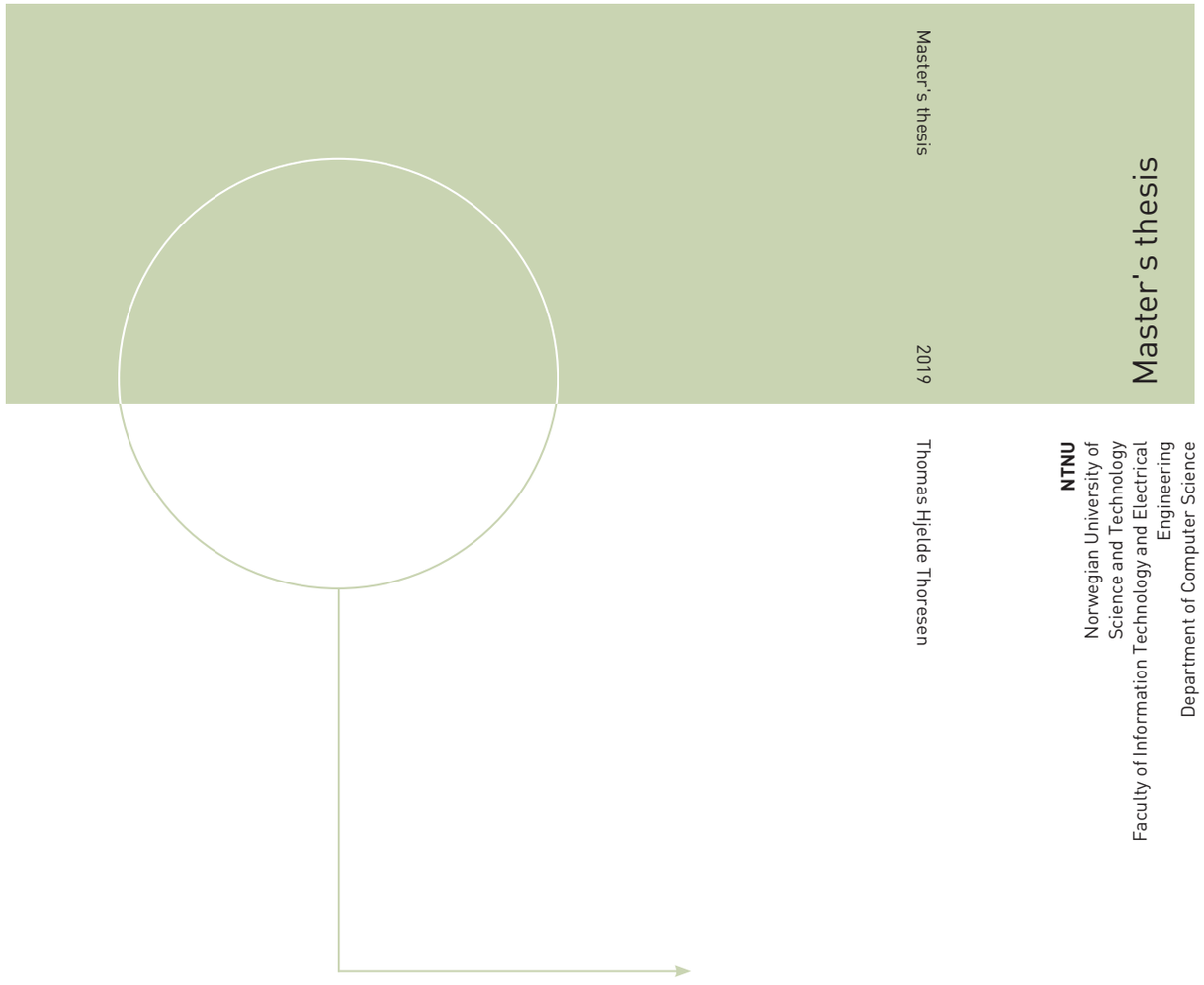


Master's thesis

2019

Thomas Hjelde Thoresen

**NTNU**
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Computer Science

Thomas Hjelde Thoresen

# Deep Learning for Fault Prediction in Offshore Wind Turbines

May 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU

Norwegian University of
Science and Technology

# Deep Learning for Fault Prediction in Offshore Wind Turbines

Artificial Intelligence Group
Submission date: May 2019
Supervisor: Professor Keith L. Downing
Co-supervisor: Vidar Slåtten, Ph.D., Equinor ASA

Norwegian University of Science and Technology
Department of Computer Science

# Abstract

This thesis investigates the application of a deep learning-based fault prediction system for offshore wind turbines based on SCADA data. The goal of the system is to provide early warning of faults, and thus reduce both the maintenance costs, as well as cost related to downtime.

