Development of a Virtual Sensor using Convolutional Neural Networks

Angelos Kylikas

Diploma Thesis

School of Naval Architecture and Marine Engineering
National Technical University of Athens

Supervisor: Associate Professor G. Papalambrou

Committee Members:

Associate Professor Ch. Papadopoulos
Associate Professor D. Lyridis

October 2022

Acknowledgements

The current work, concludes my academic journey at the School of Naval Architecture and Marine Engineering of the National Technical University of Athens. These five years, have been without a doubt the most prosperous of my life and have laid the foundations for many more to come.

The work described in this thesis was carried out at the Laboratory of Marine Engineering (LME) at the School of Naval Architecture and Marine Engineering of the National Technical University of Athens, under the supervision of Assistant Professor George Papalambrou.

I owe my greatest appreciation to my thesis supervisor, Associate Professor George Papalambrou, for giving me the opportunity to work on this fascinating topic, which required multidisciplinary skills. I would also like to thank him for his patience, continuous support, vision as well as the lessons he taught us as undergraduates in control engineering and programming.

I would also like to thank Mr. Vasileios Karystinos, Phd candidate at LME , for the fruitful discussions on my thesis topic as well as the apt and insightful comments regarding the implementation of the code.

Finally, I would like to express my sincere gratitude to my family, Avra, and my friends, for the unceasing encouragement, support and motivation throughout my studies. It has truly been a blast.

Abstract

In the current thesis, the development of a virtual sensor with the use of deep convolutional neural networks is investigated. The application, concerns a hybrid marine engine and the embedding of the software on it. On the first part, the mathematical foundations of the neural networks are described, with particular focus on convolutional neural networks and the tools required. Following that, the fundamental parameters of the engine operation are presented, concentrating on the parameters important for the current application. The next chapter, concerns the presentation of the physical system itself as well as the data used pre-processing. Additionally, the software embedding on the engine with the use of a Raspberry Pi coupled with a CAN bus controller, is discussed . The following part, is dedicated to specifying the model architectures as well as the statistical description of the data and their relationships. After selecting the inputs of the models, their training and the corresponding accuracy results are presented. Lastly, concluding remarks are discussed. The whole thesis, is carried out with the use of Julia Programming Language, which is also used for the software embedding on the Raspberry Pi.

Περίληψη

Στην παρούσα διπλωματική εργασία, εξετάζεται η δημιουργία ενός εικονικού αισθητήρα με χρήση συνελικτικών νευρωνικών δικτύων. Η εφαρμογή του αισθητήρα αφορά την ενσωμάτωση του κώδικα σε μια ναυτική υβριδική μηχανή. Αρχικά, το μαθηματικό υπόβαθρο για τα νευρωνικά δίκτυα παρουσιάζεται με ιδιαίτερη έμφαση στα συνελικτικά δίκτυα και τα εργαλεία που θα χρησιμοποιηθούν. Ακολούθως, αναφέρονται οι βασικές παράμετροι λειτουργίας της μηχανής εμβαθύνοντας κυρίως σε αυτές που αφορούν την παρούσα εργασία. Το επόμενο κεφάλαιο, αφορά στην παρουσίαση του φυσικού συστήματος καθώς και την προεπεξεργασία των δεδομένων που θα χρησιμοποιηθούν. Παράλληλα παρουσιάζεται και η δυνατότητα εφαρμογής των μοντέλων στην μηχανή μέσω ενός Raspberry Pi και ενός CAN bus controller.Έπειτα, πραγματοποιείται στατιστική περιγραφή των δεδομένων καθώς και των μεταξύ τους επιδράσεων ενώ παρατίθενται και η βασική αρχιτεκτονική των μοντέλων. Αφού επιλεχθούν τα μεγέθη που χρησιμοποιούνται ως είσοδοι στα μοντέλα, οπτικοποιούνται και ποσοτικοποιούνται τα αποτελέσματα από την εκπαίδευση των νευρωνικών δικτύων όσον αφορά την ακρίβεια. Το τελευταίο κομμάτι αφιερώνεται στα συμπεράσματα από την εργασία καθώς και σε μελλοντικές προτάσεις. Όλοι οι κώδικες της διπλωματικής αναπτύχθηκαν σε γλώσσα Julia.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Applying machine learning techniques for the prediction of dynamical systems, is an ever increasing trend in the recent years with multiple advantages [3]. Machine learning models, are inherently able to capture and extract complex behaviours and patterns, from observed data, with a low computational cost. An industry, that is currently trying to reap the fruits of theses advances is the Shipping Industry. A field of application, consists the sensors onboard and specifically those that have to do with the engines. Physical sensors are expensive, occupy a lot of space, require experienced personnel and are often noisy in working conditions and subsequently are not always suitable for real-time monitoring and prediction. On the contrary, a common practice, especially in the automotive industry in the recent past, is the use of artificial neural networks for the modelling of certain engine parameters based on measurements that are easily and robustly available.

1.1 Motivation

Shipping is responsible for carrying around 80% of the global trade [4]. At the same time, CO2 emissions from international shipping were estimated (2012) to be 2.2% of global anthropogenic emissions [5]. From the above, it is evident that even minor advances of technology in this sector result in major economic as well as environmental gains. The major driving force and energy producer of a ship and consumer is it's power train, usually a diesel engine. In this scope, major efforts and research have been made in the recent years for an optimization of it's performance and a concurrent emissions reduction. As in most industries there is no panacea for the problems faced. A versatile, multidisciplinary approach is required in order for the Shipping sector to become sustainable.

The current regulations by the IMO impose cups on emissions that have to do with hydrocarbons, nitrogen-oxides (NO_X), sulphur- oxides (SO_X) as well as particulate matter (PM) . To address these regulations in place, solutions like alternative fuels are coupled with others like scrubbers etc. Parallel to the environmental transition another revolution is taking place, that of the digital transformation and the Internet of Things. As a result, systems are transformed consistently to cyber-physical, with tremendous potential in advancements regarding monitoring, optimization, safety and automation.

In addition to the above, the Research and Development of new engines is nowadays based on mathematical models. In other words, what drives the state of the art research is the ability to be able to generate insightful mathematical models that simulate the behavior of an engine in practice. The aforementioned is what is called, a "digital-twin" of the system, which vastly contributes to accelerating the development of products as well as

other things like predictive maintenance [6].

In this context, with the development of electronic engines in the recent decades, the optimal control of them has gained importance as a decisive factor of their performance, emissions as well as fuel consumption. The central part of an electronic engine is it's Engine Control Unit (ECU). The main input and output signals of the engine originate from the ECU and govern it's behavior. In order to make correct decisions regarding actions like the spray timing, accurate, informative and timely input signals are required. The implementation of a virtual sensor in the engine, allows for reduced equipment complexity, faster calculations, a bypass when a signal is not available/ easily measured as well as the ability to introduce new metrics previously not available easily, as for example the content of NO_X in the exhaust gases.

1.2 Problem Framework

Regarding the current thesis the problem concerns the development of a virtual sensor for the hybrid engine **HIPPO-2** of the Laboratory of Marine Engineering of the National Technical University of Athens. The purpose of the virtual sensor, is to predict certain parameters that are considered important for the engine operation, using deep convolutional neural networks that receive as input other engine parameters that are easily and robustly measured/ observed. Of particular interest, are the content of NO_X, the air-fuel ratio λ as well as the fuel consumption and manifold pressure.

The applicability of convolutional neural networks for the prediction of time series data is investigated. The concept is that due to the convolution, the networks will "filter" the data and recognise certain patterns that will allow for a reduced-order representation of the physics involved in the problem and consequently enable real-time observability. The design of a virtual sensor for an engine requires that one understands the basic physics behind certain phenomena. It is not just a black box implementation and besides statistics these need to be considered. The mapping of the input data to the output data is evaluated initially off-line, meaning in a data set not trained.

In order to train the models, an adequate and large enough data-set is required which is obtained from the Laboratory of Marine Engineering with scientific measurements that have taken place in the past during loading cycles of the engine. The data are analyzed statistically in order to obtain an understanding of their interaction/ correlation. It is important to choose wisely the inputs of the models for robust predictions with low computational cost. It is noted, that the signals used are only signals available outside the engine originating from the ECU. Another aspect is the accuracy of the physical sensors as well as the ability to easily obtain those measures from a normal marine diesel engine.

Of significant importance, is the implementation of the models developed in real-time engine operation. For this purpose, a budget option is utilized with the use of a small single-board computer, widely used by hobbyists, the **Raspberry Pi**. In this application, the collection of signals, their timing and priority, play a significant role and are made through a CAN bus protocol. A decoding as well as a pre-processing of the information is then materialized, followed by the model-based prediction.

1.3 Literature Review

The accurate modelling of engine parameters involves the complex simulation of both thermodynamic, chemical and fluid dynamics processes that require significant computational resources. On that account, on- board computational applications must settle for a reduced accuracy. The goal is to be able to obtain an adequate accuracy while maintaining a low computational cost. An accurate model-based approach is proposed in [7] by Onder and Guzella, achieving a respectful performance for the virtual sensing of NO emissions. However, the first-principles model is limited in the steady state.

The most common and up to date approach for parameters estimation are the static maps or polynomial models where a large ammount of experimental data is required for the information to realized. Tschnz et al. [8] proposed a polynomial model based approach for the Soot sensing, on a steady- state map supplemented with a dynamic model for the transient charging of the engine. The authors underscore, the advantages of using the model for control but also point out the diminished extrapolation ability when provided with data outside the usual range of inputs.

As a result, as stated in [9] the solution should be a truly multidimensional, adaptive, learning technique that does not require the arduous development of an engine model, while having superb performance and emissions prediction capabilities across the full life of the engine, for all engine operating conditions. Neural network-based engine modeling offers all of these capabilities and taking advantage of this, neural networks-based virtual sensors have been developed for power-trains. A first approach for this kind of problem is presented in [10] where a vanilla multi layer perceptron (single layer feed forward neural network) is utilized for the prediction of the volumetric efficiency of the engine η_ν, required to be able to predict the manifold pressure based on three other engine parameters. The authors state the large amounts of applications of these models as black box identifiers, as well as the flexibility and the ease o implementation.

Another worthwhile peace of work is that of Arsie et al. [11], where a virtual sensor for NOx emissions in Diesel engines, is developed based on recurrent neural networks. The authors underline, the promising performance of the sensor in transient conditions, while stating that they have taken into consideration the physical mechanisms of the emissions formation. They also point out at the importance of obtaining data covering the whole engine domain, as well capturing both high and low frequency dynamics. The same author and his colleagues, extend this work in [12] taking into account the Selective Catalytic Reduction implemented in the system. Additionally, the algorithms are equipped with online least-squares adaptation in order to consider for effects such as aging of the engine.

More recent advances include, the use of deeper and more sophisticated architectures of neural networks. The Laboratory of Marine Engineering recent work [13] takes advantage of the Recurrent and Time-Delay Neural Networks for the prediction of NO_X emissions. It is reported that the real-time validation accurately captured the dynamic behavior of the system in most operating areas. It is also stated, that the developed models in comparison with a first-principle, physics-based virtual sensor and an engine map produced comparable results and executed seamlessly fast.

The primary use of Convolutional Neural Networks (CNN) is concerned with machine learning applications that have to do with imaging and computer vision. Nevertheless, cases where CNN's are used for time series classification [14] or modeling [15] have arisen

in the recent years. In [16], CNN's are used as tools to forecast time series, using the WaveNet architecture. They report, an efficient and easily interpretable network that can act as a strong baseline for forecasting. In [17], a multichannel CNN architecture is proposed to process time series in a temporal way for activity recognition. They report that the model constructed outperforms the state of the art. From the above, it can be concluded that the architecture of a CNN is application based and therefore multiple architectures will be explored in the current thesis in order to locate the optimal one.

1.4 Thesis Outline

The thesis outline is as follows:

- **Chapter 2** introduces the basic principles of neural networks, as well as the basic tools of the convolutional neural networks. Moreover, the procedure of training is presented.
- **Chapter 3** presents the engines major operational characteristics that are of interest for the current thesis. Particular importance is paid to the Fuel Consumption, the Air Fuel Ratio as well as the Nitrogen Oxides.
- **Chapter 4** presents the physical system under study, the hybrid engine. Additionally the training data-set is visualized and explored followed by a preprocessing of the data.
- **Chapter 5** describes the architectures of the models constructed as well as the basic building blocks. An exploratory data analysis is performed on the data set to determine the relationships between each parameter.
- **Chapter 6** showcases the training procedure as well as the results regarding the accuracy of the models both in training data as well as in validation data. The model structures are visualized.
- **Chapter 7** Final thoughts and remarks are discussed. Future work is proposed.

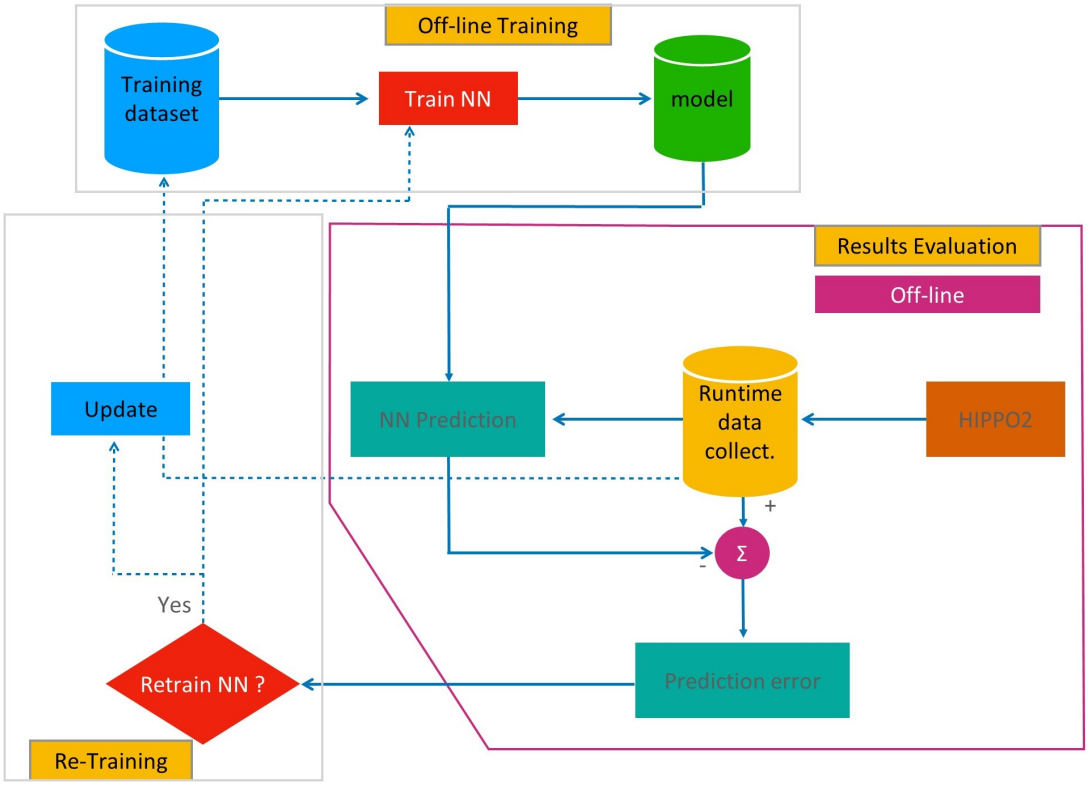

Figure 1.1: Flow Chart of the Thesis Outline

Chapter 2

Principles of Neural Networks

As engineering advances, humans are constantly looking for solutions that are able to revolutionize the industry. In this perspective, Machine Learning (**ML**) and it's applications have gained prominence in the recent decades as it seems that they can play a great role in solving problems that human engineering still possesses a significant role in setting the problem framework. The objective of ML, is to to make educated decisions or predictions based on given data. A major part of machine learning consists of *Supervised Learning*, which entails learning a mapping between a set of input variables X and an output variable t and applying this mapping to predict the outputs for unseen data [18]. The inputs x are also called features and the output t can be called target.

One of the algorithmic approaches of supervised learning are the Neural Networks (NN) and more specifically Artificial and Deep Neural Networks (DNN's). The term 'neural network' traces back it's roots to biology and the way the brain learns from experience [19]. A neural network is defined as an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the interunit connection strengths, or weights, obtained by a process of adaptation to, or learning from, a set of training patterns [20]. In the current chapter much of the content presented, is obtained from [21] and [22].

2.1 Supervised Learning Tasks

The two major tasks of supervised learning as utilized in engineering applications are the construction of models for regression and classification. In the current thesis, the goal is to predict the values of certain time-series given values of other time-series that are supposedly correlated with them, a task that is called regression.

Regression is concerned with the prediction/forecast of one or more continuous *target* variables y given the value of a D-dimensional vector x of *features*. An example of the regression task is the prediction of the temperature tomorrow based on observations of other parameters such as today's humidity and the prevailing winds. In other words, given a training data set consisting of N observations $\{x_n\}$, where $n = 1, ...N$, accompanied by corresponding target values $\{t_n\}$, the objective is predict the value of t for a new value of x.

In classification problems, the aim is given an input x to assign it to one of K discrete classes C_k where $k = 1, ..., K$. Typically, the classes are taken to be disjoint, meaning a one to one mapping of inputs to outputs. A binary classification problem example, is that given an image the model should be able to predict whether the animal is a cat ($y = 1$)

or not ($y = 0$). These type of problems, predicting the class label, are called pattern recognition.

2.2 Defining the Problem

As stated previously, the goal of the *learning process* is given a training data set consisting of N observations $\{x_n\}$ and corresponding target values $\{t_n\}$ to be able to predict a new value t. An naive approach, is directly constructing an appropriate function y(x) whose values for new inputs x constitute the predictions for the corresponding values of t. In other words, it is assumed that the target value is given by a deterministic function $y(\mathbf{x},\mathbf{w})$ with additive Gaussian noise,

$$t = y(\mathbf{x},\mathbf{w}) + \epsilon \tag{2.1}$$

where ϵ is a zero mean Gaussian random variable with precision (inverse variance) β. From a probabilistic perspective, we aim to model the predictive distribution $p(t|x)$ as this approach encapsulates the uncertainty of the value of t given a value of x.

$$p(t|\mathbf{x},\mathbf{w},\beta) = \mathcal{N}(t|y(\mathbf{x},\mathbf{w}),\beta^{-1}) \tag{2.2}$$

Utilizing this conditional distribution, we can make predictions of t, for any new value of x. In order to measure the effectiveness of the process the predictions should minimize the expected value of a conveniently chosen loss function. An ordinary option, of loss function for real-valued variables is the squared loss, for which the optimal solution is given by the conditional expectation of t. The after mentioned can easily be extended for multiple target values t and multiple dimensions.

$$Loss(guess, actual) = (guess - actual)^2 \tag{2.3}$$

2.3 Linear Basis Function Models

Despite the fact that, linear models have notable restrictions as practical techniques for regression, particularly for problems involving input spaces of high dimensionality, they posses nice analytical properties and form the foundation for the neural networks discussed later. Their biggest advantage, is their ability to always have a solution (fit) as their log negative likelihood is convex [21].

To parameterize and approximate a function that resembles a relationship between input and output data an elementary model for regression is the linear combination of the input variables. This approach is called *linear regression*,

$$y(\mathbf{x},\mathbf{w}) = w_0 + w_1x_1 + ... + w_Dx_D = w^Tx + w_0 \tag{2.4}$$

where $w^Tx =< w, x >$ is the inner product between the weight vector w and the feature vector x. This function defines a linear hyperplane, with normal vector $w \in R^D$ and an offset $w_0 \in R$ from the origin. However, the very simplicity of this approach, imposes limitations as only linear relationships can be approximated. Reinforcing the above model, linear combinations of fixed nonlinear functions of the input variables are introduced, performing what is called a feature transformation,

$$y(\mathbf{x},\mathbf{w}) = \sum_{j=0}^{M-1} W_j \phi_j(x) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \tag{2.5}$$

where $\mathbf{w} = (w_0, ..., w_{M-1})^T$ and $\boldsymbol{\phi} = (\phi_0, ...\phi_{M-1})^T$.

The $\phi_j(x)$ functions are called basis functions and the parameter w_0 a bias parameter which compensates for difference between average target values and weighted sum of averages of basis function values. This technique is called basis transformation and a simple example is a polynomial transform which for one dimensional data is given by $\phi(x) = [1, x, x^2, x^3, ...]$, the well known polynomial regression. In the above notation a dummy 'basis function' $\phi_0(x) = 1$ is used.

Although, the use of nonlinear basis functions allows for a nonlinear relationship between the function $y(\mathbf{x},\mathbf{w})$ of the input vector $\mathbf{x}$, the relationship still remains linear in terms of the weights $\mathbf{w}$. The drawback of the polynomial expansion is that polynomials are a global approximation to the function, meaning that local changes in variables alter globally the approximation. To improve flexibility, the use of a series of local approximations is proposed defining a set of basis functions that have local support, *B-Splines*. Another popular basis function, is the *sigmoidal* basis function of the form,

$$\phi_j = \sigma\left(\frac{x - \mu_j}{s}\right) \tag{2.6}$$

where $\sigma(\alpha)$ is the *logistic sigmoid function*,

$$\sigma(\alpha) = \frac{1}{1 + exp(-\alpha)} \tag{2.7}$$

An analogous approach is the use of '*tanh*' function as it is directly related to the logistic sigmoid with the relationship $tanh(\alpha) = 2\sigma(\alpha) - 1$. Other choices to construct what is called *Linear Basis Function Models* include the use of *Fourier* basis or *wavelets*.

2.4 Neural Networks

As the parameters in a regression problem grow, the need for use of more basis functions arises complexing the optimization problem. A natural extension to the basis function expansion is to equip the feature extractor, that is to say the basis functions, with its own parameters. In this manner, the number of the basis functions is fixed, but is allowed to be flexible and adapt its parameters during training. A general representation of the models constructed for regression and classification, based on combinations of fixed nonlinear basis functions is the following,

$$y(\mathbf{x},\mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right) \tag{2.8}$$

where $f(\cdot)$ denotes a nonlinear activation function, a function that connects an input to an output, $f : \mathcal{R} \to \mathcal{R}$. In the case of regression the activation function vanishes to a plain identity. The creation of basis functions $\phi_j(\mathbf{x})$, that depend on parameters of course can be executed in multiple ways. A convenient approach is, to construct basis functions that are of the form of the equation (2.8) so that each basis function is itself a linear combination of the inputs, where the coefficients in the linear combination are adaptive parameters.

This construction can be repeated recursively, to create more and more complex functions, which is the main idea behind Deep Neural Networks (DNN).

The before mentioned reasoning gives rise to the basic neural network model, which is composed by applying a sequence of these functional transformations. Initially, M linear combinations of the input variables $x_1, ..., x_D$ are computed,

$$\alpha_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{2.9}$$

where $j = 1, ..., M$. The number (1) of the superscript indicates the number of the *layer* that the *weights* parameters $w_{ji}^{(1)}$ and *biases* $w_{j0}^{(1)}$ belong to. The quantities α_j are called *activations* or *pre-activations*. Following that, a transformation is performed utilizing a differentiable, nonlinear activation function $h(\cdot)$ that yields,

$$z_j = h(\alpha_j) \tag{2.10}$$

The z_j quantities correspond to the outputs of the basis functions in (2.8) that, in the context of neural networks, are called *hidden units*. The nonlinear functions $h(\cdot)$ are, depending on the application, for example, chosen to be sigmoidal functions such as the logistic sigmoid for classification. Following (2.8), these values are again linearly combined to give *output unit activations*.

$$\alpha_k = \sum_{i=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{2.11}$$

where $k = 1, ...K$, and K is the total number of outputs. Similarly, the above transformation resembles the second layer of the network. Merging the two transformations and applying an activation function $f(\cdot)$ of our choice the resulting network is the following,

$$y_k(\mathbf{x},\mathbf{w}) = f\left(\sum_{i=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{2.12}$$

Equation (2.12), implies that the above neural network model is an nonlinear mapping from a set of input variables x_i to a set of output variables y_k controlled by a vector $\mathbf{w}$ of adjustable parameters. With some algebraic manipulation the same equation can be rewritten as,

$$y_k(\mathbf{x},\mathbf{w}) = f\left(\sum_{i=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i\right)\right) \tag{2.13}$$

Equation (2.13) encloses two steps of processing, each of which can be viewed as a perceptron [23], with the difference of continuous nonlinear activation functions, instead of non-differentiable Heaviside functions. In other words, it resembles a *multilayer perceptron* (MLP), which is in the current context will be called *feed forward neural network*. The key property of this type of networks, is that they are differentiable, laying the foundations for network training.

Another interesting property of these type of networks is that, if we choose the activation functions of all the hidden units to be linear then given any such network, it is proven that an equivalent network without hidden units always exists. This follows from the fact

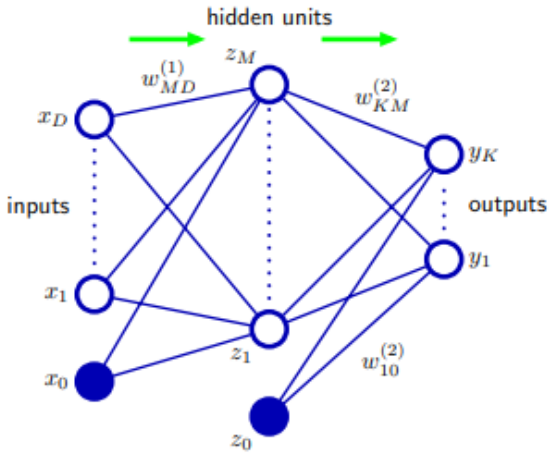

Figure 2.1: Network Diagram corresponding to (2.12).The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links inputs coming from additional input and hidden variables x_0 and z_0. Arrows showcase the direction of information flow through the network during *forward propagation*

that the composition of successive linear transformations is itself a linear transformation. However, if the number of hidden units is smaller than either the number of input or output units, then the transformtions that the network can generate, are not the most general possible linear transformations from inputs to outputs because information is lost in the dimensionality reduction at the hidden units.

Additionally, it is proven that networks with this type of architecture are very good *global approximators*. In [24], it is demonstrated that any continuous function can be uniformly approximated by a continuous neural network having only one internal, hidden layer and with an arbitrary continuous sigmoidal nonlinearity. Although, the properties proved are really strong, the authors state that, the important questions that still has to be clarified is concerned with feasibility, namely how many terms in the summation (or equivalently, how many neural nodes) are required to yield an approximation of a given quality. This is the spot where the engineer intervenes, locating the the trade-off between desired accuracy and computational effort.

The network architecture shown in figure (2.2) is the one that is actually most frequently utilized. Despite that, a plethora of variations exist and are developed constantly. For instance by considering additional layers of processing each consisting of a weighted linear combination of the form (2.11) followed by an element-wise transformation using a nonlinear activation function. Other options include, for example using *skip-layer* connections.

2.4.1 Network Training

Drawing an analogy with the subject of polynomial curve fitting, a straightforward strategy for solving the issue of calculating the network parameters is the minimization of a sum-of-square error function. Given a training set comprising a set of input vectors $\mathbf{x}_n$, where $n = 1, ..., N$, together with a corresponding set of target vectors t_n, the error function is given as,

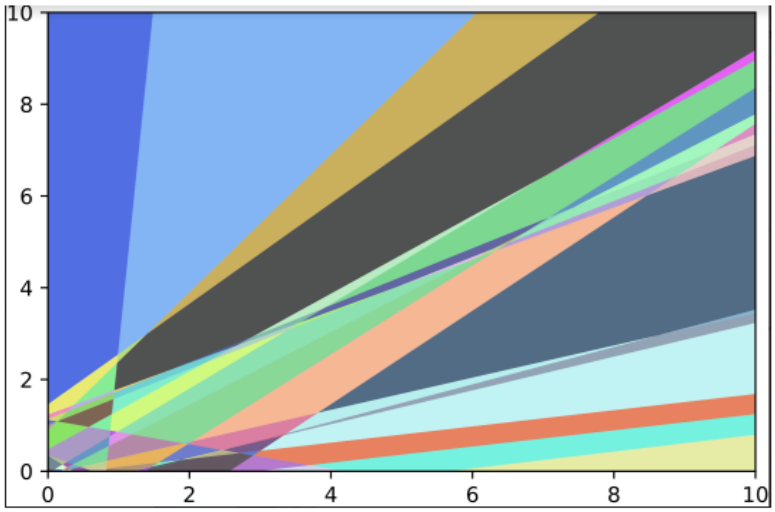

Figure 2.2: A decomposition of $\mathcal{R}^2$ into a finite set of linear decision regions produced by an *MLP*. The intuition tells us that the reason for such good approximation properties, is that each hidden unit can specify a half plane, and a sufficiently large combination of these can "carve up" any region of space, to which we can associate any response [**?**].

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2 \tag{2.14}$$

For the conditional distribution given by (2.2), given that such a network should approximate any continuous function, it is necessary to assume that the output unit activation function is the identity, from $\mathbf{x}$ to y. Similarly, given a set of N independent, identically distributed observations $\mathbf{X} = \mathbf{x}_1, ..., \mathbf{x}_1$, along with corresponding target values $\mathbf{t} = t_1, ..., t_N$, the corresponding likelihood function is constructed,

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta) \tag{2.15}$$

Taking the negative logarithm, we obtain the error function,

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}ln\beta + \frac{N}{2}ln(2\pi) \tag{2.16}$$

which is utilized to compute the parameters w and β. The maximization of (2.16) coincides with the minimization of the sum-of-squares error function, which is the normal practice in modern deep learning,

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - y_n\}^2 \tag{2.17}$$

The value of $\mathbf{w}$ that is obtained by minimizing $E(\mathbf{w})$ is designated as $\mathbf{w}_{ML}$ as it corresponds to the maximum likelihood solution. With the knowledge of the value of $\mathbf{w}_{ML}$ the value of β can be computed with the minimization of the negative log likelihood that yields,

$$\frac{1}{\beta_{ML}} = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w_{ML}) - y_n\}^2 \tag{2.18}$$

It can be showcased, that there is a natural pairing of the error function (given by the negative log likelihood) and the output unit activation function. In the regression scenario, the network can be considered as having an output activation function that is the identity, $y_k = a_k$. The associated sum-of-squares error function is characterized by the following property,

$$\frac{\partial E}{\partial \alpha_k} = y_k - t_k \tag{2.19}$$

which will be utilized in the *error backpropagation*.

Parameter Optimization

The next step, is finding a weight vector $\mathbf{w}$ that minimizes the selected function $E(\mathbf{w})$. From a geometric point of view, the error function can be seen as a surface that lies above the weight space as in figure (2.3).

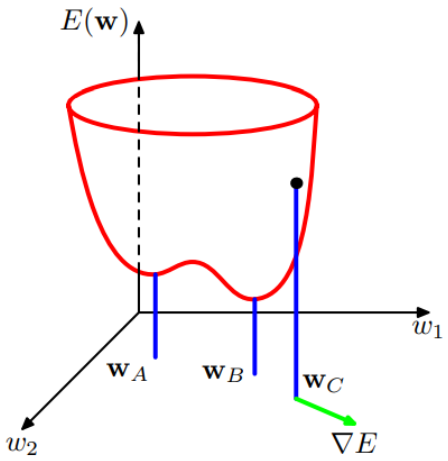

Figure 2.3: Geometrical view of the error function $E(\mathbf{w})$ as a surface sitting over weight space. Point $\mathbf{w}_A$ is a local minimum and $\mathbf{w}_B$ is the global minimum. At any point $\mathbf{w}_C$, the local gradient of the error surface is given by the vector ∇E.

As the error function $E(\mathbf{w})$ is a smooth function of $\mathbf{w}$, the points where the gradient vanishes with respect to the weight space are defined as *stationary points* and can be further categorizes into minima, maxima and saddle points. From the basic principles of Vector Calculus we know that a small step in the direction of $-\nabla E$ leads closer to a stationary point and thus to a smaller value of the error function. As the objective is to find a vector $\mathbf{w}$ that minimizes $E(\mathbf{w})$ we are presented with an optimization problem. What makes this optimization, tricky is that usually the error function (whichever is chosen) is non-linearly dependent on the weights and biases and as a result plenty of weight space points, where the gradient vanishes, emerge.

It is also common to have many non-equivalent stationary spots, particularly multiple non-equivalent minima. A global minimum is one that corresponds to the minimal value of the error function for any weight vector. Any further minima that have larger error function values are referred to as local minima. It may not be required to identify the global minimum for a successful application of neural networks (and typically, it won't be known whether the global minimum has been identified), but it may be necessary to compare numerous local minima in order to find a sufficiently good solution. This is where the *learning rate*, as discusses later, plays a significant role.

The optimization problem is of course solved numerically. Plenty of methods exist in the literature with the most, relying on an iterative scheme. The process initiates with an conveniently chosen initial value for the weights $\mathbf{w}^{(0)}$ that may be resembled by a heuristic approach. For example, the weights may be initialized by a *normalized initialisation*, as suggested in [25],

$$w_{i,j} \sim U\left(-\sqrt{\frac{1}{m+n}}, \sqrt{\frac{1}{m+n}}\right) \tag{2.20}$$

where m is the number of inputs to the layer, n and U the uniform distribution. The initialisation of the weights affects the optimization problem as it might be decisive as to where the gradient will lead to. From this point on the procedure advances through the weight space in recursive steps in the following manner,

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{2.21}$$

where τ labels the iteration step. The way the weight vector $\Delta\mathbf{w}^{(\tau)}$ is updated depends on the technique utilized.

2.4.2 Gradient Descend

Plenty of algorithms utilize the gradient information in order to update the weights in (2.21). The most common strategy is take the weight update to be a small step in the direction of the negative gradient, that gives,

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w}^{(\tau)} \tag{2.22}$$

where the parameter $\eta > 0$ is called the *learning rate*. The gradient is reevaluated for the new weight vector following each such update, and the procedure is then repeated. It is worth mentioning that the error function is defined with respect to a training set, meaning that in each step the whole of the training set needs to be evaluated in order to compute the new ∇E. The methods that make use of the entire data set simultaneously are called *batch* methods, that are also used in the current thesis.

