MASTER'S THESIS | LUND UNIVERSITY 2018

Evaluation of Artificial Neural Networks for Predictive Maintenance

Anders Buhl, Hugo Hjertén

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2018-03

Evaluation of Artificial Neural Networks for Predictive Maintenance

(Feed Forward, Convolutional & LSTM Artificial Neural Networks for Predictive Time Series Classification of Machine Generated Data)

Anders Buhl
dat13abu@student.lu.se

Hugo Hjertén
elt12hhj@student.lu.se

June 8, 2018

Master's thesis work carried out at Jayway Malmö.

Supervisors: Andreas Kristiansson, andreas.kristiansson@jayway.com
Johan Lindén, johan.linden@jayway.com
Patrik Persson, patrik.persson@cs.lth.se

Examiner: Flavius Gruian, flavius.gruian@cs.lth.se

Abstract

This thesis explores Artificial Neural Networks (ANNs) for predictive time series classification for Predictive Maintenance (PdM). Time slicing and time shifting are methods used, to enable the models to find features over time, and to predict into the future, respectively. Architectures of increasing complexity are explored for Feed Forward Neural Networks (FFNNs), Convolutional Neural Networks (CNNs) & Long Short-Term Memory (LSTM) networks, of which the best performing are compared. CNNs & LSTM are found to perform better than FFNNs since they are designed to handle sequences of data. This research shows that a model with high accuracy might in fact be a bad model for PdM, especially when the data set is imbalanced. Additional metrics such as Confusion Matrices and Receiver Operating Characteristic (ROC)-curves are needed to evaluate models. This thesis shows that consistent, representative and a lot of data of good quality is needed for a well performing ANN. ANNs for PdM reduces the required domain knowledge, and perform well for common/frequent classes, but less so for the less frequent classes.

Keywords: Predictive Maintenance, Time Series Classification, Machine Learning, Artificial Neural Networks, FFNN, CNN, LSTM

Acknowledgements

We would like to extend our deepest appreciation to the people that helped us during this master thesis.

A special thanks to Mattias Ohlsson, researcher & senior lecturer at the Faculty of Science, Lunds University, for providing us with expertise whenever we felt lost in the jungle of Keras.

Thanks to Jon Bokrantz, PhD Student at the Department of Industrial and Materials Science, Division of Production Systems, Chalmers, for inspiring us and helping us understand how far the industry has come, and wants to come, concerning Industry 4.0 & Predictive Maintenance.

And finally thank you to Kai Goebel, Tech Area Lead at Discovery and Systems Health, Intelligent Systems Division, NASA Ames Research Center, for providing us with the data set, along with the previous research papers on the same data set.

Contents

1 Introduction **9**
1.1 Background 9
1.2 Research Question 10
1.3 Related Work 11
1.4 Contribution 13
1.5 Structure 13

2 Theory **15**
2.1 Predictive Maintenance 15
2.2 Data Type & Format 17
2.3 Artificial Neural Networks - Overview 20
2.4 Artificial Neural Networks - Types 23
2.5 ML Performance Evaluation 27

3 Method **31**
3.1 Data Preprocessing 31
3.2 Tool Selection 35
3.3 Framework 36
3.4 Type, Architecture & Parameters 37

4 Evaluation **39**
4.1 Results 39
4.2 Discussion 48

5 Conclusion **55**
5.1 Future Work 57

Bibliography **59**

Appendix A File formats **65**

Appendix B The development process 71

Acronyms

ANN	Artificial Neural Network.
CNN	Convolutional Neural Network.
FFNN	Feed Forward Neural Network.
LSTM	Long Short-Term Memory.
ML	Machine Learning.
PdM	Predictive Maintenance.
RNN	Recurrent Neural Network.
ROC	Receiver Operating Characteristic.

Chapter 1
Introduction

Machine Learning (ML) can be used to perform Predictive Maintenance (PdM) on machines. With the vast amount of time series data constantly being produced by machines in factories and plants, such as sensor and control values, there is a lot of information available to predict breakdowns of the machines.

This thesis will investigate how Artificial Neural Networks (ANN) can be used to perform PdM. More specifically, Feed Forward Neural Networks (FFNN), Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks will be focused on, to compare performance for predictive time series classification.

1.1 Background

Maintenance of industrial machines is important to maximize up-time and quality of production. Lack of maintenance will result in a decrease in production or reduced quality. Unnecessary maintenance leads to high maintenance costs which are both unnecessary economic and environmental costs.

Internet of Things (IoT) is increasingly used and allows devices to connect to networks and to send and exchange data. This technology is being spread to factories and industrial plants, where the first step is connecting machines and gathering data. Machine-generated data is collected and stored in the cloud to be utilized for visualization and analysis. Important aspects for the end-user is how this data can be utilized to improve functionality, visualization, and analysis to improve maintenance and operability of industrial plants. *Industry 4.0* is the term used when referring to the digitalization of factories and plants, of making them smarter and more autonomous.

Machine-generated data will generally look very different from factory to factory; there is no industry standard for the format of the data. Additionally, the regularity, the amount and the quality of the machine-generated data vary enormously. However, a common factor is that the data is sequential, since factories and industrial plants often are in operation

24/7, constantly producing data. The benefit of time series data is that it is possible to see changes in value over time.

Predictive Maintenance (PdM), a method for optimizing machine maintenance, is using time series machine data to predict the need for maintenance of connected machines. By connecting machines and thereby being able to remotely monitor and control machines, PdM can be performed. PdM has been discussed and researched for several years (e.g. [Dekker, 1996] and [Mobley, 2002]). The common idea is that by monitoring indicators of the operating condition of the machines, the information will be available for ensuring the maximum interval between repairs and minimize the number and cost of unscheduled outages caused by machine failure [Mobley, 2002].

Machine Learning (ML), and more specifically Artificial Neural Networks (ANNs), has seen a major upswing in popularity and usage across many industries. The breakthrough has come with the increased computational power of modern computers, that can handle massive amounts of data to construct advanced, and often efficient ANN's that can find relations in large data sets that otherwise have been hard to find. Since machine-generated data sets are very large, it becomes interesting to use ANN's to perform PdM.

The amount of available machine-generated data will continue to increase and is a prerequisite for predictive maintenance. Research have been done in using ML for PdM [Cline et al., 2017; Wu et al., 2018; Singh, 2017; Xiao, 2015; Kim et al., 2015; Xie et al., 2015]. This thesis will research how ANN can be used to implement PdM in the machine industry.

1.2 Research Question

The main focus for this thesis is on **how ANNs can be used for predictive time series classification to facilitate PdM**. Some alternatives to ANNs for PdM, that previously have been investigated, is mentioned in section 1.3. More specifically a few questions are asked, which this thesis aims to investigate.

- How can ANNs handle time series/continuous data, and how good are they at finding features over time?
- How can ANN models for PdM be evaluated and compared?
- What are the benefits & limitations of using ANN models for PdM?

1.2.1 Data Set

The data set used for research in this thesis is a data set from the Data Challenge at the Annual Conference of the Prognostics and Health Management Society 2015 [Prognostics and Health Management Society, 2015]. The data set is divided into three parts containing different kind of samples from a plant. Part one and two are the input data consisting of sensor values from different parts of the plant at certain time stamps. The third part is the target data and consists of time intervals with a specific error code. The data sets contain different problems which can occur in machine-generated data sets. These problems include missing values when sensor values are missing for a timestamp and imbalanced data when one or more target classes (or error codes) are under-represented.

This data set is chosen because it contains many of the attributes that can be expected when working with PdM. Previous research has been done on this data set using other techniques than ANNs which makes this data set an even better choice. The purpose of the ML model is to classify time sequences, where each class corresponds to a certain error code or state.

1.3 Related Work

There is a lot of research that has been done on ML and ANNs, especially in the recent years. PdM has also been investigated, with different interpretations of what PdM is, as well as how it can be done. However, the investigation of using ANN for predictive time series classification for doing PdM is limited.

1.3.1 Machine Learning for Predictive Maintenance

[Cline et al., 2017] cover machine learning for predictive maintenance. Cline et al. discuss how ML analysis can be used to improve maintenance by identifying failures and improving preventive maintenance. A real-world use-case on a gas & oil drilling plant, where historic data have been gathered for a long time, is used to conduct experiments. In the paper, they were able to increase the failure detection from roughly 1% to 61%. It is concluded that the presented model resulted in a major improvement compared to the current system.

LSTM have been used in previous research when predicting Remaining Useful Life (RUL) for turbofan engines [Wu et al., 2018]. In this case, LSTM networks are applied to a regression problem when calculating RUL. Although the method is applied only to one type of use case, Wu et al shows that the model is general enough to be applied to different use cases which predict condition based on sensor values. Vanilla LSTM resulted in increased performance compared to Recurrent Neural Networks (RNN) and Gated Recurrent Units [Wu et al., 2018]. From the RNN architectures mentioned above, this thesis will use LSTM.

1.3.2 LSTM for anomaly detection

[Singh, 2017] has used an unsupervised approach in using a Long Short-Term Memory (LSTM) network for anomaly detection in sequential time series data. One approach is to predict a sequence of data and compare the predicted sequence of data with a real sequence of data. A LSTM model is trained and validated using normal data to be able to predict a normal sequence q time steps ahead. After training on normal data, a different data set containing anomalies is used and an anomaly score is calculated using a threshold on prediction density on a maximum likelihood estimation. The threshold is set to minimize the number of false positives. Keras with Tensorflow as backend was used for evaluation (see section 3.2 for more about these tools). The author concluded that how hyperparameters should be used and set in a network depends on the data set. Three different data sets with different characteristics were used and each used different hyperparameters and settings in the LSTM layers. A simpler approach using less complex networks was used and the author suggests researching simpler models before using LSTM nodes. The reason is that it is hard to understand more complex networks and they are thereby harder to configure and optimize. LSTM networks will only have an advantage if dependencies in the data set are complex and long-term.

Initial research in this thesis will focus on simpler models and extend models to include more complex nodes, e.g. LSTM. However, this thesis will not include unsupervised learning and will not train on a normal sequence to be able to recognize anomalies, since the data set used in this thesis contains output values for each set of input signals. Instead, the models will be trained to predict all kinds of states, i.e. normal and faulty states.

1.3.3 PHM 2015

The data set used in the thesis was part of a competition and efforts made in the competition PHM 2015, Prognostics and Health Management Society [2015], is documented through multiple papers [Xiao, 2015; Kim et al., 2015; Xie et al., 2015].

[Xiao, 2015] investigates ML techniques to model each plant separately and predict the start and end time of time intervals for error codes. The investigated ML techniques include K-nearest neighbor, Naive Bayes, Logistic Regression, Random Forest and Gradient Boosting. The results were promising and demonstrated the usefulness of data-driven ANNs, such as Convolution Neural Networks. However, ANNs were left as future work [Xiao, 2015].

[Kim et al., 2015] suggest an approach to first construct a physical representation of the data to be able to extract features from the input dimension. The features were used to recover and predict fault logs using a Fisher Discriminant Analysis (FDA). The FDA outperformed both Support Vector Machine and K-nearest neighbor.

[Xie et al., 2015] approach the problem by first cleansing the data and aligning the data by time stamps, followed by conducting feature extraction combining different features into feature vectors. The feature vectors are used with Random Forest and Gradient Boosting to classify error codes. Random Forest performed better than Gradient Boosting in most cases.

The three papers propose using ML to solve the PdM problem. Neural networks were not an approach included in any of the papers, but it was suggested as future work.

1.3.4 Current Products and Services on the Market

At the writing of this thesis project, PdM is a very hot topic in the machine industry. Most companies wish to be a part of the technological advancement, to embrace the *Industry 4.0*, and many companies have taken initial steps. The first steps include connecting machines and collecting data to cloud services (classic IoT ideas). However, there are a few companies that currently offer concrete PdM solutions[1].

[1]The following companies are just a selection of what is available on the market. The mentioned companies were approached at Underhåll (Europe's 1. Maintenance Trade Fair and Conference, `https://en.underhall.se/`) on 13 March 2018.

For instance Sigma IT Consulting offer a service which they call *Asset by Sigma*. It is marketed as a PdM service, that has been implemented for a few customers in small-scale scenarios [Sigma IT Consulting, 2018]. The implemented cases have so far been a variant of PdM: by examining how much different sensor values deviate from the expected normally distributed values, they provide a health status out of 100%. When the health status is below a certain threshold level, operators are notified that maintenance is needed. By definition this is not a real *prediction*, since a predicted time of break down (or similar) is not provided, but rather a current health status. This is still very useful, and the tool is also supposed to have support for prediction. The algorithms in *Asset by Sigma* are based on ML.

Vikon offers a product to facilitate maintenance of machines - condition based monitoring [Vikon, 2018]. By monitoring vibrations of different frequencies in different parts of a machine, and comparing with expected values for respective component in data sets which they have gathered over the past 20 years, they provide individual health statuses of different components as well as the machine as a whole. This is quite similar to that of Sigma, but the big difference is that Vikon has a more niche market focusing on vibration. Unlike Sigma, Vikon successfully reuses data from other applications for new areas of applications.

RS Production is a product by [Good Solutions, 2018] that helps to optimize the Overall Equipment Efficiency (OEE) for its customers. Good Solution's tool facilitates the gathering and visualization of data, and is an example of many companies that have taken big steps towards Industry 4.0, but that have not implemented PdM solutions as of yet.

1.4 Contribution

ML, especially ANNs, is gaining popularity and usage in the industry as well as academically, and is hence experiencing a lot of development. Though there is research on applying ML in many areas, there is a limited amount of research that has been done on how to apply ANNs for PdM. This thesis will help future research by showing that numerous evaluation techniques are necessary to effectively evaluate the success of a model for PdM, as well as by demonstrating techniques for finding features over time (time slicing) and for predicting into the future (time shifting).

1.5 Structure

This report is structured as follows. In chapter 2 the underlying theory for this thesis project is laid out. PdM and the data requirements needed for PdM is presented in this chapter followed by theory covering Artificial Neural Networks. This chapter also discusses different ways to evaluate the performance of ML models.

Chapter 3 explains the different parts of the method used in the thesis, covering the steps that are taken for preprocessing the data set, the tools that are used in this thesis, followed by a presentation of the custom framework constructed. This chapter also presents the different specific architectures and hyper-parameters of ANNs that are evaluated.

In chapter 4 the results are presented, followed by a discussion and a conclusion.

Chapter 2
Theory

This chapter presents the concept of PdM in section 2.1 and present the type and format of data applicable for PdM in section 2.2. Section 2.3 will explain ANNs as well as some essential ML theory, followed by a presentation of some ANN variants in section 2.4. Finally different ways to evaluate the performance of ML models is covered in section 2.5.

2.1 Predictive Maintenance

The area of Predictive Maintenance (PdM) has been discussed and researched since the 90's (e.g. [Dekker, 1996] and [Mobley, 2002]). The common idea is that by monitoring indicators of the operating condition of the machines, the data will be available to ensure the maximum interval between repairs and minimize the number and cost of unscheduled outages caused by machine failure [Mobley, 2002]. However, Mobley claims that PdM is more than that; it is a philosophy that uses the actual operating condition of plant equipment and systems to optimize total plant operation.

Maintenance in the industry can be either preventive or corrective maintenance. *Corrective* maintenance is performed to correct errors after breakdowns or wear. *Preventive* maintenance is anything done preemptively to avoid failures, breakdowns and unnecessary wear. The simplest kind of preventive maintenance is time-based maintenance where parts are replaced or repaired according to a predetermined schedule. This can result in replacing and repairing parts and machines that are in fact not in need of replacement or repair, which increases cost and environmental impact. A more advanced type of preventive maintenance is state-based maintenance. State-based maintenance is conducted according to the state of the machine instead of a predetermined schedule and can be divided into two main parts; *diagnostics* and *prognostics*. In diagnostics, the sensor data is interpreted and used to estimate the state of a machine, and in prognostics, the sensor data is used to predict when a mechanical failure or breakdown will occur. One technique for prognostic maintenance is Predictive Maintenance (see figure 2.1).

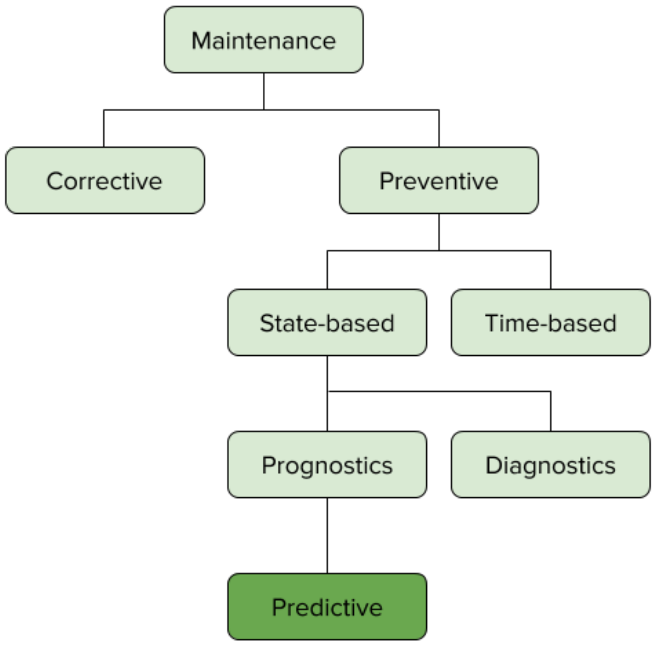


Figure 2.1: A presentation of different aspects of maintenance

PdM can be model-based, where a model of a machine is constructed and used to interpret sensor data and to predict the state of the machine. Different approaches can be used for system modeling. Some approaches require a domain expert to construct digital models for all processes in a system (examples of parameters derived from system models can be seen in table 2.1). ML for system modeling requires less domain knowledge since the models are trained to simulate a certain system instead of needing domain experts to define system models. In this thesis ML models will be trained to model a plant based on control signals and sensor values and thereby predict future error codes.

