less, as the values are not squared as can be seen in Equation 2.9.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.9}$$

There are also some metrics which are less sensitive towards the scale of the variable being evaluated. One of them is the mean absolute percentage error (MAPE). The equation for the MAPE can be seen in Equation 2.10 which provides the measure for the deviation. One can see that the MAE and MAPE are similar in that they are both not as sensitive to outliers since they are not squared. The MAPE has the benefit of being a percentage and comparable between different aerodynamic coefficients which are operating on different scales of magnitude.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|y_i - \hat{y}_i|}{|y_i|}\right) \cdot 100 \tag{2.10}$$

The final metric which could be used is specific to the application of the thesis. Since there are tolerances which evaluate whether the model itself is within an acceptable band of the true data, these could be leveraged to create a metric. The metric that is devised is the percentage of the predictions that are within the tolerance of the test data. The metric is calculated by looping through the testing dataset, after which the model provides a prediction for each of the test polar points, and it is evaluated whether they fall within the tolerance of the test data. The percentage within tolerance (PWT) is presented in Equation 2.11.

$$\text{Percentage within Tolerance (PWT)} = \frac{\text{Points with } |y_i - \hat{y}_i| < \text{Tolerance}}{\text{Total Nr. Points}} \tag{2.11}$$

2.2.4 Computational Resources

The machine learning models which are trained during this thesis are trained on a Linux workstation Graphics Processing Unit (GPU), of which the specifications cannot be provided due to confidentiality. The GPU is chosen for the parallel processing capabilities, as it is optimised for the matrix operations which arise when training machine learning models. The Python libraries which are used to develop these machine learning models, such as PyTorch, are capable of accessing the GPU. In this manner, developers can optimise the code to utilise the GPU's full capabilities. Airbus Defence and Space have the facilities for high computational resources, and use many of these workstations, allowing for parallelization of the training of the different models. At the department over 10 workstations were available to run machine learning models.

2.3 Aerodynamics of Combat Aircraft with Canard-Delta Wing Configuration

To better understand why the aerodynamic data coming from the Eurofighter is so difficult to model using traditional methods, a deeper look into the aerodynamics of combat aircraft is taken.

The Eurofighter is a highly manoeuvrable, canard delta wing aircraft. These aircraft are usually unstable at subsonic speeds and stabilise at transonic and supersonic speeds. This comes with the benefit of being highly manoeuvrable at subsonic speeds, as the maximum lift that can be generated using the canard configuration is higher. The primary contributor to the strong aerodynamic performance of this aircraft configuration is vortex lift. It allows the Eurofighter to be highly manoeuvrable at the cost of strong nonlinear aerodynamic behaviour, with the interaction of the vortices playing a large role in this.

In [49], vortex flow devices such as strakes are explored to enhance the manoeuvrability of an aircraft with a highly similar configuration to the Eurofighter. At high angles of attack, many vortical structures exist on the aircraft. These originate not only from the highly swept wing design, but also the canards, nose/cockpit, and engine intake as can be seen in Figure 2.11. The paper highlights the difficulties in controlling these vortices at different operating conditions, such as in sideslip. The vortices coming off of the canards can interfere with the vortical structures generated from the swept wing, and create highly nonlinear behaviour of the aircraft during operation. Furthermore, different operating conditions, such as angle of attack, sideslip angle, but also canard deflection, or leading edge flap deflection can completely alter the location and strength of the vortical structures which dominate the lift and thus behaviour of the aircraft. An

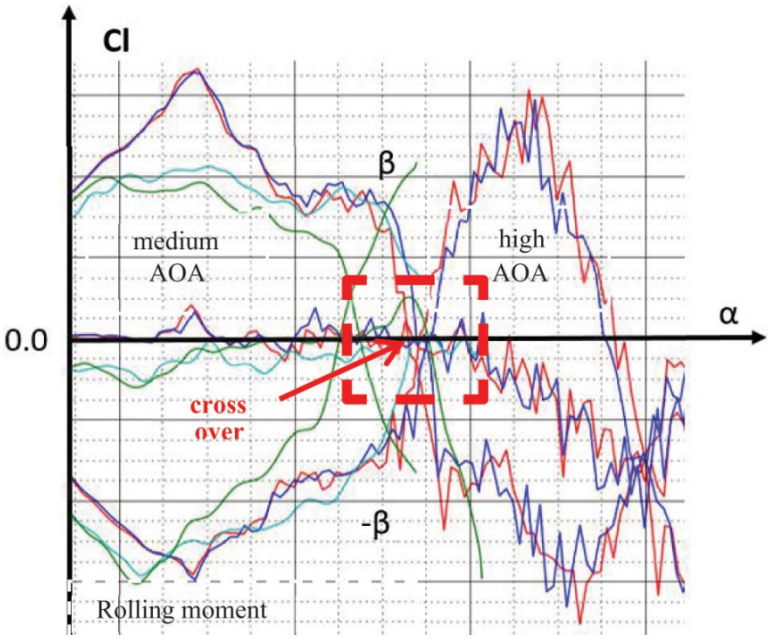


Figure 2.10: Rolling moment coefficient of the Eurofighter Typhoon series production aircraft as a function of angle of attack, for positive and negative sideslip angles, indicating the highly nonlinear nature of the canard-delta wing configuration. (from [49])

example of this is given in Figure 2.10, where the rolling moment coefficient C_l is given as a function of the angle of attack for the positive and negative angles of the sideslip. One can see that the behaviour is for example not linearly shifted but inverted. Furthermore, at zero angle of sideslip, the rolling moment coefficient is not just the average of the two but also decreases from a given angle of attack.

Another example of this is provided in [106], where the authors took a model resembling the Sukhoi-30, a Soviet-Union-designed canard-delta wing aircraft, and tested the effect of the canard deflection angle on the stall angle of attack. The paper highlights the dominant effect of the rolled-up vortex effect the canard has on the main wing. For example, at high angles of attack, the vortex can reattach the flow on the main wing. Furthermore, the interaction of different vortices has a strong effect on the lift and moment coefficient of the aircraft, and these effects are nonlinear. Therefore, an effect which reduces the lift at around 40 deg AOA, can increase it again at higher angles of attack.

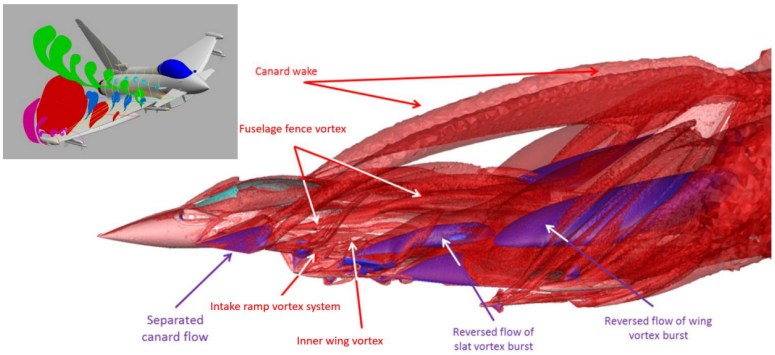


Figure 2.11: Delta-Canard vortex flow structure visualised using the lambda2 criteria at subsonic speeds and high angle of attack. (from [49])

Chapter 3

Literature Study

In this chapter, a literature study is presented on the topic of aerodynamic dataset modelling using machine learning approaches that would be capable of achieving the goals that are outlined in chapter 2. The literature study is subdivided into five parts. First, a deeper look is taken towards general data science methods to predict tabular datasets in section 3.1. Afterwards, three sections are made that go deeper into the workings and literature of Tree-based models, Artificial neural networks, and finally Bayesian neural networks in section 3.2, section 3.3, and section 3.4 respectively. Afterwards, a shorter section is provided on different ensemble methods in section 3.5. Finally, a conclusion and the research questions of the thesis are outlined in section 3.6.

3.1 Prediction of a Tabular dataset

There a many methods that come to mind when thinking about predicting a tabular dataset in the form which has been shown in section 2.2. In this section, the most common methods for predictive modelling are provided, which are not either tree-based models which will be covered in section 3.2, artificial neural networks which are covered in section 3.3, and finally Bayesian neural networks which are covered in section 3.4.

3.1.1 Linear Regression

To be able to perform linear regression on a dataset one needs to make certain assumptions about the dataset that is being used. Firstly, the homogeneity of the variance, requires that the error present in the data does not change significantly over the values of the input features (homoscedasticity). Secondly, the input features are independent, the data is normal, and finally, the data should be best predicted by a straight line or plane in multiple dimensions. Although the observations and input features are independent as has been shown in section 2.2, the aerodynamic characteristics of the Eurofighter are highly nonlinear from section 2.3. Therefore, the assumption about the linearity of the data cannot be made. The other issues are that the data is not normally distributed, and the data coming from the wind tunnel is not homoscedastic.

3.1.2 Polynomial Regression

Other models such as least squares, Ridge regression, Lasso regression, ElasticNet or any other regression model where the function that is being regressed needs to be specified up front are unfeasible for this thesis. This is because the shape of the function is not known beforehand, and a polynomial for example would not be able to capture the nonlinearity in so many dimensions. The strength of machine learning models is that they are capable of modelling nonlinear relationships with little prior knowledge required of the input data, in contrast to polynomial regression, where the constants which define the polynomial need to be defined before they can be regressed.

3.1.3 Radial Basis Functions

Radial basis functions (RBFs) are functions whose values depend on the distance from a given point, which is usually some centre point. Many different radial basis functions exist, such as Gaussian, multiquadric, inverse quadratic, inverse multiquadratic, and others, which all provide a measure for the distance to a given reference point [23]. Radial basis functions are not only used to perform regression themselves but are often used as a kernel in support vector regression, or Gaussian process regression, which will be covered later. Usually, a function approximation is made by making a weighted sum of different RBFs as can be seen in Equation 3.1. This is also considered to be similar to a single-layer neural network which uses an RBF as its activation function. Each neuron or RBF is then assigned a central point or vector for multiple dimensions from which the input vectors or training data can be compared.

$$y(\mathbf{x}) = \sum_{i=1}^{N} w_i \varphi \left(\|\mathbf{x} - \mathbf{x}_i\| \right) \tag{3.1}$$

Many industry examples exist of using radial basis functions to perform regression on nonlinear datasets. In [99], RBF networks have been applied to the aerodynamic modelling of aircraft, where it was used to calculate the stability derivatives. The RBF neural network was used in this application, as the rate of convergence is higher than a classical neural network. However, the radial basis function does not have the possibility of going deep with many layers as a neural network. Therefore, it is to be seen whether it can properly model the highly nonlinear behaviour of a combat aircraft, as the paper tested the network on the ATTAS aircraft from DLR, which resembles a short-range commercial aircraft. Another application found for RBF was to predict the container handling capacity of the Shanghai harbour [35]. The paper, unfortunately, provides little information on the type of input data. However, it is stated that the multiple inputs have strong nonlinear effects on the containing handling capacity output of the network. It is very hard to determine how complex or nonlinear the output of the model is from the provided information. Therefore, it might be that RBFs are capable of outperforming multi-layer perceptrons (MLPs) for smaller and simpler problems. However, when the architecture starts getting significantly large, MLPs with their ability to form deep neural networks will begin to outperform RBFs, although this cannot be determined from the literature found. In [119], RBFs are compared to classical MLPs employing backpropagation. It was concluded in this paper that RBF networks are better for data with regular peaks and valleys, as they are more efficient than traditional networks in this case. However, for functions without regular peaks and valleys, multi-layer perceptrons are preferred. Furthermore, RBF functions are more tolerant to noisy input data compared to MLPs. In the application of the thesis, the data although possibly a bit noisy, does not have regular peaks and valleys.

3.1.4 Gaussian Process Regression

Kriging otherwise referred to as Gaussian process regression is a very popular method for surrogate modelling and regression. It was invented for predicting the most likely location to find gold, based on nearby drilling samples [64], and has since undergone some developments that make it a widely used model in statistics and surrogate modelling. The benefit of Gaussian process (GP) regression compared to the methods that were investigated before is that it is non-parametric and therefore no prior shape of the function should be known. Furthermore, GPs are not only useful for their predictive properties but also for their probabilistic approach. Each output value of the model is also accompanied by the uncertainty of this prediction. A GP regression is defined by

its mean function, and the covariance function or kernel. The kernel is used to represent how correlated the outputs are for two inputs. Common kernels are the RBF, Matérn, polynomial and periodic kernels. These form our prior beliefs about the dataset and different functions are sampled from these kernels [28]. The challenges of GP regression come when using an increasing dataset size. The overall computation complexity scales with order N^3 with N the number of data points [112]. This is because the covariance matrix needs to be inverted during training. When one considers the dataset of the thesis with over 100,000 data points, the number of data points and complexity could become an issue for GPs. Therefore, Bayesian neural networks are considered in section 3.4, as they are said to be more scalable towards a larger number of data points, while also providing the uncertainty quantification as with the Gaussian process. There have been techniques suggested to reduce the computational complexity of GP models which was done in [74], where sparse kernels were used. However, the further investigation of scalable Gaussian processes will be discussed in section 3.4.

3.1.5 Support Vector Regression

Support Vector regression (SVR) tries to find the curve that minimizes the distance between the data and this curve up to a constant ϵ called the margin. Support vector machines also scale poorly with an increasing size of the dataset. Meaning that with the increasing size of the dataset, they become more difficult to train and computationally expensive. They do however form a big part of machine learning models being used in the industry for regression and classification tasks. Many examples exist such as [92] for predicting solar radiation, or in [55] for predicting the useful remaining life in machines, or [90] in the field of hydrology and hybrid models. [90] Identifies the key drawbacks of SVR being the difficult selection of appropriate linearisation function and kernel function, and their low interpretability. Furthermore, they are also poor in extrapolating, and the predictive performance can sometimes be an issue. Some key benefits being provided are their good performance for classification and robustness to noise and outliers. In [113], where SVR is used in combination with a genetic algorithm to optimise the hyperparameters of the model, to forecast the annual load forecasting of the electric grid. The authors provide evidence of the SVR being very capable for small sample problems and having good performance due to the genetic algorithm used.

3.2 Tree-based Models

This section explores the most common types of tree-based models which are used for regression, namely decision trees, Random Forest, and gradient-boosting models, and their applications.

Tree-based models, such as Random Forest models, XGBoost, or LightGBM regression models, are some of the most popular and widely used models to perform regression on tabular datasets. A paper which shows the strengths of tree-based models on tabular data is given by [46], which shows that the superiority of deep learning methods such as neural networks is not clear when using tabular data. Therefore, from this paper one can determine that tree-based models are competing with ANNs for the superiority of the regression models, which makes them highly interesting to investigate.

This section is split up into four parts. First, decision trees are discussed in subsection 3.2.1. After this, Random Forest models are discussed in subsection 3.2.2, and gradient-boosting models in subsection 3.2.3. Finally, applications of tree-based models are provided in subsection 3.2.4.

3.2.1 Decision Trees

Tree-based models, as the name suggests, are built up of decision trees. An example of a decision tree is shown in Figure 3.1, where the moment coefficient C_m is predicted with a decision tree, based on the angle of attack, sideslip angle and Mach number. A decision tree is built up of internal nodes, which are coloured blue in Figure 3.1, which split the data into two branches. There are also terminal nodes or leaves, which are then the values for C_m. A decision tree is drawn with the root at the top and the leaves at the bottom. In a real dataset and real decision tree model, of course, many more internal nodes and end nodes will be present, but to explain a tree-based model a simple decision tree is chosen.

To make a decision tree, the model needs to know when and how to split the data. This is done by using metrics such as the mean square error (MSE) or the variance reduction at each split. The algorithm computes the variance reduction or reduction in MSE for each possible split of the data, and the one with the largest reduction in the variance or MSE after performing the binary split is chosen [20]. More in detail this step is performed by ordering the dataset from smallest to largest values for a given input feature. Afterwards, two values are selected for which the average value is calculated. This represents the split in data, which we shall denote as A for now $A = \frac{x_1+x_2}{2}$. The outputs of the model for values lower than A, denoted now as B are then calculated as the average output value for the data points which fall within this split $B = \frac{y_{b,1}+y_{b,2}+...+y_{b,n}}{n}$. The output values

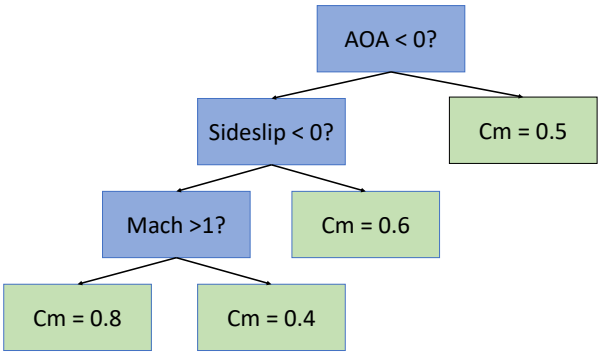


Figure 3.1: Example decision tree using the angle of attack (AOA), sideslip angle, and Mach number.

for input values higher than A are the average output values for the inputs larger than A, denoted as C, $C = \frac{y_{c,1}+y_{c,2}+...+y_{c,n}}{n}$. After this, the sum of the squared residuals of each of the members of the split is calculated w.r.t. the values B and C for input values lower and higher than A respectively. This is then repeated for all possible combinations of x, and the one with the lowest sum of the squared residuals is chosen. When multiple input features are present, this procedure is repeated over all the input features, the one with the best sum of squared residuals, is taken as the root. Then this is repeated for each next node.

The splitting of the branches of the tree is stopped based on a few possible scenarios. It could be that the maximum depth of the model is prescribed, meaning that the maximum number of splits from root to leaf is predefined. Another method to stop splitting could be that there needs to be a minimum number of data points left in a particular split to make a split of the data. Finally, a minimum reduction in MSE can be used to make a split.

The advantage of decision trees as a machine learning model is their easy interpretability and easy implementation. Furthermore, the dataset does not need to be normalised for using it in a decision tree. Decision trees also have some disadvantages, which are their tendency to overfit the data. A right value for tree-depth needs to be found, that allows a model to be sufficiently complex but not so much that it does not generalise. Decision trees also tend to be very unstable in the way the splits are created, they exhibit high variance. With a small change in the input data, the root split might be different, which is then propagated downwards throughout the whole tree. A method for reducing this is called bagging, which will be explained later. Another limitation of decision trees is their smoothness, as the number of splits that are made is finite, meaning the output is step-wise jumping from the value of one split to another.

Due to their limitations, their application is very limited in the industry. However, they

form the basis for all the other tree-based models which will be explained in the next sections. Therefore, due to their importance, more time was spent explaining decision trees to make the understanding of the following subsections easier. Now that the basis of decision trees is provided, together with its limitations, more advanced types of decision trees can be considered that aim at resolving the above weaknesses, and provide models competitive with other machine learning models in terms of performance.

3.2.2 Random Forest

A Random Forest model differentiates itself from a decision tree in a few key ways.

- Create N bootstrapped datasets.
- Only use a random number of input features for each step of the tree.
- Do this for N decision trees and average their prediction.

A bootstrapped dataset is created by sampling data points from a dataset, with replacement. Meaning that a single data point can be present in the bootstrapped dataset multiple times. The probability of having a data point in a particular bootstrap dataset is around 63%. This means that around 37% of the data points in the original data set are not present in the bootstrap dataset [34]. This is why many different bootstrapped datasets are used.

When creating each decision tree of the Random Forest, not all input features are used, but only a random set of features at each step of the decision tree [50]. Therefore, for example, at the first split of the decision tree, the first, and last features are used. Afterwards, the second, third, fourth, and so on. This makes sure that the different trees are uncorrelated since they are randomly generated. This also helps with overfitting, and generalisability of the Random Forest as all the decision trees are learnt differently.

Finally, many decision trees using the above two tactics are used to average over many decision trees. The procedure of using bootstrapping and then aggregating many different decision trees to eventually have one remaining model is called bagging [19]. The name originates from **b**ootstrap **agg**regat**ing**.

This technique is implemented for many reasons. Firstly it is proven to reduce overfitting, as the algorithm is less likely to fit noisy or random variations in the data by using random subsets of the data at each step. Since the method uses an ensemble of many different decision trees, all trees must be different and not correlated to each other. Otherwise, the method would not have many advantages over a normal decision tree. A Random Forest

model is also much better at generalising to unseen data, as the randomness associated with training the Random Forest allows it to better capture the underlying pattern in the data [21].

3.2.3 Gradient Boosting Models

A gradient boosting model builds sequential decision trees, where the next tree corrects the mistakes of the previous one. AdaBoost is the simplest gradient boosting machine one can think of, as the tree size is only a single split (called stump). Other models include lightGBM, CatBoost, and XGBoost, which will be further discussed in this subsection.

Adaptive Boosting (AdaBoost)

The three concepts that differentiate adaptive boosting from a Random Forest model are.

- Combines weak learners to make a prediction (stumps).
- Some of the trees are weighted higher than others for the prediction of the output.
- Each weak learner is made by taking the mistakes of the previous weak learners into account.

Firstly, the model is initialised with a constant value. This constant value is the average of the outputs. Secondly, one starts by making the first tree. This is done in a few steps, firstly one calculates the negative of the derivative of the loss function and uses this to calculate the residual of each of the output samples. This can be conceptualised since the loss function is commonly taken as $L = \frac{1}{2}(\text{Observed} - \text{Predicted})^2$, and therefore the derivative is just the residual. However, when a different loss function is taken, a so-called pseudo-residual is calculated as the derivative might not be exactly equal to the equation for the residual. The next step is to make a regression decision tree, to predict the pseudo-residuals for each of the output values. Usually, the maximum depth or maximum number of terminal leaves is specified as a hyperparameter of the model. For each leaf in this tree, an output value γ is computed which is the solution to a minimization problem. This minimization problem solves the sum of the loss function between the output value and the sum of the predicted value of the previous tree and γ. For the loss function $L = \frac{1}{2}(\text{Observed} - \text{Predicted})^2$, γ is simply the average of the predicted residuals of the created decision tree. Finally, the new prediction can be made by multiplying the different γ of each of the leaves of the generated decision tree with a

learning rate and adding it towards the output value of the tree made in the previous step, and this will generate the newly updated prediction of the model.

If the process of generating a tree based on the residuals, calculating the γ and then updating the predictions using the previous predictions is repeated many times, a gradient boosting model is created [33, 38].

There are three highly popular gradient boosting models in the industry, which do not use stumps but allow for more splits in the decision trees that build them. These are called XGBoost [26], lightGBM [58], and CatBoost [87]. All are variations of the classical gradient boosting method.

There are some key differences between each of the models in terms of the decision trees they create. Firstly, in the way they create the decision trees. CatBoost, creates only symmetrical trees, which makes for faster computation, and evaluation. This means that at each level of the tree, only 1 feature is evaluated. The effect of this is that the model is less sensitive to hyperparameter changes, meaning the model provides a good model without training the hyperparameters. LightGBM and XGBoost use asymmetric trees. Therefore, they do not enjoy this benefit for hyperparameter tuning. Furthermore, XGBoost makes level-wise growth, while lightGBM uses leaf-wise growth, meaning it selectively chooses to further develop one branch of the tree, which makes lightGBM have smaller overall models, which helps with computational speed.

CatBoost has a particularly special way of dealing with categorical data as the name suggests, for the dataset of the thesis this is less important as there are no categorical features, only numerical values. CatBoost also uses something called ordered boosting, which was invented to combat the overfitting of gradient boosting methods. The idea is that the value in a leaf of the decision tree provides an estimate of the gradient of the points that are in this leaf. However, this value is biased as the approximation of the gradient is based on the same data points as the decision tree is built upon, meaning it will be overly confident. Therefore, to combat this, CatBoost uses random permutations of the dataset in each step of making a decision tree, to make unbiased predictions.

The hyperparameters that are tunable with these models are mostly similar, with the two most important hyperparameters that need to be tuned being the number of trees, and the maximum depth of each of the trees. In the literature, it is generally known that each of these models has the capability of performing the best on a specific dataset, and therefore each model should be tested on a specific dataset to determine the best-performing one. Furthermore, it is also known that lightGBM and CatBoost are generally faster than XGBoost. Therefore, it is essential to determine whether the increase in training time is worth the possible performance gain with XGBoost in the thesis.

The release date of each of the models is 2014 for XGBoost, 2016 for lightGBM, and

CatBoost in 2017. In each of the publications from these models, the next model claims to outperform the previous model. This means that lightGBM outperforms XGBoost, while CatBoost in turn outperforms both of these models, on the testing datasets used for XGBoost. However, whether this is automatically true for the aerodynamic dataset used in this thesis is to be seen and will be evaluated during the thesis.

3.2.4 Applications

In [46], the authors provided performance measures that provide evidence of the superiority of tree-based models over neural networks for medium-sized data, which is for around 10,000 data points. The paper still lacks evidence on large and very large datasets which are more relevant in this case, as the number of data points is more in the order of 100,000.

In [81], tree-based models are compared to other machine learning models such as Gaussian Process Regression, Support Vector Regression, logistic regression, AdaBoost, and K-NN, for predicting liver disease. The paper finds that Random Forest models perform best. However, each specific application could require a different machine learning algorithm that could be optimal for the problem at hand. In [16], the focus lies on comparing different models on tabular data as well. This paper is inline with [81] which states that it is difficult to beat tree-based models for heterogeneous data, which includes labels, numbers and classes together. Homogeneous data uses only one type of data, as the datasets which are used during the thesis. The paper does make a good case about the papers which compare different models on the fact that they all use a wide variety of different datasets, and there is thus no unified benchmark. All the papers mention that in some cases, classical deep learning approaches beat tree-based models. However, without unified benchmarking, it is difficult to determine whether deep learning approaches always have this capability. It could be that in some papers the neural networks are not properly designed and trained, and therefore tree-based models, which could be more robust towards a given data set, perform better in these cases. Furthermore, [16] state that transformers and hybrid models form the state of the art for tabular data with regression, and that regularization, interpretability and computational efficiency of hybrid models are still the lowest hanging fruit in performing deep learning on tabular data sets.

3.3 Artificial Neural Networks

This section provides all the relevant literature on applying artificial neural networks (ANNs) towards regression problems. Firstly, a background is given about neural networks in subsection 3.3.1. Following this, the literature which is available about aircraft employing ANNs is provided in subsection 3.3.2. Afterwards, some general methods and literature are found on surrogate modelling using ANNs in subsection 3.3.3. After this, state-of-the-art research is presented in subsection 3.3.4. After which transfer learning, and genetic algorithms are investigated in subsection 3.3.5, and subsection 3.3.6 respectively. Finally, a conclusion is made on the literature study for artificial neural networks in subsection 3.3.7.

3.3.1 Neural Networks Background

Neural networks draw inspiration from the biological understanding of the brain. A neural network employs a model of a neuron in the sense that many signals come into the neuron, either as an input value or from other neurons. The neuron then performs a mathematical operation and produces an output that is then transmitted to other neurons. The mathematical operation that is performed uses weights to determine the importance of the inputs coming to the neurons. These weights ultimately define the neural network, together with the architecture or layout of the different layers and the number of neurons in the network. A schematic of a neural network is provided in Figure 3.2, where a feed-forward neural network with a single hidden layer is presented, also known as a multi-layer perceptron. The input layer contains 3 input neurons, which are connected to all 5 hidden neurons, where a so-called bias is added to each neuron. Finally, each hidden layer neuron is connected to each of the output layer neurons. An example computation of how the output of the hidden layer neuron h_1 is calculated is given in Equation 3.2. Here, the corresponding weight is multiplied by the corresponding input and summed over all input neurons of the previous layer. Finally, a bias b_1 is added to the neuron to shift the output before passing it through an activation function, Φ.

$$h_1 = \Phi\left(\sum w_{i,1} \cdot x_i + b_1\right) \tag{3.2}$$

It has been shown by [54] that a multilayer perceptron (MLP) with enough neurons is capable of approximating any continuous function. To train ANNs, one requires forward propagation and backpropagation, the former to provide the prediction of the network with the current weights, and the backpropagation to be able to adjust the weights based on the gradients of the loss function w.r.t. each of the weights.

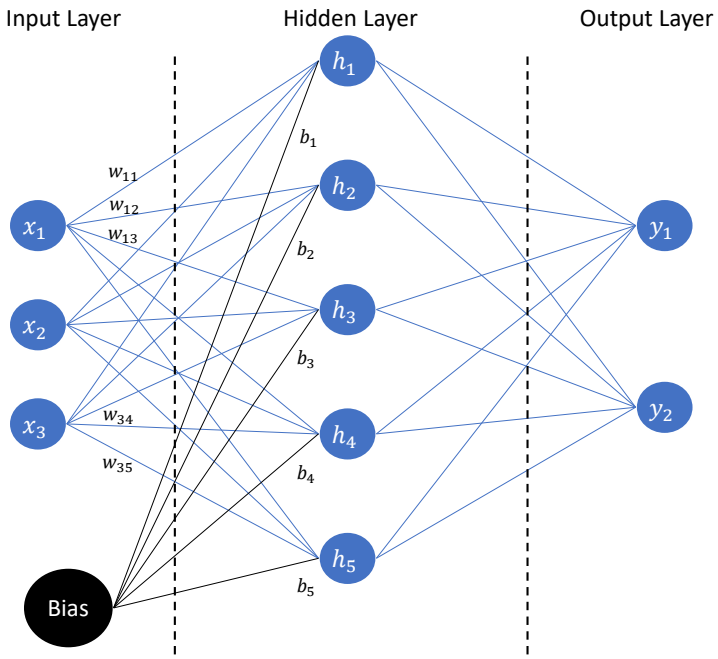


Figure 3.2: Schematic of a feedforward artificial neural network with one hidden layer, 5 hidden neurons, with 3 inputs and 2 outputs.

Therefore, an artificial neural network can be described by the number of inputs, the number of hidden layers, the number of neurons in each of the hidden layers, the number of outputs, the type of activation function used for each neuron, and the optimiser function that is used. Furthermore, other principles can also be applied such as regularization or dropout, which are more general terms that can be applied to any neural network. One of the most important parameters is the number of hidden layers and how many neurons each layer contains also called the depth and width of the network respectively. As has been shown by [72], the number of neurons needed to approximate many piece-wise smooth functions is exponentially larger for a single hidden layer network than for a deep network. Therefore, the depth of a network is a highly important parameter for function approximation, with a deeper network with fewer neurons in each layer being more favourable than choosing a more shallow network with more neurons in each layer.

3.3.2 Applications on Aircraft and Aerodynamics

Neural networks have been employed widely in combat aircraft design, mostly in the form of neural controllers, where neural networks are used and integrated into the flight control system due to their nonlinear modelling capabilities. Most neural networks used for controllers focus more on on-line learning and want to approximate the control law that governs the system. Many examples exist: [104, 41, 105, 42, 18], even with an autoencoder in front of the ANN [68], or on a commercial aircraft model [4]. Therefore, it is safe to say that in the field of combat aircraft, and even other aircraft, neural controllers are by far the most popular. However, these applications are not as relevant for this literature research, as they solve a different problem. Therefore, in this subsection, the papers found that closely resemble the problem of the thesis are presented.

In the paper on reducing wind tunnel data requirements using neural networks [97], the goal was to minimise the amount of wind tunnel data required to completely define an aerodynamic model to save costs. The paper uses the subsonic high Alpha research concept (SHARC) which used the NASA Ames Research Centre's wind tunnel. Although having a similar objective as the thesis, the 20 different flap configurations of the SHARC aircraft are low compared to the thousands for the thesis, and therefore the architecture that can be used in [97] can be much simpler, with only 15 hidden neurons in a single hidden layer MLP. Another example of this is in [83] where the lift and drag of an aircraft is predicted. With only 36 combinations of leading and trailing edge flap deflections, the model could remain simple and still provide good predictions.

In [96], a delta wing of 70 deg sweep is periodically moved between 0 and 90 degrees, after which the force and moment coefficients are measured. The neural networks form a better approach for modelling aerodynamics during a high-load, high-angle-of-attack Cobra manoeuvre due to their ability to model highly nonlinear behaviour and changing flight conditions. For this application, a feed-forward neural network with 2 hidden layers is used. The paper illustrates once more that a relatively simple neural network is capable of modelling the nonlinear aerodynamics of a combat aircraft. The architecture of the neural network could be further optimised in the current age with different activation functions, and optimisers such as Adaptive Moment Estimation (Adam).

For commercial aviation, there are many examples of networks being used for different problems in the field. These include fuel consumption [7, 108], aerodynamic coefficient prediction [111, 73], fault detection in flight test data recorders [77], and surrogate modelling of the aircraft [100, 44]. In these papers, multi-layer perceptrons are used for function evaluation. The mostly older papers employ the Levenberg-Marquardt algorithm as the optimisation algorithm for faster convergence compared to classical back-propagation using stochastic-gradient descent. The problems at hand are usually quite simple, requiring not more than 2 hidden layers in the neural network, with less than 10 neurons in each layer. These simple architectures show that the aerodynamic behaviour

of commercial aircraft is much simpler than the ones of combat aircraft. Commercial aircraft do not fly at high angles of attack and are designed for safety and efficiency, instead of pure aerodynamic performance for manoeuvring. This is reflected in the areas where neural networks are applied in commercial aviation.

In [98], a comparison is made between Gaussian process regression, reduced-order modelling using Isomap manifold learning [37], proper orthogonal decomposition and neural networks in their ability to reconstruct the surface pressure distribution on the NLR7301 airfoil and the NASA Common Research Model (CRM). The airfoil and CRM were both tested at subsonic and transonic regimes, introducing strong non-linearities into the data in terms of shock waves. For the airfoil, the optimum architecture was found to have 11 layers, with 518 neurons in the first layer, with a shrinkage factor of 0.89, meaning that by progressing one hidden layer deeper, the amount of neurons is only 89% of the previous layer. The model for the CRM test case had 7 hidden layers, and 221 initial neurons, with a shrinkage factor of 0.95. The authors conclude that neural networks perform far better than any of the other techniques that were evaluated, at the high cost of finding the right hyperparameters of the model, which was identified as a key step in having the improved performance. Furthermore, the neural networks were able to extrapolate beyond the training data, and outside the operating window well. This paper not only shows the strong capabilities of a feedforward neural network in predicting surface pressure distributions on a wing and airfoil, which is usually something that is predicted using convolutional neural networks but also that many hidden layers are needed when dealing with highly nonlinear data such as the one seen in transonic aerodynamics, with shock waves and possible boundary layer separation.

In [101], the flow field of airfoils is predicted using deep learning. First, convolutional neural networks (CNNs) are used to learn a lower-order representation of the geometry of the airfoils. Following that, a multi-layer perceptron is constructed which learns from over 60 million data points coming from over 110 different airfoils which come from CFD simulations. The model has as input, the 16 parameters which define the airfoil shape coming from the CNNS, as well as the Mach number, and angle of attack, to predict the pressure, and velocity components in x and y directions. The ReLU activation function is used, as well as the ADAM optimiser, with 10 hidden layers, and 1000 neurons per hidden layer. The final model has an accuracy of over 97% for the test cases. The paper shows another example of a deep neural network to predict flow fields. The model has adequately learnt the highly nonlinear Reynolds Averaged Navier Stokes (RANS) equations, using a point-by-point method. This shows that for the level of non-linearity a deep neural network is necessary, but also a significant amount of training data.

3.3.3 Surrogate Modelling of Nonlinear Systems

Surrogate modelling is the field of creating a simpler model of a complex system to save cost, time or other resources. This simpler model can then be used to perform optimisations, sensitivity analysis or risk analysis, to reduce the cost of simulating or acquiring the data using the complex system. Surrogate modelling is closely related to the problem presented in the thesis as the data which is collected during the wind tunnel tests is acquired in discrete locations of the domain, and collecting wind tunnel data at increasingly smaller step sizes is extremely costly. Therefore, essentially a surrogate model is created using neural networks to sample the entire design space many times to provide, for example, the derivatives for different variables. Therefore, it is interesting to investigate some surrogate modelling techniques in the scientific literature.

Surrogate models are often used in the field of optimisation, and this is also the case when looking for literature on ANNs being used for surrogate modelling. For example, in [116], a surrogate model is made for the optimisation of macroscale elastic structures and in [86], surrogate models are made for the optimisation of manufacturing processes. In both cases, the surrogate models are used to optimise the given input parameters while using fewer total simulations of the original problem. The domain is sampled using a design-of-experiment scheme, such as Latin-hypercube sampling, after which the model can be trained. In [116], the performance of the model is increased by adding the derivative of the objective function during training using the Sobolev norm. In [86], a 7-layer neural network was used to optimise the 50 input parameters.

In [47], physics-informed neural networks (PINNs) [91] are used and applied to inversion and surrogate modelling of solid mechanics. Besides applying the surrogate model to a nonlinear elastoplasticity problem, the authors use transfer learning to significantly speed up the convergence of the problem. This could be a possible implementation to increase the convergence speed of a neural network being applied to a different dataset of the Eurofighter and increase the robustness and convergence speed. In [63], a surrogate model is made for mining process optimisation. The paper described neural networks as perfect candidates for surrogate modelling, as the models are flexible and provide fast predictions using parallel computing. The authors use an evolutionary algorithm to optimise the architecture of the network, which is adapted from one used by [53], and can add/remove hidden layers, change the number of neurons, the activation functions, and change the learning rate. This method for finding the optimal architecture could be much more efficient than for example grid search. The surrogate model could approximate the original function with an error of 0.45% while using only 1/1000 of the original data. Two other examples are given by [78], where a surrogate model is used to model the building energy simulation program, and in [71], where a neural network-based surrogate model is used to model reactive transport modelling, of fluid flowing through a porous medium.

In [114], a comparison was made between different types of neural networks (ANNs, RBFs, CNNs, GANs) in creating a surrogate model for convective heat transfer performance. It found positive results for all types of neural networks that were tested. However, the ANN was found most suitable for datasets with more than 1000 training samples, where there was no need to fully reconstruct the image of heat of the modelled U-channel. The paper concluded from its literature search into neural networks for surrogate modelling that often the models could only be trained with limited data, and this often led to overfitting. To mitigate this, surrogate models should respond to the input variables in a wide range, and be provided with sufficient training data.

One can conclude that surrogate modelling with neural networks is another closely related problem to the master thesis. The literature found is very recent and in a wide variety of fields. The literature provided some interesting ideas to be applied to the aerodynamic model that is created, but the research gaps are very specific to the given research field and cannot be transferred easily to aerodynamic modelling.

3.3.4 State of the Art for Artificial Neural Networks

In [1], which poses a state-of-the-art survey on ANNs, some key problems of current ANN studies are identified. These include creating more robust models, model explainability, improving extrapolating ability, and uncertainty quantification. Furthermore, the authors urge more research into genetic algorithms, hybrid neural networks, and input parameter selection, for better effectiveness and efficiency.

The use of hybrid neural networks is very common in current applications of ANNs, often combining an ANN with a different model such as an attention network, as has been done by [70], and [25]. The attention not only provides a more accurate result by making a hybrid model but also for the attention network to select the most important input parameters. Furthermore, other hybrid models can be made by using other models in combination with an ANN, such as convolutional neural networks or Long-short term memory neural network models as has been done in [118], and [69]. All of these combine multiple models, often the MLP being used to perform a more simple part of the prediction, to improve the accuracy of the overall model. The hybrid models offer many advantages in terms of accuracy, robustness towards the training data, generalizability, and extrapolating ability, and could be a key focus for the master thesis [117]. The most common methods include bagging, boosting, stacking and voting. Since bagging and boosting are common attributes of tree-based machine learning models, they are left to section 3.4, while stacking and voting are methods that learn the weights that should be attributed to each of the models in the hybrid models, and have been proven to increase the performance of the overall model [93]. These are discussed in section 3.5.

Model explainability also forms a key issue in the use of neural networks in aviation.

Due to their strict safety standards, often a more explainable model is preferred over the black box approach that neural networks offer. Model explainability will thus form a key point of focus for being able to employ and certify the neural networks as an aerodynamic model on a combat aircraft. Explainable A.I. is a hot topic, and two approaches can be found in the literature, transparency design and posthoc explainability. Examples of transparency design and posthoc can be found in [122] and [6] respectively. However, more research is proposed for regression neural networks.

3.3.5 Transfer Learning

Transfer learning is the concept of transferring knowledge from one model trained on a different but similar dataset towards another model. A common example is that a machine learning model that is trained to recognise oranges, can be a good starting point for a model that recognises lemons.

In [125], it is claimed that the main need for transfer learning is when the target training data is limited. Therefore, the learning of a larger, similar dataset can be used to fine-tune the model on this smaller dataset. In the review papers of transfer learning [125, 84], more attention is spent on the use of transfer learning for classification in the field of image recognition, and text recognition. Transfer learning is a well-documented field with more than 700 academic papers being written in the field [125].

In [8], water quality predictions are made by transfer learning between nearby water quality measuring points. Here, an Echo state network is used together with multi-source transfer learning, where the common features of the neighbouring water quality measurement locations are combined. This resulted in a model which had better predictions, and lower bias compared to traditional recurrent neural networks.
In [79], a neural network is trained to model the operating conditions of a simple Rijke tube that is heated. Transfer learning is used to transform this simple model towards a model of a lean-premixed combustor for which the data is scarce and expensive to obtain. The model contains 9 hidden layers with a maximum of 128 neurons in the fifth hidden layer and 4 neurons in the first and last hidden layers. To transfer learn to the original model, a single layer is added, and the last three layers of the original model are retrained, the first six layers are not retrained (frozen). A reported increase of 22% in the R^2 score is attributed to the use of transfer learning.
In [40], a method is shown to improve the performance compared to standard kernel Ridge regression. It is based on the re-weighting of the target dataset by minimising the objective function of the regression problem. In [17], the power of transfer learning is mostly attributed to the reduced training time of the retrained model.

Transfer learning appears to be underexplored for regression on large datasets in the literature, with only a few examples being found dealing with this topic. Therefore, the

thesis can provide a unique investigation into the application of transfer learning using machine learning models for regression on large datasets from combat aircraft.

3.3.6 Genetic Algorithms

This subsection delves deeper into the topic of genetic algorithms for hyperparameter optimisation. This is because the state-of-the-art literature for artificial neural networks pointed out that genetic algorithms for hyperparameter optimisation form a key field of interest to perform research into.

In [32], genetic algorithms are used for having a robust process of optimising the hyperparameters of a predictive model that monitors business processes. In this study, it was also highlighted that genetic algorithms can also be used for multi-objective functions where a Pareto front can be obtained which provides the best parameter combinations. Genetic algorithms reduced optimization times, and could find the right balance between different dimensions in the multi-objective function and were scalable towards larger datasets.
In [123], an evolutionary algorithm is used to optimise the hyperparameters of a CNN on the CIFAR-10 dataset. The paper shows the values it used for the number of individuals in the population (500), number of generations (35), and mutation rate (0.05), which could provide a benchmark when creating a custom genetic algorithm. The genetic algorithm provided improved performance over manual, grid, and random search. Furthermore, the paper shows that the fitness values cannot be found in the first 10 of 35 generations, after which some competitive models arise. Although the paper optimises the kernel sizes, it leaves optimising the number of layers, and number of neurons in each layer up to future work.

As was stated in the state-of-the-art literature study for ANNs in subsection 3.3.4, more research is required in the field, as genetic algorithms are promising for optimising the hyperparameters, and not much prior research is performed.

3.3.7 Conclusion

In conclusion to this section, one can state that neural networks have been applied to many cases of aircraft and aerodynamic modelling. Most neural networks use multi-layer perceptrons as function approximates, with 1 or 2 layers, as the function which needs to be approximated is relatively simple. Most of these publications date back to the 1990s and early 2000s. Many of them employed the Levenberg-Marquardt backpropagation algorithm to train the neural network, for the increased convergence speed, which was very important in the time when computers and processors were less powerful. In the

more recent papers, examples can be found of deeper architectures that are modelling highly nonlinear data, and therefore require a deep network with many neurons. Here the LM is less popular in favour of the ADAM optimisation technique. For the state-of-the-art methods, a hybrid approach, combining two different models provides many benefits. In the literature, the trend towards using attention networks in the hybrid models is evident. Furthermore, genetic algorithms form a key research area in optimising the hyperparameters of the models. As does transfer learning to decrease the training time of a model trained on a new dataset. It is advised that the use of hybrid models, attention networks, genetic algorithms, and transfer learning are investigated during the thesis.

3.4 Bayesian Neural Networks

In this section, a literature study is provided on Bayesian neural networks (BNNs). In section 3.1 the uncertainty quantification properties of Gaussian processes were praised for their application towards an aerodynamic model. This is because the aerodynamic model uses tolerances of the model which are chosen such that the model is conservative and the aircraft can always be flown safely. However, when a machine learning model is implemented to replace the conventional aerodynamic model that is described in subsection 2.1.1, the uncertainty coming from the machine learning model could become the new tolerances, or be used to aid in choosing the tolerances. This means that the tolerances could be tuned locally for certain parameter combinations, and angle of attack ranges. Allowing the aircraft to be flown on the edge of the flight envelope, and extracting even more performance out of the aircraft. BNNs thus offer an extra element compared to an ANN as it can also provide an uncertainty measure for a prediction. Specifically, BNNs are investigated due to their scalability towards larger datasets compared to Gaussian process regression.

The section is subdivided into different subsections. Firstly a background on BNNs is given in subsection 3.4.1, whereafter Monte Carlo sampling methods are discussed in subsection 3.4.2. Afterwards, the variational inference methods are discussed in subsection 3.4.3. Finally, a method named Linearised Laplace is discussed in subsection 3.4.4, finished by the applications of all the methods in subsection 3.4.5.

3.4.1 Background on Bayesian Neural Networks

A Bayesian neural network (BNN) replaces the weights and biases in a conventional neural network, with a distribution over these parameters. A single set of weights and biases in the network will no longer be a single scalar value, but for example a normal distribution over both of these values, each having their mean and standard deviation

associated with it. The goal of a BNN is to provide, besides the output, an uncertainty correlated to this output. BNNs are not optimizing, but rather marginalising. Therefore, they are finding the best distribution of weights. Compared to ANNs which use maximum likelihood estimation (MLE), a BNN uses maximum a posteriori estimates (MAP). In ANNs, methods such as gradient descent can be used to find the optimal values of the parameters of the model, while for BNN it uses different methods to determine the MAP. Bayesian techniques also offer a framework to understand many regularization techniques used in ANN architectures. BNNs can be considered special cases of ensemble learning, where many average-performing predictors perform better than a single expert predictor [57]. BNNs could provide better predictions compared to ANNs but mainly they are investigated for their uncertainty predictions.