The identification of an adequate minimum that satisfies the desirable approximation accuracy requires that a try-error process is performed starting, from different randomly chosen initial weights and learning rate and assessing the performance/ outcome. However, there is an online implementation of gradient descent that has been successful in training neural networks on massive amounts of data . Maximum likelihood error functions for a set of independent observations consist of a sum of terms with one term for each data point.

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \tag{2.23}$$

One data point individually is used to update the weight vector in online gradient descent, sometimes referred to as *sequential gradient descent* or *stochastic gradient descent*. This method converges quicker than the one in relationship (2.22) but fails to minimise the error close the vanishing gradient point. However, whether this effect is positive or negative depends on the point of view as points called *flat minima* might be sought after as they make such solutions more robust and easily generalized. This problem is associated with a common problem in neural networks, *over-fitting*.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}) \tag{2.24}$$

By cycling through the data either in order or by picking random points with replacement, this update is repeated. Of course, there are intermediate situations when the updates are based on collections of data points, *mini batches*. Equation (2.24) can be reinforced with a plethora of ways, including for example the momentum method [26].

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}) + \alpha \Delta \mathbf{w}^{(\tau)} \tag{2.25}$$

More advanced options will be discussed later.

2.4.3 Backpropagation

Finding a method that is effective and fast for assessing the gradient of an error function $E(\mathbf{w})$ for a feed-forward neural network is the next step. This is accomplished by the famous *backpropagation algorithm*, which resembles the back propagation of the error in the network. As in neural network training the metric of the performance is usually a sum-of-square error type function of the form (2.23), that is similar to the one used in least-squares regression. The key distinction is that, unlike classical regression, which starts with the assignment of an explicit model generated using analytical knowledge of the system under study, backpropagation develops an implicit model. The form of this model is only restricted by the bounds on the set of all functions that the selected architecture can implement. Consequently, backpropagation learning can be considered to be a generalization of classical regression, a sort of 'hyper regression' ([27].

Starting with a plain linear layer model, for which the outputs y_k are given as linear combinations of the input variables x_i,

$$y_k = \sum_i w_{ki} x_i \tag{2.26}$$

Following that, we introduce the an error function that, for a particular input pattern n,

$$E_n = \frac{1}{2} \sum k(y_{nk} - t_{nk})^2 \tag{2.27}$$

where $y_{nk} = y_k(x_n, w)$ and the relevant network loss function (2.23). Taking the gradient of this particular error function with respect to a weight w_{ji} we get,

$$\frac{\partial E_n}{\partial w_j i} = (y_{nj} - tnj) x_{ni} \tag{2.28}$$

which can be understood as local estimation of an error $y_{nj} - t_{nj}$ connected with the output w_{ji} of the link and magnified by the input x_{ni} of the link. Generalizing the above, we can obtain a general feed forward neural network, each unit being the form of

$$a_j = \sum_i w_{ji} z_i \tag{2.29}$$

and,

$$z_j = h(a_j) \tag{2.30}$$

For each pattern in the training set, we will assume that we have supplied the corresponding input vector to the network and calculated the activations of all of the network's

hidden and output units by applying (2.29) and (2.30). Because there is a forward flow of information through the network, this process is often referred to as *forward propagation*.

By applying the chain rule and neglecting the form of the input pattern n we get,

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial \alpha_j} \frac{\partial \alpha_j}{\partial w_{ji}} \tag{2.31}$$

We introduce a new parameter,

$$\delta_j = \frac{\partial E_n}{\partial \alpha_j} \tag{2.32}$$

After some algebra we obtain,

$$\delta_j = h^{'}(a_j) \sum_k w_{kj} \delta_k \tag{2.33}$$

where k are the output units. Equation (2.33) is the *backpropagation formula* which indicates that the value of δ for a specific hidden unit can be computed by propagating the δ's backwards from the units higher up in the network as in figure (2.4)

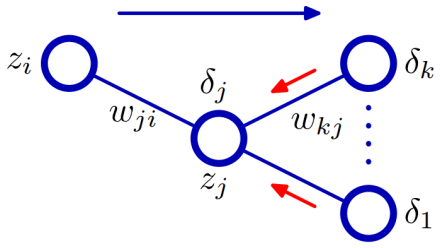

Figure 2.4: Backpropagation demonstration in a hidden unit *j*. Blue arrow for forward propagation and red arrow for backward propagation of error information

Finally, the derivative of the total error can be computed for the whole batch as as a simple summation for every input pattern.

2.5 Convolutional Neural Networks

A *Convolutional Neural Network* (CNN), introduced by [28] is a type of deep neural network usually utilized for applications that have to do with imaging or some kind of grid. In Convolutional Neural Networks the usual matrix multiplication is replaced with the *convolution operation*. The main reason that these networks were initially developed, is their ability to be invariant to certain transformations of the input. For example, if an image of a handwritten digit is provided, no matter the (small) rotation of it, the location in the picture, or the scale of it, the networks should interpret the information the same way.

Considering an image input, convolutional neural networks are characterized by three main features that enable them for good predictions:

1. loacal receptive fields: meaning they extract local features for nearby pixels,

2. weight sharing: local features that are useful in one region might be useful also in other regions,

3. subsampling: later stages of processing in the network use information from merged features to detect higher-order features.

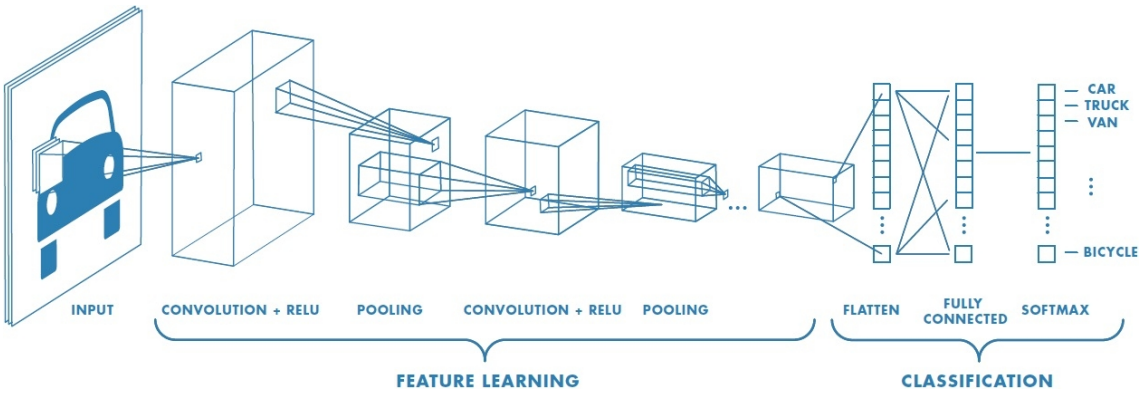

Figure 2.5: A typical modern CNN for image classification

Additionally, considering again an input of an image $x \in \mathcal{R}^{WHC}$, where W is the width, H is the height, and C is the number of input *channels* (C=3 for RGB color), the weight matrix of a simple Feed Forward NN would be required a size of $(W \times H \times C) \times D$, where D is the number of outputs (hidden units). It is obvious that this number gets really big for reasonably sized images. The basic idea behind CNN's is to divide the input into overlapping 2d image patches and compare each patch with a set of small weight matrices, or *filters*, which represent parts of an object. This can be thought of as a type of template matching. The number of parameters is significantly reduced because the templates are small (for example 3x3 or 5x5).

2.5.1 Convolutional Layers

Convolutional Operator

Convolution is a mathematical operation performed on two functions (f and g) to produce a third function, that expresses how the shape of one is modified by the other. The mathematical formulation of it, in the case of 1-dimension, is defined as follows,

$$(f * g)(t) = \int_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau) \tag{2.34}$$

with $(*)$ being the convolution operator. In machine learning terminology, the function $f(\cdot)$ resembles the *input* and function $g(\cdot)$ resembles what is called a *filter* or *kernel*. The output function of the convolution, also called convolution, is referred to as *feature map*. Popular engineering applications that take advantage of the convolution include signal processing, relevant to the current thesis.

In the case of machine learning the continuous functions are replaced with finite-length vectors, a kind of function defined in certain points. The evaluation of f in certain points $\{-L, -L+1, ..., 0, 1, ..., L\}$ yields the weight vector $w_{-L}...w_L$. The same goes for the evaluation of g at point $\{-N, ..., N\}$ that produces the feature vector $x_{-N} = g(-N)...x_N = g(N)$. Then, supposing that the weight vector is symmetric, the equation (2.34) takes the following form

$$(w * x)(i) = w_{-L}x_{i-L} + ... + w_{-1}x_{i-1} + w_0x_i + w_1x_{i+1} + ... + w_Lx_{i+L} \tag{2.35}$$

which is called *cross correlation* and is usually the case in machine learning where the weight vector is symmetric. This exact property is what makes the application of CNN's really interesting for time-series prediction. The above can be rewritten in order to eliminate negative indices,

$$(w * x)(i) = \sum_{u=0}^{L-1} w_u x_{i+u} \tag{2.36}$$

In 2d, for a 2d filter $\mathbf{W}$ with a size $H \times W$ we get:

$$(\mathbf{W} * \mathbf{X})(i,j) = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} w_{u,v} x_{x+u,j+v} \tag{2.37}$$

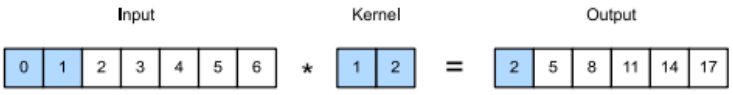

Figure 2.6: 1d cross correlation

Other Features

Of course, the convolution is generalized for multiple dimensions and multiple channels. Other features include what is called *padding*. This means, that if the size of the output is desired to be the same size as the input, a border of 0s is added in the boundaries of the matrixes. *Stride* is also utilized meaning skipping the filtering every some number of input.

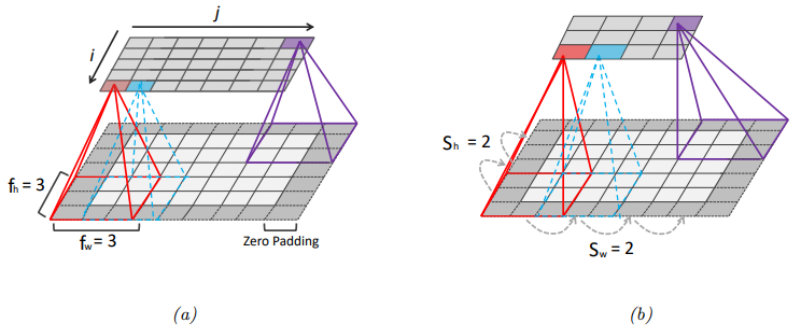

Figure 2.7: Illustration of padding and strides in 2d convolution. *(a)* We apply "same convolution" to a 5×7 input (with zero padding) using a 3×3 filter to create a 5×7 output. *(b)* Now we use a stride of 2, so the output has size 3×4.

2.5.2 Pooling Layers

Equivariance is a property of convolution that preserves information about the location of input features (modulo reduced resolution). In some cases, we want to be location-independent. When performing image classification, for example, we may simply want to know if an object of interest (e.g., a face) is present anywhere in the image.

One simple method is max pooling, which simply computes the maximum over its incoming values, as shown in Figure 14.12. Another option is to use average pooling, which replaces the maximum with the mean. In either case, the output neuron responds in the same way regardless of where the input pattern occurs within its receptive field (we apply pooling to each feature channel separately).

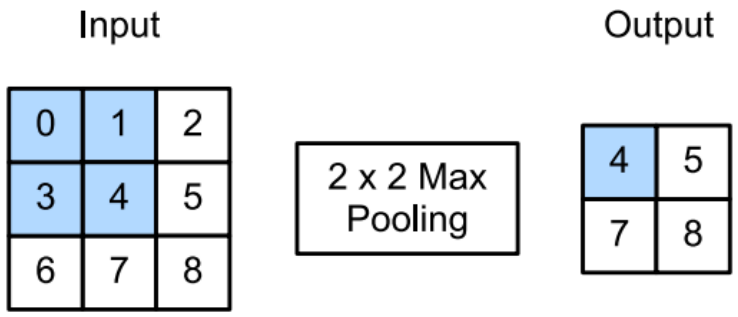

Figure 2.8: Illustration of maxpooling with a 2x2 filter and a stride of 1

2.5.3 Normalization Layers

Batch normalization (BN) is the most widely used normalization layer . When the activation distribution within a layer is averaged across the samples in a minibatch, it has a zero mean and unit variance. More specifically, we replace the activation vector z_n (or, in some cases, the pre-activation vector a_n) for example n (in some layer) with $\tilde{z}_n$n, which is calculated as follows,

$$\tilde{z}_n = \gamma \hat{z}_n + \beta \tag{2.38}$$

$$\hat{z} = \frac{z_n - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.39}$$

$$\mu_B = \frac{1}{|B|} \sum_{z \in B} z \tag{2.40}$$

$$\sigma_B^2 = \sum_{z \in B} (z - \mu_B)^2 \tag{2.41}$$

where, B is the minibatch containing example n, μ_B is the mean of the activations for this batch, σ_B^2 is the corresponding variance, $\hat{z_n}$ is the standardized activation vector, $\tilde{Z_n}$ is the shifted and scaled version (the output of the Batchnorm layer), β and γ are learnable parameters for this layer, and $\epsilon > 0$ is a small constant.

Chapter 3

Engine Operational Characteristics

In this chapter the basic physical parameters characterizing the performance and behavior of a diesel engine are presented. While in the past, the focus lied on the efficiency, today it also lies on the emissions side. Measuring and predicting the parameters of the engine is useful for control purposes but also, for monitoring. For example, engine deterioration can be be measured effectively comparing expected values, given by a model, and actual values measured real time. Following that the engine used for the model development is presented along with the data collection process.

3.1 Main Engine Parameters

In the study of Internal Combustion Engines it is common to introduce some major parameters that are descriptive of the thermodynamic and chemical processes occurring. These include the temperature before and after the engine (exhaust temperature). Additionally, the pressure after the engine is an important parameter indicating the load on the engine. Another parameter, is the Torque produced by the engine which in Marine Engineering applications produces the thrust required by the propeller. Finally, the engine speed, in other words the engine cycle reciprocations per second. Other parameters that are of particular importance for modelling are presented below.

3.1.1 Engine Fuel Consumption

Ships are propelled through their main drive-train, a diesel engine usually for merchant ships. These engines convert chemical energy (hydrocarbons) to kinetic through the combustion of the fuel, a fast chemical reaction. The fuel in use in marine diesel engines is the Heavy Fuel Oil. The fuel consumption is measured in terms of mass or volume of fuel burned divided by the time interval. The fuel consumption is a really important factor, as it is the major daily cost of the operation of a ship. Also, it significantly affects the emissions as the more the consumption, for a given power output, the more the emissions. This interest has been largely magnified by the introduction of the *EEXI* metric, that all she merchant ships should comply by the first periodical survey 2023. The fuel consumption is directly incorporated in the attained EEXI metric,

$$EEXI\left(\frac{gr}{tonmile}\right) = \frac{FOC \times C}{Capacity \times V_s}$$

where *FOC* is the fuel consumption and C a factor the multiplied with it yield CO_2 emmisions. As a result, it is one of the most closely monitored parameters of the ship, traditionally measured through noun-reports. Despite that, with the advent of digitalization new possibilities arise. Last but not least, the fuel consumption is directly correlated with other engine parameters that are of our interest for prediction.

3.1.2 Intake Manifold Absolute Pressure (MAP)

The Inlet Manifold Absolute Pressure corresponds to the pressure measured at the intake manifold of the engine, before entering the cylinders. This pressure is build by the compressor of the turbocharger of the engine and is also called 'turbocharger boost pressure' and is required in order for the engine to be able to burn more fuel. The measurement of MAP is directly provided to the ECU of the engine and utilized for the calculation of air density and consequently the air mass flow rate. The air mass flow rate determines the correct air-fuel ratio for complete combustion provided by the fuel metering, as well as the advance or retard of ignition timing.

3.1.3 Air-fuel equivalence ratio and Lambda (λ)

For the combustion of the fuel to occur, oxygen is required for the chemical reaction. Adequate amounts of air need to be provided in the combustion chamber for effective and safe combustion. *Complete Combustion* is defined as the chemical reaction of hydrocarbons (without impurities) in the presence of enough oxygen that produces water vapor and carbon dioxide while the whole quantity of fuel is oxidized. *Air-fuel ratio* (AF or AFR) is the ratio between the air mass m_a and fuel mass m_f, present in the engine at a given moment,

$$AFR = \frac{m_a}{m_f} \tag{3.1}$$

The air-fuel ratio, required for a *complete* combustion to occur is called *stoichiometric*. In the case of diesel fuel the ratio is around 14.5 grams of air for a complete combustion of one gram of fuel. When the air-fuel ratio is greater than the stoichiometric ratio, the air-fuel mixture is said to be *lean*, and when it is less than the stoichiometric ratio, the air-fuel mixture is said to be *rich*.

In real operation, diesel engines do not work at the stoichiometric point, but rather on the lean side with AFR values in the range of 1:18 to 1:70 depending on the operating conditions. The ratio between the actual air-fuel ratio (AFR_{actual} and the stoichiometric one is called *Air-Fuel Equivalence Ratio* or *Lambda* (λ) and is defined as,

$$\lambda = \frac{AFR_{actual}}{AFR_{stoichiometric}} \tag{3.2}$$

A 'rough' estimate is that for full throttle the value of λ is in the range of 1.2-2. Figure (3.1) showcases the connection between AFR and loads and the importance of incorporating model building first-principle physics in model building.

3.1.4 Nitrogen Oxides (NOx)

Diesel engines have relatively low engine-out emissions. In particular, hydrocarbon and carbon monoxide can usually be neglected. The main pollutant species in the exhaust gas are nitrogen oxide (NO_X), particulate matter (PM), and sulphur oxides SO_Xwhich

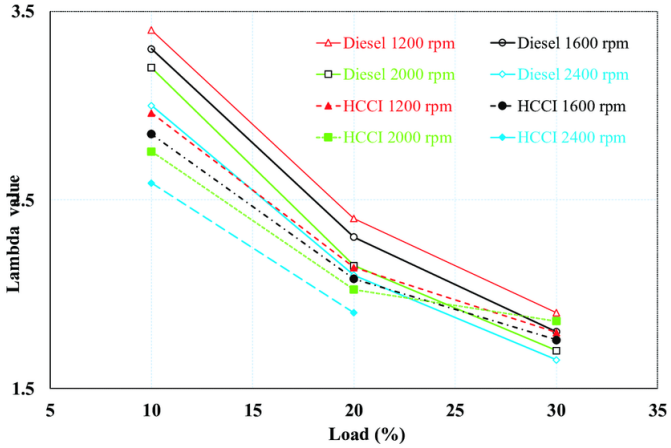

Figure 3.1: Lambda values of the diesel and the HCCI engines at different speeds and loads [1].

has been addressed in the recent regulations regarding the use of scrubbers or low sulphur content fuel. Consequently, the control and monitoring of NO_x's content in the exhaust gases is really important.

Formation of NOx

With the use of chemical kinetics, it can be concluded that the quantity $\frac{d[NO]}{dt}$ is determined by the temperature of the working fluid and the oxygen concentration. High values of these concentrations, combined with sufficient time windows for the reactions to occur, result in high formation rates of NO. For an adiabatic-constant pressure combustion the NO formation peaks for a stoichiometric composition and rapidly vanishes as the mixture becomes leaner/richer. A side effect of this formation is that NO formed in the flame zone can be rapidly converted to NO_2 reacting with water [29].

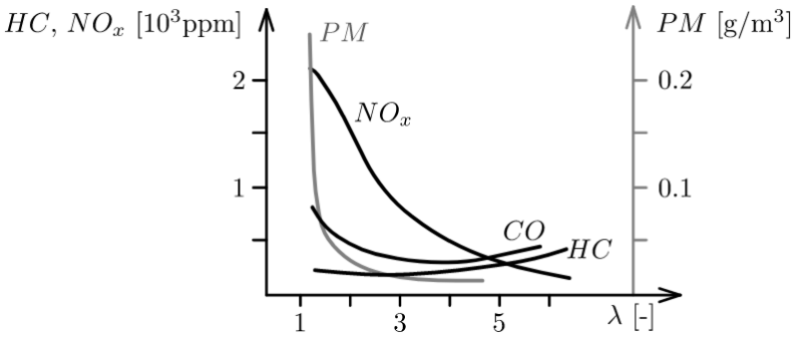

Figure 3.2: Engine-out emission of Nitrogen oxide NO_X, hydrocarbon HC, and particulate matter PM of a direct-injection Diesel engine as a function of air-fuel ratio λ [2]. There is a contradiction between, fuel consumption and low hydrocarbon emission for stoichiometric mixtures and low NO_x emissions for leaner mixtures.

NOx Tier III Requirements

The NO_x control requirements of Regulation 13 of MARPOL Annex VI apply to each marine diesel engine with a power output greater than 130 kW installed on a ship. The

Tiers applied depend on the ships engine, namely it's revolutions per minute, as well as the operational area of the ship. Tier I and Tier II limits are applied globally while Tier III standards are currently applied only on *Emissions Control Areas*. Consequently, the actual monitoring of the vessels emissions is of vital importance in order for it to comply with regulations. Physical measurements are a bit tricky and this is where a model-substitute of an physical sensor could play a role. Additionally, new techniques regarding NO_X-based control have arisen.

Tier	Ship construction date on or after	Total weighted cycle emission limit (g/kWh) n = engine's rated speed (rpm)		
		$n < 130$	$n = 130 - 1999$	$n \geq 2000$
I	1 January 2000	17.0	$45 \cdot n^{(-0.2)}$ e.g. 720 rpm - 12.1	9.8
II	1 January 2011	14.4	$44 \cdot n^{(-0.23)}$ e.g. 720 rpm - 9.7	7.7
III	1 January 2016	3.4	$9 \cdot n^{(-0.2)}$ e.g. 720 rpm - 2.4	2.0

Figure 3.3: MARPOL Annex VI for NO_X emissions

3.1.5 Exhaust Gas Recirculation (EGR)

Exhaust Gas Recirculation, is one of the most prominent techniques to significantly reduce internal combustion engines Nitrogen Oxides emissions. The operating principle of the installation of such a system, is that a portion of the exhaust gases is recirculated back to the intake manifold ans subsequently to the combustion chamber. This portion, is defined as,

$$EGR\% = \frac{m_{EGR}}{m_i}, \tag{3.3}$$

where $m_i = m_{air} + m_{fuel} + m_{EGR}$ and m_{EGR} the mass of exhaust gases recirculated back. The idea is that, due to the presence of the exhaust gases , a 'dilutent' effect takes place, which in combination with the high specific heat rate (c_p) associated with the triatomic molecules, result in the direct reduction of the adiabatic flame temperature and consequently on the formation of 'thermic' NO_x. Concurrently, the oxygen content is reduced which further decreases the production of nitrogen oxides. However, the above come at a price as negative effects include worsening of specific fuel consumption and particulate emissions as well decreasing lubricating oil quality and engine durability [30]. From the above, it is easily concluded that the content of NO_X during the operation of an engine is directly correlated with the EGR command of the engine.

The EGR system is active, mainly during partial engine loads and at low and medium engine speed areas, where oxygen is in excess. In the high engine loads (torque), the EGR system is deactivated, the cylinders being filled only with air, ready for combustion. There are two different kinds of external EGR systems: Low Pressure and High Pressure, which differ in how much pressure the recirculated exhaust gases are under. The High Pressure type is installed in the experimental setup for this study, as shown in figure (3.4), where the exhaust gases are collected before entering the turbine and then reintroduced in the intake manifold after the compressor. It is crucial to remember, that although while figure (3.4) does not exactly correlate to the installed EGR system of the test-bed in this instance, it has been included for convenience.

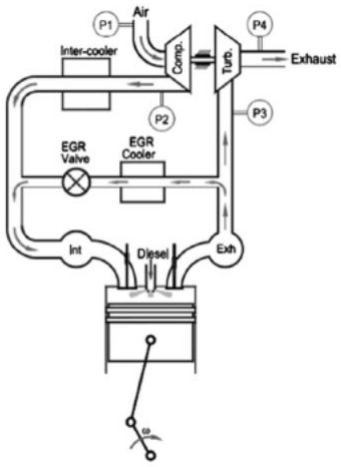

Figure 3.4: A typical *High Pressure* EGR system.

One of the main components of an EGR system is the EGR Valve, which allows the exhaust gases to recycle back from the exhaust manifold into the intake manifold. The EGR initiates once the engine has started and attained a stable operating temperature. At low speeds, and therefore at low loads, a small portion of oxygen is required, so the valve gradually opens. However as more torque and power is required, for example during full acceleration, the EGR valve closes to ensure as much oxygen enters the cylinder. In the scope of the current thesis, a crucial parameter, for the model building, is the EGR Valve Position, percentage wise, meaning the percentage of how open is the valve. This parameter is directly correlated, from a physical perspective, to the NO_X content in the engine.

Chapter 4

Data Processing

In order to create accurate predictive models that capture the physics of the system, a deep understanding of the physical model is required. To do that, it is necessary to have an intuitive grasp of the data collected and to assess their linkages and correlations. In the current thesis the form of the data, are signals-time series gathered from sensors mounted on an experimental test-bed. The accurate capturing of the data, the synchronization and cleaning of it significantly affect the process of developing a superior performing virtual sensor. Traditionally, in machine learning, the following steps for robust model development are suggested: Feature Selection for input, Data Transformation, Feature Engineering and Dimensional Reduction.

4.1 The Physical Model

The experimental facility **Hybrid Integrated Propulsion Powertrain 2 (HIPPO-2)** test bed, seen in figure (4.1) , consists of three main subsystems: the Internal Combustion Engine (ICE), an Electric Motor/ Generator (EM) and an Electric Brake (EB), all mounted on a common rotating shaft. This means, that the ICE and the EM contribute together, at the same rotations speed, to compensate for the torque demand simulated by the water brake. The motivation for such *hybrid* systems is the ability of the EM to assist the ICE in low speed and load (1200 rpm and 30% of nominal load), where the fuel and emissions efficiency of the ICE are lower. Additionally, as ICE transient response is poor, primarily due to the turbocharger lag phenomenon, the EM dynamics allow for immediate torque availability.

The EM is a standard ABB made, AC induction 3-phase, 4-pole motor, with a rated power of 90 kW at 1483 rpm. An 9.3-liter, 6-cylinder, turbocharged and after-cooled CATERPILLAR, CAT 9.3 $ACERT^{TM}$ industrial diesel engine powers the hybrid setup. According to its load diagram , this engine can provide a maximum torque of 1597 Nm at 1400 rpm, and a maximum power of 261 kW at 1800-2200 rpm. Based on the speed reference set and the speed measurement deviation, the electronic control unit (ECU) regulates the engine speed. Following that, using the factory-set lookup tables, the internal combustion engine's ECU regulates the fuel rate in the cylinders under closed loop control.

As the IC and the EM operate in parallel configuration, their combined maximum output is rated at 351 kW with an operating speed range of 600 to 2200 rpm. The diesel engine is equipped with state of the art emission reduction technologies, an Exhaust Gas Recirculation (EGR) system, controlled by the ECU as well as a Selective Catalytic Reactor (SCR) NO_X abatement systems. The engine is also equipped, with an integrated after treatment unit which includes a Diesel Particulate Filter (DPF).

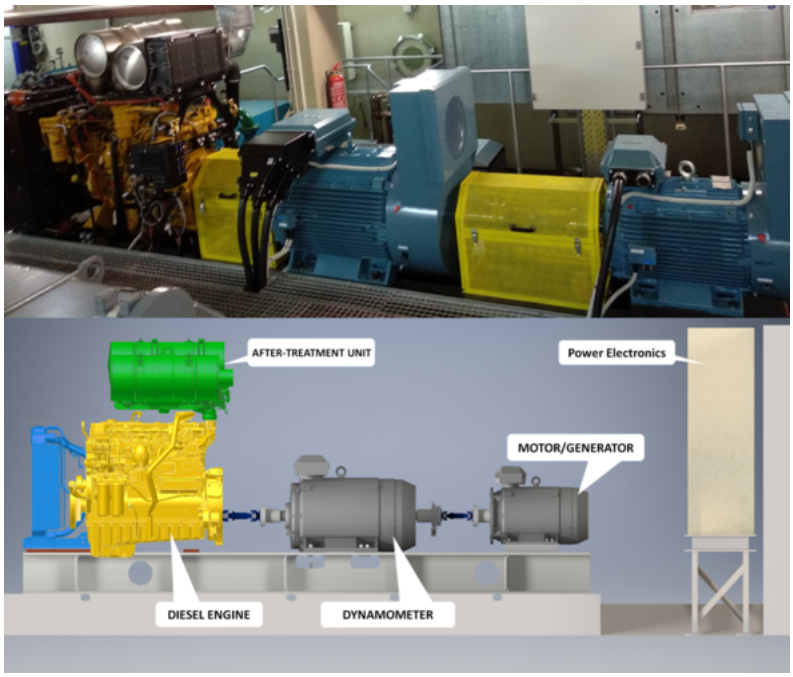

Figure 4.1: The experimental test-best **HIPPO-2** of the Laboratory of Marine Engineering (LME) of the National Technical University of Athens

4.2 Data Collection

The control and data acquisition system of the testbed is based on the sensors that are installed by the manufacturer on the ICE and the Electric motors and CAN bus communication protocol [31]. Additionally, during modifications, two coriolis mass-flow meters and analog torque sensors were added by the Laboratory of Marine Engineering, which measure the fuel mass flow and density on the feed and return pipes. Other important sensors constitute the torque-meter between the IC and the EM, the NO_X and O_2 sensor. Additionally, multiple metrics are provided by the engine ECU itself including engine speed, exhaust gas temperature, the EGR command and so on.

A dSPACE MicroAutobox II controller board that has been programmed in the MATLAB/Simulink environment oversees and controls the entire test-bed. The MicroAutoBox II is a real-time system that can run autonomously and allows for quick function prototyping, much like an engine control unit (ECU). The system, along with the sensors and their connections, is presented in figure (4.2).

The data acquired in the current thesis were gathered by the Lab's data acquisition system, from the experienced personnel of the Laboratory. Following that, a post-processing occurred where the data were cleaned and evaluated. Additionally, an evaluation was done regarding the sensors readings, the noise, as well as significant outliers.

The experimental facility, is capable of simulating multiple load-scenarios due to the electronic brake. Examples include a steady state load application, accelerations with variable speeds and torque as well as simulating a *propeller load*. Another interesting example is the step-load and the effects it has on the other engine parameters. Significant effort was made by the Laboratory, so that the data covers the entire operating region/ *envelope* of the engine regarding torque and engine speed, as well as different load scenarios.

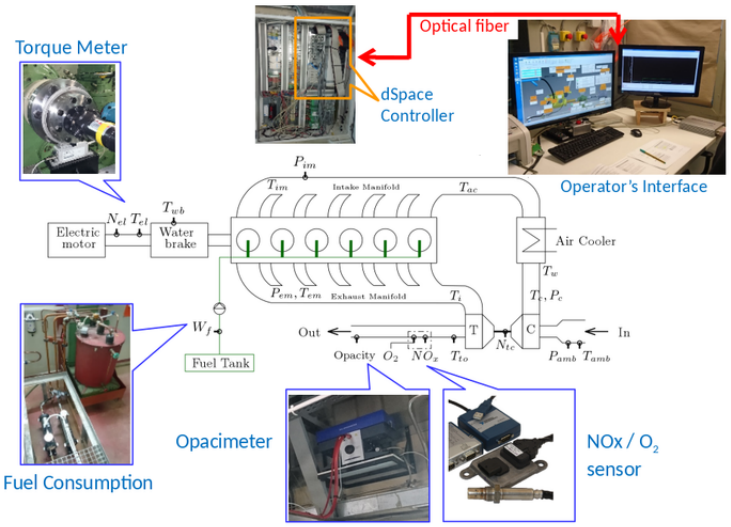

Figure 4.2: Sensors, rapid prototyping and operator interface of LME/NTUA experimental facilities

4.3 Software Embedding

In the context of the current thesis, online predicting capabilities of the models were also explored. For this reason a Raspberry Pi coupled with a CAN Bus are utilized for the communication and signal receiving from the ECU of the ICE as seen in figure (4.3). More precisely, a Raspberry Pi 4 is used which is a single board small modular computer, with an with a 1.5 GHz 64-bit quad core ARM Cortex-A72 processor and 1GB of RAM.

The board's ability to run embedded Linux in particular makes the resulting platform accessible, adaptable, and powerful for engineers that seek rapid prototyping. Together, Linux and embedded systems facilitate device development for future challenges in the Internet of Things (IoT), robotics, cyber-physical systems, 3D printing, advanced vehicular systems, and a plethora of other applications.

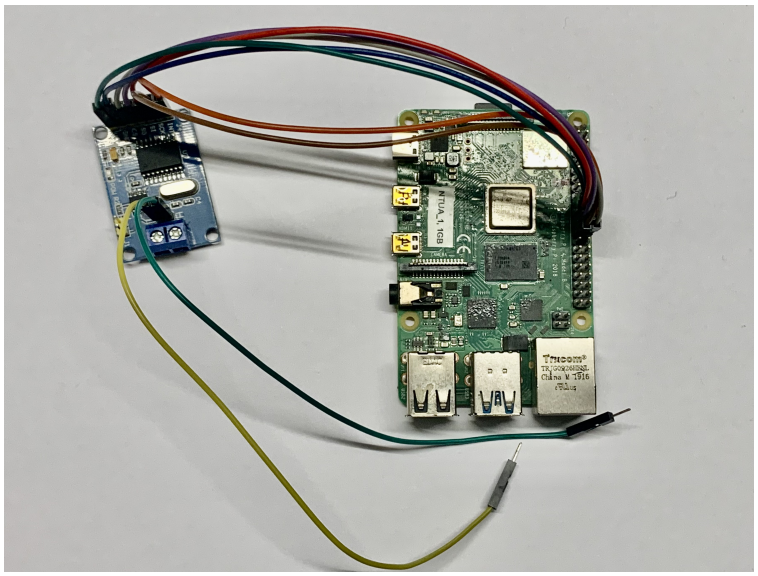

Figure 4.3: The CAN Bus Module (left) and the Raspberry Pi (right). The two free wires, the 'high' and the 'low', are connected to the engine's ECU.