The overall objective with PdM is to predict future break-downs and service needs early enough for the problem to be fixed. For smaller issues that are easy to adjust, a short prediction period is required. When the issue concerns bigger parts that might have to be replaced, the prediction period needs to be longer. Depending on the data and application, different period lengths might be applicable.

Vibration Monitoring	analyze the vibration energy in the range 1-30,000 Hz.
Thermography	monitor the heat (IR) produced by the machines.
Tribology	lubricating oil analysis and wear particle analysis.
Ultrasonics	monitor noise frequencies above 30 kHz.
Visual Inspection	

Table 2.1: Typical techniques for PdM [Mobley, 2002]

2.1.1 Benchmark

An easy approach to evaluating a PdM program's success is to look at the cost and productivity benefits compared to the previous maintenance program. Predicting 50% of all failures with a PdM solution, compared to predicting 15% with a previous maintenance program, is a massive improvement. However, even with an obvious increase in profitability, the industry today is hard-convinced with a solution that doesn't offer near to 100% prediction. Therefore it is important to be able to benchmark models and compare their performance to decide which is better. As of today, there is no official way to benchmark the success of a PdM program or PdM algorithm.

Efforts are being made to construct an evaluation tool of maintenance, *Smart Assessment Maintenance (SMASh)*. SMASh is an initiative to define what Smart Maintenance is and to construct an evaluation too [Bokrantz et al., 2017]. Smart Maintenance focuses on the organizational structure of maintenance, from the competence of maintenance engineers to the technical solutions of being able to perform Smart Maintenance. The technical part is based on data-driven decisions and can be used to either augment human decision making or automate parts of the maintenance workflow. Prerequisites for data-driven decisions are the gathering of data, connecting of machines and gathering and storing of control signals and sensor data. PdM is part of the data-driven decision making procedure and is one step to augment or automate decisions.

2.2 Data Type & Format

In modern factories and plants, massive amounts of data is produced around the clock. In a typical production line, there are multiple machines that are synchronized and working together. Which data is logged and/or available in such a system (or a single machine) varies greatly; sensor values, states, error codes, calculations, images etc. Additionally, the format, size and consistency of the data is also very varied. However, an attribute the data commonly shares is that it is sequential; the data is sampled at a regular time interval (see section 2.2.2). It is possible to see changes in data values over time.

2.2.1 Data for Predictive Maintenance

To successfully perform PdM (see section 2.1) the data must contain certain information. Ideally it should contain target values that represent machine wear and/or breakdown. Realistically the machine-generated data will not contain this in plain sight, or not at all. Often the data needs to be pre-processed; filtered and/or transformed.

A typical scenario at a factory/plant would be to have a large data set; sensor values and other variables logged at a continuous interval. Hopefully some of this data represents one, or several, of the following information which can be used for target values:

- Healthy state vs error state
- Error codes
- Machine and/or sensor errors
- Wear and tear

Extensive domain knowledge is needed to successfully be able to interpret the data correctly, to extract the relevant target data. What information is best used for PdM, and how it is to be extracted, will vary from data set to data set, hence specific domain knowledge about the data is essential. Other issues that may arise with data sets include:

- Missing values; inconsistency in data
- Imbalanced data (certain outputs are over/under-represented)
- Lack of data for certain categories/situations

2.2.2 Time Series Data

All the data sets within the scope of this thesis are time sequential. In its simplest form, a time series data set can be split up into input data, $\mathbf{x}$, and output data, $\mathbf{d}$. The output is what a ML model is supposed to output for the corresponding input.

For instance, the input data can be in the form $[\mathbf{x}_0, \cdots \mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \cdots \mathbf{x}_N]$ where $i \in [0, N]$, $\mathbf{x}_i$ is an array of input values (of a constant length $n + 1$) at time i, and $N + 1$ is the number of measurements in the time sequence (see figure 2.2). This matches to output data in the form $[\mathbf{d}_0, \cdots \mathbf{d}_{i-1}, \mathbf{d}_i, \mathbf{d}_{i+1}, \cdots \mathbf{d}_N]$, where $\mathbf{d}_i$ is an array of output values (of a constant length $m + 1$) at time i (see figure 2.3). Sequential data can be processed in *time windows* of size smaller than or equal to the time sequence.

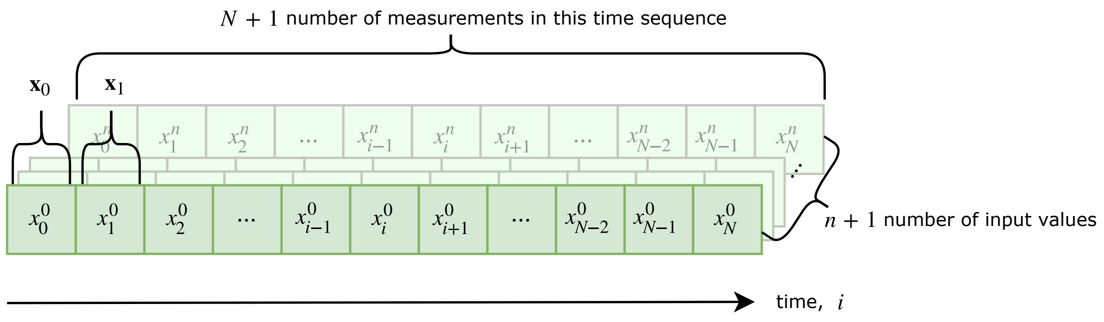

$N + 1$ number of measurements in this time sequence
$\mathbf{x}_0$
$\mathbf{x}_1$
x_0^n x_1^n x_2^n … x_{i-1}^n x_i^n x_{i+1}^n … x_{N-2}^n x_{N-1}^n x_N^n
x_0^0 x_1^0 x_2^0 … x_{i-1}^0 x_i^0 x_{i+1}^0 … x_{N-2}^0 x_{N-1}^0 x_N^0
$n + 1$ number of input values
time, i

Figure 2.2: Overview of the input data $\mathbf{x}_i$ over time. The time sequence is $N + 1$ long, and each $\mathbf{x}_i$ has $n + 1$ values.

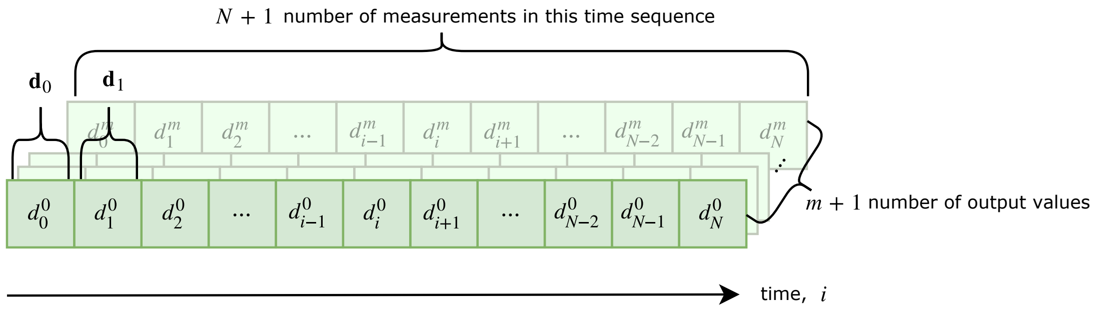

$N + 1$ number of measurements in this time sequence
$\mathbf{d}_0$
$\mathbf{d}_1$
d_0^m d_1^m d_2^m … d_{i-1}^m d_i^m d_{i+1}^m … d_{N-2}^m d_{N-1}^m d_N^m
d_0^0 d_1^0 d_2^0 … d_{i-1}^0 d_i^0 d_{i+1}^0 … d_{N-2}^0 d_{N-1}^0 d_N^0
$m + 1$ number of output values
time, i

Figure 2.3: Overview of the output data $\mathbf{d}_i$ over time. The time sequence is $N + 1$ long, and each $\mathbf{d}_i$ has $m + 1$ values.

The length $n + 1$ of $\mathbf{x}_i$, corresponds to i.e. the number of logged sensor values at each time i. Similarly, the length $m + 1$ of $\mathbf{d}_i$ depends on how many classes there are if it is a classification problem, or the number of values predicted if it is a regression problem.

A data set often contains multiple independent time sequences, of varying lengths $N + 1$. This is fine, as long as n and m are constant.

The output for a classification problem is defined as a binary value: 1 if the input is a positive match to that output class, or 0 if the input is not a positive match to the output class. The output for a multi-class classification problem is defined as binary array of size $m + 1$. For example if index 0 has the value 1 and the rest have 0 (i.e. [1, 0, 0]), the input is a positive match to class 0. See section 2.3.2 for more about classification problems.

2.2.3 Data Set Chosen

To answer the research questions (see section 1.2), a data set was needed that fulfilled as many criteria as possible as explained in section 2.2.1. The data set chosen for this thesis has a large number of input data, a fixed number of output data that represents different states (one healthy state and several error states), and it is time sequential. Most importantly, there over hundred-thousand data samples available in the data set, with a very frequent, and generally consistent, amount of logging. Research has previously been done on the PHM 2015 data set (see section 1.3.3), with a similar goal of performing PdM - it is possible to perform predictive time-series classification with this data set.

The domain knowledge for this data set is very limited. The first file describes sensor values and control values from different components within a plant. The number of components in a plant varies from 3 to 20 (see appendix A.1). The second file describes measurements from different zones in the plant and each zone has two measured values (see appendix A.2). For both types of input files, the sampling frequency is 15 minutes, but the samples from the different components and, zones within the same time group are not sampled at the same timestamp. The last file contain the targets and represent error codes for a given time interval (see appendix A.3). In the description of the data set only error codes 1-5 are considered of interest. It is stated that error codes are independent of sensor data outside of three hours [Prognostics and Health Management Society, 2015].

The different files do not have the same start and end time. This is a problem because all inputs are needed for each timestamp and each timestamp for the input data have to be matched with targets. Also, not all timestamps within the interval are represented in each file. Both of these problems are referred to as missing values and are common problems within ML and is also expected for machine-generated data. The number of missing values is small ($\lesssim$ 1%), and some missing samples may be from the same timestamp. Figure 2.4 shows the number of missing values of different error codes for one component. There are many techniques how to handle missing values (see section 3.1.1 for more information).

Figure 2.5 shows that the data set is heavily imbalanced; it is clear that error code 0 is the dominating error code. This is because this is the "healthy" state, when there is no error state of consequence. Error code 6 is the second most common error code, but according to the description, this error code does not represent any faults of interest. Error 4 is almost completely absent only occurring 0.076% of the time samples, which is normal for PdM. This is a problem when training a network to detect these uncommon error classes.

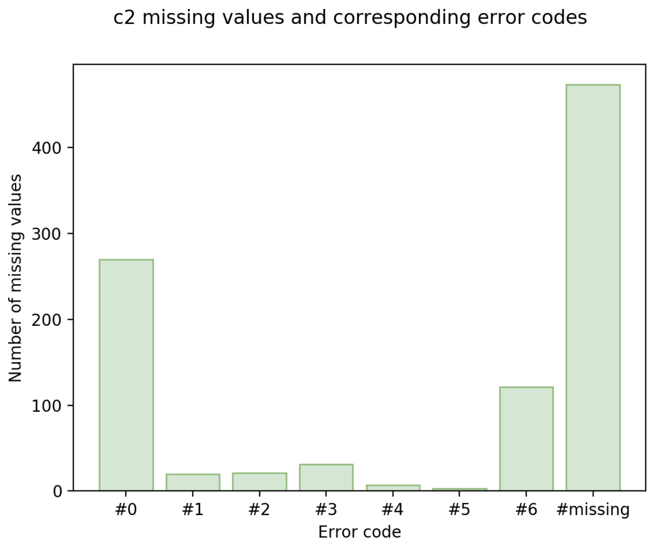


Figure 2.4: Number of missing values for each corresponding error code (#0-#6) and total number of missing values for one component (#missing). #0 represents number of missing timestamps with error code 0. The sample is from plant 1, component 2.

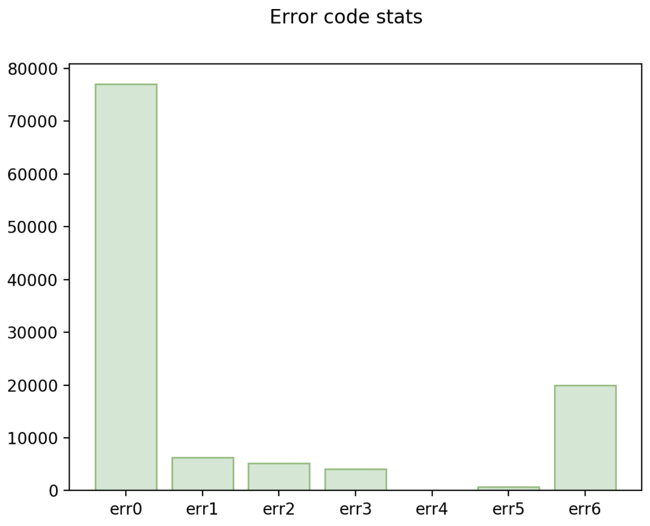


Figure 2.5: Distribution of the different error codes for plant 1.

2.3 Artificial Neural Networks - Overview

Artificial Neural Networks (ANNs) is one of the most popular machine learning model types used today, a model that initially took inspiration from the structure of neuron networks in the brain.

The ANN is a network of nodes, or artificial neurons, which can be compared to neurons in the brain. The most basic node (also known as a perceptron) receives inputs, $\boldsymbol{x}$, that are multiplied with corresponding weights, $\boldsymbol{\omega}$. A node will pass the weighted sum of the inputs and a weighted bias through an activation function φ to convert the input into a more useful output $\boldsymbol{y}$ (see fig. 2.6 and eq. 2.1) [Goodfellow et al., 2016].

$$y_i = \varphi(\sum_{k=1}^{N} \omega_{ik} x_k + \omega_{i0}) \tag{2.1}$$

In other words, a neuron computes an output depending on the inputs, where the weights express the *importance* of the respective inputs to the output.

The simplest version of a neural network is connecting the input node to the output nodes. Usually this is not enough to model more complex problems and therefore a *multi-layer perceptron*, with one or more hidden layers of nodes, can be used (see fig. 2.7). An important aspect of a basic ANNs is that the output from a node will be used as an input for all the nodes in the next layer. By fully connecting all nodes, relations can be found between all nodes.

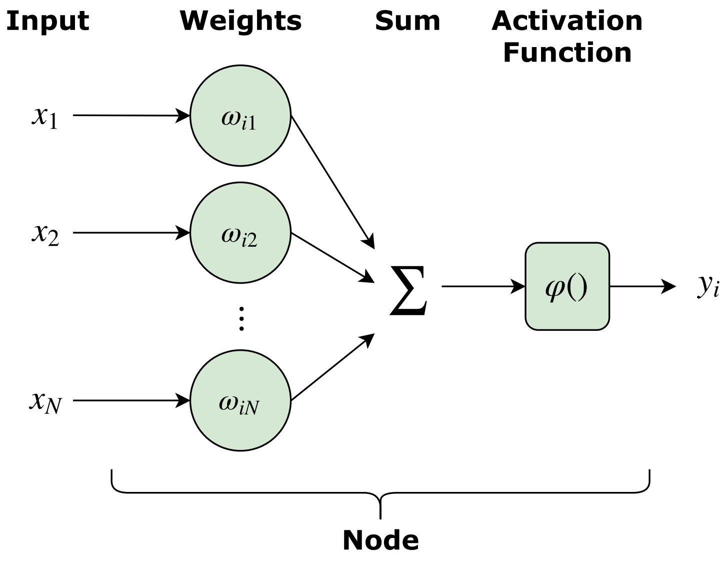


Figure 2.6: A typical artificial neuron, or node, with multiple inputs (x_k), weight values (ω_{ik}), where the weighted sum is passed through an activation function $\varphi()$.

The output $\mathbf{y}$ is defined by a function $f(\mathbf{x})$, where $f(x) = f^1(f^2(...f^n(\mathbf{x})...))$ and f^i is the function in layer i, consisting of the weights $\boldsymbol{\omega_i}$. The training of ML models consists of updating the models weights, ω, by minimizing some *cost function* or *loss function*, $E(\mathbf{x}, \omega)$.