The complete architecture of a fault prediction system is described and implemented, including the data sources, labelling procedure, data preprocessing, train/test/validation split procedure, classification component and alarm control system. The main focus of the thesis has been to evaluate the performance of different classifier components of the system.

Deep learning has shown great promise for time series classification problems, and several deep learning architectures are implemented as the classification component of the system. These are then compared to a random forest model. Such models are considered to be the state-of-the-art in this domain.

The deep learning models show better performance than the random forest model on all experiments. I show that the system is able to detect faults in advance, but the number of false alarms given in order to do so varies greatly with different fault categories. Some fault categories are found to be better suited for modelling than others, and large variations in results are also observed with regards to how long time in advance we want to be able to predict a fault. The performance of the investigated deep learning models varies across the fault categories, and it is not possible to declare a "winner" across the board.

Whether the amount of faults predicted vs. the false alarm rate justify building a production version of the system, must be further evaluated from a business perspective by domain experts.

The primary contribution of this thesis is the design and implementation of a complete fault prediction system for offshore wind turbines based on SCADA data. Additionally, different deep learning architectures have been evaluated as the classifier component of this system, and demonstrated ability to improve upon the state-of-the-art. Additional problem-specific insights are also presented, in order to facilitate future work in this area.

# Preface

This thesis concludes my Master of Science in Computer Science at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU). The supervisors for this thesis has been Professor Keith L. Downing, IDI, and Vidar Slåtten, PhD, Equinor ASA. In addition to my supervisors I would like to thank Jørgen Longva, Norconsult Informasjonssystemer AS, for providing feedback on writing style, as well as the New Energy Solutions department of Equinor ASA, for providing such an interesting dataset and important information about the wind turbine domain. My employer, Norconsult Informasjonssystemer AS, has also been very helpful and flexible with regards to the work on this thesis.

The last, but not least acknowledgement is for my incredible wife, May-Britt, for supporting my pursuit of this master's degree in every way.

Thomas Hjelde Thoresen
Trondheim, May 31, 2019

# Contents

**Abbreviations** **xiii**

**1 Introduction** **1**
1.1 Background and Motivation . . . . . 1
1.2 Goals and Research Questions . . . . . 3
1.3 Research Method . . . . . 3
1.4 Thesis Structure . . . . . 3

**2 Background Theory** **5**
2.1 Supervisory control and data acquisition (SCADA) data . . . . . 5
2.2 Fault prediction methods . . . . . 5
2.2.1 Forecast-based fault prediction . . . . . 6
2.2.2 Anomaly-based fault prediction . . . . . 6
2.2.3 Supervised fault prediction . . . . . 7
2.3 Capturing temporal features . . . . . 8
2.3.1 Adding manually crafted time-related features . . . . . 8
2.3.2 Adding lagged timesteps as features . . . . . 9
2.4 Domain specific feature engineering . . . . . 9
2.5 Time series classification . . . . . 9
2.5.1 Instance-based time series classification . . . . . 10
2.5.2 Feature-based time series classification . . . . . 11
2.6 Class imbalance . . . . . 11
2.6.1 Undersampling majority class . . . . . 12
2.6.2 Random oversampling of minority class . . . . . 12
2.6.3 Oversampling synthetic examples of minority class . . . . . 12
2.6.4 Algorithm-based handling of class imbalance . . . . . 12
2.6.5 Cost-sensitive learning . . . . . 12
2.7 Evaluation metrics . . . . . 13
2.7.1 Confusion matrix . . . . . 14
2.7.2 Accuracy . . . . . 14

2.7.3 Precision . . . 15
2.7.4 Recall . . . 15
2.7.5 F score . . . 15
2.7.6 Precision recall curve . . . 16
2.7.7 ROC-curve . . . 16
2.7.8 AUC . . . 17
2.8 Artificial Neural Networks (ANN) . . . 17
2.9 Autoencoder . . . 18
2.10 Deep Learning . . . 19
2.11 Convolutional Neural Networks (CNN) . . . 20
2.11.1 Squeeze-and-excitation networks . . . 21
2.12 Recurrent Neural Networks (RNN) . . . 22
2.12.1 Long Short-Term Memory (LSTM) Networks . . . 22
2.12.2 Gated Recurrent Unit (GRU) . . . 22
2.13 Entity embedding of categorical variables . . . 23
2.14 Regularization techniques . . . 23
2.14.1 Early stopping . . . 23
2.14.2 Dropout . . . 24
2.14.3 Weight regularization . . . 24
2.14.4 Activity regularization . . . 25
2.14.5 Batch normalization . . . 25
2.15 Chapter summary . . . 25