The Controller Area Network (CAN) was developed by Robert Bosch GmbH for automotive applications in the early 1980s and publicly released in 1986. Typically, CAN interconnects a network of modules (or nodes) using two wire, twisted pair cable. CAN is a serial, multimaster, multicast protocol, which means that when the bus is free, any node can send a message (multimaster), and all nodes may receive and act on the message (multicast). The node that initiates the message is called the transmitter, while any node not sending a message is called a receiver.

Messages are assigned static priorities, and a transmitting node will remain a transmitter until the bus becomes idle or until it is superseded by a node with a higher priority message through a process called arbitration. A CAN message may contain up to 8 bytes of data. A message identifier describes the data content and is used by receiving nodes to determine the destination on the network. The CAN bus is embedded in the engine and sends and is able to receive messages.

The Raspberry PI does not have native CAN support. Nevertheless, the Linux kernel supports CAN drivers and more precisely SocketCAN implemented in Python programming language. In order to control the CAN of the engine, a CAN Bus Module based on the CAN bus controller MCP2515 and CAN transceiver TJA1050 is used, provided in the same module. The control of the CAN Bus device is performed by SPI interface.

Using a Julia Language script, provided by the Laboratory the models constructed in the current thesis can easily be embedded in the engine operation. The process goes as follows. Initially, the signals received by the engine are decoded, with emphasis on the signals chosen. The signal recognition is performed, through the message arbitration id of the signal which is given by SAE J1939 Standards Collection. Following that, the normalization of the signals values, is done and consequently they are transformed to a suitable format for the models.

Lastly, the predicted values are computed and stored for evaluation. The above process, runs recursively every 0.01 second which is the rate chosen for the current application. Additionally, the bitrate is set to 250,000 or 25Kb/s suited for the CAN Bus of the HIPPO-2 engine. It is also noted, that signals can be send back and consequently engine control can also be done through this way.

4.4 Data Overview

In this section a presentation of the data used in the current thesis takes place. The data gathered from HIPPO-2 are raw data measurements which as has been discussed, have been accordingly modified. The data-set is comprised of six multiple sub-data-sets of engine cycles experiments with various load characteristics. These cycles do not correspond to some standard procedure, while the data is sampled at a *100Hz* rate. From figure (4.11) it can be concluded that the data are really noisy with plenty of spikes that resemble a realistic loading scenario. The speed range is from 850 rpm to 1700 rpm. The operating windows last maximum for fifteen minutes, while the EGR% command at some cases remains closed while on others it opens after some point. The signals, with the mean, minimum and maximum values are presented in table (4.1)

Variable	Mean	Minimum	Median	Maximum
Time [s]	451.215	0.0	445.18	987.248
NOx [ppm]	372.784	12.6	406.8	809.7
Fuel Consumption [kg/h]	25.8975	0.0	24.8	54.35
lambda [-]	3.07255	1.22717	1.57979	72.3762
Exhaust Gas Mass Flow [kg/h]	595.932	253.2	527.6	1077.0
Intake Pressure [kPa]	69.3676	4	58.0	164
Torque Reference [%]	48.2821	0	50.0	86
Rot. Speed [rpm]	1312.88	843.0	1313.75	1712.25
Engine Torque [Nm]	615.747	-175.291	657.865	1195.46
EGR Command [%]	21.3062	0.0	25.9825	56.09
Exhaust Gas Temperature [C]	345.231	143.5	370.125	438.312

Table 4.1: Data Overview.

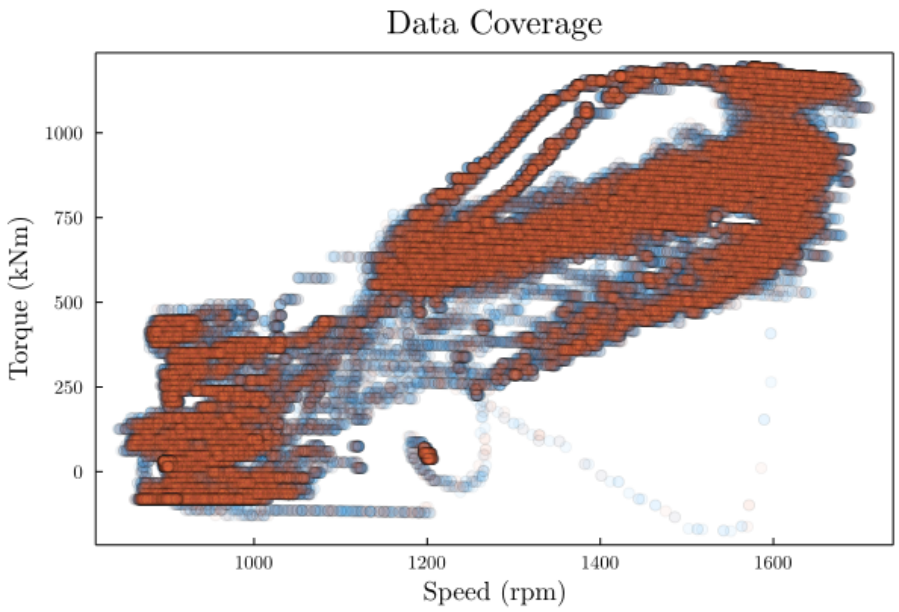

Figure 4.4: The data points of Speed (rpm) vs Torque (kNm) of the engine collected. Blue denotes the *test* data and red the *train* data.

The goal is to predict the following values:

1. **NOx** content (ppm) .
2. **Air Fuel Equivalence Ratio** (λ) value (-).
3. **Fuel Consumption** (kg/h)
4. **Manifold Pressure** (kPa)

The prediction of those quantities is based on data given from other quantities. It is important to analyze their relationship. In the following figures an example *Data-set* is presented where the values of the signals as a function of time are presented. Intuitively, it is obvious that some patterns in those signals are evident. For example, we observe that peaks in the Torque Reference (4.22) coincide with peaks of the Engine Speed (4.31), and therefore someone could argue that using both signals would not contribute qualitatively

in strengthening the information incorporated in a model. In other words, feature selection needs to take into account the linkages of the signals in order to accelerate and optimize the training. Additionally, the histograms of every feature are displayed in order to have an understanding of the dispersion of the data. On the x axis, the values of the parameters are ploted while on the y axis the frequency, meaning how often the same value is present in the data-set. Also, the empirical distribution function is ploted, which at every given point, is the fraction of observations of the measured variable that are less than or equal to the specified value.

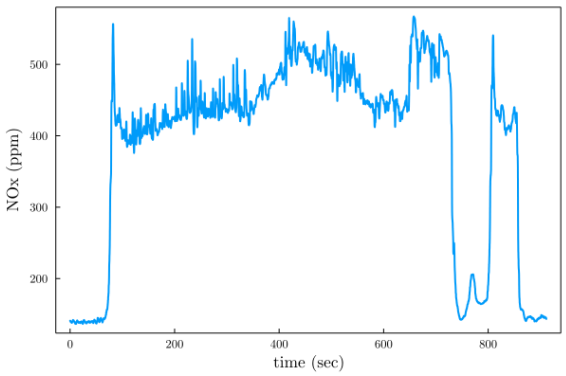

Figure 4.5: NO_x

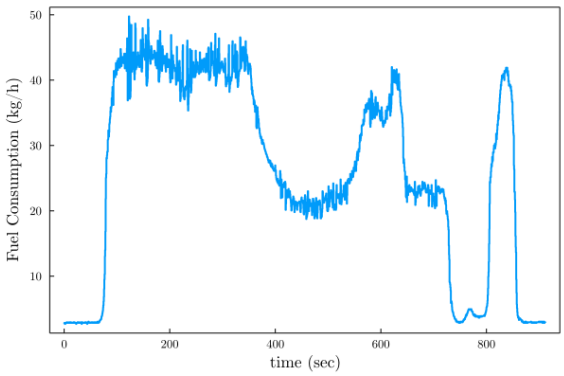

Figure 4.6: Fuel Consumption

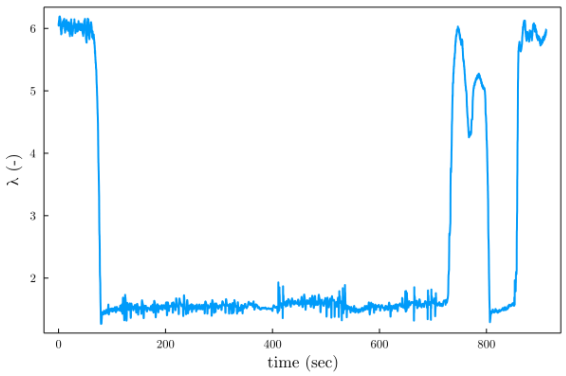

Figure 4.7: λ

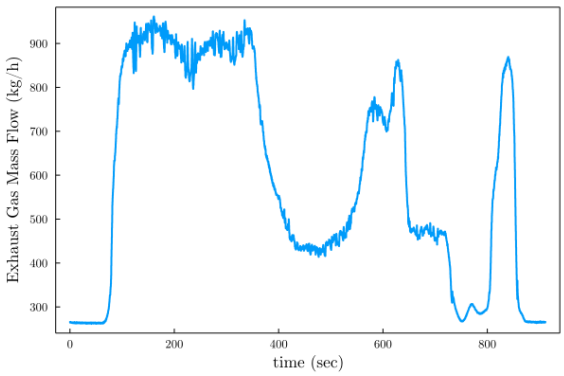

Figure 4.8: Exhaust Gas Mass Flow

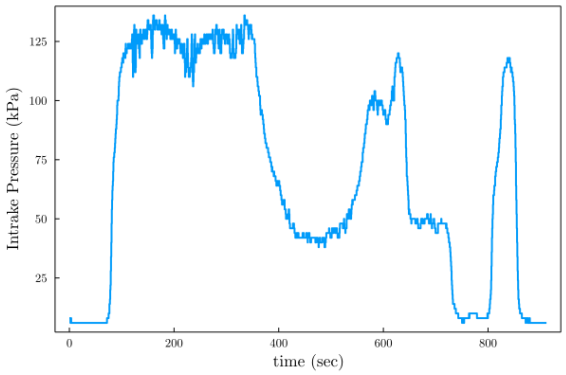

Figure 4.9: Manifold Pressure

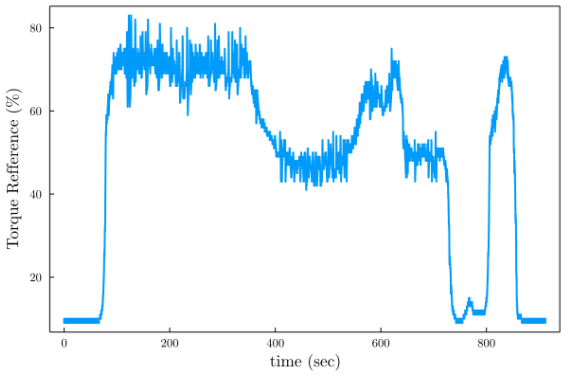

Figure 4.10: Torque Reference

Figure 4.11: A Data-set example

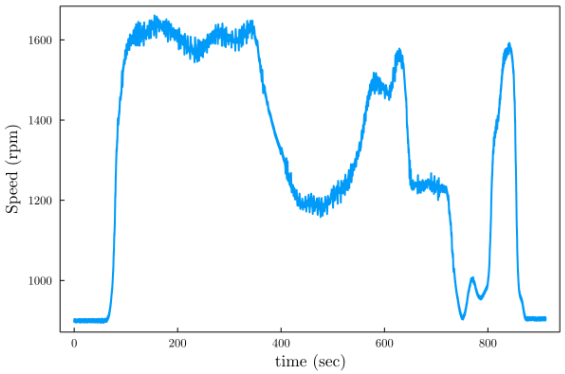

Figure 4.12: Rotating Speed

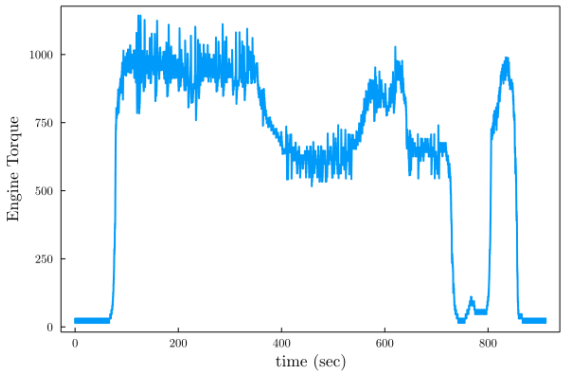

Figure 4.13: Engine Torque

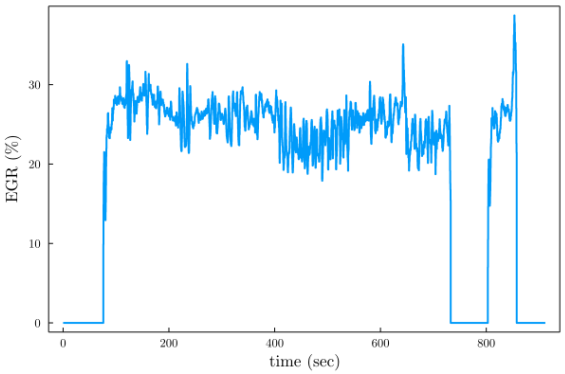

Figure 4.14: EGR%

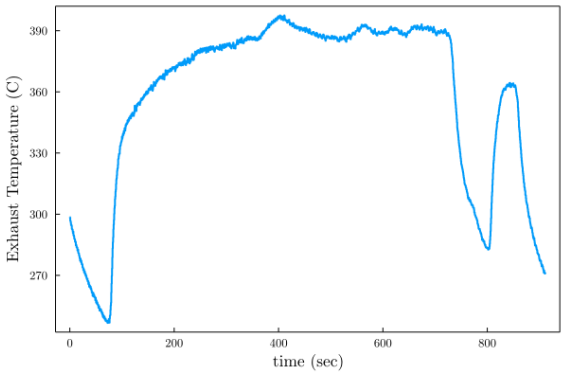

Figure 4.15: Exhaust Gas Temperature

Figure 4.16: A Data-set example continued

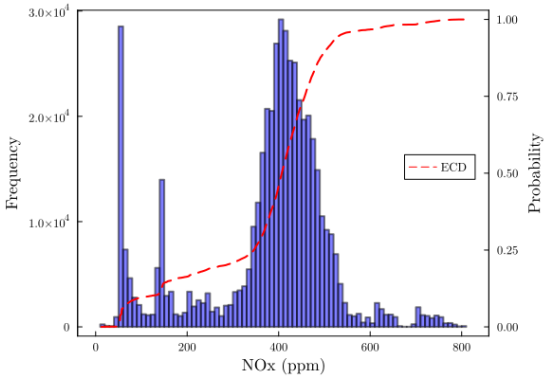

Figure 4.17: NO_x

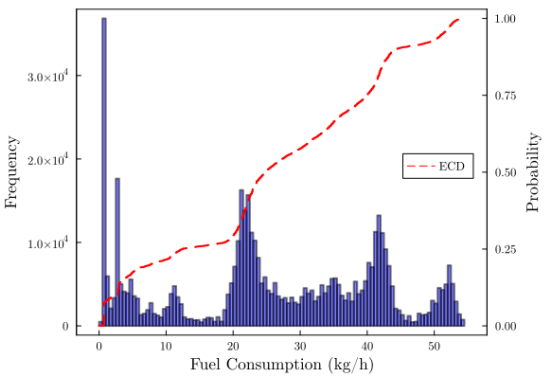

Figure 4.18: Fuel Consumption

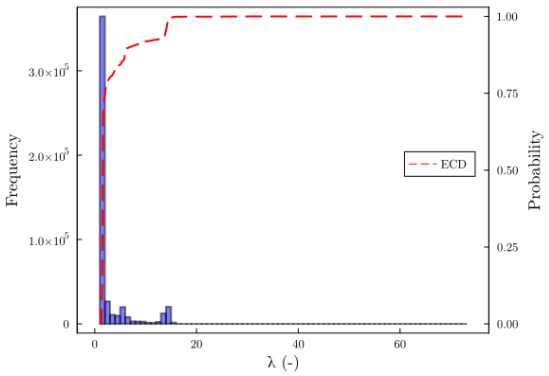

Figure 4.19: λ

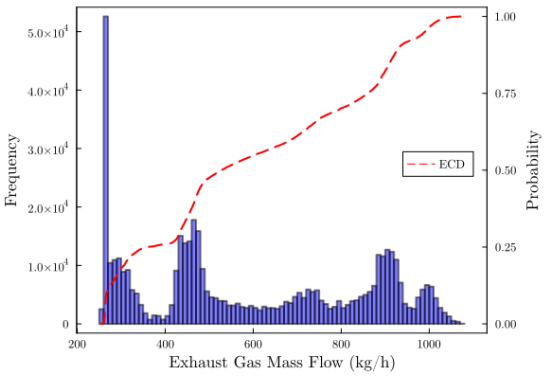

Figure 4.20: Exhaust Gas Mass Flow

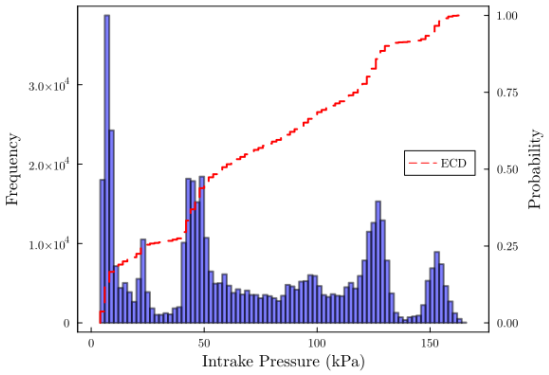

Figure 4.21: Manifold Pressure

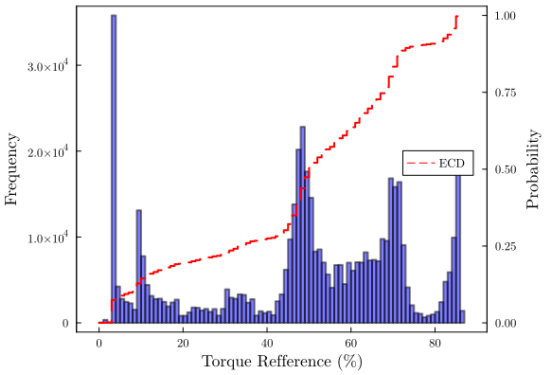

Figure 4.22: Torque Reference

Figure 4.23: Histograms of the data.

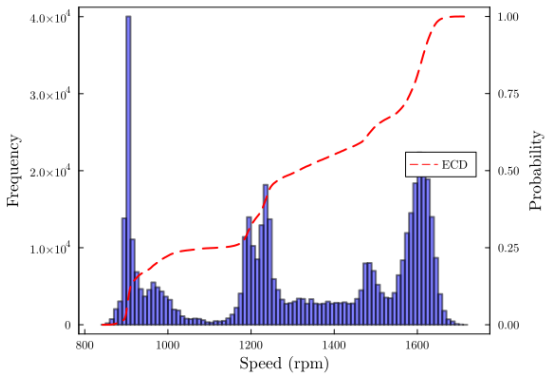

Figure 4.24: Rotating Speed)

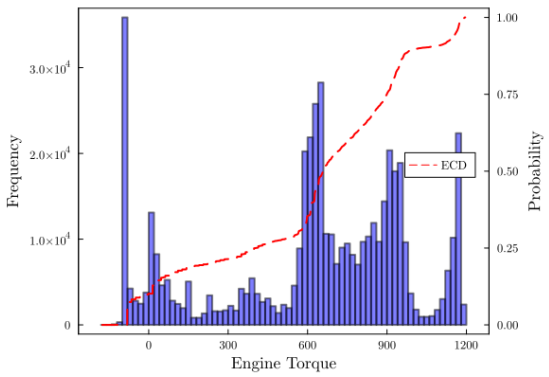

Figure 4.25: Engine Torque

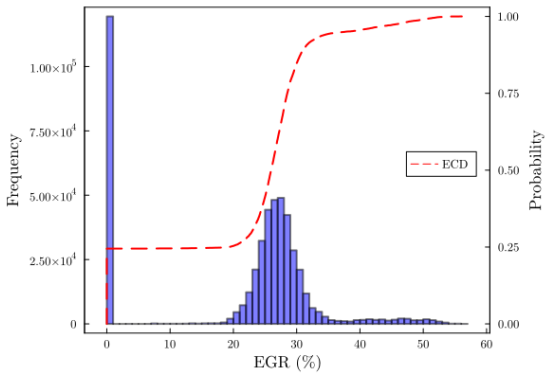

Figure 4.26: EGR%

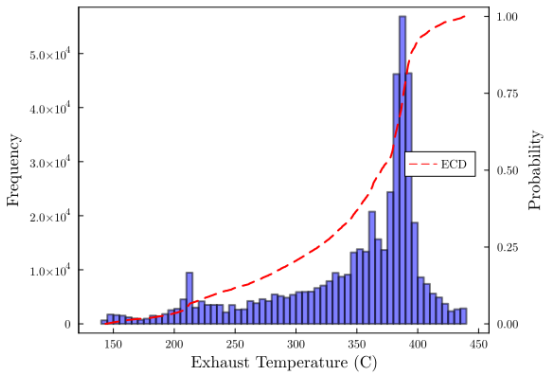

Figure 4.27: Exhaust Gas Temperature

Figure 4.28: Histograms of the data continued.

4.5 Data Re-Sampling

Additional modifications need to take place in order to optimize the process of training. A first step, is the data re-sampling, meaning altering manually the data frequency. The whole data set contains 487,953 data points for each feature measured. The samples are given every 0.01 seconds, a frequency considered very high for such signals as the oscillations and meaningful value changes, occur in longer periods. This means that the physics of the problem do not change rapidly between those time-spans and consequently there is no point in keeping this excess of information. Additionally, from a control perspective, again such high rates are not utilized in practice as they are not realistic for real applications of Internal Combustion Engines Operation. Another advantage of down-sampling, is the reduction in training time allowing for faster model-design. The raw data and the re-sampled are plotted in figures (4.29) and (4.30) where one point was kept for every ten samples.

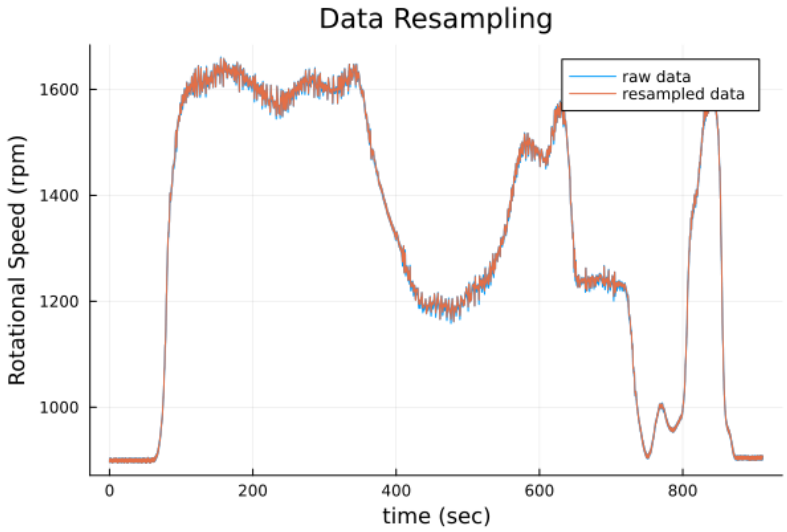

Figure 4.29: Raw and Re-sampled data.

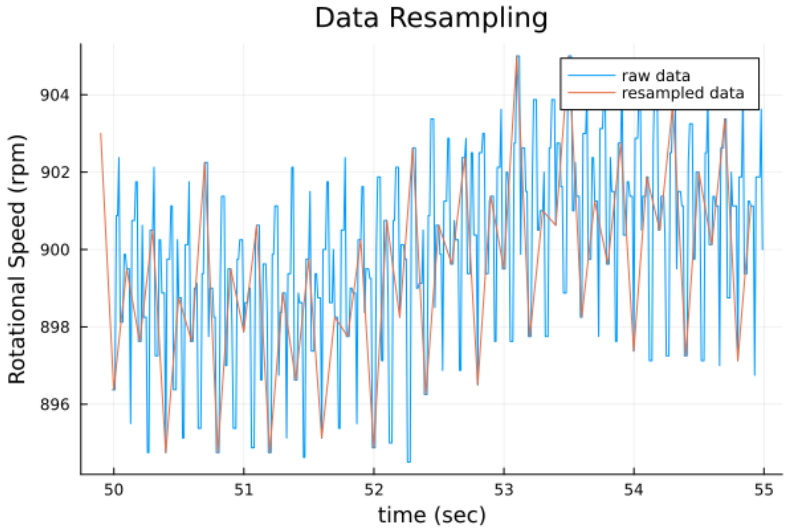

Figure 4.30: Zoom into Raw and Re-sampled data from figure (4.29).

4.6 Feature Scaling/ Normalization

As it is evident in table (4.1) the different signals are characterized by different scales. For example the maximum value of Rotating Speed is 1712 while the average value of λ is 3.07. This contrast, is able to significantly slow down the training process, if it ever converges. For this reason, as demonstrated in multiple papers (e.g. [32]), it is a common practice in machine learning to scale the data with different methods depending on the application as well as the descriptive statistics of the variable used. A simple approach is the Min-Max scaling. Considering a distributed parameter (x_n) and the corresponding scaled values $(x_n^{'})$ we have,

$$x_n^{'} = \frac{x_n}{Max(x_n) - Min(x_n)} \tag{4.1}$$

It is also common to scale the parameters, between two numbers, usually 0 and 1. This particular choices is not advised for the current application as it may scale with respect to outliers, or wrong sensor readings, and as a result mislead the training process.

$$x_n^{'} = \frac{x_n - Min(x_n)}{Max(x_n) - Min(x_n)} \tag{4.2}$$

The method used in the current thesis is that of *Z-Score transform*,

$$x_n^{'} = \frac{x - \mu}{\sigma} \tag{4.3}$$

where μ is the mean of x_n, and σ the standard deviation. As a result the distribution of the parameters changes as seen in figure (4.33), where the value of speed changes and from values between 700-1700 rpm it is bounded between -2 and 1.5. In this way, the data is scaled accurately around the zero (mean) with a unit-variance.

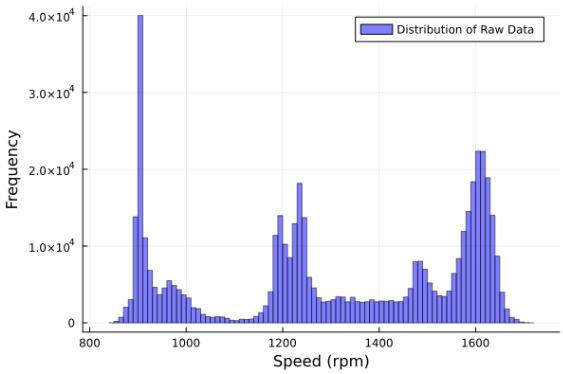

Figure 4.31: Raw Speed Data

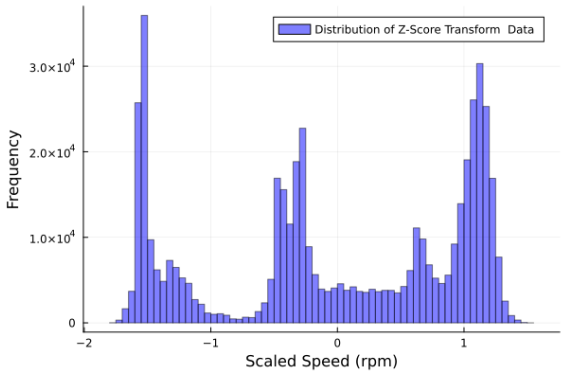

Figure 4.32: Transformed Speed

Figure 4.33: Histograms of Raw data versus Transformed. The shape stays the same, the values change

4.7 Data Transformation

After performing the pre-processing of sections (4.5) and (4.6), the samples need to be reshaped in order to be compatible with the models constructed. The imported data are in the form of *Data Frames*, where the columns correspond to the signals and the rows to the the values of these at every time-step. The first row corresponds to the timestamp of every signal. The models constructed in the current thesis, are sequence based meaning that each input is a sequence of a certain number of consecutive timestamps. Initially a *feature selection* occurs, meaning selecting the signals that are used in the model to predict the target quantity and neglecting the rest rows.

Additionally, the neural-networks models constructed in the current thesis are designed as *observers*, meaning estimating the value of a parameter at time step t_n when given an input sequence at time steps $(t_0, ..., t_n)$. Two types of architectures are investigated regarding their approximation qualities. The first type of architecture, is a single-channel architecture. If we consider an input sequence $x(t_0 : t_n)$ of n consecutive time steps, with κ signals incorporated in vector x, and a corresponding target sequence of $y(t_n)$, then the input matrix of the model is a matrix of size $n \times \kappa$.

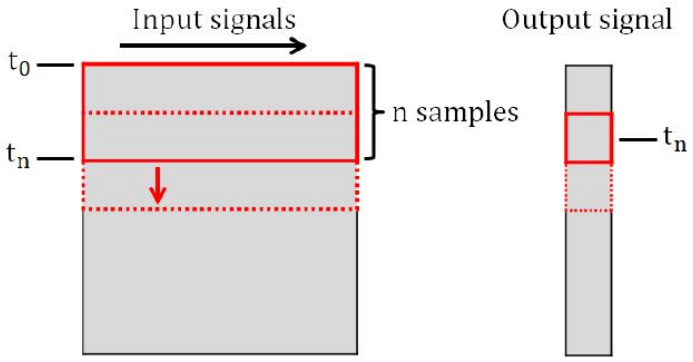

Figure 4.34: Demonstration of the size of the input matrix as well as the size of the target matrix. For example, utilizing five features and ten consecutive time steps, the size of the input matrix will be 5x10, while the output-target matrix always remains a 1x1.

Of-course the training data-set does not include a single input matrix but rather multiple matrixes with corresponding multiple target values. As a result, a training set of K samples, gives the following batch of inputs:

$$X = [x(t_0 : t_n), x(t_1 : t_{n+1}), ..., x(t_{K-n} : t_K)] \tag{4.4}$$

and a target output batch:

$$Y = [y(t_n), y(t_{n+1}), ..., y(t_K)] \tag{4.5}$$

The neural network forward propagates each input x and measures the error relative to the target output y. The values before time step n are all discarded as is is not possible to construct a meaningful input matrix.

The alternative architecture of a deep convolutional neural network under investigation is the multi-channel one. In this approach every signal is feeded into one channel. Every channel has as input a vector with a size equal to the timesteps utilized for the prediction.

The target sequence remains the same as a 1×1 matrix. In other words for K timesteps, M features, and generally N batches the input of the neural network for training is a matrix of the form $1 \times K \times M \times N$.

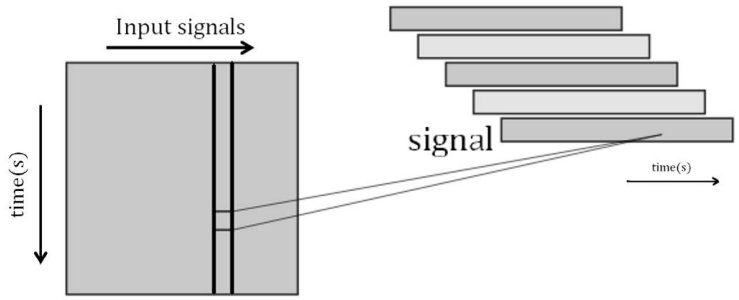

Figure 4.35: Each signal is feeded to one channel. In the same example as in figure (4.34) for 5 signals and 10 time steps, five channels are made each one with an input matrix of 1x10. So if for example, the total number of these matrixes is 1000, then the input matrix of the neural network is a matrix os size $1 \times 10 \times 5 \times 1000$.

4.8 Data Split

The next step in the pre-processing procedure is the data split. The usual approach is to split the data-set into a training sub set and a test sub set. The reason being, that the the goal of the regression process is not, to be able to predict accurately this specific data-set but, to be trained by those samples and through feature extraction to be able to generalize the model's accuracy in unseen data. In other words, the goal is to accomplish the maximum predicting accuracy not on the training data set, but rather on new unseen samples. Moreover, the concept of *transfer learning* requires that the model is taught the physics and not the specific system statistics. The beforementioned arise from a usual problem in supervised learning called *overfitting*. Overfitting, resembles the risk of intensive training on the training data set that results in fitting the *noise* in the data by memorizing the various peculiarities rather than finding a general predictive rule [33].

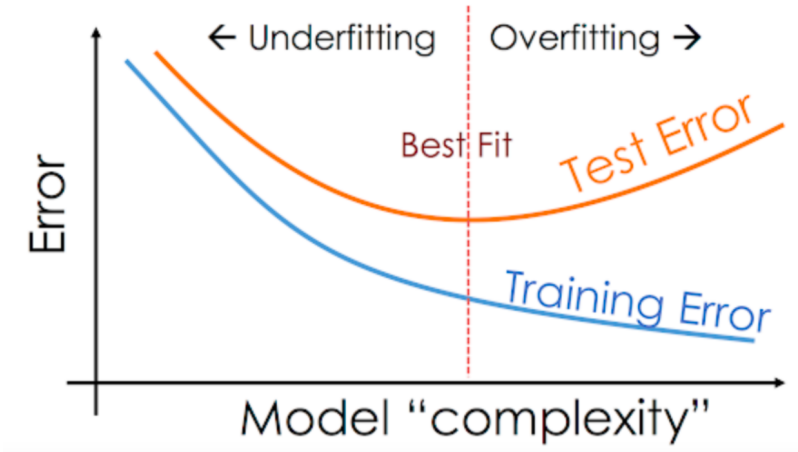

Figure 4.36: Demonstration of Overfitting. Nevertheless, underfitting should also be avoided. The engineer should locate the "sweet spot" between over-engineering and under-engineering. This can be achieved by monitoring the convergence-divergence of the test error and the train error.