The cost function is measured by measuring how much the output $\mathbf{y}$ and the targets $\mathbf{d}$ disagree. This function will be minimized with respect to the weights, ω, by updating ω according to an optimization method. The update of weights will be propagated backwards in the network starting at the output layer, called *back-propagation*. One feed forward pass and one back-propagation of all data is called one *epoch*. ANNs are usually trained for hundreds of epochs before the optimizer algorithm finds a minimum in the loss function.

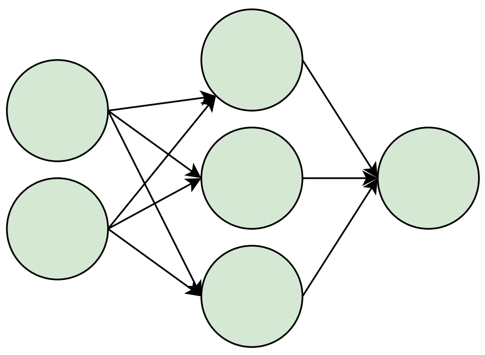


Figure 2.7: A simple *Multi-layer perceptron* with one hidden layer of three nodes. Each circle is a separate artificial neuron.

2.3.1 Supervised Learning

Learning problems for ML algorithms can be divided into two categories: supervised learning and unsupervised learning. For supervised learning there are available targets **d** for inputs **x**, and the learning problem is to construct a model that best maps **x** to **d**. Unsupervised learning can be used when targets **d** are not available, and the learning problem is to use techniques such as clustering to find targets **d**, followed by creating a model that best maps **x** to **d**.

The learning problem for PdM can either be supervised or unsupervised, depending on the available data. The potential targets **d** for PdM could be *Remaining Useful Life*, machine break-downs, service reports or similar. If no targets **d** are available unsupervised learning algorithms can be used to retrieve features and patterns from the data [Bishop, 2006]. In this thesis all data sets have available targets and **only supervised learning methods are investigated**.

2.3.2 Classification & Regression

Regression and classification are two kinds of supervised learning. Models designed for regression problems will generate one or more linear outputs, e.g. *Remaining Useful Life*, and models designed for classification problems generate discrete values associated with a class, e.g. error codes or broken parts.

Models for classification problems will generate a probability that the input belongs to an output class. Using this probability, a decision must be taken whether the input is to be mapped to that output class or not; a cut-off threshold. Often a cut-off threshold of 0.5 is used, i.e. the input is classified to the output class only if the probability is higher than the cut-off limit. For multi-class classification problems, generally the class with the highest probability is picked as the output class.

2.3.3 Data Splitting

When selecting and training a ML model, three data sets are needed: training, testing and validation. The original data set is split into the three parts and usually the training set is the largest since the model needs a lot of data to train. The training data set is used to train the model. For each type of model architecture, multiple hyper-parameters (learning rate [Bishop, 2006], max-pooling) can be optimized. The model is trained with different hyper-parameters and afterwards all models are evaluated using the test data set. The parameters of the model with best performance on the test set are chosen as the optimal set of parameters. To compare different model architectures, each model architecture is evaluated on the validation data set using the optimal parameters. It is important that each data set is independent of each other and no samples occur in more than one set, otherwise the performance measure will not be an unbiased performance measure. If the performance measure is biased it might not be a good representation of the models ability to generalize the underlying problem. Instead a good performance measure might depend on characteristics of a specific data set [Bishop, 2006].

2.4 Artificial Neural Networks - Types

There are many different varieties of ANNs - many different techniques that can be used. Each version boasts certain benefits and/or drawbacks.

This section will present the following ANNs: Feed Forward Neural Networks, Convolutional Neural Networks and Long Short-Term Memory. These ANNs are chosen since they are good representations of different kinds of ANNs as well as different levels of complexity.

2.4.1 Feed Forward Neural Networks

The Feed Forward Neural Network (FFNN) is the most "basic" type of ANN and works like the simplest networks explained in section 2.3. The network consists of an input layer with the same number of nodes as there is input data and an output layer with the same number of nodes as there is output data. Between these layers there are a number of hidden layers and these layers can have any amount of nodes. A key feature with a FFNN is that all the nodes in a layer are connected with all the nodes in the previous and next layer.

2.4.2 Recurrent Neural Networks

A Recurrent Neural Network (RNN), is a ANN with node connections *between* nodes in the same layer or nodes at even lower levels (see fig. 2.8). This makes it possible for the network to have a memory and let output depend on previous input. It is important to be able to account for dependencies between instances in sequential input. An important aspect of RNNs is the concept of weight sharing, since the same weights are used over the entire sequence. Weight sharing will make it possible the generalize over the sequence, which will make it possible to observe reoccurring sequences within a sequence in the same way [Goodfellow et al., 2016].

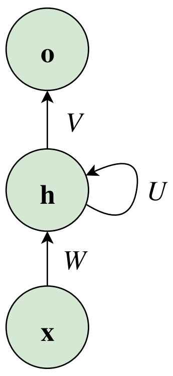


Figure 2.8: A RNN with a hidden layer using the previous output, weighted with U, as input to itself. **x** is the input, **h** is the hidden layer and **o** is the output. W, U and V are weights between the nodes.

To represent how the hidden output is passed forward in time, *unfolding in time* is used. Figure 2.9 demonstrates how the weights W, U and V are shared through time.

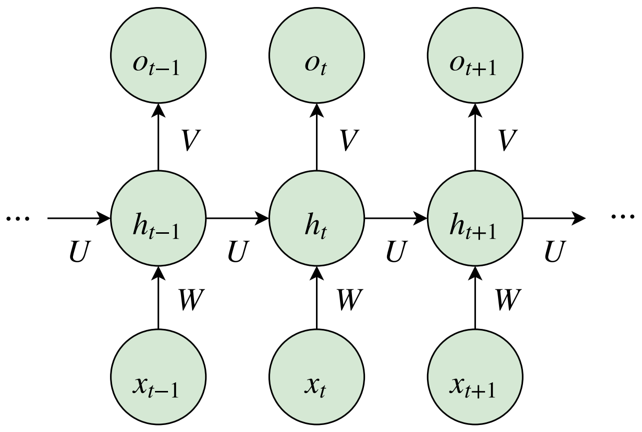


Figure 2.9: The RNN in fig. 2.8 unfolded in time, t. x_t is the input, h_t is the hidden layer and o_t is the output. W, U and V are weights between the nodes.

Feed forward in a recurrent network is performed on the unfolded graph. Each state is evaluated sequentially and saved, until used in back-propagation. The gradients of the RNN is calculated the same way as for a feed forward network, through back-propagation. Also called *Back-Propagation through Time (BPTT).*

A problem with BPTT is vanishing or exploding gradients, when gradients are calculated in many steps. Multiplying small weights ($0 < \omega < 1$) many times will result in values close to or equal to zero. Because weights exponentially get smaller, it makes it hard for RNNs to track long term dependencies,[Gers et al., 2000].

2.4.3 LSTM

Long Short-Term Memory (LSTM), is a special type of neural node. Each node contains one or more memory cells, as well as gates controlling the flow of updates inside the node. The internal state of the LSTM node is controlled by the *Constant Error Carousel (CEC)* [Olah, 2015]. The internal state is protected by the input and output gates which eliminate the problem of vanishing gradients (see figure 2.10 and equation 2.2). The gates in the cell limits the update of states to only include relevant input and ignore irrelevant input and noise. One important gate is the forget gate, which controls how much of previous states the cell remembers. This will make the cell reset itself instead of becoming constant [Gers et al., 2000]. LSTM can be seen as a more advanced type of RNN, as well as a solution to vanishing/exploding gradients (see section 2.4.2).

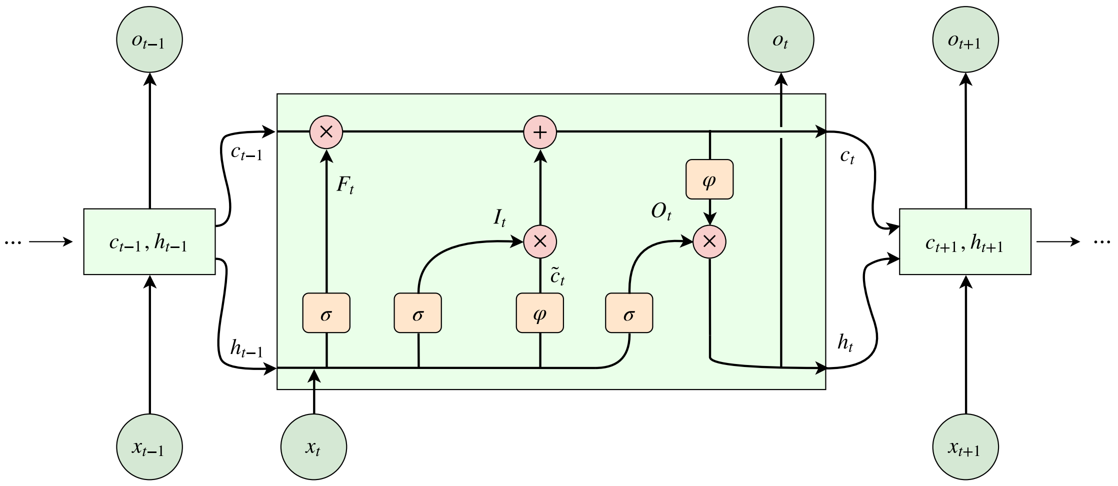


Figure 2.10: Schematic graph of an LSTM node at time t [Olah, 2015]. It contains three gates: forget gate F_t, input gate I_t and output gate O_t. Additionally it contains an intermediary gate $\tilde{c}_t$, which creates candidate values that could be added to the state c_t.

Equations 2.2 show that each gate $(F_t, I_t, O_t, \tilde{c}_t)$ has separate weights $(\omega_F, \omega_I, \omega_O, \omega_{\tilde{c}}$ respectively). Additionally each gate has a separate activation function; either a logistic activation function, σ, that outputs numbers between $[0, 1]$, or a tanh activation function, φ, that outputs numbers between $[-1, 1]$.

$$\begin{aligned}
F_t &= \sigma(\omega_F \cdot [h_{t-1}, x_t] + b_F) \\
I_t &= \sigma(\omega_I \cdot [h_{t-1}, x_t] + b_I) \\
O_t &= \sigma(\omega_O \cdot [h_{t-1}, x_t] + b_O) \\
\tilde{c}_t &= \varphi(\omega_{\tilde{c}} \cdot h_{t-1}, x_t] + b_c) \\
c_t &= F_t \cdot c_{t-1} + I_t \cdot \tilde{c}_t \\
h_t &= O_t \cdot \varphi(c_t) \\
o_t &= h_t
\end{aligned} \tag{2.2}$$

2.4.4 Convolution Neural Network

CNNs are named after the mathematical operation **convolution** (see equation 2.3). In a CNN at least one of the layers performs a convolution. They are used for data that can be convolved over one or more dimensions, such as spatial convolution of images and temporal convolution over time series.

$$(f * g)_t = \int_{x=0}^{t} f(t-x)g(x)dx \tag{2.3}$$

Unlike FFNNs that are fully connected (all the nodes are connected with one another), CNNs use multiple weight kernels of a fixed size that are used to perform convolution over the input. Hence weights are shared between each convolution. For time sequential data the input is convoluted over time and features are extracted in time steps. The length of a kernel determines the size of the time window considered in the convolution. The number of kernel filters affects the number of features that the convolution layer will be able to find (see figure 2.11).

It is common to use a pooling layer after a convolution layer, e.g. max pooling, which selects the maximum value within a window [Goodfellow et al., 2016].

Convolution can be performed in multiple layers. After all convolution layers the data is flattened into a fully connected layer and can be connected to a FFNN [Goodfellow et al., 2016].

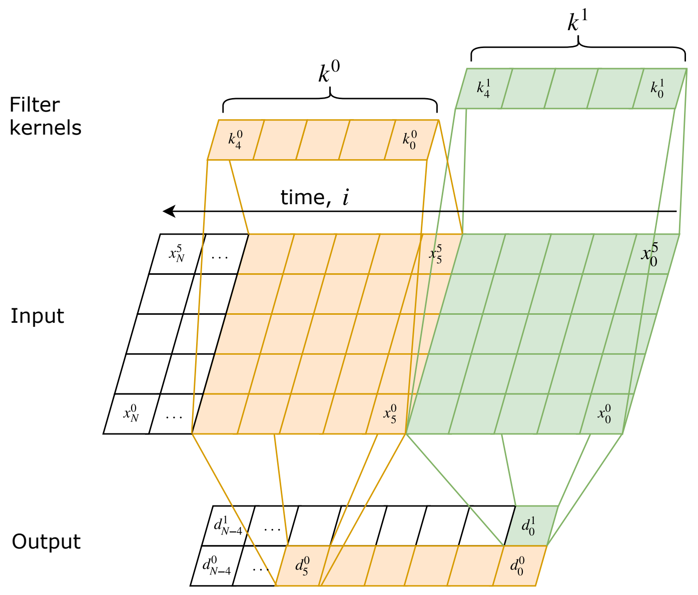


Figure 2.11: Convolution over time with two filters of kernel size 5 and input dimension of 5. Each one of the two filters convolve over the input dimension of 5 and sequence length of 5 in strides of one resulting in two output sequences of length $N-4$, one for each filter.

2.5 ML Performance Evaluation

How well a machine learning model performs is very important to measure, but can also be hard to quantify. For supervised machine learning models (see section 2.3.1) the learning problem consists in finding a model that can produce an output $\mathbf{y}$ as close as possible to the target $\mathbf{d}$ by minimizing the loss function E for a given model, providing a *loss* value and an *accuracy*.

The loss value is used during training to follow the progress of training. The training objective is to minimize the loss value. The accuracy will only tell to what degree the output $\mathbf{y}$ and the targets $\mathbf{d}$ agree (or disagree) for that specific data set. There is no guarantee that what the models has learned is true; the model was trained to perform optimally on the training data. Therefore it is important to make an unbiased and independent measuring of the performance. The sets of data have to be divided between training, testing and validation in order to measure the performance on data the models have not seen before [Japkowicz and Shah, 2011] (see section 2.3.3). For a fair comparison of the performance of different models it is important that the models are trained, optimized and evaluated using the same partitioning between training, testing and validation data sets.

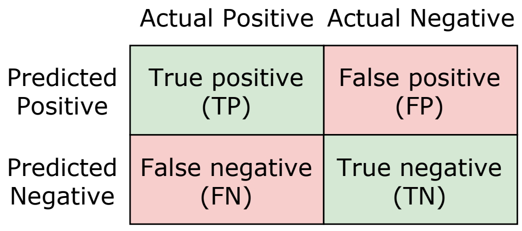
Accuracy is a single value performance measure to which degree the model's predicted values align with the expected/true value, and is especially effective for regression problems and binary classification problems (see 2.3.2 for more about regression and classification problems). However, this value can be misleading and provide a false indication of the actual performance of the model, especially for imbalanced multi-class classification problems (see section 2.2.3). There are other evaluation metrics such as the *confusion matrix* and the *Receiver Operating Characteristic (ROC)* curve that may provide a better performance measure of the model.

2.5.1 Evaluation Metrics

It can be very interesting to look at how many times a model predicted the correct class, as well as how many times it predicted the wrong class, and even more specifically which class it predicted when it predicted incorrectly (for multi-class classification problems). The number of times a model correctly predicts a certain class is called a *true positive* (TP). Similarly, *false positive* (FP) is the number of times a certain class is predicted when it was false, *false negative* (FN) is the number of times a certain class is not predicted when it was in fact true, and *true negative* (TN) is when a class correctly was not predicted (see figure 2.12).

These values can be used for a class to look at the *sensitivity* and *specificity* (see equation 2.4). Sensitivity, or *true positive rate* (TPR), is a measure of how well the model can positively classify a class. Specificity, or *true negative rate* (TNR), is a measure of how well the model can reject a class.

$$\begin{aligned} \text{Sensitivity} = \text{TPR} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\ \text{Specificity} = \text{TNR} &= \frac{\text{TN}}{\text{TN} + \text{FP}} \end{aligned} \tag{2.4}$$



	Actual Positive	Actual Negative
Predicted Positive	True positive (TP)	False positive (FP)
Predicted Negative	False negative (FN)	True negative (TN)

Figure 2.12: When evaluating the performance of a model it becomes interesting to look at the number of *true positive* (TP), *false positive* (FP), *false negative* (FN) and *true negative* (TN).