An important aspect of BNNs is the way they are trained for Bayesian inference. Since the weights and biases of the neural network are no longer fixed values, but distributions over the weights and biases, probability distributions are used to describe the weights and biases. The goal is to update the prior distributions of the weights and biases, and therefore compute a posterior distribution. For this step, Bayes rule is needed, which calculates the posterior distribution of the model parameters, given the data. Bayes rule is given in Equation 3.3, which says that the posterior equals the likelihood multiplied by the prior, and normalised by the evidence. In Equation 3.4 the same equation is written in its familiar form, where the probability of the weights w given the data D, or the posterior, is the likelihood of the data given the weights, multiplied by the prior distribution $P(w)$, normalised by the evidence $P(D)$.

$$\text{Posterior} = \frac{\text{Likelihood} \cdot \text{Prior}}{\text{Evidence}} \tag{3.3}$$

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)} \tag{3.4}$$

Which can be rewritten

$$p(w|D) = \frac{p(D|w)p(w)}{\int p(D|w)p(w)dw} \tag{3.5}$$

To predict the output for a given input x, an average is made over all possible weights, weighted by the posterior probabilities.

$$p(y|x, D) = \int P(y|x, w) \cdot P(w|D)dw \tag{3.6}$$

The problem with a BNN starts here as the evidence or more specifically the integral in the denominator of Equation 3.5, $\int p(x, y|w)p(w)dw$ which is called the marginal is said

to be intractable, which means it cannot easily be calculated. For complex models such as a neural network, it is a high dimensional probability distribution. This is because of the size of the problem and the sheer number of weights and biases in a neural network no analytical solution of the integral can be found. Any integral over all possible weights would be intractable for any reasonably sized network. To solve these integrals, or find a reasonable estimation of the integral, two methods are proposed: variational inference, and Monte Carlo sampling methods. Variational methods approximate the posterior distribution with a parameterized approximate posterior. As for sampling methods, a finite set of network weights are sampled that match the posterior distribution.

3.4.2 Monte Carlo Sampling Methods

Sampling methods as discussed above use Monte Carlo integration which uses a finite set of random samples to approximate an integral. In this case, the integral is the intractable integral over the weights. Therefore, the Monte Carlo methods randomly sample a finite set of weights and use them to approximate the integral.

One can do this in several ways. Firstly, Monte Carlo Integration, where the marginal data likelihood and posterior distribution are calculated. As said before, the Monte Carlo integration method samples randomly from the network using random weights from the prior. To make a good estimate of the uncertainty, a significant number of weights need to be sampled from the network. This means the original problem of requiring to compute an integral over an almost infinite number of weights has been replaced by sampling over a finite but incredibly large number of weights [13]. For this reason, other methods such as Markov Chain Monte Carlo (MCMC) are preferred and often used for training BNNs.

MCMC is a method to sample from complex probabilistic distributions. MCMC does this by making a Markov Chain, where each link of the chain is a sample of the distribution. During the process, small steps are taken in many random directions from the current state. The acceptance rule is there to assess whether each random walk brings one closer to the minimum of the objective function. Since the samples are formed using a chain, the samples are dependent. To make them independent samples one can decide to only take the Nth sample, and discard the rest of the chain to obtain independent samples. MCMC allows sampling from a large set of distributions and scales well with increasing dimensionality [13]. There are many forms of the algorithm such as the Metropolis, and Metropolis-Hastings algorithms which provide further improvements or specific scenarios in which these algorithms perform better. The most relevant MCMC method is the Metropolis-Hastings algorithm [95]. This is because it does not require the exact probability distribution function to sample from, an approximation that is proportional to the true function is sufficient.

Besides Markov chain Monte Carlo methods, there is also the more advanced Hamiltonian Monte Carlo algorithm which uses the concept of Hamiltonian dynamics to more efficiently sample the probability distributions [12]. It uses Hamiltonian dynamics where equations describing the kinetic and potential energy of the system are used to determine the behaviour of the system over time. This method allows for more efficient exploration of the target space [89]. However, to solve the Hamiltonian equations, numerical integration must be performed, which requires the determination of some hyperparameters such as the step size, and number of integration steps. An even further extension to this method is the No-U-Turn Sampler (NUTS), which dynamically determines the hyperparameters of the Hamiltonian Monte Carlo method [52]. The method uses no-U-turn criteria to determine when the algorithm must be stopped as the trajectory of the Hamiltonian cannot turn back on itself. NUTS is more efficient and effective for complex and high-dimensional target distributions. The lack of scalability makes them less popular than variational inference.

3.4.3 Variational Inference

Variational inference, unlike MCMC, is not an exact method, but rather an approximate one. Instead of sampling from the exact posterior distribution, an approximate distribution is created, called the variational distribution ($q(\phi)$). Which is parameterized by a set of parameters ϕ. After which the values of the parameters ϕ are learned to match the exact distribution as closely as possible. Therefore, to perform variational inference one needs a method to measure the similarity between the approximate posterior and true posterior, but also a method to optimize the level of similarity.

The Kullback-Leibler Divergence (KL divergence) [66] is there to quantify how much difference there is between two distributions. It represents the average number of extra bits required to encode a sample from the true posterior distribution using the variational distribution. The KL divergence can be defined in terms of entropy, and cross-entropy of two probability distributions p, and q. With p the true distribution and q the predicted distribution.

$$\text{KL Divergence}(p||q) = \text{Cross-Entropy}(p, q) - \text{Entropy}(p) \tag{3.7}$$

The entropy in this case is a measure of the uncertainty, with higher entropy corresponding to higher uncertainty. If one considers that the entropy E can be written as $E(p) = \sum p \cdot log(p)$, and the cross-entropy as $E(q) = \sum p \cdot log(q)$. This allows to rewrite Equation 3.7.

$$\text{KL Divergence} = D_{KL}(p||q) = -\sum p(x) \cdot log\left(\frac{q(x)}{p(x)}\right) \tag{3.8}$$

However, since one is dealing with continuous random variables the summation is re-

placed with an integral. The KL divergence is given below for Bayesian inference.

$$D_{KL}(p||q) = \int p(x) \cdot log\left(\frac{p(x)}{q(x)}\right) dx \tag{3.9}$$

The KL divergence cannot be minimised since the exact posterior is not known. Therefore, the evidence lower bound (ELBO) is derived. If the ELBO is maximised the KL divergence is minimised, without the need for the exact posterior distribution. In Equation 3.10, the evidence lower bound is provided.

$$\text{Evidence Lower Bound (ELBO)} = \log(P(D)) - D_{KL}\left(q_\phi \| P\right) \tag{3.10}$$

This equation is known as the ELBO, which serves as the loss function. Since the probability of the data $log(P(D))$, depends only on the prior distribution, minimizing the second term $D_{KL}\left(q_\phi \| P\right)$, maximises the ELBO. The second term $D_{KL}\left(q_\phi \| P\right)$, is the KL divergence between the variational distribution q_ϕ, and the prior distribution. Maximising the ELBO is usually performed using stochastic variational inference.

Mean-Field Inference

Mean field inference is the concept of assuming the approximate posterior distribution (variational distribution) is fully factorized. A coordinate ascent algorithm can be derived to maximise the ELBO, and indirectly minimise the KL divergence [110]. This method can be inefficient for large datasets such as the one dealt with during the thesis since a full pass is required of the dataset at each iteration. Therefore, a stochastic method can also be devised, called stochastic variational inference (SVI). SVI tries to apply the properties of stochastic gradient descent towards Bayesian learning. This means that instead of a full pass, only a part of the dataset is needed to pass over [51]. This allows the algorithm to be used on very large datasets. The downside of this method is that it ignores the dependencies of the weights of the neural networks, which inhibits its ability to learn the real posterior distribution and model uncertainty [82].

Bayes-by-backprop and Reparametrization Trick

Another branch of variational inference is using a variation of backpropagation, which was used for training ANNs, but redesigned and applied towards BNNs. The algorithm aims to maximise the ELBO function to minimise the KL divergence of a variational distribution that is aimed to approximate the true posterior distribution over the weights. The main idea is that unbiased Monte Carlo estimates of the gradient of the ELBO function are calculated to learn the distribution over the weights of the network [15]. To calculate the gradients of the loss function, the reparametrization trick is required to be able to use classical backpropagation on the ELBO [61]. The reparametrization trick samples from a standard normal Gaussian distribution for example, which is then multiplied by the standard deviation and the mean added to it. The algorithm achieved good

results on classic example datasets such as MNIST, or a toy nonlinear regression dataset. The performance of the algorithm is advertised as being similar to that of dropout and scales well with dataset size as stochastic gradient descent schemes can be used, and can be run on GPU. Another method which aims to apply backpropagation towards BNNs is given in [48]. Here the assumption of a Gaussian posterior is made which can make the convergence more efficient. In [48], predictions made on real datasets showed that the method indeed has increased speed while offering good predictive capabilities. A variation of Bayes-by-backprop is given in [94] using PAC-Bayes upper bound on the risk. Bayes-by-backprop indeed minimises the objective function, but likely it is not doing this to a global minimum, but to a local minimum, which might have issues with generalising. The method provided in [94] aims to provide a more robust way of obtaining a generalisable BNN, and results were inconclusive as, depending on the dataset, the uncertainty estimates would improve or degrade.

Dropout

The variational dropout method for training a variational distribution to approximate the true posterior distribution draws its inspiration from the dropout training method that is used for regularizing ANNs [82]. In the literature, the methods show low computational complexity, with competitive performance. However, it has often unsatisfactory uncertainty predictions [36]. Also, in [39] a variant of Monte Carlo Dropout is presented for BNNs. It provides excellent predictive performance compared to variational inference and probabilistic back-propagation. The trade-off between computational complexity and uncertainty quality was left to follow-up work. Furthermore, the majority of the dataset sizes that were tested were below 10,000 data points.

3.4.4 Linearised Laplace

Another approach for providing uncertainty estimations is a method that transforms a trained neural network into a BNN post-training. One of these methods is called a Laplace approximation (LA), and it has become more known due to its post-hoc nature. The ANN can first be fine-tuned, after which the LA is applied, turning it into a model that can provide an uncertainty measure. It does this by locally approximating the Bayesian posterior with a Gaussian distribution. It has mostly been applied to image datasets such as MNIST [65], where it could provide solid predictions and uncertainties at the cost of high computational power. Not only the LA exist, but many other variants as well, such as linearised Laplace, or accelerated Linearised Laplace approximation [30]. All of these methods try to compress the matrices that need to be inverted or calculated and provide approximations towards these, as they require so much computational power and are not scalable. In the literature search that was conducted the methods were applied often towards 1-d regression datasets [30, 65, 10] but never towards more complex

multiple input datasets with large dataset size of over 100,000 data points. The regression methods are often compared against each other on these small datasets which look like sine waves. Therefore, it is not known whether it is even possible to perform Linearised Laplace on the large regression datasets of the thesis.

3.4.5 Applications

The applications of Bayesian neural networks can mostly be found in fields where knowing the uncertainty of the prediction adds value, such as in the medical field and general health fields [120, 29, 107], financial industry [56], and weather predictions [59].

In [67], a regression task of the quality of concrete is made using a BNN, which employs MCMC as its training method. This is because the available samples were low at 215, and very expensive per sample. The BNN outperformed the ANN which used just 10 hidden units. In [24], BNNs are used to predict the stock price before and during the Covid-19 pandemic. This is done using state-of-the-art parallel tempering MCMC with Langevin-gradients. This type of forecasting problem is slightly different from creating a predictive model. However, it can be compared to the model needing to extrapolate towards a region where it does not have data. The model performed adequately compared to the state-of-the-art for the ANNs. In [103], different training methods of BNNs are compared, in particular Monte Carlo sampling methods and variational approaches, but also Gaussian approximations such as the Laplace approximation and Monte Carlo dropout are considered. The five regression datasets used are homoscedastic 1-dimensional regression datasets, with a low number of training points (less than 200). This means that although the comparison of the models is one of the best that can be found in the literature, the datasets which are used are not at all comparable to the one used in the thesis. This begs the question of whether the conclusions made in the paper can be transferred towards the aerodynamic dataset of the thesis. In [124], a BNN is made using variational inference to predict floods. Afterwards, it was compared to ANNs and Gaussian process regression. The inflow data is heteroscedastic, meaning the variance changes over the input features, which is similar to the dataset in the thesis. The BNN was more accurate than the other models which were considered in the paper and provided good forecast uncertainty. In [109], BNNs are used to learn damage features from a finite element model to provide monitoring of the structural health of a structure. In this paper, the MCMC methods are discarded due to their high computational cost, as well as Gaussian process regressors, in favour of BNNs with variational inference methods. A stochastic gradient descent technique to optimise the ELBO loss function is used. In the application, 2 hidden layers were used, with 50 neurons per layer each. The training set consisted of 2,000 training samples, which is already one of the larger datasets that have been encountered for BNN in this literature study.

3.4.6 Conclusion

In the previous subsections, a deeper look was taken towards the literature on both sampling and variational inference methods for calculating the posterior distribution of BNNs. In this subsection, a small summary and comparison is made.

Sampling methods such as MCMC are slower but are sampling from the exact posterior, therefore it has no bias [14]. MCMC has been studied extensively and is more of interest to be used for smaller datasets where the extra computational cost is happily paid for more precise samples. Variational inference is much faster and is scalable towards larger problems. It has been shown to not necessarily provide worse predictions than MCMC, but this is case-dependent. What has been found is that often the uncertainty is underpredicted, meaning it is overconfident in the results [43]. The problem with the aerodynamic model that is trying to be modelled is that it requires high accuracy predictions, while also being scalable towards large datasets. As the dataset itself is large for the combat aircraft but also needs to be scalable towards even larger datasets in the future.

From the literature study and reading many literature reviews for BNNs, it can be seen that although BNNs are by no means a new field, the research in reducing the computational cost of BNNs to enable more widespread use is growing. Most of the modern research is focused on the variational inference methods due to their capability to use similar backpropagation algorithms as ANNs [43].

Very few industry examples of applications exist, and even fewer in aviation, where only one was found. Most industry examples are from industries where the available data is very scarce, and MCMC methods are chosen. However, for the application that presents itself in the thesis, with the very large dataset, the only sensible choice that can be made from the literature study is to use variational inference, and Bayes-by-backprop as they are the most scalable options.

3.5 Ensemble Methods

In this section, ensemble methods for regression are investigated. Since many different models are being explored in this literature study, the opportunity or necessity to make a combination of different models might present itself. This can be performed to achieve better generalisation towards a test dataset and improve the predictions coming from each of the base models.

In [80], a review of ensemble approaches which are most commonly used for regression is provided. Firstly, a definition of ensemble methods is provided to be a set of models, which are each obtained by applying a learning process, and this ensemble of models is combined in some way to create a final prediction. In the initial step of generating the set of models, some redundant models, which do not add towards the final predictive capabilities could be present. A pruning step can then be applied towards the ensemble set. Finally, a strategy needs to be created to optimally combine the ensemble of models.

Two ensemble methods have already been encountered during the discussion of tree-based models in section 3.2, namely bagging, and boosting. Bagging, or in full, bootstrap aggregating is a method developed to ensemble multiple machine learning models to reduce the variance. It is most commonly applied to decision tree methods, as was the case with the models in chapter 4. The method works by sampling N training datasets from the original dataset with replacement, therefore a data point might be found in multiple training datasets. For N large enough it is expected that (1-1/e) = 63% of unique samples are present in each dataset, which is called a bootstrap sample. Afterwards, the N models are trained and their outputs are averaged. Bagging requires the data to have variance for it to work. If the data has variance, then bagging can help avoid overfitting, stabilize the predictions, and increase the accuracy. However, due to the computational intensity of creating N models, with N being as large as possible, and that one assumes that all models have already been trained, this method is not used to combine different models in the thesis. Boosting is a term that refers to converting weak learners to strong learners. A weak learner is a model which performs only slightly better than random guessing, it is often defined in the context of classifier models. However, it can be extended to regression models when considering the error in percentage form. If a 50% error would be a model which randomly guesses the output, a weak learner model performs slightly better. The idea of a boosting algorithm is to train many weak learners sequentially, with each sequential model focusing on the weak points of the previous model. Although boosting is highly popular and advanced for tree-based models, such as extreme gradient boosting, for neural networks it is not as widespread. There are examples of people applying the algorithm towards neural networks. However, this is more niche and for neural networks or other models stacking is a more widespread ensemble method. In model stacking the optimum values of the weights are determined between the different base models.

In [22], one of the first implementations and research into stacked models is provided. The general conclusion that is made, and can be used during the thesis, is that stacking works better when the base models are more dissimilar. Some industry examples of stacked regression are provided in [60, 9, 75]. In [60], the electricity consumption of hot roll milling is predicted using a stacked model. The stacked model performed better than either of the base models and combined tree-based, support vector machines, and linear models.
In [9], the first feature importance of a stacked model is performed using the permutation variable importance method to obtain an explainable ensemble method.
In [75], an attention layer is used as its ensemble meta-model. As explained in subsection 3.3.4, attention networks form the state-of-the-art for many machine learning applications. Therefore, it is highly interesting to employ an attention ensemble model and compare it to a different meta-model. In [75], the ensemble technique is compared to the base models, a simple averaged ensemble, and a weighted average ensemble. The attention ensemble model outperforms the RMSE of the weighted average ensemble by around 15%, showing the power of this technique. The limitation of the case study which employs an attention ensemble method is that only tree-based models namely, XGBoost, Random Forest, and AdaBoost are used, and no other models such as neural networks.

3.6 Literature Study Conclusion and Research Objectives

In this section, a conclusion is made of the literature research in subsection 3.6.1, after which the research objectives of the thesis can be provided in subsection 3.6.2.

3.6.1 Literature Study Conclusion

For the literature study, many different candidate models for the aerodynamic model, shown in subsection 2.1.2, are provided. These were split up into different groups. Firstly, there were all the models used for performing regression on a tabular dataset, then tree-based models, artificial neural networks (ANNs), Bayesian neural networks (BNNs), and finally ensemble methods.

From the general data science methods available to regress the aerodynamic dataset, the Gaussian processes were deemed highly interesting. However, their scalability towards larger datasets is a concern. Therefore, later in the literature study, BNNs were investigated further. The other models considered in this section were the linear, and polynomial regression. These were not investigated further due to them requiring to specify the function shape up front. Radial Basis functions (RBF), which could also be considered a single-layer neural network with the RBF function as its activation func-

tion have the generalisability but not the depth which is required to be able to model the large and highly nonlinear aerodynamic dataset. Finally, support vector regression (SVR) was considered which although being common in the literature for regression, required careful selection of the linearisation function and kernel function. The Kernel trick which uses non-linear kernels such as the RBF can significantly increase the computational complexity for large datasets, and therefore SVR is not considered further for the application in this thesis.

The tree-based models were considered next, where decision trees, Random Forest, and gradient boosting models were investigated. From the literature study performed towards these models, many examples could be found for the Random Forest, and gradient boosting models, but not for single decision trees. Their simplicity and variability during training make them hard to implement in an industrial setting. The other models had plenty of examples and were praised for their competitive performance compared to neural networks. One paper in particular compared many different tree-based models towards neural networks, and concluded that the superiority of ANNs can not be guaranteed on tabular data [46]. Therefore, during the thesis, it can be investigated whether the tree-based models can also outperform other models on the aerodynamic dataset.

From the literature study performed on ANNs, one can determine that there have been many examples of ANNs being applied in aviation, combat aircraft, and other aerodynamic modelling problems. Many of these applications were performed over 30 years ago and used simple architectures, with outdated optimisation algorithms such as the Levenberg-Marquardt algorithm. The state-of-the-art research on ANNs focuses on hybrid approaches, where many different models are used to create a single model. Furthermore, genetic algorithms were called upon to research further, as they could form a great way of optimising the hyperparameters of a neural network. Therefore, research is proposed during the thesis not only towards applying neural networks towards the aerodynamic dataset, but also using hybrid approaches, with some of the other models considered in the literature study, and genetic algorithms to optimise the hyperparameters.

Finally, BNNs were investigated for the uncertainty quantification of the aerodynamic dataset, which is regarded as a potential way of reducing the tolerances of the aircraft's aerodynamic model and improving the flying capabilities of the aircraft. Their scalability compared to Gaussian processes makes them an interesting candidate. The two methods for training the network, Monte Carlo sampling methods, and variational inference, have been studied extensively and applications were found for both. Markov Chain Monte Carlo (MCMC) has been praised for its predictive performance and uncertainty quantification. However, it is a very computationally expensive method and does not scale well with larger datasets. It is for this reason that not many examples could be found of larger datasets for MCMC methods. Therefore, variational inference methods are preferred, specifically Bayes-by-backprop due to their similar nature to standard

neural network training, and their claimed on-par performance with MC dropout while having a better uncertainty prediction. Furthermore, the method scales well for larger datasets, and therefore it is determined to be an excellent candidate for the aerodynamic model.

3.6.2 Research Questions

From the conclusion of the literature study provided in subsection 3.6.1 one can set up the research questions for the thesis.

Since many different models were investigated and deemed promising for modelling the aerodynamic dataset of a combat aircraft, a general research question is formulated but also several sub-questions specific towards each machine learning model. Before formulating a research question, one can reiterate the main goal of the thesis which can be extracted from section 2.1. The main goal of the thesis is to create a process to model the aerodynamic dataset of a combat aircraft. This model and process should be capable of quickly and accurately creating a regression model. Furthermore, the model should be scalable towards a larger dataset, and the process needs to be robust so that an update or change in the dataset can be implemented.

Now that the main objective is defined, the research questions can be formulated. The main research question is:

Research Question: *Which machine learning model among tree-based models, artificial neural networks, and Bayesian neural networks demonstrates the best predictive performance inside the tolerances for the aerodynamic coefficients?*

This research question can be subdivided into multiple subquestions for the artificial neural network, tree-based models, and Bayesian neural networks respectively. Important aspects in each of these research areas are their predictive performance, training time, multiple single-output models or a single multiple-output model, and generalizability. Other aspects of the research questions are how genetic algorithms can be implemented, and ensemble methods. The specific research questions that have been devised are presented in the list below.

- Q1: What is the capability of tree-based models to model the aerodynamic dataset of a combat aircraft?
  - Q1.1: Which tree-based model performs best in terms of MSE, and percentage within the tolerance?
  - Q1.2: What is the effect of the non-smooth nature in modelling the dataset, and which methods can be applied to mitigate any negative effects?
  - Q1.3: Does the increased explainability of tree-based models offer a competitive advantage over the other machine learning models?
  - Q1.4: Which tree-based models are capable of making a multiple-output model, and does it outperform single-output models for each of the aerodynamic coefficients?
- Q2: What is the capability of artificial neural networks to model the aerodynamic dataset of a combat aircraft?
  - Q2.1: What is the predictive performance in terms of MSE, and percentage within the tolerance of a test dataset?
  - Q2.2: Does a multiple-output model outperform single-output models for each of the aerodynamic coefficients?
  - Q2.3: What are the benefits of genetic algorithms in optimising the hyperparameters of a neural network, and what is their computational cost?
  - Q2.4: Does transfer learning improve the MSE when training a neural network, furthermore, does it reduce the training time of a neural network, and by how much?
  - Q2.5: What other methods could be employed to improve the predictive performance of neural networks?
- Q3: What is the capability of Bayesian Neural Networks to model the aerodynamic dataset of a combat aircraft and provide an uncertainty prediction?
  - Q3.1: What is the predictive performance in terms of MSE, and percentage within the tolerance of a test dataset?
  - Q3.2: What is the correlation between the tolerances and the uncertainties which are coming from the Bayesian neural network, and does the reliability of uncertainties provide a reason for reducing the tolerances of the dataset?
  - Q3.3: Are Bayesian neural networks scalable towards larger datasets, and does their training time hold back their use?
  - Q3.4: Does a multiple-output model outperform single-output models for each of the aerodynamic coefficients, and why?
- Q4: What is the increase in performance when creating a stacked model between any of the previously mentioned machine learning models?

- Q4.1: How do the predictive performances of tree-based, artificial neural networks, and Bayesian neural networks compare in terms of MSE and percentage within tolerance?
- Q4.2: How do the training times of the different models compare?
- Q4.3: Which meta-model provides the highest performance increase for the ensemble model, and at what computational cost?
- Q4.4: What increase in performance can be expected when implementing a stacked model?

Chapter 4

Tree-Based Models

In this chapter, the application of tree-based models towards the prediction of the aerodynamics of a combat aircraft is shown and discussed.

For this chapter, the research questions which are aimed to be answered are provided below (from subsection 3.6.2). The main research question which is answered in this section is: *What is the capability of tree-based models to model the aerodynamic dataset of a combat aircraft?* Which can be further subdivided into different subquestions as can be seen below:

- Q1.1: Which tree-based model performs best in terms of MSE, and percentage within the tolerance? (section 4.3)
- Q1.2: What is the effect of the non-smooth nature in modelling the dataset, and which methods can be applied to mitigate any negative effects? (section 4.2)
- Q1.3: Does the increased explainability of tree-based models offer a competitive advantage over the other machine learning models? (section 4.4)
- Q1.4: Which tree-based models are capable of making a multiple-output model, and does it outperform single-output models for each of the aerodynamic coefficients? (section 4.3)

Based on these research questions, an initial hypothesis can be formulated. The initial hypothesis for tree-based models is that they are very competitive in terms of performance compared to other machine learning models. However, their smoothness might limit their applicability towards aerodynamically modelling a combat aircraft. To test

this hypothesis and answer the research questions, an experiment is devised, where each of the models explained and researched in the literature study are trained and compared on the combat aircraft aerodynamic dataset. After which, their training times and predictive performance will be evaluated and compared against each other.

The models which are considered in this section are the ones from the literature study. Namely, the Random Forest model, which combines many decision trees using ensemble techniques of bootstrapping and bagging. Then there are the gradient boosting models, AdaBoost, XGBoost, LightGBM, and CatBoost. From the literature study, the CatBoost model should outperform the XGBoost and lightGBM models, in both predictive performance and training time, which will be assessed in this chapter.

The chapter starts with a description and manual optimisation of the hyperparameters of each of the models in section 4.1. This section not only contains the optimisation but also a small enquiry into the smoothness of tree-based models, and how this can be resolved. The smoothness of tree-based models is discussed in section 4.2. In section 4.3, the results of the tree-based models are presented, after which the explainability of the models is discussed in section 4.4.

4.1 Hyperparameter Optimisation

Even though tree-based models are known for their ease of implementation and relatively low number of hyperparameters of each of the models, one still needs to make sure that a comparison can be made between all of them using an optimised model for each. That is why in this section, the hyperparameters of each of the models are chosen and optimised.

For this initial study, a small hyperparameter optimisation is performed. This optimisation is not performed using any optimisation algorithm but by manual optimisation to test out different hyperparameter configurations and bring all the models to a comparable level where the models have reached, or almost reached their limit on the dataset. As has been tested in subsection A.1.6, in Appendix A, optimising with algorithms and extensively searching the hyperparameter space would only result in a small performance gain. In the specific experiment that is performed for the CatBoost model, the total training and optimising time increased by a factor of 100, which resulted in a 10% reduction in the MSE. Therefore, two things are concluded. First, the difference in the models is such that optimisation would not change the outcome of testing the hypothesis in this section. Secondly, the gain in performance is too low, and the risk of overfitting is too high, for the computational cost of running this optimisation.
Each model trained for optimising the hyperparameters used a 5-fold split cross-validation with the results that are shown being the average of the 5 splits. In subsection B.1.1, an analysis is performed on the number of splits for the k-fold cross-validation,

which concluded that the 5-fold cross-validation provides a more consistent estimate of the test loss, while also having lower training time than a 10-fold cross-validation. The optimisation is performed on the pitching moment coefficient C_m, using dataset 1 (section 2.2) since the computational cost of tree-based models is relatively low as will be seen in this section. The pitching moment coefficient is used to determine the optimal hyperparameters as it is the hardest aerodynamic coefficient to predict, with the tightest tolerances, and most critical behaviour for the unstable combat aircraft configuration.

The actual optimisation of each of the models that were described in the literature study is shown in Appendix A, section A.1. In this section, only a summary of the optimisation is provided in Table 4.1 to draw some conclusions on the differences between these models.

In Table 4.1, a summary of the optimised Random Forest, AdaBoost, XGBoost, lightGBM and CatBoost models is presented. What can be seen from the table is that CatBoost has by far the best model performance, and AdaBoost by far the worst. The difference between the best and worst models is significant in terms of MSE, but also in terms of training times. One can see that the training time in seconds is relatively low for all the models. The training time is so low that it is not relevant to consider the training time as a real constraint for using any of these models. It is hypothesised that the AdaBoost model, with the use of its single split decision trees, is not capable of modelling the dataset. Furthermore, the training time is an outlier compared to the other models as is the MSE, which is almost two orders of magnitude larger than the Random Forest model. Therefore, it is excluded from the further analysis in this thesis.

From the initial literature study of the models performed in section 3.4, it was expected that XGBoost would be the slowest model to train. However, this is not the case for the optimised models presented in Table 4.1. The reason for this can be hypothesised to lie in the convergence of the different models. While lightGBM and CatBoost are faster at building their decision trees, they require more decision trees than XGBoost, as the test loss of XGBoost levels off faster. This is shown in the difference of convergence between the models, which is shown in Appendix A, subsection A.1.7, in Figure A.1, Figure A.2, and Figure A.3 for the CatBoost, lightGBM, and XGBoost respectively.

The CatBoost model has the lowest MSE of the tree-based models on the pitching moment coefficient. One observation that can be made is that the models are ranked based on their date of release, with the Random Forest model being the oldest and CatBoost the most recent. One can imagine that the developers of each of the models are implementing improvements towards the other models. Since no categorical features are used in the training of the models, the strength of CatBoost being optimised for categorical features is not utilised. However, CatBoost distinguishes itself by making symmetrical decision trees and ordered boosting to prevent overfitting. Therefore, either or both could be contributing towards the CatBoost models' success.

Table 4.1: Summary of the MSE and training time for the optimised Random Forest, XGBoost, lightGBM, and CatBoost models.

Model	Time [s]	MSE
Random Forest	8.1	4.80e-5
AdaBoost	737	1.56e-3
XGBoost	31.4	1.16e-5
LightGBM	43.5	9.06e-6
CatBoost	58.4	4.40e-6

Besides the MSE, one can also look at the actual predictions that are made on the aerodynamic dataset, and compare their performance for some of the control surface deflections, and flight parameter settings. In Figure 4.1 and Figure 4.2, there are two examples of two test data polars, with the predictions of the optimised Random Forest, XGBoost, lightGBM and CatBoost models. The models have been trained using k-fold cross-validation, and the predictions of one of the optimised models are shown below. The predictions are made on a sample of the test data set, which none of the models have seen while training. This means it is the models' direct capability to generalise to unseen aerodynamic data for a combat aircraft. Firstly, the same conclusions can be made from the figures as from Table 4.1, the best-predicting models are the LightGBM and CatBoost models. The XGBoost and Random Forest models perform worse than the CatBoost and LightGBM models, as they exhibit an offset from the test data in both examples. In Figure 4.2 both models even perform large jumps at nearly identical locations at higher angles of attack. The reason for these large jumps will be investigated in more detail in section 4.2 as well as possible ways of counteracting this. Luckily, the other two models (lightGBM, and CatBoost) do not exhibit this sort of behaviour, and although none of the models are considered to be smooth, the lightGBM and CatBoost models follow the test dataset, and their jumps are minor compared to the Random Forest and XGBoost model.

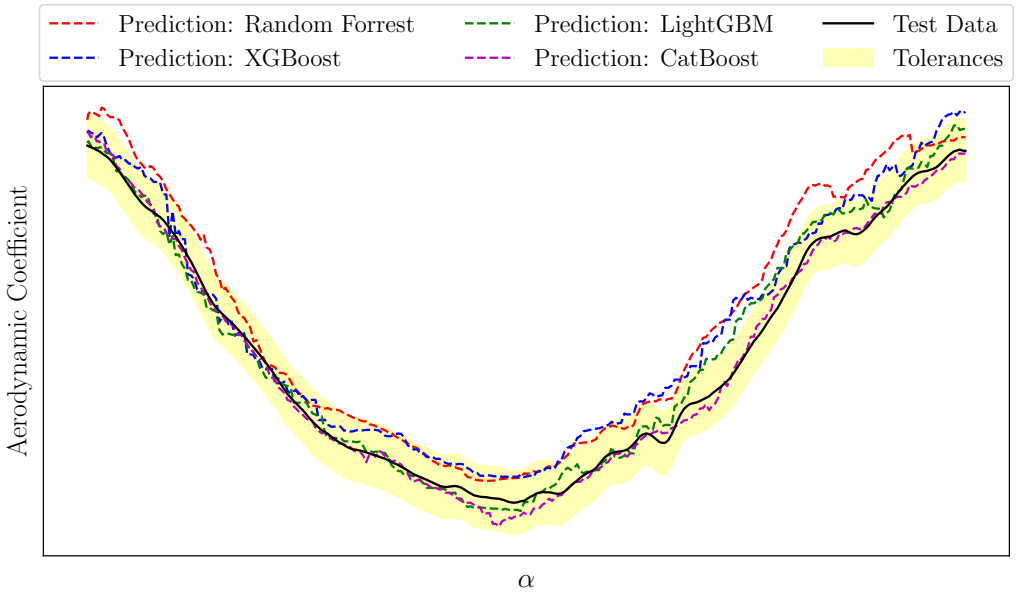


Figure 4.1: Prediction of test data polar 1 for the Random Forest, LightGBM, XGBoost, and CatBoost models.

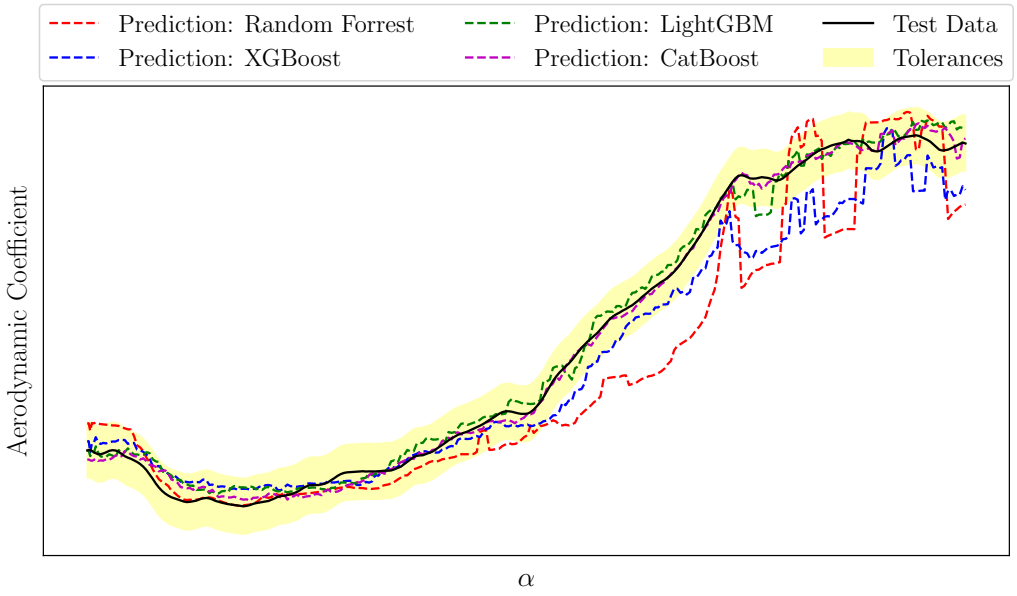


Figure 4.2: Prediction of test data polar 2 for the Random Forest, LightGBM, XGBoost, and CatBoost models.

4.2 Tree-based Model Smoothness

The non-smooth behaviour of tree-based models, as was exemplified in section 4.1, and more specifically in Figure 4.2, was determined to be a limitation and thus a concern for modelling the aerodynamic dataset, and this section aims to investigate it in more detail.

In Figure 4.3, a plot is provided that gives the aerodynamic coefficient over 2 input features. One is the continuous angle of attack, while the other is one of the input features, that is not continuous. The specific input feature that is used cannot be shown due to confidentiality. The data that is used is coming from dataset 1, with only 2 input features selected for this example. The test and train data are also provided in the plot. A tree-based model such as the ones used to make the Random Forest model, and the XGBoost model, splits the data between the training data it receives. This process of splitting the data and predicting the average value below and above the split boundary has been explained in section 3.4. Here one can see that the data is split along the angle of attack, as some of the jumps seen in the figure are discontinuous. Furthermore, one can see even more clearly the splits that have been made along the "Input feature" direction. For example at a high value of the input feature, between the second and third training polar counted from the top, the aerodynamic coefficient jumps values exactly in the middle between training polar 2 and 3. This is the exact location where the test polar is shown, as the run plan of the wind tunnel data gathering has equally spaced the polars along the input feature direction.

Another important element towards the puzzle of why these jumps are seen in the test polars in Figure 4.2 is that the wind tunnel polars are not perfectly maintaining a continuous value of the input feature. This is due to the wind tunnel, which as a function of the angle of attack cannot maintain an exact value for the input feature. As the test polar is located exactly on this split boundary, and the value can go over and under the decision boundary, it is bouncing between two levels of the decision tree, this is then shown in the plot Figure 4.2 as sharp jumps between the two levels.

What can be concluded is that even though the data that is seen by the CatBoost and Random Forest model is identical, the split for the CatBoost model is not made in the same location as for the Random Forest since the CatBoost model is not making a split with a decision tree in the same way a Random Forest model does. The CatBoost model makes sequential decision trees based on the residuals, meaning that when a value falls in between two training polars, the predictions are not like a step function, but rather a summation of all the previous decision trees, multiplied by a learning rate. This comes down to the fact that the CatBoost model can provide smoother predictions in the region where the Random Forest will make a discrete step in its predictions.

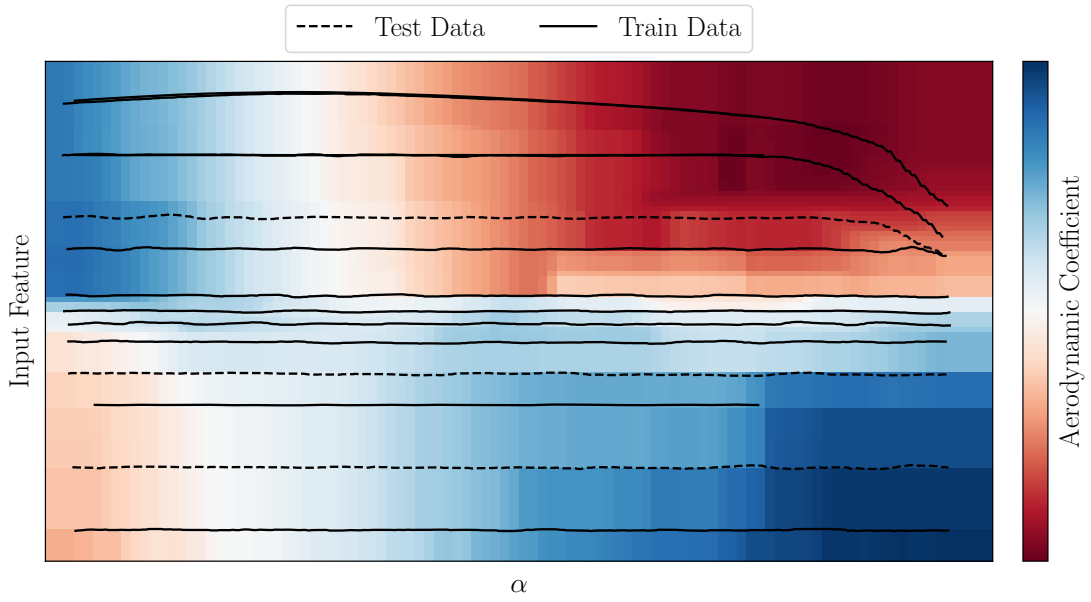


Figure 4.3: Random Forest model trained on the angle of attack and one input feature. Showing a plot of the response surface for one of the aerodynamic coefficients.

The simple technique of moving average smoothing can be used to filter out the small jumps the tree-based models make. As it is extremely simple to implement, and requires no modification towards the model. However, as was seen in section 4.2 some of the models which are investigated in this section contain jumps between different levels of the tree-based models. If one could identify when such a jump is occurring, then a measure could be taken to level out these jumps. However, the problem is much larger, that right in the middle of a training polar, the tree-based model such as a Random Forest discretely jumps between the level of the one training polar towards the next. To smooth this, a different approach must be taken that is not so evident, as one needs to distinguish between a sharp jump in the aerodynamic coefficient and the model which is jumping between different splits in the prediction. The aerodynamic coefficients of the combat aircraft are extremely nonlinear meaning that making this distinction could be very difficult.

4.3 Results

In this section, the results of each of the tree-based models are provided. The models have been trained on dataset 1, using the hyperparameters which have been optimised in section 4.1. Dataset 1 is explained in section 2.2. In subsection 4.3.1, the single-output model results are presented, and in subsection 4.3.2, the multiple-output model results are presented.

4.3.1 Single-Output Models

In this subsection, the results of the single-output models that are trained for each of the aerodynamic coefficients are provided. The MSE of the models are provided in Table 4.2, and the percentage within tolerance of the test dataset in Table 4.3.

In Table 4.2 the MSE of all the models are shown for all the aerodynamic coefficients. The results for the pitching moment coefficient could already be deduced from section 4.1, where the CatBoost model was the clear best-performing model. The LightGBM model was significantly worse but still provided good predictions. Finally, the XGBoost, and Random Forest models provide less good predictions for the MSE as the jumps that were extensively discussed in section 4.2 cause the MSE to be very large.

Table 4.2: Mean squared error (MSE) comparison for all tree-based single-output models and aerodynamic coefficients. RF = Random Forest, XGB = XGBoost, LGB = LightGBM, CAT = CatBoost.

	RF	**XGB**	**LGB**	**CAT**
$\boldsymbol{C_m}$	4.802e-05	1.163e-05	8.497e-06	**4.390e-06**
$\boldsymbol{C_l}$	1.013e-05	7.010e-06	6.370e-06	**5.703e-06**
$\boldsymbol{C_n}$	4.628e-06	3.252e-06	3.459e-06	**2.914e-06**
$\boldsymbol{C_x}$	1.159e-05	2.507e-06	**1.587e-06**	2.524e-06
$\boldsymbol{C_y}$	8.719e-06	**6.472e-06**	1.497e-05	1.212e-05
$\boldsymbol{C_z}$	1.494e-04	6.774e-05	5.863e-05	**4.677e-05**

In Table 4.3 the percentage within tolerance (PWT) for each of the models is provided. One can see that the CatBoost is by far the most capable model, as it predicts 97% within the tolerance of the pitching moment coefficient test dataset. However, what could not be deduced from the moment coefficient is that the XGBoost model is also highly capable and even the best model for the C_n, and C_y. However, it has more variation than the CatBoost model. The CatBoost model performs very well for all the coefficients, while the XGBoost model has two coefficients which it does not predict well at all, compared to its average, these are the C_m, and the C_x. The same trends are present for the lightGBM model, and Random Forest model. These models also

have lower PWT values specifically for the coefficients C_m, and C_x. Since the moment coefficient is the most nonlinear aerodynamic coefficient, it is not surprising that some of the models struggle with it. However, from C_x this quality is not known, and it is hypothesised that it is caused by the non-smoothness of the tree-based models, which are struggling to model the smooth behaviour of the C_x. Therefore, it is left to the artificial neural network and Bayesian neural network chapters (chapter 5, chapter 6) to investigate whether the smoother models have this same drop in performance for C_x or not.

Table 4.3: Percentage of the model predictions within tolerance of the test dataset (PWT), comparison for all the tree-based models. RF = Random Forest, XGB = XGBoost, LGB = LightGBM, CAT = CatBoost.

	PWT RF [%]	**PWT XGB** [%]	**PWT LGB** [%]	**PWT CAT** [%]
$\boldsymbol{C_m}$	68.8	87.3	91.4	**97.0**
$\boldsymbol{C_l}$	94.7	96.2	96.3	**96.7**
$\boldsymbol{C_n}$	95.7	**97.9**	97.4	**97.9**
$\boldsymbol{C_x}$	72.8	89.2	92.5	**96.7**
$\boldsymbol{C_y}$	99.1	**99.5**	97.9	98.2
$\boldsymbol{C_z}$	99.3	99.8	**99.9**	**99.9**

4.3.2 Multiple-Output Models

In this subsection, multiple-output models of the tree-based models are compared to their single-output counterparts to compare the performance.

Firstly, one can see that there are only multiple-output models for the Random Forest model and XGBoost models. The lightGBM and CatBoost models do not support multiple-output regression and therefore are not calculated.

Since the multiple-output models were assumed to require a more elaborate model, a separate manual optimisation is performed for the Random Forest and XGBoost models. As will be clear from the discussion below, the XGBoost model could maintain its hyperparameters. The Random Forest hyperparameters were changed from 100 to 200 estimators.

In Table 4.4 the MSE of the multiple-output models of the Random Forest and XGBoost model are presented, with the single-output variants also present in the table such that a comparison can be made. Firstly, what can be noticed immediately is that the XGBoost model has the same MSE as its single-output counterpart. Meaning that the sequential trees that are made, are made in parallel. With the same seed specified in the model, the model provides an identical result towards training multiple single-output models.

For the Random Forest model, this cannot be said, since the multiple-output model is far worse performing than the single-output model.

The same trends are carried over towards the percentage within the tolerance of both models. The XGBoost model for the multiple-output and single-output models is still identical to the PWT. The PWT for the Random Forest model is considerably lower than the single-output counterpart, as was the case for the MSE.

From this, the conclusion is made that multiple-output models do not provide improved performance compared to their single-output counterparts.

Table 4.4: Mean squared error (MSE) comparison for the tree-based multiple-output models compared to the single-output models for all aerodynamic coefficients. RF = Random Forest, XGB = XGBoost

	RF Multi	**RF Single**	**XGB Multi**	**XGB Single**
C_m	1.042e-04	4.802e-05	1.163e-05	1.163e-05
C_l	2.379e-05	1.013e-05	7.011e-06	7.010e-06
C_n	3.409e-05	4.628e-06	3.252e-06	3.252e-06
C_x	6.812e-05	1.159e-05	2.507e-06	2.507e-06
C_y	5.962e-05	8.719e-06	6.472e-06	6.472e-06
C_z	2.515e-04	1.494e-04	6.774e-05	6.774e-05

Table 4.5: Percentage of the model predictions within tolerance of the test dataset (PWT), comparison for the tree-based multiple-output models and single-output models for all aerodynamic coefficients. RF = Random Forest, XGB = XGBoost.