The usual practice, is to split the data set into 70% training data and 30% test data, if enough data are available for training as seen in figure (4.4). The data-splited, in our case the input sequences, are randomly chosen and shuffled before provided to the network. Lastly, a certain data-set is kept for validation purposes as well as fine-tuning the architectures of the networks (4.37).

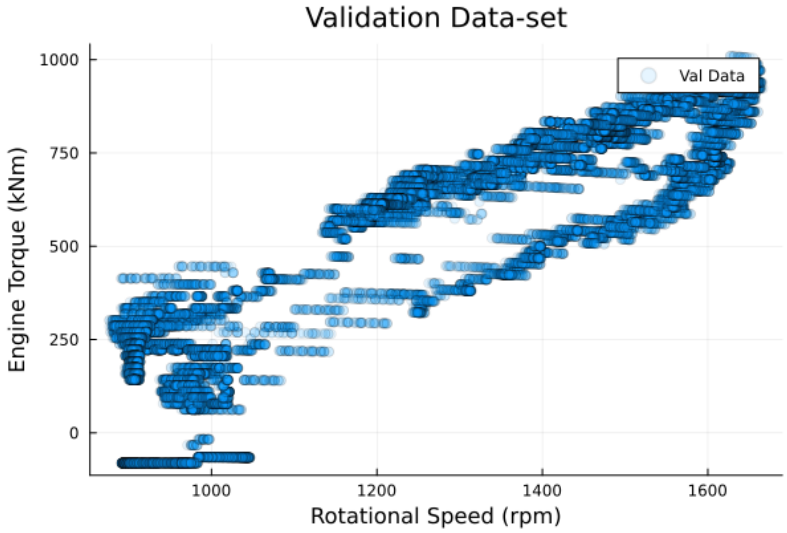

Figure 4.37: The data points of the Validation Data set in terms of engine operational regions.

Chapter 5

Networks Structure

In this chapter, the model structure is discussed as well as the process of training. Moreover, the inputs of the networks are chosen based on *Exploratory Data Analysis*. In neural networks in general, the search for a superior performing model design is an ever going one. In other words, the problem is not closed problem, but rather an open one where researchers are constantly looking for new architectures with new features and types of layers. The process of selecting a single architecture, is usually done by a try and error approach or more generally with *heuristic* techniques, such as grid search ans so on.

5.1 Architectures of Models

Regarding convolutional neural networks, a variety of different architectures have been tested with the passage of time. Usually the bench-marking for such networks takes place in image classification contests-challenges, like for example *ImageNet* contest. Therefore, the selection of an architecture is not as straightforward as a plain feed forward neural network, or a Long-Short-Term-Memory (LSTM) network, where the options are limited. In the current thesis, it was decided to construct two types of architectures.

The first architecture, as mentioned in section (4.7), treats the input data for the prediction as a type of grey scale 2-dimensional image with every pixel being a value of a certain signal at a certain time stamp. The intensity of white or black corresponds to the value of the parameter. This architecture is inspired by traditional Convolutional Neural Networks whose task is usually to classify pictures and label them.

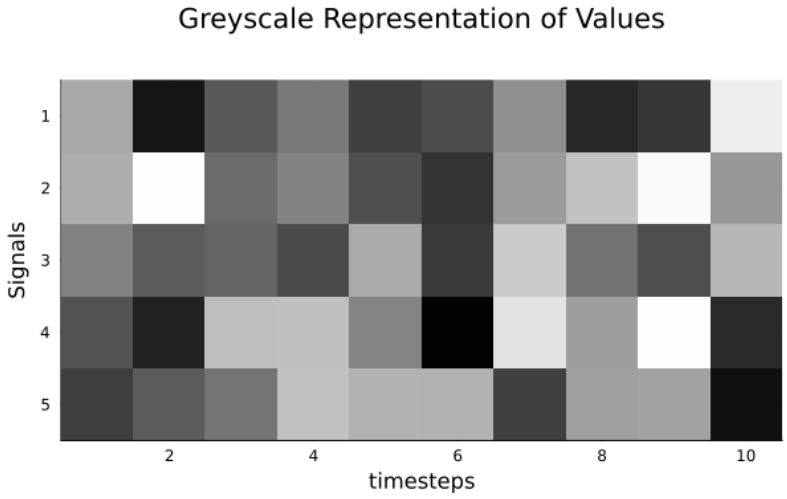

Figure 5.1: A representation of a matrix of size 5x10, meaning 5 signals and 10 time-steps as a grey-scale image.

Multiple really deep and sophisticated CNN's have been developed for classification purposes. These include popular ones like the *LeNet*, in figure (5.2), *AlexNet*, *ResNet*, *GoogLeNet*, *DenseNet* and more [34], [35]. The basic building block of these architectures is of course the *Convolutional* layer, hence the name. The difference lies on the way the convolution is utilized and the inter-connecting layers between the convolution blocks. The most common layers in use, are:

1. *Dense Connection or Fully Connected*: are a type of layer in a deep neural network that use a linear operation where every input is connected to every output by a weight.

2. *Pooling Layers or Sub-Sampling*: provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map. Average pooling and max pooling.

3. *Batching Layers*: Inputs are linearly transformed to have zero mean and unit variance.

4. *Flatten Layer*: Dimensional reduction of the input to a 1-dimensional vector in order to be more convenient for processing in a loss function or feeding it into a Dense layer.

5. *Residual and Skip-Connections*: Skip connection as the name suggests skips some layers. Residual connection is a type of skip connection that learns residual functions with reference to the input layer. This type of connections are used to train really deep CNN's.

6. *Concatenate Channels*: Fusing the information from different channels of the network.

In the first selected architectures, the skip connections are not utilized as the networks are not so big. At the same time, BatchNorm layers are used after the convolution as it was found that they adapt really good in training, in contrast to MeanPool layers which resulted in networks overfitting the presented data and as a result were only used once at the end of the networks before the Dense layers. The Dense layers at the last part, act as predictive layers, after the Convolutional part where features of the signals were extracted. In general, the architecture of the networks, is similar to those constructed in the past for classification, with the difference attributed in the use of the type of activation function. It is also found, that increasing the channels from one in the input, further increases the accuracy of the network without a need to make it deeper.

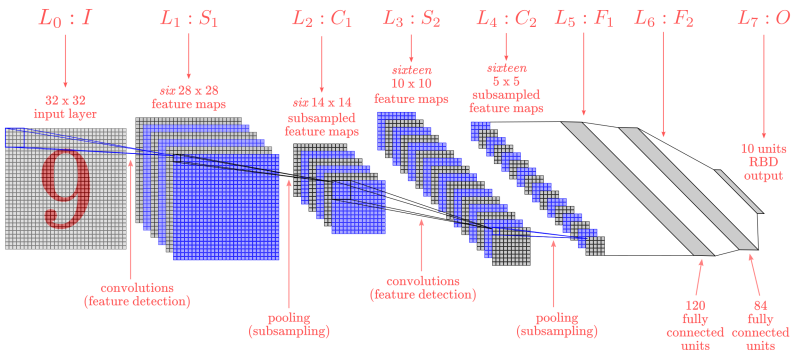

Figure 5.2: A demonstration of the architecture of LeNet. Aside from the convolutin blocks, there are pooling blocks. Additionally, we observe, that as the network gets deeper, the feature maps increase as in the beginning basic features are being extracted. As the data propagates, more global features are being extracted.

The second architecture investigated in the current thesis is inspired by signal processing and more specifically audio processing. Every signal is feeded into a different channel and the convolution takes place on itself, meaning the kernel is one dimensional and so is the convolution [36]. This means, that a kind of autocorellation is being performed, keeping the bigger and most important frequency, and ditching the smaller ones. This technique

smoothens the signal and in theory should smoothen the predictions of the network and eliminate the spikes seen in the signals collected. Again, the usual architectures are constucted for classification purposes but with the correct modifications they can be tuned to cope with regression problems.

In this architecture, multiple channels are used as multiple signals are used as inputs. The output remains the same, a target value. As the architecture gets deeper in this networks and the parameters more, the skip-connection is utilized as well as channel concatenation for information fusion, inspired by the DenseNet architecture. Again, BatchNorm is used to normalize the channels, while the convolutions have one and two cells kernels. A flattening of the layers is done at the end. Lastly, a Dense architecture is added at the end, after the feature extraction for prediction properties.

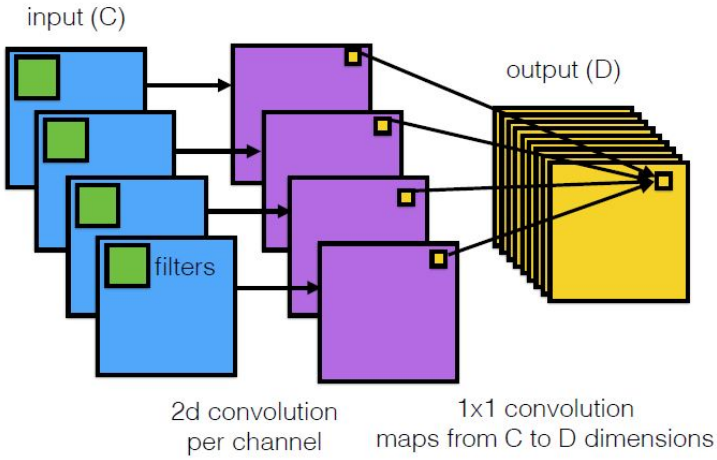

Figure 5.3: Demonstration of the inspiration for the second architecture. Each of the C input channels undergoes a 2d convolution to produce C output channels, which get combined to produce D output channels

5.2 Optimizer Selection

The training of the neural network, as discussed in section (2.4.2), is performed with the aid of gradient information. The optimization, is a slow process, as the it is an non-convex problem, if performed only by stochastic gradient descend as the convergence is slow due to slow-down observed in regions called local optima, where the surface is much more steeper in one direction than on others, as seen in Figure (5.4). To improve the empirical performance of SGD, more advanced optimizes have been developed utilizing the momentum, called *Adaptive Optimization Algorithms*. These algorithms, reduce the oscillations that occur in gradient descent, as the algorithm takes big steps in one direction, the steepest descend, and then other steps to correct, as the next step is in another direction.

Regarding the learning rate, for training to succeed contradicting problems arise. In order to reduce the loss quickly and not be trapped in a "bad" minimum a large learning rate is required. On the other hand, in order not to bounce in narrow valleys or oscillate around a minimum a small learning rate is required. These constraints lead to a general policy of using a larger step size initially, and a smaller one as the process advances.The momentum gradient descent update rule, for the whole batch, is given by the following equations.

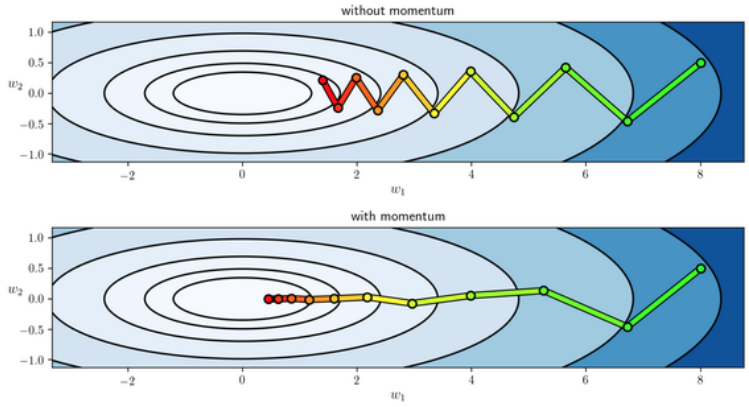

Figure 5.4: Demonstration of converging steps in the case of Stochastic Gradient Descend with and without momentum.

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw \tag{5.1}$$

$$W = W - \alpha \cdot v_{dw} \tag{5.2}$$

$$v_{db} = \beta \cdot v_{db} + (1 - \beta) \cdot db \tag{5.3}$$

$$b = b - \alpha \cdot v_{db} \tag{5.4}$$

The value of β denotes the momentum, while α or η represent the learning rate and W the weight. A small parameter ϵ is included for gradients normalization.

5.2.1 RMSprop

The RMSprop, from "root mean squared", proposed by Geoffrey Hinton, uses a moving average of squared gradients to normalize the gradient. As a result the step size is normalized (momentum), increasing the step for small gradients to avoid the problem of *Vanishing Gradient*, and decreasing the step for bigger gradients to avoid the problem of *Exploding Gradient*. The learning rate, changes as the training proceeds, rather than being a constant hyperparameter The update rule is given as:

$$s_{dw} = \beta \cdot s_{dw} + (1 - \beta) \cdot dw^2 \tag{5.5}$$

$$W = W - \eta \cdot \frac{dw}{\sqrt{s_{dw}} + \epsilon} \tag{5.6}$$

$$s_{db} = \beta \cdot s_{db} + (1 - \beta) \cdot db^2 \tag{5.7}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{s_{db}} + \epsilon} \tag{5.8}$$

5.2.2 ADAM (Adaptive Moment Estimation)

The Adam optimizer, presented in [37], combines the ideas of Stochastic Gradient Descent with momentum and RMSprop algorithms. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum based on adaptive estimates of lower-order moments. It is one of the state-of the art algorithms widely used in machine learning today and is given by:

$$\upsilon_{dw} = \beta_1 \cdot \upsilon_{dw} + (1 - \beta_1) \cdot dw \tag{5.9}$$

$$\hat{\upsilon_{dw}} = \frac{\upsilon_{dw}}{1 - \beta_1^t} \tag{5.10}$$

$$\upsilon_{db} = \beta_1 \cdot \upsilon_{db} + (1 - \beta_1) \cdot db \tag{5.11}$$

$$\hat{\upsilon_{db}} = \frac{\upsilon_{db}}{1 - \beta_1^t} \tag{5.12}$$

$$s_{dw} = \beta_2 \cdot \upsilon_{dw} + (1 - s_2) \cdot dw^2 \tag{5.13}$$

$$\hat{s_{dw}} = \frac{s_{dw}}{1 - \beta_2^t} \tag{5.14}$$

$$s_{db} = \beta_2 \cdot \ s_{db} + (1 - \beta_2) \cdot db^2 \tag{5.15}$$

$$\hat{s_{db}} = \frac{s_{db}}{1 - \beta_2^t} \tag{5.16}$$

$$W = W - \eta \cdot \frac{\hat{\upsilon_{dw}}}{\sqrt{\hat{s_{dw}}} + \epsilon} \tag{5.17}$$

$$b = b - \alpha \cdot \frac{\hat{\upsilon_{db}}}{\sqrt{\hat{s_{db}}} + \epsilon} \tag{5.18}$$

In the before mentioned equations, t stands for the iteration number while, typically, the scaling hyperparameter β_2 is initialized to 0.999 while the momentum decay hyperparameter β_1 is set to 0.9. Notice that, If we set $\beta_1 = 0$ and no bias-correction, we recover RMSProp, which does not use momentum. Again the learning rate adapts, and only the initial learning rate is prescribed. The Adam algorithm can be extended using the L_p norm, called AdaMax. In the current thesis the optimizer Adam is used for all the training of the networks, with initial learning rates ranging from 0.1 to 0.0001.

5.3 Activation Functions

The decision of which activation function to use is really important in structuring a network. Choosing a simple linear activation function, would make the network also a linear model. In the early days of neural networks the standard choice was the sigmoid (logistic) function, which is nonlinear. This function is a perceptron approximation of the Heaviside function with the good quality of being continuous and differentiable everywhere. This function is really good operating as a separator and that is why it is usually used in the last part of classification networks . However, it does not serve regression problems so well.

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}} \tag{5.19}$$

However from figure (5.5), it can be extracted that the sigmoid function saturates at 1 for large positive inputs and at 0 for large negative inputs. The same goes for the *tanh* function but at the points 1 and -1 respectively. This means that in saturated regimes, the gradient of the output tends to be 0, and as a result will not allow any error from higher layers to propagate back.

In the training of very deep models, the gradient tends to become either very small (this is called the vanishing gradient problem) or very large (this is called the exploding gradient problem), because the error signal is being passed through a series of layers which either amplify or diminish it. To heal this problem, other activation functions were discovered.

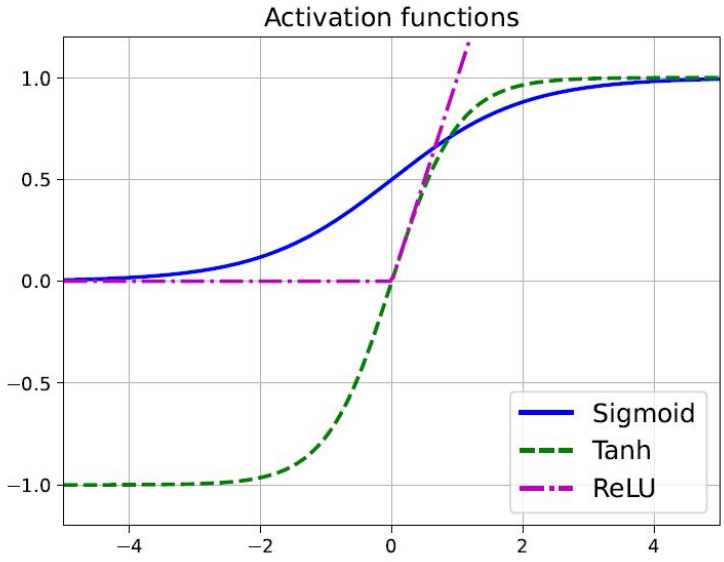

Figure 5.5: Graphical demonstration of activation functions, Sigmoid, tanh and ReLU.

ReLU

The *ReLU* (Rectified Linear Unit), proposed in [38] is a nonlinear activation function, that encapsulates properties of a linear. As the later part of the function is linear it means that it never saturates when the input gets large. ReLU, also allows the networks to easily obtain sparse representations and train faster in back-propagation. Additionally, it has a trivial representation only using a max function and a trivial derivative. It is also able to produce real zero values in contrast to other similar functions like the softmax. Nevertheless, it is still nonlinear allowing the network to learn nonlinear mappings. Its mathematical formulation is as follows,

$$ReLU(\alpha) = max(\alpha, 0) \tag{5.20}$$

Hoever, if the weights are initialized to be large and negative, then it becomes very easy for some components to take on large negative values, which are "turned off" to zero. This results in neurons staying permanently off, a problem called the dead "ReLU". To treat this, variations of ReLU have been propose that are non-saturating. The leaky ReLU, *LReLU*, is definied as:

$$LReLU(\alpha; \kappa) = max(\kappa\alpha, \alpha) \tag{5.21}$$

which allows a small proportion of the gradient to pass when the neuron is turned off. Another popular choice is the *ELU*, which is a smooth function in contrast to LReLU.

$$ELU(\alpha; \kappa) = \begin{cases} \kappa(e^{\alpha} - 1) & \text{if } \alpha \leq 0 \\ \alpha & \text{if } \alpha > 0 \end{cases} \tag{5.22}$$

5.4 Metrics

To measure how accurate the predictions of the model are, certain metrics should be introduced computing the error relative to the real values. The predictions are evaluated

after the training has finished, by feeding the model of the neural network with data and storing the output. In regression analysis evaluation and in neural networks, the mean squared error (MSE) is a common function that is not only used as a loss function but also as a metric. For a sample of n points of inputs and n measurements of the target feature and i denoting every single point, the mean squared error is given as,

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(Y_i - \hat{Y}_i)^2 \tag{5.23}$$

which calculates the mean of the square of error. $\hat{Y}_i$ denotes the prediction for every single input, while Y_i denotes the actual (target) value of the quantity. However, as this error largely penalizes some errors that might be associated with outliers or wrong sensor measurements the mean absolute error (MAP) is also used,

$$MAE = \frac{1}{n}\sum_{i=0}^{n}|Y_i - \hat{Y}_i| \tag{5.24}$$

which computes the average of the absolute of the errors. Another interesting metric is that of mean absolute percentage error (MAPE), which measures the error between the actual and the predicted values, as a percentage. Nevertheless, this metric is not so informative is the case of scaled data and is not used in the current thesis.

$$MAPE = \frac{1}{n}\sum_{i=0}^{n}|\frac{Y_i - \hat{Y}_i}{Y_i}| \tag{5.25}$$

Lastly, a measure called R-squared, or coefficient of determination, is used. It quantifies the degree of any linear corellation between the predicted and the actual values and is given by,

$$R^2 = 1 - \frac{\sum_{i=0}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=0}^{n}(Y_i - \overline{Y})^2} \tag{5.26}$$

where, $\overline{Y}$ is the mean of the observed data. The maximum value of R^2 is 1, meaning that the prediction exactly coincides with the observe value for every point. If R^2 is 0 it means, that the networks predicts the average value of the observations every time and thus, is useless. R^2, can also be viewed as the fraction of unexplained variance.

The above measures are calculated both on the training data as well as the test data, for network design as well as for monitoring the over-fitting. For example, a small training error and a big test error resemble that the model may have memorized the training data and not learned the actual mapping-features. Lastly, the same procedure is followed on the validation data, where the error is expected to be bigger but not significantly as generally the purpose is to predict unseen data and not already seen values.

5.5 Input Selection

Selecting the inputs of a predictive models is one of the most important steps in achieving the desirable accuracy as well as generalization properties. As the choice of selecting as inputs of a model the whole range of available measurements is not a viable one for various reasons such as models getting really big and un-trainable, or the fact that on board measurements cannot actually capture some measures such as NO_x's leads to the necessity

of reducing the input features. It is common practice in regression problems, to perform what is called an *Exploratory Data Analysis*, which is a method of examining data sets to highlight their key properties, frequently utilizing statistical graphics and other types of data visualization. In other words what is sought after, is a quantitative understanding of the data interactions and correlations, in order to conclude which features are actually useful to predict the target variable.

In this thesis four major techniques are utilized in order to have an accurate understanding of the data set. Firstly, a *Pearson Correlation Coefficient*, computation is done, presented in the form of a heatmap in figure (5.6) in order to asses the correlations of the data two by two. For a sample of n measurements x and y, the sample correlation coefficient is given as,

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}} \tag{5.27}$$

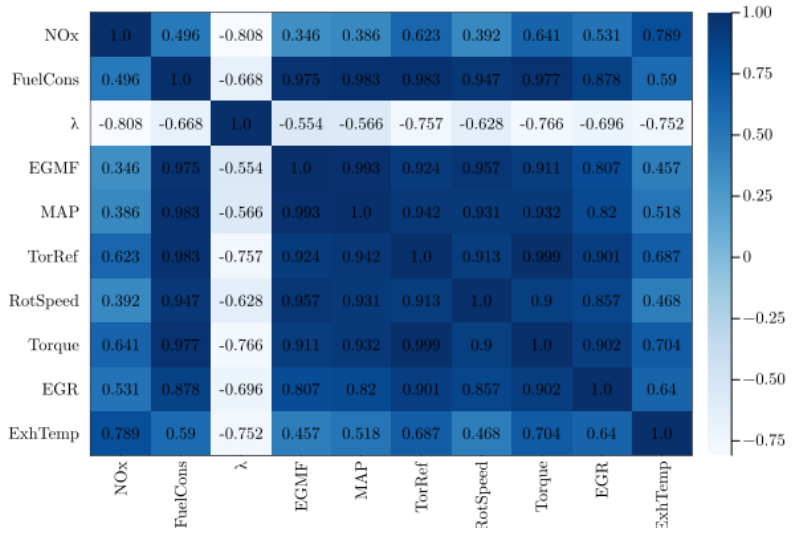

Figure 5.6: Pearson Corellation Coefficient Heatmap of the Training Dataset

From figure (5.6) we can obtain a lot of conclusions about the relationships. It is obvious, that the correlation of a metric with itself is one. We observe that the λ is negatively correlated with all of the features, something expected from the physics as for example the more the fuel consumption the less the λ. It should be stated here, that a negative correlation of one corresponds to a perfect linear relationship of the variables, and consequently it is as informative as a positive correlation of one, so it is not the sign that interests us but rather the absolute value.

Other interesting observations, are concerned with the correlations of the Nitrogen Oxides with the rest of the parameters which is only strongly correlated with λ and the Exhaust Gas Temperature, while not correlated with for example the Manifold Pressure. On the contrary, Fuel Consumption is highly correlated with most of the measures, meaning that it should be easier to predict its values based on these. Finally, Manifold Pressure is highly corellated with most of the parameters except the Nitrogen Oxides and λ.

The next two metrics utilized are the F-test score as well as the Mutual Info Regression. The F-test, initially performs a linear regression between two variables, in our case the target variable and the different feature each time, and then tests what is called a *null*

hypothesis. What it measures, is the joint statistical significance of variable(s) with the target. It should be stated that there are a lot of different types of F-Scores. In our case, we seek an intuitive understanding, so we care about the outcome of an F-test,

$$F = \frac{explained\ variance}{unexplained\ variance} \tag{5.28}$$

The higher the value of F, the higher the covariance between the two variables, and so the higher the probability of doing a good prediction using this value. *Mutual Information* is the measure of two variables is the measure of their mutual dependence. In other words, it quantifies the "amount of information", obtained about one (random) variable, in our case the target variable, by observing the other (random) variable, the feature under test every time. Again an intuitive explanation, is given as,

$$MI(feature; target) = Entropy(feature) - Entropy(feature|target) \tag{5.29}$$

where entropy, resembles a kind of joint distribution. The range of the MI score is from 0 to ∞. The greater the value, the more closely this feature matches the objective, suggesting that we should include it in the training dataset.

An additional tool that we use is the *Principal Component Analysis*, which is a wide used technique for reducing the dimensionality of a dataset that consists of multiple dimensions, features in our case. In order to achieve this, the data are linearly transformed into a new coordinate system, where (most) of the variance in the data may be expressed with fewer dimensions than the initial data. Many studies plot data in two dimensions and visually locate groups of closely linked data points using the first two main components.

A set of p unit vectors, the i-th of which is the direction of a line that best matches the data while being orthogonal to the first i-1 vectors, make up the main elements of a collection of points in a real coordinate system. A line that minimizes the average squared perpendicular distance between the points and the line is said to be the best fit. The method of computing the principal components and using them to perform a change of basis on the data is known as principal component analysis (PCA). Occasionally, the first few principal components are used while the remainder are ignored, as seen in figure (5.7).

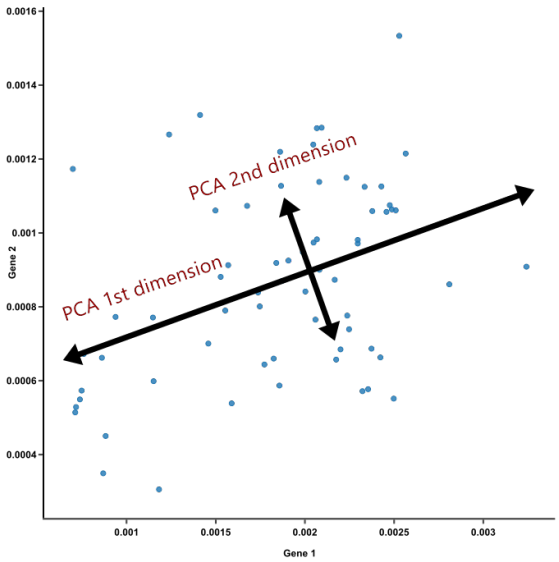

Figure 5.7: Demonstration of the first principal components/dimensions of a data-set.

In the scope of the current thesis, PCA is utilized to visualize the required dimensions/features to achieve a very good accuracy. In other words, we seek for the least amount of components that almost maximize the explained variance and set this value as a rough estimate of the required features of input for the model.

5.5.1 NOx Model Input Selection

The first visualization, consists of multiple scatter plots between the target variable, NO_x, and every feature in figure (5.8). Due to the amount of points, a heatmap is presented in order to be able to clearly identify the relationships as well as the density of measurements in each point. The point of interest, is determining whether there is a linear or other kind of relationship between the two variables. Additionally, the sparsity of data is also something to look after. From figure (5.8) it cannot be directly concluded whether there are linear relationships.

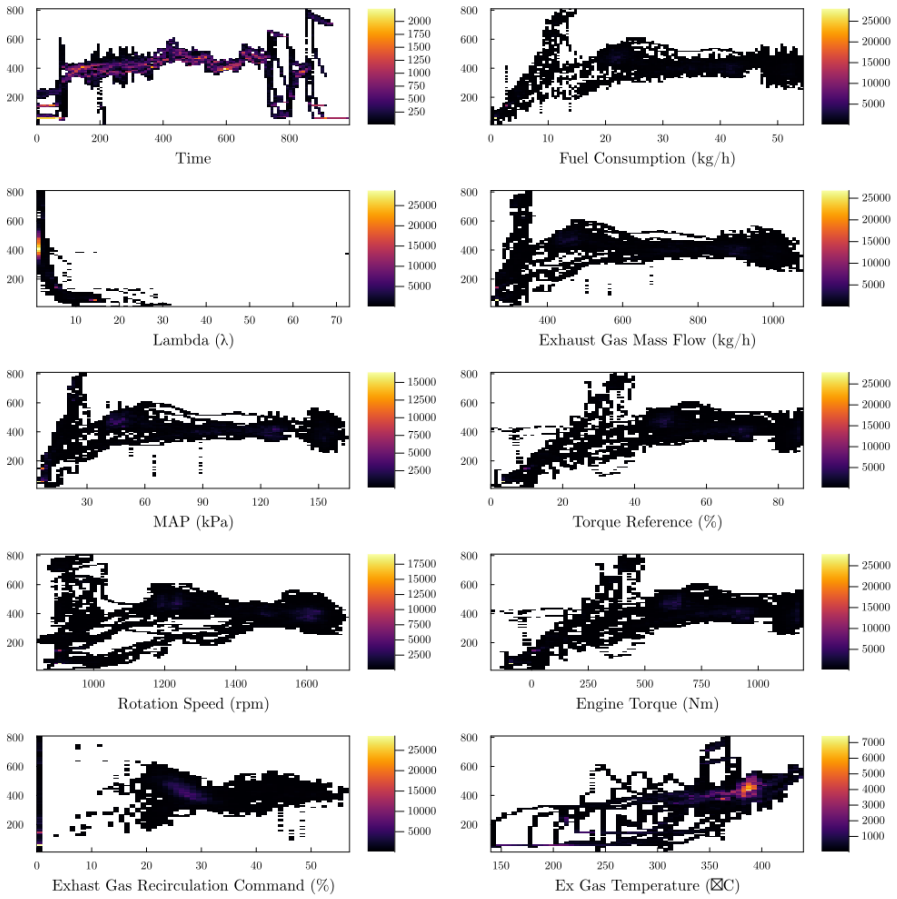

Figure 5.8: 2-dimensional Histograms of the target parameter NOx and each feature, in the form of heat-map.

The next step is assessing the outcomes of the tests performed, in figure (5.9). From the F-regression, it can be extracted that the content of Nitrogen Oxides is strongly related to λ, Exhaust Gas Temperature and Engine Torque and Torque Reference. Since Torque and Torque Reference contain the same information one is neglected.

The Mutual Info Regression diagram informs as also about the correlation with the fuel consumption. As we can see, the tests provide different results due to their different

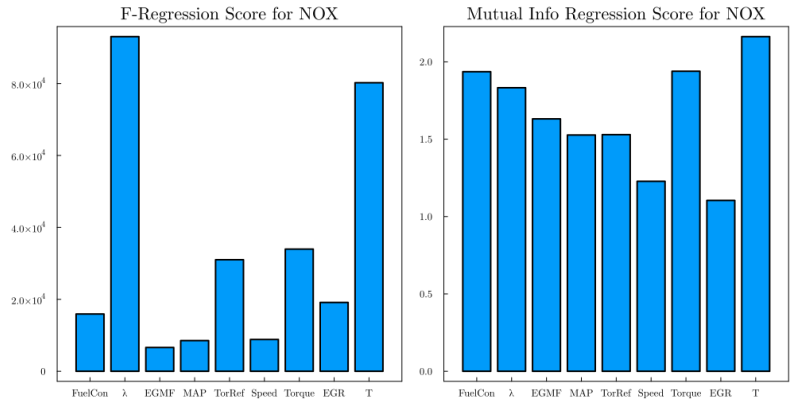

Figure 5.9: F-test score and Mutual Info Regression score for NOx and the the relevant features.

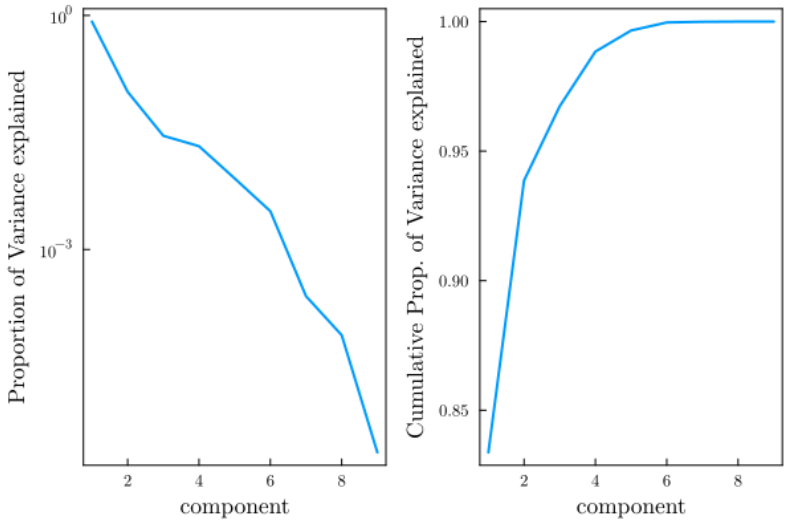

Figure 5.10: Proportion of Variance Explained and Cumulative Proportion of Variance Explained with respect to the number of components for the feature selection of NOx's

mathematical structure. Nevertheless, they provide a adequate first estimation needed to start the training process.