Other useful evaluation metrics is *precision* & *recall* (see equation 2.5). Precision is the ratio of true positive prediction to the total number of positive predictions, whilst recall is the same as sensitivity.

$$\begin{aligned}\text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ \text{Recall} = \text{Sensitivity} &= \frac{\text{TP}}{\text{TP} + \text{FN}}\end{aligned} \tag{2.5}$$

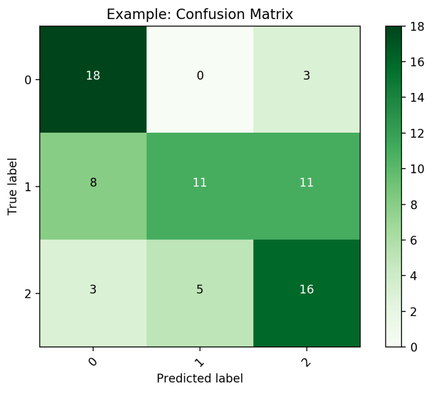
2.5.2 Confusion Matrix

The confusion matrix can be used for multi-class classification problems, to get a better understanding of how well the model is predicting. It shows which class the model has predicted in relation to the actual class. Sometimes it can be helpful to see what class the model is actually predicting instead of the true/real class, and can give insight to relations and/or similarities between classes that were not previously obvious [Shengping and Gilbert, 2017]. Figure 2.13 shows an example of how a confusion matrix can look like. Usually it is more informative to normalize the values, especially if the distribution between the classes is uneven.

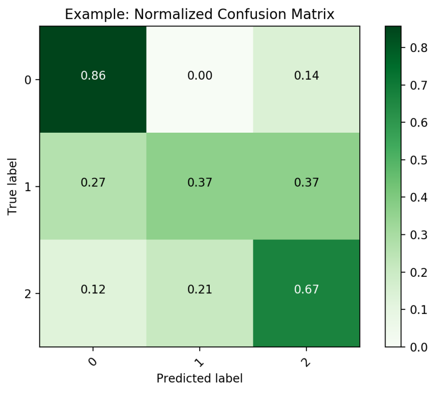
It is also very easy to extract the previously mentioned evaluation metrics (see section 2.5.1) from the confusion matrix. Figure 2.14 shows how the TP, FP, FN & TN values are extracted for class 4 (the data set in this picture is that which is later used in this thesis), which for instance can be used to calculate sensitivity and specificity.

2.5.3 Receiver Operating Characteristic Curve

Especially when working with imbalanced data, plotting Receiver Operating Characteristic (ROC) curves and calculating the *area under the curve* (AUC) of the ROC curve can be a better measure, than simply using accuracy [Metz, 1978]. For example, if 95% of a data set has an output that maps to class 0, and the remaining three classes represent the remaining 5%, the model would receive 95% accuracy even if it constantly predicted class 0 no matter the input. That is, a class that "stupidly" predicts the same output every time could receive a high accuracy.



(a) Not normalized



(b) Normalized

Figure 2.13: Confusion matrix examples. The model has correctly guessed class 0 86% of the time, but class 1 only 37% of the time.

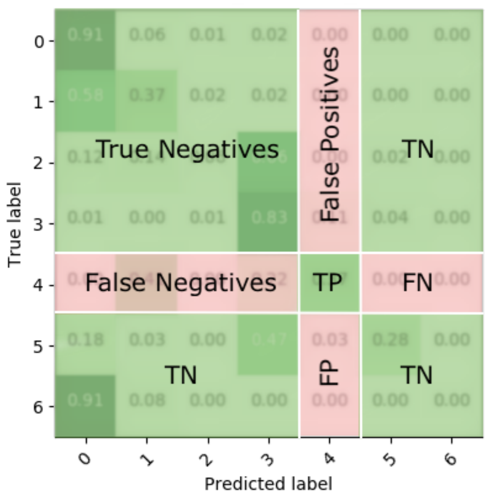


Figure 2.14: Demonstration of how the TP, FP, FN & TN values can be extracted from a confusion matrix.

ROC curves (see the example in figure 2.15) provide a better measure. By calculating the sensitivity (or TPR) and 1−specificity (or false positive rate, FPR) when using different cut-off thresholds (see section 2.3.2 for more about cut-off thresholds), a figure can be drawn with TPR on the y-axis and FPR on the x-axis. The plots are binary, meaning that each class-curve is completely independent. Each point in the curve can be seen as

$$(x, y) = (\text{FPR}(t), \text{TPR}(t)) \tag{2.6}$$

where t is a cut-off threshold value.

The closer a curve is to the top left corner of the figure, the better the model has been at predicting that specific class. The $y = x$ line is added as a reference; any curve that is close to this line means that the model is no better at predicting that class than if it was predicting

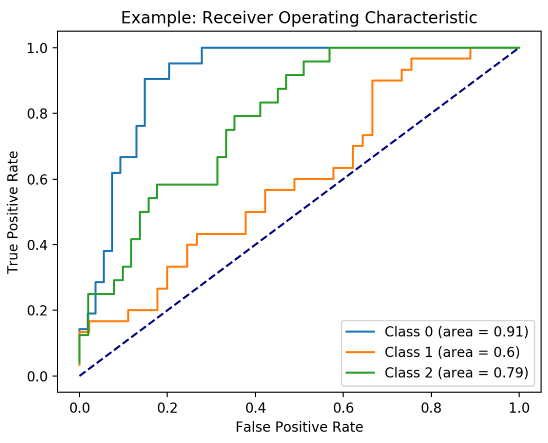


Figure 2.15: ROC curve example, with the same data used as in figure 2.13. In this example the model does quite a good job at predicting class 0, but class 1 less so.

at random. Any curve that is below the reference line is performing worse than random, meaning that it is predicting the class wrong more than it is predicting it correctly. Hence, the AUC value of this curve can be used as a reference for comparing the performance of a model. The closer the AUC value is to 1, the better the model is at predicting that class, and if it is below 0.5 it is worse than if it was predicting at random.

Chapter 3
Method

The goal is to see how well a few different ANNs variants perform for predictive time-series classification. The types of ANNs that will be examined are FFNN, CNN, and LSTM, using the same data data set for a fair comparison.

This chapter will go through the steps taken to preprocess the data (section 3.1), the tools used (section 3.2), the framework that is constructed (section 3.3), and finally the different architectures and hyper-parameters that will be tested.

3.1 Data Preprocessing

The data contains data sets for different plants, with different number of components, which results in different input dimensions (for more information about the data set, see section 2.2.3). A solution that can handle multiple input dimensions can be done by advanced convolution, or by creating separate models for each input dimensions. The lack of domain knowledge of the data set (e.g. knowledge about the placement and type of sensors and components) makes it hard to know what similarities and differences there are in the data for each plant. Hence, within the scope of this thesis data from only one plant is used. *Plant 1* is chosen, at random.

The following preprocessing is done in this order on the data set. This cannot be done in another order due to practical reasons and to facilitate the preprocessing, e.g. there cannot be missing data when mapping input with output, and the standardization is done before the data is reformatted because it is easier to do it in that order. The steps are explained more in depth in the following sections:

1. Handle missing data
2. Map input with output
3. Standardize
4. Reformat data (time slicing & time shifting)

5. Split data whilst ensuring that all classes are represented in all sub-sets
6. Compensate for imbalanced data

3.1.1 Handle Missing Data

Many techniques are available for handling missing values and some approaches suggest using separate ML models to generate missing values [Goodfellow et al., 2013]. To fill missing values two methods are used, mean and mean difference. The mean value is used for most sensor values and it ensures that the missing value is within the range of the present values. The mean was chosen after gathering statistics to rule out any dependency between missing value and output error codes (see figure 2.4 for an example and compare to 2.5). One type of sensor value, *e1*, should be continually growing and therefore the mean difference is added to the previous value or subtracted from the next value, depending on availability. In some cases many values were missing in the beginning, these samples are set to 0.

3.1.2 Map Input to Output

To organize the data it is important to map each time sample in the input with the corresponding error code. Start time and end time for error code time samples are determined by the start and end time for the input files. All time samples were set to error code 0 and then for each time interval in the error code file, the error code was changed to the correct error code. Some error code intervals overlapped in time and since only error code 1-5 was of interest, error code 6 is only allowed to overwrite error code 0. The result is a seen in A.6. For all time intervals not included in the error code file, the error code is 0 and is considered to be no error.

3.1.3 Standardization

In a standard control system there are many sensors involved, often with a large variation in range. Many studies have shown that having data that is standardized will improve a ML algorithm greatly, especially for neural networks [Graves, 2012]. Strictly this is not necessary, as the input weights can be re-scaled to fit inputs of different ranges, but in practice it will make it easier to train the model.

Z-normalizing or standardization is the practice of ensuring that all input vectors have a mean 0 and standard deviation 1. The mean is calculated

$$m_i = \frac{1}{|S|} \sum_{x \in S} x_i \tag{3.1}$$

and standard deviation is calculated

$$\sigma_i = \sqrt{\frac{1}{|S|} \sum_{x \in S} (x_i - m_i)^2} \tag{3.2}$$

for each component of the input vector, and is used to standardize the input vector $\mathbf{x}_i$ using

$$\mathbf{x}_{i,\,\text{new}} = \frac{\mathbf{x}_i - m_i}{\sigma_i} \tag{3.3}$$

This procedure does not alter the information in the data set, but it improves the performance by putting the input values in a range more suitable for the standard deviation functions. It is important that both the training and testing/validating sets are standardized with the same mean and standard deviation.

3.1.4 Reformatting Data

When working with time series data (see section 2.2.2), there are usually patterns and relations that can be found over time. Considering this *time slicing* and *time shifting*[1] are used in this thesis. Additionally, the data is reformatted to fit the different types of ANNs.

Time Slicing

With time series data, there is often relation in the data over time. It might be useful to also look at the previous value(s) to predict the output, rather than just look at the current value. Time slicing, or sometimes called "moving window" or "rolling window", is the practice of using the current value, along with a fixed number of previous values, as input data [Keras, 2018].

In the example which can be found in figure 3.1, a time slice of size 5 is used. For instance at time i the input values $[\mathbf{x}_{i-4}, \mathbf{x}_{i-3}, \mathbf{x}_{i-2}, \mathbf{x}_{i-1}, \mathbf{x}_i]$ are used, and are mapped to the output $\mathbf{d}_i$.

The data in this thesis is always preprocessed to use a time slice of 24, which us equal to six hours. This is chosen since it is stated that errors are independent of sensor values outside a three hour window (see section 2.2.3) and six hours lets the model "remember" twice as long as it is supposed to predict.

Time Shifting

The input data $\mathbf{x}_i$ matches to output data $\mathbf{d}_i$, but sometimes it could be interesting to match the input to another output. For instance it could be interesting to match the input data $\mathbf{x}_i$ to a future output $\mathbf{d}_{i+n}$, where n is the number of data outputs in the future/past. Time shifting is used in this thesis as a suggested solution for prediction.

In figure 3.2 an example is demonstrated, using a time shift of 5. This means that for time instance i, the input data is $\mathbf{x}_i$ and the output data is $\mathbf{d}_{i+5}$.

For the data set in this thesis the targets are always shifted 12 steps, which is equal to three hours. A time shift of three hours is chosen since it is stated that errors are independent of sensor values outside a three hour window (see section 2.2.3).

Reformat to Fit Respective ANN

The data is reformatted to fit the specific ML models. FFNNs do not have any support for remembering backward in time and therefore the input dimension to a FFNN is one whole time slice. E.g. if the time slice is 24 time steps and the input dimension of the data set is 54, the input dimension of the FFNN is $24 \cdot 54 = 1350$. CNNs and LSTM models only

[1]Not to be mixed with *time stamp*, which is simply data at one time instance

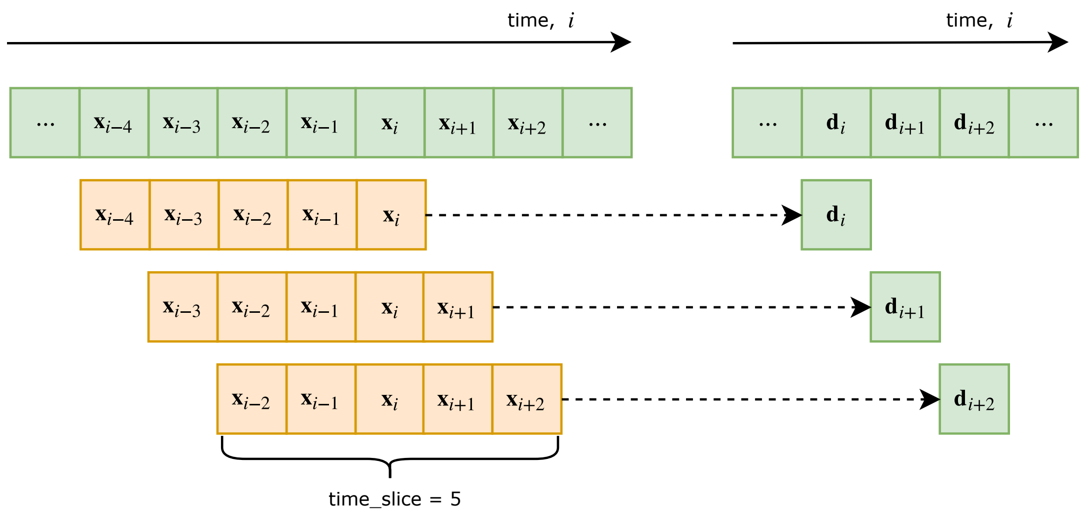

time, i
time, i
$\dots$ $\mathbf{x}_{i-4}$ $\mathbf{x}_{i-3}$ $\mathbf{x}_{i-2}$ $\mathbf{x}_{i-1}$ $\mathbf{x}_{i}$ $\mathbf{x}_{i+1}$ $\mathbf{x}_{i+2}$ $\dots$
$\dots$ $\mathbf{d}_{i}$ $\mathbf{d}_{i+1}$ $\mathbf{d}_{i+2}$ $\dots$
time_slice = 5

Figure 3.1: Time slicing overview, showing which data is selected. In this example, the five most previous $\mathbf{x}$ is the input data, instead of just the current $\mathbf{x}$. This corresponds to the current output $\mathbf{d}$.

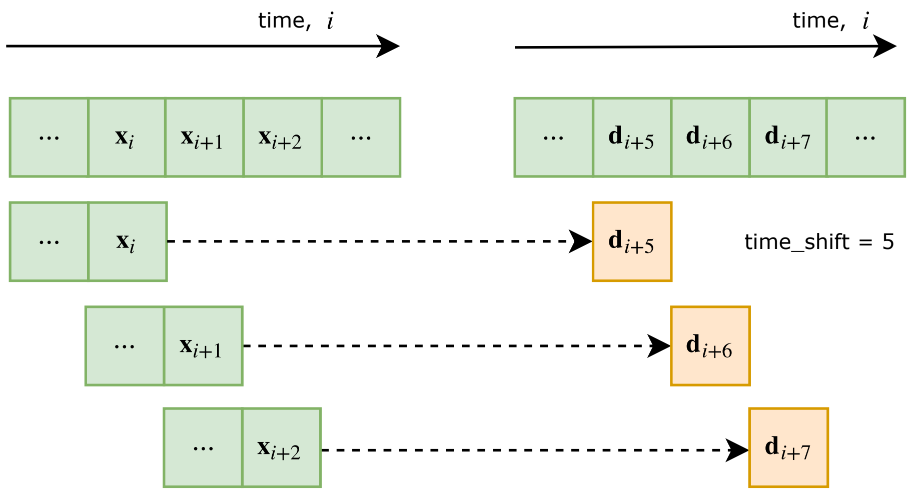

time, i
time, i
$\dots$ $\mathbf{x}_{i}$ $\mathbf{x}_{i+1}$ $\mathbf{x}_{i+2}$ $\dots$
$\dots$ $\mathbf{d}_{i+5}$ $\mathbf{d}_{i+6}$ $\mathbf{d}_{i+7}$ $\dots$
time_shift = 5

Figure 3.2: Time shift overview, showing which data is selected/predicted. In this example with a time shift of 5, for each input data $\mathbf{x}_i$, the output data is $\mathbf{d}_{i+5}$.

consider a limited number of time stamps at a time ($\leq$ 24 time stamps for CNN and one time stamp for LSTM), resulting in a smaller input dimension.

3.1.5 Data Splitting

Before running the ML models, the data is split into three parts: training, testing and validation (see section 2.3.3). Training set is assigned to the first part of the data set (the earliest samples). Test set is assigned to the second part and validation set is assigned to the last part. Because of the imbalance between classes, testing and validation sub-sets do not contain all classes. To be able to achieve an unbiased performance measure it is important that all classes are represented in all subsets. Otherwise rare classes which are harder to predict will not be a part of the performance measurement. Time slice and output pairs are moved from the training subset to either the test or the validation subset, to ensure that all classes are represented in the different subsets. No data samples are fabricated or reused, since a data sample may not exist in more than one subset.

3.1.6 Compensate for Imbalanced Data

For plant 1 error code 0 is represented in 68% of all time samples. Imbalance between classes can be handled in different ways, through over-sampling, under-sampling and cost-sensitive learning [He and Garcia, 2009; Huang, 2015; Wang et al., 2016]. Imbalance is handled by applying different weights for different classes in the cost function when training the model [Chollet et al., 2015]. This means that a faulty classification of a rare class is penalized more heavily than a faulty classification of a more common class. This is not handled in the preprocessing, but in the actual training step.