	RF Multi [%]	**RF Single** [%]	**XGB Multi** [%]	**XGB Single** [%]
C_m	49.5	68.8	87.3	87.3
C_l	85.1	94.7	96.2	96.2
C_n	88.2	95.7	97.9	97.9
C_x	33.8	72.8	89.2	89.2
C_y	93.2	99.1	99.5	99.5
C_z	98.2	99.3	99.8	99.8

4.4 Feature Importance of Tree-Based Models

In this section, a closer look is taken at the explainability of tree-based models. The feature importance calculates how important, that one of the features is towards the final prediction, and is easily extracted from the trained models. There are multiple methods for achieving this, and each tree-based model calculates the feature importance differently. One thing that is very important to note, is that the feature importance can only be as good as the model. If the model is predicting poorly, the feature importance might not be so valuable, as there is not much value in explaining how a model makes a bad prediction. Therefore, the feature importance of the CatBoost model is discussed in this section. As could be seen in section 4.3, the CatBoost model is on average the best-performing model. The feature importance of the Random Forest, XGBoost, and LightGBM is left for section A.2, of Appendix A.

The feature importance of the CatBoost model is calculated based on how important each feature was in the splits of the decision trees. To calculate the feature importance of, for example, the Mach number, the original training data will be passed through the model with the Mach number being randomly permuted. This means that actually, the aerodynamic coefficient is no longer corresponding to the correct Mach number, while the other features remain constant. The aggregated error that is made by doing this for the Mach number features, tells one about the importance of this feature for the output value. One can also additionally check the change of the loss function when this is performed.

The feature importance for the CatBoost model for all the aerodynamic coefficients is provided in Table 4.6. What one would like to see from the feature importance is that the features, which one traditionally attributes towards an aerodynamic coefficient, are also learnt to be important by the model. One can see for example that for the pitching moment coefficient C_m, the flap deflections are the most dominant feature, together with of course the angle of attack. This can be very well understood as the flaps are the main mechanism to pitch the aircraft together with the canard, which also has a significant impact. Values such as the sideslip, front slats, and rudder deflection have a negligible impact. For the rolling moment coefficient C_l, the sideslip angle forms a significant contribution. The flap deflections still retain their dominant effect even for the rolling moment coefficient. The reasoning for this is that the split flap configuration, where one flap is up and the other is down creates a huge rolling moment, which is also one of the main methods for rolling the aircraft. As for the yawing moment coefficient, it is very evident that the rudder has the biggest influence, which is of course very logical. Also, the sideslip angle becomes the second most dominant feature due to the Weathervane effect. Finally, for the side force coefficients, one can see that for the C_x, which is closely related towards the drag coefficient, the angle of attack forms the most dominant feature, together with the Mach number, and again the flaps. For the side force coefficient C_y

Table 4.6: Feature Importance of the control surface deflections, angle of attack, sideslip angle, and Mach number for the CatBoost model, for all the aerodynamic coefficients.

	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	**M**	$\boldsymbol{\eta}$	$\boldsymbol{\varepsilon}$	$\boldsymbol{\delta_1}$	$\boldsymbol{\delta_3}$	$\boldsymbol{\zeta}$
$\boldsymbol{C_m}$	22.158	0.147	15.247	12.687	1.731	28.186	19.724	0.121
$\boldsymbol{C_l}$	10.401	10.796	2.352	1.412	0.709	38.288	34.775	1.267
$\boldsymbol{C_n}$	9.801	17.540	4.108	1.506	0.484	10.293	9.734	46.535
$\boldsymbol{C_x}$	39.046	0.092	12.758	9.484	10.376	9.149	18.454	0.640
$\boldsymbol{C_y}$	7.414	52.406	3.158	0.641	0.419	8.641	8.942	18.378
$\boldsymbol{C_z}$	67.100	0.082	5.310	0.551	0.290	12.729	14.316	0.009

the sideslip angle is the most important together with the rudder deflection. Finally, for the C_z, which is closely correlated towards the lift coefficient, the angle of attack is a very dominant feature.

One can conclude from this section that the feature importance has a high correlation with the true physical understanding of the aircraft's behaviour. This indicates that the CatBoost model learnt the behaviour of the aircraft and which features are more important for the longitudinal coefficients, and which for the lateral coefficients. When one returns towards the research question that is aimed to be answered by this section, namely: *Does the increased explainability of tree-based models offer a competitive advantage over the other machine learning models?*, one can only say that the increased explainability cannot be directly linked to better predictive performance, and is more a result of the model. However, it does provide an insight into whether what the model learnt, actually makes sense. From the feature importance, one can detect if a model is relying on physically meaningful features, which could be hard to detect with a model that does not provide the feature importance. Furthermore, it could provide an easier way towards certification and compliance with any regulatory body. Furthermore, one could determine that for example for the C_z the sideslip and rudder deflection are almost useless for the prediction of C_z, which could facilitate removing the feature from the analysis, but without the feature importance, one could never determine this so easily. Therefore, feature importance with their added information towards the user of the model, does indeed provide a competitive advantage, or an extra reason to consider them, as compared to a model that does not provide feature importance.

4.5 Conclusion

To end this section, the hypothesis that was stated at the beginning of this section is provided again. The hypothesis was that tree-based models are very competitive in terms of performance compared to other machine learning models. However, their smoothness might limit their application in the industry. From the above analysis, one can conclude that the hypothesis is partly true since it has been shown that the CatBoost and LightGBM models are an extremely well-performing model for predicting the aerodynamic dataset of a combat aircraft, and can generalise well towards unseen data. The smoothness of both these models was better than the Random Forest and XGBoost models, but could still be a concern for their implementation. Even though the prediction of these models follows the test dataset well, it is unclear whether the smoothness would allow them to ever be capable of replacing the full aerodynamic dataset without additional smoothing techniques, since often the derivative of the aerodynamic coefficient is also an important characteristic of the aircraft.

When one returns towards the research questions which were posed at the beginning of this chapter.

- **Q1.1: Which tree-based model performs best in terms of MSE, and percentage within the tolerance?**
  From section 4.3, one can determine that the CatBoost model is the most consistently well-performing model, with the LightGBM model and XGBoost model being the best for the C_x, and C_y respectively. For the PWT the CatBoost model is shown to be even more dominant, only being beaten by the XGBoost model for C_y. Tree-based models achieve a PWT of over 96.6% for each of the aerodynamic coefficients, showing that the aerodynamic dataset can be predicted well by tree-based models. It is hypothesised that CatBoost performs so well due to the symmetrical decision trees which create the model, and ordered boosting to prevent overfitting.

- **Q1.2: What is the effect of the non-smooth nature in modelling the dataset, and which methods can be applied to mitigate any negative effects?**
  From section 4.2, one can distinguish two non-smooth characters of the tree-based models. One is where the prediction is jumping between two levels of the model, which is very hard to resolve, and forms a problem for their implementation. Specifically, Random Forest and XGBoost models suffer from this behaviour. However, the non-smooth nature of CatBoost and lightGBM could be resolved by simple mean averaging and does not form a significant problem for using them to model the aerodynamics of a combat aircraft. Especially looking at the performance of XGBoost, Random Forest, and LightGBM on the C_x coefficient, which is smoother

than the other coefficients, the general conclusion can be made that these models are not optimised to predict smooth functions.

- **Q1.3: Does the increased explainability of tree-based models offer a competitive advantage over the other machine learning models?**
  In section 4.4, the feature importance was determined to be able to tell the user the physical features the model had learnt for predicting each of the aerodynamic coefficients. This was identified as a competitive advantage for tree-based models.

- **Q1.4: Which tree-based models are capable of making a multiple-output model, and does it outperform single-output models for each of the aerodynamic coefficients?**
  From subsection 4.3.2, the Random Forest and XGBoost models were the only ones supporting a multiple-output model. From this section, it was determined that creating a multiple-output model has a detrimental effect on the prediction of the Random Forest model and no influence on the XGBoost model. Furthermore, due to the low training times of the tree-based models, it is advised to use single-output models when using tree-based models.

After first analysing other machine learning models in the next chapters of the thesis, the tree-based models will be compared against neural networks, and Bayesian neural networks in chapter 7.

Chapter 5

Artificial Neural Networks

In this chapter, artificial neural networks (ANNs) are investigated for their capability to model the aerodynamic dataset. For more information on what ANNs are and how they operate, one can consult the literature study section on ANNs in section 3.3.

In this chapter one aims to answer the research questions that were devised at the end of the literature study in subsection 3.6.2. The main research question for this chapter is: *What is the capability of Artificial Neural Networks to model the aerodynamic dataset of a combat aircraft?* This research question can further be split up into the subquestions which are shown below.

- Q2.1: What is the predictive performance in terms of MSE, and percentage within the tolerance of a test dataset? (section 5.3)
- Q2.2: Does a multiple-output model outperform single-output models for each of the aerodynamic coefficients? (section 5.3)
- Q2.3: What are the benefits of genetic algorithms in optimising the hyperparameters of a neural network, and what is their computational cost? (section 5.2)
- Q2.4: Does transfer learning improve the MSE when training a neural network, furthermore, does it reduce the training time of a neural network, and by how much? (section 5.4)
- Q2.5: What other methods could be employed to improve the predictive performance of neural networks? (section 5.5)

The chapter is split up into different sections. Firstly, the specifications of the ANN

are provided in section 5.1, where different studies are made to determine the optimal hyperparameters of the ANN. Afterwards, genetic algorithms are extensively discussed in section 5.2 and their implementation towards the hyperparameter optimisation of ANNs is provided. After the genetic optimisation of the hyperparameters, the results of the ANN are provided in section 5.3. Two additional investigations are made after the results of the optimised ANN are provided. This is firstly the concept of transfer learning in section 5.4, and subsequently whether augmenting the training data by adding the gradient at each point is beneficial for the predictive performance of the model in section 5.5. Finally, a conclusion is presented in section 5.6.

All the results of the models which are shown in this chapter have been trained using a 5-fold cross-validation. This is because in subsection B.1.1, an analysis is performed on the number of splits for the k-fold cross-validation, which concluded that the 5-fold cross-validation provides a more consistent estimate of the test loss, while also having lower training time than a 10-fold cross-validation.

5.1 Categorical Hyperparameter Selection

In this section, some general observations about the neural network that will be created are discussed and optimised. These observations relate to the categorical hyperparameters of the ANN, which still have a significant impact on the performance and generalizability of the neural network. Therefore, a careful selection needs to be made. Due to the tedious nature of finding the optimal values of the hyperparameters of an ANN, the optimisation is moved to Appendix B, section B.1. In this section, a summary is provided of the findings that can be found in section B.1.

The batch size controls how many points from each of the polars are seen by the model for a single step in the optimiser. It is hypothesized that a large batch size of over 10,000 points is required, such that each step of the optimiser contains at least a few points from each polar in the training data. The study performed in subsection B.1.2, proved that a batch size of 128 was optimal, and larger batch sizes were more unstable during convergence and had higher training times. The learning rate that is chosen for the ANN was shown to be optimal at 1e-4 in subsection B.1.3. This is because the higher learning rate of 1e-3 converged towards a higher value of the test MSE loss. The lower learning rate of 1e-5 converged so slowly that it did not reach the MSE of the 1e-4 learning rate model in the specified number of epochs. Batch normalisation allows for faster convergence due to the possibility of using larger learning rates [85]. However, in the study performed in subsection B.1.4, comparing a model with and without batch normalisation showed that batch normalisation had an apparent effect on the convergence pattern to lower the learning rate. Therefore, the hypothesis could be true that it allows for higher learning rates. However, finding the maximum learning rate to which batch

normalisation can then be applied was not deemed useful since the learning rate of 1e-4 was already determined. The activation function is an important element of the ANN as it has a large effect on the ease of training the model. In subsection B.1.5, different activation functions were tested and from this analysis, the ReLU, LeakyReLU, and PReLU, had similar convergence patterns and MSE of the test loss. Therefore, it was decided to take the ReLU activation function for its simplicity compared to the other activation functions. The dropout variable controls the probability of a given neuron being excluded from a training step. This reduces the probability of an ANN to rely too heavily on only a few neurons in the network. From the analysis performed in subsection B.1.6 a dropout value p of 0.5 was deemed optimal due to it having the lowest average MSE on the test data. Dropout is commonly used to reduce overfitting, as is L1, and L2 regularization. However, it was deemed unnecessary to implement regularization as none of the networks that were run up to this point showed signs of overfitting. Different optimisation functions are also compared against each other in subsection B.1.7. From this analysis, it was shown that the ADAM optimiser was most capable. The ADAM optimiser is used in combination with the mean squared error (MSE) as its metric. Finally, the number of epochs is taken to be 300 which was on average double the required number of epochs which were used to perform the different analyses shown in section B.1. Early stopping would be used such that the training of the models which were no longer improving the MSE of the test dataset would be stopped.

The only parameters left to be determined are the number of hidden layers, as well as the number of neurons in each of the layers. For determining the value of these hyperparameters, an optimisation algorithm is used. This will be discussed in section 5.2.

5.2 Genetic Algorithms for Hyperparameter Tuning

In this section, genetic algorithms are investigated in their ability to optimise the hyperparameters of a machine learning model. The section is split up into the development and comparison of the algorithm to other methods, and the results of the optimisation of the hyperparameters, in subsection 5.2.1, and subsection 5.2.2 respectively.

5.2.1 Algorithm Development and Rationale

Before diving deeper into the optimisation of the hyperparameters of machine learning models with genetic algorithms, a step back is taken to discuss why there is a need for genetic algorithms in the first place.

The easiest and most brute-force method that one can think of is grid search. It creates a grid of points to evaluate in the design space and evaluates them. Afterwards, it chooses the point with the best evaluation of the cost function. One can imagine that for two input parameters with 6 values per parameter, one can have 36 combinations. If a 5-fold cross-validation is used for each hyperparameter combination, then the total number of models that need to be trained rises to 180. Therefore, it is evident that this method requires a lot of computational time.

Another method which is popular to tune hyperparameters is random search. Random search has been proven to be more efficient than grid search algorithms [11]. In the article, the authors attribute the efficiency of randomly searching the design space, to the inefficiency of grid search. Grid search spends a lot of time and resources on hyperparameters that do not contribute much towards the loss function, while too little on the parameters that do matter. Furthermore, due to the statistical independence of the points, the search algorithm can be interrupted at any time, and still form a good experiment. The downside of each point being independent of another is that the optimisation algorithm does not utilize the information that is hidden in the parameters of the optimisation. When a point with a good combination of parameters is found such that the loss function is minimised, it should indicate that the minimum is more likely to be around the values of these parameters, than other parameter combinations with worse fitness values. This principle can be leveraged using genetic algorithms.

Genetic algorithms are used for optimising the hyperparameters and are a branch of algorithms called evolutionary algorithms. From the literature study which was performed in subsection 3.3.6, genetic algorithms were suggested as excellent methods for hyperparameter tuning of machine learning models, which still required more research. A genetic algorithm, as the name suggests is inspired by biological evolution, where animals have evolved through mutation and cross-over of their genes, and this is aimed to be recreated with this optimization algorithm.

In section B.2, of Appendix B, a comparison is made between genetic algorithms, random search, and grid search algorithms. The study applies the three different algorithms towards a 2D, and 3D test function. This is done to validate the results of the studies which were referenced above, and also to develop the genetic algorithm from scratch, which is fine-tuned to optimise the hyperparameters of a neural network. As the references indicated, the genetic algorithm outperformed the random search algorithm, which in turn outperformed the grid search algorithm.

The algorithm for the genetic optimisation that is devised specifically for this application is shown in algorithm 1. It starts by randomly sampling a population in the function space. Each member of the population is described by its genes, with the genes in this case being the hyperparameters of the model. The size of the population of a genetic algorithm is a parameter which needs to be chosen carefully. Larger populations make

```
Initialization: Randomly sample a population of size N
for generation i in total number of generations do
    Calculate the fitness of each member of the population;
    Select the fittest individuals for reproduction (tournament selection);
    while selected indices < population size do
        Choose K random indices out of the population;
        From these K, choose the one with the best fitness function;
    end
    Add the fittest X% of the population to the next generation;
    for Individual in Fittest (1-X)% of population do
        Perform cross-over of 2 fit parents;
        if random standard normal number < mutation rate then
            Perform random mutations on their offspring;
        else
            continue;
        end
    end
end
```

Algorithm 1: Genetic Optimization Algorithm.

it easier to find a global optimum while increasing the computational cost significantly.

Then for each generation in the total number of generations, there are three steps. Firstly, the fittest individuals are selected from the population through tournament selection. This works by selecting K random individuals from the population and selecting the fittest one from this group. This is then repeated until the entire population is replaced with the individuals who have won the tournaments. The parameter K needs to be carefully selected, as choosing a too large part of the population to perform the tournament selection results in the chance of having all the same individuals in the next population too high. When K is too low, the tournament selection resembles random sampling.

Automatically the fittest 25 to 50% of the population are added to the next generation. In the context of genetic algorithms, this is called elitism, where the fittest individuals are transferred over towards the next generation. This is not only positive that the best individuals will be present in the next generation but also limits the number of function evaluations that need to be performed from generation to generation. The next step is to perform a cross-over of the fittest individuals. Cross-over is aimed to simulate the reproduction process, where the genes of the parents are fused and create a child with mixed genes. In the case of hyperparameter optimisation, it is the goal of cross-over that when the parameters or part of the parameters of the fittest individuals are crossed over, they generate an even fitter individual. In the example of having two hyperparameters

as the genetic code of an individual, the cross-over is simply performed between the first gene and the second. When there are more than two hyperparameters, the choice can also be made to mix the first gene of one parent, with the two last genes of the second parent. While other combinations are also feasible, like mixing the middle gene of the first parent, with the first and last of the second. In the algorithm proposed in this thesis, two parents always generate two children, to maintain the same number of the population. The second child is always the complement of the first. For example, if the genes of the first parent are 'A1, B1', and the second parent 'A2, B2', then child one would be 'A1, B2', while the second child is 'A2, B1'. In the very specific example of having a parent with for example neurons per layer [10, 20, 30], and [50, 60, 70, 80, 90], the cross-over is performed such that the split boundary is randomly chosen for parent 1 and 2, therefore one could use for example the first layer and combine it with the last 4 layers of the second parent. The last 2 layers of parent 1 are then combined with the first layer of the second parent.

Finally, for each child which has been created, a mutation is applied only when the random number that is generated from a standard normal distribution is smaller than the mutation rate. The mutation rate is a parameter of the genetic algorithm which determines how frequently or with what probability a child is given a genetic mutation to their genes. A high mutation rate will make it highly probable that a random mutation towards one or more of the genes is made. When a child is to be mutated, the choice has been made in designing the algorithm for this thesis that the gene on which the mutation is performed is randomly chosen. This is in contrast to completely providing a new set of parameters, as this can lead to too much exploration. With only performing a mutation to one of the genes, at least one of the parameters which constituted a good individual is maintained.

5.2.2 Results for Hyperparameter Optimisation using Genetic Algorithms

In this subsection, the hyperparameter optimisation of a neural network is performed. The specifications of the neural network that is being optimised can be found in Table 5.1. These hyperparameters were determined to be optimal in section 5.1.

What can be seen from the table is that the model is being trained on the full aerodynamic dataset, denoted as dataset 1 in section 2.2, having 8 input parameters but only 1 output parameter. During this optimisation, the number of layers of the network, and the number of neurons per layer of the network is optimised. This is done using the genetic optimisation method described in subsection 5.2.1.

In Table 5.2, the specifications of the genetic algorithm used to optimise for the number of layers and number of neurons in each layer are shown. Since the resources to optimise

Table 5.1: Specifications of the artificial neural network being optimised by a genetic algorithm.

Data and Model Parameter	Value
Dataset Number	1
Batch Normalisation	No
Dropout (same for each layer)	0.5
Activation Function	ReLU
Batch Size	128
Learning Rate	1e-4
Optimizer	ADAM
Loss Function	MSE
Epochs	300
Number of Layers	Optimised
Number of Neurons per Layer	Optimised
Number of Folds	5

the hyperparameters of the genetic optimisation algorithm itself were not available, the choice of the specifications of the genetic algorithm was based on the observations that were made in subsection B.2.1 and subsection B.2.2, for which the hyperparameters were from a well-known reference on genetic optimisation [76]. Therefore, to keep the computational cost low, the population size is taken to be 50, and the number of generations is taken as 6. Furthermore, the mutation rate is placed at 0.2. Meaning that on average there is a 20% chance of a child having a random mutation. Finally, the tournament size is taken to be 20% of the population size, to have a good balance between having a tournament that can include high-fitness individuals, and not too large that the tournament is won by the same individual most of the time. Furthermore, only 25% elitism is used since the algorithm should perform sufficient exploration.

Table 5.2: Specification of the Genetic optimisation algorithm.

	Value
Population Size	50
Number of Generations	6
Mutation Rate	0.2
Tournament Size	6
Min/Max Number of Layers	4-8
Min/Max Number of Neurons	8-1024

In Figure 5.1, the mean square error of all the points in a particular generation is shown. One can see that as the generations progress, the algorithm converges towards a particular value of several layers and several neurons. One can see that the best MSE value that is found in the first generation is much higher than the best MSE in the final generation, proving that the algorithm made better combinations of the parameters to

find the optimum. The best value coming from the optimisation is 5 hidden layers, with corresponding hidden neurons of 872, 866, 1016, 982, and 407 respectively.

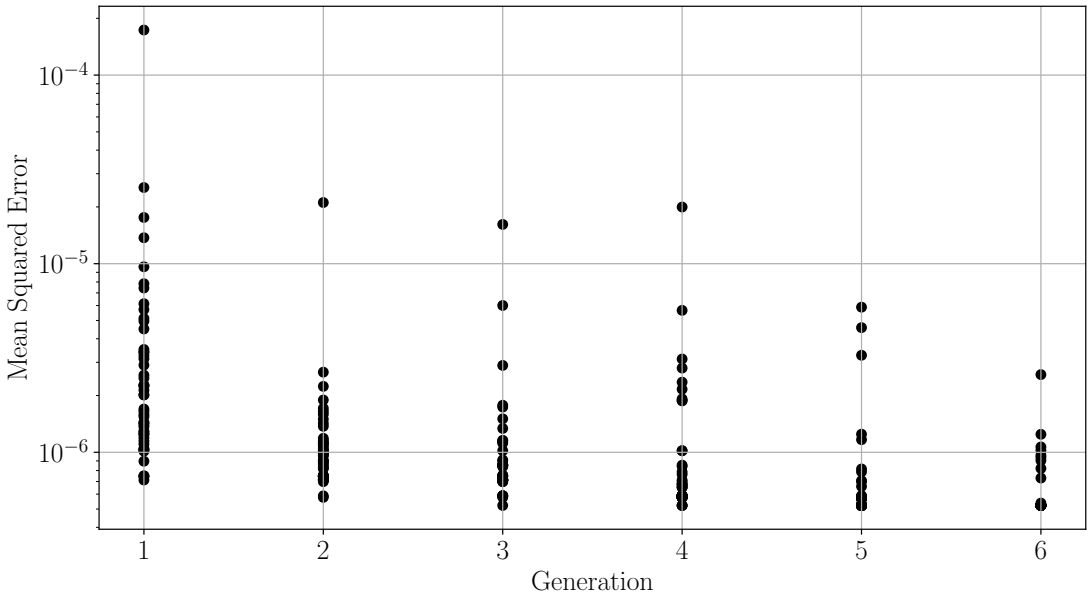


Figure 5.1: Genetic Optimisation of the number of hidden layers, and number of neurons per layer of an artificial neural network. Convergence of the mean squared error for each member of the population over the different generations of the optimisation.

Something that must also be discussed is the total training time of the algorithm. With a population of 50 and 6 generations of the genetic algorithm, a total of 300 runs should be performed. However, when one considers that due to the elitism, the number of unique runs is only 137, the actual number of models that need to be trained is far lower. With an average training time of around 40 minutes per model run on dataset 1 (section 2.2), the training time would be 19 days. Therefore, some additional steps must be taken to reduce the total training time of genetic optimisation. Firstly, this training time of 40 minutes is the time needed to complete all 300 epochs of the model. However, using early stopping the models usually are stopped before the halfway mark. Meaning the time can be reduced by half using early stopping. Next, the biggest reduction that can be made is removing the 5-fold validation split. The 5-fold split is not desirable to be removed, since then the optimiser is finding an optimal architecture for modelling a specific dataset, and one would rather want it to find the optimal architecture for modelling the entire aerodynamic dataset. Another option is that the first population of 50, which are all unique could be run in parallel on multiple computers. In this manner, especially when the optimisation is performed over the weekend, the computations can be distributed over 5 workstations. Meaning that the training time would be cut by five-fold for each generation. These two updates reduce the training time of the genetic

optimisation to 2 days.

Now that the number of hidden layers, and neurons of the hidden layers have been determined, one can use the other hyperparameters shown in Table 5.1 to create the ANNs for each of the aerodynamic coefficients. The results of the trained models for each of the aerodynamic coefficients are shown in section 5.3.

5.3 Results

In this section, the results of the optimised artificial neural networks that have been run are presented. This will be done in terms of the performance metrics mean squared error (MSE), mean absolute percentage error (MAPE) and percentage of the predictions of the test dataset which fall within the tolerances (PWT).

In Table 5.3, the hyperparameters of the optimised neural network are provided. These are the hyperparameters that were used in section 5.2, during the genetic optimisation of the hyperparameters. The hyperparameters are kept constant when applying the model towards different output parameters such as the pitching, rolling, yawing moments, or side force coefficients. This is because the pitching moment coefficient C_m is the most nonlinear to model and it is assumed that the parameters which allow the network to model the pitching moment will also be capable of modelling the other coefficients. With dropout and early stopping it is made sure that no overfitting of the model on the other coefficients occurs, as these coefficients likely require a simpler architecture.

Seven different models are run, one for each aerodynamic coefficient of the aerodynamic model, and one model which combines and predicts all of these at once. The single-output model results are shown in subsection 5.3.1, while the results of the multiple-output model are shown in subsection 5.3.2. The convergence patterns for all the models can be found in Appendix B, section B.3.

5.3.1 Single-Output Models

In Table 5.4, a table is shown with a summary of the MSE, MAPE, and PWT of all the single-output models. The MAPE calculates the difference of the model predictions w.r.t. the true value as a percentage deviation from the true value. Generally, values of under 5% are considered to be excellent, which all the models are under. The MAPE is useful as it is invariant to the scale of the output value. This will become important as the pitching moment coefficient, even though it is unitless, operates at a different scale to for example the side force coefficient in z-direction.

Table 5.3: Specifications of optimised artificial neural network.

	Value
Number of Inputs	8
Number of Outputs	1
Number of Training Points	353 000
Number of Training Polars	1241
Number of Testing Points	85 000
Number of Training Polars	292
Batch Normalisation	No
Dropout (same for each layer)	0.5
Activation Function	ReLU
Batch Size	128
Learning Rate	1e-4
Optimizer	ADAM
Loss Function	MSE
Epochs	300
Number of Layers	5
Number of Neurons per Layer	872, 866, 1016, 982, 407

In terms of the MAPE, the side force coefficient in the z-direction is predicted the best, followed by the side force coefficient in the x-direction, and finally the pitching moment coefficient. However, possibly the most useful metric in determining how well a coefficient is predicted by the neural network is the percentage of the coefficients that are within the tolerance (PWT) of the test data. The model has around 95% of its predictions within the tolerances of the test data for all the aerodynamic coefficients. Meaning that the test dataset is extremely well predicted.

One can see why one needs to use additional metrics besides the MSE when looking at the prediction of C_z. When one would only look at the MSE one would think that the C_z prediction is the worst. However, it is one of the easiest coefficients to predict since it varies smoothly, and is not as nonlinear as the other coefficients. In Figure B.32, Figure B.33, two example plots are provided with predictions. These plots are randomly selected from the test dataset. One can see that although the dataset varies at the high values for the angle of attack, the general form of polars is very similar. It is due to the larger magnitude of the C_z compared to the other coefficients, that its MSE is so high. However, when looking at the MAPE and PWT the performance is best out of all the coefficients. Therefore, the MSE is useful to compare different models on the same aerodynamic coefficients, while the MAPE and PWT can be used for the comparison between different coefficients.

Table 5.4: Summary of the MSE, MAPE, and percentage of the predictions that fall within the tolerances of the test dataset (PWT) for a single-output model for each aerodynamic coefficient.

Coefficient	MSE Test	MAPE	PWT [%]
C_m	6.105e-06	0.31	94.9
C_l	5.724e-06	4.13	95.3
C_n	2.759e-06	0.82	98.1
C_x	9.223e-06	0.13	96.0
C_y	6.738e-06	0.82	99.6
C_z	6.544e-05	0.04	99.6

5.3.2 Multiple-Output Models

In this subsection, the multiple-output models are discussed for the artificial neural network.

In Table 5.5, one can look at the relation between the single-output models and the model combining all different inputs. When considering the average MSE of the single-output models, and comparing it towards the overall MSE of the multiple-output model, the multiple-output model is only 2.5% worse in MSE. For the MAPE the mean of all the single-output models is 1.04 compared to 2.53 for the multiple-output model. Meaning that for the MAPE, there is a significant difference in having the single-output models, compared to the multiple-output model. In Table 5.5, the prediction of the single- and multiple-output models for each of the aerodynamic coefficients are provided. From this, it can be seen that the multiple-output model performs much worse for each coefficient than the multiple-output model, by at least 10%. Only the C_z is predicted better by the multiple-output model, which due to the larger scale of the C_z means the total MSE is only 2.5% worse. The average percentage difference is 24% between the multiple-output model and the single-output model, but if the mean MSE of the single-output model is compared towards the MSE of the multiple-output model, this is only 2.5%. The conclusion of Table 5.5, the single-output models are far more capable of predicting the aerodynamic coefficients, C_m, C_l, C_n, C_x, and C_y, while C_z is more capable of being predicted by the multiple-output model.

Finally, in Table 5.6, the percentage within tolerance for the multiple-output model is shown and compared to the single-output model. The multiple-output model can be seen to have worse performance than the individual models, with a reduction in the percentage within a tolerance of around 1% for each of the coefficients, except the C_x where it is almost 4%. While the PWT for the C_z of the multiple-output model is only 0.2% higher than the single-output model.

To conclude this section, one can look at 2 random polars that are selected from the

Table 5.5: MSE difference between the single-output models for the aerodynamic coefficients, compared to the multiple-output model's prediction.

Coefficient	MSE Single	MSE Multi	Percentage Difference [%]
C_m	6.105e-06	7.132e-06	+16.8
C_l	5.724e-06	6.303e-06	+10.1
C_n	2.759e-06	3.554e-06	+28.8
C_x	9.223e-06	1.631e-06	+77.8
C_y	6.738e-06	8.087e-06	+20.0
C_z	6.544e-05	6.029e-06	-7.9
Mean	1.462e-05	1.499e-05	

Table 5.6: Percentage within tolerance (PWT) comparison for the single- and multiple-output models of the artificial neural network on all aerodynamic coefficients.

	ANN Single [%]	ANN Multi [%]
C_m	94.9	93.5
C_l	95.3	94.0
C_n	98.1	97.4
C_x	96.0	91.9
C_y	99.6	99.3
C_z	99.6	99.8

dataset for each of the 6 coefficients, and for the multiple-output model in Appendix B, section B.4. Each of the plots is not shown in the chapter itself as the MSE, MAPE, and percentage inside the tolerances provide a deep enough understanding at this point of how well the model approximates the aerodynamic dataset.

5.4 Transfer Learning

In this section, a deeper look is taken towards transfer learning. A concept which, from the literature study performed in subsection 3.3.5, can decrease the total training time of a machine learning model, and possibly increase its performance.

Transfer learning (TL) is the concept of taking a model which has been trained on one dataset, and using the learning from the one dataset and using it to kickstart the training on the next dataset. The idea is that the more general features which are learned from one dataset can be applied to a different dataset. The advantages could therefore include shorter training times, better performance, and better generalization [85]. An important note that needs to be made here is that dataset 1 is used from section 2.2, and compared against dataset 2. As explained in section 2.2, the difference between datasets 1 and 2

is the empty versus full payload underneath the aircraft.

Some research questions that are aimed to be answered in this section are:

- How many layers need to be added or retrained to obtain the best MSE result?
- Does using a model with transfer learning reduce the training time? By how much?
- Does a transfer learned model reduce the MSE compared to a model trained from scratch?

In Figure 5.2, two images are provided to explain the need for transfer learning. In the top and bottom figures, subsonic and supersonic test data are provided. Two neural networks are trained, one using data from a baseline store configuration (empty) from dataset 1, the other with a full weapons armament, from dataset 2. An example scenario where the FCS needs to adjust the aerodynamic behaviour based on the store configuration is as follows. When the aircraft takes off with a full armament to perform a mission, and after the mission is performed and the payload has been delivered, the weapon bays will be empty. The most obvious change can be thought of in the drag of the aircraft since the cross-sectional area of the aircraft is significantly reduced. However, in this neural network, a random aerodynamic coefficient is considered as an output, and the effect of having a different store configuration is exemplified by the two plots shown. The model which is trained on the empty configuration in red, can provide a good prediction of the test data of the empty configuration. The same cannot be said about the model trained on data from the full configuration, which has different aerodynamic characteristics. Also in Figure 5.2, a prediction of a model which has been transfer learned to fit the data of the empty aircraft configuration is provided. Which can be seen to fit the test data at least as well as the model which was specifically trained on this dataset.

In Figure 5.3, one can see the average convergence pattern of 5-fold cross-validation for the empty data model, full payload data model, and finally, the transfer learned model from empty data towards the full. One can see that the convergences are all very similar. However, the rate of convergence of the TL model is much faster. Therefore, the initial hypothesis seems to be correct, transfer learning does increase the rate of convergence of a model, and speeds up the training.

When one returns to the research questions asked at the beginning of this section, the first question considered was: *How many layers need to be added or retrained to obtain the best MSE result?* When one considers Table 5.7, many different combinations of added layers and retraining layers are provided. One can see that not many layers have to be added or retrained to obtain an improved MSE. Therefore, what can be concluded from this is that the neural network can learn the general trends of an aerodynamic dataset and that a single layer can fine-tune the general trends in a dataset, towards a

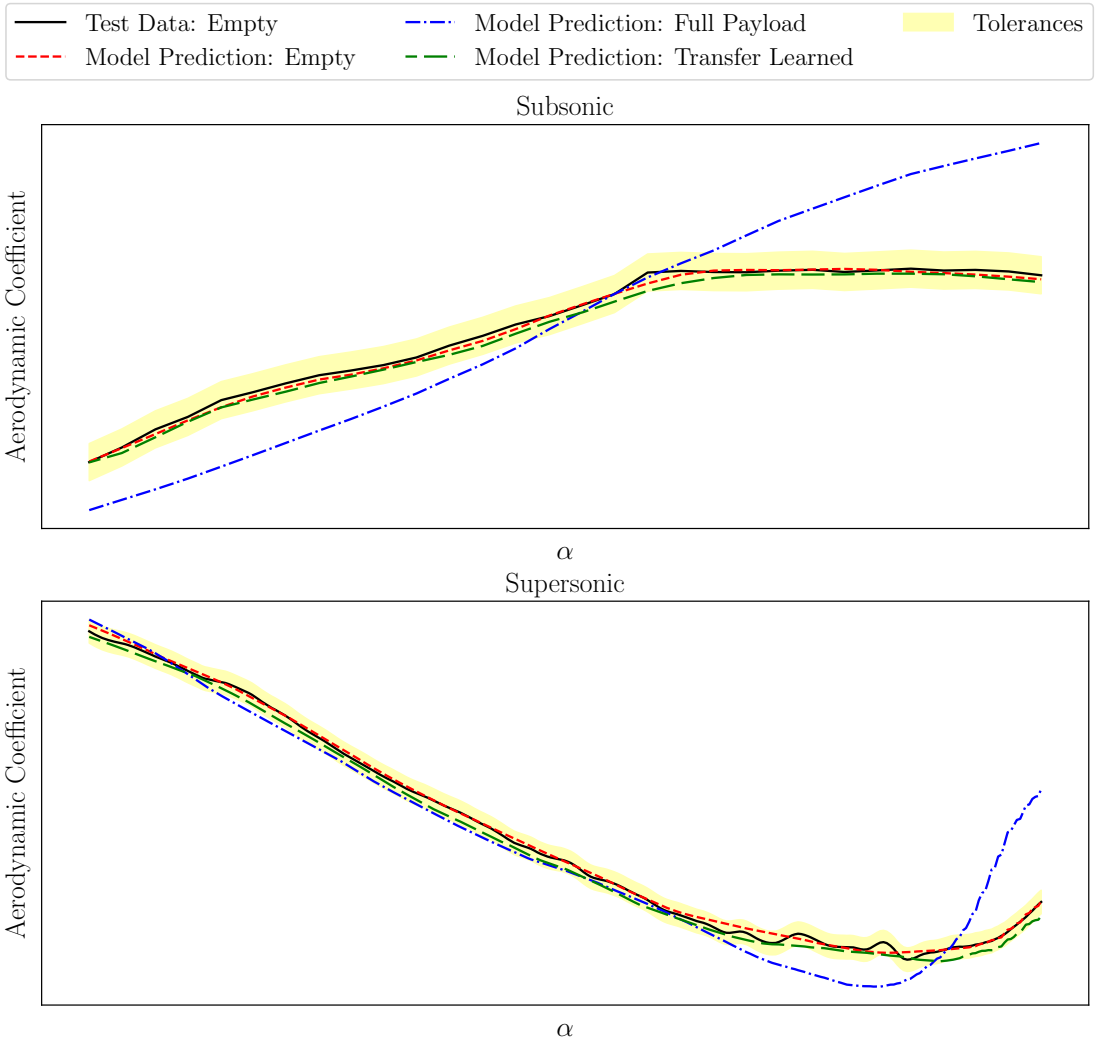


Figure 5.2: Example test data prediction by an artificial neural network for the empty and full payload configuration, and transfer learned model of an aerodynamic coefficient at subsonic and supersonic conditions.

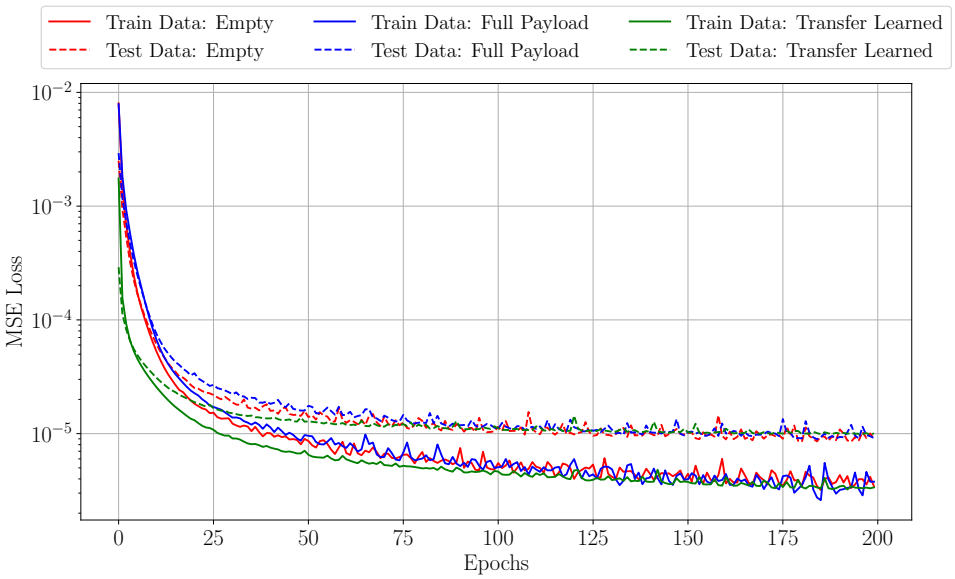


Figure 5.3: Convergence of the transfer learned model compared to the original models on the full and empty payload configuration.

more accurate model. To answer the question of how many layers need to be added to obtain the best MSE result, one could look at Table 5.7 and determine from the last column that the answer is 1 new layer and retraining 2 layers. However, when looking further one can see that the MSE of the transfer learned models are extremely close together, with the difference between the best and worst model being only 0.27e-6, or about 3% more than the minimum value. This difference can very well be because of random fluctuations in the training process, and cannot be sufficient to provide a clear conclusion. What can be concluded from this result is that likely retraining the minimum amount of layers, being only the final layer in this case, is sufficient in transfer learning a model trained on one dataset, and retraining it on a different dataset.

Secondly, when considering the question *Does using a model with transfer learning reduce the training time? By how much?*, one can see that transfer learning reduces the training times in two ways. Firstly in Figure 5.3, one can see that in the first 50 epochs especially, the slope of the MSE loss of the training and test loss is much steeper, and the training is therefore faster in reaching a minimum of the loss function. Eventually, the transfer learned model converges to the same value of the MSE loss. However, the training was still performed faster, especially when one considers using early stopping or using a threshold for training the models. In the case of Figure 5.3, the models trained on their respective datasets took 2431 and 2415 seconds, to fit a model to datasets 1 and 2 respectively. While the model using transfer learning took only 2090 seconds, a 14% difference. This is just an anecdotal comparison, and the actual difference in training

Table 5.7: Comparison of different architectures used for transfer learning. Empty, refers to dataset 1, while full refers to dataset 2, which are both found in section 2.2. TL = Transfer Learned.

	New Layers	Retrained Layers	Train MSE Loss	Test MSE Loss	Training Time [s]
Empty	-	-	3.47e-06	3.10e-05	2431
Full	-	-	3.27e-06	8.65e-06	2415
Empty TL to Full	0	1	3.26e-06	8.63e-06	2091
Empty TL to Full	0	2	2.543e-06	8.74e-06	2191
Empty TL to Full	0	3	2.67e-06	8.50e-06	2221
Empty TL to Full	1	2	2.64e-06	8.47e-06	2198
Empty TL to Full	1	1	3.15e-06	8.68e-06	2181
Empty TL to Full	2	1	3.12e-06	8.70e-06	2198

time is dependent on the total number of layers which have to be retrained. Therefore, a closer look can be taken to Table 5.7, where the training times are provided for each of the models which have been trained. The maximum training time of the transfer learning model is 8.3% for the model retraining 3 layers, and the minimum is 14% for retraining only a single layer.

Finally, the following question was asked: *Does a transfer learned model reduce the MSE compared to a model trained from scratch?*. The answer to this is answered again by looking at Figure 5.3 and Table 5.7. In Figure 5.3 one can see that the transfer learned model does not have a better MSE than the original trained models, which is confirmed by Table 5.7. For the model trained on dataset 1, the test loss MSE is higher than the one of dataset 2 since a non-optimised model is used here to illustrate the effect of transfer learning. The model that is used in this discussion only has 4 layers, with [1024, 512, 256, 128] neurons per layer. One can see that on dataset 1 the performance is lower than the one found in section 5.2. This is due to the network containing fewer layers, and thus fewer neurons. The performance on dataset 2 is much better since it is much smaller and therefore the fewer layers do not make such an impact. However, when the layers that are used to train on dataset 1, are provided with an extra layer, or retrained, the test loss becomes similar to the one seen for dataset 2. With this in mind, the conclusion can be made that the MSE of a transfer-learned model on the aerodynamic dataset of a combat aircraft does not provide better predictive performance than a model specifically trained on the target dataset.

5.5 Augmenting Artificial Neural Network Inputs with Gradient Information

In section 5.3, the results were provided for the ANN that was optimised using a genetic algorithm. In this section, efforts will be made to further improve the performance of the ANN.

The industry of military aircraft is demanding the highest standards and this means best performing aerodynamic models. Therefore, extra efforts are made to improve the performance of the ANN. In Figure 5.4, an example polar for one of the aerodynamic coefficients is provided together with the prediction of the optimised ANN. One can see that the model sometimes has issues following the strong curvature change that is present at low to medium angles of attack. The hypothesis is made that if the ANN had the information on the gradients of the test data, then it might be more capable of anticipating the sharp changes in the test data. Therefore, in this section, it is tested whether adding the gradients of the data as an input parameter to the ANN has the capability of increasing the ANN performance.

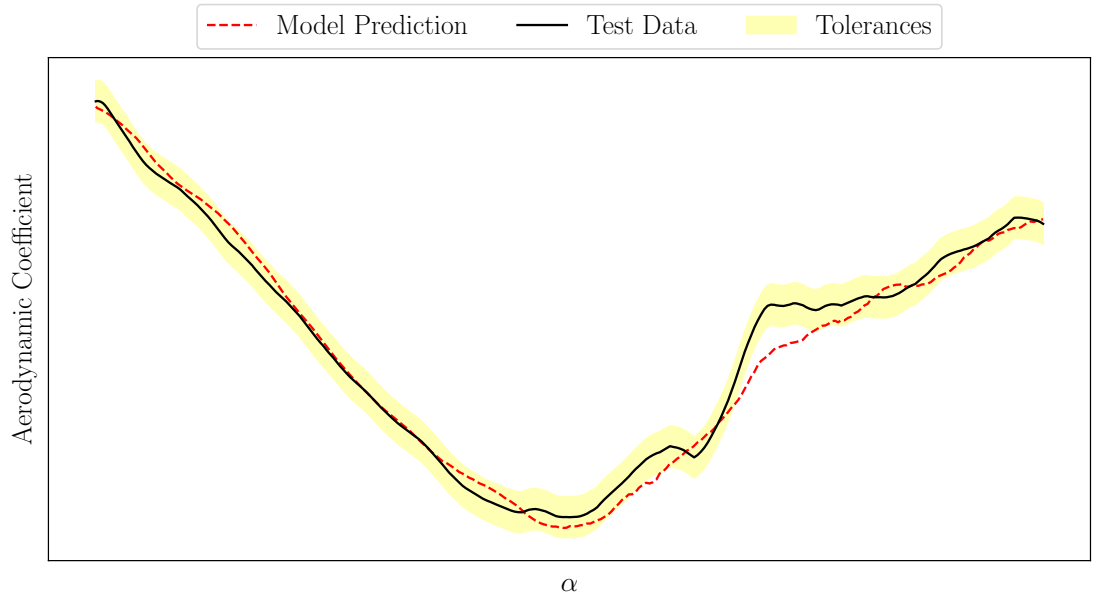


Figure 5.4: Example polar and the prediction of the optimised ANN on a test data polar.

5.5.1 Methodology

To test whether adding the gradients of the data could increase the ANN's performance, one must first devise a methodology for doing so.

Below, a summary is provided of the methodology for creating a model that uses the gradients of the training data to train a model and evaluate it.

1. Calculate the gradient of each polar along the input feature α.
2. Train a neural network which uses these gradients at each point as an additional feature.
3. For predicting a test polar:
    - Train a new model which predicts the gradients based on the input features.
    - Use the prediction of the gradient, from the known input features, to predict the aerodynamic coefficient.