What is interesting, is that the statistics, do not capture all the underlying relationships. The content of NOx, is directly correlated to the EGR% command from a physics point of view, ant thus we would expect that to be present in those diagrams.

Following the tests, the PCA is performed in the feature data. On the left side the proportion of variance explained with respect to the number of components is plotet, while on the right side the cumulative sum of the proportion of variance explained with respect again to components. From figure (5.10) it can be concluded that roughly five input features are required for a model to obtain full proportion of variance explained.
After taking into consideration all the above conclusions, the inputs can be selected. Of course, also a try error approach is utilized in training trying different combinations of inputs. Additionally, the EGR% command is also included for reasons discussed before. The inputs and output are seen in figure (5.5.1).

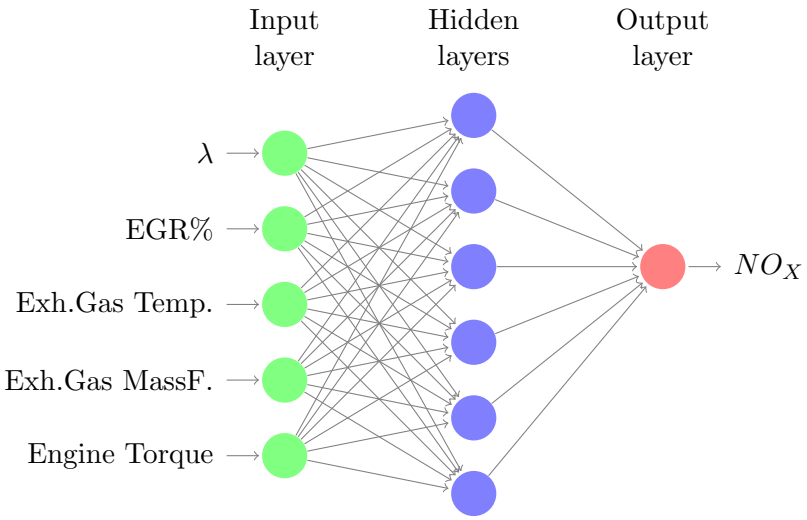

Figure 5.11: Demonstration of the inputs of the model for NO_X prediction. These are the λ, the EGR% command, the Fuel Consumption, the Exhaust Gas Mass Flow and the Exhaust Gas Temperature.

5.5.2 Lambda Input Selection

Observing the pairwise distribution of the figure (5.12) again no particular relationship outcomes can be extracted. What is of particular importance is the fact that the values of lambda in comparison with every feature seem to be very dense as lambda usually takes values in a particular interval. Exceptions consist, some spikes and higher values that may be associated with wrong sensor readings.

From the figure (5.17) and the F-regression part, it can be extracted that the value of λ is highly correlated with the value of NO_X, the Torque and the Exhaust Gas Temperature (T). The mutual info regression on the same figure, attributes a large proportion of the variance of the λ to Fuel Consumption aside from the other parameters that stay the same regarding their ranking.

Despite the above it should be stated that the NO_X parameter cannot be used for predicting the λ as firstly no such measure is present in an actual engine and secondly the value of λ is on the contrary used to predict the Nitrogen Oxides. Following the tests, the PCA is performed in the feature data. From figure (5.14) it can be concluded that roughly four input features are required for a model to obtain full proportion of variance explained.

After taking into consideration all the above conclusions, the inputs can be selected. Of course, also a try error approach is utilized in training trying different combinations of inputs. The inputs and output are seen in figure (5.5.2).

5.5.3 Fuel Consumption Model Input Selection

In figure (5.16) the pairwise distributions between the Fuel Consumption and the rest of the features is presented. In this figure, the relationships are relatively straightforward. Visually, it is evident that a linear relationship exists between the values of fuel consumption and the values of Exhaust Gas Mass Flow, Manifold Pressure, Torque and Rotation Speed. Therefore, someone could argue that the consumption is easily predicted based on those values.

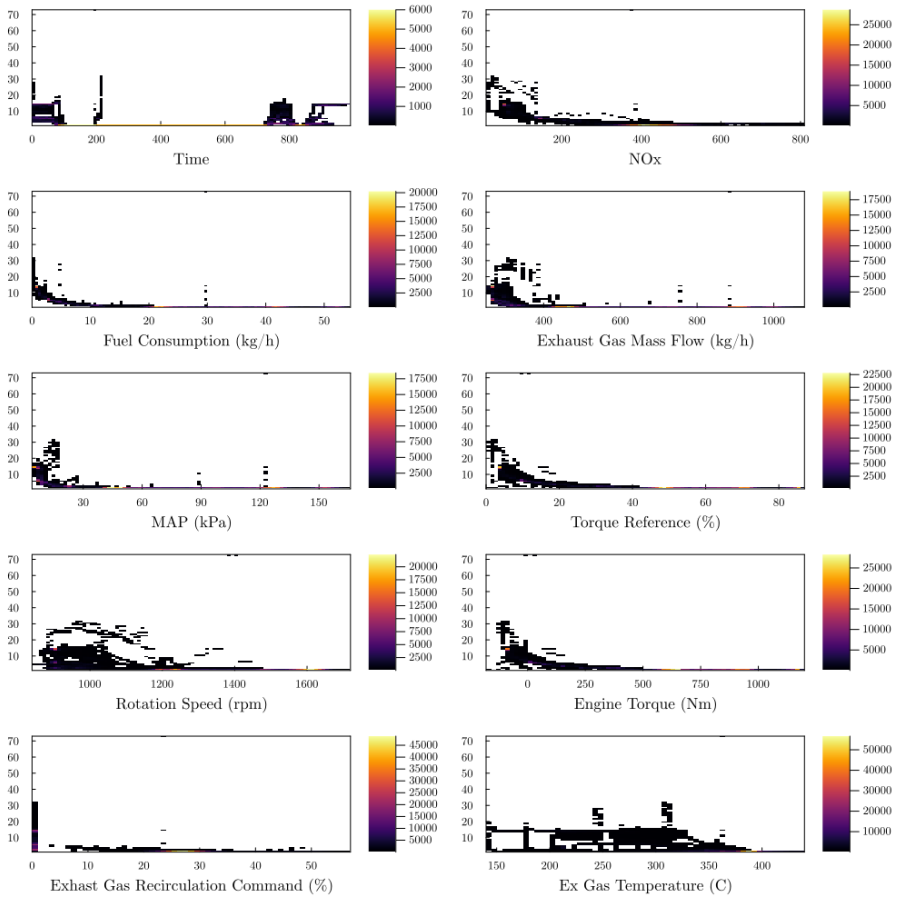

Figure 5.12: 2-dimensional Histograms of the target parameter λ and each feature, in the form of heat-map.

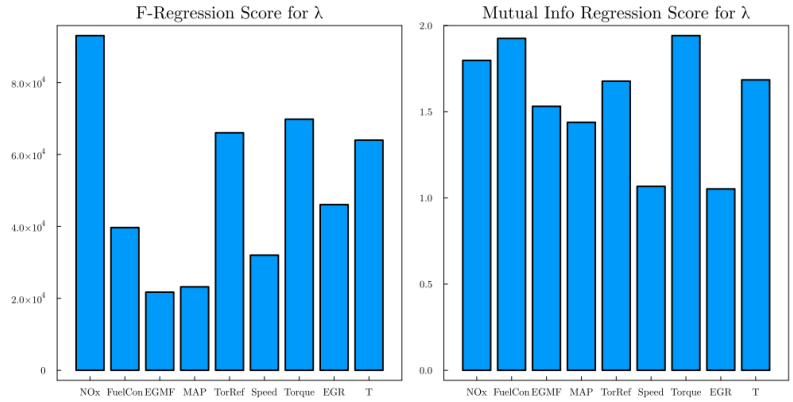

Figure 5.13: F-test score and Mutual Info Regression score for λ and the the relevant features.

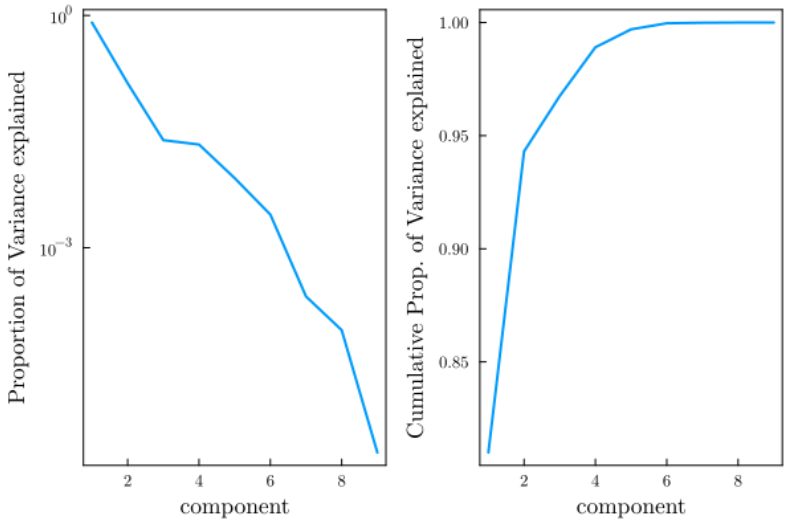

Figure 5.14: Proportion of Variance Explained and Cumulative Proportion of Variance Explained with respect to the number of components for the feature selection of λ

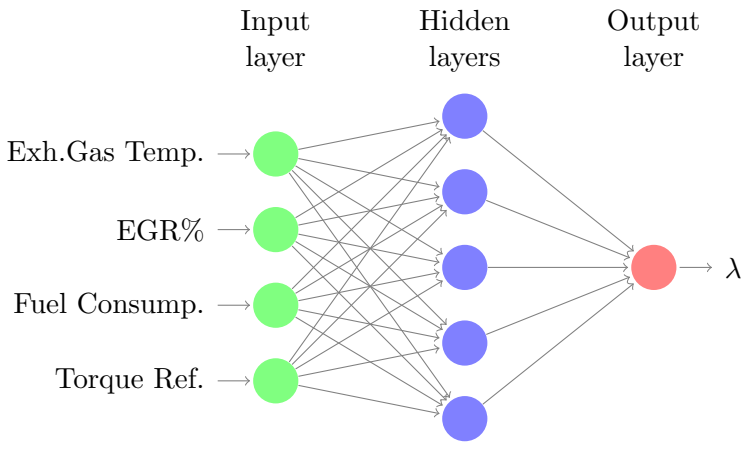

Figure 5.15: Demonstration of the inputs of the model for λ prediction. These are the λ, the EGR% command, the Fuel Consumption, the Exhaust Gas Mass Floww and the Exhaust Gas Temperature.

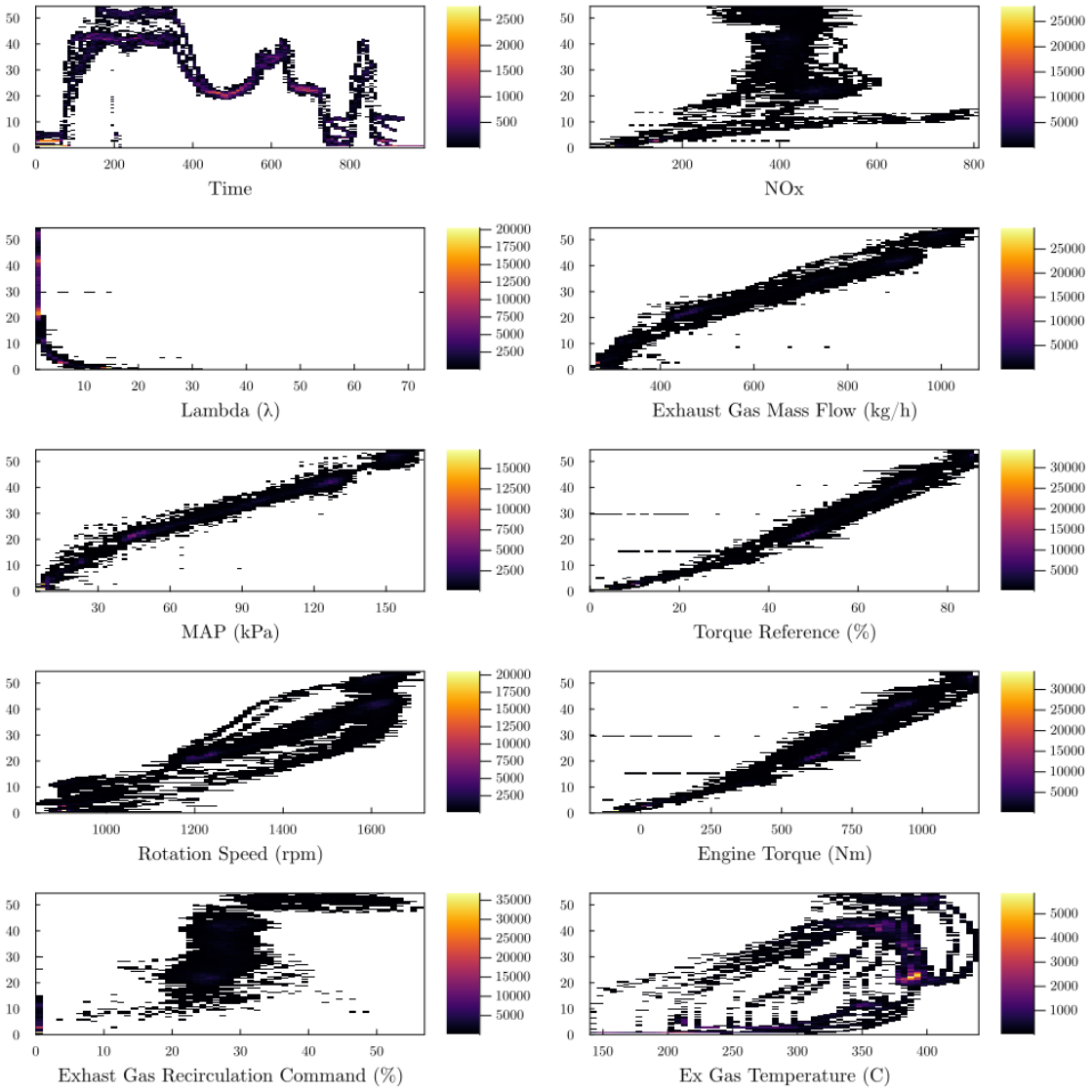

Figure 5.16: 2-dimensional Histograms of the target parameter Fuel Consumption and each feature, in the form of heat-map.

From figure (5.17), regarding the F-Regression score, we see a high correlation between the target value and the Manifold Pressure, the Torque Reference and the Manifold Pressure. The Mutual Info Regression, flips the Torque Reference with the absolute Engine Torque, Exhaust Gas Mass Flow and finds a high correlation with NO_X. However again, we assume that Nitrogen Oxides sensors are not available to provide measurements as inputs.

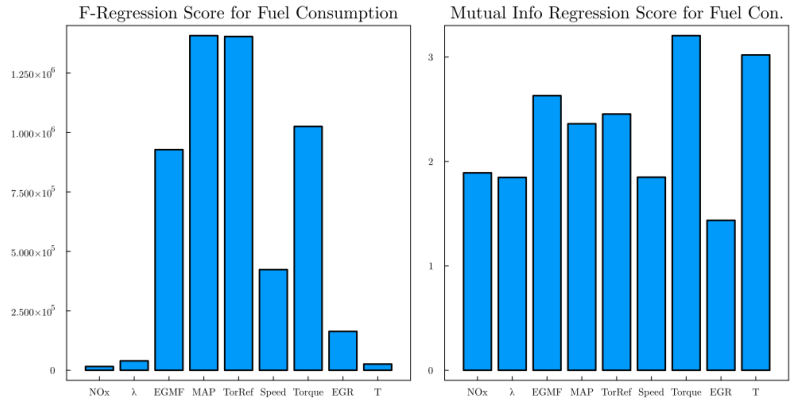

Figure 5.17: F-test score and Mutual Info Regression score for Fuel Consumption and the the relevant features.

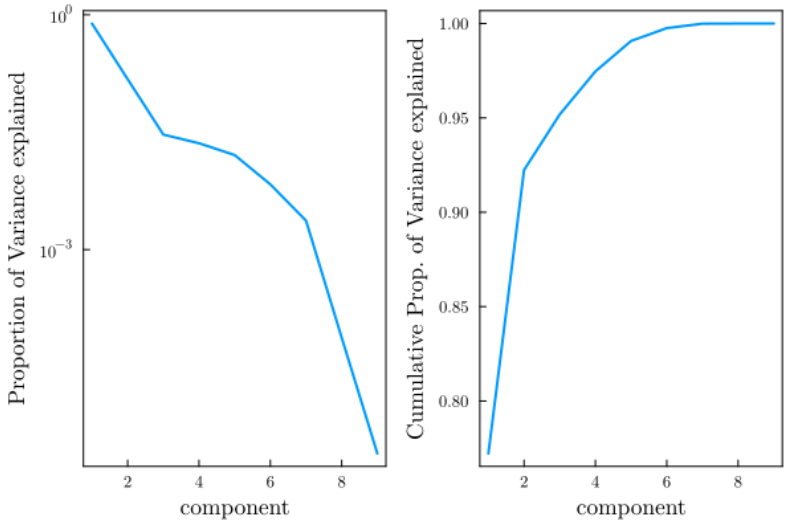

Figure 5.18: Proportion of Variance Explained and Cumulative Proportion of Variance Explained with respect to the number of components for the feature selection of Fuel Consumption

From figure (5.18) it can be concluded that roughly four input features are required for a model to obtain full proportion of variance explained for the features that will construct the Fuel Consumption Model. The models inputs are presented in (5.5.3) and are based on the above outcomes.

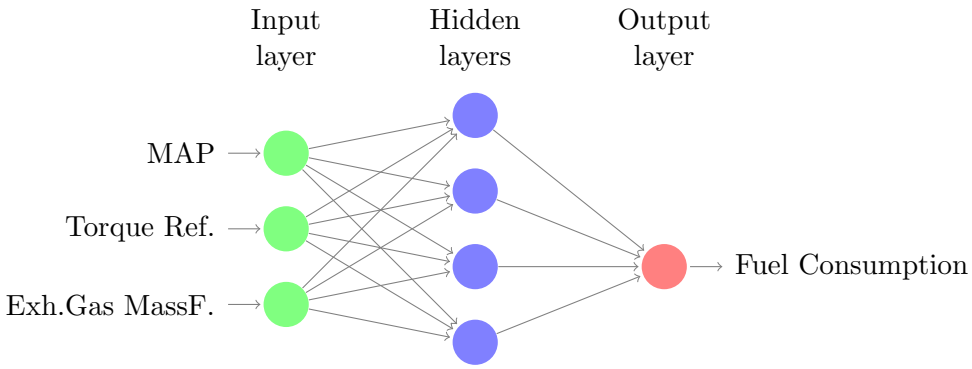

Figure 5.19: Demonstration of the inputs of the model for Fuel Consumption. These are the λ, the EGR% command, the Fuel Consumption, the Exhaust Gas Mass Flow and the Exhaust Gas Temperature.

5.5.4 Manifold Pressure Mosel Input Selection

The pairwise distributions between the Manifold Pressure and the remaining features are shown in figure (5.20). The relationships in this figure are rather clear-cut. Visually, it is clear that there is a linear relationship between manifold pressure and rotation speed, torque, exhaust gas mass flow, and fuel consumption. Given those values, the manifold pressure should be easily predicted.

Regarding the F-Regression score, we can observe from figure (5.21) that there is a strong association between the goal value and the fuel consumption and exhaust gas mass flow. The Engine Torque and Exhaust Gas Temperature are added via the Mutual Information Regression (T).

From figure (5.22) it can be concluded that roughly four input features are required for a model to obtain full proportion of variance explained for the features that will construct the Fuel Consumption Model. The models inputs are presented in (5.5.4) and are based on the above outcomes.

The Manifold Pressure model needs around four input features to get the entire proportion of variation explained for the variables that make up the Fuel Consumption Model, according to figure (5.22). Based on the aforementioned results, the model inputs are shown in (5.5.4).

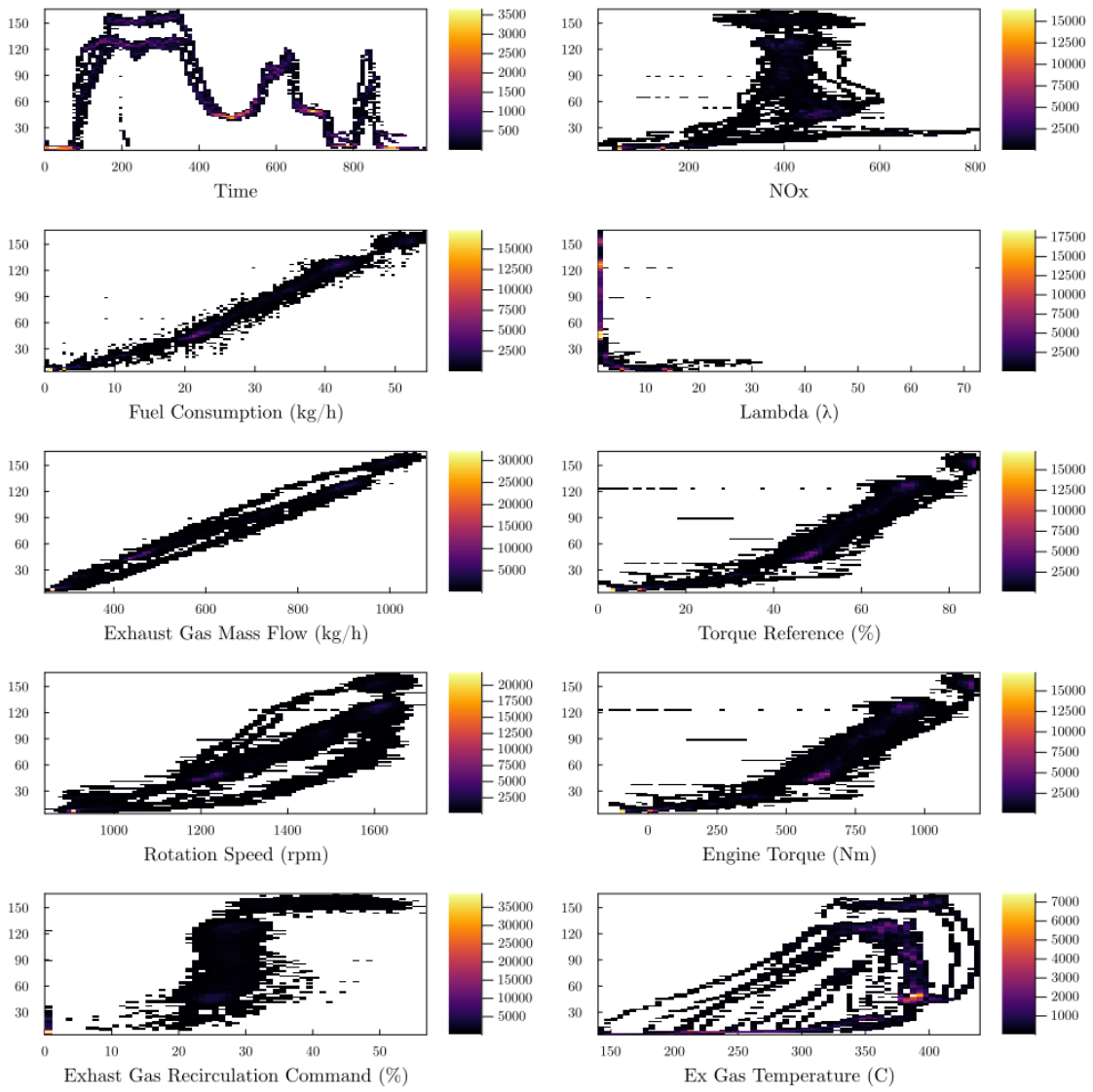

Figure 5.20: 2-dimensional Histograms of the target parameter Manifold Pressure and each feature, in the form of heat-map.

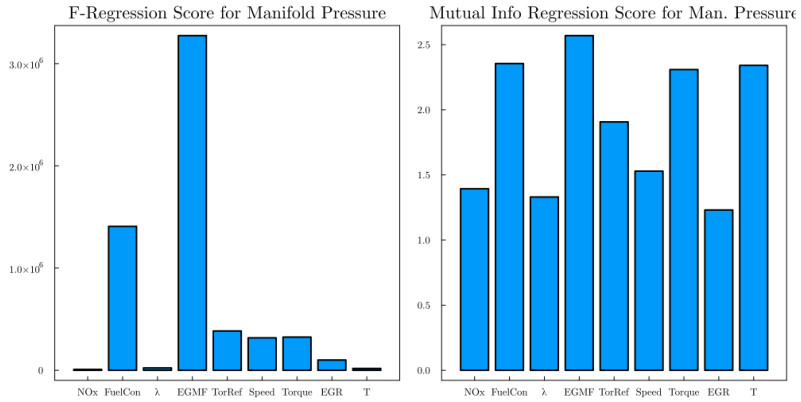

Figure 5.21: F-test score and Mutual Info Regression score for Manifold Pressure and the the relevant features.

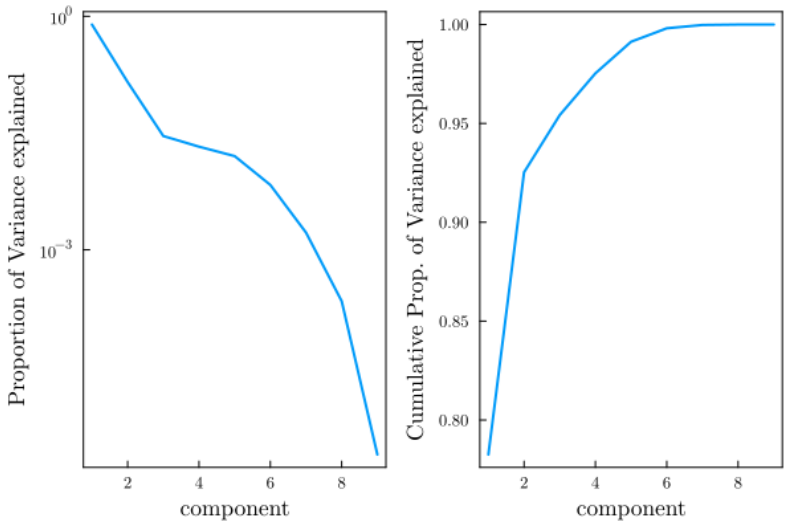

Figure 5.22: Proportion of Variance Explained and Cumulative Proportion of Variance Explained with respect to the number of components for the feature selection of Manifold Pressure.

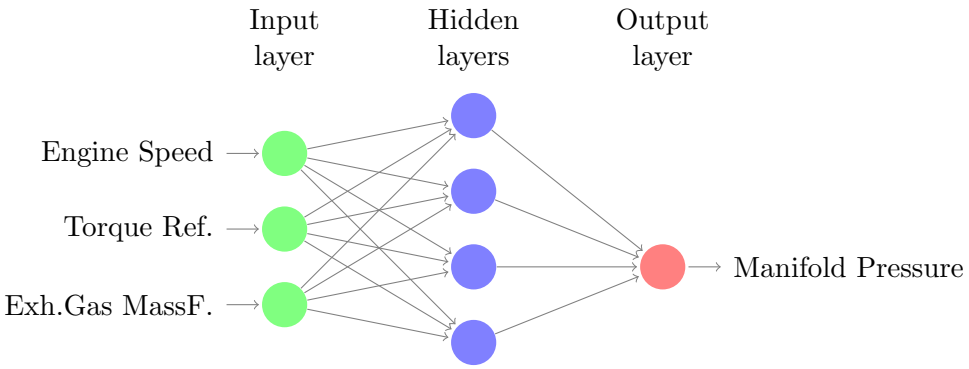

Figure 5.23: Demonstration of the inputs of the model for Manifold Pressure. These are the Engine Torque, the Exhaust Gas Mass Flow and the Engine Speed.

Chapter 6

Training and Results

In the current chapter, the finalized architectures of the models are presented as well as the training results for them. The process, entails a significant amount of hieuristic approach, as the process is very much a try-error one. After a lot of design iterations the multi channel architecture was opted for. The reason was, that the models constructed with this type of approach filtered the information better and were able to produce results easily generalized. Additionally, it is observed that the image-like architecture overfitted the data and was performing accurately on the validation data.

6.1 Flowchart of Training

The model construction initiates with the target parameter and the input selection done in the previous chapter. Following that, the data are normalized. From then on, the try error process begins where different architectures are investigated, as well as different specifications of each architecture tried. Subsequently the different models constructed are initially evaluated on the training and testing data.

Following that, they are also evaluated on the validation data which is the important par. In parallel, what happens in the hyperparameter tuning. Meaning that once a certain type of architecture is picked, certain variations of parameters such as the activation function the kernel size and the depth of the network are tried out in order to find the optimal recipe. The setup also includes the type of loss function chosen as well as the optimizer, its learning rate and the amount of epochs. The training proceeds as follows:

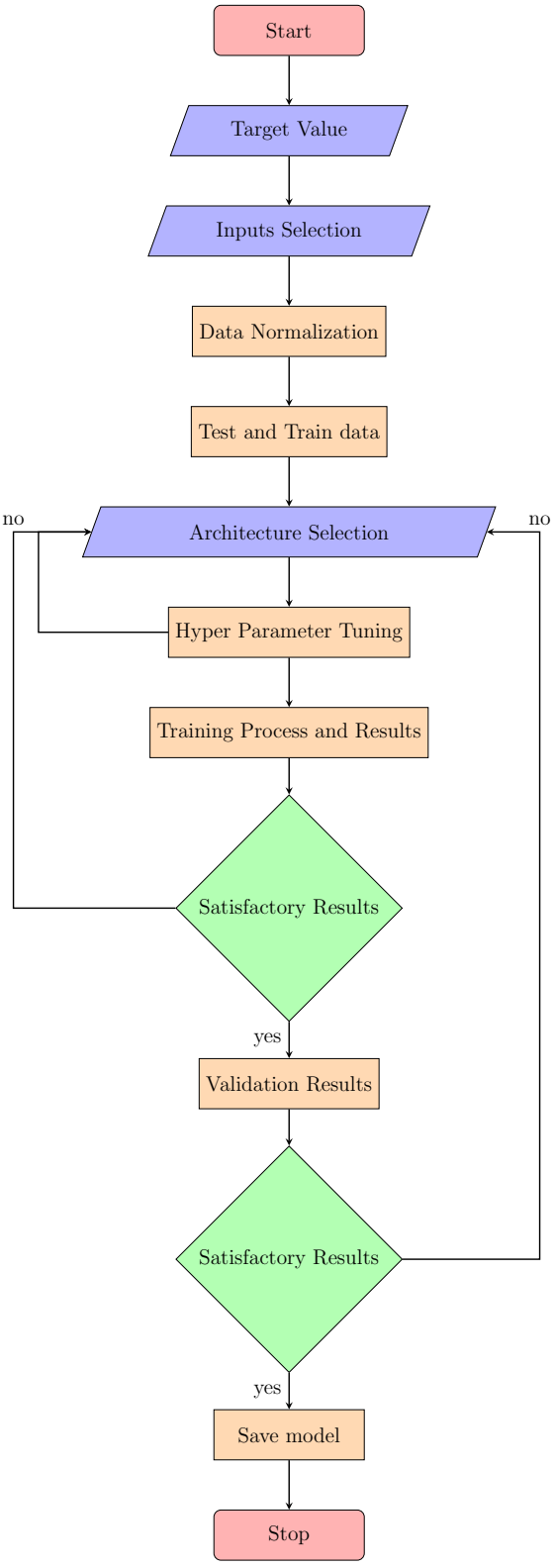

Figure 6.1: Flowchart of the training procedure as well as the evaluation of results. A try-error process.

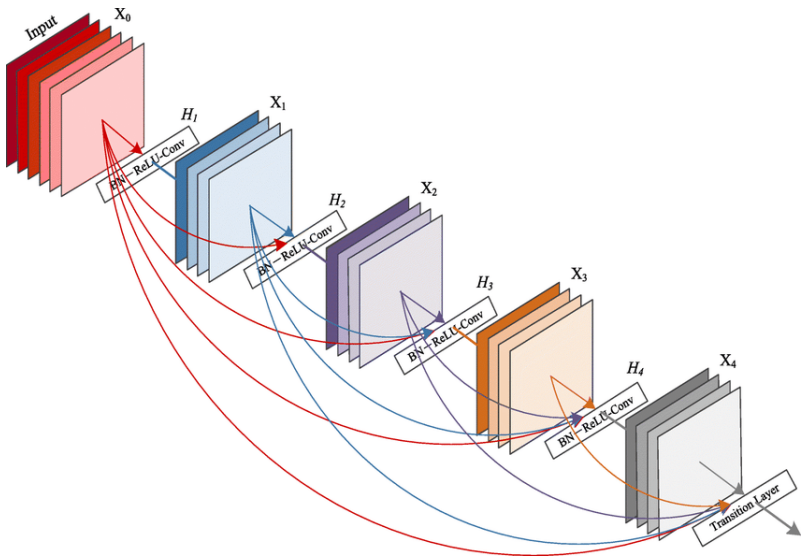

Figure 6.2: Illustration of the DenseNet architecture, the inspiration for the currently employed one. Skip connections are present between the blocks concatenating the channels at the end. The channels are visualized by the rectangles. Note that because of the concatenation, the channel number is added.

6.2 NO_X Model

The model for the prediction of Nitrogen Oxides is configured to take five inputs. These are, the λ, the EGR% command, the Exhaust Gas Temperature, the Exhaust Gas Mass Flow and the Engine Torque. The model accepts as input the values at ten time steps of each feature in a different channel, five in total for this one. The model can be broken up in to five different parts.

The channel growth rate in this network is five, meaning that 5 channels are added in each part, except the first one where channels grow from five to fifteen. A skip-connection exists between each block, meaning that the second block is connected with a skip connection with the third, the fourth and the fifth. The third is connected with the fourth and so on. At the end of each skip connection a channel concatenation takes place, meaning that the information is fused, after the initial processing of each channel separately in the first and second block.