3.2 Tool Selection

To train a network, a ML framework is useful. This can either be built from scratch which allows for a lot of customization, but can be tedious and time consuming. Alternatively an already built and available framework can be used.

Keras with Tensorflow as back-end is chosen for construction of neural networks. The framework is mature and widely used, which results in broad community and substantial documentation. Keras provides an easy-to-use API for Tensorflow and allows for faster development speed at lower rate of control, since less parameters within training can be tweaked.

3.2.1 Keras

Keras is a high-level API building neural networks using Tensorflow and other libraries for numerical computations as back-end [Keras, 2018]. Tensorflow is a library for numerical computations using data flow graphs and is widely used for ML [Abadi et al., 2016; Tensorflow, 2018]. It is written in `Python` and allows for fast experimental iterations. The core of Keras development is a model and the model is made-up by different layers. The model used in this research is a sequential model, a model with a sequential stack of layers. Keras also has support for more complex neural network graphs. A wide range of layers is available with possibility to tweak variables associated with each type of layer. Keras has support for the neural network architectures mentioned in section 2.4.

3.3 Framework

To facilitate the testing of multiple ANNs, an extensive and custom framework is iteratively developed by the authors to allow for quick and easy parameter changes. This framework consists of a data processing class that parses and prepares the data; a model factory to construct different kinds of ANNs (models); and a model handler that trains, benchmarks and evaluates the models. See figure 3.3 for an overview of the different components.

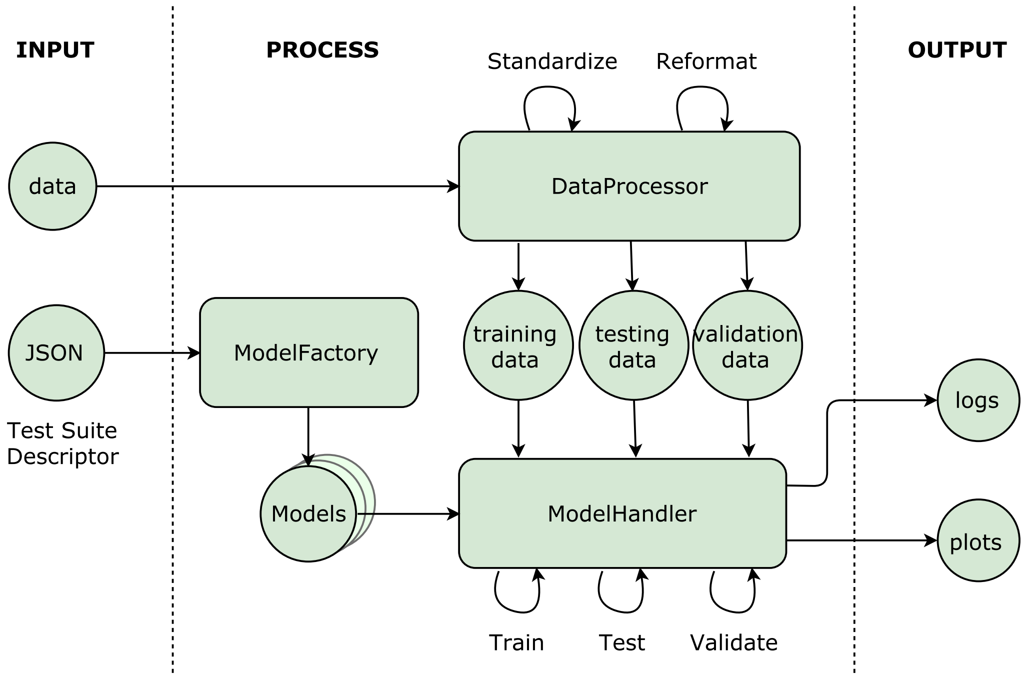


Figure 3.3: Flowchart of the flow within the framework.

DataProcessor

The class `DataProcessor` is designed to parse different data sets, and to process it to fit different models. Once the data is parsed, it is standardized (see section 3.1.3) as well as reformatted to fit a certain time slice or a certain time shift (see section 2.2.2). It also splits the data into training, testing and validation sets, according to specified ratio. The first part of the data set is considered the training set, the second part is considered the testing set and the last part is considered the validation set. The data splitting is performed in this way to keep the time-dependence and to make the split reproducible.

Model Classes

The class `ModelFactory` is used for creating model instances with Keras (see section 3.2.1), model instances which are wrapped into the custom class `Model`. The class `Mod-elHandler` facilitates the management of training, testing and displaying the results for the different models.

Input & Output

Since there are many possible test combinations to find the optimal ML settings, a flexible but robust solution is needed to easily tweak certain parameters. The *Test Suite Descriptor*, a JSON file of custom structure, is developed to ease testing of many combinations. This easy to edit configuration file instructs the framework which data set to test on, how the data set is to be pre-processed, which models are to be built, as well as the details to how the models are to be built. The benefit of using the JSON format is that a Test Suite Descriptor file can be reused, or tweaked and then reused.

When a test suite has finished (when the models have been built, trained and tested), the results are presented in log files as well as in plots.

3.4 Type, Architecture & Parameters

The research in this thesis consists of setting up and training different ANN models and comparing them by measuring their performance. Each type of ANN can have varying architectures, by varying the number of hidden layers and number of nodes in each layer. Additionally, for each architecture there are other parameters that affect the performance of the model, eg. learning rate. This means that theoretically there are infinitely many combinations of model types, architectures and parameters. It is not possible to try all combinations.

Once a model is built, the same training data set, testing data set and validation data set is used, for a fair comparison between all models. The performance of the model is measured by looking at the loss of the training data set and the loss of the testing data set. Additionally, several of the different evaluation metrics mentioned in section 2.5 will be used, such as confusion matrix and ROC curves.

3.4.1 Test Setup

Many of the combinations of architectural design and parameters chosen in this thesis have logical motivations, such as zero or one hidden nodes. Other combinations have been chosen based on previous research, common practice and with the guidance of our supervisors. However, many combinations have iteratively been chosen based on previous results within this thesis.

When deciding combinations of model type, architecture and parameters, there are a few key things that are taken into consideration. These are presented in table 3.1.

3.4.2 Iterative Process

When constructing ML models, common practice is to start with a simple model and add complexity to try to improve the performance.

For example, a first step is to construct simple models of type FFNN with no or only one hidden layer. To test whether it is more effective with fewer or more nodes in the hidden layer, three models are built with 1, 10 and 100 nodes respectively. For each of

Parameter	**Values**
Type of ANN	FFNN, CNN, LSTM
Nbr of Hidden Layers	None, 1, 3
Nbr of Nodes in Hidden Layer	None, 1, 10, 25, 50, 100
Architecture	Decreasing/increasing/varied number of nodes
Nbr of Convolutional Filters	Few (8), many (128)
Nbr of Convolutional Kernels	Few (3), many (24)
Max Pooling	None, 3, 5, 7

Table 3.1: Different types of parameters available when deciding model combinations to test.

these models, different learning rates from 0.00001-0.01 are tested to find the optimal learning rate. The performance on the test data set of these different models is compared.

A next step is to allow the FFNN model to find more connections, and thereby more advanced patterns in the data, by increasing the number of hidden layers and nodes. For example three hidden layers allowed for different model shapes, e.g. [10, 50, 10], [50, 10, 50] and [50, 25, 10]. In the two first examples the dimension is increased or decreased, and then taken back to its previous dimension to let the model represent the data in fewer or more dimensions. The third example gradually reduces the dimension to better fit the output dimension of the data set[2].

To better consider the time aspect of the data two techniques are examined: CNNs and LSTM. The first approach is to use one or two convolution layers, which will convolve the input over time for each time slice. Different kernel sizes and lengths are explored to find settings with best performance.

The approach used for LSTM is the same as for FFNN, where different architectures and different learning rates are tested to find the models with the best performance.

After testing the performance of each model, the performance is evaluated using the validation data set. This is to achieve an unbiased performance measure and not let specific attributes in one data set affect the performance measure. The results presented in section 4.1 are measured on the validation data set.

[2]The output dimension is 7.

Chapter 4
Evaluation

4.1 Results

This section presents the results of training the ANN models. The validation performance of the models will be presented along with the respective prediction accuracy, considering metrics mentioned in 2.5.1. For each new model architecture, only the best performance is presented.

Initially, models are trained with 300 epochs, but some are trained with 300 additional epochs if the loss is clearly still decreasing; hence all models presented in this section have been trained with either 300 or 600 epochs. Training an additional 300 epochs only indicated minor performance gains. The training of models was not limited by time and therefore the models were trained with the same amount of epochs to make a fair comparison.

First the results are presented for each model type. Afterwards, the results are summarized and the different model types are compared.

4.1.1 FFNN

When training FFNN models it quickly became apparent that a smaller learning rate is necessary to be able to learn anything; the best performing models use a learning rate of 10^{-4}. Using learning rates bigger than this results in extremely volatile training and validation loss, whilst smaller learning rates never reach an optimum training and validation losses, even after many epochs.

Table 4.1 presents the results for a few different architectures, gradually increasing in complexity. For the first part, the model trained without compensating for the imbalanced data is not considered. The model with the highest accuracy, with only one hidden layer, has 100 hidden nodes whilst the model with the lowest loss has one hidden node. Figures 4.1a, 4.1b, 4.1c and 4.1d indicate that the single layered models have low accuracy for a

Table 4.1: Summary of results from the FFNN models

Nodes in Layers	Validation Loss	Validation Accuracy	Precision	Recall
None	4.3373	0.0783	0.0839	0.0569
1	2.3968	0.2541	0.0000	0.0000
10	3.0249	0.4377	0.4458	0.4192
100	2.8347	0.5049	0.5168	0.4849
10-50-10	2.7730	0.3628	0.3714	0.3539
50-10-50	3.4743	0.2824	0.3219	0.2204
50-25-10	2.5986	0.5617	0.5704	0.5464
50-25-10 (imbalanced)	1.6855	0.6366	0.6400	0.6291

$$\begin{aligned}
\text{TP} &= 299 \\
\text{TN} &= 20856 \\
\text{FP} &= 1474 \\
\text{FN} &= 27 \\
\text{Sensitivity} = \text{Recall} &= 0.9172 \\
\text{Specificity} &= 0.9340 \\
\text{Precision} &= 0.1786
\end{aligned} \tag{4.1}$$

Equation 4.1: Class 3 performance of FFNN model with one layer of 100 hidden nodes.

few classes. For the model with one layer of a 100 hidden nodes, class 2 and class 4 are only predicted correct 9% and 4% respectively, of all true labels. Instead, class 2 and class 4 are miss-classified as class 3, 70% and 81% of all true labels respectively. Overall many of the classes are miss-classified as class 3, e.g. for class 2 (70%), class 4 (81%) and class 5 (45%). Class 3 is predicted correctly with 87% accuracy, but the high number of false positives result in a low precision (see equation 4.1).

The model with the lowest loss, and highest accuracy, precision and recall is the imbalanced model, i.e. the model not applying different weights for the different classes in training. However, figure 4.1g and figure 4.1h show that class 0 is over-represented in prediction. When comparing sensitivity and specificity for the model with imbalanced training, the sensitivity is almost 0 and specificity is almost 1 for all classes except for class 0. For class 0 the specificity is almost 0 and the sensitivity is almost 1. For the model with balanced training, sensitivity and specificity are closer to each other for most classes, except for class 6 (see table 4.2).

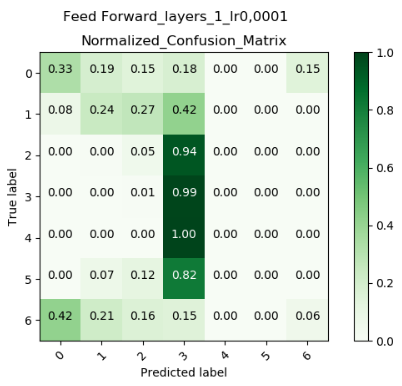
The multi-layer FFNN model with the lowest loss and the highest accuracy is the model with the architecture 50-25-10. The model has low accuracy for some classes (see figure

Table 4.2: Sensitivity (Se) and Specificity (Sp) for the FFNN with architecture 50-25-10, with balanced and imbalanced training

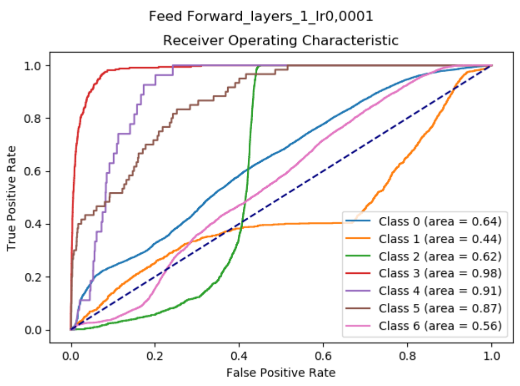
Class	0		1		2		3	
Model	Se	Sp	Se	Sp	Se	Sp	Se	Sp
50-25-10	0.787	0.466	0.635	0.880	0.135	0.989	0.813	0.933
50-25-10 (imbalanced)	0.984	0.038	0	1	0.061	0.987	0.273	0.996

Class	4		5		6	
Model	Se	Sp	Se	Sp	Se	Sp
50-25-10	0.333	0.997	0.283	0.940	0	1
50-25-10 (imbalanced)	0	1	0	1	0	1

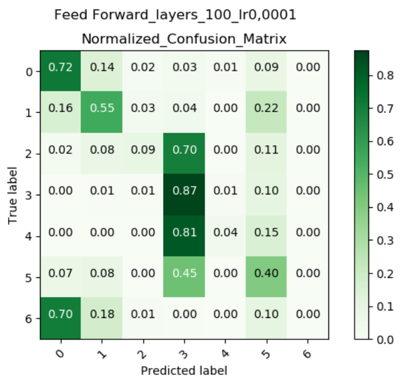
4.1e and 4.1f).



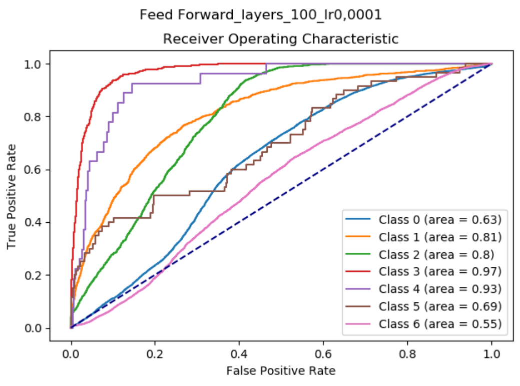
(a) Normalized confusion matrix for FFNN network with one hidden layer of one hidden node.



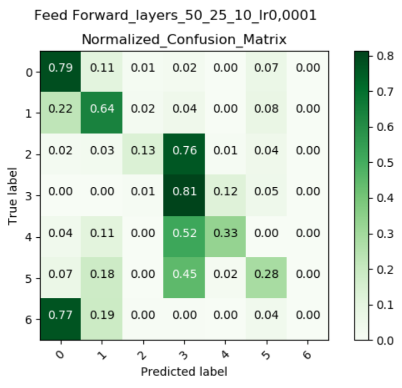
(b) ROC-curve for each class for FFNN network with one hidden layer of one hidden node.



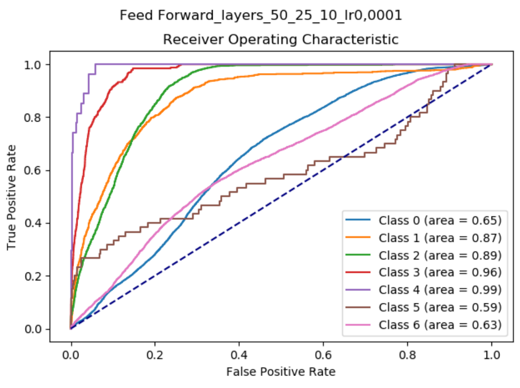
(c) Normalized confusion matrix for FFNN network with one hidden layer of 100 hidden nodes.



(d) ROC-curve for each class for FFNN network with one hidden layer of 100 hidden nodes.

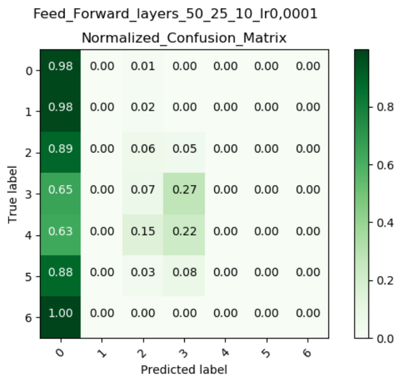


(e) Normalized confusion matrix for FFNN network with three hidden layer of 50, 25 and 10 hidden nodes.