Some extra notes should be made for the first point. The reason that only the derivative along the angle of attack is taken, and not along the other input features, is because the data exists as polars. Therefore, the derivatives along the other features cannot easily be calculated as, first, a model would need to be made to take a derivative at each point of the model, which is not possible if the goal of this method is to create the model itself. The gradient will be calculated using a second-order central difference scheme, with non-equal spacing between the points. As the angle of attack does not have a perfectly equal step size over the angle of attack range. The equation that is used on the central points of the polar is shown in Equation 5.1 [88]. At the edges, a standard forward or backward difference is used.

$$f = \frac{h_1^2 f(x + h_2^2) + (h_2^2 - h_1^2) f(x) - h_2^2 f(x - h_1^2)}{h_1 h_2 (h_1 + h_2)} \tag{5.1}$$

For the second point, the gradient at each data point in the direction of the angle of attack is added as a new feature, besides the original ones that can be seen in a dataset in section 2.2.

For the third point, an important note needs to be made. Although the derivative is taken for the entire dataset, to evaluate the model, one needs to assume that the gradient is not available. When a new data point is aimed to be predicted, one does not know the gradients for it. Therefore, when a prediction is made, the gradient will also need to be predicted to use it as an input feature towards the model. A separate model will

need to be created that predicts the gradient of the aerodynamic coefficient along the angle of attack, based on the input features seen in section 2.2. This model could be any model. However, since this model does not need to be smooth, and needs to predict very nonlinear gradients of the data, the CatBoost model from chapter 4 is chosen. It is very fast to train, does not require many hyperparameters to be tuned, and can therefore be the perfect model to test the hypothesis.

5.5.2 Results

To test the hypothesis of whether calculating the gradient w.r.t. the angle of attack of an aerodynamic coefficient and using it as an additional feature to predict the aerodynamic coefficient itself is beneficial, two models are trained. Firstly, the baseline model without any gradient information. Secondly, a model with identical parameters except for an additional feature which contains this gradient. For this, dataset 1 will be used which has been outlined in section 2.2.

Firstly, one can consider the convergence figure that is presented in Figure 5.5. Here the two models that have been trained are compared based on their convergence pattern and convergence speed. In this figure, one can see the training loss, testing loss, and the best iteration that was found for both models. One can see that the difference in MSE on the test data is very marginal, and the convergence is not very different between the two models. Something extremely important is that for this figure the exact gradients of the test data are inserted into the model during the convergence. As for the convergence of the model, the influence of the additional model which predicts the gradients is not desired.

Next, one can look at the gradients which are predicted by the CatBoost model, as it is an important step in the methodology. Therefore, the gradient of the CatBoost model that is made, and the true gradients are compared. When the CatBoost model is trained on the gradients of the training data, and a model is created to predict the gradients for the test data, the coefficient of determination or R^2 score is -0.41, while the MAPE is 76.7%. These are not good scores and indicate that the model is bad in explaining not only the variance in the data but also providing predictions. By definition of the R^2 score, it means that using the mean value of the dataset would result in a better prediction. Even though the gradient is not well predicted, the model trained with the gradient still performs well on the test dataset. In Figure 5.6, the gradient of the aerodynamic coefficient w.r.t. the angle of attack is provided for a test polar. Also shown in Figure 5.6, is the test polar and the prediction from the ANN with and without gradient information. One can see that the difference between the two models in the top plot is not significant meaning that both are similar in their predictive performance. In the bottom plot, the prediction of the gradient by the CatBoost model is provided. One can see that the gradient of the wind tunnel data is very peaky, and not smooth.

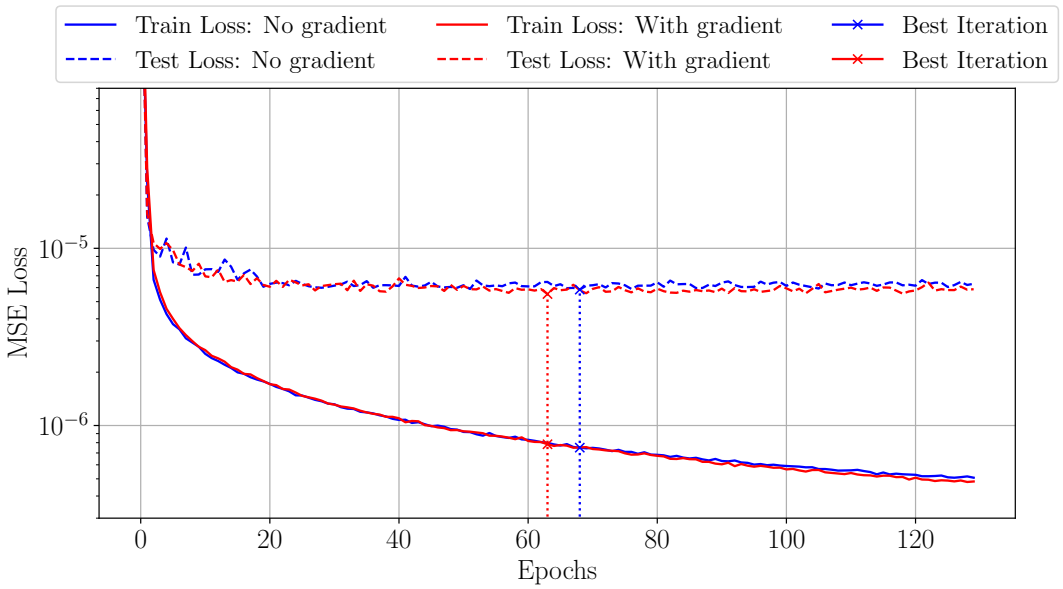


Figure 5.5: Convergence comparison of the model trained with and without the additional feature which contains the gradient calculated w.r.t. the angle of attack.

The model seems to predict general trends, such as when the gradient is positive the model also predicts a positive gradient, and so on. The gradient values shown in the plot are not the actual gradient but have also been normalised to be able to use it as an input feature in the neural network. It could be that even though the predictions of the CatBoost model of the gradients are not optimal, the very fact that it only predicts the general trends is sufficient, or even beneficial for the model predictions. To test this hypothesis more models need to be made.

In Table 5.8, the average of 10 models that are trained with gradients of the aerodynamic coefficient w.r.t. the angle of attack, and without are shown. Furthermore, a distinction is made between the models predicted with the exact gradients from the test data, and the predicted gradients using the CatBoost model. The standard model, without any gradients added as additional features, performs worse than the model trained with gradients. The difference between predicting these gradients, and using the exact ones is marginal. Still, the MSE of predicting the gradients is lower than using the exact gradients, and the PWT is higher.

From Table 5.8, one can conclude that adding the gradient information along the angle of attack as an additional parameter allows the neural network to make better predictions, which reduces the MSE, and increases the PWT compared to the standard model. However, the actual reason as to why using the predicted gradients is on par with using

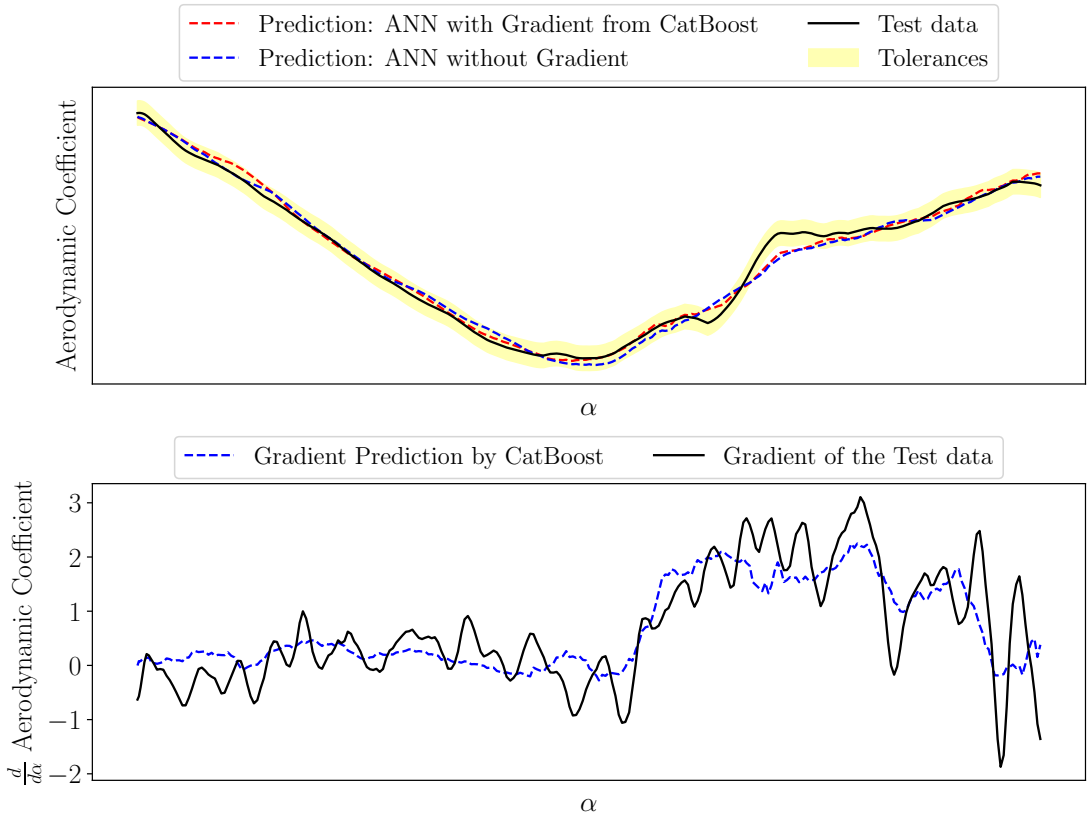


Figure 5.6: Example polar with the prediction of the optimised neural network, and the neural network with added gradient as a feature on the test data polar.

the exact gradients is not exactly known and is left up to future research into the topic. The hypothesis is that when the neural network predicts an unseen polar, it might be confusing for the model to receive such sharply changing gradients, while the predicted gradients are smoother, and provide a general trend, like positive or negative, or small or large. Therefore, the hypothesis is that the model could use these to determine the magnitude of the aerodynamic coefficients, while the sharply changing gradients still aid the model, but not as much as the predicted gradients.

Table 5.8: Comparison of the averages of 10 models trained with gradients of the aerodynamic coefficient w.r.t. the angle of attack, evaluated with the exact gradient from the test data, and the predicted gradients using the CatBoost model in terms of MSE, and percentage within tolerance (PWT) including standard deviation (STD). Compared against the average and STD of 10 standard ANNs with the parameters shown in section 5.3.

	Mean MSE	**STD MSE**	**PWT [%]**	**STD PWT**
Standard Model	6.21e-6	2.25e-7	94.6	0.0035
Exact Gradients for Evaluating	5.82e-6	1.04e-7	95.2	0.0038
Predicted Gradients from CatBoost	5.76e-6	0.82e-7	95.1	0.0038

5.6 Conclusion

In this section, a conclusion is made of the entire artificial neural network chapter.

In this chapter, the categorical hyperparameters of the artificial neural network were discussed and optimised in section 5.1. The only parameters left to be determined were the number of hidden layers, as well as the number of neurons in each of the layers. For determining the value of these hyperparameters, a genetic algorithm was developed to optimise the number of hidden layers, and the number of neurons per layer of the network in section 5.2. From this, it was determined that 5 hidden layers, with 872, 866, 1016, 982, and 407 neurons per layer respectively, were optimal. Afterwards, the results were provided for the single- and multiple-output models. After this, transfer learning and augmenting the training data using gradients was investigated. For the conclusions that are made in these final sections, one can return to the research questions shown at the beginning of this chapter.

- **Q2.1: What is the predictive performance in terms of MSE, and percentage within the tolerance of a test dataset?**
  In section 5.3, it was determined that for the optimised neural network the performance of the model was good, with the single-output models having on average over 95% of their predictions within the tolerance of the test dataset. The exact values for the MSE and PWT can be found in Table 5.4.

- **Q2.2: Does a multiple-output model outperform single-output models for each of the aerodynamic coefficients?**
  In section 5.3, the MSE and PWT are compared to their single-output counterparts. From Table 5.5, and Table 5.6, one can determine that the multiple-output model performs on average much worse in terms of MSE, and only a few percent different in terms of PWT compared to the single-output models. Only the C_z was marginally better predicted by the multiple-output model. The reason for this is due to the scale of the C_z being larger than the other aerodynamic coefficients, meaning it dominates the MSE calculation during training of the multiple-output model. This means that the backpropagation algorithm adjusts the weights of the network to predict the C_z better than the other coefficients. In the future, one could use a different metric such as the MAPE for the convergence of the model, or normalise the outputs for training. In general, the conclusion can be made that single-output models provide much better predictive performance than multiple-output models. Especially when the scale of the output values is not identical.

- **Q2.3: What are the benefits of genetic algorithms in optimising the hyperparameters of a neural network, and what is their computational cost?**
  In section 5.2, an algorithm was designed to optimise the hyperparameters of a neural network. The genetic algorithm was compared against the grid search and random search algorithm. First, it was proven that the random search algorithm is superior to grid search in most cases. Afterwards, it was shown that the genetic algorithms, with their capabilities to utilise the information, that certain parameter combinations are superior, and utilise this information to provide a better exploration, but also exploitation of the design space forms their biggest advantage compared to other techniques. Especially when optimising the number of layers, and the number of neurons in each of the layers, the genetic algorithm is highly efficient as the number of combinations that could be tested is endless. The random nature of genetic optimisation coupled with the exploitation of good parameter combinations makes it ideal for optimising the number of layers, and neurons. Furthermore, when in the future categorical hyperparameters are aimed to be optimised, the genetic algorithm has the infrastructure to optimise these hyperparameters simultaneously, providing a big advantage over other methods. In subsection 5.2.2, a methodology is provided for keeping the time needed for genetic optimisation to a minimum. This allows the genetic algorithm to utilise a 5-fold cross-validation per fitness function evaluation, 50 members of the population for

6 generations, for a training time of 2 days. This is made possible by using early stopping, making sure the training of each model is only around 20 minutes, and parallelization on different workstations.

- **Q2.4: Does transfer learning improve the MSE when training an artificial neural network, furthermore, does it reduce the training time of an artificial neural network, and by how much?**
  From the examples provided in section 5.4, one can say that a neural network trained on one store configuration is not generalisable towards another, and transfer learning or retraining is required. From section 5.4, one can also conclude that transfer learning does not necessarily decrease the MSE of the transfer learned model compared to a model purely trained on the target dataset. However, it does significantly speed up the convergence. From the simulations performed in section 5.4, a decrease between 8 and 14% in training time can be expected.

- **Q2.5: What other methods could be employed to improve the predictive performance of artificial neural networks?**
  In section 5.5, the models trained with the gradient w.r.t. the angle of attack included as an additional feature towards the neural network, provided a 7% reduction in MSE, and 0.5% increase in the PWT. Therefore, it can be said that the performance can indeed be improved by including the gradient information of the target variable for models which struggle with following the strong curvature gradients in the data. This research question is further explored in chapter 7, where stacked models are employed to increase the performance of artificial neural networks.

Chapter 6

Bayesian Neural Networks

In this chapter, Bayesian neural networks (BNNs) are investigated for their capabilities to predict and model the aerodynamic dataset of a combat aircraft. This is done in a similar method as has been done in chapter 4, and chapter 5 where first the model is provided and discussed. Afterwards, the model is optimised for its hyperparameters, and finally, the results are shown on the aerodynamic dataset for all the aerodynamic coefficients.

In the preparation of training the BNNs, two extra steps are made in the data preprocessing. Firstly, the output data is normalised to zero mean and unit variance. This step is required to utilise the Bayes-by-backprop algorithm which will be further explained in section 6.1. Secondly, the averaging of the duplicate polars, which is a preprocessing step that is outlined in subsection 2.2.1, is not performed. This is because a BNN is designed to provide the uncertainty in the model. This uncertainty is also linked to the input data of the model. Therefore, including all the data of the model could allow the BNN to learn where the data is more uncertain, and thus provide a useful evaluation of the uncertainty of the model. It is still made sure that input parameters that are used for training the model, are not found in the testing dataset. This is done to ensure a valid evaluation of the model on data that the model has never seen.

Once again the research questions which this chapter should answer are provided. The main research question is: *What is the capability of Bayesian Neural Networks to model the aerodynamic dataset of a combat aircraft and provide an uncertainty prediction?*, this research question can be divided into many subquestions which are shown below.

- Q3.1: What is the predictive performance in terms of MSE, and percentage within the tolerance of a test dataset? (section 6.3)

- Q3.2: What is the correlation between the tolerances and the uncertainties which are coming from the BNN, and does the reliability of uncertainties provide a reason for reducing the tolerances of the dataset? (section 6.4)
- Q3.3: Are Bayesian neural networks scalable towards larger datasets, and does their training time hold back their use? (section 6.3)
- Q3.4: Does a multiple-output model outperform single-output models for each of the aerodynamic coefficients, and why? (section 6.3)

This chapter first starts with the model background in section 6.1, which goes deeper into detail on which model is used specifically as the BNN. Also included in this section is a small motivation of why a heteroscedastic model is utilised to model the dataset of a combat aircraft, and the difference between aleatoric and epistemic uncertainty. Next, the hyperparameters of the model are optimised in section 6.2, after which the results for the single and multiple-output models can be provided in section 6.3. After the results are provided, the uncertainties coming from the model are validated in section 6.4. Finally, a conclusion is provided in section 6.5.

6.1 Model Background

In this section, the model and method of the BNN are elaborated on. In the literature study shown in chapter 3, section 3.4, several BNNs are explored. In short, a BNN is a neural network which, instead of having fixed weights and biases to describe the connections between neurons, has probability distributions over these parameters. Meaning that each weight and bias can be described by a probability density function. The foundation of BNNs is to view the BNN as a conditional model, parameterized by the weights and bias of the parameters. A conditional probability distribution uses Bayes' rule to calculate the posterior distribution of the model, after seeing the data. However, the intractability of the integrals involved in calculating the posterior distribution makes the BNN require special methods to compute the posterior. There are two major ways to compute this posterior distribution. Firstly, Monte Carlo sampling methods, which sample some exact weights from the posterior distribution to approximate the entire posterior distribution, which is very computationally intensive for large networks. The other method is variational inference, where an approximate posterior is created which is learnt to match the exact posterior as closely as possible. From the literature study, it was concluded that the Bayes-by-backprop using variational inference is an excellent candidate to model the aerodynamic dataset due to its scalability towards large datasets but also can provide faster results than a Markov Chain Monte Carlo method.

The training of the BNN will be detailed more closely in this section. To quickly recap, in

this method variational inference is used to approximate the true posterior distribution p over the weights of the network by a variational distribution q. The parameters θ of the variational distribution q are learnt by sampling from the variation distribution while seeing the data, to minimise the Kullback-Leibler (KL) divergence. The corresponding loss function is known as the evidence lower bound function or ELBO which has been discussed more in detail in section 3.4. To minimise the ELBO, the weights of the variational distribution are learnt using gradient descent using a method called Bayes-by-backprop. To be able to use backpropagation, the parameters need to be differentiable, which is not the case at the moment. Therefore, one requires something known as the reparametrization trick [61]. The trick is that samples, denoted as ϵ, are drawn from a simple distribution such as the standard normal distribution, after which the sampled ϵ is transformed in the parameters θ using a function f(ϵ) based on the mean and standard deviation. This function is differentiable, and can therefore be learned using gradient descent. This is summarized in Equation 6.1, where the initial parameter θ is shown to be dependent on the mean and standard deviation. ϵ is sampled from a standard normal Gaussian. Therefore, it is a random number sampled from a distribution with zero mean and unit standard deviation. Finally, the reparametrization trick is provided, where the variable ϵ is transformed into a sample from the target distribution by shifting it by μ and scaling by σ. These parameters can then be learned using gradient descent. In the model used for the thesis, an additional trick is used to avoid negative values of the standard deviations, to improve the convergence [15]. The logarithm is taken w.r.t. a variable ρ which then equals the standard deviation σ. This entire procedure is performed firstly to calculate the weights of the network and then repeated to find the biases of the network.

$$\begin{aligned} \theta &= (\mu, \sigma^2) \\ \epsilon &\sim N(0,1) \\ f(\epsilon) = \text{weights} &= \mu + \sigma \cdot \epsilon \end{aligned} \tag{6.1}$$

As was the case for the artificial neural network, batch learning is used, since the dataset is too large to be passed through the network in one pass. Furthermore, batch learning offers significant speed increases compared to direct gradient descent. Therefore, the first step is to loop over the training dataset, using a batch. The samples are then passed through the BNN, which consists of a few Bayesian layers, for a few times. The value that controls how many times the batch is passed through the network is called the number of samples being taken from the model, which corresponds to how many Monte Carlo samples are taken to estimate the expected outcome of the model, which is a hyperparameter that will be tuned in the next section. Each Bayesian layer provides the output value, which consists of the mean, and standard deviation of each output value, and the KL divergence. These are all used to calculate the total loss of the model. First, the difference between the output value of the network should be minimised w.r.t. the

actual output value. This is done using the log Gaussian loss which will be explained in the next paragraph. Also, the KL divergence should be minimised since the variational posterior q needs to match the exact posterior p as closely as possible. The equation used to calculate the KL divergence is shown in Equation 6.2 [102], which is a special version of the KL divergence between two normal distributed distributions since the priors of these distributions are normal distributions. Therefore, the KL divergence is calculating the difference between the variational distribution, and the prior distribution of the BNN.

$$D_{KL}(p||q) = \log\left(\frac{\sigma_2}{\sigma_1}\right) + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2} \tag{6.2}$$

The fit loss evaluates how close the prediction coming from the model is to the actual output of the training data. For this, a log Gaussian loss is used, which is a loss function that also takes into account the uncertainty in the predictions of the network. The fit loss shown in Equation 6.3 is a summation of the log Gaussian loss over the different number of samples that are taken to estimate the fit loss. The first term is the squared difference between the true value $\hat{y}$ and the predicted value y_i, normalised by the variance. The second term is a regularization term penalising overly uncertain predictions, multiplied by the number of dimensions of the outputs $\mathrm{Nr}_{\mathrm{dim}}$. Finally, the last term is a constant from taking the logarithm of the normal distribution and does not affect the convergence. The mean and standard deviation of the predictions y_i and σ_i respectively, are thus indirectly updated by changing the mean and standard deviations of the weights and biases of the BNN.

$$\text{Fit Loss} = \sum_{i=1}^{NumberSamples} \left(\frac{1}{2}\frac{(\hat{y} - y_i)^2}{\sigma_i^2} + \mathrm{Nr}_{\mathrm{dim}} \log(\sigma_i) + \frac{1}{2}\mathrm{Nr}_{\mathrm{dim}} \log(2\pi)\right) \tag{6.3}$$

Afterwards, the fit loss and KL loss are summed and the backpropagation algorithm takes a step to update the mean and standard deviations of the weights and biases of the network. This is then performed for all the batches, and performed until convergence of the fit loss and KL divergence is obtained.

6.1.1 Motivation for Using Heteroscedastic Model

In the dataset, the variance of the data is not equal over the entire dataset, which means that applying a homoscedastic model towards this dataset would be a limitation, as it assumes constant variance over the entire dataset. The aerodynamic dataset which is analysed also has varying variances over the different control surface deflections and flow parameters. It is known from section 2.2 that the dataset is generated from wind tunnel data, with many sources of noise and variance. The noise which is embedded in the data at high angles of attack for example, will be much larger than when the aircraft is positioned at low angles of attack in the wind tunnel. This is not only due to the wind tunnel itself, but also the unsteady effects which are more prominent at high angles of attack. Therefore, assuming homoscedasticity cannot be done for the aerodynamic dataset of a combat aircraft, and a heteroscedastic model is made for the entire dataset.

6.1.2 Aleatoric Versus Epistemic Uncertainty

In Figure 6.1 and Figure 6.2, a test dataset is shown to exemplify the relation between noise in the data and the aleatoric uncertainty of the model. One with lower random noise added to the true sinusoidal function and one with higher noise. This is done to compare the aleatoric uncertainty coming from the BNN for a dataset with low and high uncertainty.

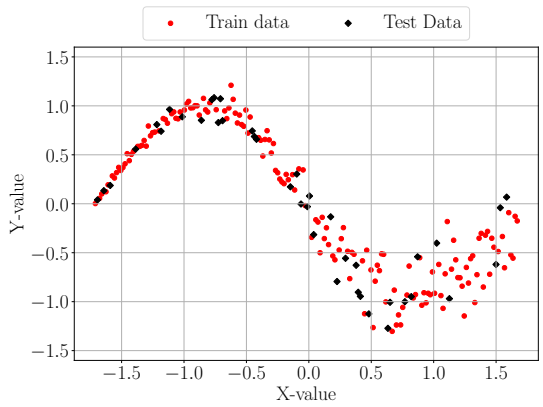


Figure 6.1: Train and test split for Bayesian neural network example with low noise.

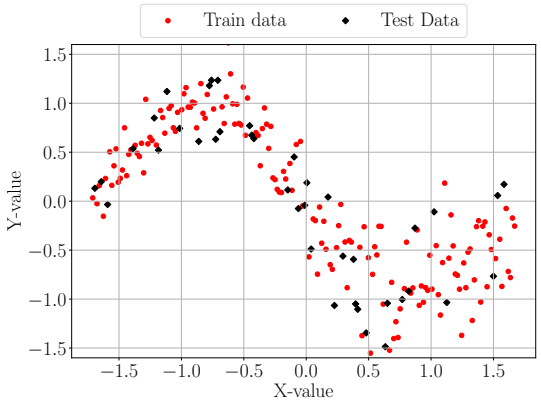


Figure 6.2: Train and test split for Bayesian neural network example with high noise.

In Figure 6.3 and Figure 6.4, the predictions coming from the models which are trained on the dataset provided in Figure 6.1 and Figure 6.2, are given together with their

uncertainties. One can see clearly from the figures that aleatoric uncertainty is larger in the model trained on the dataset with higher noise. This is because the aleatoric uncertainty is the uncertainty in the data. Meaning that when the noise in the data gets larger, the aleatoric uncertainty will increase. The convergence curves of the models seen in Figure 6.3 and Figure 6.4 can be found in Appendix C, section C.1.

The epistemic uncertainty comes from the uncertainty in the model. This can also clearly be seen in Figure 6.3. When the model provides predictions outside the range of training data, the epistemic uncertainty increases significantly while the aleatoric uncertainty remains relatively constant. This is because the model uncertainty outside the region of data is much higher. A way to conceptualise this is that the posterior distribution has seen the data after training, and therefore the sampled predictions should pass through the training data. However, outside the known training data, the model could go up, down, or any way it likes, as the model has not been fit towards any data there. Therefore, there are larger differences in the sampled model, and this increases the epistemic uncertainty.

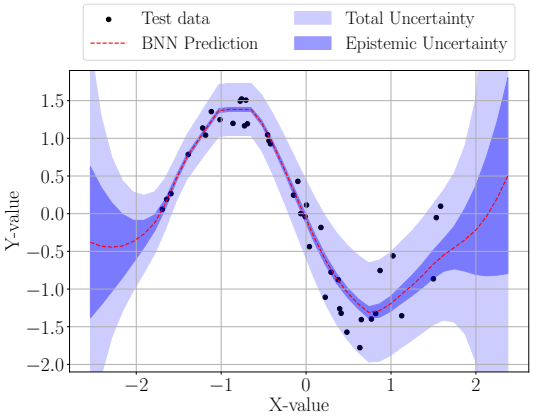


Figure 6.3: Bayesian neural network prediction and uncertainties for low noise case.

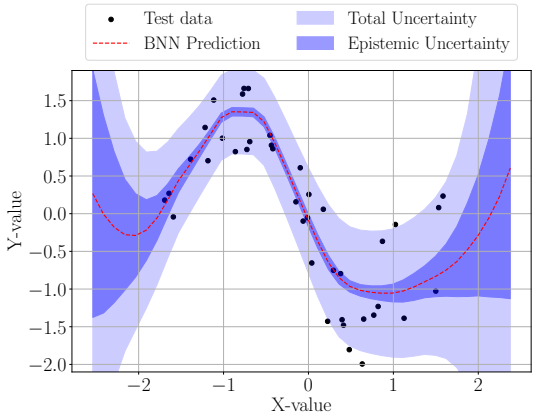


Figure 6.4: Bayesian neural network prediction and uncertainties for high noise case.

6.1.3 Calculation of Uncertainties

In this subsection, the calculation of the aleatoric and epistemic uncertainties is discussed, and their significance is highlighted.

Aleatoric Uncertainty

The aleatoric uncertainty, or uncertainty in the data, is a direct output of the BNN. The aleatoric uncertainty is the measure of the average entropy for a particular set of weights and biases [31], and is calculated using the standard deviation of the predictions of the BNN. As was explained in section 6.1, a BNN outputs the mean and standard deviation of a prediction for a given input value. The standard deviation coming from the network is directly correlated towards the variance in the data. By using the log Gaussian fit loss, the model should have learnt from the data that, when the KL divergence is low, a larger deviation from the true data is correlated towards a higher variance, and thus standard deviation coming from the model.

Therefore, the aleatoric uncertainty is calculated by obtaining the standard deviations of a given input value many times, and an average is taken. For the post-processing, 100 Monte Carlo samples are drawn. During training 25 were drawn to estimate the KL divergence and fit loss. However, during post-processing, much fewer predictions are made overall, and therefore the number of samples is not limited by the training time of the model. The reason for taking many samples from the network is that the random variables that are used inside the layers of the network make the predicted value and thus also standard deviation slightly different for each prediction. The differences in predictions with the same input values are related to the epistemic uncertainty, which is discussed below.

Epistemic Uncertainty

The epistemic uncertainty, or the model uncertainty is calculated by the prediction variability. Since the weights and biases are treated as random variables in a BNN, samples drawn from the network can have different output values for the same input value. The measure of how much the predictions differ is captured by epistemic uncertainty. The epistemic uncertainty is calculated by taking the standard deviation of the predictions of the model, over many samples, which as discussed for the aleatoric uncertainty above, is taken as 100 samples for post-processing.

6.2 Hyperparameter Selection

In this section, the hyperparameters of the BNNs for the prediction of the aerodynamic dataset are selected. The hyperparameters of a BNN are similar to the ones of an artificial neural network, with the most notable difference being the number of Monte Carlo samples used to estimate the prediction and standard deviation of a prediction. Due to the large training time of a BNN, dataset 3 shown in section 2.2 is used to optimise the hyperparameters. This is because the dataset is smaller, with fewer input parameters, allowing for faster convergence and comparison.

The training time of a BNN is such that besides choosing a smaller dataset and only optimising on the pitching moment coefficient, a simple method is employed for choosing the hyperparameters. A total of seven models are trained to find a good combination of the hyperparameters, which are shown in Table 6.1. The hyperparameters which are optimised are the number of neurons in the hidden layers, which is chosen to be a constant value for each layer. Next, the number of hidden layers, the learning rate, batch size, and finally the number of samples drawn from the network. A baseline model is trained using the parameters shown in the column for Model 1. From this baseline model, one parameter is varied for each model, after which in model 7 a combination is made of all the best parameters from the previous models. A detailed explanation is provided in section C.2, of Appendix C on what the reasoning is for each of the different models. The reasoning is based on the convergence pattern of the KL divergence loss, and fit loss, as well as examples of predicted test polars. Besides the above-mentioned measures to reduce the computational cost of optimising the hyperparameters of the BNNs, 5-fold cross-validation is not performed for each parameter combination. As BNNs are praised for their resistance to overfitting, the choice is made to not perform cross-validation to keep the training time down to a minimum.

The conclusion from section C.2 is that the BNN hyperparameters have a profound effect on the predictive performance and ability to provide the uncertainty measure. A careful trade-off between the number of layers and number of neurons needs to be found, as it has a large effect on the stability of the fit loss during training. Furthermore, more neurons are beneficial compared to more hidden layers, as can be seen when looking at models 1, 2, and 3 in Table 6.1. The number of samples drawn from the model had the largest effect on the MSE of the model. Therefore, the number of samples drawn will be increased to 25. Furthermore, the batch size which has been varied between 16384, and 4096, had a small effect on the MSE of the model, but a larger effect on the uncertainty of the model. The lower batch size showed reduced confidence in the uncertainty, but a reduced MSE. Therefore, the lower batch size is preferred. Finally, the lower learning rate increased the training time but reduced the MSE significantly. Therefore, the value of 1e-4 is taken. When these options are combined in model 7, one can see that the MSE of this particular model is the lowest out of all the models tested. Therefore, one

could conclude that the linear method, although not providing the absolute minimum, provided a better model than either of the other models tested before.

Table 6.1: Table showing the specifications of each of the models which have been created for optimising the hyperparameters of the Bayesian neural network.

	Number of Neurons	Hidden Layers	Learning Rate	Batch Size	Samples Drawn	MSE
Model 1	512	2	1e-3	16384	10	4.41e-6
Model 2	1024	2	1e-3	16384	10	3.98e-6
Model 3	1024	1	1e-3	16384	10	3.95e-6
Model 4	1024	1	1e-3	16384	25	3.51e-6
Model 5	1024	1	1e-3	4096	10	3.91e-6
Model 6	1024	1	1e-4	16384	10	3.48e-6
Model 7	1024	1	1e-4	4096	25	3.38e-6

The hyperparameters of the model trained on dataset 3 are used to model each of the aerodynamic coefficients on dataset 1 in section 6.3. Although likely not the optimum solution, the assumption is made that the hyperparameters used for training a model on dataset 3, will be sufficient and capable of training a model on dataset 1.

6.3 Results

In this section, the results of the BNN architecture that has been optimised in section 6.2 are shown. As was done for the artificial neural network, a model is created for each aerodynamic coefficient, as well as a combined model to compare their performance. Although it is known from the artificial neural network, that the performance of an individual model is far greater than a combined model, the increased training time of a BNN might require considering the training time more carefully, and hence also a multiple-output model. The values of the hyperparameters of the optimised model can be found in Table 6.2.

6.3.1 Single-Output Models

In this subsection, the single-output models that are created for each of the aerodynamic coefficients are presented. These models are made using the hyperparameters that are determined from section 6.2, and are found in Table 6.2. Due to the high computational cost of training a BNN, they cannot be optimised for each aerodynamic coefficient separately. This could result in a small loss of performance. However, the nature of the dataset means that the moment coefficient is the most difficult to predict, due to it being

Table 6.2: Specifications of optimised Bayesian neural network.

Data and Model Parameter	Value
Dataset Used	1
Batch Normalisation	No
Dropout (same for each layer)	No
Activation Function	ReLU
Batch Size	4096
Learning Rate	1e-4
Optimizer	ADAM
Epochs	50000
Number of Hidden Layers	1
Number of Neurons per Layer	1024

the most nonlinear. Therefore, it is a reasonable assumption that a model architecture that works on the most difficult aerodynamic coefficient to predict will generalise towards the other coefficients.

In Table 6.3, a summary of all the model's performance is provided in terms of MSE, MAPE, and percentage inside the tolerance (PWT) for the coefficients. What can be seen is that the performance of the BNN is high, as on average over 95% of the predictions of the test dataset are within tolerance. Furthermore, the MAPE is below 5% which means it has excellent predictions. Furthermore, the percentage within tolerance (PWT) value for each of the aerodynamic coefficients, is above 94%, with the C_n, C_y, and C_z even providing a PWT of above 97%.

Table 6.3: Summary of the MSE, MAPE, and percentage of the predictions of the Bayesian neural network that fall within the tolerances of the test dataset (PWT) for a single-output model for each aerodynamic coefficient.

Coefficient	MSE Test	MAPE	PWT [%]
$\boldsymbol{C_m}$	4.619e-06	0.209	96.6
$\boldsymbol{C_l}$	6.554e-06	1.872	94.1
$\boldsymbol{C_n}$	3.285e-06	0.768	97.3
$\boldsymbol{C_x}$	7.831e-07	0.141	96.7
$\boldsymbol{C_y}$	8.932e-06	1.842	99.0
$\boldsymbol{C_z}$	4.995e-05	0.042	100

To truly obtain a good view of the performance of the BNN and its uncertainties that are provided, one can look at Figure 6.5, where the 6 models provide a prediction of all the aerodynamic coefficients. In this plot, one can see the test data, tolerances, prediction of each of the individual models on the aerodynamic coefficients, and finally the total uncertainty and epistemic uncertainty. In the figure, one can see that all variables are well-predicted by the BNN. The function is smooth and can handle large gradients in

the dataset such as the sharp kink that can be seen for the C_l. The little oscillations that are present in the C_l, C_n, and C_y are not modelled by the model. However, if one looks at the tolerances, these could be due to the wind tunnel data containing random errors. Another interesting thing about the model is that the epistemic uncertainty for all the models is very small. The total uncertainty is often much larger and usually an order of magnitude lower than the tolerances of the test data. For example for the C_n, the uncertainty is very large in the region of high angle of attack, and one can see that indeed the prediction is less good there, likely because the data is very noisy there, and the model could not learn properly what should be done there. The heteroscedasticity of the model is in the case of C_n also clearly visible, in the middle angle of attack range the model is quite certain about its prediction while at high angles of attack, it is not.

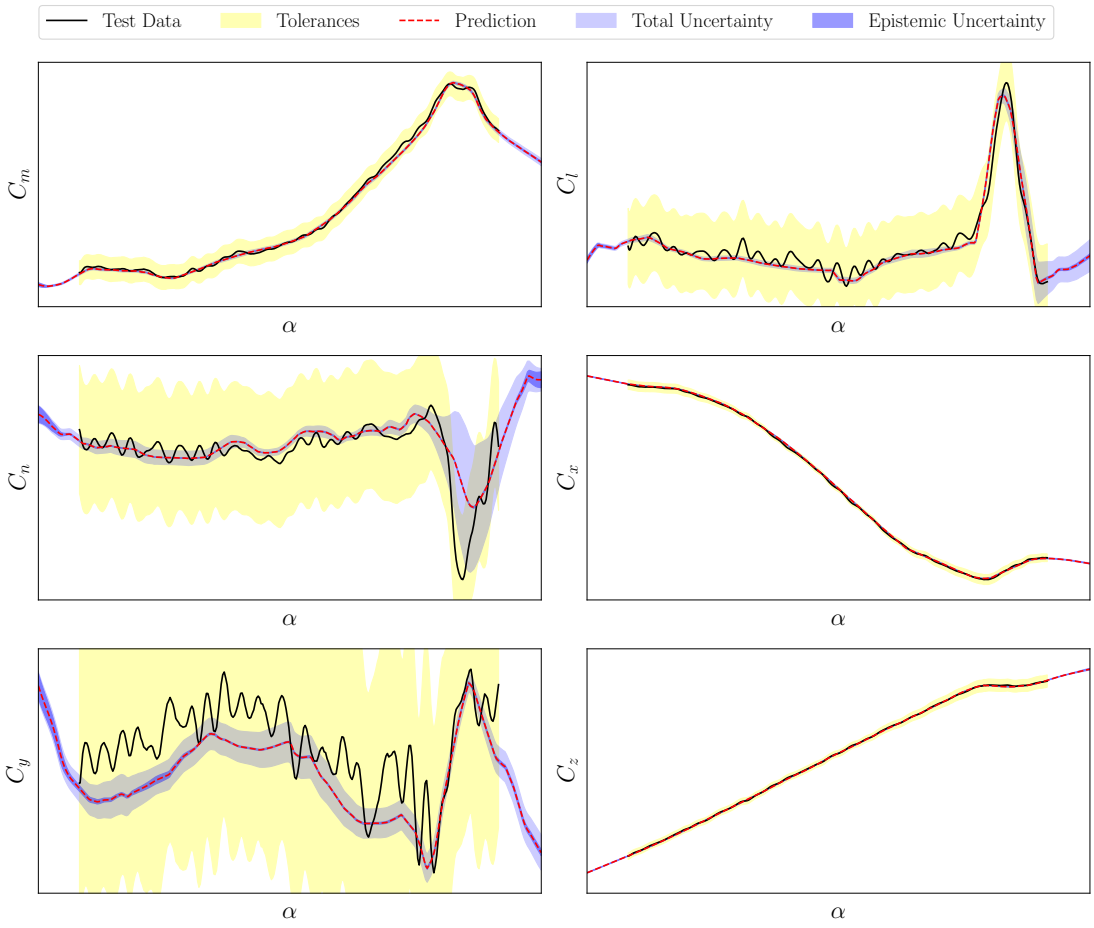


Figure 6.5: Example plot from the single-output Bayesian neural network model for each of the aerodynamic coefficients.

6.3.2 Multiple-Output Models

In this subsection, the multiple-output BNN model is shown, and its performance is discussed.

In Table 6.4 the MSE, MAPE, and percentage within tolerance for the multiple-output model are shown. The table shows the individual performance of the multiple-output model, on each of the coefficients separately. One can see that the performance is still good, with a high percentage of the predictions being inside the tolerance. However, the performance is noticeably lower than that of the single-output models for C_m, C_l, and C_x, while for the C_n, C_y, and C_z the difference is not noticeable.

The important consideration of this multiple-output model is the training time. The training time for the multiple-output model was significantly higher than the single-output model, training for 32 hours. However, it is for 6 output values, and not just 1. When compared to the single-output model which has a training time of almost 22 hours one can see that the multiple model is significantly better when compared to training the single-output models in series. However, luckily the single-output models do not need to be trained in series. Airbus Defence and Space have significant computational power, where each model can be trained in parallel, therefore to train 6 single-output models is still only 22 hours when run in parallel. While the multiple-output model is not only harder to train and to converge, but takes longer to train.

Table 6.4: Summary of the MSE, MAPE, and percentage of the predictions of the Bayesian neural network that fall within the tolerances of the test data (PWT) for a multiple-output model for each aerodynamic coefficient.

Coefficient	MSE Test	MAPE	PWT [%]
C_m	8.471e-06	0.216	91.6
C_l	7.944e-06	1.641	92.8
C_n	3.275e-06	0.563	97.7
C_x	1.537e-06	0.185	92.9
C_y	7.441e-06	1.589	99.3
C_z	1.021e-04	0.034	99.2

A final look can be taken at the predictions that are made on the aerodynamic coefficients from the multiple-output model. There one can see the predictions of all the models. As explained above the C_n, C_y, and the C_z have a better prediction from the multiple-output model, and this is also reflected in this plot, since for the C_n, the model is predicting much better, with less uncertainty and the high angle of attack behaviour. The lower aleatoric uncertainty for the multiple-output BNN compared to the single-output model is interesting as the uncertainty in the data has not changed between the two models, but the model's perception of the uncertainty has changed. This trend of lower uncertainty for the multiple-output model is carried over to the other aerodynamic coefficients which

it predicts worse than the single-output model. Therefore, it can be concluded that a multiple-output model has lower total uncertainty coming from the model, regardless of whether it performs better or worse than the single-output model. For the C_m one can see a clear difference between the single-output model, and the multiple-output model which does not predict the output as well.

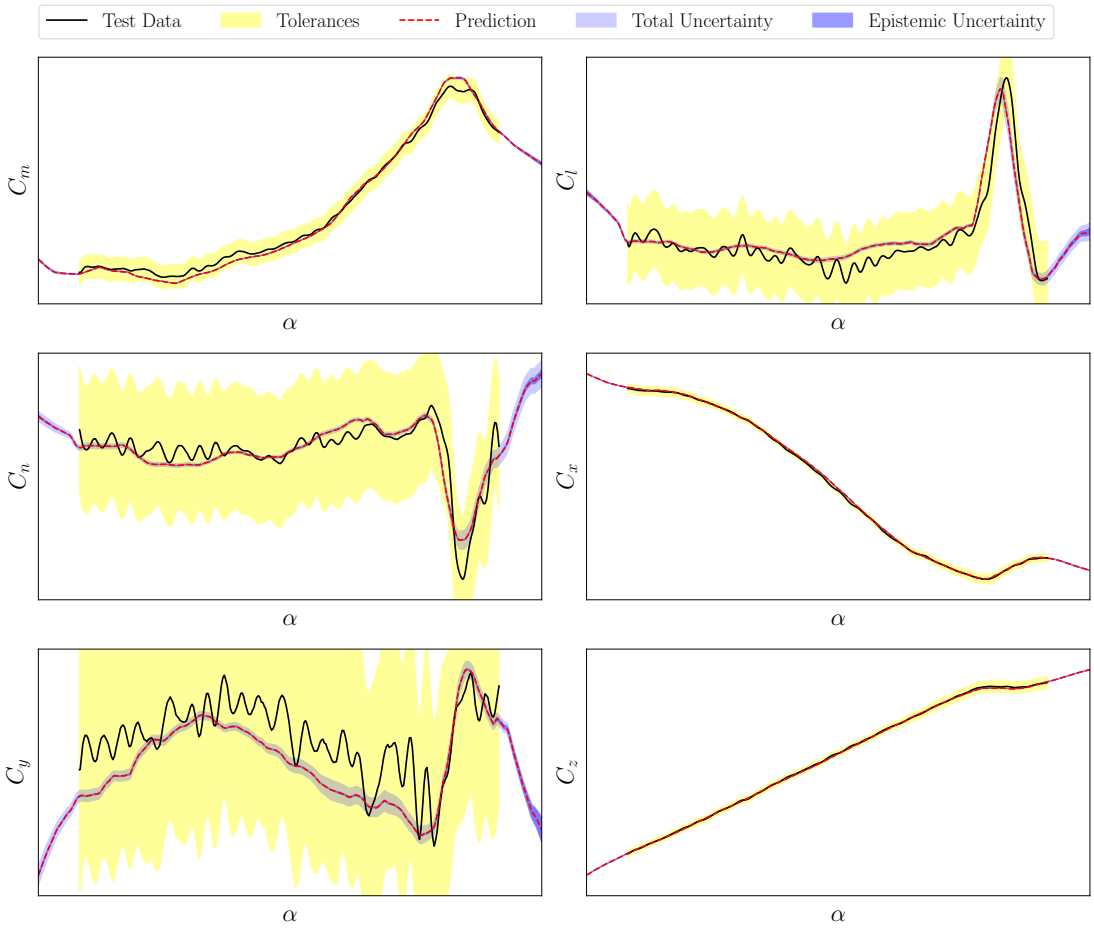


Figure 6.6: Example plot from the multiple-output Bayesian neural network model.

To conclude this section, one can deduce from the discussion above that BNNs are capable predictors of the aerodynamic dataset. The multiple-output model with its longer training time is capable of predicting the coefficients C_n, C_y, and C_z more accurately than the single-output models. However, for the most important coefficient C_m the prediction of the single-output model is significantly better, as is also the case for the C_l, and C_x. One could use the predictions of the multiple-output model for the coefficients C_n, C_y, and C_z, while C_m, C_l, and C_x could come from the single-output models. However, this comes at the cost of having to run the multiple-output model, which has an in-

creased training time. Besides the predictive performance of the BNN, the uncertainties which come from the model need to be investigated. This is done in section 6.4.