In the first part, each channel is processed by a convolutional kernel of 1×1 size, and the channels are magnified by a factor of three, meaning they become fifteen. What follows, is a BatchNorm layer for the whole channels as well as a MeanPool layer of size 1×3. The next part consists of three convolutional blocks, called also Dense Blocks. In each of every of these blocks, a BatchNorm is performed in all the channels, followed by a Convolutional Layer of a kernel of size 1×1, again then a Batchnorm and then again a Convolutional Layer of kernel size 1×3.

After the four blocks, where the feature extraction has been materialized, the predicting part follows. This is made of, a Batchnorm for all the channels, and an Adaptive Mean Polling of size 1×1. Adaptive average pooling is simply an average pooling operation that, given an input and output dimensionality, calculates the correct kernel size necessary to produce an output of the given dimensionality from the given input. After a flattening layer, two classic densely connected layers of size 30 are placed for prediction of the output.

All convolutions are performed with a stride equal to one, a ReLU activation function, and a zero padding on the sized in order to keep the same dimensions. Also, there is no bias used. In the densely connected layer ReLU is also used.

6.2.1 Training Results

The training of the network, is performed with an Adam optimizer and a learning rate of 0.01. The epochs used are one thousand. As seen in figure (6.3) the training process converges. The convergence is not easy and as seen the learning rate changes automatically along the process by the optimizer. The training error is always smaller than the test error, but they remain close meaning that there is no overfitting.

The error starts to be really small after around epoch 250. The convergence is also monitored by a callback function, which evaluates the mean squared error in each epoch for the training and the test data and checks whether there is overfitting or slow convergence. It is also able to stop the training if needed, if the error seems to significantly rise for some consecutive epochs. As, seen in the figure that is not the case.

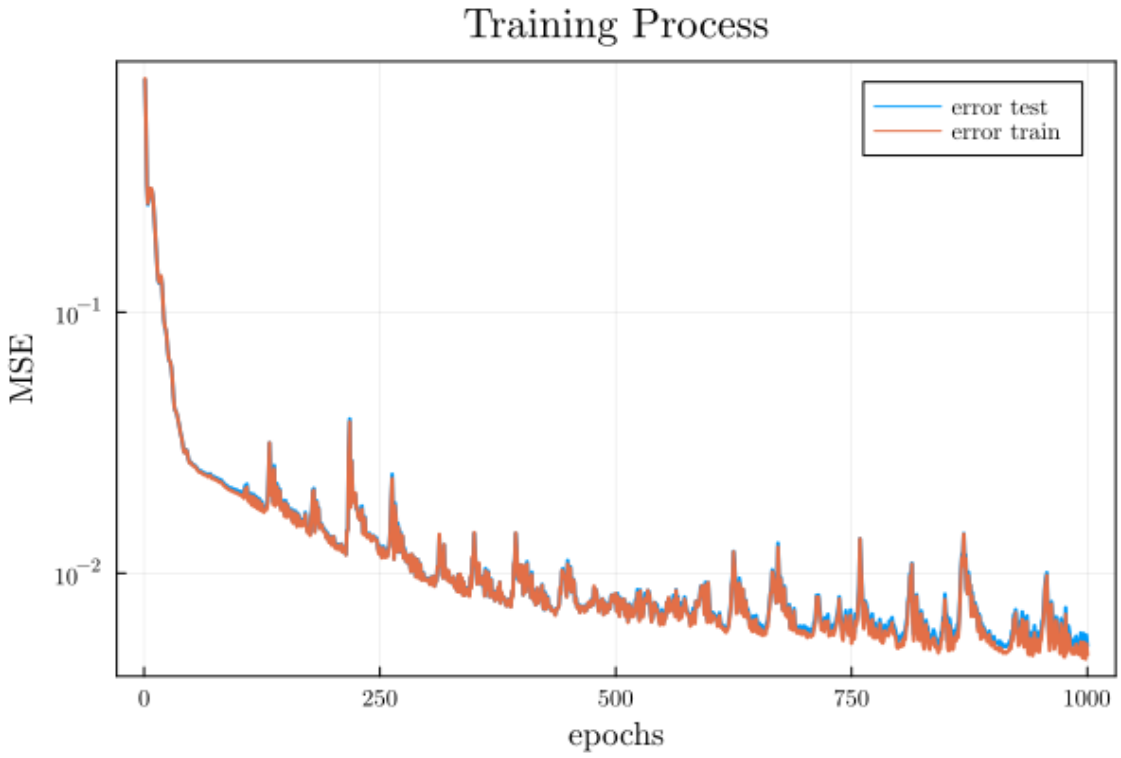

Figure 6.3: Overview of the Training Process of the NO_X model.

Optimizer	Adam
Learning Rate (η)	0.01
Epochs	1000
Training Time	85.6 seconds
Trainable Parameters	5446
Timesteps	10
Rate	1Hz

Table 6.1: NO_X model main characteristics.

The training and test results are presented in table (6.2). These, are evaluated as very accurate both on the training as well as on the test data which is the important part. As expected, the error is a bit higher in the test data which the network has not seen while training.

Metric	Value
Train Comb. Accuracy (R^2)	0.9952
Test Comb. Accuracy (R^2)	0.9948
Mean Comb. Square Error in Train Dataset	0.004867
Mean Comb. Square Error in Test Dataset	0.005257
Mean Comb. Absolute Error in Train Dataset	0.04880
Mean Comb. Absolute Error in Test Dataset	0.05134

Table 6.2: Training and Test results for the NO_X model.

In figure (6.4), the visualization of the predicted values and the real values for one of the data sets is presented. In general, the two time-series ploted, greatly coincide. Additionally, no significant time lag is evident anywhere from the observer constructed. The spikes produced by the prediction are more than the spikes of the original signal.

In figure (6.5) a zoomed region for a time interval of 100 seconds is presented. Again, we observe that the raw predictions are noisy but their values and temporal correlation with the real values are assessed as very good.

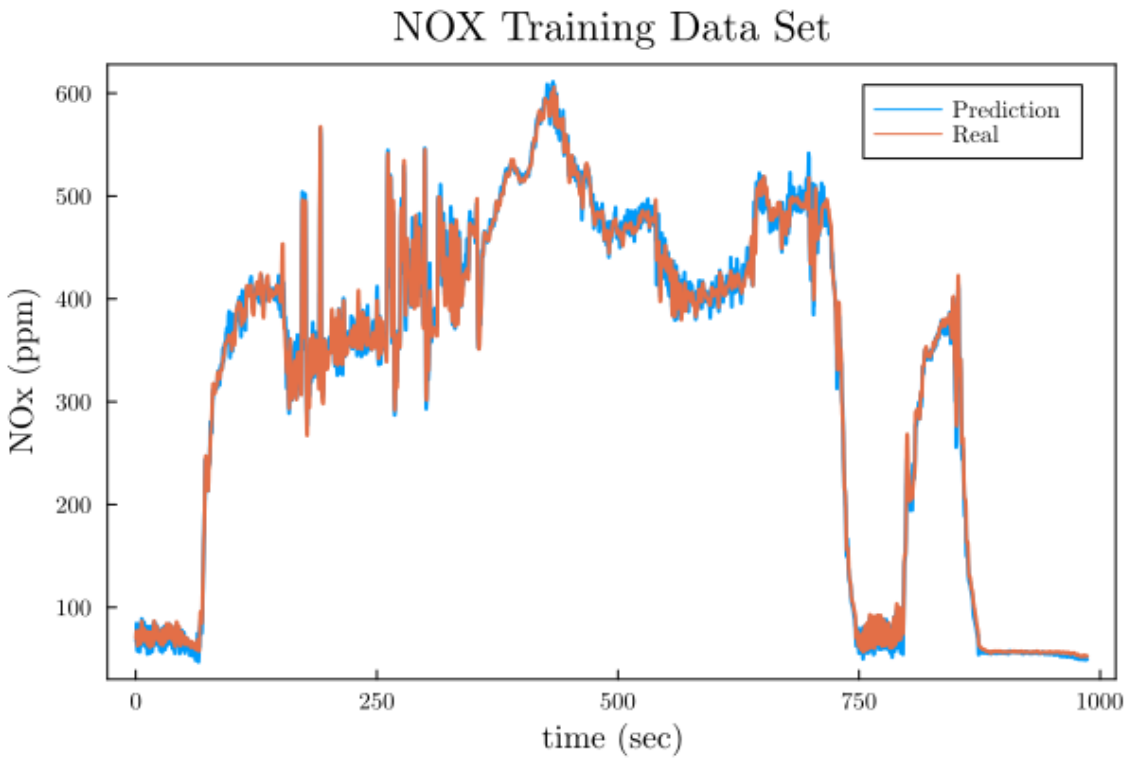

Figure 6.4: Visualization of the NO_X model performance on a training dataset.

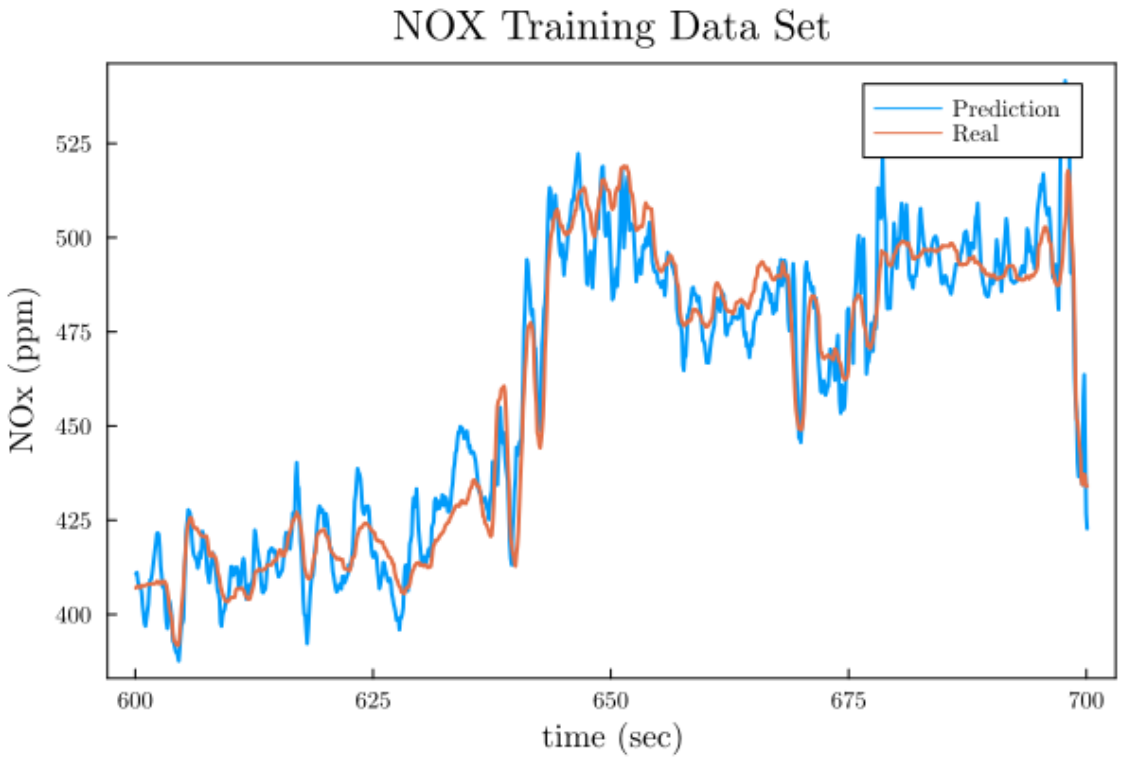

Figure 6.5: Zoom in the visualization of the NO_X model performance on a training dataset.

Figure 6.6: Scatter plot of the performance of the NO_X model on the Training and Test data.

6.2.2 Validation Results

Following the training and testing results, the accuracy of the model is tested on a validation data set, meaning data completely not seen previously by the network. The descriptive statistics of the validation dataset, are not far from the descriptive statistics of the training data set. Nevertheless, the loading cycle is completely different and therefore, a good test. In table (6.3), the quantification of the accuracy of the model is predicted regarding the validation dataset.

The accuracy is really high taking into account that capturing the dynamics of such a signal is a really difficult process. Moreover, although the accuracy is not as high as desired, it should be stated that the accuracy is computed on the scaled data, and therefore it is relative to the scale of the data. A 3% deviation relatively, is not large we take into account the scale of the variable (some hundreds).

Metric	**Value**
Validation Comb. Accuracy (R^2)	0.9711
Mean Comb. Square Error in Validation Dataset	0.02328
Mean Comb. Absolute Error in Validation Dataset	0.1048

Table 6.3: Validation results for the NO_X model.

In figure (6.7) the performance of the constructed model is pictured. It is asessed as very accurate, as there is no time lag while the absolute values are really close. The model fails to follow the sharp rise of the real values at around 100 seconds. This is also evident in the scatter plot figure (6.9), where some points that diverge from the y=x line are present for values between 200 and 300.

However, for the rest of the time series, the resemblance is really close. Again the produced signal is noisier from the actual, for reasons that have to do probably with the frame rate which could be higher. In the zoomed time interval between 600 seconds and 700 seconds, figure (6.8), it is evident that the values are really close. The trend of the real time series is accurately reproduced.

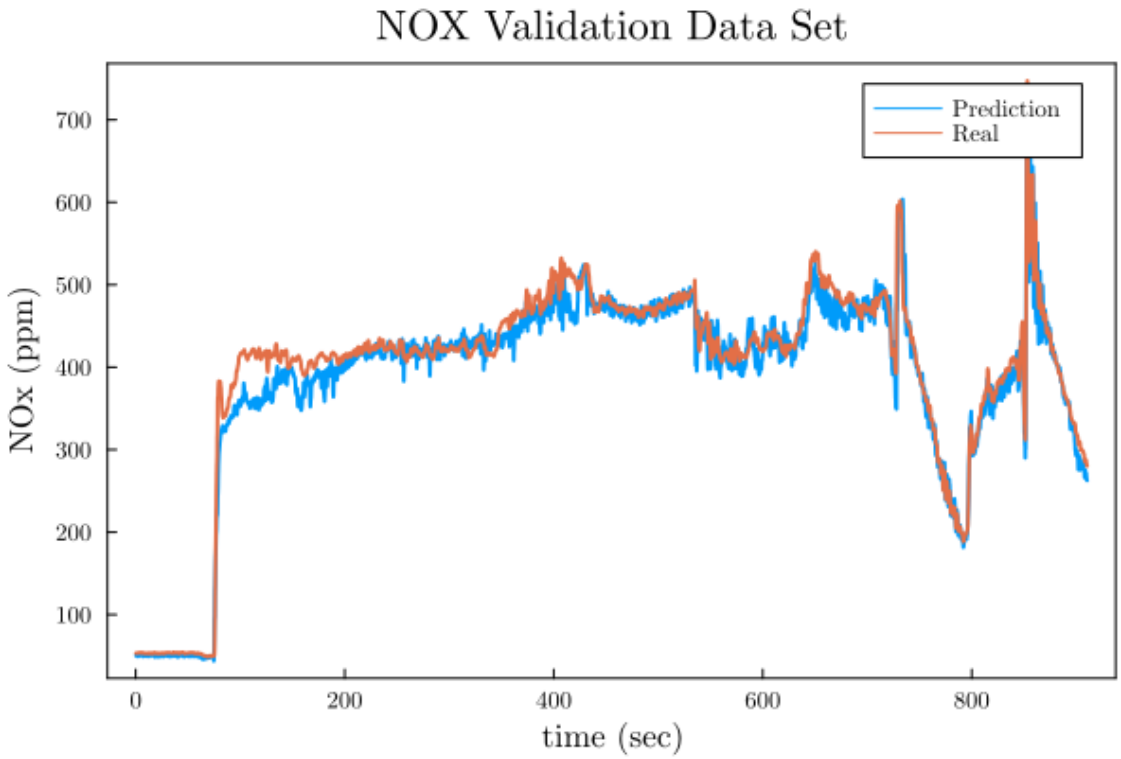

Figure 6.7: Visualization of the NO_X model performance on a validation dataset.

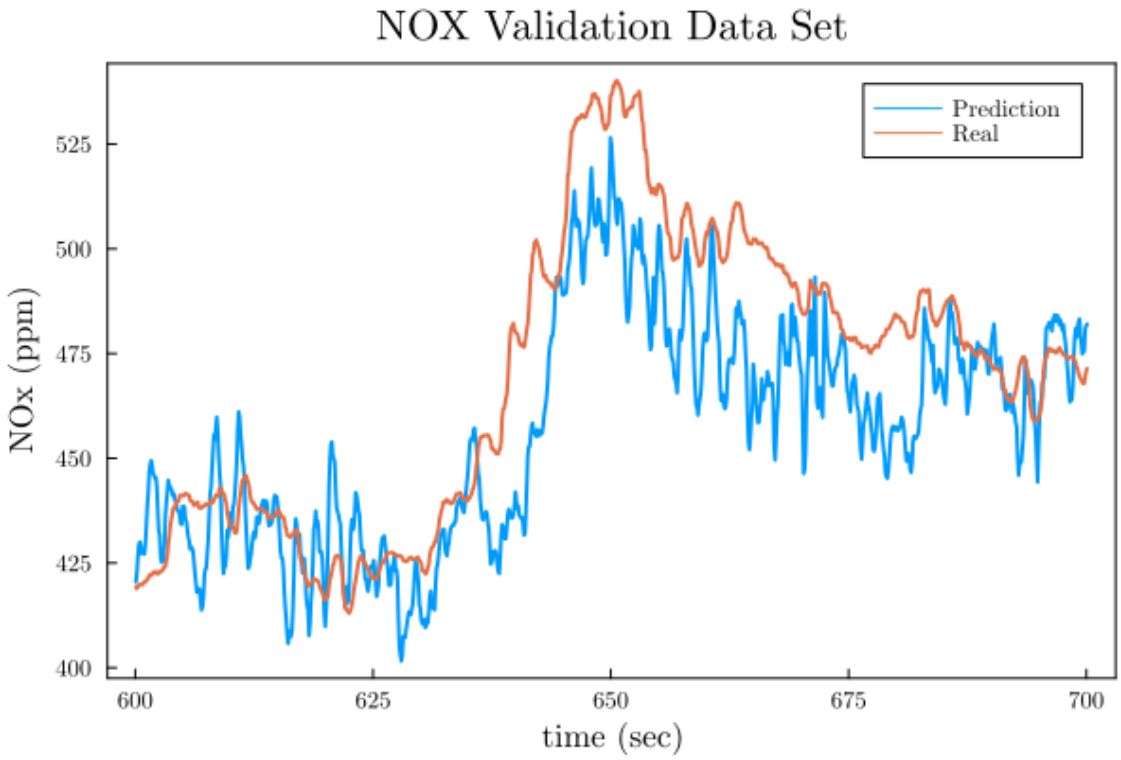

Figure 6.8: Zoom in the visualization of the NO_X model performance on a validation dataset.

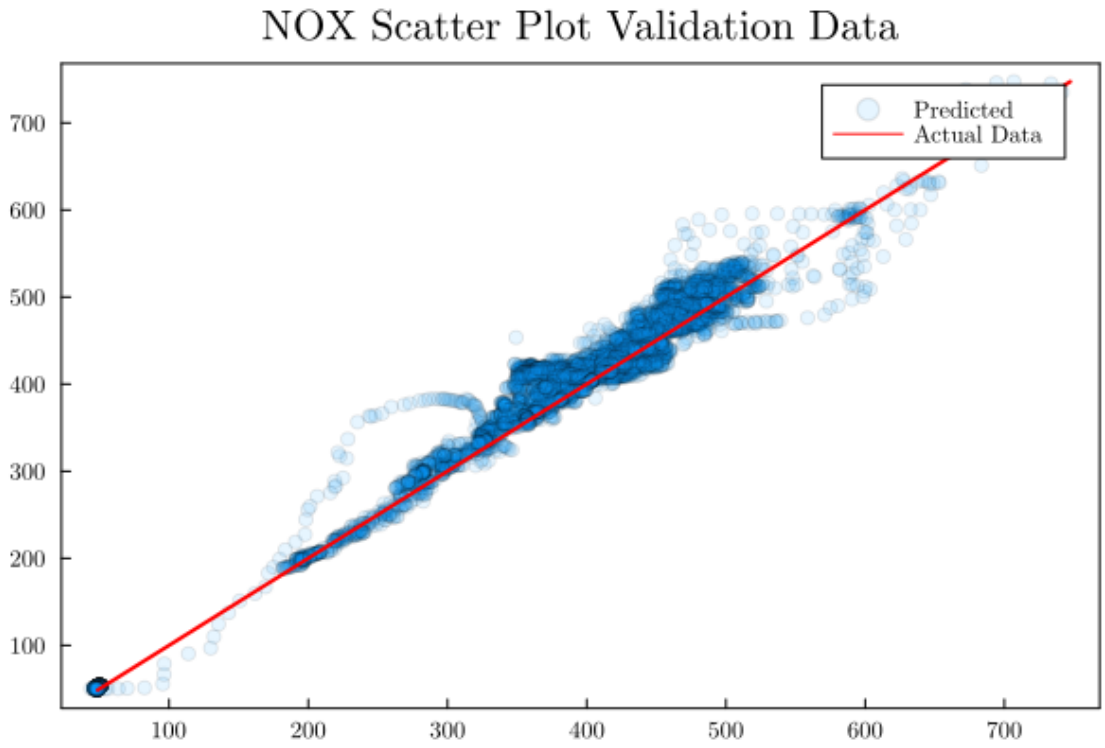

Figure 6.9: Scatter plot of the performance of the NO_X model on the Validation data.

Overall the model is successfully in fulfilling its observing mission. It predicts the values of the content of NO_X's accurately and timely and is able to perform very well on unseen data which is the objective of training a neural network. The above ideas, are not conclusive as the model needs to be tested real time and be reevaluated for its performance as well as real time computational cost in the embedded hardware.

6.3 Lambda (λ) Model

The model for the prediction of λ is configured to take four inputs and follows a similar architecture to the Nitrogen Oxides model constructed. These are, the Exhaust ,the EGR% command, the Exhaust Gas Temperature, the Exhaust Gas Mass Flow and the Engine Torque. In other words, we configure it for simplicity with the same inputs as the NO_X model except of course with itself. The model accepts as input the values at ten time steps of each feature in a different channel, four in total for this one. The model can be broken up in to five different parts the same way as the NO_X one.

The channel growth rate in this network, in similar fashion is set to five, meaning that 5 channels are added in each part, except the first one where channels grow from four to fifteen. A skip-connection exists between each block, meaning that the second block is connected with a skip connection with the third, the fourth and the fifth. The third is connected with the fourth and so on. At the end of each skip connection a channel concatenation takes place, meaning that the information is fused, after the initial processing of each channel separately in the first and second block.

In the first part, each channel is processed by a convolutional kernel of 1×1 size, and the channels are magnified. What follows, is a BatchNorm layer for the whole channels as well as a MeanPool layer of size 1×3. The next part consists of three convolutional blocks,

called also Dense Blocks. In each of every of these blocks, a BatchNorm is performed in all the channels, followed by a Convolutional Layer of a kernel of size 1×1, again then a Batchnorm and then again a Convolutional Layer of kernel size 1×3.

After the four blocks, where the feature extraction has been materialized, the predicting part follows. This is made of, a Batchnorm for all the channels, and an Adaptive Mean Polling of size 1×1. Adaptive average pooling is simply an average pooling operation that, given an input and output dimensionality, calculates the correct kernel size necessary to produce an output of the given dimensionality from the given input. After those, tow classic densely connected layers of size 30 are placed for prediction of the output.

To maintain the same dimensions, all convolutions are carried out with a stride of 1, and zero padding on the sides. Furthermore, no bias is employed. ReLU is utilized in both the densely connected layers as well as the convolutions.

6.3.1 Training Results

The training of the network, is performed with an Adam optimizer and a learning rate of 0.01. The epochs used are again one thousand. As seen in figure (6.10) the training process converges. Once again, the training converges as both the training error and the test error vanish as the iterative process advances simultaneously.

Once more, the convergence is not straightforward and a lot of epochs are required. The learning rate is adapted by the optimizer in order for the error to drop more. After the epoch 900 we observe that the test error and train error have started ti diverge from each other and consequently the training stops at 1000 epochs in order to avoid overfitting the training data.

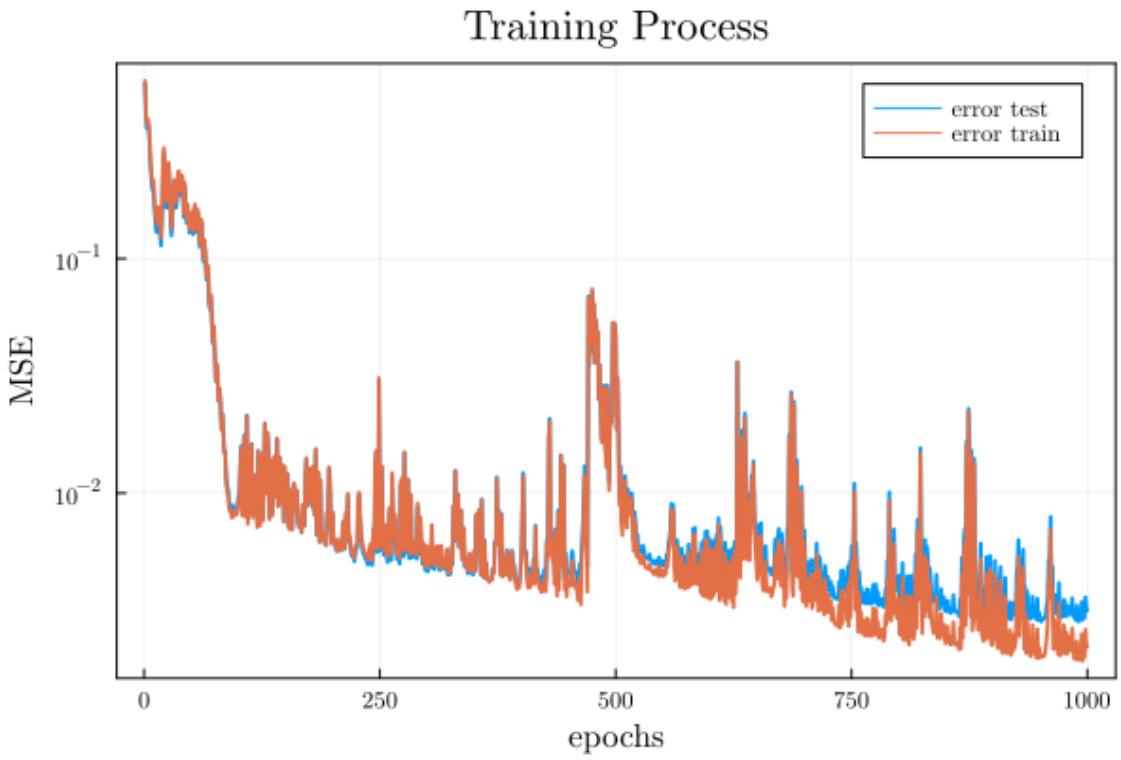

Figure 6.10: Overview of the Training Process of the λ model.

Table (6.5) displays the test and training results. These are rated as being extremely

Optimizer	Adam
Learning Rate (η)	0.01
Epochs	1000
Training Time	85.41 s
Trainable Parameters	5431
Timesteps	10
Rate	1Hz

Table 6.4: λ model main characteristics.

accurate on both the test data, which is what matters, and the training data. The error is larger in the test data than it is in the training data, as expected.

Metric	**Value**
Train Comb. Accuracy (R^2)	0.9979
Test Comb. Accuracy (R^2)	0.9969
Mean Comb. Square Error in Train Dataset	0.002206
Mean Comb. Square Error in Test Dataset	0.003155
Mean Comb. Absolute Error in Train Dataset	0.017455
Mean Comb. Absolute Error in Test Dataset	0.018524

Table 6.5: Training and Test results for the λ model.

Figure (6.11) shows a depiction of the actual values and anticipated values for one of the data sets. The two time-series plots generally have a high degree of agreement. Furthermore, no discernible temporal lag from the observer-constructed is visible anywhere. More spikes than the original signal's spikes are produced by the prediction.

A magnified area for a period of 100 seconds is shown in figure (6.12). The raw forecasts are noisy, as we have seen before, but their values and temporal association with the actual values are rated as being extremely good.

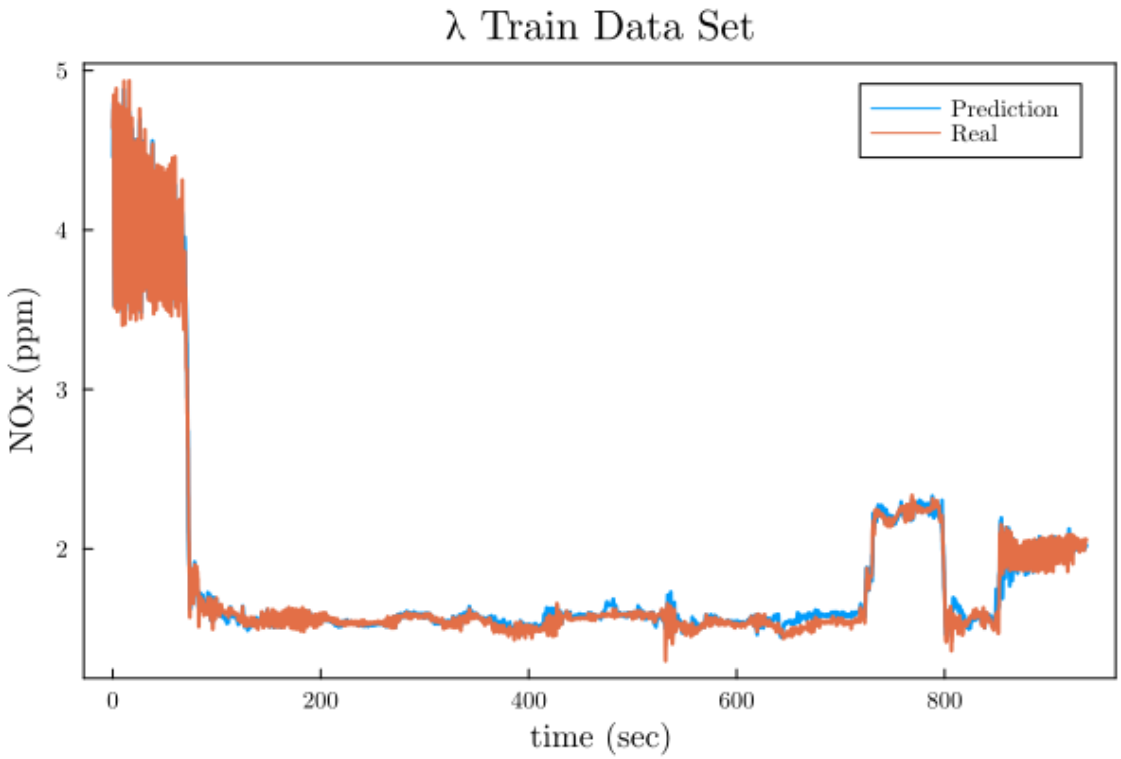

Figure 6.11: Visualization of the λ model performance on a training dataset.

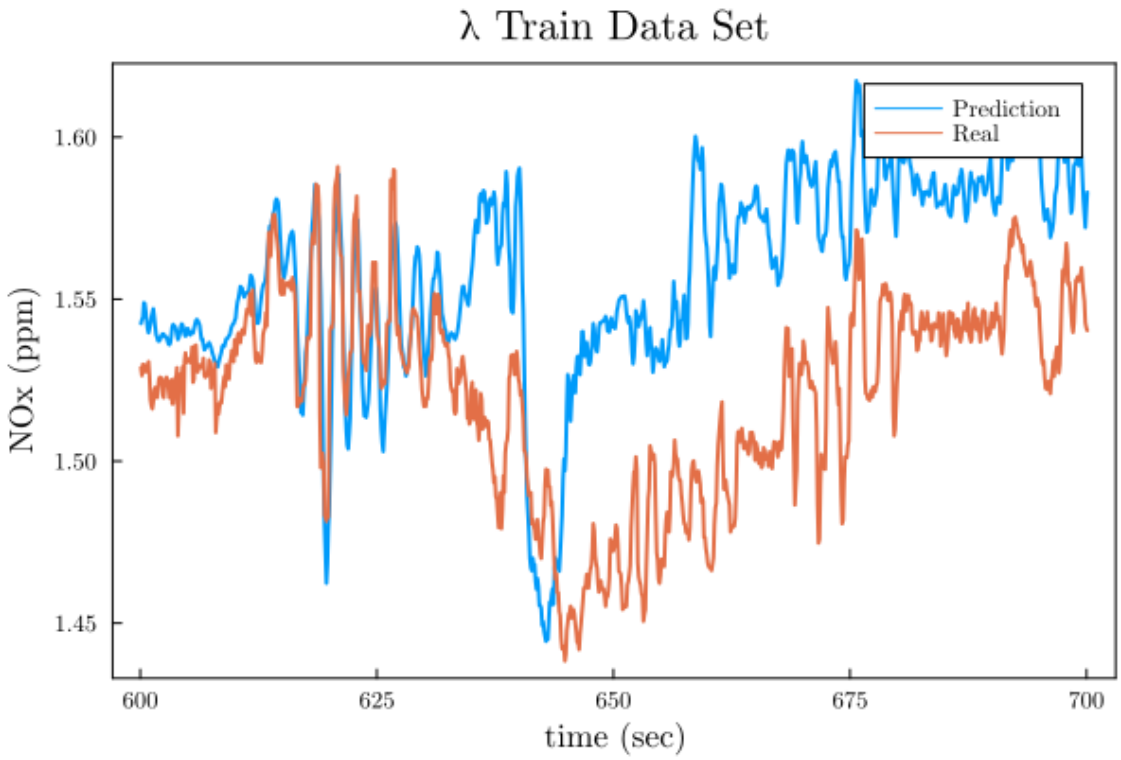

Figure 6.12: Zoom in the visualization of the λ model performance on a training dataset.