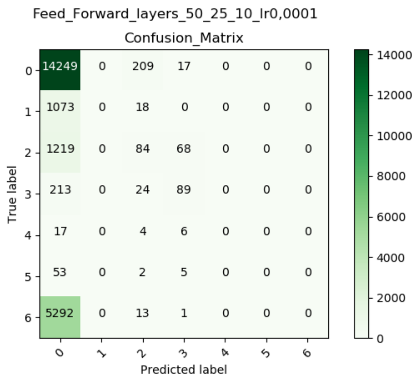


(f) ROC-curve for each class for FFNN network with three hidden layer of 50, 25 and 10 hidden nodes.

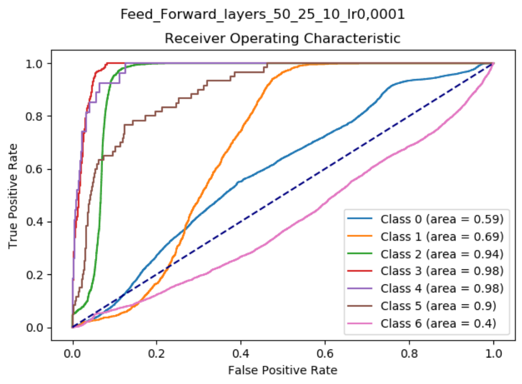
Figure 4.1: The results for FFNN



(g) Normalized confusion matrix for imbalanced FFNN network with three hidden layer of 50, 25 and 10 hidden nodes.



(h) Confusion matrix for balanced FFNN network with three hidden layer of 50, 25 and 10 hidden nodes.



(i) ROC-curve for each class for imbalanced FFNN network with three hidden layer of 50, 25 and 10 hidden nodes.

Figure 4.1: The results for FFNN

4.1.2 CNN

By adding convolutional layers, the hope is to aid the models to find the patterns available over time. Similarly as with FFNNs, using a learning rate of 10^{-4} resulted in the best performance. The training time of ~15-20 minutes is similar to FFNNs.

Table 4.3: Summary of results from CNN models

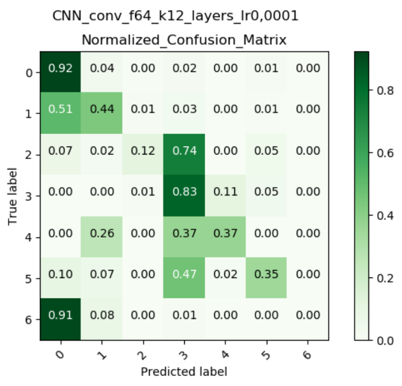
Conv. Filters	**Conv. Kernels**	**Validation Loss**	**Validation Accuracy**	**Precision**	**Recall**	**Max Pool**
8	3	2.2505	0.5616	0.5836	0.5167	5
16	3	2.3024	0.5997	0.6067	0.5703	5
32	3	2.2766	0.5780	0.5858	0.5557	3
64	3	2.3097	0.6081	0.6172	0.5922	7
128	3	2.0077	0.6224	0.6271	0.6125	3
8	12	2.0994	0.6005	0.5785	0.4086	7
16	12	1.9256	0.6210	0.6282	0.5932	5
32	12	2.2603	0.5926	0.6010	0.5728	5
64	12	2.6001	0.6314	0.6339	0.6267	0
64	12	1.8788	0.6130	0.6224	0.5931	3
128	12	2.3107	0.6088	0.6106	0.5922	5
8	24	2.0190	0.5666	0.5822	0.5553	0
16	24	2.3998	0.5450	0.5480	0.5358	0
32	24	2.2713	0.5906	0.5980	0.5829	0
64	24	2.4807	0.6068	0.6099	0.6024	0
128	24	2.4300	0.6133	0.6148	0.6058	0
32-16[1]	12-6[1]	2.0849	0.5363	0.5702	0.4926	3

[1]Two convolutional layers. First layer with 32 filters and kernel size 12 and second layer with 16 filters and kernel size 6.

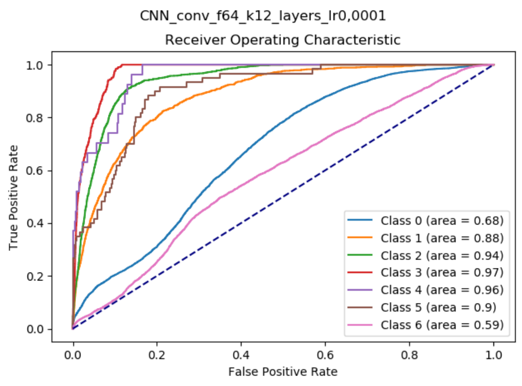
Table 4.3 presents the results for a few different CNNs with different architectures. The models presented do not contain any fully connected hidden layers, only convolutional layers. The combination of convolutional layers and fully connected layers is omitted at an early stage as initial tests do not indicate any performance benefits. Instead, varying numbers of filters and kernels is tested, as well as different max pooling sizes.

The model with the lowest loss and highest accuracy has one layer of convolutional nodes with 64 filters and a kernel size of 12. The model reached the highest accuracy with no max pooling and the lowest accuracy with max pooling 3.

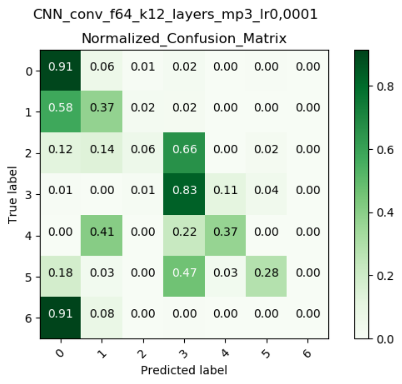
Overall, the performance of the different models is very similar. Initial research in this thesis did not indicate increased performance for multi-layered CNNs and further research is omitted.



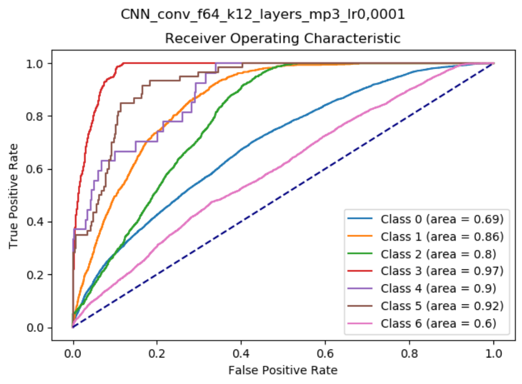
(a) Normalized confusion matrix for CNN network with one hidden layer of 64 filters and kernel size 12 and no max pooling.



(b) ROC-curve for each class for glscnn network with one hidden layer of 64 filters and kernel size 12 and no max pooling.



(c) Normalized confusion matrix for CNN network with one hidden layer of 64 filters and kernel size 12 and max pooling 3.



(d) ROC-curve for each class for CNN network with one hidden layer of 64 filters and kernel size 12 and max pooling 3.

Figure 4.2: The results for CNN

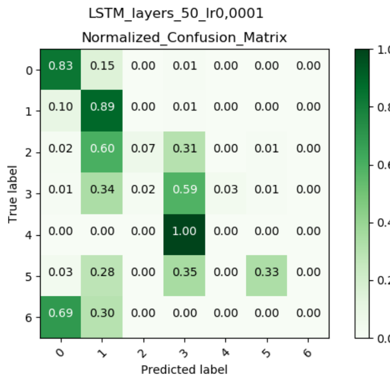
4.1.3 LSTM

Table 4.4: Summary of results from LSTM models

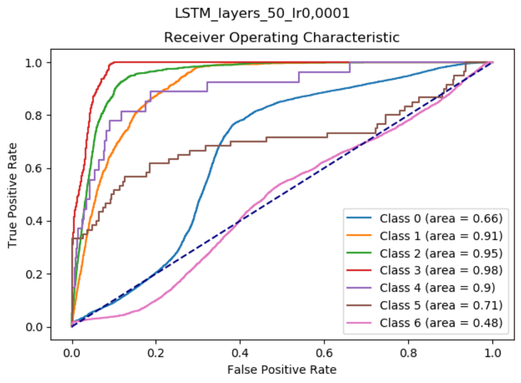
Nodes in Layers	Validation Loss	Validation Accuracy	Precision	Recall
1	2.0223	0.3730	0.0000	0.0000
10	2.2247	0.5374	0.0946	0.0401
25	1.8833	0.5750	0.5786	0.5648
50	1.9024	0.5890	0.5914	0.5846
100	1.7565	0.5978	0.5997	0.5878
25-25-25	1.8549	0.5918	0.5941	0.5793
100-100-100	1.9590	0.5957	0.5994	0.5882

The first observation made when training LSTM networks is the increased training time. The training time is 7 to 14 times longer for LSTM compared to FFNNs, increasing from ~15-20 minutes to ~110-280 minutes.

For single layer LSTM model, 100 hidden nodes achieve the highest accuracy and lowest loss and 50 hidden nodes achieve the second highest accuracy and the second lowest loss. Adding more layers to three layers with 25 nodes each decreases the accuracy slightly and increases the loss slightly. Three layers with 100 hidden nodes each have slightly increased accuracy compared to three layers of 25 hidden nodes each (see table 4.4). Overall the training time increases from ~6500s to ~16500s, when the number of layers is increased from one to three.



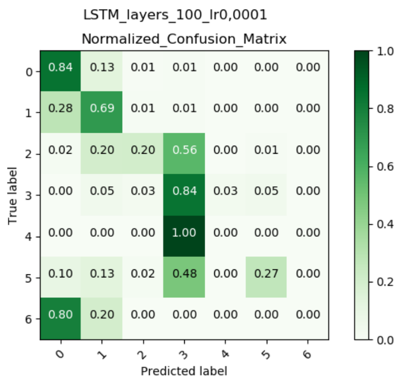
(a) Normalized confusion matrix for LSTM network with one hidden layer of 50 hidden nodes.



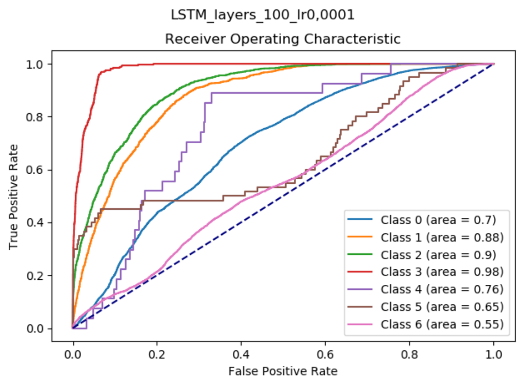
(b) ROC-curve for each class for LSTM network with one hidden layer of 50 hidden nodes.

Figure 4.3: The results for LSTM

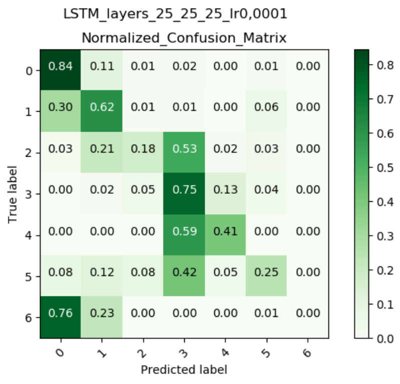
The four LSTM models with the highest accuracy and lowest loss are the model with one hidden layer of 100 nodes, the model with one hidden layer of 50 nodes, the model



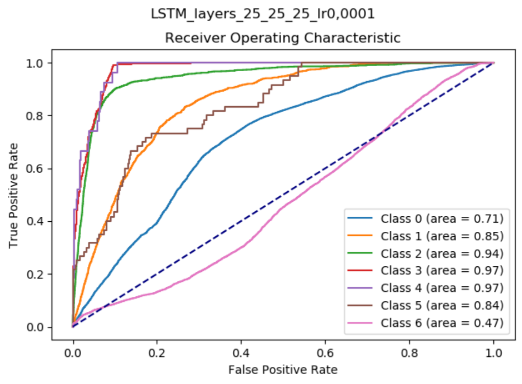
(c) Normalized confusion matrix for LSTM network with one hidden layer of 100 hidden nodes.



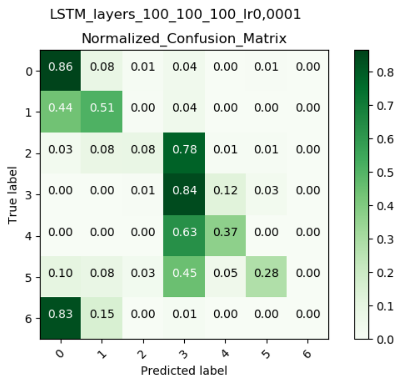
(d) ROC-curve for each class for LSTM network with one hidden layer of 100 hidden nodes.



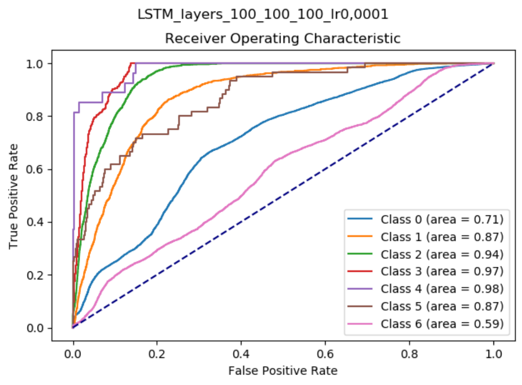
(e) Normalized confusion matrix for LSTM network with three hidden layers of 25 hidden nodes.



(f) ROC-curve for each class for LSTM network with three hidden layers of 25 hidden nodes.



(g) Normalized confusion matrix for LSTM network with three hidden layers of 100 hidden nodes.



(h) ROC-curve for each class for LSTM network with three hidden layers of 100 hidden nodes.

Figure 4.3: The results for LSTM

with three hidden layers of 25 nodes each and the model with three hidden layers of 100 nodes each. Looking at the confusion matrix and ROC-curve for each model it seems like they achieve similar results and both have low accuracy for class 5 and 6 (see figures 4.3).

4.1.4 Summary

Table 4.5 summarizes the results of FFNN, CNN and LSTM. The models included are the models with the lowest loss and highest accuracy for each model type.

When comparing the models they achieve similar performance. FFNN have slightly lower accuracy and slightly higher loss compared to CNN and LSTM. LSTM has lower loss, but lower accuracy than CNN.

Overall, all model types achieve an accuracy between 55-65% and a loss between 1.75-2.60. Comparing confusion matrices, all models have a low number of true positives for class 5 and class 6 and a high number of false positives for class 3 (see figure 2.14 for information on confusion matrices). This indicates that class 5 and class 6 are hard to classify and class 3 is often confused with other classes.

Table 4.5: Summary of results of the best performing models of the different model types.

Model	**Hidden Layers**	**Validation Loss**	**Validation Accuracy**	**Precision**	**Recall**
FFNN	50-25-10	2.5986	0.5617	0.5704	0.5464
CNN	f64-k12-mp0[1]	2.6001	0.6314	0.6339	0.6267
CNN	f64-k12-mp3[1]	1.8788	0.6130	0.6224	0.5931
LSTM	100	1.7565	0.5978	0.5997	0.5878

[1]f = filters, k = kernels and mp = max pooling.

4.2 Discussion

This section will go through the results that were presented in the results section (section 4.1). Section 4.2.1 discusses the effects of different steps during the preprocessing. This is followed by a discussion around choices during the training process, in section 4.2.2. Section 4.2.3 compares and discusses the observed performances of the different models. Finally section 4.2.4 discusses the performance of the models, considering the purpose of PdM.

4.2.1 Data Preprocessing

During the data preprocessing, there are two steps taken that are of bigger interest. These are the *filling in of missing values*, and the *reformatting of data*.

Filling in Missing Values

All the ANNs in this thesis require a constant input dimensions. This means that the input vectors have to contain all input values for all time stamps, it is not possible to have missing values.

The methods used to solve this in this thesis are using either the mean value for that specific sensor, or using the mean difference (see section 3.1), i.e. a constant value is added in place of a missing value. The danger with these methods is that the models potentially could find relations and patterns that do not actually exist, by introducing biases. So apart from the obvious risk that these values are completely wrong compared to the real missing value, there is a risk that "fake information" is introduced instead of finding the most realistic substitution of the missing value.

Also, filling missing values with mean values might remove information if the missing value is, or contributes to, a feature in the data set, i.e. a severe break down/error making the machine unable to log that value. In that case adding a mean value could be entirely faulty. Analysis of the data was done to rule out that missing values is a feature (see section 3.1.1), but further research is necessary to understand what effect this method has on the data set and how this might affect the results.

Reformatting Data

Two major reformatting steps are taken during the preprocessing; *time slicing* and *time shifting*. These two are the key steps taken to be able to find features over time, as well as predicting values in the future.

By using a time slice of 24, the models are always provided with logs from the previous 6 hours, instead of just the current logs. This allows for the model to learn and find patterns/features that occur over time (see section 3.1.4). Naturally, the input dimension becomes much larger when time slicing is used, which means that there is more data to be processed for the models. This will take much longer time to train and could result in a much more complicated ANN. For FFNNs, the time aspect within a time slice is lost, since the whole time slice is presented to the network at once. Both CNNs and LSTMs handles a time slice in a time steps. CNNs considers a time window smaller than or equal to the time slice and moves the time window one time step at a time. LSTMs handles the data within a time slice one sample at a time. When the last sample in a time slice has been processed it makes a prediction for that time slice. However, more data also results in more features that the models can find. Also worth keeping in mind is that if a model is trained with a certain time slice, this is the time slice that always must be used for that model.