6.4 Validation of the Uncertainty Estimates

In this section, the uncertainties coming from the BNNs are validated. With validation, the goal is to quantify whether the uncertainties coming from the BNN are related towards the tolerances of the dataset. Most importantly this section aims to answer whether a larger uncertainty in the predictions of the BNN is also related towards a higher deviation from the test data.

The uncertainties coming from the model can be calculated as explained in subsection 6.1.3. Already when looking at Figure 6.5 one can see that the epistemic and aleatoric uncertainties do not encompass the test data. An example of a pitching moment coefficient polar is provided in Figure 6.7 that showcases this. However, one can see that at a low angle of attack, and in the middle range, the aleatoric uncertainty increases locally, which seems to correlate with a higher deviation from the test polar. Therefore, the hypothesis is that the total uncertainty increases when the prediction is further from the test data. To test this hypothesis a correlation plot can be made showing the uncertainty on one axis, and the deviation from the test polar on the other, which is done in Figure 6.8, and Figure 6.9.

In Figure 6.8, and Figure 6.9 the aleatoric uncertainty plotted as a function of the RMSE of a prediction of the BNN is provided for the pitching moment coefficient. What can be seen is that there is not a strong correlation for either the train or test dataset w.r.t. the aleatoric uncertainty. This is confirmed when the Pearson correlation coefficient is calculated, which gives 0.0411 for the train, and 0.226 for the test dataset. This means that both have a positive correlation, which indicates that when the RMSE of a prediction increases, the aleatoric uncertainty also increases. However, the correlation is not strong at all. For the epistemic uncertainty provided in Figure 6.10, and Figure 6.11, for the train and test data respectively, one can see similar trends. The Pearson coefficient for the epistemic uncertainty is in line with the aleatoric uncertainty, with a correlation of 0.356, and 0.231 respectively. Therefore, one can conclude that specifically for the test dataset, the aleatoric and epistemic uncertainty increase with a weak positive correlation to a prediction further away from the test data.

Unlike what was seen in Figure 6.7, where the total uncertainty remained relatively constant outside the region of the test data. In general, the total uncertainty increases drastically outside the region where the model has seen data. When one looks at Figure 6.12, and Figure 6.13, which show the aleatoric, and epistemic uncertainty over a 2-dimensional domain for a BNN trained on the pitching moment coefficient. In these

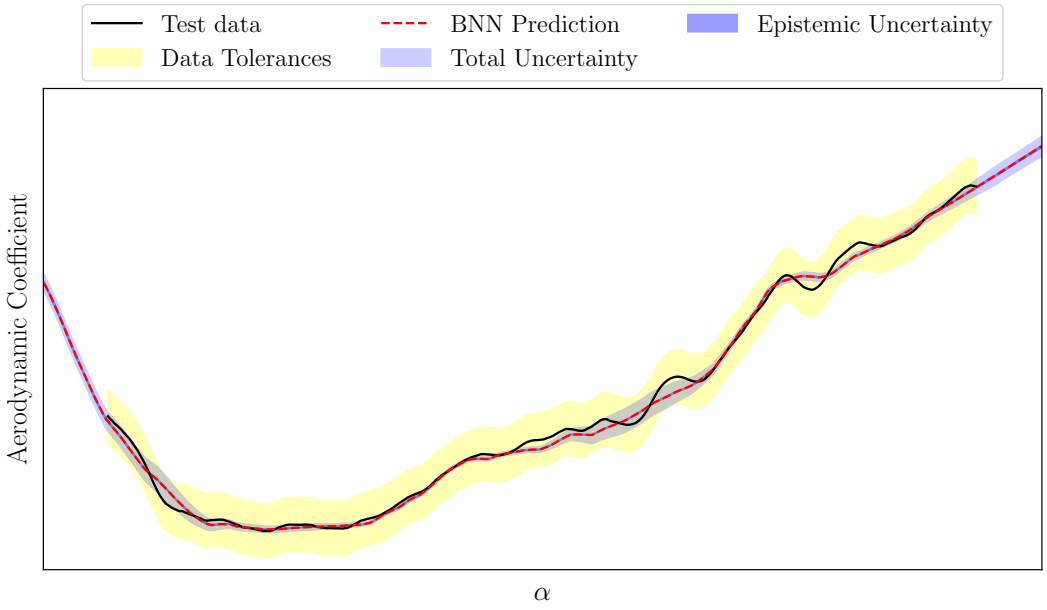


Figure 6.7: Example polar for the pitching moment showcasing the uncertainty of the prediction of the optimised Bayesian neural network.

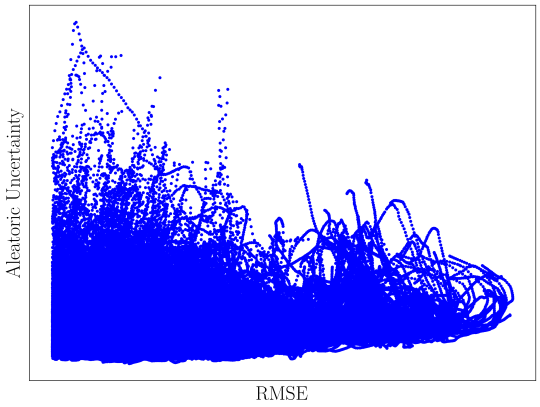


Figure 6.8: Aleatoric uncertainty plotted against the RMSE for the training data.

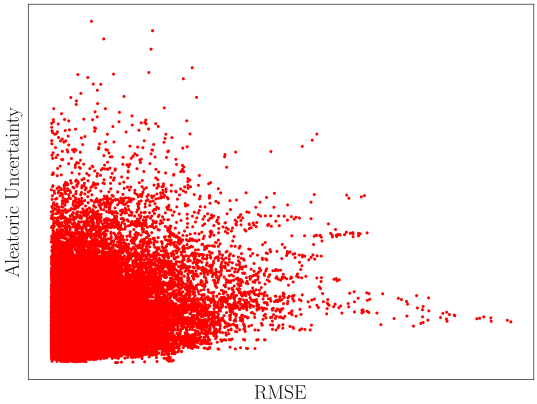


Figure 6.9: Aleatoric uncertainty plotted against the RMSE error for the test data.

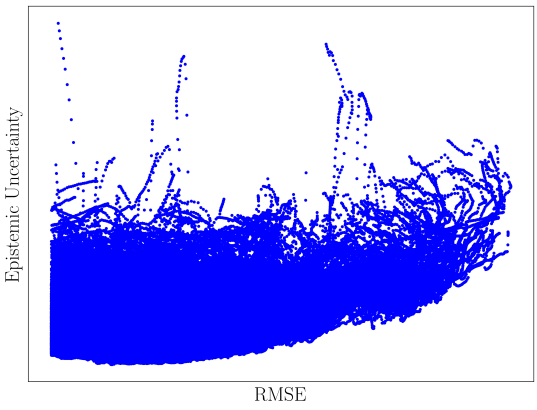


Figure 6.10: Epistemic uncertainty plotted against the RMSE for the training data.

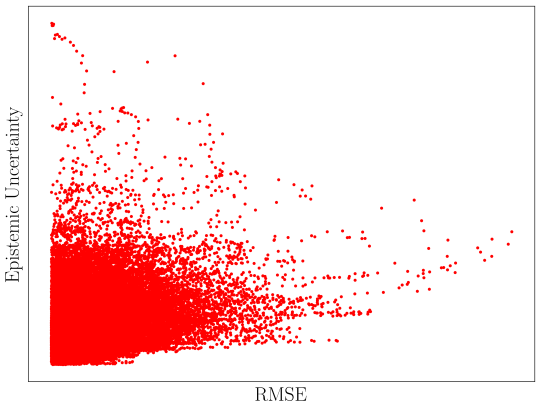


Figure 6.11: Epistemic uncertainty plotted against the RMSE error for the test data.

figures, both uncertainties have been rescaled between 0 and 1 for the confidentiality of the data. From Figure 6.12, and Figure 6.13, one can see that the epistemic uncertainty increases drastically when the angle of attack is increased beyond the training data, but this effect is also seen along the other input parameter. For the aleatoric uncertainty, this is not the case, except for the corner points of the training data domain of this specific plot, where likely the uncertainty in the data is higher.
Therefore, it can be concluded that the aleatoric uncertainty of a BNN has variations, but does not necessarily increase drastically outside the training data domain. This is because the model cannot teach itself to increase the uncertainty coming from the data (aleatoric uncertainty) if there is no data. On the other hand, the epistemic uncertainty does increase drastically outside the training data region. This is because the model uncertainty in this case is very high since the prior distribution can produce any shape of the function without being penalised during the training of the model in this region.

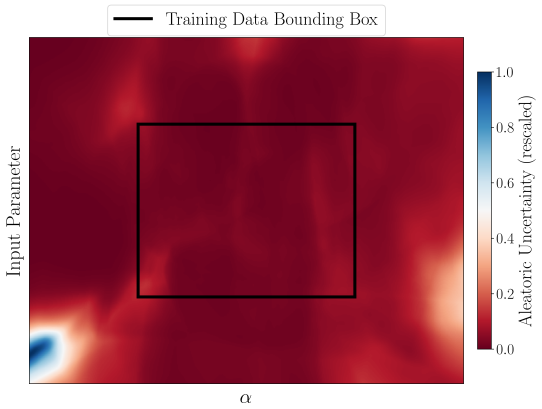


Figure 6.12: Aleatoric uncertainty coming from the BNN model as a function of angle of attack and an additional input parameter.

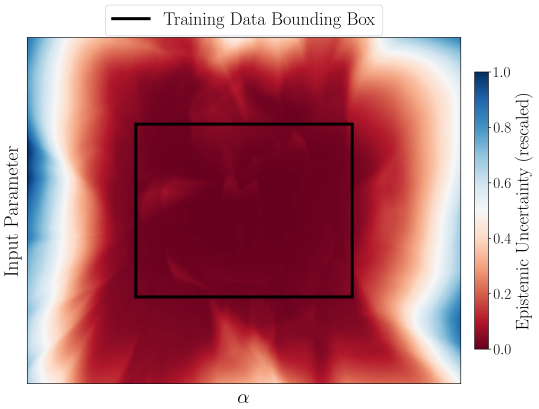


Figure 6.13: Epistemic uncertainty coming from the BNN model as a function of angle of attack and an additional input parameter.

6.5 Conclusion

In this section, a conclusion is provided on Bayesian neural networks which were applied towards the aerodynamic dataset.

In this chapter, the background of the heteroscedastic BNN model, trained with Bayes-by-backprop, was explained in detail in section 6.1. After this, the model's hyperparameters were optimised in section 6.2. Next, the results of the single-, and multiple-output models were provided in section 6.3. Finally, the uncertainties coming from the network were aimed to be validated in section 6.4.

The research questions which were posed at the beginning of this chapter will now be provided once more.

- **Q3.1: What is the predictive performance in terms of MSE, and percentage within the tolerance of a test dataset?**
  In section 6.3, the results for the MSE and percentage within the tolerance are provided for the single- and multiple-output models. What can be concluded is that the pitching moment coefficient, which is the most nonlinear and difficult to predict has a PWT of 96.6% and a very low MAPE and MSE. Meaning that the aerodynamic dataset is predicted very well by the BNN. The other aerodynamic coefficients are in line with these results, with the C_n, C_y, and C_z standing out as they have a PWT of over 97%. For the multiple-output model, one can see that the pitching moment coefficient is much less well-predicted, which is also the case for

the C_l, and C_x. For the C_n, and C_y the prediction of the multiple-output model is marginally better than the single-output model. What was further concluded was that the uncertainty coming from the multiple-output model was on average lower than the uncertainty coming from the single-output model, which is independent of the aerodynamic coefficient, and the relative predictive performance.

- **Q3.2: What is the correlation between the tolerances and the uncertainties which are coming from the BNN, and does the reliability of uncertainties provide a reason for reducing the tolerances of the dataset?**
  What could be concluded from the plots shown in the results and validation of uncertainty sections (section 6.3, and section 6.4) was that the total uncertainty coming from the Bayesian neural network was much lower than the tolerances of the test data. This means the Bayesian neural network was overconfident in the results. The other explanation could be that the tolerances are too conservative. However, this explanation is not valid since the uncertainties coming from the BNN do not encompass the test dataset.
  As was seen in the analysis of section 6.4, the RMSE of a prediction, and the aleatoric, or epistemic uncertainty are correlated. However, the correlation is low and does not constitute the conclusion of reducing the tolerances of the model to the uncertainties that are given by the BNN.

- **Q3.3: Are Bayesian neural networks scalable towards larger datasets, and does their training time hold back their use?**
  The scalability of Bayesian neural networks was extensively tested during this thesis. The network trained in this thesis used stochastic Bayes-by-backprop. This method allowed the model to learn a very large training dataset of over 350 thousand points. The dataset of the combat aircraft was larger than the largest regression dataset that could be found in the literature (section 3.4). The single-output models trained on average around 22 hours on the workstations described in subsection 2.2.4. Therefore, when even larger datasets are used, the training time will increase further, but the architecture will be scalable towards even larger datasets by adjusting the batch size. Although the computational cost of BNNs is much higher than ANNs and tree-based models (by a factor of around 30, and 80,000 respectively), the performance of BNNs is such that when the goal is to create the best-performing regression models, which are naturally resistant to overfitting, and provide uncertainty measures, BNNs form a unique and worthwhile investment.

- **Q3.4: Does a multiple-output model outperform single-output models for each of the aerodynamic coefficients?**
  As discussed in the first research question, the multiple-output model marginally outperforms the single-output model for the C_n, and C_y, and performs very well on the C_z. However, for the C_m, C_l, and C_x the predictions are noticeably worse. The difference in PWT between the single-, and multiple-output models are 5%, 1.3%, and 3.8% respectively. The multiple-output model predicts the C_n, C_y, and C_z well, and performs worse for the other coefficients. Since the behaviour of a

multiple-output model predicting the C_n, C_y, and C_z is also seen for the ANN multiple-output model, one could determine that these coefficients are easier to predict, not only for single-output but also for multiple-output models. Therefore, the chance of having a prediction from the multiple-output models, which is slightly better than the single-output for these coefficients is greater, but does not provide evidence of the superiority of multiple-output models.

Chapter 7

Comparison and Stacked Models

In this chapter, a comparison is made between the tree-based models from chapter 4, the artificial neural networks (ANNs) from chapter 5, and finally the Bayesian neural networks (BNN) from chapter 6. This is done in section 7.1. After the comparison is made, and the strengths and weaknesses of each of the models are discussed, a stacked model is attempted to be made, to further improve the performance of the models in section 7.2.

In this chapter, research question 4 is the main research question that is aimed to be investigated. The research question is: *What is the increase in performance when creating a stacked model between any of the previously mentioned machine learning models?* This can be further split into multiple subquestions shown below:

- Q4.1: How do the predictive performances of tree-based, ANNs, and BNNs compare in terms of MSE and percentage within tolerance? (section 7.1)
- Q4.2: How do the training times of the different models compare? (section 7.1)
- Q4.3: Which meta-model provides the highest performance increase for the stacked model? (section 7.2)
- Q4.4: What extra computational cost is attributed towards the application of a stacked model? (section 7.2)

Furthermore, an additional question has been raised in chapter 4 on tree-based models. In chapter 4, it was observed that the Random Forest, XGBoost, and lightGBM models

showed a consistent drop in performance for the C_m, and C_x, while the models were predicting the remaining aerodynamic coefficients well. Furthermore, the CatBoost model did not have this dip in performance. Therefore, analysing whether this phenomenon is also present in the ANN and BNN can provide evidence of this being a model-dependent issue.

7.1 Comparison of the Models

In this section the models which have been studied up until now in chapter 4, chapter 5, and chapter 6, are provided and compared against each other. Among the considered models are the Random Forest, XGBoost, LightGBM, CatBoost, artificial neural network, and finally the Bayesian neural network. For each of them, their performance on the aerodynamic coefficients will be compared.

Firstly, about the dataset that is used to compare all the models. The dataset is the same for all the models to provide a fair comparison between them and is equal to dataset 1 which has been shown in section 2.2, and has been used in each of the respective chapters about the respective models. There are 8 inputs, a single or 6 outputs, 1241 training polars (352905 points), and 154 testing polars (43730 points). Each model is trained using the hyperparameters which were deemed optimal in their respective chapters. The comparison will be made in tabular form for all.

7.1.1 Single-Output Models

In Table 7.1, a summary is provided of the MSE for all single-output models for each of the aerodynamic coefficients. In the table, the minimum MSE of the model for each specific aerodynamic coefficient is highlighted. One can see that in terms of MSE for the pitching moment coefficient C_m the CatBoost model has the best predictive capability, followed closely by the BNN. The ANN follows closely behind. This trend continues throughout the other coefficients where the CatBoost, ANN, and BNN are exchanging the role of being the best model in terms of MSE. It is only for the side force coefficient in the y-direction that the XGBoost model is outperforming all the models, with the ANN and BNN following shortly after.

In Table 7.2, the percentage of the predictions that fall within the tolerances are provided (PWT). The same trends as with the MSE are present in the percentage difference metric. For the C_m, C_l, and C_n the best models for the MSE are also the best models for the percentage difference. However, for the force coefficients, the same cannot be said. Although the models which perform well on MSE are still performing well in terms

Table 7.1: Mean squared error (MSE) comparison for all models and aerodynamic coefficients. RF = Random Forest, XGB = XGBoost, LGB = LightGBM, CAT = CatBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network.

	RF	**XGB**	**LGB**	**CAT**	**ANN**	**BNN**
C_m	4.802e-05	1.163e-05	8.497e-06	**4.390e-06**	6.105e-06	4.619e-06
C_l	1.013e-05	7.010e-06	6.370e-06	**5.703e-06**	**5.724e-06**	6.554e-06
C_n	4.628e-06	3.252e-06	3.459e-06	2.914e-06	**2.759e-06**	3.285e-06
C_x	1.159e-05	2.507e-06	1.587e-06	2.524e-06	9.223e-07	**7.831e-07**
C_y	8.719e-06	**6.472e-06**	1.497e-05	1.212e-05	6.738e-06	8.932e-06
C_z	1.494e-04	6.774e-05	5.863e-05	**4.677e-05**	6.545e-05	4.909e-05

of percentage difference, one can see that for example for the C_x, the CatBoost and BNN have identical values for the percentage of the predictions that fall within the tolerance. However, the MSE of the CatBoost and BNN differ by over a factor of 3, with the BNN model having far lower MSE than the CatBoost model. This can be understood that the CatBoost model is following the dataset very closely, but when it makes an error, the error is very large, which is seen in the MSE. This is in line with what has been discussed in chapter 4. On the other hand, the BNN has a good MSE, meaning that it follows the test data, and has fewer outliers. Therefore, when it does go outside the tolerances, it is not significant. For C_y, and C_z, the model performance is so high that the differences between the models become insignificant, as the models are almost perfectly predicting the dataset. The MAPE for the individual models can be found in Appendix D, section D.1, as it was not deemed useful to discuss when the MSE of the different models is already discussed.

Table 7.2: Comparison of all the models and aerodynamic coefficients of the percentage of the model predictions within the tolerance (PWT) for the single-output models of the test dataset. RF = Random Forest, XGB = XGBoost, LGB = LightGBM, CAT = CatBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network.

	RF [%]	**XGB** [%]	**LGB** [%]	**CAT** [%]	**ANN** [%]	**BNN** [%]
C_m	68.8	87.3	91.4	**97.0**	94.9	96.6
C_l	94.7	96.2	96.3	**96.7**	95.3	94.1
C_n	95.7	97.9	97.4	97.9	**98.1**	97.3
C_x	72.8	89.2	92.5	**96.7**	96.0	**96.7**
C_y	99.1	99.5	97.9	98.2	**99.6**	99.0
C_z	99.3	99.8	99.9	99.9	99.6	**100**

Returning quickly to whether the ANN and BNN also see a drop in performance specifically for the C_m and C_x one can look at Table 7.2. The ANN and BNN models exhibit slightly lower PWT for the C_m than their average values. However, when one considers that for the tree-based models, the drop was significant compared to the average value, but also compared to a reference coefficient which is well predicted, such as C_l.

One can conclude that the ANN, and BNN models do not suffer from the same drop in performance for the C_m, or C_x as the tree-based models (Random Forest, XGBoost, LightGBM) do.

The training times of each of the models will be compared for dataset 1, the largest dataset available. For the tree-based models, the training time was between 10 seconds for the Random Forest, and 60 seconds for the CatBoost model. For the artificial neural networks, the training time was around 2500 seconds per model, meaning around 42 minutes. Finally, the Bayesian neural network training time was around 22 hours. Therefore, one can see that for the amount of training time that is needed, the CatBoost model is by far the most cost-effective around.

7.1.2 Multiple-Output Models

In this subsection, the multiple-output models that are created for the aerodynamic dataset are evaluated and compared for the tree-based, ANN, and BNN models.

Firstly, one can see that there are only multiple-output models for the Random Forest model, XGBoost, artificial neural network, and the Bayesian neural network. The light-GBM and CatBoost models do not support multiple-output regression and therefore are not calculated.

In Table 7.3, and Table 7.4 the MSE and percentage within the tolerance (PWT) are respectively shown for each of the multiple-output models. The model outputs 6 values, for 8 input features. However, to visualise the MSE and PWT, they are calculated for each aerodynamic coefficient separately, to be able to compare it towards the single-output models. To obtain the collective MSE or PWT one simply needs to average the values found in Table 7.3, and Table 7.4 column-wise.

The very first thing that can be noticed from Table 7.3 is that the MSE of the multiple-output model for the XGBoost model, is the same as making a single-output model for each of the aerodynamic coefficients. Furthermore, it can be seen that the artificial neural network is dominant in terms of the best MSE out of the multiple-output models which are compared in Table 7.3. Only for the C_n and, C_y the XGBoost model provides a lower MSE than the ANN. The BNN is competitive with the ANN for each of the coefficients besides the C_z, without ever being able to outperform it. Even for the pitching moment coefficient, the BNN cannot outperform the ANN, which was its strong suit in the single-output model.

For the PWT that can be seen Table 7.4, the XGBoost model performs well since the multiple-output model has the same performance as the single-output model and single-output models are more capable of predicting each aerodynamic coefficient than

Table 7.3: Mean squared error (MSE) comparison for all multiple-output models and aerodynamic coefficients. RF = Random Forest, XGB = XGBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network.

	RF	**XGB**	**ANN**	**BNN**
C_m	1.042e-04	1.163e-05	**7.132e-06**	8.471e-06
C_l	2.379e-05	7.011e-06	**6.303e-06**	7.944e-06
C_n	3.409e-05	**3.252e-06**	3.553e-06	3.275e-06
C_x	6.812e-05	2.507e-06	**1.631e-06**	1.537e-06
C_y	5.962e-05	**6.472e-06**	8.087e-06	7.441e-06
C_z	2.515e-04	6.774e-05	**6.029e-05**	1.021e-04

multiple-output models. Besides the C_m, and the C_x, which are predicted best by the ANN and BNN respectively, the XGBoost model has the highest PWT for the remaining aerodynamic coefficients.

Table 7.4: Percentage within tolerance (PWT) comparison for all multiple-output models and aerodynamic coefficients. RF = Random Forest, XGB = XGBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network.

	RF [%]	**XGB** [%]	**ANN** [%]	**BNN** [%]
C_m	49.5	87.3	**93.5**	91.6
C_l	85.1	**96.2**	94.0	92.8
C_n	88.2	**97.9**	97.4	97.7
C_x	33.8	89.2	91.9	**92.9**
C_y	93.2	**99.5**	99.3	99.3
C_z	98.2	**99.8**	**99.8**	99.2

In Table 7.5, the PWT of the multiple-output models are compared towards their single-output counterparts. One can see a clear picture, the best model for a given aerodynamic coefficient is never beaten by its multiple-output counterpart. Even when a multiple-output model outperforms its single-output counterpart, there is another model which outperforms both. A multiple-output model in general could be capable of providing a better prediction of a given aerodynamic coefficient when it has learnt to transfer the information from one of the coefficients towards the others. The other way is because it might have learned shared information between the output values. For the multiple-output ANN, the C_z is marginally better predicted than its single-output counterpart. For the BNN, this is the case for the C_n and C_y. For the ANN it was shown in section 5.3, that the ANN multiple-output model was primarily trained to fit the C_z due to its scale compared to the other coefficients. When comparing the PWT of the models, the difference is marginal at 0.2%. With this in mind, one could also theorise that the randomness in the training could be at the foundation of the difference between the multiple-output models and the single-output models. For the BNN, the inherent random noise that is sampled in the Bayesian layers of the network to perform the

reparametrization trick is not fixed, and this random noise that is generated could mean that the multiple-output model, has had a random lucky draw, or the single-output had a random unlucky draw in training the model which resulted in the lower performance. Sadly, performing a large average of the BNN is very costly. In general, the conclusion can be made that single-output models provide much better predictive performance than multiple-output models.

Table 7.5: Percentage within tolerance (PWT) comparison for all single- and multiple-output models and aerodynamic coefficients. RF = Random Forest, XGB = XGBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network.

	RF [%]		**XGB [%]**		**ANN [%]**		**BNN [%]**	
	Single	**Multi**	**Single**	**Multi**	**Single**	**Multi**	**Single**	**Multi**
C_m	68.8	49.5	87.3	87.3	94.9	93.5	96.6	91.6
C_l	94.7	85.1	96.2	96.2	95.3	94.0	94.1	92.8
C_n	95.7	88.2	97.9	97.9	98.1	97.4	97.3	97.7
C_x	72.8	33.8	89.2	89.2	96.0	91.9	96.7	92.9
C_y	99.1	93.2	99.5	99.5	99.6	99.3	99.0	99.3
C_z	99.3	98.2	99.8	99.8	99.6	99.8	100.0	99.2

7.1.3 Conclusion

In this section, the single-output and multiple-output models have been shown and compared between each other. From the tables shown in subsection 7.1.1, and subsection 7.1.2 one can conclude that the CatBoost, ANN, and BNN are the most competitive in terms of their MSE and PWT. However, the CatBoost, ANN, and BNN are not always performing the absolute best, as for the side force coefficient in y-direction C_y the XGBoost model has the lowest MSE out of all the models. This makes choosing one model that is the best overall tricky, as for each aerodynamic coefficient that is predicted, the outcome is different. Furthermore, it is not because one of the models is very good for a certain aerodynamic coefficient, that the others are not performing well, there is performance that can be extracted from each of the individual models. Therefore, in section 7.2 an ensemble technique called model stacking will be investigated. It investigates whether one can go from many models that all predict the dataset in varying levels of proficiency, towards a single model, whose performance is better than any of the models individually.

7.2 Stacked Models

Stacking models is an ensemble modelling technique, which means that multiple different models are combined to provide the output. This process is often employed to improve the predictive accuracy. There are many different ensembling techniques to combine the outputs of different models into one. Some basic ensembeling methods are averaging, or weighted averages. Some more advanced methods are stacking, which is the focus of this section and is discussed and researched in the literature study in section 3.5. However, there exist more techniques, such as blending, bagging or boosting which are also discussed in section 3.5, but not applied here. From the literature study on model ensemble techniques and stacking, it was found that stacking works better when the base models are more dissimilar. Furthermore, the meta-model, which is the model trained to determine the optimal combination of base models, can be a wide variety of different models such as linear models, tree-based models, neural networks, or even attention networks as was shown in section 3.5. The analysis in this section will validate the results found in the literature study and apply model stacking towards the models which have been considered for aerodynamically modelling the dataset of a combat aircraft.

In this section, the methodology for stacking models is provided in subsection 7.2.1. After which the meta-model is chosen in subsection 7.2.2, and finally the stacking model results are provided in subsection 7.2.3.

7.2.1 Methodology for Stacking Models

In this subsection, the methodology for stacking different models is provided.

To create a stacked model, four core steps need to be performed.

1. Split the data into training, validation and test data.
2. Train and tune the hyperparameters of the models which are to be stacked together.
3. Choose and train the meta-model, with the predictions of the models constituting the stacked model, known as the base models.
4. Evaluate the performance of the stacked model.

1. Test, Train, Validation split

The total training dataset is split into three groups. The training dataset is taken to be around 80% of the data. This dataset is used to train the models which constitute the

stacked model. The validation set is made up of around 10% of the data. The validation set is used to evaluate the performance of the models which are being trained. The hyperparameters can be optimised with this dataset, and the predictions of the models on this dataset are used to train the meta-model. Finally, the test dataset is taken to be around 10% of the total dataset. The test dataset has not been used up to this point, and this is because it is used to evaluate the performance of the stacked model. Therefore, this dataset has never been seen by any of the base models, or the meta-model itself. It can therefore give a true evaluation of the stacked model.

2. Train and Tune the Models

The next step of creating a stacked model is to train the models which will make up the stacked model. This is done conventionally as has been described before in the respective sections of the machine learning models, except it is evaluated on the validation set and not the test dataset. Furthermore, in this step, the decision on which models to use can be made. As described above, the power of model stacking is that many average-performing models can be combined to make a great-performing model. So it might be worth adding an under-performing model to the stacked model. The choice of base models are the Random Forest model, XGBoost, LightGBM, CatBoost, artificial neural network, and Bayesian neural network. These are chosen since the tree-based models, have a very low cost of training, and perform well with very low training times, and hyperparameter tuning. Then artificial neural networks and Bayesian neural networks are used since these are the main models which are being developed in this thesis, meaning that these are the models of which the performance is desired to be increased. Furthermore, with the addition of the ANN and BNN, the base models are trained very differently and will have different strengths, which can be utilized in the stacked model.

3. Choose and Train the Meta-model

The choice of meta-model is important as it can provide the last boost in performance for the complete model. Therefore, a careful evaluation of the meta-model needs to be performed. For simplicity, three models are considered to be the meta-model, this is a simple linear regression model, which is chosen for its simplicity and interpretability. Then a neural network and extreme gradient boosting model are chosen, as the architecture for training these models has already been developed in the thesis, and thus can be implemented readily. The attention network that was proposed during the literature study (section 3.5) is left for future research.

The training of the meta-model is different to the training of the base models, as the meta-model is trained from the outputs of the base models on the validation data, rather than directly from the data. Thus, to train the meta-model, each base model provides their prediction on the validation data. The meta-model is then trained based on the

predictions of the base models on the validation data, and the outputs of the validation data.

4. Evaluate Performance

Finally, with the meta-model trained, predictions can be made on the test dataset. This should provide increased performance for the stacked model, compared to the base models when both are being evaluated on the test dataset. This is a dataset that both the base model and the stacked model have not encountered. Therefore, it can provide a good metric for whether the stacked model increased the performance of the base models.

7.2.2 Choosing the Meta-model

The choice of the meta-model to be used for stacking the models is between an artificial neural network, a tree-based model which is chosen to be XGBoost, and finally a simple linear regression model. This stacked model will find the optimal weights for weighting the different models. Below, the characteristics of the meta-models which are used to perform the model stacking are provided. The hyperparameters of the linear regression model are not provided as the model does not have any.

ANN meta-model

- Single layer ANN
- 1000 epoch convergence
- Batch size equal to validation dataset size
- Learning rate: 1e-4

XGBoost meta-model

- Max depth of trees: 6
- Loss function: mean square error
- Learning rate: 0.3
- Number of estimators: 100

In Table 7.6 the mean squared error of the base models on the test dataset for the pitching moment coefficient are provided in the columns. The pitching moment, since it is the most critical aerodynamic coefficient is once again used to decide which meta-model to use. The meta-model can then be extrapolated towards the other aerodynamic coefficients. The most interesting takeaway of Table 7.6 is that for the ANN meta-model with 10, and 50 hidden neurons, the stacked model performs worse than the best-performing base model. When considering that when the neurons increase to 300 and more, the performance is better than the best base model. This indicates that the reason for the low performance is due to the meta-model not properly learning the underlying pattern of the base model predictions. The performance of the best ANN meta-model reduces the MSE by 24% compared to the best base model. This shows

the power of stacking models, as even though the individual MSE of the base models is higher, the stacked model provides a lower MSE for the test dataset. For the XGBoost meta-model, the stacked model performs slightly better than the best base model, with a MSE decrease of 8%. Finally, the linear regression meta-model performs best, and much better than any of the base models. The linear regression meta-model has a MSE decrease of 25% compared to the best base model. This shows that the linear regressor is the best meta-model in this experiment. The only stacked model which has comparable MSE performance is the ANN meta-model with 10,000 neurons, which takes more time to train and is less interpretable than the linear meta-model. From the linear meta-model, the coefficients can be extracted which provide a measure for how much each of the base models is weighted in the final prediction. This is shown in Table 7.7.

Table 7.6: Mean Squared Errors (MSE) on the test dataset of the base models and stacked model, for the pitching moment coefficient C_m, using an ANN, XGBoost, and linear meta-model. RF = Random Forest, XGB = XGBoost, LGB = LightGBM, CAT = CatBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network.

Meta-model	RF	XGB	LGB	CAT	ANN	BNN	Stacked
ANN (10 neu)	4.80e-05	1.16e-05	8.49e-06	4.39e-06	6.11e-06	4.62e-06	3.04e-05
ANN (50 neu)	4.80e-05	1.16e-05	8.49e-06	4.39e-06	6.11e-06	4.62e-06	4.60e-06
ANN (300 neu)	4.80e-05	1.16e-05	8.49e-06	4.39e-06	6.11e-06	4.62e-06	3.50e-06
ANN (500 neu)	4.80e-05	1.16e-05	8.49e-06	4.39e-06	6.11e-06	4.62e-06	3.49e-06
ANN (1500 neu)	4.80e-05	1.16e-05	8.49e-06	4.39e-06	6.11e-06	4.62e-06	3.43e-06
ANN (10000 neu)	4.80e-05	1.16e-05	8.49e-06	4.39e-06	6.11e-06	4.62e-06	3.32e-06
XGBoost	4.80e-05	1.16e-05	8.49e-06	4.39e-06	6.11e-06	4.62e-06	4.03e-06
Linear	4.80e-05	1.16e-05	8.49e-06	4.39e-06	6.11e-06	4.62e-06	3.28e-06

In Table 7.7, the coefficients of the linear meta-model are provided for the pitching moment coefficient. One can see that the CatBoost, ANN, and BNN form the largest portion of the model predictions, with the Random Forest, XGBoost, and LightGBM models providing a much smaller contribution. One can see from Table 7.7, that the Random Forest model hurts the stacked model performance since it has a negative coefficient. Therefore, in the pursuit of the minimum MSE model, and maximum percentage within tolerance, the Random Forest model is removed to determine whether this would increase the performance. The MSE of a stacked model with the Random Forest model removed, and retaining the other models, comes down to 3.279e-06, and a percentage within tolerance of 97.99%. Compared to the original 3.277e-06, and a percentage within the tolerance of 98.03%, the stacked model performance is worse in this case. Even though the coefficient of the linear meta-model indicated it had to subtract the Random Forest from the model slightly. Regardless of this result, the model ensembling technique does require different models that each contribute in their own way to increase the model performance. The weighting of the lower-performing models is insignificant compared to the highest-performing model, and the loss or gain in performance by adding them is limited. The complete effect on the MSE of subtracting a given base model from the stacked model can be seen in Appendix D, section D.2. In

section D.2, the same conclusions can be made for the other aerodynamic coefficients, that even though a positive coefficient is seen, the effect on the MSE could be negative, and vice versa. However, these effects are only seen for the models which have a very low contribution towards the stacked model. When one also considers the training time of the models that are underperforming, which for the Random Forest is under 10 seconds, and the XGBoost model around 1 minute, the small extra benefit, is also only at a very low computational cost. Therefore, it is still advised to add each model which is available towards the stacked model. Furthermore, as one has seen from section 7.1, the Random Forest, or XGBoost model might not perform well on the pitching moment coefficient but do perform better for some of the other aerodynamic coefficients, which means that the models could become important in the stacked models for some of the other aerodynamic coefficients.

Table 7.7: Coefficients of the linear meta-model trained on the pitching moment.

	RF	**XGB**	**LGB**	**CAT**	**ANN**	**BNN**
Coefficients	-0.0336	0.00430	0.0281	0.448	0.174	0.379

7.2.3 Model Stacking Results

In this subsection, the results for model stacking for each of the aerodynamic coefficients, and all the models are shown. This allows one to quantify the increased performance that has been gained from the model stacking. This subsection shows an example prediction of the stacked model in Figure 7.1. Furthermore, three tables are presented. The weights of the base models are shown in Table 7.8, which can be used to evaluate the importance of each base model towards the stacked model. Secondly, Table 7.9 shows for each aerodynamic coefficient the MSE for the stacked model and other models. Finally, the percentage of the predictions that fall within the tolerances for each of the models, and aerodynamic coefficients is shown in Table 7.10.

In Figure 7.1 an example plot is provided from the aerodynamic dataset. In the top plot, a test data polar is provided, and the prediction of the stacked model is given. In the bottom plot, the predictions of the base models are provided. One can see the different characteristics of the base models. Firstly, the ANN and BNN models are much smoother than the tree-based models. In this specific plot for the pitching moment coefficient, one can see that the ANN and CatBoost models are performing the best, which is in line with the results shown in Table 7.1. However, in this example, the BNN does not provide a good prediction of the particular polar. The power of model ensembeling can therefore be used since on average the BNN is providing low MSE predictions. Since it is not only the BNN model making a prediction, and the stacked model is created by adding the ANN and CatBoost models mostly, the overall stacked prediction is performing much better. In this plot the ANN would be capable of providing a good prediction on its own but, this is not the case for every polar.

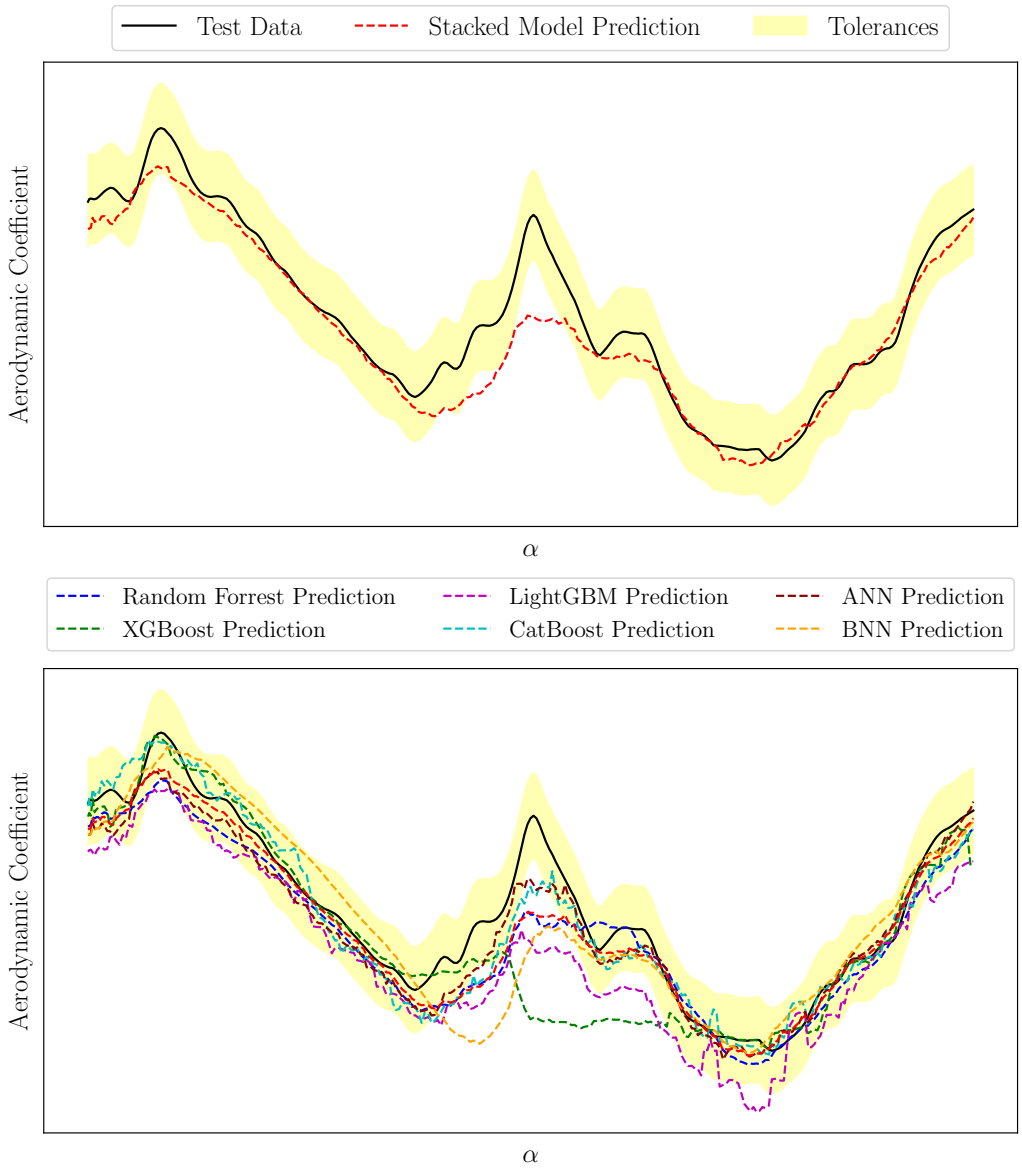


Figure 7.1: Example plot of the stacked model prediction of a test data polar using a linear regression meta-model and base models: Random Forest, XGBoost, lightGBM, Artificial Neural Network, and Bayesian Neural Network.

Table 7.8: Weights of the base models of the stacked model for each of the aerodynamic coefficients. RF = Random Forest, XGB = XGBoost, LGB = LightGBM, CAT = CatBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network.

Coefficients	RF	XGB	LGB	CAT	ANN	BNN
C_m	-0.0336	0.00430	0.0281	**0.448**	0.174	0.379
C_l	0.0735	**0.230**	0.117	0.213	0.189	0.178
C_n	0.015	0.116	0.00460	0.291	**0.372**	0.204
C_x	-0.0330	0.0310	0.0584	0.239	0.294	**0.412**
C_y	0.0886	0.107	0.342	-0.101	0.149	**0.410**
C_z	0.0198	0.167	0.0119	**0.307**	0.221	0.272

In Table 7.8, the weights of the base models for the different aerodynamic coefficients are provided. What can be noticed is that there is a correlation between the coefficients of the models and the MSE and the percentage of the predictions within tolerance. However, the weights are not exactly ranked based on the MSE or percentage within the tolerance. For example for the aerodynamic coefficient C_l, the lightGBM model receives the largest weight, while it only has the third lowest MSE out of all the models and the second-highest percentage within tolerance (Table 7.9, Table 7.10).
One can see from Table 7.8, that each of the models has at least one aerodynamic coefficient where its predictions form a significant portion of the stacked model. Stronger still is that each model besides the XGBoost, and Random Forest model has once the largest contribution towards the stacked model. This indicates that each model in the model ensemble is there for a reason, and can provide extra insight into a particular coefficient. If the model is not important the magnitude of the weight will be low, and then it does not influence much if one removes it. Therefore, the choice is made to have all models as base models, as the meta-model can learn to neglect a model when it provides poor predictions.

When looking at the MSE and percentage within tolerance (PWT) shown in Table 7.9, and Table 7.10 respectively, one can determine very quickly that the stacked model increases the performance for each aerodynamic coefficient. For the MSE error, the stacked model decreases the MSE by an average of 27%. For C_m, it does this by 34%, for C_l by 33%, for C_n by 31%, for C_x by 10%, for C_y by 25%, and for C_z by 25%. These are significant improvements, which are obtained for almost no extra computational cost since the training of the linear meta-model is extremely fast (under 1 second). For the percentage within tolerance, the stacked model is better than each of the best individual models for the aerodynamic coefficients. However, the effects are less pronounced. Meaning that on average an increase of 0.5% can be expected of the percentage that the predictions fall within the tolerance. The difference between the metrics shows that the stacked model is primarily averaging out the very bad predictions of the best models, which decreases the MSE dramatically but will not have such a large impact on the percentage within tolerance, as a prediction that is outside the tolerance could still be outside the tolerance

with the stacked model, but will likely be closer towards the test data meaning that the MSE is lowered dramatically but not the percentage within tolerance.

Table 7.9: Mean squared error (MSE) comparison for all models and aerodynamic coefficients with stacked model included. RF = Random Forest, XGB = XGBoost, LGB = LightGBM, CAT = CatBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network.

	RF	**XGB**	**LGB**	**CAT**	**ANN**	**BNN**	**Stacked**
C_m	4.802e-05	1.163e-05	8.497e-06	**4.390e-06**	6.105e-06	4.619e-06	3.277e-06
C_l	1.013e-05	7.010e-06	6.370e-06	**5.703e-06**	**5.724e-06**	6.554e-06	4.309e-06
C_n	4.628e-06	3.252e-06	3.459e-06	2.914e-06	**2.759e-06**	3.285e-06	2.105e-06
C_x	1.159e-05	2.507e-06	1.587e-06	2.524e-06	9.223e-07	**7.831e-07**	7.098e-07
C_y	8.719e-06	**6.472e-06**	1.497e-05	1.212e-05	6.738e-06	8.932e-06	5.153e-06
C_z	1.494e-04	6.774e-05	5.863e-05	**4.677e-05**	6.545e-05	4.909e-05	3.733e-05

Table 7.10: Percentage of the model predictions within the tolerance of the test dataset (PWT), comparison for all the models including stacked model and aerodynamic coefficients. RF = Random Forest, XGB = XGBoost, LGB = LightGBM, CAT = CatBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network.

	RF [%]	**XGB** [%]	**LGB** [%]	**CAT** [%]	**ANN** [%]	**BNN** [%]	**Stacked** [%]
C_m	68.8	87.3	91.4	**97.0**	94.9	96.6	98.02
C_l	94.7	96.2	96.3	**96.7**	95.3	94.1	97.2
C_n	95.7	97.9	97.4	97.9	**98.1**	97.3	98.7
C_x	72.8	89.2	92.5	**96.7**	96.0	**96.7**	97.5
C_y	99.1	99.5	97.9	98.2	**99.6**	99.0	99.7
C_z	99.3	99.8	99.9	99.9	99.6	**1**	99.9

7.3 Conclusion

In this chapter, the different models which were discussed in this thesis, namely the tree-based models, artificial neural networks, and Bayesian neural networks, were compared against each other. Furthermore, a stacked model was created to improve the performance of the individual base models.

The easiest way to conclude this entire chapter is to return towards the research questions which were posed at the start of the chapter. The main research question was: *What increase in performance can be expected when implementing a stacked model?*, for this one can look at the subquestions below and finally one can answer the main research question.