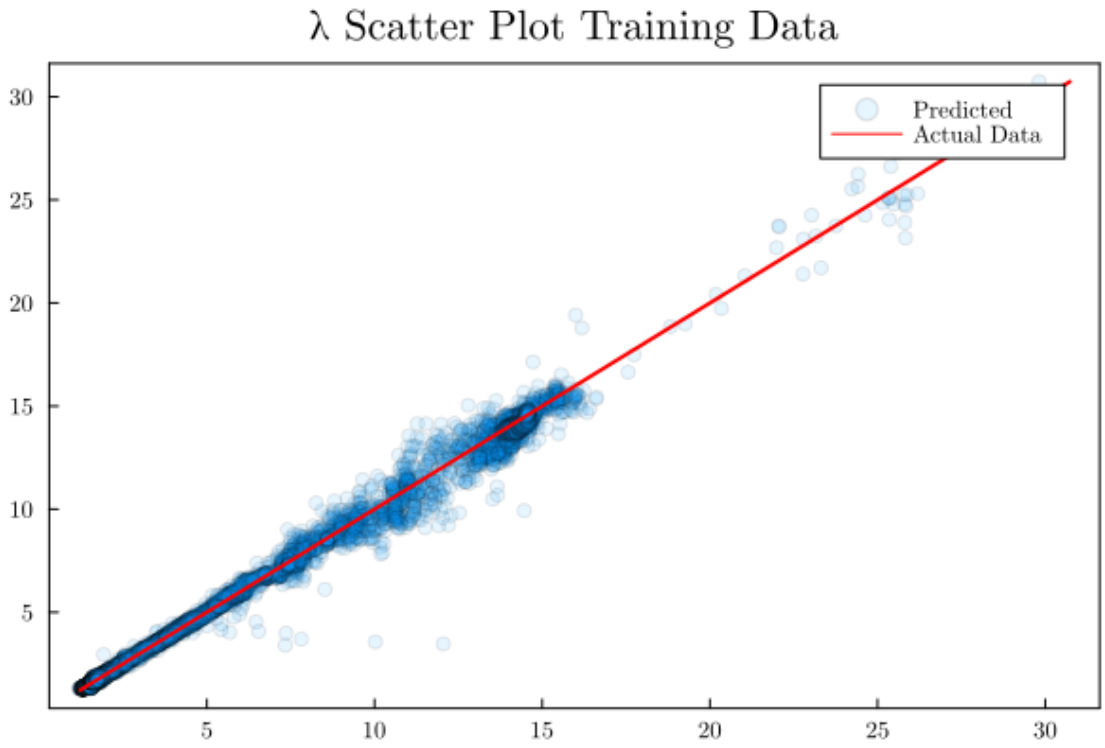

Figure 6.13: Scatter plot of the performance of the λ model on the Training and Test data.

6.3.2 Validation Results

Following the results of the training and testing phases, the model's correctness is evaluated using a validation data set, which consists of data that the network has never seen before. It is again stated that, the training data set's descriptive statistics and the validation dataset's descriptive statistics are very similar. This means that the minimum, maximum and median values should be of the same scale. However, the loading cycle is quite distinct and is a useful test. Table (6.6) presents a quantification of the model's accuracy in relation to the validation dataset.

Considering how challenging it is to capture the physics of such a signal, the accuracy is really high.

Metric	Value
Validation Comb. Accuracy (R^2)	0.9980
Mean Comb. Square Error in Validation Dataset	0.001901
Mean Comb. Absolute Error in Validation Dataset	0.01720

Table 6.6: Validation results for the λ model.

Figure (6.14) illustrates the created model's performance. Given that there is no temporal lag and the absolute numbers are extremely near, it is considered to be very accurate. The representation of the actual signal is very accurate with only some faulty predictions around 850 seconds at the end of the cycle.

This can also be seen in the scatter plot figure (6.16), where certain spots for values between 200 and 300 deviate from the y=x line. This happens for some low engine loads,

at the beginning of the cycle, that correspond to high λ values and as stated for some measurements at the end of the cycle.

The similarity is really striking for the remaining time series, though. Once more, the generated signal is noisier than the actual. In the zoomed time interval between 600 seconds and 700 seconds, figure (6.15) , it is evident that the values are really close, with absolute deviations smaller than 0.05.

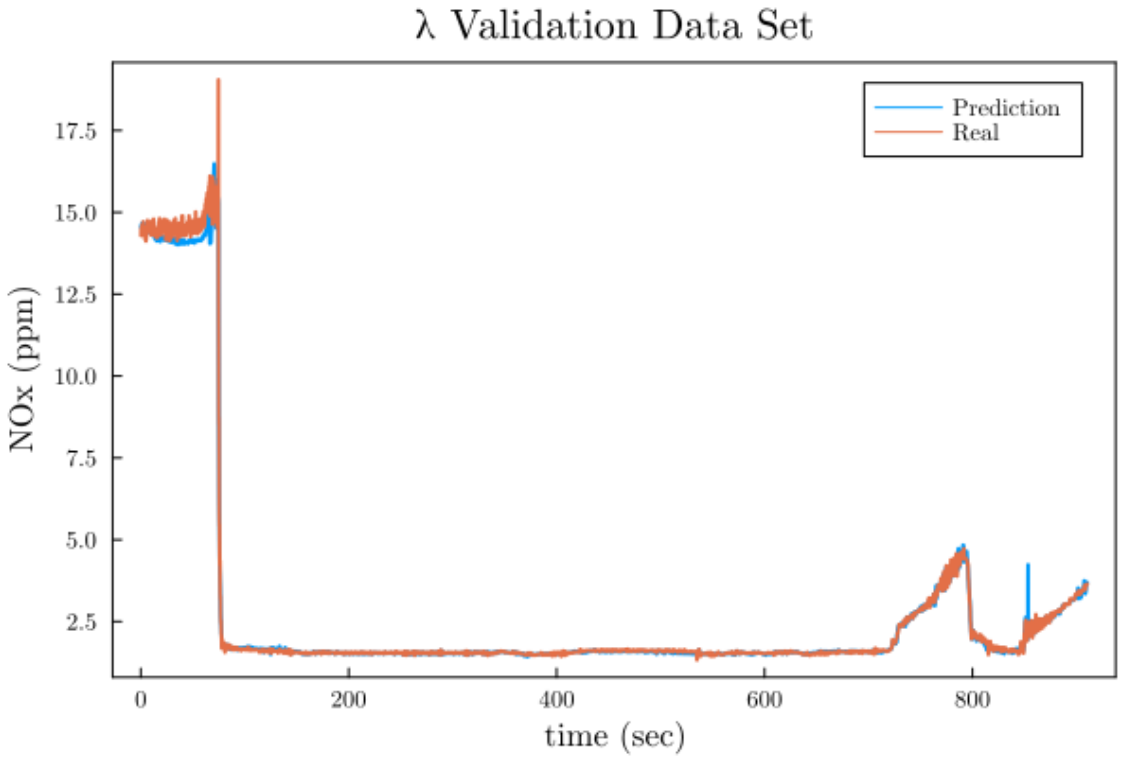

Figure 6.14: Visualization of the λ model performance on a validation dataset.

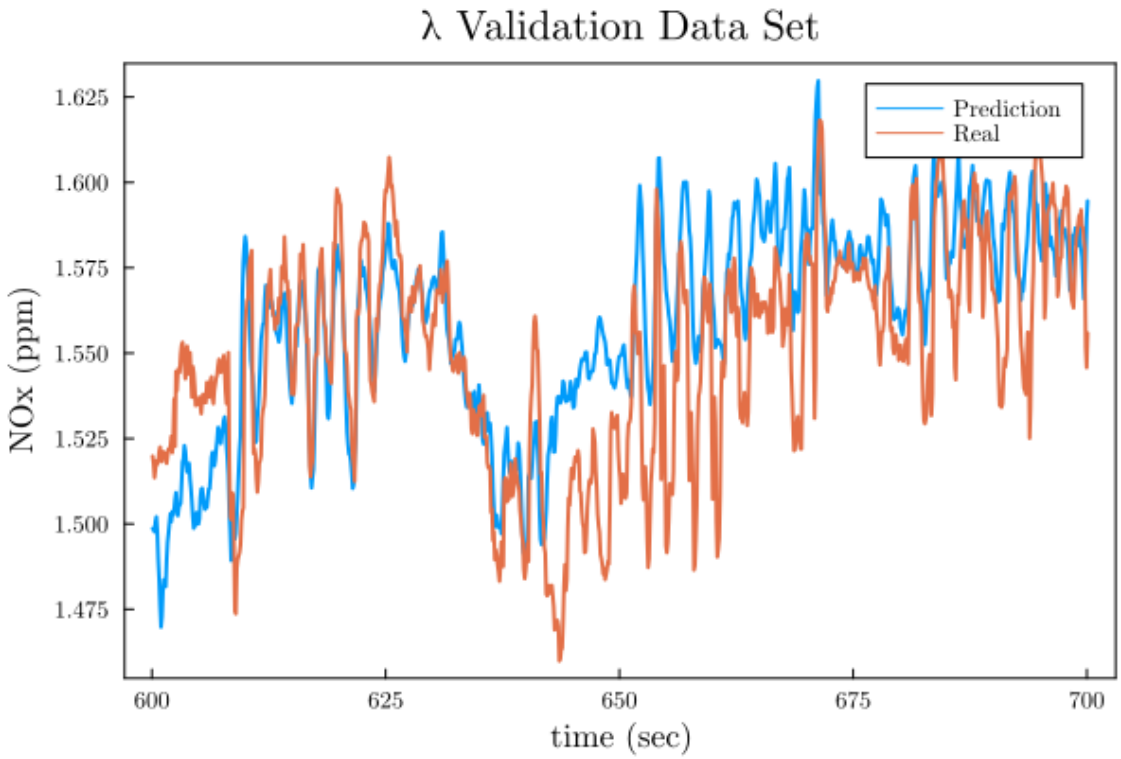

Figure 6.15: Zoom in the visualization of the λ model performance on a validation dataset.

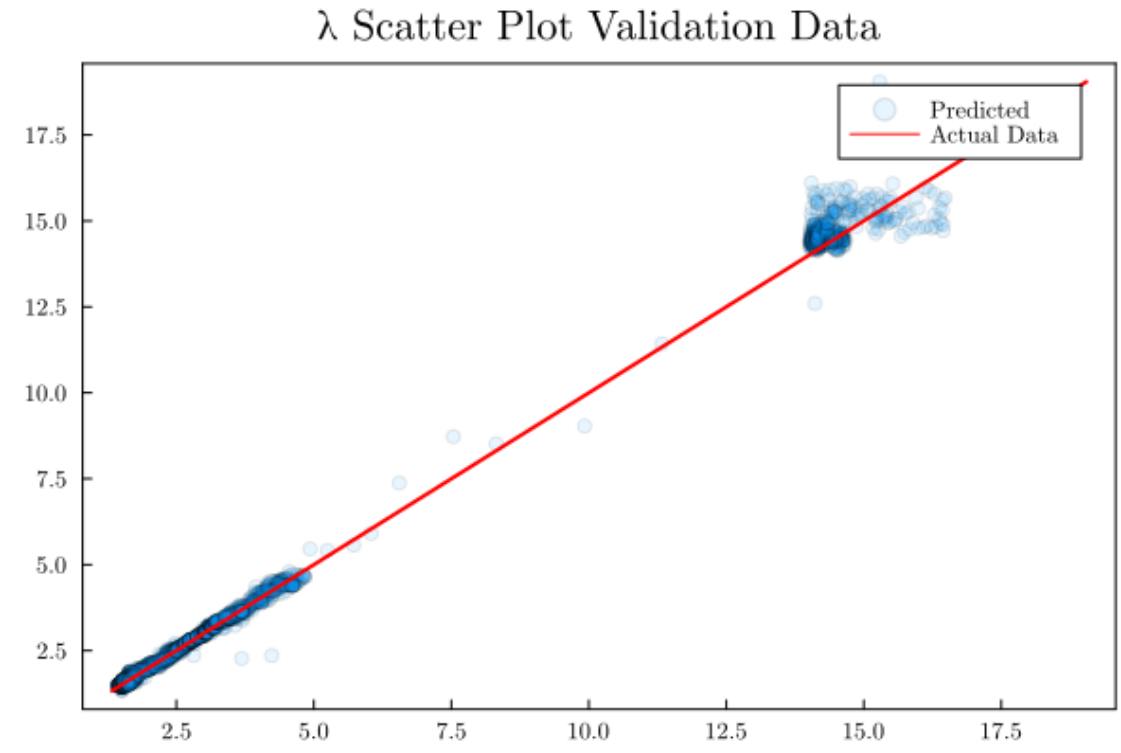

Figure 6.16: Scatter plot of the performance of the λ model on the Validation data.

Overall, the model is effective at carrying out its observational task. The goal of training a neural network is for it to perform well on unknown input, and it predicts the values λ

accurately and promptly. The aforementioned conclusions are not definitive because the model needs to be validated in real time and reevaluated for both its performance and the embedded hardware's real-time computing cost.

6.4 Fuel Consumption Model

Three inputs can be used with the fuel consumption prediction model as it is set up. These are the Exhaust Gas Mass Flow, the Torque Reference, and the Manifold Pressure. The values at ten time steps for each feature in a distinct channel—three in total for this one—are accepted as input by the model. The model can be divided into five distinct components.

With the exception of the first part, where channels increase from five to fifteen, this network has a five channel growth rate, meaning that five channels are added in each part. Each block has a skip connection, which means that the second block has a skip connection to the third, fourth, and fifth blocks. The third is linked with the fourth, and so forth. After each channel has been initially processed independently in the first and second block, a channel concatenation occurs at the conclusion of each skip connection, meaning that the information is fused.

In the first part, each channel is processed by a convolutional kernel of 1×1 size, and the channels are magnified by a factor of three, meaning they become fifteen. What follows, is a BatchNorm layer for the whole channels as well as a MeanPool layer of size 1×3. The next part consists of three convolutional blocks, called also Dense Blocks. In each of every of these blocks, a BatchNorm is performed in all the channels, followed by a Convolutional Layer of a kernel of size 1×1, again then a Batchnorm and then again a Convolutional Layer of kernel size 1×3.

After the four blocks, where the feature extraction has been materialized, the predicting part follows. This is made of, a Batchnorm for all the channels, and an Adaptive Mean Polling of size 1×1. Adaptive average pooling is simply an average pooling operation that, given an input and output dimensionality, calculates the correct kernel size necessary to produce an output of the given dimensionality from the given input. After those, tow classic densely connected layers of size 30 are placed for prediction of the output.
The forecasting portion comes after the first four blocks, where the feature extraction has been manifested.This is composed of an Adaptive Mean Polling of size 1×1 and a Batchnorm for each of the channels. Following that, two conventional, densely connected layers of size 30 are set up to anticipate the outcome.

Once more, all convolutions are performed with a stride equal to one, a ReLU activation function, and a zero padding on the sized in order to keep the same dimensions. Also, there is no bias used. In the densely connected layer ReLU is also used.

6.4.1 Training Results

The training of the network, is performed with an Adam optimizer and a learning rate of 0.01. One thousand epochs are employed. Figure (6.17) illustrates how the training process converges. The convergence is very smooth and the error vanishes to less than $10^{(} - 3)$. There is no overfitting as the training error is always less than the test error, while they continue to be close, following the same trend. After approximately epoch 250, the error starts to become quite minor.

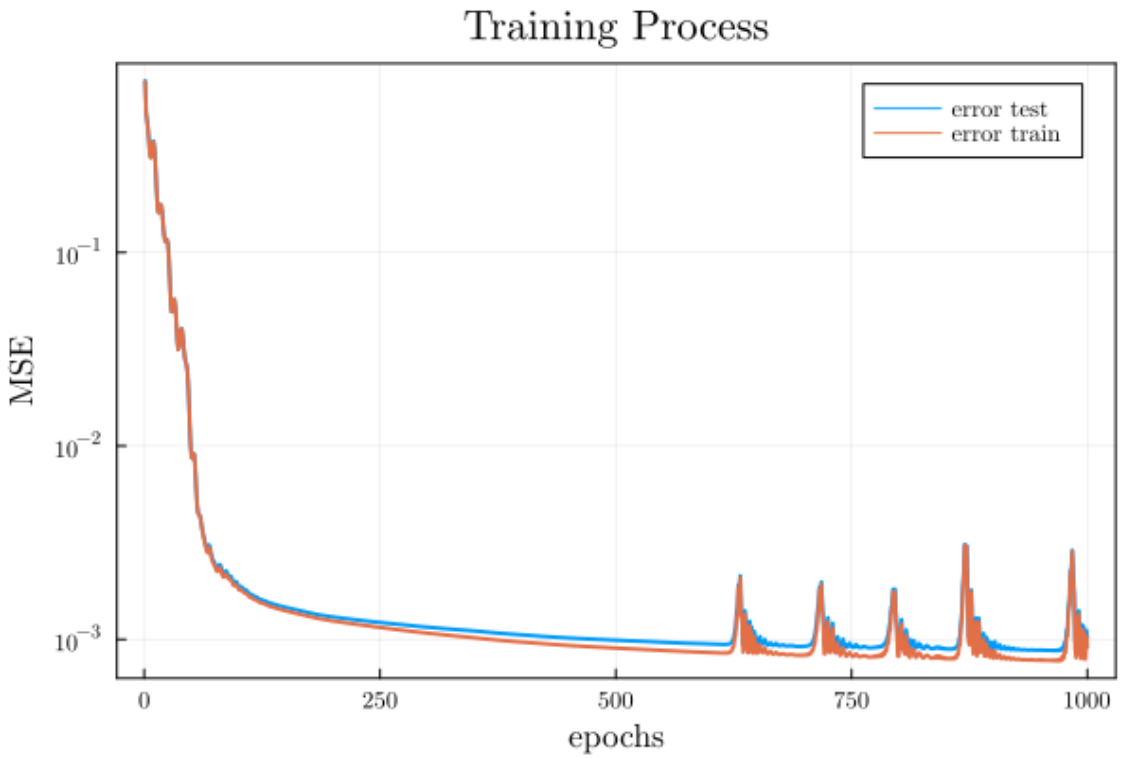

Figure 6.17: Overview of the Training Process of the Fuel Consumption model.

Optimizer	Adam
Learning Rate (η)	0.01
Epochs	1000
Trainable Parameters	0.004867
Timesteps	10
Rate	1Hz

Table 6.7: Fuel Consumption model main characteristics.

The training and test results are presented in table (6.8). On both the relevant test data and the training data, these are assessed as being incredibly accurate. As anticipated, the error is slightly bigger in the test data than it is in the training data.

Metric	**Value**
Train Comb. Accuracy (R^2)	0.9990
Test Comb. Accuracy (R^2)	0.9990
Mean Comb. Square Error in Train Dataset	0.0009156
Mean Comb. Square Error in Test Dataset	0.001018
Mean Comb. Absolute Error in Train Dataset	0.02086
Mean Comb. Absolute Error in Test Dataset	0.02167

Table 6.8: Training and Test results for the Fuel Consumption model.

In figure (6.18), the visualization of the predicted values and the real values for one of the data sets is presented. It is evident that the Fuel Consumption observer follows exactly the actual data without any lag or overshoot or undershoot. That is confirmed also in the

scatter plot, figure (6.20) where the vast majority of the data are in agreement with the actual, denoted by the straight line.

In figure (6.19) a zoomed region for a time interval of 100 seconds is presented. In this case, the model follows almost exactly all of the spikes of the real signal. The noisy behavior is limited.

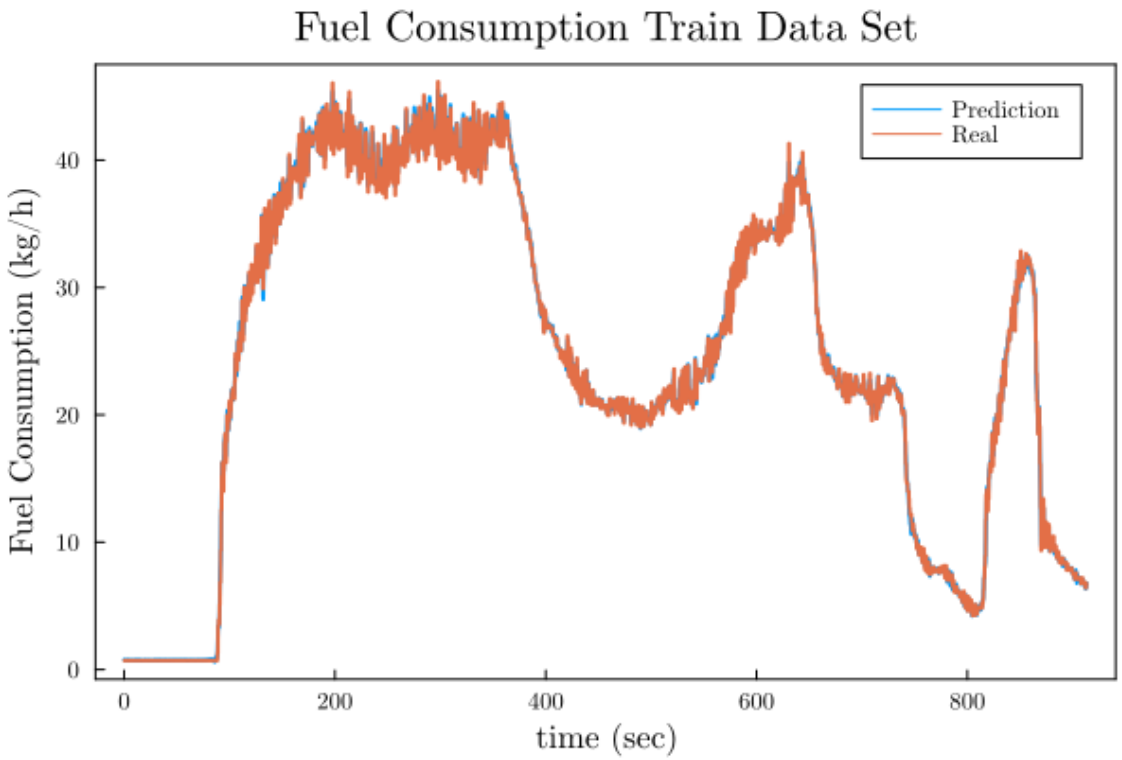

Figure 6.18: Visualization of the Fuel Consumption model performance on a training dataset.

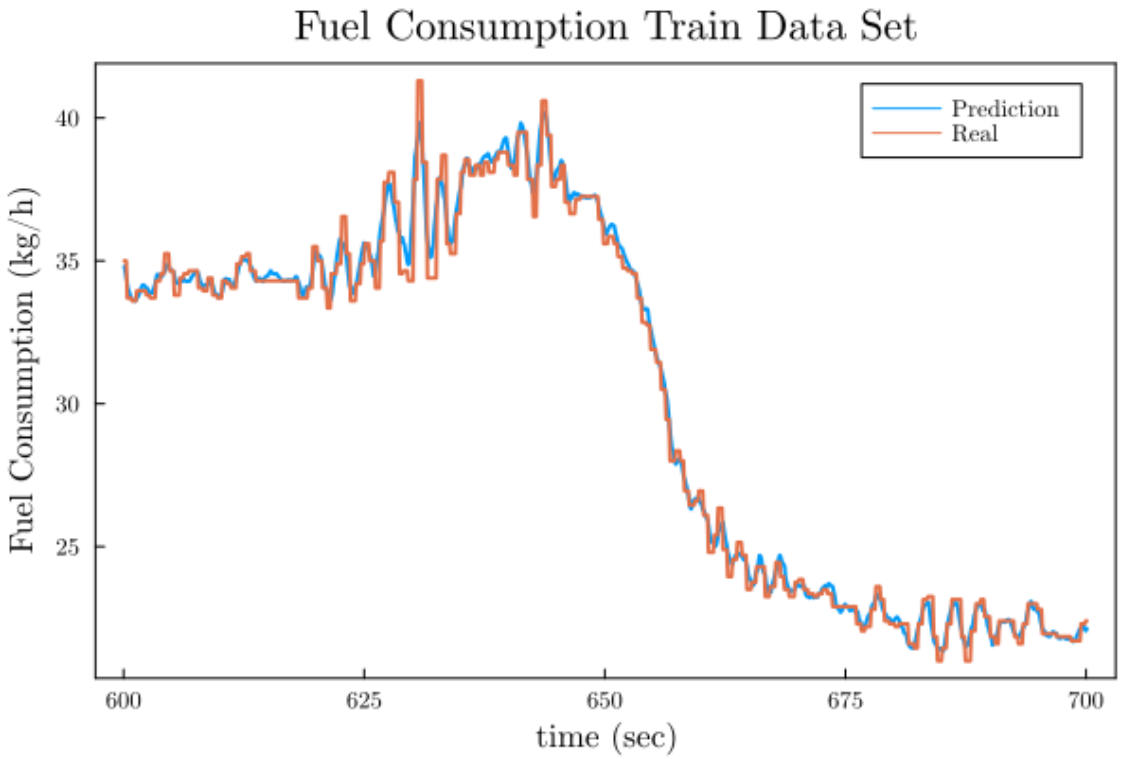

Figure 6.19: Zoom in the visualization of the Fuel Consumption model performance on a training dataset.

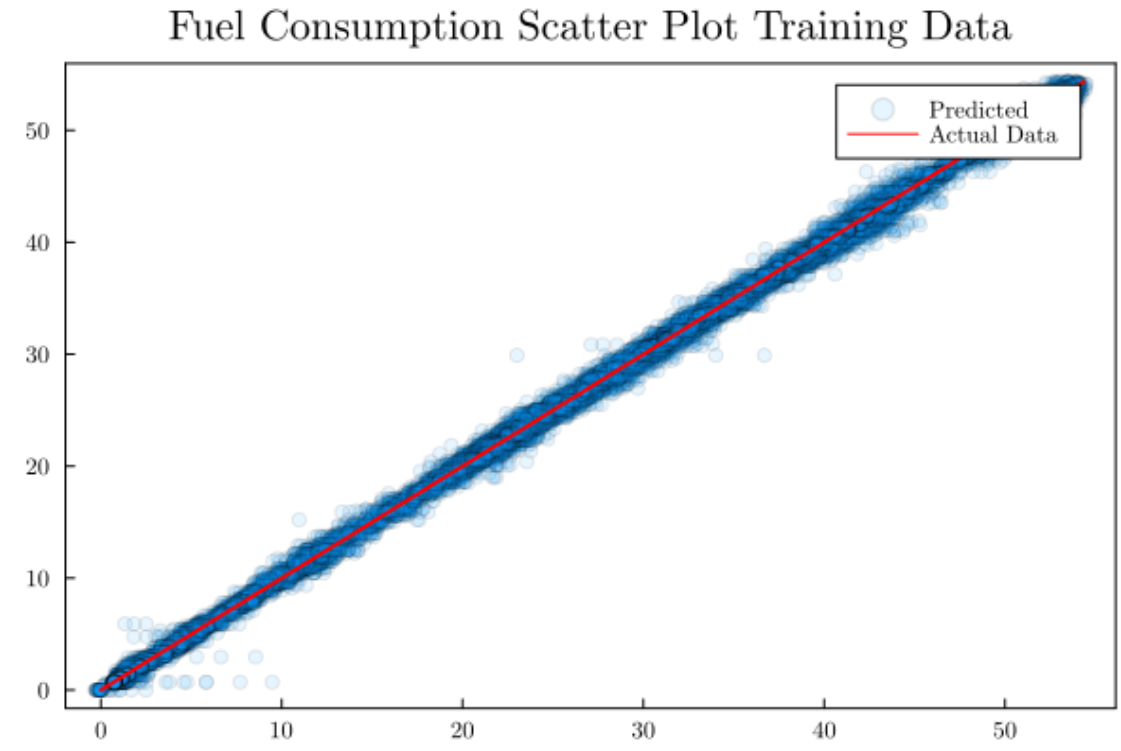

Figure 6.20: Scatter plot of the performance of the Fuel Consumption model on the Training and Test data.

6.4.2 Validation Results

The correctness of the model is evaluated on a validation data set, in the wake of the training and testing outcomes. The quantification of the model's correctness in relation to the validation dataset is presented in table (6.9).

The accomplished accuracy of the model on the validation data is very high showing a very good behavior of generalization in the context of the specific the engine different loading cycles. The validation combine accuracy is equal to that of training. It should be stated here though, that the Fuel Consumption signal is easier to predict compared to the previously described.

Metric	Value
Validation Comb. Accuracy (R^2)	0.9991
Mean Comb. Square Error in Validation Dataset	0.0006676
Mean Comb. Absolute Error in Validation Dataset	0.01825

Table 6.9: Validation results for the Fuel Consumption model.

In figure (6.21) the performance of the constructed model is pictured. The predicted values follow exactly the validation ones. Again no delay is present between the real signal and the predicted. No significant divergence is obderved in scatter plot, figure (6.23), for the whole range of values thath the parameter takes.

In the zoomed time interval between 600 seconds and 700 seconds, figure (6.22), it is evident that the values are extremely close. The trend of the real time series is accurately reproduced and the small discrepancies quickly corrected.

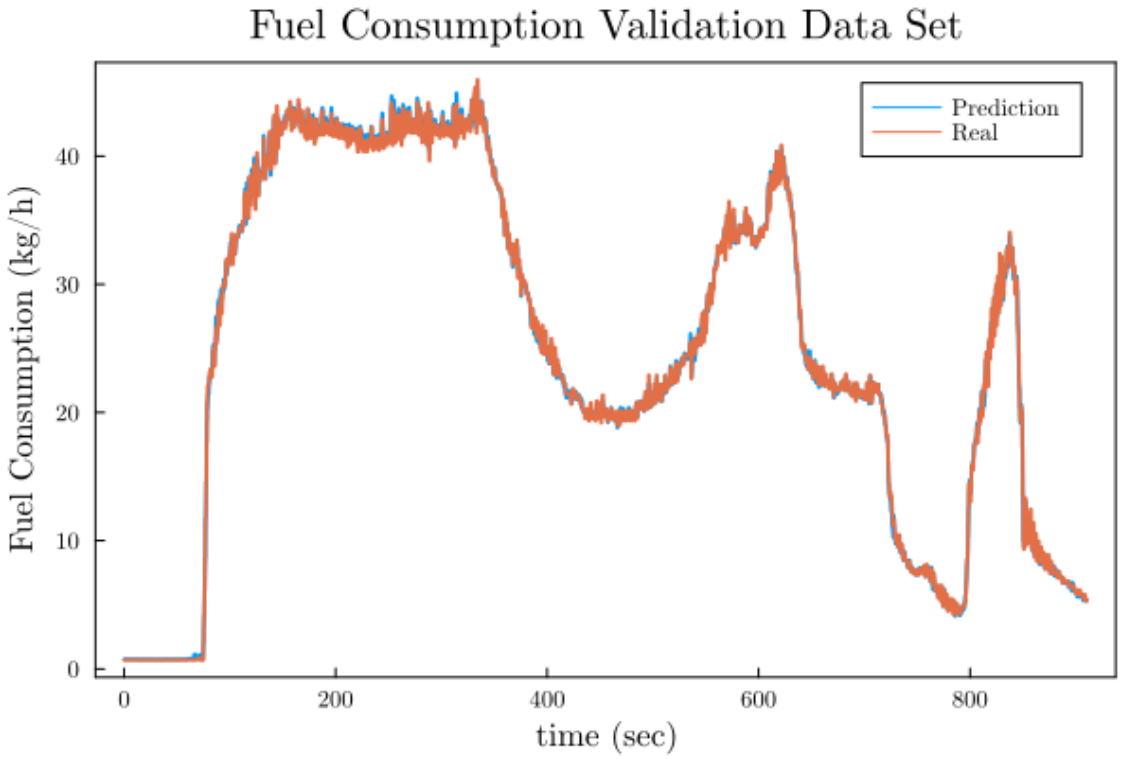

Figure 6.21: Visualization of the Fuel Consumption model performance on a validation dataset.

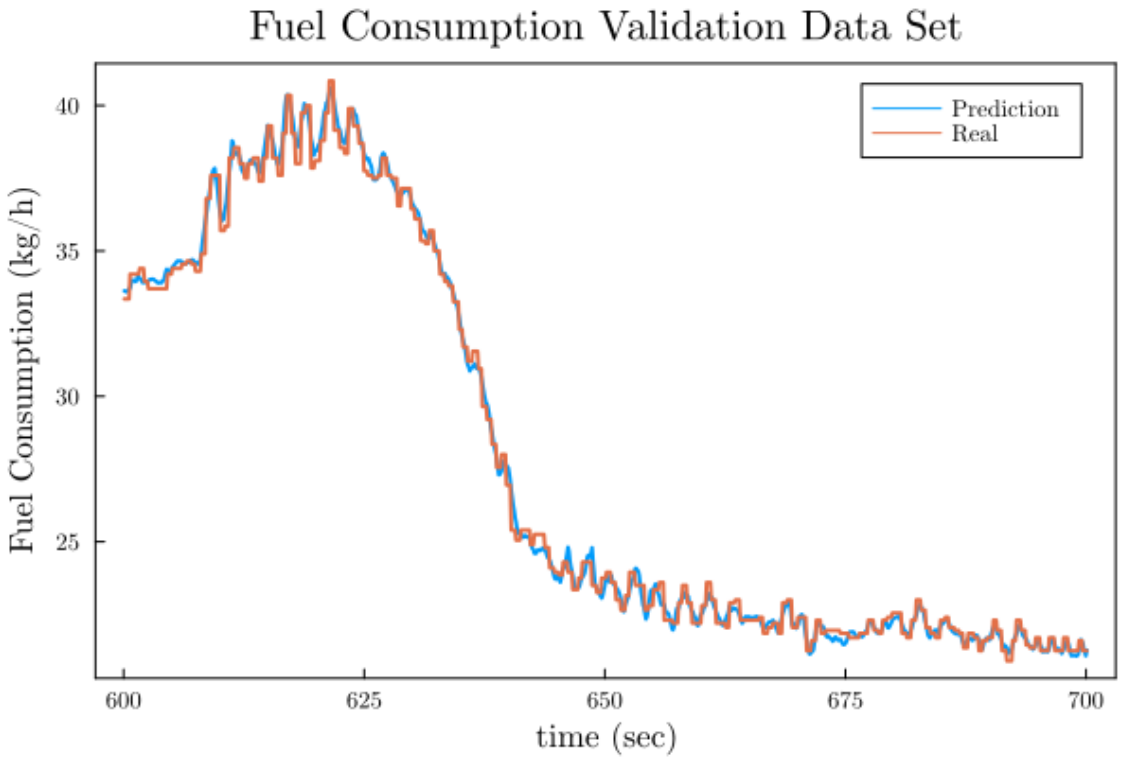

Figure 6.22: Zoom in the visualization of the Fuel Consumption model performance on a validation dataset.