Time slicing allows the model to find features over time, but it still does not solve the goal of predicting. This is why time shifting is used (see section 3.1.4). By taking our data, and shifting the output values 12 steps, each input will instead correspond to the output which happens in 3 hours. When these input-output pairs are fed into the model during training, the model is in effect learning to connect the features found in the input with an output in the future. This means that the models trained in this thesis are static in the sense that they only can predict 3 hours ahead, and cannot predict the current output that corresponds to the current input. However, for the purpose of PdM, this is exactly what is of interest.

The danger of having a time shift that is too big is losing the true feature that binds the input with the "real" output. For example, there is a very prominent feature for input x_i which results in the output d_i = class 4 which happens to be a very serious error code[1]. But when this data is shifted 12 steps, x_i is now matched with d_{i+12} = class 0 which is the "healthy" error code and therefore quite uninteresting. The model will now learn to connect a very prominent feature with a rather unimportant output, and similarly, it will learn to connect perhaps a very non-prominent feature with the very important output d_i. However, this problem is hopefully avoided since a time slice is used that is twice the size of the time shift, which should encompass all the features that could appear (it is stated that errors are independent of sensor values outside a three-hour window).

4.2.2 Choices During Training

When preprocessing the data, designing the models and training the models, a number of choices are made. These choices are discussed below.

Balancing Classes

One of the biggest difficulties encountered with the data set is the heavy imbalance of output classes. To balance the training of the networks, the loss function is weighted. The method was chosen after exploring other alternatives, primarily alternatives implemented in Keras and Scikit Learn were explored. Attempts were made to over-sample less common classes and under-sample over-represented classes. These attempts did not perform well, but further research could be interesting.

In table 4.1 a comparison can be made between the two FFNNs with the hidden layers [50-25-10]; one model with a weighted loss function and one without. When not compensating for the imbalanced data during the training, a validation accuracy of almost 64% was achieved, whilst only 56% was achieved when weighted loss functions were used. Weighing the loss function forces the model to learn to predict under-represented classes, but it also results in a lower overall accuracy. Looking at the confusion matrices (figure 4.1g and 4.1e) the actual predictions are visualized and it is clear that the imbalanced model performs bad when trying to predict other classes than class 0.

This is reinforced when looking at the individual sensitivity and specificity values found in table 4.2. For the imbalanced model, the sensitivity is 0 or very close to 0, and the specificity is 1 or very close to 1, for all classes except class 0. However, class 0 has a sensitivity close to 1 and a specificity close to 0. The large sensitivity for class 0 suggests that the model is very good at predicting True Positives for class 0 but extremely bad at picking True Negatives for class 0; it is almost exclusively guessing class 0 every time. When compensating for the imbalanced data, these values are significantly better, but far from great.

So why not pick the imbalanced model when it clearly has a better validation accuracy and validation loss? This is when domain knowledge is of importance. As it turns out, class 0 is the "healthy" error state, meaning that this is the output expected when all is good. In other words, it is of much greater importance to positively predict the other classes, error states of potentially severe consequences. These error states could lead to

[1]In reality, it is unknown how critical the different error codes are.

breakdowns, disruptions in the product line or even safety consequences. It is much more preferable to compensate for the imbalanced data, to be able to predict the other classes. When conducting PdM, the "non-healthy" error states are the most important predictions and therefore it is more important for the model to be able to predict all other classes except class 0. A model with high accuracy, but only predicting the same classes is useless for PdM.

The next natural question is whether the method chosen in this thesis is the best way to compensate for imbalanced data. Section 3.1.6 mentions a few alternatives that are available. There is a possibility that another method is more effective, which would be interesting to investigate.

Architectural Complexity

The architecture of ML models has a great effect on the models potential of learning them from the data. The approach used in this thesis is starting simple and adding complexity in iterations trying to find an optimal architecture. Starting with different single layered architectures and moving towards multiple layered architectures.

For FFNN and LSTM models the number of nodes for the single-layered models is increased by a factor of 10 and then complexity is added by adding more hidden layers. Three types of three-layered FFNN architectures are researched, increasing dimension (10-50-10), decreasing dimension (50-10-50) and triangular (50-25-10). The number of nodes for the models is chosen to be almost equal. The triangular architecture has the highest accuracy and slightly higher loss compared to the model with one hidden layer of one node. Also, when comparing confusion matrices for FFNN (figure 4.1a, 4.1c and 4.1e), the triangular architecture performs the best and shows signs of managing to predict class 5 and class 6. For LSTM more single layered set-ups are tried, since the LSTM nodes are more complex as default. For the multi-layered LSTM, only models with same sized layers are added since they perform better. The LSTM model with one hidden layer of 100 nodes has the highest accuracy and lowest loss and seems to be the model that performs best. When comparing confusion matrices for the LSTM model with one hidden layer of 100 nodes and the LSTM model with three hidden layers of 25 nodes each (figure 4.3c and 4.3e), the LSTM model with three hidden layers of 25 nodes each has a more even distribution of true positives.

When designing the architecture for CNN models, the approach is different. The number of convolution filters (which limits how many features the convolution will be able to extract from the input data) and the size of the convolution kernel (the size of the time window the kernel covers) are the parameters in focus. First attempts are made with a small kernel size of three (45 minutes) and with 8-128 filters. In line with the description of the data set, that error codes are independent of sensor data outside of three hours (see section 2.2.3), the kernel size is increased to 12 (three hours) and with 8-128 filters. The kernel size is limited by the input time slice of 24 (see section 3.1.4) since the kernel cannot look outside the input time slice. The CNN model with best performance is 64 filters and 12 kernels (with different max pooling, see table 4.3).

The architectural design of the ML models was done iteratively based on discussion with supervisors and experience from testing. The iterative design process could definitely be extended further to include more examples, but which architectures work best is highly dependent on the data and the complexity of the features in the data set.

4.2.3 Model Evaluation

When evaluating the models both accuracy and loss is considered as well as confusion matrices and ROC-curves. Table 4.5 shows that the best CNN and LSTM models are slightly better than FFNN models. This is also reflected in the confusion matrices, where especially the frequency of false positives for class 3 is lower for CNN and LSTM models compared to FFNN models. One of the reasons for the improved performance is that CNNs and LSTMs are designed to handle series of data and to be able to find feature and patterns over multiple samples. Also, the input dimension for the FFNN models is huge in comparison.

When considering training time there was a huge increase for LSTMs and especially for multi-layered LSTM models. FFNNs and CNNs had similar training time and therefore one could argue that CNNs are the best models. Training time might not be an issue, but it depends on the data set, the environment where the models will be run and the available resources.

In the comparison above only models trained with a weighted loss function are considered to keep the comparison fair. As mentioned in section 4.2.2, the FFNN model achieved the highest accuracy and lowest loss when not compensating for the imbalanced data. From a PdM point of view, this model is almost useless since it only predicts one class and will not be able to predict any errors. This suggests that loss and accuracy are not good evaluation metrics for ML models used for PdM. Loss and accuracy will give an indication of which models seem to perform better. Especially loss over epochs can be used to get an indication whether the model is getting better or not and a low accuracy indicates that a model is bad. To get a more extended evaluation of a model more metrics are needed. Therefore confusion matrices and ROC-curves are considered, since they are a representation of the actual classification made by the model.

There is an important difference between ROC-curves and confusion matrices. The prediction of the network is always a vector of values between [0, 1], which can be seen as probabilities for each class, e.g. [0.25, 0.0, 0.15, 0.05, 0.05, 0.1, 0.4]. When plotting the confusion matrix, a decision is made to always choose the class with the highest probability. However, the ROC-curve instead considers the probabilities and evaluates the predictions using different cut-off thresholds (see section 2.5.3). This may result in bad performance for a class according to the confusion matrix, but good performance according to the ROC-curve, if the class rarely achieves the highest probability. For example prediction vector [0.51, 0.0, 0.0, 0.0, 0.49, 0.0, 0.0] will result in the confusion matrix choosing class 0, even though the probability for class 0 and class 4 are very close. There are other ways to choose the predicted class, for example choosing class 0 with a probability of 51% and choosing class 4 with a probability of 49%.

4.2.4 Performance Considering PdM

As explained in section 2.1, the goal of PdM is to predict breakdowns and/or error codes. The data set used in this thesis contains error codes from the start, so no preprocessing was necessary to extract the information from raw data. Instead, it is more interesting to examine how the ANNs cope with the predicting.

By using a time slice of 24, features can be found over the past 6 hours, and by using a time shift of 12, the model will always predict 3 hours ahead (see section 4.2.1 for more). A prediction of 3 hours will only be helpful to avoid critical errors where a warning of 3 hours is enough. Generally, predictions further in the future are wanted to provide the machine maintainer with enough to time react to the error code. More time is needed to plan for a delivery of parts to replace broken machines, to plan for the actual maintenance, to temporarily divert production to another machine whilst maintenance is being performed etc.

To predict further in the future, a bigger time shift is needed, and indirectly a bigger time slice to avoid the potential problem of losing the true feature that binds the input with the "real" output (explained in section 4.2.1). Having a big time shift is not problematic, but it requires a larger time slice. Say for example that the model is supposed to learn to predict 31 days into the future instead of 3 hours, a time shift = 2976[2] is needed. This means that the time slice needs to be at least the same size to encompass the input bound with that output, 2976 time steps in the future. Additionally, say that there are features in time that will affect the output, that stretch as far back as 93 days (instead of 3 hours), a time slice = 8928[3] is needed. Using a time slice of 8928 will result in an input dimension of $8928 \cdot 54$[4] $= 482112$ for a FFNN. Clearly this is huge and could introduce issues that are not found with the much smaller input dimensions used in this thesis.

An alternative to using a big time slice could be to use *stateful* LSTM. Stateful LSTM is identical to the LSTM networks that are researched in this thesis, with one big difference: the internal states inside the nodes (see figure 2.10) are not reset when training in-between batches [Keras, 2018]. At an early stage of this thesis work stateful LSTM is applied to the data set, but with little success. The main problem was the size of the training batches and how the data should be reformatted for stateful LSTM. However, it could be very interesting as an alternative to big time slices when there are dependencies that stretch over long periods of time.

[2]Since data is logged 4 times every hour (every 15 minutes): $4 \cdot 24 \cdot 31 = 2976$

[3]$4 \cdot 24 \cdot 93 = 8928$

[4]There are 54 input values for every time step.

Chapter 5
Conclusion

This chapter summarizes the findings that are made in the thesis, as well as answering the research questions (see section 1.2).

How can ANNs handle time-series/continuous data, and how good are they at finding features over time?

FFNNs, CNNs and LSTMs can only handle a limited time window, since they require a limited input size. Since they all have the same requirement, a fair comparison of performance can be made when the data is preprocessed the same way for all three ANNs.

The simplest approach to handle a time window is by feeding the whole time window into a fully connected FFNN. The input dimension becomes very large and the time aspect is lost since all samples within a time window are given the same importance; nothing indicates for the model that there is a sequential relation between different inputs. This does not stop the FFNNs from finding these features over time (especially if they are prominent), but they are simply not optimized to do so.

The second approach is to utilize convolution in a CNN and let the network find features over time through convolution. The length of the kernels determines the size of the time window considered in the convolution, and indirectly the input dimension. The filters that are used to extract features are "rolled" over the data, in the direction of the time. Without extensive domain knowledge it becomes difficult to know the optimal number of kernels and filters, a number linked to the number of features over time that can be expected in the data. However, if the optimal number of filters and kernels are chosen, CNNs have the potential to find features available over time.

The last approach is to introduce a memory state into the network by using LSTM. The internal weights introduced in each node provide the model with another means to control how long a feature should be remembered over time. Intuitively LSTM networks are the most optimized for time series data, for finding features over time.

A drawback for all three types of ANNs is that the models are not that flexible. Once a time slice has been chosen, the input dimension to the model is also chosen, but more importantly it ultimately decides how much "historic" data is needed for the model to produce an output. For example, if a time slice of 3 months is chosen, data stretching back 3 months in time is needed for the model to produce an output. It would be necessary to wait three months before the model can be used on a new plant.

Applied on the data set in this thesis, the overall evaluation shows a small performance improvement for CNN and LSTM, over FFNN. The performance of CNN and LSTM were similar. LSTM networks have support for handling very long sequences and might have a slight edge over CNN if features can be found over a longer time period, than the time slice of 24 that is used in this thesis. While the results in this thesis claim that CNNs & LSTM networks are better suited for time-series data than FFNNs, it is difficult to choose the best ANN. Inevitably the best performance difference out of CNNs and LSTM will depend on the applied data set and the characteristics of that data set.

How can ANN models for PdM be evaluated and compared?

When evaluating ANNs for PdM it is important to consider the application domain, the domain where the ANN is to be used. The loss over training epochs is important to evaluate whether a model is getting better, and the accuracy is a measure of how well the network is classifying the samples. When considering the application domain and characteristics of the data set used in this thesis, accuracy is not enough. The data set used in this thesis is very imbalanced which results in a high accuracy, but extremely biased results with potential severe consequences. A model with high accuracy might in fact be a very bad model for PdM, especially if one class is over-represented.

Therefore, additional metrics are needed which represent the actual predictions. Confusion matrices and ROC-curves are used to examine the actual output of the ANNs.

Specifically for PdM the number of true positives and false negative for certain classes are more important than for other classes. It is more important to accurately predict error codes corresponding to severe errors. Therefore, it might be good to lower the accuracy of the model to increase the number of true positives and decrease the number of false negatives for specific classes. The overall goal for PdM is to minimize the number and cost of unscheduled machine failures and the ANNs should be evaluated on their ability to minimize these measures.

What are the benefits & limitations of using ANN models for PdM?

ANNs are versatile ML models, which can be applied to many different domains. Although, there are a limitations to using ANNs for PdM.

As mentioned, the input size to an ANN has to be fixed and missing values have to be filled with substitute values. Hence, it is hard to represent missing values as a feature of the data set. To be able to train a machine learning model, a lot of data is needed. This data should be representative for the whole domain, otherwise the model will not learn all corner cases. Additionally the data has to be of good quality, otherwise it will be hard for the model to generalize the underlying function. If the data set is imbalanced and certain classes are over-represented in the training, the model will over-train for these specific classes. To solve for imbalanced data, the classes have to be balanced, e.g. by weighting the loss function or by over-sampling. Another potential drawback of ANNs is that the training requires a significant amount of computing resources, which also may take a lot of time.

Multiple benefits are noticed during research within the thesis. Using ANNs for PdM still requires domain knowledge, but greatly reduces the extent to which it is needed. Instead of the network architect having to extract features from the data, the network itself will be trained to detect useful features. ANNs perform very well for samples that are frequent in the training set, i.e. the normal cases, which is connected to the networks potential difficulty to classify unusual classes. There are types of ANNs that are designed to handle sequences of data, e.g. CNN and LSTM. These models will be able to consider the dependency between samples over time. By presenting additional training samples to the ANN, the network will improve and therefore, it will be possible to improve the network over time.

5.1 Future Work

Several extensions and different approaches can be researched as future work.

ANNs not included in this thesis are simple RNNs, stateful LSTMs and Clockwork RNNs [Koutník et al., 2014]. RNNs refers to ANNs with simple connections backwards in time (see section 2.4.2). Stateful LSTMs are mentioned in section 4.2.4 and refers to LSTMs where the internal state are not reset, i.e. the LSTM nodes will be able to remember arbitrarily long time backwards in time. Koutník et al. mentions a network with recurrent hidden layers processing the input at different clock rates. The clock rate of each layer is also a trainable parameter. This node type is constructed as an extension to RNNs to improve sequence prediction and classification [Koutník et al., 2014].

One discussed approach, which is only briefly researched, is using different binary classifiers to classify each class. Each classifier will be trained to make a binary classification of whether it is the correct class or not. For predictions each classifier will make a prediction and the predictions will be combined into one predicted class. How the predicted class is chosen depends on the data set and the application. This can be useful when the data is imbalanced.

One method research briefly is predicting multiple time steps ahead, e.g. prediction 12 steps forward from t_1 to t_{12}. This makes the model more adaptable and the user can more easily choose which time step of the prediction to use, instead of having to retrain the model for a different time shift.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X., and Brain, G. (2016). Tensorflow: A system for large-scale machine learning.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC.

Bokrantz, J., Skoogh, A., and Lundgren, C. (2017). Smash - smart maintenance assessment. `https://research.chalmers.se/en/project/7628` [2018-04-17].

Chollet, F. et al. (2015). Keras. `https://github.com/keras-team/keras` [2018-03-08].

Cline, B., Niculescu, R. S., Huffman, D., and Deckel, B. (2017). Predictive maintenance applications for machine learning. *2017 Annual Reliability and Maintainability Symposium (RAMS), Reliability and Maintainability Symposium (RAMS), 2017 Annual*, page 1.