- **Q4.1: How do the predictive performances of tree-based, ANN, and BNNs compare in terms of MSE and percentage within tolerance?**
  In section 7.1, the comparison is made for all the models in terms of MSE, and PWT. One can see that three models are highly dominant in their average predictive performance of all the aerodynamic coefficients, these are the CatBoost model, ANN, and BNN. For the PWT the same trends can be retrieved, with the best model for each aerodynamic coefficient having a PWT of over 96%. To compare the actual values of the MSE and PWT for the single-output models one can refer to Table 7.1, Table 7.2.
- **Q4.2: How do the training times of the different models compare?**
  The training times of the different models are compared in section 7.1, and on average the training time of the tree-based models is between 8 and 60 seconds, the lowest for the Random Forest and highest for the CatBoost model. For the artificial neural network, the training time is around 2500 seconds or around 42 minutes, already a significant increase compared to the tree-based models. Finally, the Bayesian neural network has the highest training time of around 22 hours. The high training time was anticipated from the literature study done in section 3.4, where the computational challenges of inference were stated. The large training time of BNNs does not form an immediate issue, besides the limitation that no genetic optimisation algorithms can be used at this moment to optimise the hyperparameters further.
- Q4.3: **Which meta-model provides the highest performance increase for the ensemble model, and at what computational cost?**
  The meta-models which were analysed in subsection 7.2.2 showed that the low computational cost and high performance of a linear regression function as the meta-model was a superior choice compared to a neural network, or XGBoost model as the meta-model. On top of the best performance and lowest training time, the linear regression model allows for the coefficients to be extracted, which

can give insights into the weighting of each of the base models. The application of a stacked model comes at almost no additional training time when compared to the artificial neural network, and Bayesian neural networks. The training time of the linear regression meta-model is less than 1 second, and thus a very low-cost method for increasing the performance of the base models. The biggest computational cost of the stacked model is the training of the base models. Even though the meta-model itself is negligible in terms of its training time, the addition of for example the Bayesian neural network towards the base models of the meta-model requires training the model in the first place, requiring an additional 22 hours.

- **Q4.4: What increase in performance can be expected when implementing a stacked model?**
  In subsection 7.2.3, the performance increase of the stacked model compared to the best base model is provided. The MSE of the stacked model is on average 27% lower than that of the best model for each of the aerodynamic coefficients. This is broken down into 34% for C_m, 33% for C_l, 31% for C_n, 10% for C_x, 25% for C_y, and 25% for C_z. For the PWT the difference is less pronounced, meaning that on average only an increase of 0.5% in PWT is observed. Which is broken down into an increase in PWT of 1.02% for C_m, 0.5% for C_l, 0.6% for C_n, 0.8% for C_x, 0.1% for C_y, and -0.1% for C_z, since the BNN had a perfect 100% PWT as a base model. Therefore, two things can be concluded, firstly the difference between the decrease in MSE and PWT for the stacked model shows that the stacked model is primarily averaging out the very bad predictions of the best models, which decreases the MSE dramatically but will not have such a large impact on the percentage within tolerance. This is because a prediction that is outside the tolerance could still be outside the tolerance with the stacked model, but will likely be closer towards the test data meaning that the MSE is lowered dramatically but not the percentage within tolerance. Secondly, stacked models form an excellent method in reducing the MSE and increasing the PWT, and should form the standard practice in the pipeline of creating a machine learning model to model the aerodynamic dataset of a combat aircraft.

Chapter 8

Conclusion and Recommendations

In this chapter, the conclusions which can be drawn from the research that has been performed in this thesis are provided in section 8.1. Furthermore, the recommendations that can be provided for future work are given in section 8.2.

8.1 Conclusion

To conclude the work that has been done in this thesis, one must first return towards the objectives of this thesis. The aerodynamics of a combat aircraft is a highly important feature in ensuring it is competitive and successful compared to other rival aircraft, during combat but also for the sales of the aircraft. The European fighter aircraft which utilise a canard-delta wing configuration are known to be unstable in subsonic conditions. To safely control the aircraft, not only in terms of stability but also in terms of not overloading the airframe and other features such as autopilot, a flight control system is implemented. This flight control system has as one of its inputs, an aerodynamic model of the complete aircraft. This aerodynamic model describes the aircraft's aerodynamic forces and moment coefficients, as a function of the control surface deflections, angle of attack, angle of sideslip, and Mach number for a specific combination of weapons and payloads under the wing. This aerodynamic model is made currently by linearly interpolating between the aerodynamic dataset that is created based on wind tunnel, computational fluid dynamics simulations, and flight test data. This process is performed manually and takes significant resources to produce. Often the aerodynamic model needs to be created or updated when an update is performed

on the aircraft or a new design of payload wants to be added underneath the wing, which changes the aerodynamics. Therefore, research is performed to reduce the cost of making this model. Machine learning models, which are versatile and capable nonlinear function approximators are investigated if they could replace or aid the creation of the aerodynamic model. Furthermore, this thesis aimed to research which machine learning models specifically would be optimal in predicting the aerodynamic dataset of a combat aircraft.

The literature study was performed towards the application of machine learning models for modelling the aerodynamic dataset coming from combat aircraft. The literature study showed many examples of neural networks being applied towards aircraft and combat aircraft design, but not on the same scale as is attempted in this thesis. The datasets were much smaller and simpler compared to the ones considered in the thesis. Therefore, the literature study was expanded more broadly towards different methods which could model the aerodynamic dataset. From this, three types of models could be identified which could aid in predicting the aerodynamic dataset of a combat aircraft, namely tree-based models, artificial neural networks, and Bayesian neural networks. The first type of model, a tree-based model, was found to be very competitive in predicting tabular datasets and often forms the state of the art in this field. Often, tree-based models were compared with artificial neural networks in terms of their performance. That is why artificial neural networks are also one of the models which will be implemented to model the aerodynamic dataset, their smoother nature compared to tree-based models is something which is thought to provide a competitive advantage. Finally, Bayesian neural networks are also suggested, as they not only provide a prediction but also the uncertainty of this prediction. This quality of Bayesian neural networks is found highly interesting and could allow the designers of the flight control system to know in which regions the aerodynamic model is very well-defined, and where little data causes the prediction of the aerodynamics to be very uncertain. Furthermore, the state-of-the-art machine learning models investigated during the literature study demanded more research into hybrid models, and genetic optimisation algorithms for hyperparameter tuning.

Several tree-based models were used to predict the aerodynamic dataset, these include the Random Forest, AdaBoost, XGBoost, lightGBM, and CatBoost model. CatBoost emerged as the dominant tree-based model, with LightGBM and XGBoost also showing strengths in particular areas. Adaboost performed so poorly, due to its decision trees only making a single split, that it was not considered beyond the hyperparameter optimisation stage. Tree-based models exhibit non-smooth characteristics that can impact their applicability. While the Random Forest and XGBoost models struggle with larger non-smooth deviations, CatBoost and lightGBM have slightly smoother behaviour. The smoothness of CatBoost and lightGBM can be further improved with mean averaging, making them extremely high-performing. CatBoost has shown itself to be a model that has low training times (under 1 minute), little need for hyperparameter tuning, relatively

smooth behaviour for a tree-based model, and excellent performance with over 96.7% of the predictions within the tolerance for each aerodynamic coefficient. The explainability of tree-based models forms another strength, as the feature importance metric showed that the physical features of the aircraft behaviour are learned by the models. Among the models examined, only Random Forest and XGBoost support multiple-output modelling, and it was concluded that multiple-output models were not worth using, as they did not improve predictive performance. When compared to the other models in the thesis, it is not recommended to apply tree-based models towards overly smooth datasets, as they are not optimised for these function approximations. This could be seen when it modelled the extremely smooth force coefficient in x-direction C_x. When modelling aerodynamic coefficients that have strong oscillations in the dataset, the tree-based models are excellent candidates, as the decision trees that make up the models, allow for sharp jumps.

For artificial neural networks (ANNs), the model was optimised using two techniques. Parameters such as a batch size of 128, the learning rate of 1e-4, ReLU activation function, ADAM optimiser, a dropout rate of 0.5, no batch normalisation, and no L1 or L2 regularization, were determined by comparing the different options against each other. To determine the number of layers and the number of neurons in each layer a genetic optimisation algorithm is developed. The genetic optimisation algorithm found 5 hidden layers, with 872, 866, 1016, 982, and 407 neurons in each of the layers to be the optimal choice of hyperparameters in modelling the aerodynamic moment coefficient. Also, techniques such as early stopping and parallelization were developed to reduce the time required to run this optimisation. This resulted in the neural network being able to predict on average over 95% of the test dataset within the specified tolerances for each of the aerodynamic coefficients. Multiple-output models showed a mixed result, performing better only for the side force coefficient in z-direction while performing worse for all other coefficients. Although the not fully optimised hyperparameters could be at the base of this, the larger magnitude of the side force coefficient in the z-direction means this coefficient dominates the mean squared error calculation during training of the multiple-output model. Therefore, a different metric and normalising of the outputs are recommended in the future. The neural network, as for any machine learning model, was not capable of generalising towards a different dataset, and transfer learning or retraining is required. Transfer learning reduced the training time by 14%, but did not affect the overall predictive performance compared to retraining the network. Finally, using the gradient information w.r.t the angle of attack as an additional input feature was shown to improve the predictive performance of the neural network on average by 7% for the MSE, and 0.5% for the percentage within the tolerance of the pitching moment coefficient.

The Bayesian neural network (BNN) that was developed for this thesis, uses variational inference, and Bayes-by-backprop as it is scalable towards larger datasets, and aims to keep the computational cost down compared to Markov Chain Monte Carlo methods.

Due to the high computational cost of BNNs, no genetic optimisation algorithm can be used to determine the hyperparameters of this network. A heteroscedastic model is chosen, as the variance of the model varies over the angle of attack and other parameters, allowing the uncertainties to grow and diminish over the solution space. The number of samples used to approximate the posterior distribution was identified as a key parameter in reducing the MSE of the model on the test dataset. Furthermore, the number of neurons was identified to be more important than the number of hidden layers in the network, resulting in only 1 hidden layer and 1024 neurons in the network. The predictive performance of the BNN, was on par with the ANN, and tree-based models, especially for the pitching moment coefficient where it predicted over 96.6% of the test dataset within the tolerance. When considering that the BNN was not optimised much for its hyperparameters, the model performs extremely well with such limited hyperparameter optimisation. The BNN thus has the most margin for improvement compared to any of the other models which have been considered in this thesis, while already providing some of the best predictions. The training time of BNNs was a factor of 30 larger than ANNs, and 80,000 compared to a tree-based model. A single BNN model takes around 22 hours to train, with the multiple output model taking 32 hours. The total uncertainty coming from the Bayesian neural network was much lower than the tolerances of the test data. The uncertainties coming from the BNN do not encompass the test dataset, meaning that the predictions are overconfident. Furthermore, it was shown that the RMSE of a prediction, and the aleatoric, or epistemic uncertainty are correlated. However, the correlation is low and does not constitute the conclusion of reducing the tolerances of the model to the uncertainties that are given by the BNN.

The ensemble technique of model stacking was used, as it was identified from the comparison of all the models, that each model had the capability of modelling the aerodynamic dataset. Therefore, a stacked model could leverage the strengths of each of the models and improve the performance compared to the best model significantly. The stacked model utilised a linear regression function as its meta-model due to its low computational cost, superior predictive performance, and explainability. The stacked model increased the performance in terms of mean squared error (MSE) on average by 27% and the percentage within tolerance (PWT) by 0.5%. Therefore, two things can be concluded, firstly the difference between the decrease in MSE and PWT for the stacked model shows that the stacked model is primarily averaging out the very bad predictions of the best models, which decreases the MSE dramatically but will not have such a large impact on the percentage within tolerance. Secondly, stacked models form an excellent method for reducing the MSE and increasing the PWT at no additional cost for the stacked model, and should form the standard practice in the pipeline of creating a machine learning regression model.

Finally, one can conclude that machine learning models are capable of predicting the aerodynamic dataset of a combat aircraft well. The machine learning model cannot anticipate certain aerodynamic effects if it sees no training data of this phenomenon,

and results in PWT values that are not 100%. Therefore, a complete replacement of the previous aerodynamic model using linear interpolation is likely not possible at this time, due to regulations, but also the performance that needs to be 100%. However, machine learning models allow the designers of the traditional aerodynamic model to perform this faster, and with fewer mistakes. Since the machine learning model can predict the aerodynamic behaviour in locations where no aerodynamic data is available with high accuracy.

8.2 Recommendations

In this section, recommendations are made to further improve the aerodynamic dataset modelling of combat aircraft using machine learning.

For tree-based models, XGBoost was deemed the best model for the side force coefficient in y-direction (C_y), for the single-output model. However, for the other aerodynamic coefficients, the performance was often far below that of CatBoost, and LightGBM. The lower performance of XGBoost could be linked to the earlier release date compared to LightGBM and CatBoost. Therefore, the other models had improvements over the XGBoost model. XGBoost 2.0 was released in October 2023 and due to the timing of the research performed in the thesis, the older XGBoost 1.6 version was still used. XGBoost 2.0 features many updates, among which is the implementation of multi-target trees with vector leaf outputs, meaning that the multiple-output model would no longer be identical to the single-output model. Furthermore, GPU-related optimisations allow for faster training. Therefore, with this major update in one of the machine learning models for the thesis, it is advised to apply the model towards the aerodynamic dataset.

The use of gradients of the test data was investigated in section 5.5, as a way to improve the predictive performance of a neural network. The predictive performance was improved by 7% but never used for BNNs due to the high cost of running a BNN. Therefore, it is recommended to see whether the methodology can be applied to BNNs, and obtain the same result. The method is not deemed sensible to be applied towards tree-based models, because these models are capable of making sharper jumps more easily. Furthermore, for the BNN a more elaborate optimisation is advised. The simple linear optimisation of the hyperparameters, using only seven total runs to fix the hyperparameter values, was deemed sufficient in the thesis, and produced very competitive results compared to the other machine learning models. However, the BNN is thought to contain still the most undiscovered potential out of all the models in the stacked model. Finally, to reduce the training time of BNNs, future research can be made towards the implementation of a framework for transfer learning on a BNN. As was the case for ANNs, the training time could be reduced for the BNN by using transfer learning.

During the thesis, the multiple-output models for the BNN and ANNs were trained using the optimised parameters of the pitching moment coefficient. During future research, it is advised to perform a separate optimisation for the multiple-output models. Besides the multiple-output models which are advised to be given their own hyperparameter optimisation, the single-output models can also be optimised for each aerodynamic coefficient separately. In this manner, one could determine which coefficients require a more complex or simpler architecture.
It was concluded for the ANN multiple-output model that the different magnitudes of each of the aerodynamic coefficients were at the foundation of its lower performance compared to single-output models. Therefore, for future research different metrics could be explored which are less impacted by the different magnitudes. Normalising the output values, which was already performed for the BNN, could also improve the performance of the ANN multiple-output model. The multiple-output models trained during the thesis were not included in the stacked model due to their more competitive single-output models already being included. However, it would be interesting to test whether including them in the stacked model could improve the performance of the stacked model even further. As was seen in the literature study, a paper showcasing the strength of using attention networks as the meta-model for stacked models was provided [25]. During the thesis only a linear, XGBoost, and ANN meta-model was used due to time constraints. One could investigate whether using attention networks can achieve higher performance levels than the linear model that was chosen in the thesis.

Finally, at this stage, only wind tunnel data was used to create an aerodynamic model. However, more data is available in the form of flight tests, and computational fluid dynamics simulations. This means if a method could exist to fuse the data coming from different data sources, more data would be available for training the machine learning models, which would improve its performance. Therefore, a recommendation is made to perform data fusion between different data sources, to increase the dataset size, and sampling density.

Bibliography

[1] O.A. Abiodun, A. Jantan, A.E. Omolara, K.V. Dada, N.A. Mohamed, and H. Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 2018. doi: https://doi.org/10.1016/j.heliyon.2018.e00938.

[2] Airbus. Media centre, 2023. URL https://mediacentre.airbus.com/mediacentre/search?access=searchText&q=Eurofighter&metaSearch=_.

[3] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. *arXiv e-prints*, 2019. doi: 10.48550/arXiv.1907.10902.

[4] R. Alireza and S. Fariborz. Generalization of ANN-Based Aircraft Dynamics Identification Techniques into the Entire Flight Envelope. *IEEE Transactions on Aerospace and Electronic Systems*, 2016. doi: 10.1109/TAES.2016.140693.

[5] A. Andriy. *The Hundred-Page Machine Learning Book*. 2019. ISBN 9781777005474.

[6] P. Angelov and J. Soares. Towards explainable deep neural networks (xdnn). *Neural Networks*, 2020. doi: https://doi.org/10.1016/j.neunet.2020.07.010.

[7] T. Baklacioglu. Predicting the fuel flow rate of commercial aircraft via multilayer perceptron, radial basis function and anfis artificial neural networks. *The Aeronautical Journal*, 2020. doi: 10.1017/aer.2020.119.

[8] T. Balachandran, A. Thiago, M. Naloufi, S. Souihi, L. Françoise, and A. Janne. Iot and transfer learning based urban river quality prediction. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022. doi: 10.1109/GLOBECOM48099.2022.10001249.

[9] M. Barton and B. Lennox. Model stacking to improve prediction and variable importance robustness for soft sensor development. *Digital Chemical Engineering*, 2022. doi: https://doi.org/10.1016/j.dche.2022.100034.

[10] F. Bergamin, P. Moreno-Muñoz, S. Hauberg, and G. Arvanitidis. Riemannian laplace approximations for bayesian neural networks. *arXiv preprint*, 2023. doi: 10.48550/arXiv.2306.07158.

[11] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 2012. doi: 10.5555/2188385.2188395.

[12] M. Betancourt. A conceptual introduction to hamiltonian monte carlo. *arXiv e-prints*, 2018. doi: 10.48550/arXiv.1701.02434.

[13] Bishop, C.M. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN 978-0-387-31073-2.

[14] D.M. Blei, A. Kucukelbir, and J.D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 2017. doi: 10.1080/01621459.2017.1285773.

[15] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. *arXiv e-prints*, 2015. doi: 10.48550/arXiv.1505.05424.

[16] V. Borisov, T. Leemann, K. Sessler, J. Haug, M. Pawelczyk, and G. Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. doi: 10.1109/tnnls.2022.3229161.

[17] S. Bozinovski. Reminder of the first paper on transfer learning in neural networks, 1976. *Informatica (Slovenia)*, 2020. doi: 10.31449/inf.v44i3.2828.

[18] N. Boëly and R.M. Botez. New Approach for the Identification and Validation of a Nonlinear F/A-18 Model by Use of Neural Networks. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 2010. doi: 10.1109/TNN.2010.2071398.

[19] L. Breiman. Bagging predictors. *Machine learning*, 1996. doi: 10.1007/BF00058655.

[20] L. Breiman. *Classification and Regression Trees*. Wadsworth International Group, 1984. ISBN 9780534980535.

[21] L. Breiman. Random forests. *Machine Learning*, 2001. doi: 10.1023/A:1010950718922.

[22] L. Breinman. Stacked regressions. *Machine learning*, 1996. doi: 10.1007/BF00117832.

[23] M. D. Buhmann. *Radial Basis Functions: Theory and Implementations*. Cambridge University Press, 2003. doi: 10.1017/CBO9780511543241.

[24] R. Chandra and Y. He. Bayesian neural networks for stock price forecasting before and during covid-19 pandemic. *PloS one*, 2021. doi: 10.1371/journal.pone.0253217.

[25] Q. Chen, W. Zhang, and Y. Lou. Forecasting stock prices using a hybrid deep learning model integrating attention mechanism, multi-layer perceptron, and bidirectional long-short term memory neural network. *IEEE Access*, 2020. doi: 10.1109/ACCESS.2020.3004284.

[26] T. Chen and C. Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016. doi: 10.1145/2939672.2939785.

[27] F. Chollet. *Deep Learning with Python*. Manning Shelter Island, 2018. ISBN 9781617296864.

[28] R.B. Christianson, R.M. Pollyea, and R.B. Gramacy. Traditional kriging versus modern gaussian processes for large-scale mining data. *arXiv e-prints*, 2022. doi: 10.48550/arXiv.2207.10138.

[29] S. Cramb, E. Duncan, P. Baade, and K. Mengersen. *A Comparison of Bayesian Spatial Models for Cancer Incidence at a Small Area Level: Theory and Performance*, pages 245–274. Springer International Publishing, 2020. ISBN 978-3-030-42553-1. doi: 10.1007/978-3-030-42553-1_10.

[30] Z. Deng, F. Zhou, and J. Zhu. Accelerated linearized laplace approximation for bayesian deep learning. *arXiv e-prints*, 2022. doi: 10.48550/arXiv.2210.12642.

[31] S. Depeweg. *Modeling Epistemic and Aleatoric Uncertainty with Bayesian Neural Networks and Latent Variables*. PhD thesis, Technische Universität München, 2019.

[32] C. Di Francescomarino, M. Dumas, M. Federici, C. Ghidini, F.M. Maggi, W. Rizzi, and L. Simonetto. Genetic algorithms for hyperparameter optimization in predictive business process monitoring. *Information Systems*, 2018. doi: https://doi.org/10.1016/j.is.2018.01.003.

[33] H. Drucker. Improving regressors using boosting techniques. *Proceedings of the 14th International Conference on Machine Learning*, 1997.

[34] B. Efron and R. Tibshirani. Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, 92, 1997. doi: 10.2307/2965703.

[35] X. Fanhui, S. Lixin, and Y. Zhan. A container handling capacity prediction model based on rbf neural networks and its simulation. In *2007 IEEE International Conference on Control and Automation*, 2007. doi: 10.1109/ICCA.2007.4376529.

[36] A.Y.K Foong, D. Burt, Y. Li, and R. Turner. On the expressiveness of approximate inference in bayesian neural networks. *arXiv e-prints*, 2020. doi: 10.48550/arXiv.1909.00719.

[37] T. Franz, R. Zimmermann, S. Görtz, and N. Karcher. Interpolation-based reduced-order modelling for steady transonic flows via manifold learning. *International Journal of Computational Fluid Dynamics*, 2014. doi: 10.1080/10618562.2014.918695.

[38] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 1997. doi: https://doi.org/10.1006/jcss.1997.1504.

[39] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv e-prints*, 2016. doi: 10.48550/arXiv.1506.02142.

[40] J. Garcke and T. Vanck. Importance weighted inductive transfer learning for regression. In *Machine Learning and Knowledge Discovery in Databases*, pages 466–481. Springer Berlin Heidelberg, 2014. doi: https://doi.org/10.1007/978-3-662-44848-9_30.

[41] P.A. Gili and M. Battipede. Adaptive neurocontroller for a nonlinear combat aircraft model. *Journal of Guidance, Control, and Dynamics*, 2001. doi: 10.2514/2.4827.

[42] P.A. Gili and Ruotolo. A neural gust alleviator for a non-linear combat aircraft model. *Guidance, Navigation, and Control Conference*, 1997. doi: 10.2514/6.1997-3761.

[43] E. Goan and C. Fookes. *Bayesian Neural Networks: An Introduction and Survey*, pages 45–87. Springer International Publishing, 2020. ISBN 978-3-030-42553-1. doi: 10.1007/978-3-030-42553-1_3.

[44] F. Gomec and M. Canıbek. Aerodynamic database improvement of aircraft based on neural networks and genetic algorithms. In *7th European Conference For Aeronautics And Space Sciences*, 2017. doi: 10.13009/EUCASS2017-226.

[45] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016. ISBN 978-0262035613.

[46] L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on tabular data? *arXiv e-prints*, 2022. doi: 10.48550/arXiv.2207.08815.

[47] E. Haghighat, M. Raissi, A. Moure, H. Gomez, and R. Juanes. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 2021. doi: https://doi.org/10.1016/j.cma.2021.113741.

[48] J.M. Hernández-Lobato and R.P. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. *arXiv e-prints*, 2015. doi: 10.48550/arXiv.1502.05336.

[49] S. M. Hitzel and R. Osterhuber. Enhanced Manoeuvrability of a Delta Canard Combat Aircraft by Vortex Flow Control. *Aerospace Research Central*, 2017. doi: 10.2514/1.C034473.

[50] T.K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998. doi: 10.1109/34.709601.

[51] M. Hoffman, D.M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *arXiv e-prints*, 2013. doi: 10.48550/arXiv.1206.7051.

[52] M.D. Hoffman and A. Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *arXiv e-prints*, 2011. doi: 10.48550/arXiv.1111.4246.

[53] J.H Holland. Genetic algorithms. *Scientific american*, 1992. doi: www.jstor.org/stable/24939139.

[54] K. Hornik, M.B. Stinchcombe, and H.L. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989. doi: 10.1016/0893-6080(89)90020-8.

[55] H. Huang, H. Wang, Y. Li, L. Zhang, and Z. Liu. Support vector machine based estimation of remaining useful life: current research status and future trends. *Journal of Mechanical Science and Technology*, 2015. doi: 10.1007/s12206-014-1222-z.

[56] H. Jang and J. Lee. An empirical study on modeling and prediction of bitcoin prices with bayesian neural networks based on blockchain information. *IEEE Access*, 2018. doi: 10.1109/ACCESS.2017.2779181.

[57] L.V. Jospin, L. Laga, F. Boussaid, W. Buntine, and M. Bennamoun. Hands-on bayesian neural networks: A tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 2022. doi: 10.1109/mci.2022.3155327.

[58] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017.

[59] M.S. Khan and P. Coulibaly. Bayesian neural network for rainfall-runoff modeling. *Water Resources Research*, 2006. doi: https://doi.org/10.1029/2005WR003971.

[60] Y.T. Kim, B.Y. Kim, and S.W. Kim. Multi-level stacked regression for predicting electricity consumption of hot rolling mill. *Expert Systems with Applications*, 2022. doi: 10.1016/j.eswa.2022.117040.

[61] D.P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. *arXiv e-prints*, 2015. doi: 10.48550/arXiv.1506.02557.

[62] W. Kirch, editor. *Pearson's Correlation Coefficient*, pages 1090–1091. Springer, Dordrecht, 2008. ISBN 978-1-4020-5614-7. doi: 10.1007/978-1-4020-5614-7_2569.

[63] E.J.Y. Koh, E. Amini, G.J. McLachlan, and N. Beaton. Utilising a deep neural network as a surrogate model to approximate phenomenological models of a comminution circuit for faster simulations. *Minerals Engineering*, 2021. doi: https://doi.org/10.1016/j.mineng.2021.107026.

[64] D.G. Krige. A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of The South African Institute of Mining and Metallurgy*, 1951. doi: 10520/AJA0038223X_4858.

[65] A. Kristiadi, A. Immer, R. Eschenhagen, and V. Fortuin. Promises and pitfalls of the linearized laplace in bayesian optimization. *arXiv e-prints*, 2023. doi: 10.48550/arXiv.2304.08309.

[66] S. Kullback and R.A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 1951. doi: 10.1214/aoms/1177729694.

[67] J. Lampinen and A. Vehtari. Bayesian approach for neural networks—review and case studies. *Neural Networks*, 2001. doi: 10.1016/S0893-6080(00)00098-8.

[68] B. Li, P. Gao, S. Liang, and D. Chen. Intelligent flight control of combat aircraft based on autoencoder. In *Proceedings of the 2019 4th International Conference on Robotics, Control and Automation*, New York, NY, USA, 2019. Association for Computing Machinery. doi: 10.1145/3351180.3351210.

[69] M. Li, S. Li, Y. Tian, Y. Fu, Y. Pei, Y. Zhu, and Y. Ke. A deep learning convolutional neural network and multi-layer perceptron hybrid fusion model for predicting the mechanical properties of carbon fiber. *Materials and Design*, 2023. doi: https://doi.org/10.1016/j.matdes.2023.111760.

[70] Y. Li, J. Lang, L. Ji, J. Zhong, Z. Wang, Y. Guo, and S. He. Weather forecasting using ensemble of spatial-temporal attention network and multi-layer perceptron. *Asia-Pacific Journal of Atmospheric Sciences*, 2021. doi: 10.1007/s13143-020-00212-3.

[71] Y. Li, Y. Lu, and G. Zhang. An artificial-neural-network-based surrogate modeling workflow for reactive transport modeling. *Petroleum Research*, 2022. doi: 10.1016/j.ptlrs.2021.06.002.

[72] S. Liang and R. Srikant. Why deep neural networks for function approximation? *arXiv e-prints*, 2017. doi: 10.48550/arXiv.1610.04161.

[73] D. J. Linse and R. F. Stengel. Identification of aerodynamic coefficients using computational neural networks. *Journal of Guidance, Control, and Dynamics*, 1993. doi: 10.2514/3.21122.

[74] L. Liu, Y.S. Ong, X. Shen, and J. Cai. When gaussian process meets big data: A review of scalable gps. *arXiv e-prints*, 2019. doi: 10.48550/arXiv.1807.01065.

[75] M. Lu, Q. Hou, S. Qin, L. Zhou, D. Hua, X. Wang, and L. Cheng. A stacking ensemble model of various machine learning models for daily runoff forecasting. *Water*, 2023. doi: 10.3390/w15071265.

[76] J. McCall. Genetic algorithms for modelling and optimisation. *Journal of Computational and Applied Mathematics*, 2005. doi: https://doi.org/10.1016/j.cam.2004.07.034. Special Issue on Mathematics Applied to Immunology.

[77] R. L. McMillen, J. E. Steck, and K. Rokhsaz. Application of an artificial neural network as a flight test data estimator. *Journal of Aircraft*, 1995. doi: 10.2514/3.46840.

[78] A.P. Melo, D. Cóstola, R. Lamberts, and J.L.M. Hensen. Development of surrogate models using artificial neural network for building shell energy labelling. *Energy Policy*, 2014. doi: https://doi.org/10.1016/j.enpol.2014.02.001.

[79] S. Mondal, A. Chattopadhyay, A. Mukhopadhyay, and A. Ray. Transfer learning of deep neural networks for predicting thermoacoustic instabilities in combustion systems. *Energy and AI*, 2021. doi: https://doi.org/10.1016/j.egyai.2021.100085.

[80] J. Moreira, C. Soares, A. Jorge, and J. Sousa. Ensemble approaches for regression: A survey. *ACM Computing Surveys*, 2012. doi: 10.1145/2379776.2379786.

[81] G. Mounita, M. Sarker, A. Laboni, A. Kumar, A. Sulta, and M Mehedi. A Comparative Analysis of Machine Learning Algorithms to Predict Liver Disease. *Intelligent Automation and Soft Computing*, 2021. doi: 10.32604/iasc.2021.017989.

[82] S. Nguyen, D. Nguyen, K. Nguyen, K. Than, H. Bui, and N. Ho. Structured dropout variational inference for bayesian neural networks. *arXiv e-prints*, 2021. doi: 10.48550/arXiv.2102.07927.

[83] M. Norgaard, C.C Jorgenson, and C. J. Ross. Neural Network Predictions of New Aircraft Design Coefficients. *NASA Technical Memorandum*, 1997.

[84] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 2010. doi: 10.1109/TKDE.2009.191.

[85] J. Patterson and A. Gibson. *Deep Learning A Practitioner's Approach*. O'Reilly, 2018. ISBN 9781491914250.

[86] J. Pfrommer, C. Zimmerling, J. Liu, L. Kärger, F. Henning, and J. Beyerer. Optimisation of manufacturing process parameters using deep neural networks as surrogate models. *Procedia CIRP*, 2018. doi: https://doi.org/10.1016/j.procir.2018.03.046.

[87] L. Prokhorenkova, G. Gusev, A. Vorobev, A. Veronika Dorogush, and A. Gulin. Catboost: unbiased boosting with categorical features. *arXiv e-prints*, 2019. doi: 10.48550/arXiv.1706.09516.

[88] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. Springer, 2007. ISBN 978-1-4757-7394-1.

[89] N. Radford. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2012. doi: 10.1201/b10905-6.

[90] S. Raghavendra and P.C. Deka. Support vector machine applications in the field of hydrology: A review. *Applied Soft Computing*, 2014. doi: https://doi.org/10.1016/j.asoc.2014.02.002.

[91] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 2019. doi: 10.1016/j.jcp.2018.10.045.

[92] Z. Ramedani, M. Omid, A. Keyhani, S. Shamshirband, and B. Khoshnevisan. Potential of radial basis function based support vector regression for global solar radiation prediction. *Renewable and Sustainable Energy Reviews*, 2014. doi: 10.1016/j.rser.2014.07.108.

[93] M. Re and G. Valentini. Ensemble methods: A review. *Advances in machine learning and data mining for astronomy*, 2012. doi: 10.1201/B11822-34.

[94] O. Rivasplata, V.M. Tankasali, and S. Szepesvari. Pac-bayes with backprop. *arXiv e-prints*, 2019. doi: 10.48550/arXiv.1908.07380.

[95] P.C. Robert. The metropolis-hastings algorithm. *arXiv e-prints*, 2016. doi: 10.48550/arXiv.1504.01896.

[96] K. Rokhsaz and J. E. Steckt. Use of Neural Networks in Control of High-Alpha Maneuvers. *Journal Of Guidance, Control, and Dynamics*, 1993. doi: 10.2514/3.21104.

[97] J.C. Ross, C.C. Jorgenson, and M. Norgaard. Reducing Wind Tunnel Data Requirements Using Neural Networks. *NASA Technical Memorandum*, 1997.

[98] C. Sabater, P. Sturmer, and P. Bekemeyer. Fast predictions of aircraft aerodynamics using deep-learning techniques. *AIAA Journal*, 2022. doi: 10.2514/1.J061234.

[99] J. Sanwale and D.J. Singh. Aerodynamic parameters estimation using radial basis function neural partial differentiation method. *Defence science journal*, 2018. doi: 10.14429/dsj.68.11843.

[100] N. Secco and B. Mattos. Artificial neural networks to predict aerodynamic coefficients of transport airplanes. *Aircraft Engineering and Aerospace Technology*, 2022. doi: 10.2514/6.2015-1013.

[101] V. Sekar, Q. Jiang, C. Shu, and B.C. Khoo. Fast flow field prediction over airfoils using deep learning approach. *Physics of Fluids*, 2019. doi: 10.1063/1.5094943.

[102] J Soch. The book of statistical proofs: Kullback-leibler divergence for the normal distribution, 2019. URL URL:https://statproofbook.github.io/P/norm-kl.html#:~:text=P%3AX%E2%88%BCN(%CE%BC,(1)&text=KL%5BP%7C%7CQ,%CF%8322%E2%88%921%5D.

[103] B. Staber and S. Da Veiga. Benchmarking bayesian neural networks and evaluation metrics for regression tasks. *arXiv e-prints*, 2023. doi: 10.48550/arXiv.2206.06779.

[104] S. Suresh and N. Kannan. Direct adaptive neural flight control system for an unstable unmanned aircraft. *Applied Soft Computing*, 2008. doi: https://doi.org/10.1016/j.asoc.2007.07.009.

[105] S. Suresh, S.N. Omkar, and Mani V. Nonlinear Adaptive Neural Controller for Unstable Aircraft. *Journal of Guidance Control and Dynamics*, 2005. doi: 10.2514/1.12974.

[106] S. Sutrisno, T.A. Rohmat, S.B. Wibowo, and S. Iswahyudi. Vortex dynamics study of the canard deflection angles' influence on the sukhoi su-30-like model to improve stall delays at high aoa. *Aerospace*, 2019. doi: 10.3390/aerospace6020012.

[107] A. Thomas, P. Wu, N. M. White, L. Toms, G. Mellick, and K. L. Mengersen. *An Ensemble Approach to Modelling the Combined Effect of Risk Factors on Age at Parkinson's Disease Onset*, pages 275–302. Springer International Publishing, Cham, 2020. ISBN 978-3-030-42553-1. doi: 10.1007/978-3-030-42553-1_11.

[108] A.A. Trani, T. Wing-Ho, Schilling G., H. Baik, and Seshadri A. A Neural Network Model to Estimate Aircraft Fuel Consumption. *AIAA 4th Aviation Technology, Integration and Operations (ATIO Forum)*, 2004. doi: 10.2514/6.2004-6401.

[109] M.A. Vega and M.D. Todd. A variational bayesian neural network for structural health monitoring and cost-informed decision-making in miter gates. *Structural Health Monitoring*, 2022. doi: 10.1177/1475921720904543.

[110] M.J. Wainwright and M.I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 2008. doi: 10.1561/2200000001.

[111] R. Wallach, B. Mattos, R. Girardi, and M. Curvo. Aerodynamic coefficient prediction of a general transport aircraft using neural network. *AIAA*, 2006. doi: 10.2514/6.2006-658.

[112] J. Wang. An intuitive tutorial to gaussian processes regression. *arXiv e-prints*, 2022. doi: 10.48550/arXiv.2009.10862.

[113] J. Wang, L. Li, N. Dongxiao, and T. Zhongfu. An annual load forecasting model based on support vector regression with differential evolution algorithm. *Applied Energy*, 2012. doi: 10.1016/j.apenergy.2012.01.010.

[114] Q. Wang, W. Zhou, L. Yang, and K. Huang. Comparison between conventional and deep learning-based surrogate models in predicting convective heat transfer performance of u-bend channels. *Energy and AI*, 2022. doi: 10.1016/j.egyai.2022.100140.

[115] Davies W.G., S. Babamohammadi, Y. Yang, and S.M. Soltani. The rise of the machines: A state-of-the-art technical review on process modelling and machine learning within hydrogen production with carbon capture. *Gas Science and Engineering*, 2023. doi: 10.1016/j.jgsce.2023.205104.

[116] D.A. White, W.J. Arrighi, J. Kudo, and S.E. Watts. Multiscale topology optimization using neural network surrogate models. *Computer Methods in Applied Mechanics and Engineering*, 2019. doi: 10.1016/j.cma.2018.09.007.

[117] D.H. Wolpert. Stacked generalization. *Neural Networks*, 1992. doi: 10.1016/S0893-6080(05)80023-1.

[118] S. Xie, Y. Li, H. Tan, R. Liu, and F. Zhang. Multi-scale and multi-layer perceptron hybrid method for bearings fault diagnosis. *International Journal of Mechanical Sciences*, 2022. doi: https://doi.org/10.1016/j.ijmecsci.2022.107708.

[119] T. Xie, H. Yu, and B. Wilamowski. Comparison between traditional neural networks and radial basis function networks. In *2011 IEEE International Symposium on Industrial Electronics*, 2011. doi: 10.1109/ISIE.2011.5984328.

[120] Y. Xie, D. Lord, and Y. Zhang. Predicting motor vehicle collisions using bayesian neural network models: An empirical analysis. *Accident Analysis & Prevention*, 2007. doi: https://doi.org/10.1016/j.aap.2006.12.014.

[121] S. Xiong. The reconstruction approach: From interpolation to regression. *Technometrics*, 2021. doi: 10.1080/00401706.2020.1764869.

[122] Z. Yang, A. Zhang, and A. Sudjianto. Enhancing explainability of neural networks through architecture constraints. *IEEE Transactions on Neural Networks and Learning Systems*, 2021. doi: 10.1109/TNNLS.2020.3007259.

[123] S. Young, D. Rose, T. Karnowski, S. Lim, and R. Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, 2015. doi: 10.1145/2834892.2834896.

[124] X. Zhan, Y. Qin, H.and Liu, L. Yao, W. Xie, G. Liu, and J. Zhou. Variational bayesian neural network for ensemble flood forecasting. *Water*, 2020. doi: 10.3390/w12102740.

[125] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning, 2020.

Appendix A

Tree-Based Models

In this chapter, the appendix for the tree-based models is provided. The appendix is split up into two sections. Firstly, the hyperparameter optimisation of the tree-based models is provided in section A.1. Secondly, the feature importance of the optimised models is provided in section A.2.

A.1 Tree-Based Models Hyperparameter Optimisation

In this subsection, the hyperparameters of the Random Forest, AdaBoost, XGBoost, lightGBM, and CatBoost models are optimised.

A.1.1 Random Forest

In Table A.1, the optimisation for the Random Forest model is provided. In this model the number of decision trees which are made by the model are varied. There are more parameters which can be varied such as the maximum depth of each tree, the minimum samples required to perform a split in the decision tree, and the minimum number of samples per leaf. However, for this optimisation, the parameters are specified to split the decision tree until only 1 sample is present in a leaf, and two samples are required to split a node. Any different combination for this parameter did not result in an improvement in the performance. One can furthermore see, that increasing the number of trees in the Random Forest merely increases the training time, and does not improve the MSE of the test dataset. Here the optimisation resulted in almost no decrease in MSE. Therefore,

the optimal value is kept at 100 estimators, meaning 100 different trees in the Random Forest.

Table A.1: Random Forest optimization of MSE and training duration for aerodynamic dataset 1.

Iteration	Estimators	Time (s)	MSE
1	500	35.9	4.8e-5
2	250	18.6	4.8e-5
3	100	8.1	4.8e-5

A.1.2 AdaBoost

In Table A.2, the optimisation of the AdaBoost model is performed. As can be seen in the table, the hyperparameters of the model are the learning rate and number of estimators. Also provided is the time it took to train the model, and the MSE of the test data of the aerodynamic coefficient being used to optimise the model. What can be seen is that the initial performance of the model is incredibly low, as the MSE of this model is almost two orders of magnitude larger than the Random Forest model shown in Table A.1. Furthermore, the training time is on the first iteration already three times higher than the Random Forest model. Nevertheless, the number of estimators is increased by a factor of 10. This increases the training time to over 10 minutes and only marginally increases the MSE to a level which is still unacceptable. If one just quickly looks at the percentage within the tolerance metric that is introduced in subsection 2.2.3, one can see that the AdaBoost model has a percentage within the tolerance of 15%, which is low when the most basic model, the Random Forest has 68%. Therefore, the optimisation is stopped and not continued further, since optimising a model that is not capable of predicting the dataset is not useful.

Table A.2: AdaBoost optimization of MSE and training duration for aerodynamic dataset 1.

Iteration	Learning Rate	Estimators	Time (s)	MSE
1	0.1	200	98.7	1.8e-3
2	0.1	2000	737.7	1.56e-3

A.1.3 XGBoost

In Table A.3, the results for the optimisation of the XGBoost model are shown. The XGBoost has as its most important hyperparameters the number of estimators, the learning rate, and the depth of each of the trees. What can be seen from the optimisation is that increasing the estimators increases the performance up to a certain point.

However, when comparing the best iteration, number 4 and the second best number 8, the parameters are equal besides the estimators, and their performance is incredibly similar. Increasing the tree depth also increased the performance, and a tree depth of 15 was shown to be a good candidate. Finally, the learning rate was kept lower at 0.1, as increasing the learning rate to 0.3 did not increase the performance of the model. The optimisation reduced the MSE by around 20% compared to the initial parameters, and the optimal performance of the 8th run is kept as the best parameter combination as the difference in MSE is not significant between the 4th and 8th run. With a lower number of iterations the training time is improved, and likely since the pitching moment coefficient is the most nonlinear, the extra iterations will not be a good reason to select this option since the complexity can only be lower for the other coefficients.

Table A.3: XGBoost model optimization of MSE and training duration for aerodynamic dataset 1.

Iteration	Estimators	Learning Rate	Tree Depth	Time (s)	MSE
1	150	0.1	9	24.4	1.48e-5
2	300	0.1	9	51.7	1.23e-5
3	250	0.1	9	40.4	1.24e-5
4	250	0.1	15	38.1	1.15e-5
5	250	0.1	25	48.3	1.18e-5
6	250	0.3	15	24.2	1.67e-5
7	250	0.1	6	29.2	2.56e-5
8	100	0.1	15	31.4	1.16e-5

A.1.4 LightGBM

In Table A.4, the LightGBM model optimisation results are shown. LightGBM has as its most important hyperparameters, the number of leaves that can form at each tree, the learning rate, and the number of rounds it is trained for, or in other words how many trees are formed. From the optimisation, one can see that the model performs much better than XGBoost and the Random Forest. Finally, one can also see that from the initial parameters, the learning rate is kept at 0.1, and the number of rounds the model is trained, and the number of leaves in each of the trees, has been increased to 2000 and 100 respectively. Resulting in a 15% decrease in MSE compared to before the optimisation.

A.1.5 CatBoost

Finally, in Table A.5 the optimisation of the CatBoost algorithm is provided. The most important hyperparameters are again the number of estimators, or how many

Table A.4: LightGBM model optimization of MSE and training duration for aerodynamic dataset 1.

Iteration	Number Leaves	Learning Rate	Rounds	Time (s)	MSE
1	63	0.1	1000	22.2	1.05e-5
2	63	0.3	1000	19.4	1.35e-5
3	63	0.1	1500	29.7	9.57e-6
4	63	0.1	2000	39.9	9.08e-6
5	31	0.1	2000	35.5	9.64e-6
6	100	0.1	2000	43.5	9.06e-6

trees will be constructed sequentially, the learning rate, and the individual tree depth. Immediately, one can see that the initial starting point of CatBoost, is already better than any of the other models, and the final best model has impressively low MSE with the number of rounds being doubled from 2000 to 4000, the learning rate kept constant, and finally the depth of the tree being increased from 6 to 9.

Table A.5: CatBoost model optimization of MSE and training duration for for aerodynamic dataset 1.

Iteration	Rounds	Learning Rate	Depth	Time (s)	MSE
1	2000	0.1	6	20.9	7.75e-6
2	4000	0.1	6	42.5	6.07e-6
3	4000	0.3	6	45.1	5.27e-6
4	4000	0.3	9	64.7	4.98e-6
5	4000	0.1	9	58.4	4.40e-6
6	4000	0.1	15	112.1	1.07e-5

A.1.6 Optimisation Algorithm for CatBoost

In this subsection, a retrospective experiment is applied, where instead of manually optimising the CatBoost model, an optimisation algorithm is used. This could give insights into whether the assumption to perform a manual optimisation for the hyperparameters of the tree-based models was valid.

To do this, the optimisation package Optuna is used [3]. Optuna is a popular hyperparameter optimisation package in Python and will be used to optimise the hyperparameters of the CatBoost model. The random sampler is taken as the method for providing samples for the hyperparameters, 100 trials are made using the Optuna package, with each trial representing the average MSE of a 5-fold split. The optimised hyperparameters from the CatBoost model are 4666 iterations, a depth of 10, and finally, a learning rate of 0.116, providing a MSE of 4.11e-6. This means that the Optuna optimisation tool did find around 10% additional reduction in MSE for the CatBoost model, at the

cost of 500 trained models, and therefore around 7 hours of total training time. This means that with 6 tries and a total of 30 minutes of training time, only a 10% penalty is paid. Furthermore, the fear of using an optimiser for the hyperparameters is that the model is tuned perfectly for the pitching moment coefficient which is being optimised currently but does not generalise well towards the other aerodynamic coefficients. A positive benefit of using Optuna is that the process is automated. Therefore, extra time could be offset by the fact that it is automated and can be done overnight, for several aerodynamic coefficients.