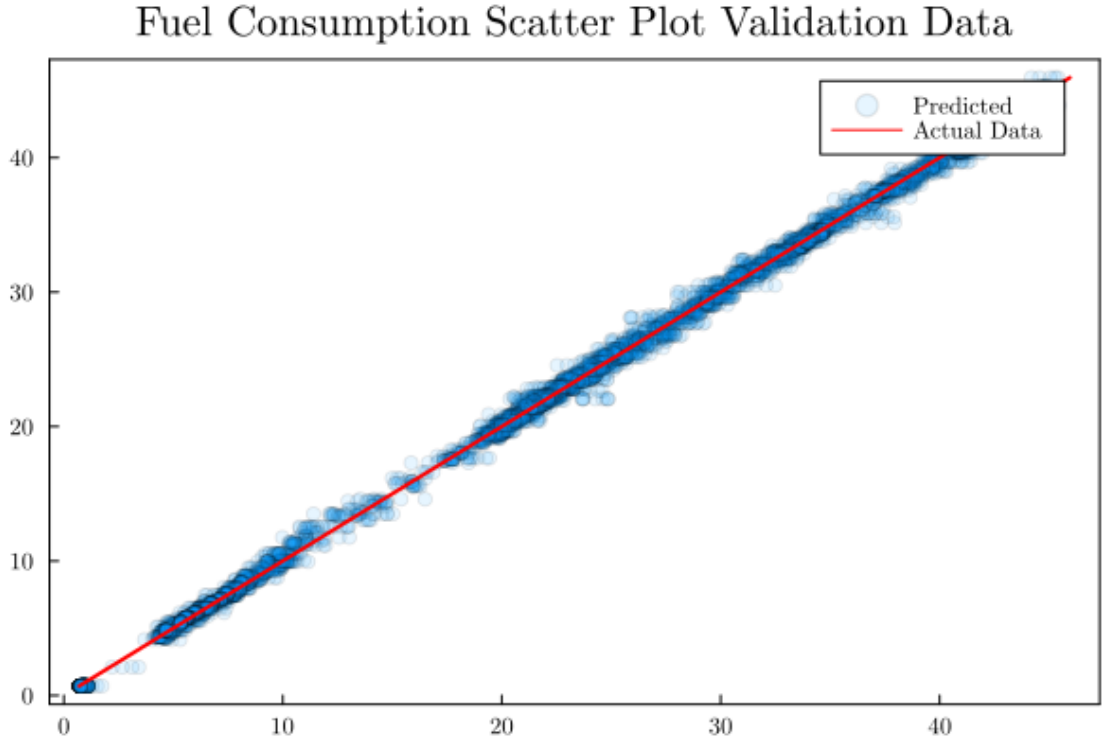

Figure 6.23: Scatter plot of the performance of the Fuel Consumption model on the Validation data.

In general, the model successfully completes its observational mission. A neural network is trained with the intention of making an accurate and timely prediction of the values of Fuel Consumption, while also being able to excel when faced with new data. Additionally, the model is considered representative of the engine and could be used for tasks that have to do with performance monitoring and predictive maintenance tasks.

6.5 Manifold Pressure Model

The manifold pressure model is designed to accept three inputs. These are the Exhaust Gas Mass Flow, the Engine Torque , and the Engine Speed. The values at ten time steps for each feature in a distinct channel—three in total for this one—are accepted as input by the model. The architecture used, is exactly the same as in the Fuel Consumption model.

This is a significant comparative advantage of these networks since it allows for continuous recycling by simply switching the input and output. This indicates that building a model is a quick and simple process that produces effective results very quickly. The aforementioned methodologies enable the introduction of *hardware-in-the-loop* concepts and enable quick product development.

6.5.1 Training Results

The training of the network, is performed with an Adam optimizer and a learning rate of 0.01. There are a thousand epochs used. The training process converges, as seen in Figure (6.17). The convergence is extremely smooth, and the error disappears to less than 10^{-3}. As long as, test error and train error, continue to be close and follow the same pattern, there is no overfitting because the training error is consistently lower than the test error.

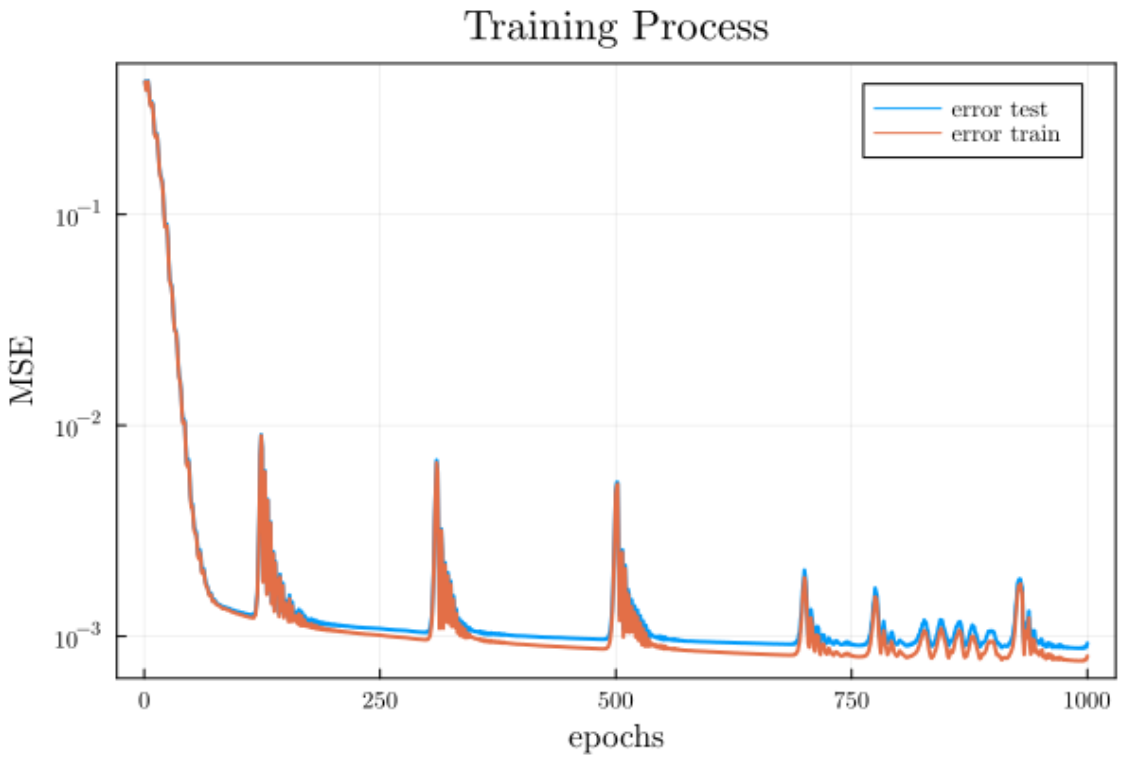

Figure 6.24: Overview of the Training Process of the Manifold Pressure model.

Optimizer	Adam
Learning Rate (η)	0.01
Epochs	1000
Training Time	83.83 s
Trainable Parameters	5416
Timesteps	10
Rate	1Hz

Table 6.10: Manifold Pressure model main characteristics.

Training and test results are shown in a table (6.11). These are evaluated as extremely accurate on both the relevant test data and the training data. The error is slightly larger in the test data than in the training data, as expected.

Metric	**Value**
Train Comb. Accuracy (R^2)	0.9992
Test Comb. Accuracy (R^2)	0.9991
Mean Comb. Square Error in Train Dataset	0.0008050
Mean Comb. Square Error in Test Dataset	0.00092721
Mean Comb. Absolute Error in Train Dataset	0.02136
Mean Comb. Absolute Error in Test Dataset	0.02248

Table 6.11: Training and Test results for the Manifold Pressure model.

Figure (6.25), depicts the visualization of predicted and actual values for one of the data sets. The Fuel Consumption observer clearly follows the actual data with no lag, overshoot, or undershoot. That is also confirmed by the scatter plot, figure (6.20) where the vast majority of the data agree with the actual, denoted by the straight line.

In figure (6.26) a zoomed region for a time interval of 100 seconds is ploted. In this case, the signal as seen is a stepping one, and the model is following the pattern but not the exact shape. Nevertheless, the absolute accuracy is really impressive.

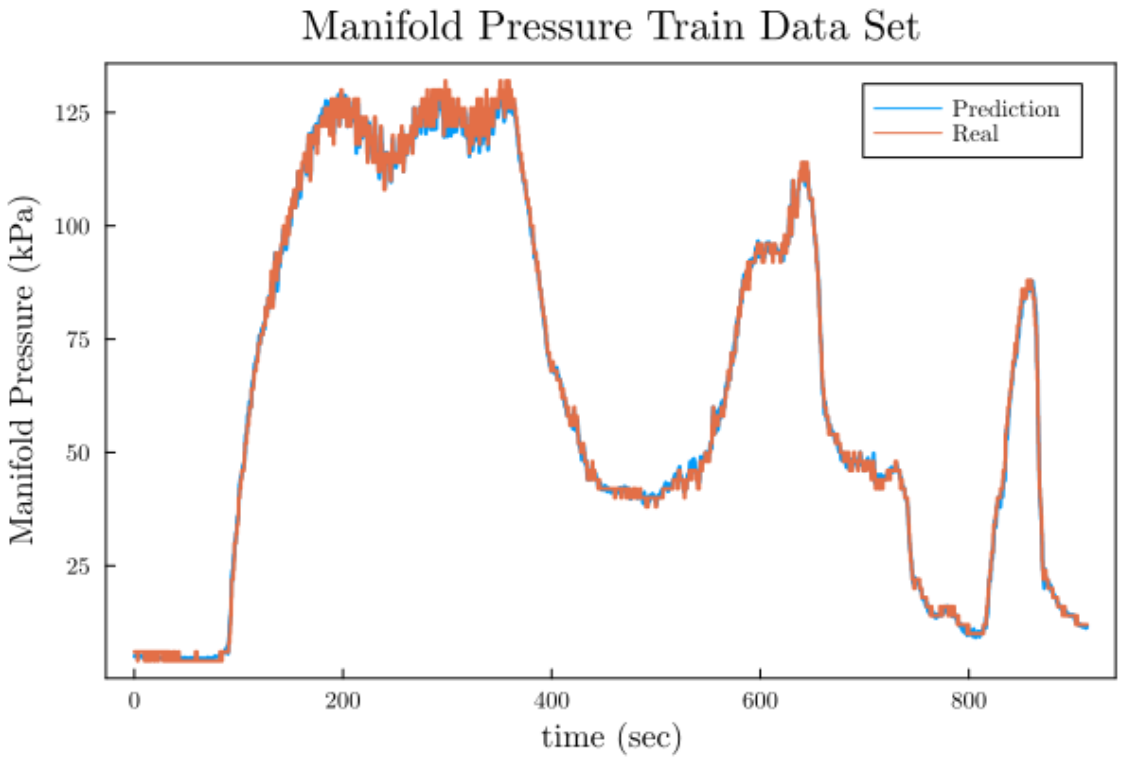

Figure 6.25: Visualization of the Manifold Pressure model performance on a training dataset.

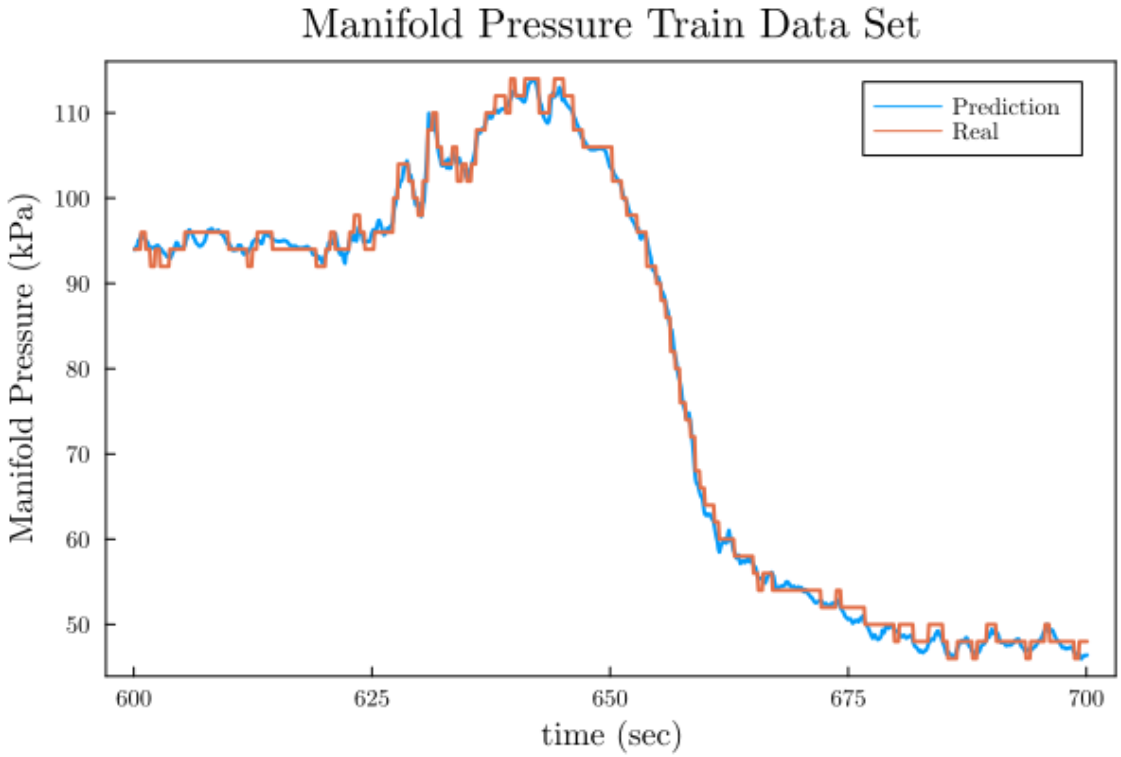

Figure 6.26: Zoom in the visualization of the Manifold Pressure model performance on a training dataset.

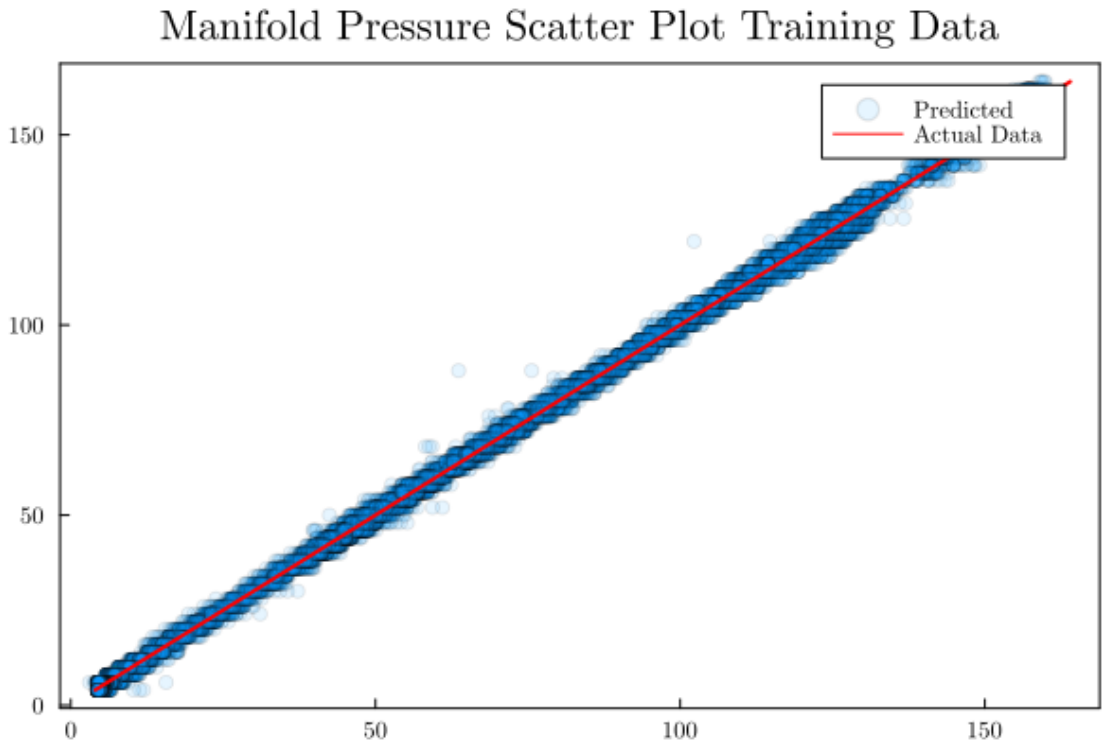

Figure 6.27: Scatter plot of the performance of the Manifold Pressure model on the Training and Test data.

6.5.2 Validation Results

The model's predictive characteristics are examined on a validation data set, after the results of the training and testing outcomes. The quantification of the model's in relation to the validation dataset is presented in table (6.12).

The achieved accuracy of the model on the validation data is very high, indicating a very good generalization behavior in the context of the specific engine different loading cycles. The validation combine accuracy is equal to that of training. It should be stated here though, that the Manifold Pressure signal is easy to predict in contrast to NO_X's and λ.

Metric	Value
Validation Comb. Accuracy (R^2)	0.9989
Mean Comb. Square Error in Validation Dataset	0.0008392
Mean Comb. Absolute Error in Validation Dataset	0.02237

Table 6.12: Validation results for the Manifold Pressure model.

In figure (6.28) the performance of the constructed model is pictured. The predicted values follow exactly the validation ones. Again no delay is present between the real signal and the predicted. No significant divergence is obderved in scatter plot, figure (6.30), for the whole range of values thath the parameter takes.

In the zoomed time interval between 600 seconds and 700 seconds, figure (6.29), it is evident that the values are extremely close. The trend of the real time series is accurately reproduced and the small discrepancies quickly corrected.

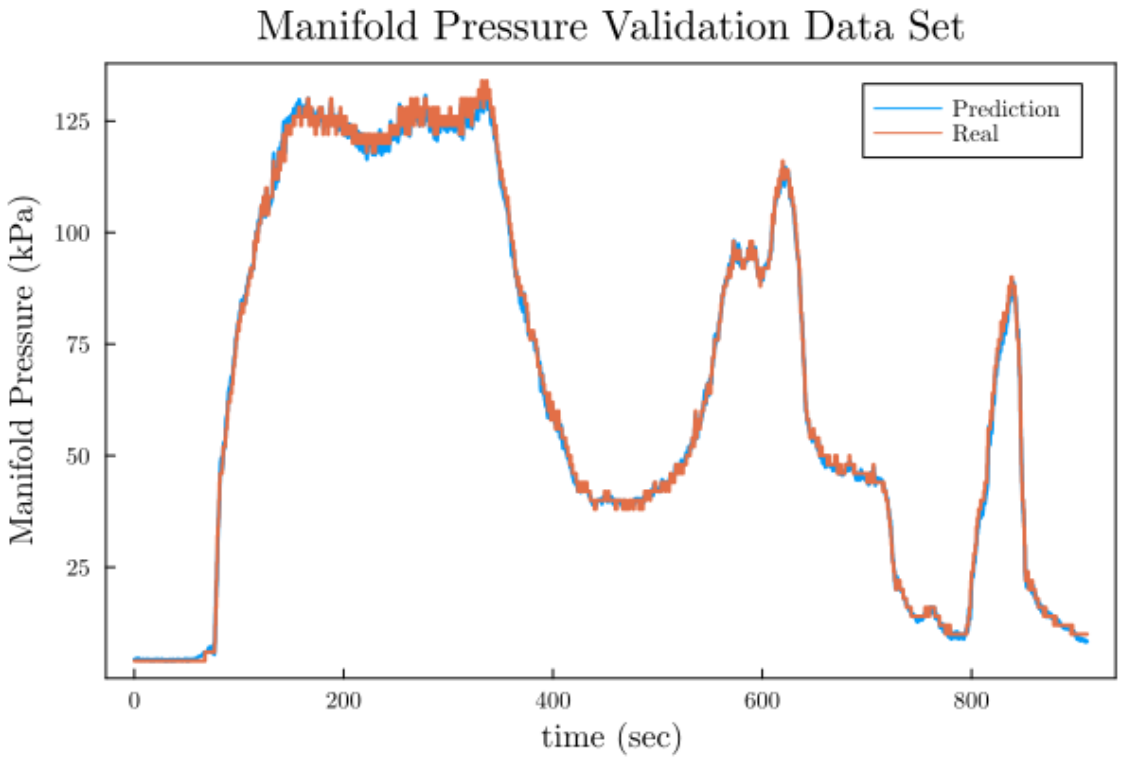

Figure 6.28: Visualization of the Manifold Pressure model performance on a validation dataset.

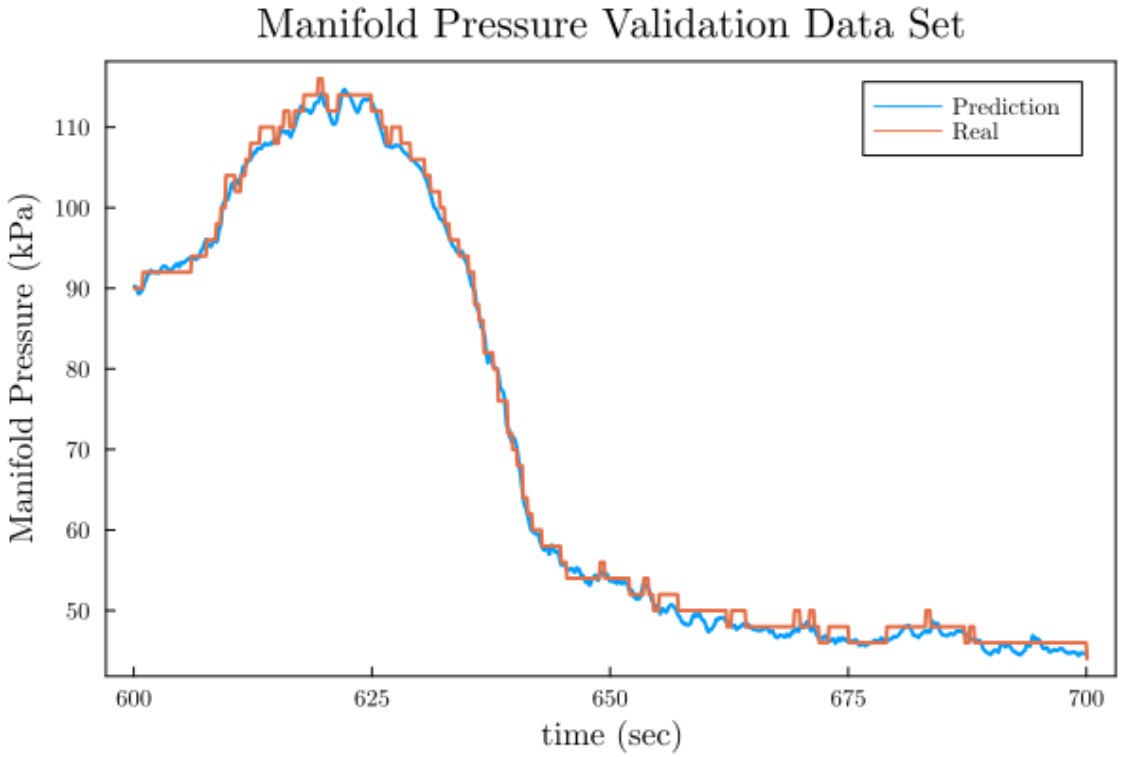

Figure 6.29: Zoom in the visualization of the Manifold Pressure model performance on a validation dataset.

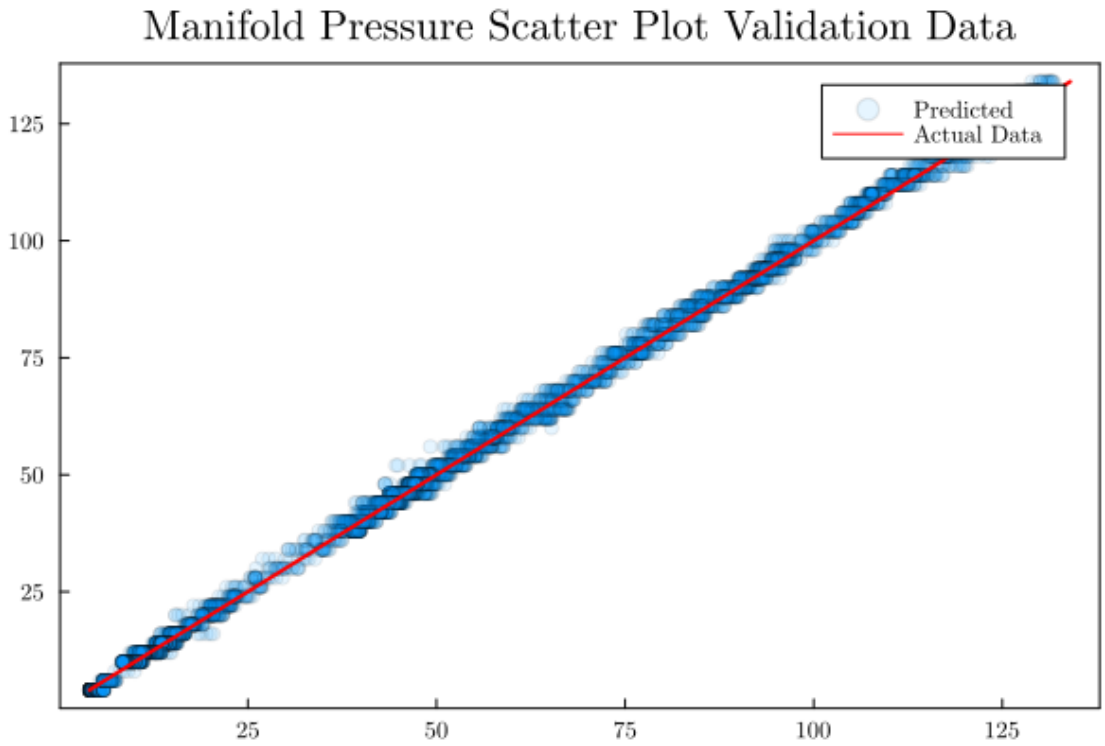

Figure 6.30: Scatter plot of the performance of the Manifold Pressure model on the Validation data.

In general, the model successfully completes its observational mission. A neural network is trained with the intention of making an accurate and timely prediction of the values of Fuel Consumption, while also being able to excel when faced with new data. Additionally, the model is considered representative of the engine and could be used for tasks that have to do with performance monitoring and predictive maintenance tasks.

Chapter 7

Conclusions and Future Work

Conclusions

In this thesis, the applicability of convolutional neural networks for the construction of virtual sensors for the HIPPO-2 engine was investigated. More specifically, models were developed for the prediction of the content of NO_X's, the λ of the engine, the Fuel Consumption as well as the Manifold Pressure.

Initially, a preprocessing of the data was materialized. Following that a statistical description of the data-set was done in order to asses the relationships between the available variables. Lastly, the candidate model architectures were presented. The multi channel one was picked, that offered data fusion later in the network.

The networks showcased superior performance regarding the accuracy capturing the physics in both the training and the validation data. In other words they were successful in mapping the inputs to the target output. Nevertheless, the predictions could be a bit smoother with less noise.

The embedding of the software was also successful. The interesting part was that using a, easy to program and fast, high level language, one can read the signals of the engine directly using a budget set up such as the Raspberry Pi and a Can-Bus. In the past, the cost of this functionality would be around 15,000 euros. Additionally, the calculations and predictions can occur seemingly fast.

Overall, the construction and training of the models was performed on Julia Programming Language. The design was easy, and the calculations really fast. Additionally, the gpu was used making the training times very small for the performance attained.

In conclusion, the virtual sensors constructed using convolutional neural networks seem very prominent for replacing traditional techniques such as engine maps. Additionally, new opportunities arise regarding the model predictive control, the monitoring of the system as well as the real time implementation of digital twins of the engines.

Future Work

This thesis, is part of an ongoing process of diploma thesis carried out at the Laboratory of Marine Engineering at the National Technical University of Athens. More precisely, these thesis are concerned with the test bed that the Lab has. As a result, each diploma thesis builds upon previously produced work and knowledge. In this scope some future work is proposed:

- Convolutional Neural Networks: Investigation of alternative and additional architectures of convolutional neural networks. The goal should be to minimize the noise of the predictive signal.
- Model Predictive Control: Use of the deep neural network models in the scope of model predictive control. They have the ability to substitute empirical models and increase the accuracy while decreasing the computational cost.
- Digital Twins: Encapsulation of the models on a digital twin of the engine, able to perform performance monitoring and predictive maintenance. Additionally, monitoring the engine deterioration and collecting data.
- Hardware in the Loop: Investigating the use of such models for the rapid prototyping of engines and the simulation of engines parameters during engine design.
- Raspberry Pi: Further investigation of the possibilities of integrating the Rapsberry Pi and the Can Bus in normal engine operation. Investigating also the possibility of sending signals and controlling the engine though it.
- Emissions: Construction of more sophisticated models for the real time prediction of all the emissions of the engine. Control of the engine based on cost functions that include the emissions in order to eliminate them as much as possible.

Bibliography

[1] Tuan Le Anh, Vinh Nguyen Duy, Ha Khuong Thi, and Hoi Nguyen Xa. Experimental investigation on establishing the hcci process fueled by n-heptane in a direct injection diesel engine at different compression ratios. *Sustainability*, 10(11):3878, 2018.

[2] Lino Guzzella and Christopher Onder. *Introduction to modeling and control of internal combustion engine systems*. Springer Science & Business Media, 2009.

[3] M Nφrgaard, O Rvn, NK Poulse, and LK Hansen. *Neural networks for modelling and control of dynamic systems*. Springer-Verlag London Limited. London. England, 2001.

[4] UNITED NATIONS. Review of maritime transport 2018, 2018.

[5] International Maritime Organization. Third imo ghg study 2014, 2015.

[6] Oleksiy Bondarenko and Tetsugo Fukuda. Development of a diesel engine's digital twin for predicting propulsion system dynamics. *Energy*, 196:117126, 2020.

[7] D Brand, C Onder, and L Guzzella. Virtual no sensor for spark-ignition engines. *International Journal of Engine Research*, 8(2):221–240, 2007.

[8] Frédéric Tschanz, Alois Amstutz, Christopher H Onder, and Lino Guzzella. A real-time soot model for emission control of a diesel engine. *IFAC proceedings volumes*, 43(7):222–227, 2010.

[9] Chris M Atkinson, Theresa W Long, and Emil L Hanzevack. Virtual sensing: a neural network-based intelligent performance and emissions prediction system for on-board diagnostics and engine control. *Progress in Technology*, 73(301-314):2–4, 1998.

[10] G De Nicolao, Riccardo Scattolini, and C Siviero. Modelling the volumetric efficiency of ic engines: parametric, non-parametric and neural techniques. *Control Engineering Practice*, 4(10):1405–1415, 1996.

[11] Ivan Arsie, Cesare Pianese, and Marco Sorrentino. Development of recurrent neural networks for virtual sensing of nox emissions in internal combustion engines. *SAE International Journal of Fuels and Lubricants*, 2(2):354–361, 2010.

[12] Ivan Arsie, Andrea Cricchio, Matteo De Cesare, Francesco Lazzarini, Cesare Pianese, and Marco Sorrentino. Neural network models for virtual sensing of nox emissions in automotive diesel engines with least square-based adaptation. *Control Engineering Practice*, 61:11–20, 2017.

[13] Nikolaos Planakis, George Papalambrou, Nikolaos Kyrtatos, and Pantelis Dimitrakopoulos. Recurrent and time-delay neural networks as virtual sensors for nox emissions in marine diesel powertrains. Technical report, SAE Technical Paper, 2021.

[14] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, 2017.

[15] Roni Mittelman. Time-series modeling with undecimated fully convolutional neural networks. *arXiv preprint arXiv:1508.00317*, 2015.

[16] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[17] Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiao Li Li, and Shonali Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[18] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008.

[19] Geoffrey E Hinton. How neural networks learn from experience. *Scientific American*, 267(3):144–151, 1992.

[20] Kevin Gurney. *An introduction to neural networks*. CRC press, 2018.

[21] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[22] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[23] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[24] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[25] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[26] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[27] Barry J Wythoff. Backpropagation neural networks: a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 18(2):115–155, 1993.

[28] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[29] John B Heywood. *Internal combustion engine fundamentals*. McGraw-Hill Education, 2018.

[30] Gamal Hassan Abd-Alla. Using exhaust gas recirculation in internal combustion engines: a review. *energy conversion and management*, 43(8):1027–1042, 2002.

[31] Texas Instruments Steve Corrigan. Introduction to the controller area network (can), 2002.

[32] Hassan Rafique, Mingrui Liu, Qihang Lin, and Tianbao Yang. Weakly-convex–concave min–max optimization: provable algorithms and applications in machine learning. *Optimization Methods and Software*, pages 1–35, 2021.

[33] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, 27(3):326–327, 1995.

[34] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[35] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[36] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J Inman. 1d convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, 151:107398, 2021.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.