Dekker, R. (1996). Applications of maintenance optimization models: a review and analysis. *Reliability Engineering System Safety*, 51(3):229–240.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451 – 2471.

Good Solutions (2018). Rs production. `http://www.goodsolutions.se/products-and-services/rs-production/` [2018-05-05].

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org` [2018-03-08].

Goodfellow, I. J., Courville, A., and Bengio, Y. (2013). Joint training deep boltzmann machines for classification.

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks.* Springer, Toronto, Ontario, Canada.

He, H. and Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9).

Huang, P. J. (2015). Classification of imbalanced data usingsynthetic over-sampling techniques. Master's thesis, University of Califorina Los Angeles.

Japkowicz, N. and Shah, M. (2011). *Evaluating learning algorithms. [Elektronisk resurs] : a classification perspective.* Cambridge : Cambridge University Press, 2011.

Keras (2018). Documentation. `https://keras.io/` [2018-03-08].

Kim, H., Ha, J. M., Park, J., Kim, S., Kim, K., Jang, B. C., Oh, H., and Youn, B. D. (2015). Fault log recovery using an incomplete-data-trained fda classifier for failure diagnosis of engineered systems. In *Annual Conference of the Prognostics and Health Management Society 2015.*

Koutník, J., Greff, K., Gomez, F., and Schmidhuber, J. (2014). A clockwork rnn.

Metz, C. E. (1978). Basic principles of roc analysis. *Seminars In Nuclear Medicine*, 8(4):283 – 298.

Mobley, R. K. (2002). *An introduction to predictive maintenance. [Elektronisk resurs].* Amsterdam ; New York : Butterworth-Heinemann, 2002.

Olah, C. (2015). Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/` [2018-02-25].

Prognostics and Health Management Society (2015). Phm data challenge 2015. `https://www.phmsociety.org/events/conference/phm/15/data-challenge` [2018-03-23].

Shengping, Y. and Gilbert, B. (2017). The receiver operating characteristic (roc) curve. *Southwest Respiratory and Critical Care Chronicles*, 5(19):34–36.

Sigma IT Consulting (2018). Asset by sigma. `http://assetbysigma.se/` [2018-05-05].

Singh, A. (2017). Anomaly detection for temporal data using long short-term memory (lstm). Master's thesis, Kungliga Tekniska Högskolan.

Tensorflow (2018). Documentation. `https://www.tensorflow.org/` [2018-04-17].

Vikon (2018). Smart condition monitoring and rotor balancing. `http://www.vikon.se/` [2018-05-05].

Wang, S., Liu, W., Wu, J., Cao, L., Meng, Q., and Kennedy, P. J. (2016). Training deep neural networks on imbalanced data sets. In *2016 International Joint Conference on Neural Networks (IJCNN).*

Wu, Y., Yuan, M., Dong, S., Lin, L., and Liu, Y. (2018). Remaining useful life estimation of engineered systems using vanilla lstm neural networks. *Neurocomputing*, 275:167 – 179.

Xiao, W. (2015). A probabilistic machine learning approach to detect industrial plant faults: Phm15 data challenge. In *Annual Conference of the Prognostics and Health Management Society 2015*.

Xie, C., Yang, D., Huang, Y., and Sun, D. (2015). Feature extraction and ensemble decision tree classifier in plant failure detection. In *Annual Conference of the Prognostics and Health Management Society 2015*.

Appendices

Appendix A
File formats

A.1, A.2 and A.3 give an example of what the the raw data looks like before it has been parsed and pre-processed, and how the data is split up into these three different kinds of formats. Similarly A.4, A.5 and A.6 show what the same data looks like once it has been parsed and processed.

A.1 Input Part A

```
1,2009-08-18 18:12:00,711,630,69,600,689,20,40,1
5,2009-08-18 18:14:27,725,460,101,705,689,20,40,1
6,2009-08-18 18:14:31,711,505,69,678,689,20,40,1
1,2009-08-18 18:14:43,705,630,69,600,689,20,40,1
2,2009-08-18 18:14:47,734,516,101,671,689,20,40,1
3,2009-08-18 18:14:51,743,637,69,595,689,20,40,1
4,2009-08-18 18:15:00,730,577,101,633,689,20,40,1
2,2009-08-18 18:29:33,721,511,101,674,689,20,40,1
3,2009-08-18 18:29:37,747,563,101,642,689,20,40,1
4,2009-08-18 18:29:45,716,572,101,636,689,20,40,1
5,2009-08-18 18:29:49,718,455,101,708,689,20,40,1
6,2009-08-18 18:29:54,712,613,5,610,689,20,40,1
1,2009-08-18 18:30:00,705,624,69,604,689,20,40,1
1,2009-08-18 18:44:31,703,621,69,606,689,20,40,1
2,2009-08-18 18:44:35,725,509,101,676,689,20,40,1
3,2009-08-18 18:44:39,743,559,101,644,689,20,40,1
4,2009-08-18 18:44:48,712,568,101,639,689,20,40,1
5,2009-08-18 18:44:52,720,451,101,710,689,20,40,1
6,2009-08-18 18:44:56,712,549,69,651,689,20,40,1
1,2009-08-18 18:59:30,700,617,69,608,689,20,40,1
```

A.2 Input Part B

```
2,2009-08-18 18:14:56,233.72,31.52
3,2009-08-18 18:14:56,629.44,49.20
1,2009-08-18 18:15:02,863.16,80.72
2,2009-08-18 18:29:41,241.44,31.41
3,2009-08-18 18:29:42,641.31,45.38
1,2009-08-18 18:29:47,882.75,76.78
2,2009-08-18 18:44:44,249.31,31.47
3,2009-08-18 18:44:44,653.34,49.02
1,2009-08-18 18:44:50,902.66,80.50
2,2009-08-18 18:59:43,257.13,31.14
3,2009-08-18 18:59:43,665.50,48.94
1,2009-08-18 18:59:49,922.63,80.08
2,2009-08-18 19:14:45,264.88,30.94
3,2009-08-18 19:14:46,677.47,48.69
1,2009-08-18 19:14:51,942.34,79.63
2,2009-08-18 19:29:53,272.75,31.17
3,2009-08-18 19:29:53,688.00,43.15
1,2009-08-18 19:29:59,960.75,74.32
2,2009-08-18 19:44:55,279.69,21.63
3,2009-08-18 19:44:56,698.84,45.73
```

A.3 Targets

```
2009-08-31 21:44:47,2009-08-31 22:29:47,6
2009-08-31 13:59:36,2009-08-31 22:44:54,6
2009-09-01 07:44:53,2009-09-01 08:14:55,6
2009-09-01 08:44:55,2009-09-01 09:29:53,6
2009-09-01 09:44:56,2009-09-01 15:30:01,6
2009-09-01 15:44:59,2009-09-01 15:59:58,6
2009-09-01 16:14:57,2009-09-01 16:29:56,6
2009-09-01 16:59:55,2009-09-01 17:30:00,6
2009-09-01 21:59:35,2009-09-01 22:14:38,6
2009-09-01 18:44:43,2009-09-01 22:44:49,6
2009-09-01 17:55:45,2009-09-02 00:00:57,6
2009-09-02 08:44:34,2009-09-02 13:44:37,6
2009-09-02 13:59:58,2009-09-02 14:44:59,6
2009-09-02 15:59:36,2009-09-02 16:29:35,6
2009-09-02 21:44:43,2009-09-02 22:29:44,6
2009-09-02 16:59:34,2009-09-02 22:44:52,6
2009-09-03 08:45:02,2009-09-03 09:30:01,6
2009-09-03 09:44:32,2009-09-03 14:29:36,6
2009-09-03 14:44:36,2009-09-03 15:14:35,6
2009-09-03 15:44:33,2009-09-03 16:45:02,6
```

A.4 Parsed Input Part A

```
,time ,s11 ,s21 ,s31 ,s41 ,r11 ,r21 ,r31 ,r41 ,s12 ,s22 ,s32 ,s42 ,...
0,2009-08-18 18:00,711.0,630.0,69.0,600.0,689.0,20.0,...
1,2009-08-18 18:15,705.0,630.0,69.0,600.0,689.0,20.0,...
2,2009-08-18 18:30,705.0,624.0,69.0,604.0,689.0,20.0,...
3,2009-08-18 18:45,703.0,621.0,69.0,606.0,689.0,20.0,...
4,2009-08-18 19:00,700.0,617.0,69.0,608.0,689.0,20.0,...
5,2009-08-18 19:15,698.0,612.0,69.0,612.0,689.0,20.0,...
6,2009-08-18 19:30,702.0,633.0,69.0,598.0,689.0,20.0,...
7,2009-08-18 19:45,705.0,729.0,69.0,537.0,689.0,20.0,...
8,2009-08-18 20:00,705.0,727.0,5.0,538.0,689.0,20.0,...
9,2009-08-18 20:15,705.0,675.0,5.0,571.0,689.0,20.0,...
10,2009-08-18 20:30,702.0,675.0,5.0,572.0,689.0,20.0,...
11,2009-08-18 20:45,702.0,624.0,5.0,604.0,689.0,20.0,...
12,2009-08-18 21:00,702.0,712.0,69.0,548.0,689.0,20.0,...
13,2009-08-18 21:15,707.0,732.0,5.0,535.0,689.0,20.0,...
14,2009-08-18 21:30,698.0,671.0,5.0,574.0,689.0,20.0,...
15,2009-08-18 21:45,703.0,723.0,5.0,541.0,689.0,20.0,...
16,2009-08-18 22:00,698.0,628.0,69.0,601.0,689.0,20.0,...
17,2009-08-18 22:15,702.0,714.0,5.0,546.0,689.0,20.0,...
18,2009-08-18 22:30,703.0,720.0,5.0,543.0,689.0,20.0,...
```

A.5 Parsed Input Part B

```
,time ,e11 ,e21 ,e12 ,e22 ,e13 ,e23
0,2009-08-18  18:00 ,,,,,,
1,2009-08-18  18:15 ,863.16 ,80.72 ,233.72 ,31.52 ,629.44 ,...
2,2009-08-18  18:30 ,882.75 ,76.78 ,241.44 ,31.41 ,641.31 ,...
3,2009-08-18  18:45 ,902.66 ,80.5 ,249.31 ,31.47 ,653.34 ,...
4,2009-08-18  19:00 ,922.63 ,80.08 ,257.13 ,31.14 ,665.5 ,...
5,2009-08-18  19:15 ,942.34 ,79.63 ,264.88 ,30.94 ,677.47 ,...
6,2009-08-18  19:30 ,960.75 ,74.32 ,272.75 ,31.17 ,688.0 ,...
7,2009-08-18  19:45 ,978.53 ,67.36 ,279.69 ,21.63 ,698.84 ,..
8,2009-08-18  20:00 ,997.59 ,65.31 ,287.34 ,30.7 ,710.25 ,...
9,2009-08-18  20:15 ,1014.97 ,68.8 ,295.03 ,30.62 ,720.19 ,...
10,2009-08-18  20:30 ,1032.69 ,74.24 ,301.81 ,30.53 ,731.5 ,...
11,2009-08-18  20:45 ,1050.03 ,61.2 ,308.56 ,21.25 ,742.03 ,...
12,2009-08-18  21:00 ,1067.28 ,60.88 ,315.66 ,21.1 ,752.19 ,...
13,2009-08-18  21:15 ,1084.31 ,55.66 ,322.34 ,20.77 ,761.97 ,...
14,2009-08-18  21:30 ,1101.16 ,58.1 ,328.88 ,20.53 ,772.28 ,...
15,2009-08-18  21:45 ,1119.16 ,60.4 ,335.53 ,20.38 ,783.63 ,...
16,2009-08-18  22:00 ,1136.75 ,70.45 ,341.69 ,20.16 ,795.06 ,...
17,2009-08-18  22:15 ,1154.09 ,71.73 ,347.59 ,28.59 ,806.5 ,...
18,2009-08-18  22:30 ,1172.25 ,76.93 ,353.84 ,28.54 ,818.41 ,...
```

A.6 Parsed Targets

```
,time ,err_code
0,2009-08-18  18:00,6
1,2009-08-18  18:15,6
2,2009-08-18  18:30,6
3,2009-08-18  18:45,6
4,2009-08-18  19:00,6
5,2009-08-18  19:15,6
6,2009-08-18  19:30,6
7,2009-08-18  19:45,6
8,2009-08-18  20:00,6
9,2009-08-18  20:15,6
10,2009-08-18  20:30,6
11,2009-08-18  20:45,6
12,2009-08-18  21:00,6
13,2009-08-18  21:15,6
14,2009-08-18  21:30,6
15,2009-08-18  21:45,6
16,2009-08-18  22:00,6
17,2009-08-18  22:15,6
18,2009-08-18  22:30,6
19,2009-08-18  22:45,6
20,2009-08-18  23:00,6
```

Appendix B
The development process

The thesis and the systems developed in the thesis were written and developed by the two authors. In the first step extensive research was conducted on ANNs, PdM and data preprocessing. The research resulted in the development of a testing framework, which was designed to preprocess data for ANNs, train ANNs and evaluate the models.

Hugo researched data preprocessing specifically for PdM and designed and developed the base for data preprocessing within the framework. Anders researched ANNs specifically for time sequential data and developed the part of the framework constructing and training of the models using Keras.

The evaluation metrics to complement accuracy and loss was researched by Hugo, which was incorporated into the evaluation part of the framework. The development of the testing suites and test setups were split between the authors, but most decisions considering testing were thoroughly discussed by both authors. Anders developed test suites for FFNNs and LSTMs and Hugo developed test suites for CNNs.

All parts of the thesis were written, changed, extended and corrected by both authors. Changes were also made after feedback from supervisors. The discussion and conclusion were by both authors and discussed by both the authors and supervisors.

Extensive discussions between the authors and supervisors were done continuously throughout the thesis.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTATIONSDAG 2018-05-31

EXAMENSARBETE Evaluation of Artificial Neural Networks for Predictive Maintenance
STUDENT Anders Buhl, Hugo Hjertén
HANDLEDARE Patrik Persson (LTH), Johan Lindén (Jayway), Andreas Kristiansson (Jayway)
EXAMINATOR Flavius Gruian (LTH)

Artificiella neuronnätverk för prediktivt underhåll

POPULÄRVETENSKAPLIG SAMMANFATTNING **Anders Buhl, Hugo Hjertén**

Prediktivt data-drivet underhåll kan bidra med besparingar och ökad produktionseffektivitet. Maskininlärning, och framförallt artificiella neuronnätverk, har visat goda förutsättningar som hjälpmedel inom data-drivet underhåll.

Maskinindustrin producerar ständigt stora mängder data som ofta förblir outnyttjad. Data som hade kunnat användas för att förbättra underhållsrutiner och därmed minska frekvensen för maskinfel och öka produktionskvaliteten. Underhåll idag kan delas upp i två delar, åtgärdande underhåll och förebyggande underhåll. Åtgärdande underhåll utförs när en maskin har gått sönder. Förebyggande underhåll utförs innan en maskin går sönder och det utförs för att förebygga och undvika att maskinen går sönder. I stort kan förebyggande underhåll delas upp i tidsbestämt underhåll och tillståndsbestämt underhåll. Tidsbestämt undrehåll innebär att alla maskiner följer samma underhållsschema, t.ex. att de underhålls en gång i månaden.

Alternativet är tillståndesbestämt underhåll, där maskinens hälsotillstånd övervakas. Det möjliggör för individuellt underhållsschema för varje maskin, vilket minskar frekvensen av service av maskiner som inte är i behov av service och ser till att service istället sätts in i rätt tid. Prediktivt underhåll är en typ av tillståndsbestämt underhåll, där felkoder förutspås för att kunna sätta in rätt åtgärder för att undvika felet. Det leder till minskad miljöpåverkan och ökade ekonomiska marginaler.

Ett sätt att utföra prediktivt underhåll är att låta maskininlärningsmodeller lära sig från tidigare data. Inom examensarbetet utvecklades artificiella neuronnätverksmodeller som tränades för att förutspå felkoder tre timmar framåt i tiden. Tre olika typer av artificiella neuronnätverk utvecklades och jämfördes, varav två är designade för att hantera data sparad över tid och därmed lättare kan hitta samband över tid.

Ofta arbetar maskiner i normaltillstånd större delen av tiden, alltså är det mesta av den tillgängliga datan utan några felkoder. Det leder till att modellen har en tendens att tränas för att enbart förutspå att modellen fungerar normalt. Exempelvis kommer en modell, som påstår att maskinen alltid fungerar normalt, uppnå en noggrannhet på 60% om maskinen fungerar normalt 60% av tiden. Ur underhållssyfte är det en helt värdelös prediktion, då det är viktigare att kunna förutspå felkoderna.

Vi ser det som en möjlighet att förbättra vår modell genom att förbättra och berika datan. Förmågan att kunna träna artificiella neuronnätverk, samt möjligheten för en bra prestanda, är starkt beroende både på den kvantitativa och den kvalitativa tillgången av data.