One can conclude from this experiment that there is still performance improvement to be found in each of the models that have been manually optimised. The cost however of finding this extra performance is extremely high, a factor of 14 in computational time, while the performance in terms of MSE has only gone down by 10%. Therefore, the initial assumption of performing manual optimisation which would result in only a small performance loss compared to using an optimiser is valid, and the manual optimisation parameters and values are kept for consistency.

A.1.7 Convergence Plots of Optimised Models

In this subsection, the convergence plots for the CatBoost, lightGBM, and XGBoost models are provided in Figure A.1, Figure A.2, and Figure A.3 respectively. The plots show the convergence of the MSE loss of the model on the test and train data.

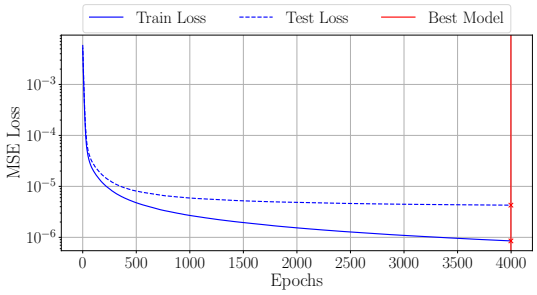


Figure A.1: CatBoost model convergence of the optimised model.

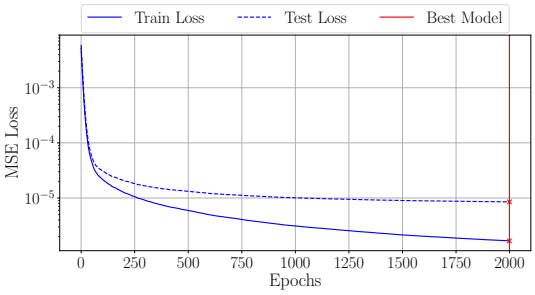


Figure A.2: LightGBM model convergence of the optimised model.

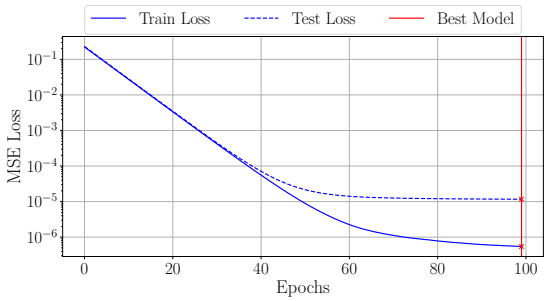


Figure A.3: XGBoost model convergence of the optimised model.

A.2 Feature Importance for Random Forest, XGBoost, and LightGBM

In this section, the feature importance of the Random Forest, XGBoost, and LightGBM models are provided. The CatBoost feature importance is shown in section 4.4, and therefore not repeated in this section. The feature importance of the remaining models which are considered in this thesis are provided here.

From Table A.6, the feature importance of the Random Forest model can be seen for each of the aerodynamic coefficients, and input parameters of the models. One can see that also for the Random Forest model the physical features of the aircraft are learnt by the model.

Table A.6: Feature importance for the Random Forest model for all the aerodynamic coefficients.

	α	β	M	η	ε	δ_1	δ_3	ζ
C_m	11.818	0.319	8.315	10.308	0.926	37.454	30.819	0.04
C_l	10.52	10.28	1.394	0.5	0.447	27.713	48.534	0.612
C_n	8.178	30.756	2.541	0.796	0.199	5.691	7.258	44.58
C_x	68.845	0.111	6.361	3.718	9.680	3.721	7.282	0.282
C_y	4.778	71.197	1.421	0.269	0.114	4.381	4.124	13.716
C_z	94.384	0.029	0.782	0.057	0.028	2.671	2.048	0.001

In Table A.7, the feature importance of the XGBoost model for all the aerodynamic coefficients, and input values are shown. What can be seen is that the relative importance of each input feature has a far greater difference than the Random Forest model. This could be because the importance has been rescaled to sum up to 100, such that they can be compared to the feature importance of the other models.

Finally, in Table A.8, the feature importance of the LightGBM model for each of the

Table A.7: Feature importance for the XGBoost model for all the aerodynamic coefficients.

	α	β	M	η	ε	δ_1	δ_3	ζ
C_m	0.890	0.021	0.667	4.587	0.291	30.608	62.851	0.084
C_l	0.541	0.627	0.040	0.060	0.089	11.498	86.554	0.590
C_n	0.289	1.572	0.057	0.052	0.028	0.951	3.190	93.861
C_x	33.169	0.020	2.567	3.915	36.249	5.989	17.173	0.917
C_y	0.489	11.619	0.100	0.064	0.098	2.394	7.477	77.757
C_z	65.275	0.016	0.500	0.136	0.113	14.814	19.135	0.013

aerodynamic coefficients and input features is provided. What can be seen from the columns of Table A.8, is that the feature importances for each of the input features do not vary much.

Table A.8: Feature importance for the LightGBM model for all the aerodynamic coefficients.

	α	β	M	η	ε	δ_1	δ_3	ζ
C_m	25.544	15.457	25.959	12.219	4.752	9.392	5.776	0.902
C_l	25.744	20.061	26.155	9.376	5.281	8.566	3.373	1.444
C_n	23.833	19.878	26.874	10.736	4.489	8.146	3.826	2.218
C_x	26.724	13.596	25.110	11.458	5.869	9.532	6.349	1.361
C_y	24.64	18.698	29.932	9.376	4.398	7.969	3.181	1.806
C_z	27.900	14.958	26.831	7.942	4.710	9.989	6.887	0.784

Appendix B

Artificial Neural Networks

In this chapter, the appendix of the artificial neural networks (ANNs) is provided. This is an appendix of the artificial neural network chapter seen in chapter 5. This chapter is split up into three parts. Firstly, the hyperparameter optimisation of the ANNs is provided in section B.1. Secondly, the genetic optimisation that is performed on a two-dimensional (2D), and three-dimensional (3D) test function to test the workings of the algorithm is provided in section B.2. Next, the convergence of the MSE loss of the test and train dataset is provided in section B.3. Finally, example plots of the predictions of the ANN on the test dataset are provided in section B.4.

B.1 Hyperparameter Optimisation

In this section, a detailed optimisation is performed of the categorical hyperparameters of the artificial neural network. The categorical hyperparameters include each hyperparameter of the neural network besides the number of layers in the network, and how many neurons they contain respectively. Firstly, the specifications of the model and the convergence comparison which is used to determine the number of folds required for performing k-fold cross-validation are shown in subsection B.1.1. The hyperparameters discussed thus include the batch size in subsection B.1.2, the choice of learning rate in subsection B.1.3, whether one should use batch normalisation in subsection B.1.4, the different activation functions in subsection B.1.5, the use of dropout in subsection B.1.6, different forms of regularization, optimisation functions, number of epochs, and loss functions are discussed in subsection B.1.7.

B.1.1 K-fold Cross-validation

In this subsection, the k-fold split cross-validation technique is explained and discussed, as it will play a vital role in comparing the different hyperparameter values in the coming sections. In this subsection, a comparison is made between the 5-fold versus 10-fold cross-validation split. Afterwards, the convergence of the 5-fold, and 10-fold cross-validation is provided in Figure B.1.

K-fold cross-validation is a technique applied to be able to have a more robust measure of a machine learning model. The full dataset is divided into k groups, where 1 of the groups is used for test data in evaluating the machine learning model, and $k-1$ is used for training the machine learning model. This method can give a better estimate of the model performance than a single test-train split since it averages the effect of choosing a particular training-test dataset combination that might be favourable for the model performance.

To choose an appropriate value for k one can look at the literature where usually a value of 5 or 10 is taken [85]. Choosing a value of 5 for k keeps the computational cost lower while it should have more variance in the predictive results. Furthermore, the reason why $k = 5$ is such a popular choice is that it splits the data into 80% training data and 20% test data. A 10 k-fold cross validation, is computationally much more demanding, while also splitting the data into 90% training and 10% testing data.

The hypothesis to be tested during this subsection is as follows: *The 5-fold cross-validation is a more cost-effective cross-validation method*, with the cost being a linear combination of computational time and variance in the solution. To test this hypothesis, a simulation is set up, where 5-fold and 10-fold cross-validation is performed with an artificial neural network on the aerodynamic dataset. The specifications of the neural network being used for this task can be found in Table B.2. Furthermore, the convergence of each of the models which are trained in the 5-fold, and 10-fold cross-validation can be seen in Figure B.1.

Table B.1 shows the mean and standard deviation of the MSE on the test dataset of the 10-fold and 5-fold. On average the MSE of the 10-fold is around 10% lower than that of the 5-fold. However, the deviation of the MSE of the test data, or in other words the standard deviation is 80% lower for the 5-fold compared to the 10-fold. Therefore, the 5-fold cross-validation provides a more consistent estimate of the test loss. Considering that the total training time of the 10-fold cross-validation is double that of the 5-fold, while the model performance is only 16% better, with 80% higher variability, the choice is made to perform 5-fold validation for any further hyperparameter selection.

Table B.1: Mean and standard deviation (STD) of the 5-fold and 10-fold cross-validation mean square error (MSE) on the test dataset.

	Mean of Test MSE	**STD of Test MSE**
5-Fold	8.17e-6	6.59e-7
10-Fold	7.34e-6	1.21e-6

Table B.2: Specifications of the artificial neural network being used for 5-fold and 10-fold cross-validation.

Data and Model Parameter	**Value**
Number of Inputs	8
Number of Outputs	1
Number of Training Points	348 000
Number of Training Polars	1226
Number of Testing Points	88 000
Number of Training Polars	307
Batch Normalisation	No
Dropout (same for each layer)	0
Activation Function	ReLU
Batch Size	8192
Learning Rate	1e-4
Optimizer	ADAM
Loss Function	MSE
Epochs	150
Number of Layers	5
Number of Neurons per Layer	512

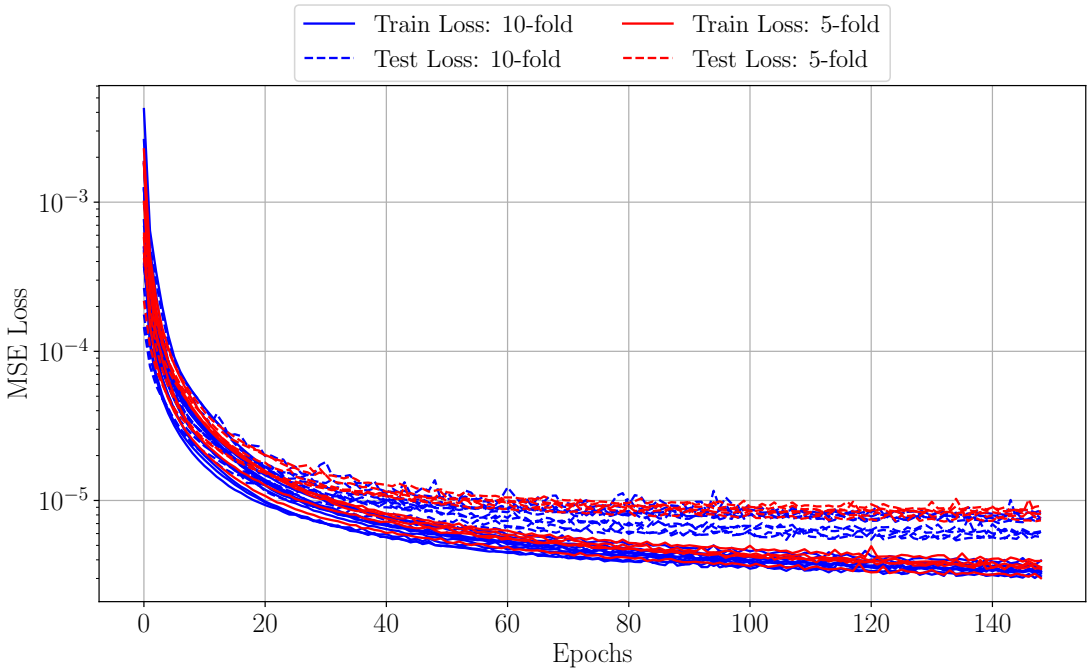


Figure B.1: Convergence of 10-fold and 5-fold cross-validation plot for identical artificial neural networks.

B.1.2 Comparison Batch Sizes

In this subsection, a deeper look is taken at the batch sizes that are taken for training a neural network. A typical size could be anywhere between 16 and 512, however, usually a rule of thumb suggestion is 32 [85]. In the situation presented during the thesis, where as described in section 2.2, the data is split polar by polar. If a batch size of 32 is taken, for a total dataset size of for example 1000 polars, each containing around 200 points, for a total of 200 000 data points for training, a batch size of 32 does not even contain a single point from each of the polars, and therefore it is thought, that this is not a good representation of the entire dataset. In contrast, when a batch size of for example 2000 is taken, then only 2 points out of each polar is taken, which could also be insufficient. Therefore, the hypothesis for this subsection is that *A batch size of around ten thousand is not only required for stable convergence but also optimal in training a neural network for the aerodynamic dataset.* This will be tested in this section by running a model with different batch sizes, and with other parameters constant to only evaluate the effect of the batch size.

In Figure B.2, and Figure B.3 two plots are shown with the convergence of 10 models, each with a different and increasing batch size. Each model is cross-validated using a 5-fold, and the mean value of the convergence is displayed in the figure. The batch size varies from 32 up to 16384, with the dataset that is being used consisting of a total of 290,000 data points, distributed over 1018 polars. 824 of them were used for training the model, equating to around 230,000 training data points. This is the dataset 1 described in section 2.2. What can be seen from Figure B.2, and Figure B.3 is that the models with a smaller batch size, in particular, 128, 256, and 512 are performing the best in terms of mean squared error, and have the least amount of variation in the training and test loss, meaning they are more robust in their convergence. When the batch size is smaller than 128, the test loss starts to become higher on average. Furthermore, the models using a smaller batch size reach the optimum value faster, it is almost like the smaller batch size models are using a higher learning rate, even though they are not. The models which perform the worst, and have the most varying convergence, are the ones with the largest batch sizes. In particular, a batch size of 8192, and 16384, which are taking a longer time to converge and vary more during training. What can therefore be concluded is that the hypothesis of requiring a large batch size to have a complete representation of the aerodynamic dataset is not true. The smaller batch sizes are capable of converging on average to a lower value of the mean squared error on the test dataset than the ones with a larger batch size.

B.1.3 Learning Rate

In this subsection, the optimal learning rate is analysed and discussed.

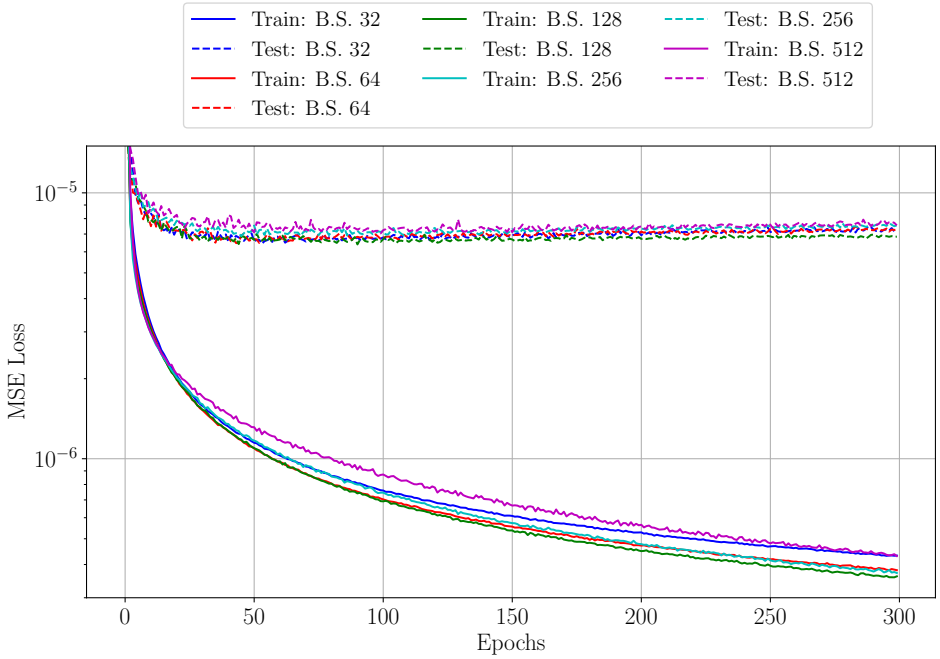


Figure B.2: Mean values of the train and test loss of a 5-fold cross-validation of an artificial neural network with varying batch size (B.S.) from 32 to 512

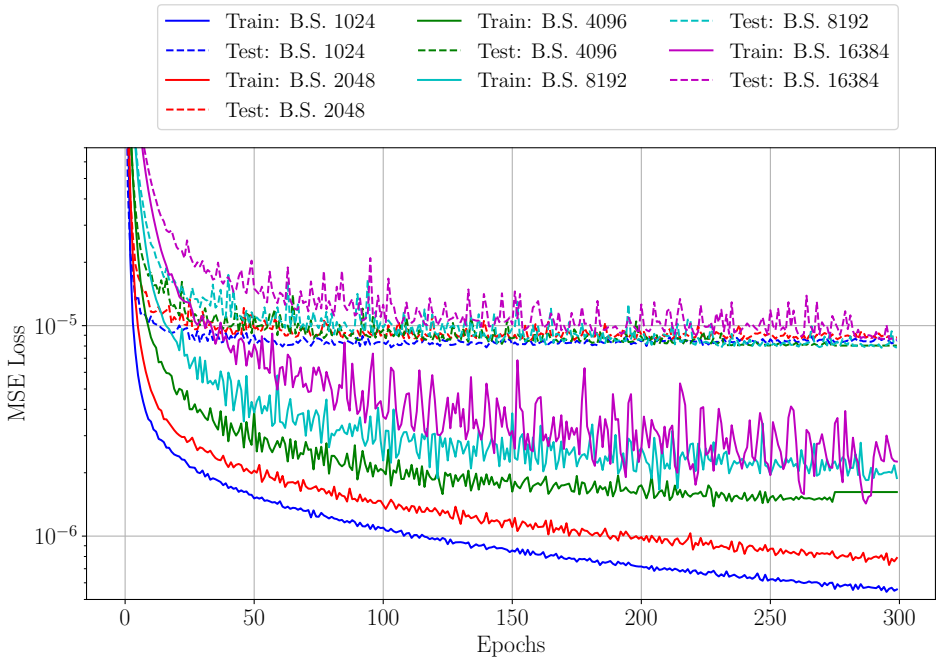


Figure B.3: Mean values of the train and test loss of 5-fold cross-validation of an artificial neural network with varying batch size (B.S.) from 1024 to 16384.

The learning rate is an important parameter for controlling the size of the updates that are taken in the gradient descent algorithm used to optimise the weights of the neural network. Selecting a too-low learning rate will require many steps to reach the optimum value. Choosing a too-high value for the learning rate might result in failed convergence, as the algorithm overshoots the optimum. Therefore, the right value needs to be found for the learning rate, which not only allows for reduced computational time but also possibly a lower MSE on the test dataset.

In Figure B.4, a comparison is made between a neural network with varying learning rates. The learning rates that have been run vary from 1e-2 to 1e-5. In Figure B.4, one can see that the learning rate of 1e-2 was not able to converge. A learning rate of 1e-3 was capable of converging towards an optimal value of the hyperparameters. However, the learning rate of 1e-4 resulted in a lower MSE on the test dataset on average. Furthermore, the lower learning rate has a steeper convergence slope and is the better choice between the two. Finally, one can compare the case where the learning rate is 1e-4 and 1e-5. Here the lower learning rate is starting to show that it will take a longer time to converge, as the slope of the convergence of the training loss is lower than that for the learning rate of 1e-4. Therefore, one can conclude that an optimal value of 1e-4 can be taken as the learning rate of the neural network.

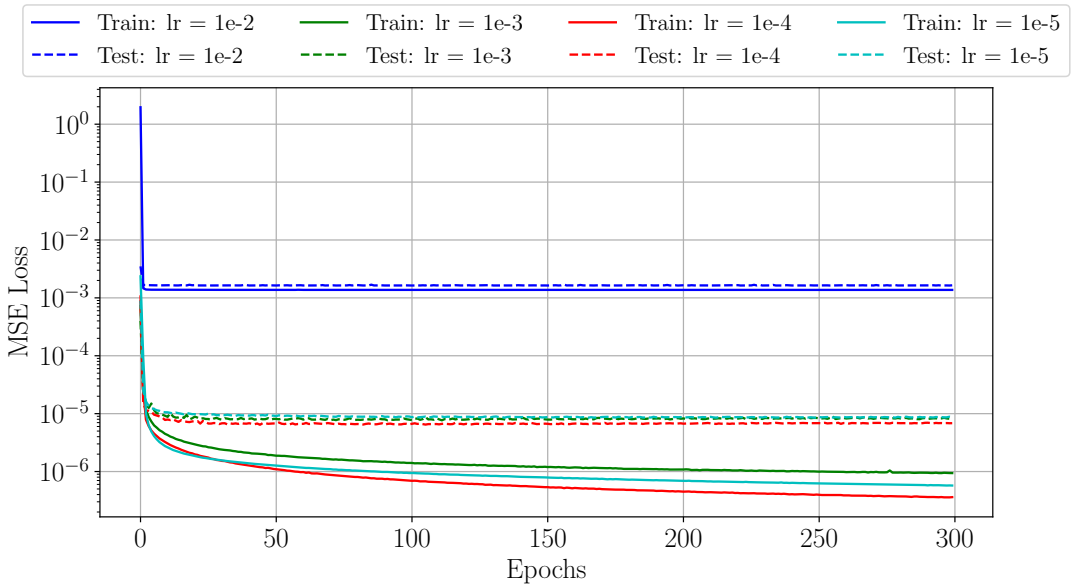


Figure B.4: Mean values of the train and test loss of a 5-fold cross-validation of an artificial neural network with varying learning rates (lr) from 1e-2 to 1e-4.

In the case of the model becoming unstable at a higher number of epochs, one can use a learning rate decay, to lower the learning rate with an increasing number of epochs. In

Equation B.1, one can see how this equation looks like, where the learning rate is scaled with the inverse of the learning rate decay rate multiplied by the number of epochs. In this manner, one could, for example, lower the learning rate from the determined optimal value of 1e-4, towards 1e-5 in 100 epochs, by using a learning rate decay rate of 0.09.

$$LR_{new} = LR_{old} * \frac{1}{1 + LR_{decay_{rate}} * epoch} \tag{B.1}$$

B.1.4 Batch Normalisation

In this subsection, an artificial neural network with batch normalisation and without batch normalisation are trained and compared against each other. Furthermore, a small investigation is made on whether a model with batch normalisation can allow for increased learning rates.

From the literature study in section 3.3, batch normalisation is known for a few things. Firstly, stabilising the learning of the neural network allows for higher learning rates. Therefore, the hypothesis of this section is: *Batch normalisation allows for faster convergence due to the possibility of using larger learning rates.* To test this, one needs to know the maximum learning rate this neural network could handle without batch normalisation, and determine whether a model with batch normalisation would be able to converge to the correct solution. However, before doing this one could look at a model using a learning rate of 1e-4, which was determined to be optimal in subsection B.1.3, and see what the effect of using batch normalisation is. This is what is shown in Figure B.5, where a comparison is made between a model with and without batch normalisation. The figure shows the average MSE loss of the 5-fold cross-validation which is performed. From this, it can be seen that the difference between batch normalisation and no batch normalisation does not have a significant impact on the convergence speed when the same learning rate is used. One can see that the training, and test loss of a model using batch normalisation is slower than that of a model not using it. This could give an idea that a higher learning rate could be used on the model with batch normalisation, as the effect of using batch normalisation is to lower the apparent rate of convergence.

From subsection B.1.3 on the learning rate of the neural network, the maximum learning rate that was tested was 1e-2, which resulted in the model diverging and not reaching the proper solution. Therefore, now a comparison can be made whether batch normalisation allows one to use this learning rate to train the neural network. A model with a learning rate of 1e-2 is run with batch normalization. What could be deduced is that the model using batch normalisation was not able to converge either. This means that now, one must try to determine the point where the model without batch normalisation does not converge, and the one with batch normalisation would converge. However, this does not seem like an intuitive way of testing the hypothesis. The model which was tested

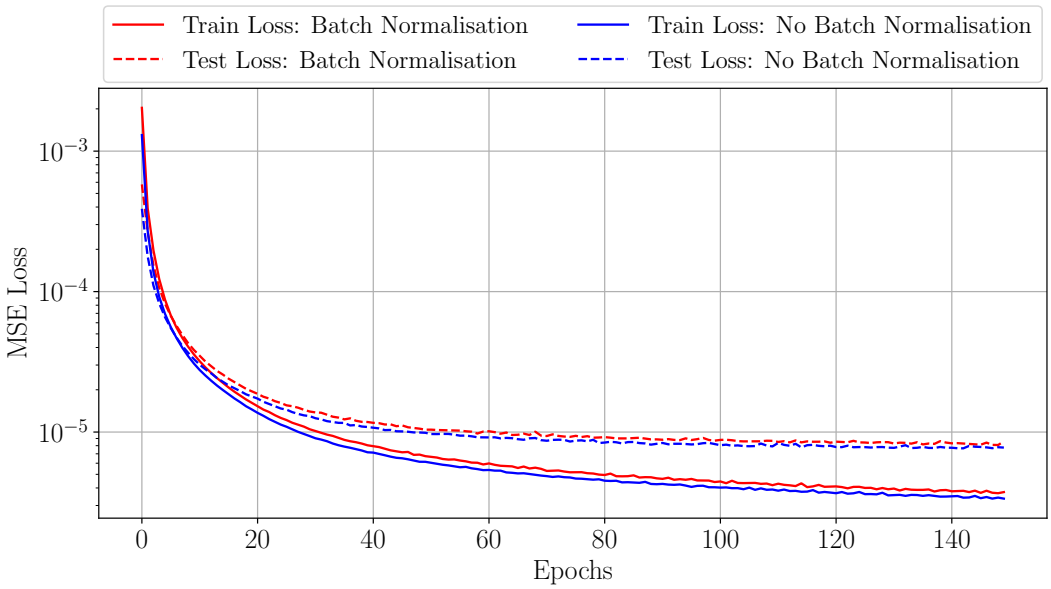


Figure B.5: Comparison of a 5-fold cross-validation performed for an artificial neural network with and without batch normalisation.

with and without batch normalisation, showed a lower convergence speed for the model with batch normalisation. Therefore, it was concluded that this could indicate that the learning rate could be increased slightly, to obtain the same convergence speed as the model without batch normalisation. For this conflicting reason, batch normalisation is not utilised for the ANN as the benefit is not immediately evident, and for utilising the supposed benefit of batch normalisation one must first know what the maximum learning rate would be, which takes significant resources.

B.1.5 Activation Functions

In this subsection, a deeper look is taken at the activation functions that are used in the neurons of the neural network, and what effect they have on the neural network.

The activation function, as shown in Equation 3.2, is a non-linear function that works on the neurons of the neural network. The activation functions that are tested in Figure B.6 are the Sigmoid, hyperbolic tangent, ReLU, and variations of the ReLU function such as SELU, LeakyReLU, PReLU, and GELU. These are chosen based on their popularity and general usability in deep neural networks [85]. The hyperbolic tangent (Tanh), and Sigmoid have the same shape except that Tanh scales between -1 and 1, while the

Sigmoid function scales between 0 and 1. The ReLU function is 0 for values smaller than 0 and equal to the input for values larger than 0. LeakyReLU scales negative values to a smaller value, such that negative values are not instantly turned to 0. This helps alleviate the dying ReLU problem. SELU, or scaled exponential linear unit is identical to ReLU, except for negative values, where an exponential function is used. It is known for its self-normalising properties, where the activation outputs converge towards zero mean and unit variance, which could help stabilise the convergence. PReLU, or parametric rectified linear unit is identical to leaky ReLU, besides the fact that the slope of the negative values is no longer a fixed hyperparameter, but is also learned during training. Finally, GELU, or Gaussian error linear unit is an activation function taking on a similar form as ReLU, however, the transition point between values smaller and larger than 0, is now a smooth Gaussian function.

In Figure B.6 the convergence of the activation functions is shown. This allows for the comparison of all the activation functions. Firstly, some obvious observations are that the Sigmoid activation function is not capable of providing a good convergence of the model. Other activation functions which do not result in good convergence are SELU, GELU, and Tanh. Finally, the competitive activation functions are ReLU, LeakyReLU, and PReLU, with ReLU and LeakyReLU, having almost the same convergence pattern. Due to the extra hyperparameter of LeakyReLU, it is decided to choose ReLU as the activation function of the model.

B.1.6 Dropout

In this section, the dropout variable is discussed and evaluated whether having dropout is beneficial for the neural network.

Dropout is the concept of randomly excluding a given neuron from the training of a neural network. The amount of dropout is characterised by the probability of a neuron being excluded from training, referred to as p. In the literature, often a dropout value of anything between 0 and 0.5 is suggested [85]. Dropout reduces the probability of the neural network relying too heavily on a set of neurons for training and promotes all neurons to contribute towards the final prediction of the model. The hypothesis for this section is: *Dropout improves the performance in terms of MSE of the neural network.* To test this hypothesis, a neural network is trained with varying levels of dropout, while keeping the other parameters constant. From this, the effect of dropout can be evaluated and the hypothesis can be evaluated.

In Figure B.7, the 5 neural networks are shown that are run with a varying dropout level. What can be seen from the figure is that models with increasing dropout, perform very slightly better on average on the test dataset. Meaning that a higher dropout value is beneficial for the performance of machine learning. It can be seen that the initial

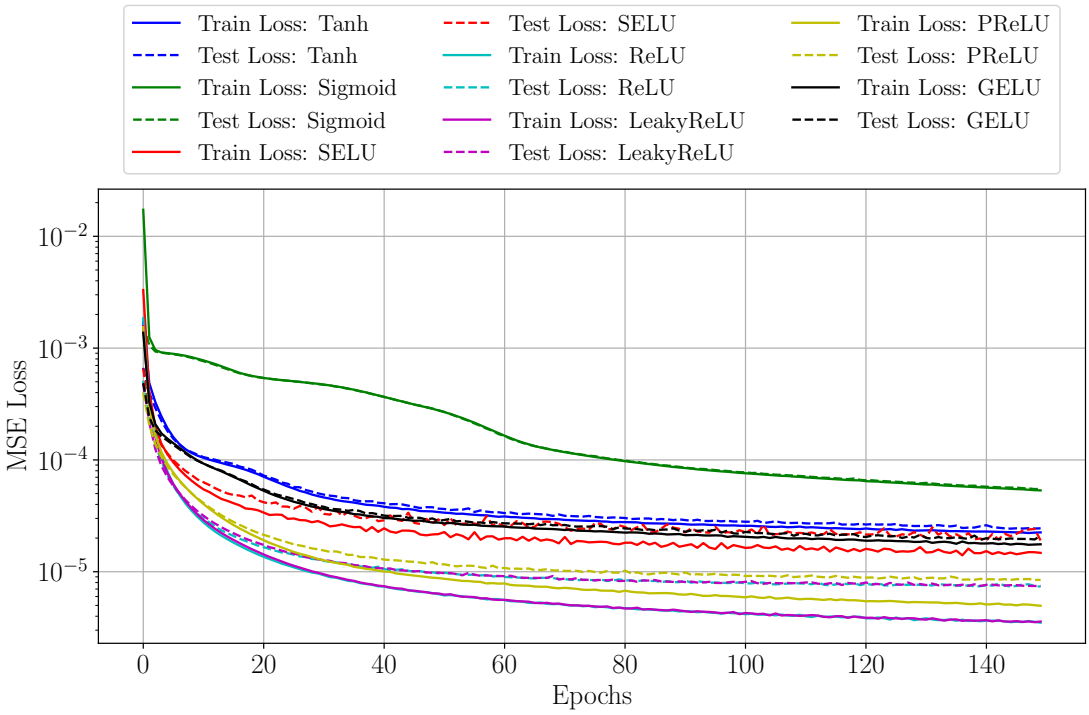


Figure B.6: Mean values of the train and test loss of a 5-fold cross-validation of an artificial neural network with different activation functions

slope of the convergence pattern of the models with increasing dropout becomes less steep. Meaning that a higher dropout requires a longer time to converge. However, in the interest of the aerodynamic model, the extra performance gain outweighs the extra training time. Therefore, one can conclude that the hypothesis stated at the beginning of this subsection is true, as a higher dropout value decreases on average the mean square error on the test dataset for a 5-fold cross-validation.

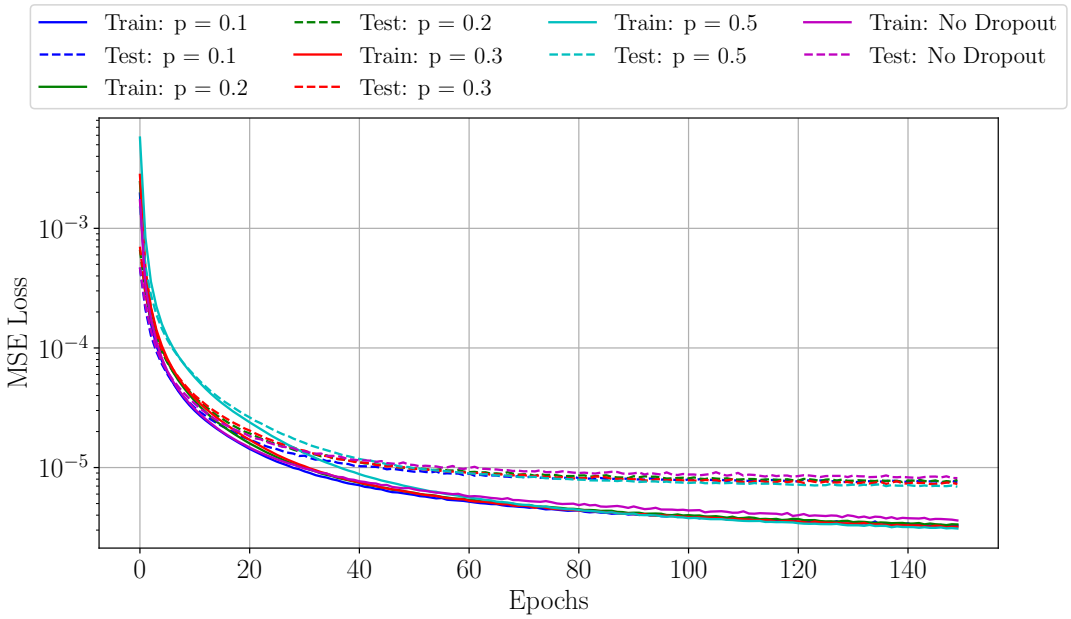


Figure B.7: Mean values of the train and test loss of a 5-fold cross-validation of an artificial neural network with varying dropout (p) levels

B.1.7 Regularization, Optimisation Functions, Loss functions, and Number of Epochs

In this subsection, four other specifications of the neural network are shortly discussed for completeness.

Regularization of the neural network is not included, as none of the previous models that have been run gave any sign of overfitting. Therefore, it is deemed unnecessary to have any regularization. For completeness 4 neural networks have been trained, one with L1 regularization, and one with L2 regularization with two different values of the L2 constant, and neither of them resulted in a convergence of the model. The results

can be viewed in Figure B.8. Therefore, instead of spending resources trying to obtain convergence of either L1 or L2 regularization, they are left out and will not be used during this thesis. Furthermore, since the addition of dropout is deemed favourable in subsection B.1.6, which also has the effect of reducing overfitting in neural networks, the addition of dropout will be deemed sufficient in preventing overfitting.

There are many different optimisation functions which can be considered in this thesis. The one that is most commonly used in the industry is ADAM, or adaptive moment estimation, and performed best in the comparison that is made in Figure B.9. There, ADAM is compared against other common optimisation functions such as traditional stochastic gradient descent, RMSprop, Momentum, and ADAgrad. Neither of them came close to matching the performance of ADAM and therefore it is chosen as the optimiser of the neural networks considered in this thesis.

The mean squared error loss function that is used in the last sections is not the only option for the loss function in neural networks. The option of using L1 loss, smooth L1 loss, or Huber loss can be considered. These loss functions are compared in Figure B.10. What has been found is that the difference in convergence is very similar, and therefore the choice is made to stick with the most common loss function namely, the mean squared error.

Finally, the number of epochs that are taken by the model is set at 300 epochs, as from the experience gained from the previous subsections, the models all reached a flattening of the test loss after around 150 epochs. Furthermore, early stopping is applied, meaning that when the model's performance does not decrease further over 15 epochs, the training is stopped and the best model of the previous epochs is kept as the trained model.

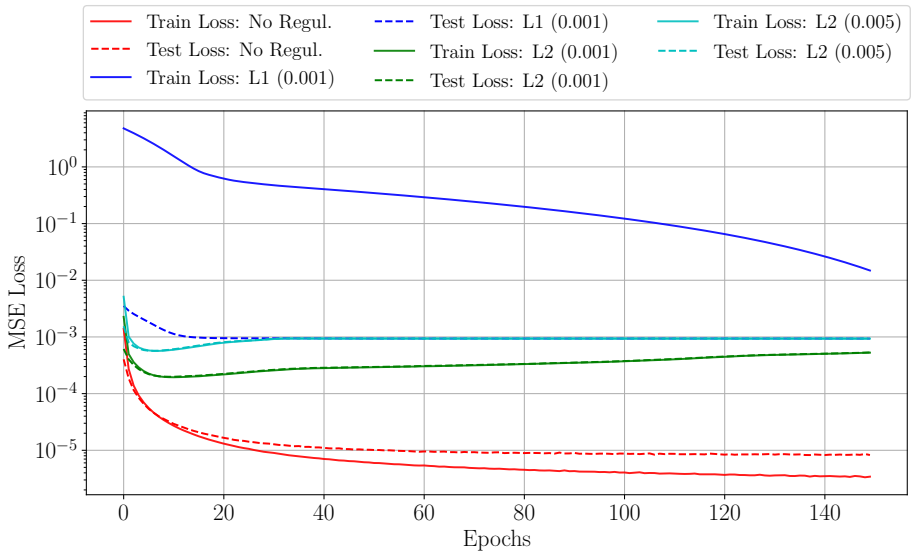


Figure B.8: Mean values of the train and test loss of 5-fold cross-validation of an artificial neural network with varying regularization (Regul.) methods

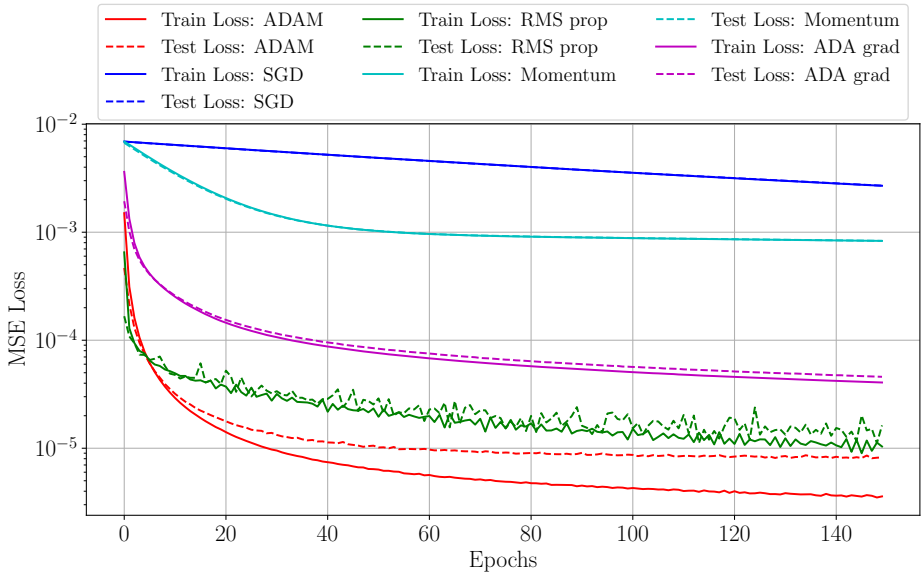


Figure B.9: Mean values of the train and test loss of 5-fold cross-validation of an artificial neural network with varying optimisation functions. SGD = Stochastic Gradient Descent

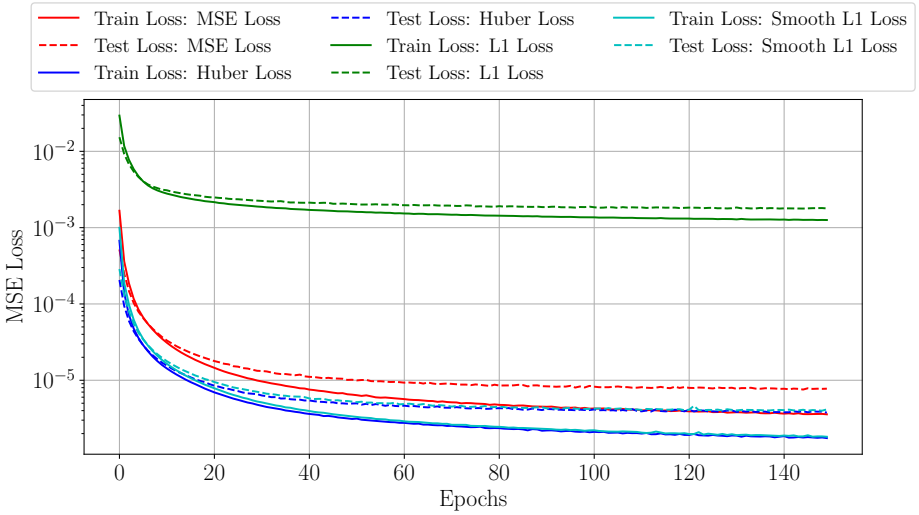


Figure B.10: Mean values of the train and test loss of 5-fold cross-validation of an artificial neural network with varying loss functions.

B.2 Genetic Algorithms Compared to Randomised, and Grid Search on 2D and 3D Test Function

In this section, genetic algorithms are applied towards the 2D Camel test function, as well as the 3D Hartmann test function. Genetic algorithms will be compared towards classical randomised, and grid search methods. This is done to see whether genetic optimisation strategies outperform the classical grid and random search algorithms. This section is split up into 2 parts. First, in subsection B.2.1 the comparison is made for a 2d function, after which the comparison is made for the 3D function in subsection B.2.2.

B.2.1 Comparison of Genetic, Randomised Search and Grid Search Optimisation Algorithms on Camel Function

In this subsection, the genetic algorithm is put to the test, and compared against the randomised search and grid search methods for finding the minimum of a function. For this example, the six-hump camel function is taken as the fitness function. The six-hump camel function is shown in Equation B.2 and also in Figure B.11 where the absolute minima are shown. The function is defined between -2 and 2 for the X-values, and -1 and 1 for the Y-values, and has two global minima at (x,y) = (-0.089, 0.713) and (0.089, -0.713), with a minimum value of -1.0326.

$$f(x,y) = \left(4 - 2.1 \cdot x^2 + \frac{x^4}{3}\right) \cdot x^2 + x \cdot y + \left(-4 + 4 \cdot y^2\right) \cdot y^2 \tag{B.2}$$

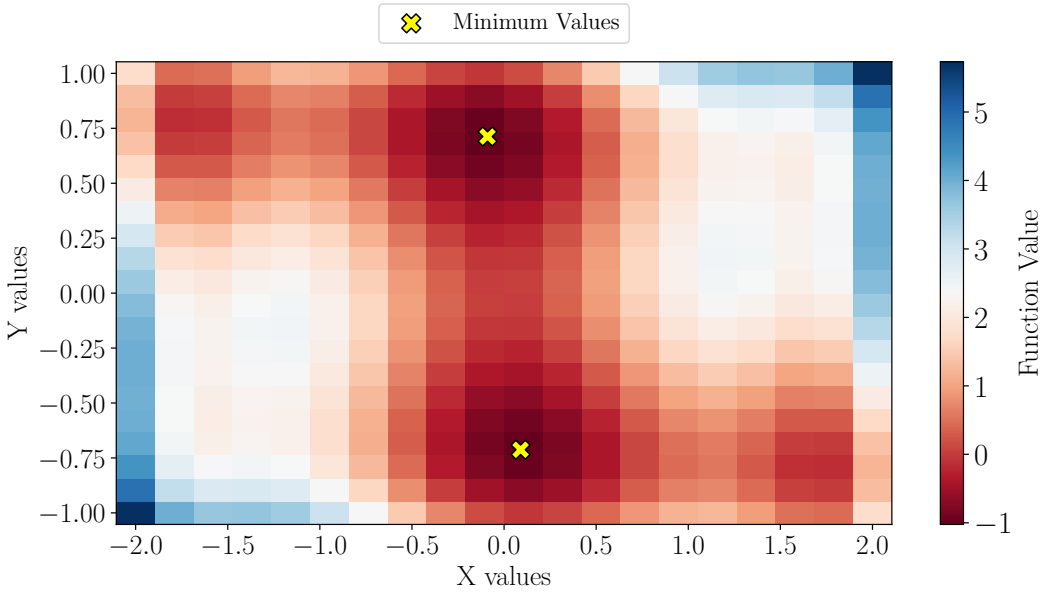


Figure B.11: Two-dimensional Camel test function and its minima.

In Figure B.12, the initial population of the genetic algorithm is shown, as well as the random search, and grid search sampling locations. The genetic algorithm has a population of 50 points, with 10 generations being used for finding the optimum locations. To make the comparison fair between all the algorithms, 500 random points are sampled for the random search algorithm, and a grid of 500 points is made for the grid search algorithm. Since the range for x is twice as large as for y for this function, twice as many points are sampled in x as in the y direction to create a rectangular grid. One can see first of all the large difference in the initial number of points between the grid/random search and the original population of the genetic algorithm. Furthermore, the fact that the random search often has some points bunched up, and then regions without a single point being sampled. The random sampling and grid search methods are entirely based on luck, as it is entirely based on having a sampled point very close towards the minimum values.

In Table B.3, the results of the optimisation are shown. What can be seen from the fitness values in the last column of the table is that although the algorithms are inherently different, the minimum fitness values are very close together. The best fitness score comes from the genetic algorithm, the grid search and finally the random search.

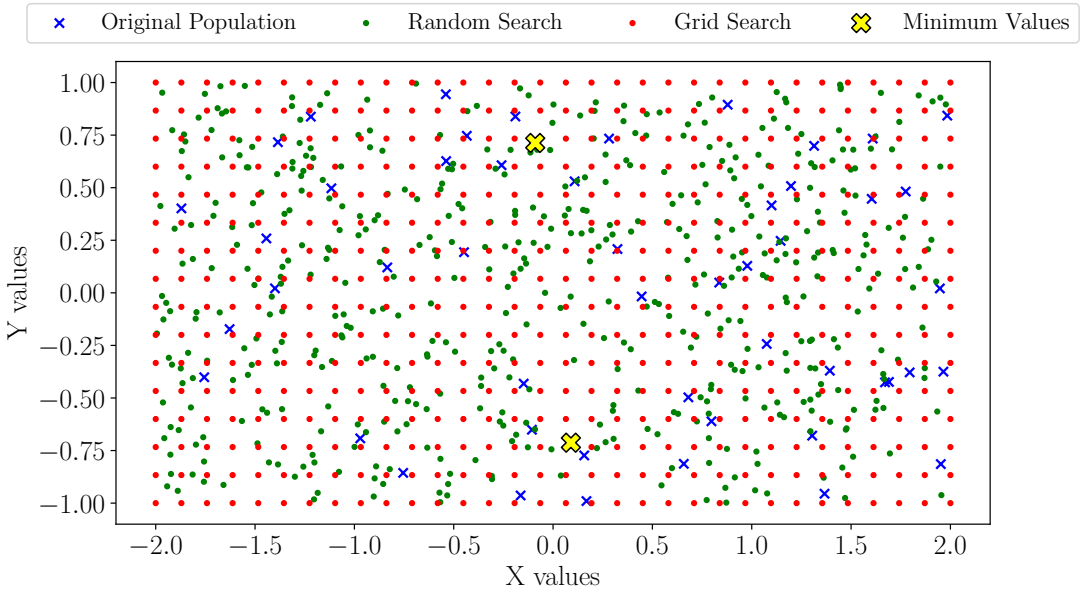


Figure B.12: Initial population of the grid search, random search and genetic optimisation algorithm for the two-dimensional Camel function.

However, the genetic algorithm can get closer to the optimum, while the random and grid search algorithms are so densely populated that it is very likely to have a point near the minimum values. Therefore, from this experiment, it can be concluded that between the three algorithms, no real decision can be made on whether one is superior, or was lucky with the initialisation, or with how the grid was laid out about the optimum. Therefore, a new experiment is set up with a higher dimensional function, which will provide a better comparison between all functions.

Table B.3: Comparison of the genetic, random and grid search algorithms for finding the minimum of the six-hump camel function.

	Population Size	**Generations**	**Fittest Genes**	**Fitness**
Genetic Optimisation	50	10	(-0.106, 0.734)	-1.027
Random Search	500	-	(-0.099; 0.743)	-1.019
Grid Search	500	-	(-0.064, 0.733)	-1.025
True Value			**(-0.089, 0.713)**	**-1.0326**

B.2.2 Comparison Genetic, Random and Grid Search on Hartmann Function

In the last subsection, inconclusive results were found between the genetic optimisation algorithm, random search and grid search algorithm for finding the minimum of a test function in two dimensions. In this section, a similar test on a three-dimensional testing function called the Hartmann function is performed. The Hartmann function can be seen in Figure B.13, it is usually defined on the hypercube x = 0, 1 in all dimensions, and has a global minimum located at (x,y,z) = (0.1146, 0.5556, 0.8525), with a value of -3.86278.

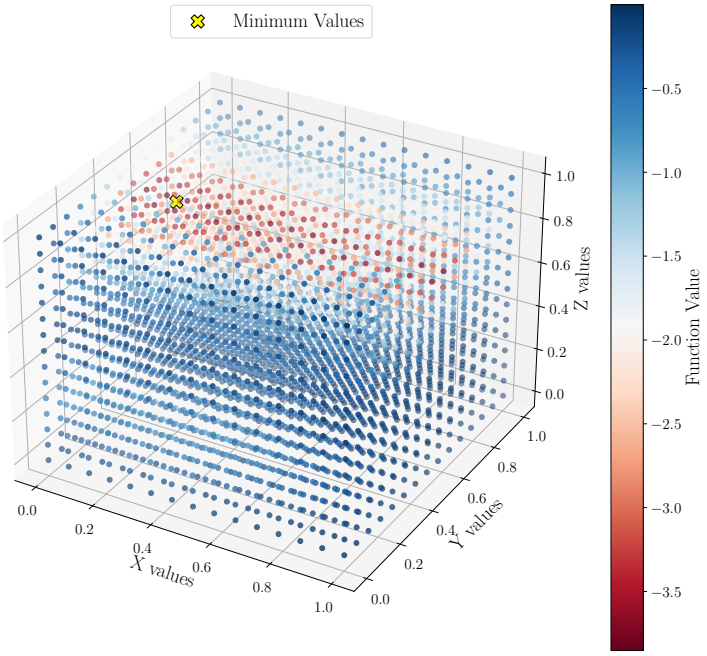


Figure B.13: Three-dimensional Hartmann test function and its minimum.

In Figure B.14 the distribution of points between the genetic search algorithm, random and grid search is provided. The total number of points for each algorithm is 500. The genetic algorithm achieves this by having 50 points in the population over 10 generations, while the two other algorithms do this by having a total number of points for the search of 500.

In Table B.4, the results of the optimization algorithms are shown. From the table, one can see that the minimum fitness is found by the genetic optimisation algorithm. Followed by the random search, and grid search algorithm. The random search algorithm is by definition random, and could mean that if a different initial population is taken, a different result could be obtained, this is in some sense also true for the genetic algorithm,

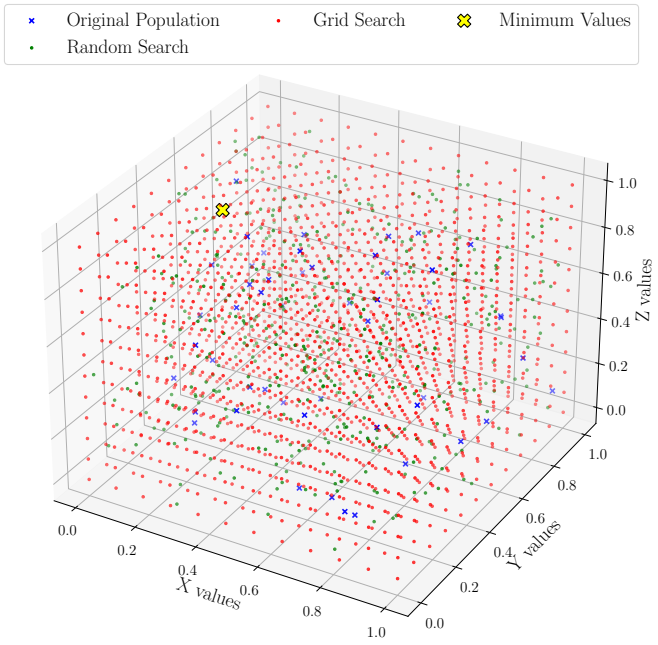


Figure B.14: Initial population of the grid search, random search and genetic optimisation algorithm for the three-dimensional Hartmann Function.

but also for the grid search algorithm, as the population size has now been chosen at 500. However, if a larger population size is taken, a tighter grid can be formed, or with a less large population, a courser grid must be taken, which can alter the location of the grid points, w.r.t. the minimum value.

In Table B.5, the average fitness value that was found by each of the algorithms considered in Table B.4 together with their standard deviation is provided. The standard deviation of the genetic and random search algorithms is calculated by repeating the optimisations 10 times, while the grid search algorithm, is performed by dividing the three-dimensional domain, in 10 different ways, by changing the grid size along one of the dimensions. From the table, one can see that the same trends remain as can be seen in Table B.4, the genetic algorithm on average finds the lowest value out of the algorithms, then closely followed by random search and finally grid search. The standard deviation of the results using the genetic algorithm is higher, but it has a lower mean value of the function. The grid search algorithm performs the worst on average and has the highest deviation in the results when changing the grid along one of the directions.

To conclude from this section, the genetic optimisation algorithm performed better than the random search and grid search algorithms for the three-dimensional test function. From this section, it is concluded that the genetic optimisation and random search algorithms are the best performing, with the genetic algorithm being the most likely

Table B.4: Comparison of the genetic, random and grid search algorithms for finding the minimum of the Hartmann function.

	Population Size	**Generations**	**Fittest Genes**	**Fitness**
Genetic Optimisation	50	10	(0.139, 0.554, 0.852)	-3.8622
Random Search	500	-	(0.126, 0.521, 0.874)	-3.7684
Grid Search	500	-	(0.125, 0.5, 0.875)	-3.6974
True Value			**(0.114, 0.555, 0.853)**	**-3.863**

Table B.5: Mean and standard deviation values of the comparison of the genetic, random and grid search algorithms for finding the minimum of the Hartmann function.

	Mean	**Standard Deviation**
Genetic Optimisation	-3.804	0.0757
Random Search	-3.778	0.0572
Grid Search	-3.713	0.0950

candidate for being the optimal hyperparameter optimization algorithm.

B.3 Convergence of Models for Aerodynamic Coefficients

In this section, the convergence plots of the optimised artificial neural networks for each of the aerodynamic coefficients are provided. The performance in terms of MSE and percentage of the predictions within the tolerances are provided in section 5.3.

The convergence plots for the C_m, C_l, C_n, C_x, C_y, C_z, and multiple-output model are provided in Figure B.15, Figure B.16, Figure B.17, Figure B.18, Figure B.19, Figure B.20, and finally Figure B.21 respectively.

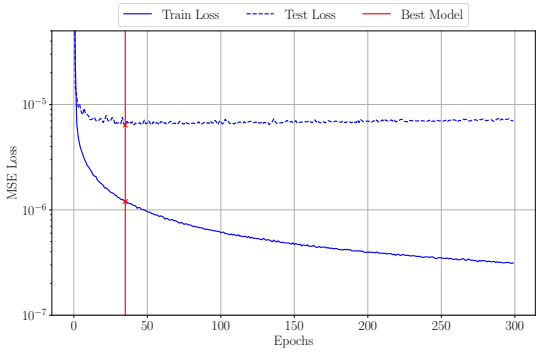


Figure B.15: Convergence of the optimised artificial neural network with parameters shown in Table 5.3 for the pitching moment coefficient C_m.

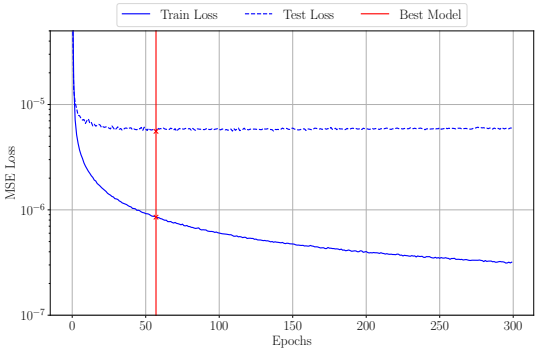


Figure B.16: Convergence of the optimised artificial neural network with parameters shown in Table 5.3 for the rolling moment coefficient C_l.

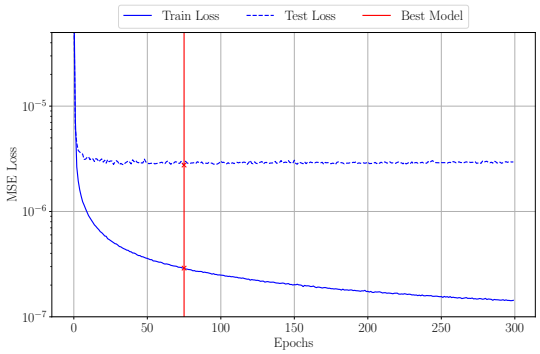


Figure B.17: Convergence of the optimised artificial neural network with parameters shown in Table 5.3 for the yawing moment coefficient C_n.

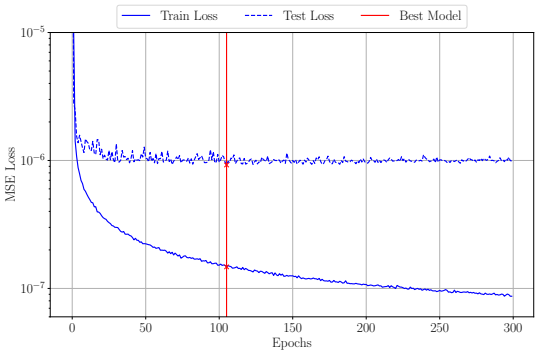


Figure B.18: Convergence of the optimised artificial neural network with parameters shown in Table 5.3 for the rolling moment coefficient C_x.

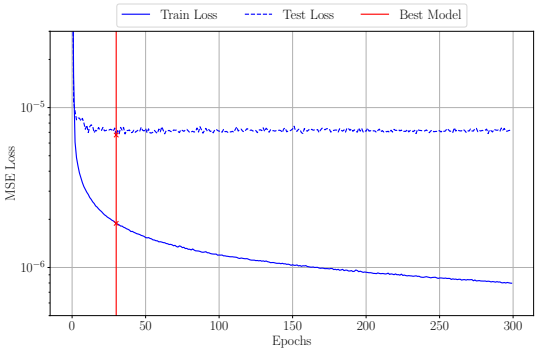


Figure B.19: Convergence of the optimised artificial neural network with parameters shown in Table 5.3 for the side force coefficient C_y.

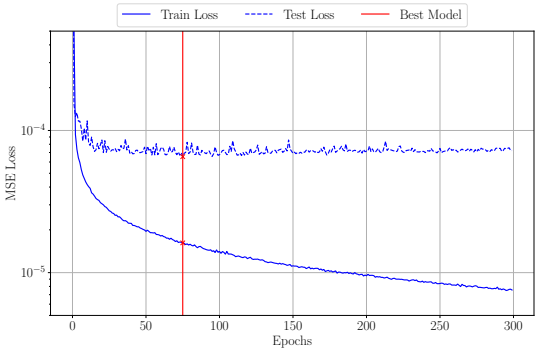


Figure B.20: Convergence of the optimised artificial neural network with parameters shown in Table 5.3 for the side force coefficient C_z.

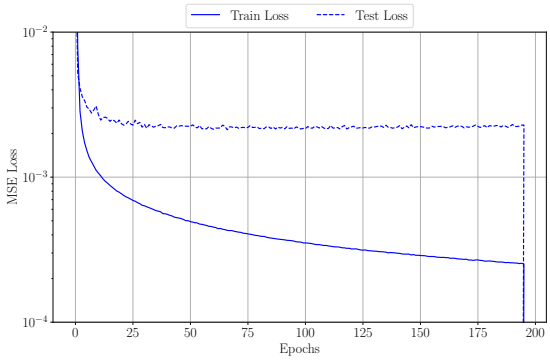


Figure B.21: Convergence of the optimised artificial neural network with parameters shown in Table 5.3 for the pitching, rolling, and yawing moment coefficient C_m, C_l, C_z and the side force coefficients C_x, C_y, C_z.

B.4 Example Prediction for Aerodynamic Coefficients

In this section, example test polars and their respective predictions from the artificial neural networks are provided. The artificial neural networks that are trained in chapter 5, show no predictions in the results section in section 5.3. Therefore, in this section, the reader is shown the predictions of each of the coefficients. For confidentiality, the control surface deflections, Mach number, angle of attack, and angle of sideslip are randomised and not disclosed.

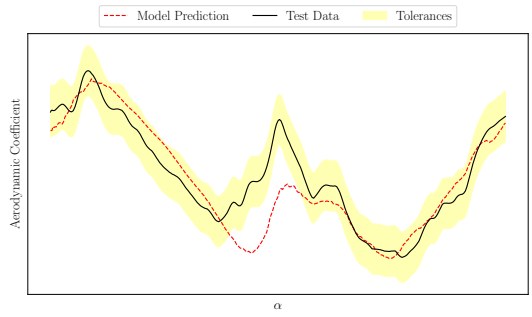


Figure B.22: Random Example 1 for C_m from the test dataset and the corresponding artificial neural network model predictions.

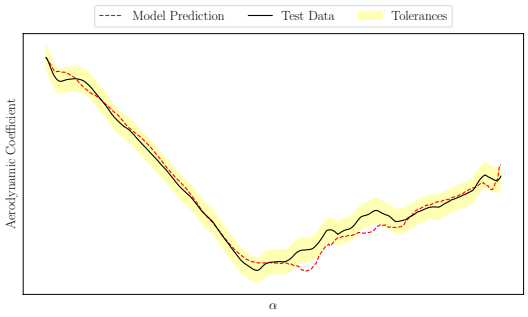


Figure B.23: Random Example 2 for C_m from the test dataset and the corresponding artificial neural network model predictions.

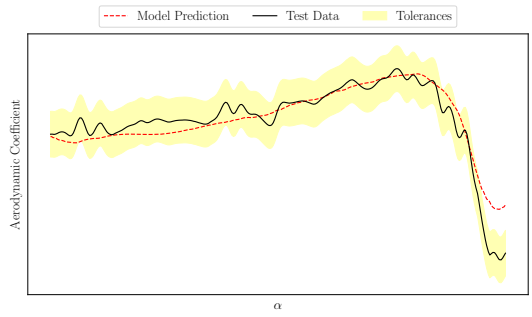


Figure B.24: Random Example 1 for C_l from the test dataset and the corresponding artificial neural network model predictions.

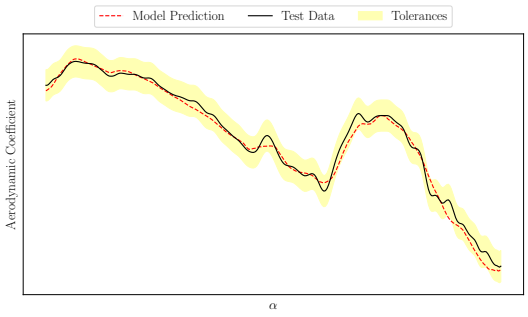


Figure B.25: Random Example 2 for C_l from the test dataset and the corresponding artificial neural network model predictions.

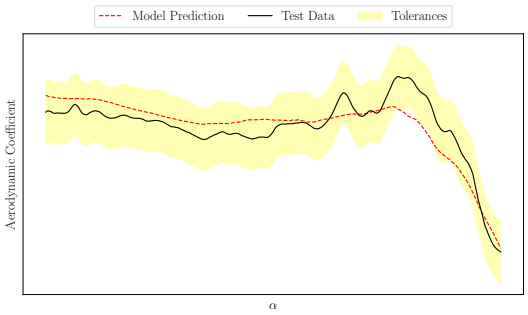


Figure B.26: Random Example 1 for C_n from the test dataset and the corresponding artificial neural network model predictions.

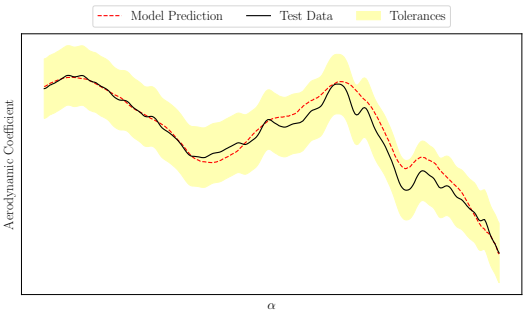


Figure B.27: Random Example 2 for C_n from the test dataset and the corresponding artificial neural network model predictions.

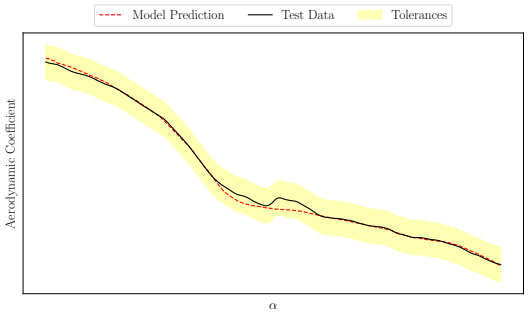


Figure B.28: Random Example 1 for C_x from the test dataset and the corresponding artificial neural network model predictions.

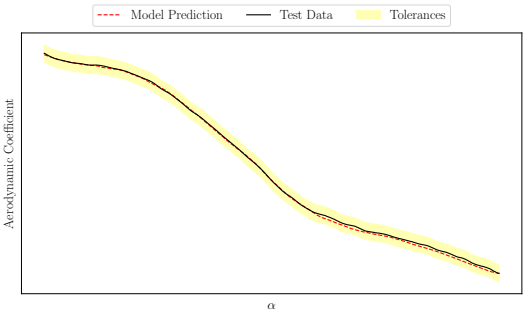


Figure B.29: Random Example 2 for C_x from the test dataset and the corresponding artificial neural network model predictions.

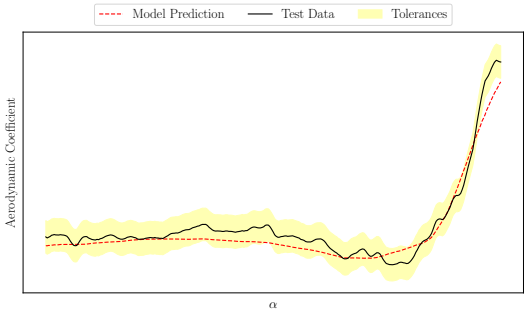


Figure B.30: Random Example 1 for C_y from the test dataset and the corresponding artificial neural network model predictions.

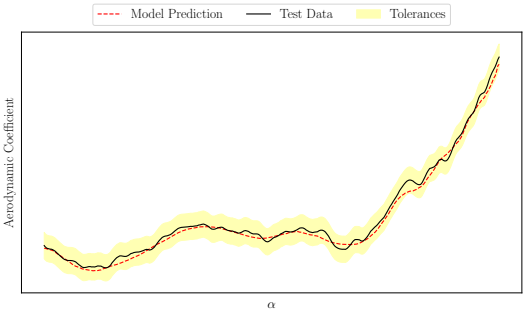


Figure B.31: Random Example 2 for C_y from the test dataset and the corresponding artificial neural network model predictions.

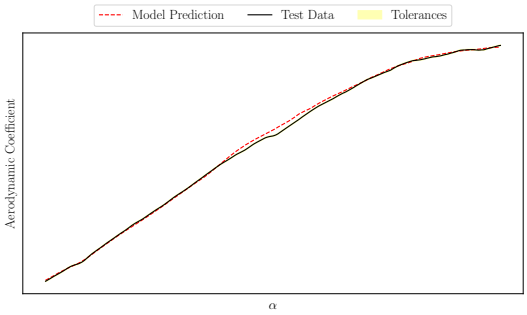


Figure B.32: Random Example 1 for C_z from the test dataset and the corresponding artificial neural network model predictions.

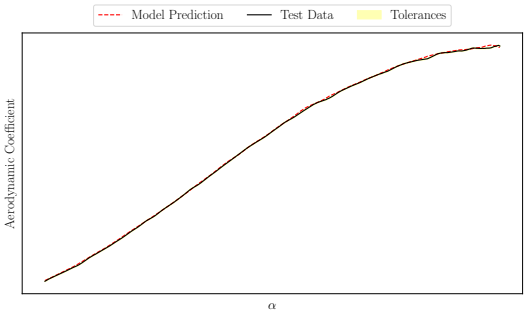


Figure B.33: Random Example 2 for C_z from the test dataset and the corresponding artificial neural network model predictions.

Appendix C

Bayesian Neural Networks

In this chapter, the appendix of the Bayesian neural network (BNN) chapter is provided. This appendix builds on the chapter on Bayesian neural networks shown in chapter 6. This chapter is split up into two sections. First, there is the model background appendix shown in section C.1, which shows the convergences of the models used to differentiate between the aleatoric and epistemic uncertainty coming from BNNs. Next, the detailed hyperparameter optimisation of the BNN is performed in section C.2.

C.1 Model Background

In this section, the convergence plots for the models trained to show the difference in aleatoric and epistemic uncertainty in subsection 6.1.2 are provided. These are shown in Figure C.1, and Figure C.2. The functions which are being modelled are a sine wave with varying random normal distributed noise shown in subsection 6.1.2.

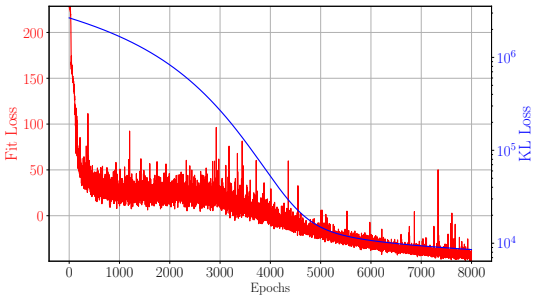


Figure C.1: Convergence of 1-dimensional Bayesian neural network for the low noise case.

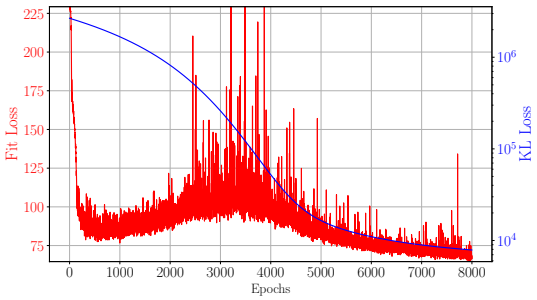


Figure C.2: Convergence of 1-dimensional Bayesian neural network for the high noise case.

C.2 Hyperparameter Selection

The hyperparameters of a Bayesian neural network are similar to the ones of an artificial neural network. In this section, they are discussed and their effect on the model performance is evaluated.

The hyperparameters which can be varied in the Bayesian neural network are the number of neurons, number of layers, learning rate, batch size, learning rate, and samples drawn from the posterior.

These hyperparameters are determined on dataset 3 since the training time of the Bayesian neural networks is expected to be much larger than the ANN. Therefore, a smaller dataset is used to quantify some foundational understanding, without having to spend too much time training the models.

In Table C.1, a summary is provided of the run plan which is used to determine the optimal settings of the hyperparameters which are seen above. Between different models, usually, a single variable is changed such that the effect of changing this variable can be evaluated.

C.2.1 Model 1: Baseline Model

In Figure C.3, and Figure C.4, one can see the convergence and prediction of a test polar respectively, of a Bayesian neural network. The specification of this BNN can be seen in Table C.1. This is considered the baseline to which the other models that are trained and shown in this section can be compared. From Figure C.3 one can see that

Table C.1: Specifications of each of the models which have been created for optimising the hyperparameters of the Bayesian neural network.

	Number of Neurons	Hidden Layers	Learning Rate	Batch Size	Samples Drawn	MSE
Model 1	512	2	1e-3	16384	10	4.41e-6
Model 2	1024	2	1e-3	16384	10	3.98e-6
Model 3	1024	1	1e-3	16384	10	3.95e-6
Model 4	1024	1	1e-3	16384	25	3.51e-6
Model 5	1024	1	1e-3	4096	10	3.91e-6
Model 6	1024	1	1e-4	16384	10	3.48e-6

the KL loss has converged when the model stopped training, furthermore, the fit loss declined sufficiently and was getting unstable at the point of 2500 epochs. Meaning that the learning rate could be too high. However, this will be determined in this section as well.

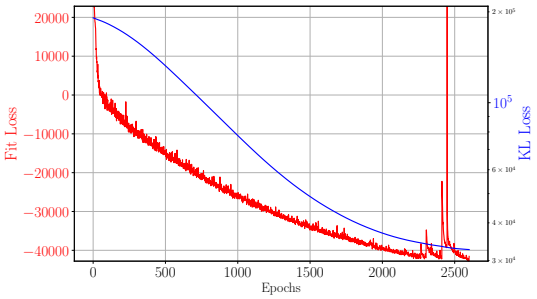


Figure C.3: Convergence of 4 input, single-output, Bayesian neural network, with 512 hidden neurons and 2 hidden layers.

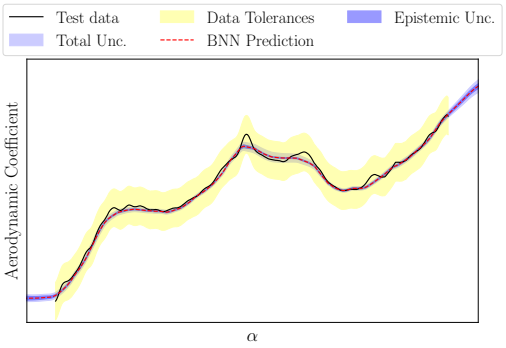


Figure C.4: Model 1: Test Polar of 4 input, single-output, Bayesian neural network, with 512 hidden neurons and 2 hidden layers.

C.2.2 Model 2: Doubling of Hidden Neurons Model 1

In Figure C.5, and Figure C.6, one can see the convergence and prediction of a test polar respectively, of a BNN. The specification of this BNN can be seen in Table C.1, and the noteworthy difference with the baseline model is that this model is trained with twice the amount of neurons in its hidden layers, being 1024, compared to only 512 in the baseline model. The notable difference is in the convergence, although the pattern and trend are the same, the KL loss is much higher than in the baseline model

case, this is directly the effect of having more neurons, and thus a higher parameter number of different parameters. Furthermore, from the test data polar that is shown in Figure C.6 the epistemic uncertainty (in dark purple) is higher than that shown in Figure C.4. This is also the effect of having more neurons, which is directly affecting the model uncertainty. The total uncertainty grows with the epistemic uncertainty since the aleatoric uncertainty remains unchained since the same dataset is used. The overall performance of the model, although not visible in the shown figure, has improved. In Table C.1 it is shown that the MSE is 3.98e-6 on the test dataset, while that of the baseline model was 4.41e-6, an improvement of around 10%.

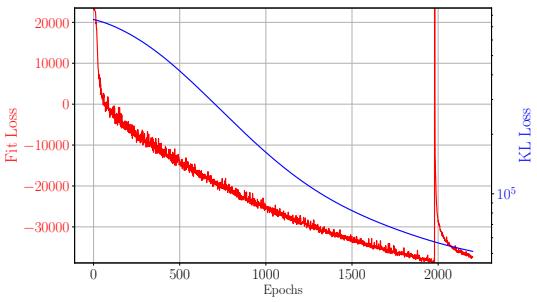


Figure C.5: Convergence of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 2 hidden layers.

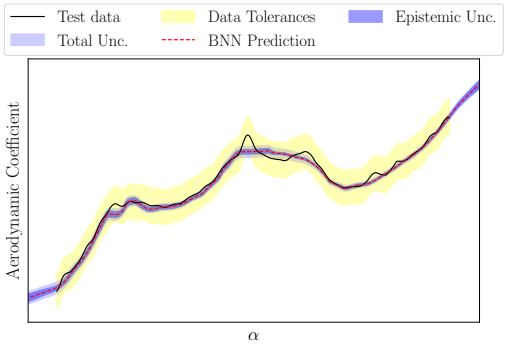


Figure C.6: Model 2: Test Polar of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 2 hidden layers.

C.2.3 Model 3: Halving the Number of Hidden Layers of Model 2

In Figure C.7, and Figure C.8, one can see the convergence and prediction of a test polar respectively, of a BNN. The specification of this BNN can be seen in Table C.1, and the noteworthy difference with the previous model 2, is that this model is trained with only 1 hidden layer instead of 2, retaining the same amount of neurons as model 2, at 1024. Firstly, the model retains almost the same MSE as model 3, meaning that doubling the number of neurons had a greater effect than doubling the number of layers. Furthermore, the epistemic uncertainty is low, compared to Model 1, but also to Model 2, meaning that the increase in the number of layers was more influential in increasing the epistemic uncertainty than the number of neurons.

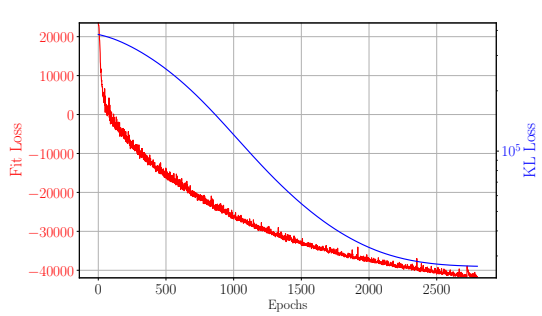


Figure C.7: Convergence of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 1 hidden layer

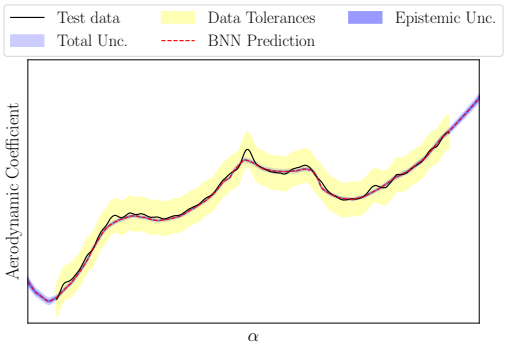


Figure C.8: Model 3: Test Polar of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 1 hidden layer

C.2.4 Model 4: Identical to Model 3 Besides Having 25 Samples Drawn

In Figure C.9, and Figure C.10, one can see the convergence and prediction of a test polar respectively, of a BNN. The specification of this BNN can be seen in Table C.1, and the noteworthy difference with the previous model 3, is that this model is trained by sampling 25 times from the posterior distribution instead of 10 times. This should, in theory, provide for a better uncertainty estimation, as the prediction is sampled 25 times, instead of just 10 times. This means that the predictions and uncertainties are averaged over a larger number of samples, allowing for a better estimation of the uncertainty. This means that in theory the predictions should be more accurate, and when looking at Table C.1 it is. The MSE is 3.51e-6, compared to 3.95e-6 for model 3. Furthermore, since more samples need to be drawn, the computational time will increase significantly. The training time of this model, although not shown in Table C.1, was around 1.6 times longer than model 3, using only 10 samples, which means that the increase in samples drawn is not directly correlated with the training time of the model. This results in the model having the lowest epistemic uncertainty out of any of the models trained so far, furthermore it also has the lowest MSE as can be seen from Table C.1. From the Figure C.9 it can be seen that the convergence of the fit loss is much smoother, furthermore, the KL loss is closer to the magnitude of model 3, however, it is converged a bit further.

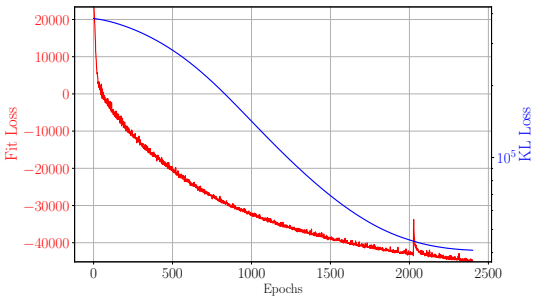


Figure C.9: Convergence of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 1 hidden layer and 25 samples drawn from the posterior distribution.

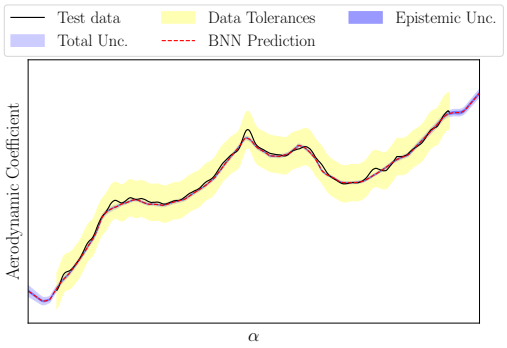


Figure C.10: Model 4: Test Polar of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 2 hidden layers and 25 samples drawn from the posterior distribution.

C.2.5 Model 5: With a Quarter of the Batch Size of Model 3

In Figure C.11, and Figure C.12, one can see the convergence and prediction of a test polar respectively, of a BNN. The specification of this BNN can be seen in Table C.1, and with this model a lower batch size is tested, compared to Model 3. This model also uses 1024 neurons with 1 hidden layer, a learning rate of 1e-3 and 10 number of samples drawn. From comparing Figure C.11, and Figure C.7, one can see that the smaller batch size has the effect of making the convergence take more epochs, while in absolute terms the fit loss and KL loss are comparable between both models. When comparing Figure C.12, and Figure C.8, there is almost no difference in the prediction as the MSE of both models are extremely similar at 3.9e-6. When comparing the total uncertainty present in the model, the lower batch size seems to have provided the model with more variation of the aleatoric uncertainty over a given polar. While the aleatoric uncertainty is relatively constant throughout the polar shown in Figure C.8, this is not the case in Figure C.12. While the epistemic uncertainty is low and rather constant, the aleatoric uncertainty grows and shrinks over the length of the polar. Likely, with the lower batch size, the learning of the posterior distribution is harder. There are 416 training polars used to train these models, with a batch size of 4096, the model only gets around 10 points per polar to train for each batch, while this is 40 for model 3 with a batch size of 16384, a significant difference and could be at the heart of why the aleatoric uncertainty varies more. For the aerodynamic model that is created, it is a higher priority to have a more accurate prediction. Therefore, a higher batch size is preferred, even if the convergence time is longer.

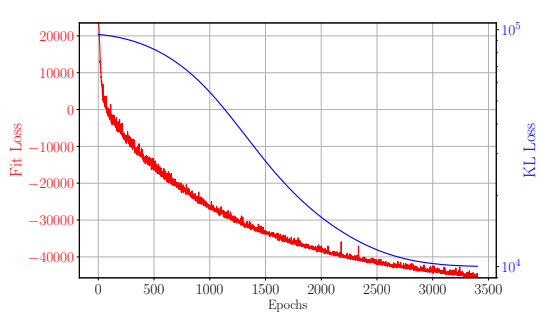


Figure C.11: Convergence of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 1 hidden layer, with a batch size of 4096 points.

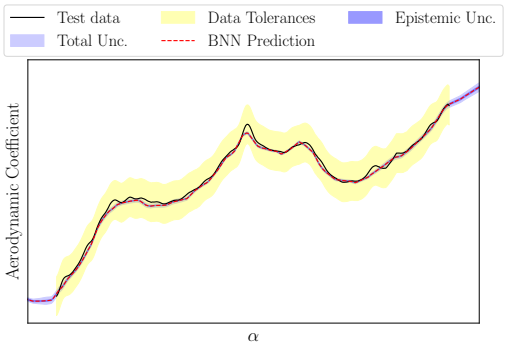


Figure C.12: Model 5: Test Polar of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 1 hidden layer, with a batch size of 4096 points.

C.2.6 Model 6: With a Learning Rate 10 Times Smaller Than Model 3

In Figure C.13, and Figure C.14, one can see the convergence and prediction of a test polar respectively, of a BNN. The specification of this BNN can be seen in Table C.1 as model 6, which has a lower learning rate than model 3 which has the same parameters apart from this. The goal of testing a model with a lower learning rate is that often a model trained with a learning rate of 1e-3 has issues like can be seen in Figure C.3, and Figure C.5, where the training becomes unstable at the end of the training cycle, and the fit loss shoots up. However, actually by lowering the learning rate the model reached the maximum training epochs after 12 hours and 45 minutes.

C.2.7 Conclusion

The conclusion from this section is that the Bayesian neural network hyperparameters have a profound effect on their performance and ability to provide the uncertainty measure. From the performance that has been shown, a careful trade-off between the number of layers and number of neurons needs to be found, as it has a large effect on the stability of the fit loss during training. Furthermore, more neurons are beneficial compared to more hidden layers. The number of samples drawn from the model had the largest effect on the MSE of the model, and if the training time allows, the number will be increased to 25 as has been tested in this section. Furthermore, the batch size which has been varied between 16384, and 4096, had a small effect on the MSE of the model, but a larger effect

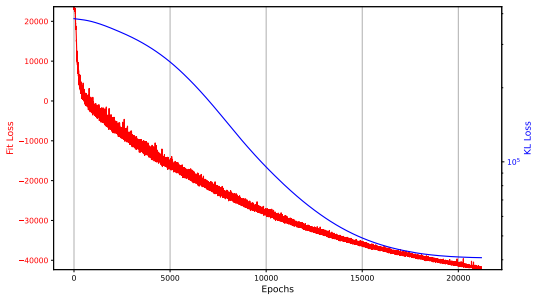


Figure C.13: Convergence of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 1 hidden layer and a learning rate of 1e-4.

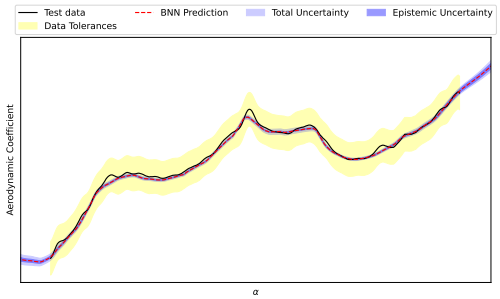


Figure C.14: Model 6: Test Polar of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 1 hidden layer and a learning rate of 1e-4.

on the uncertainty of the model. The lower batch size showed reduced confidence in the uncertainty, however, a lower MSE cannot be ignored. Both options can be a valid choice whether more interest or priority is provided towards the uncertainty or the predictive capabilities. In the interest of finding the best model to create an aerodynamic model of a combat aircraft, the lower batch size is given priority for this research. Finally, the lower learning rate increased the training time dramatically, and for the future models that are trained, the learning rate will be chosen as the maximum learning rate that still allows for convergence of the fit loss, however for now a baseline value of 1e-4 will be taken, as the lower MSE is given priority as opposed to the longer training time.

In Figure C.15, and Figure C.16, the convergence and sample polar is provided, for the best hyperparameters of the Bayesian neural network. This means a single hidden layer, with 1024 neurons, a learning rate of 1e-4, a batch size of 4096, and 25 samples drawn from the distribution. The resulting MSE is 3.38e-06, meaning the combined model is indeed the best-performing model so far.

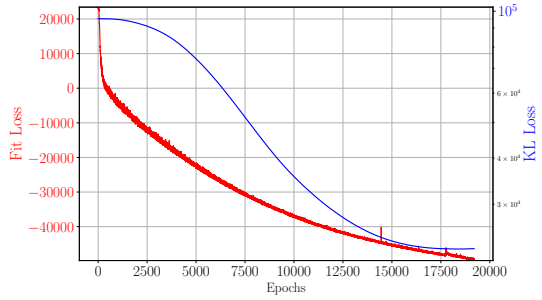


Figure C.15: Convergence of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 1 hidden layer and a learning rate of 1e-4, batch size of 4096, and samples 25 times from the posterior.

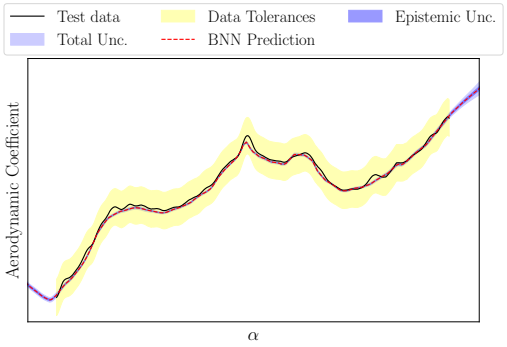


Figure C.16: Test Polar of 4 input, single-output, Bayesian neural network, with 1024 hidden neurons and 1 hidden layer and a learning rate of 1e-4, batch size of 4096, and samples 25 times from the posterior.

Appendix D

Comparison and Stacked Models

This appendix is an extension of the chapter on the comparison of all the models, and creating a stacked model in chapter 7. In this appendix, three tables are presented. Firstly, the MAPE of the single-output and multi-output models for each of the aerodynamic coefficients is presented in section D.1. Finally, the sensitivity of removing each of the base models of the stacked models in terms of the MSE is performed in section D.2.

D.1 Mean Absolute Percentage Error for Single-and Multiple-Output Models

In this section, the mean absolute percentage error (MAPE) for the single-output and multiple-output models are presented in Table D.1, and Table D.2 respectively.

The results seen in Table D.1 for the single-output models tell a slightly different story than the MSE and the percentage of the predictions within the tolerances which can be seen in section 7.1. The XGBoost model appears to have a higher performance compared to its performance for the MSE and PWT. Although the CatBoost, ANN, and BNN still perform well.

For the multiple-output model that is seen in Table D.2, the performances of the different models are to be expected when comparing them to the MSE and PWT. Only the RF model is performing much better, especially for the C_m where it is the best-performing model.

Table D.1: Mean absolute percentage error (MAPE) comparison of all the single-output models and aerodynamic coefficients. RF = Random Forrest, XGB = XGBoost, LGB = LightGBM, CAT = CatBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network

	RF	**XGB**	**LGB**	**CAT**	**ANN**	**BNN**
C_m	0.500	1.156	0.280	**0.132**	0.218	0.209
C_l	1.375	**0.953**	2.950	1.22	1.08	1.872
C_n	1.006	**0.630**	0.796	0.720	0.830	0.768
C_x	0.185	0.285	0.179	1.071	**0.135**	0.141
C_y	1.315	**0.95**	1.216	1.029	1.037	1.842
C_z	0.048	0.328	0.046	**0.039**	0.053	0.042

Table D.2: Mean absolute percentage error (MAPE) comparison of all the multi-output models and aerodynamic coefficients. RF = Random Forrest, XGB = XGBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network

	RF	**XGB**	**ANN**	**BNN**
C_m	**0.191**	0.575	1.156	0.216
C_l	1.273	4.188	**0.953**	1.641
C_n	0.702	1.234	0.630	**0.563**
C_x	**0.146**	1.234	0.179	0.185
C_y	1.227	1.742	**0.950**	1.589
C_z	0.065	0.069	0.328	**0.034**

D.2 Sensitivity of Base Models

In this section, the sensitivity of the MSE when removing one of the base models from the stacked model is shown. The sensitivity is provided as a percentage difference w.r.t. the original model. Meaning that if the percentage is positive, the MSE has increased when removing one of the models. Therefore, the performance of the models is worsening with larger percentage differences, and the importance of the model is higher as not including this model in the stacked model, increases the MSE the most.

In Table D.3, one can see the percentage difference in MSE when the specific model is removed from the analysis. The first thing that can be seen is that the magnitudes of the percentage differences are correlated towards the total change in MSE that the stacked model has over the base models. For example, for the C_m the stacked model has a 34% decrease in MSE over the best base model, while for the C_n this is only 10%. Therefore, between different aerodynamic coefficients, one must be careful in comparing the magnitudes of the percentage changes. On average one can conclude that the CatBoost, ANN, and BNN form the strongest models as on average they have the largest percentage effect on the MSE. While the Random Forest is the weakest model as for most of the aerodynamic coefficients the MSE decreases slightly when removing it from the stacked model.

Table D.3: Percentage change in MSE when a model is removed from the stacked model. RF = Random Forrest, XGB = XGBoost, LGB = LightGBM, CAT = CatBoost, ANN = Artificial Neural Network, BNN = Bayesian Neural Network

	RF [%]	**XGB** [%]	**LGB** [%]	**CAT** [%]	**ANN** [%]	**BNN** [%]
$\boldsymbol{C_m}$	0.061	0.0677	0.138	15.46	1.58	15.25
$\boldsymbol{C_l}$	-1.79	0.22	0.32	3.57	9.109	2.00
$\boldsymbol{C_n}$	-0.11	0.53	0.19	1.67	5.64	3.93
$\boldsymbol{C_x}$	-0.41	-1.064	4.10	0.35	11.34	15.90
$\boldsymbol{C_y}$	0.46	7.12	0.47	1.51	6.03	1.19
$\boldsymbol{C_z}$	-0.644	3.27	0.319	7.17	0.56	6.82