**3 Related work and motivation** **27**
3.1 Structured literature review protocol . . . 27
3.1.1 Search engines . . . 28
3.1.2 Search process . . . 28
3.1.3 Research questions (for search) . . . 28
3.1.4 Inclusion and evaluation criteria . . . 29
3.1.5 Data extraction . . . 29
3.2 Fault prediction in wind turbines from SCADA data . . . 30
3.2.1 Wind Turbine Fault Detection Based on SCADA Data Analysis Using ANN . . . 30
3.2.2 Automatic Fault Prediction of Wind Turbine Main Bearing Based on SCADA Data and Artificial Neural Network . . . 31
3.2.3 A Data-Driven Approach for Condition Monitoring of Wind Turbine Pitch Systems . . . 32
3.2.4 Wind Turbine Fault Detection Using Denoising Autoencoder with Temporal Information . . . 33
3.2.5 Wind Turbine Gearbox Failure Identification With Deep Neural Networks . . . 35

3.2.6 The Prediction and Diagnosis of Wind Turbine Faults . . . 35
3.2.7 Diagnostic Models for Wind Turbine Gearbox Components Using SCADA Time Series Data . . . 36
3.2.8 Learning Deep Representation of Imbalanced SCADA Data for Fault Detection of Wind Turbines . . . 36
3.2.9 Comparative Analysis of Neural network and regression based condition monitoring approaches for wind turbine fault detection . . . 37
3.2.10 A Robust Prescriptive Framework and Performance Metric for Diagnosing and Predicting Wind Turbine Faults Based on SCADA and Alarms Data with Case Study . . . 38
3.2.11 Summary and state-of-the-art . . . 38
3.3 Deep learning for Multivariate Time Series Classification . . . 41
3.3.1 Deep learning for time series classification: a review . . . 42
3.3.2 Time series classification from scratch with deep neural networks: A strong baseline . . . 42
3.3.3 An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling . . . 42
3.3.4 Section summary . . . 43
3.4 Chapter Summary . . . 45

**4 Architecture** **47**
4.1 Data Sources . . . 49
4.1.1 SCADA data . . . 50
4.1.2 Allocation data . . . 50
4.1.3 Alarm data . . . 52
4.2 Labelling Algorithm . . . 53
4.3 Training, Validation and Test Split . . . 54
4.4 Data Cleaning and Preprocessing . . . 56
4.4.1 Handling Missing Data . . . 56
4.4.2 Feature Scaling . . . 56
4.4.3 Feature Engineering . . . 57
4.4.4 Categorical Variable Encoding . . . 57
4.4.5 Sample Generation . . . 57
4.5 Alarm Control System . . . 58
4.6 Evaluation metric . . . 59
4.7 Deep Learning Models . . . 61
4.7.1 Output layer . . . 61
4.7.2 Loss function . . . 61
4.7.3 Diagram notation . . . 62
4.7.4 Common Embedding Module . . . 63

4.7.5 MLP Architecture . . . 64
4.7.6 CNN Architecture . . . 64
4.7.7 LSTM Architecture . . . 64
4.7.8 Hybrid Multivariate Time Series Network (HMVTS) Architecture . . . 64
4.7.9 Regularization Methods . . . 66
4.8 Chapter summary . . . 66

**5 Experiments and Technology** **71**
5.1 Hardware . . . 71
5.2 Software . . . 71
5.2.1 Pandas . . . 71
5.2.2 Scikit-learn . . . 72
5.2.3 TensorFlow . . . 72
5.2.4 Keras . . . 72
5.2.5 MLFlow . . . 72
5.3 Hyperparameters . . . 73
5.3.1 Baseline model: Random Forest . . . 73
5.3.2 Neural Networks . . . 73
5.4 Results . . . 74
5.4.1 Random Forest . . . 74
5.4.2 MLP . . . 75
5.4.3 CNN . . . 75
5.4.4 LSTM . . . 75
5.4.5 HMVTS . . . 76
5.5 Summary of results . . . 76
5.5.1 PF window comparison . . . 77
5.5.2 Fault category comparison . . . 77
5.5.3 Model comparison . . . 77
5.5.4 Metric comparison . . . 78
5.5.5 Practical implications . . . 80

**6 Evaluation and Conclusion** **81**
6.1 Thesis Summary . . . 81
6.2 Goal Evaluation . . . 82
6.3 Discussion . . . 82
6.4 Research contributions . . . 83
6.5 Future Work . . . 84

**Bibliography** **85**

**Appendices:**

**A Training history** **91**

**B ROC curves on test set** **97**

# List of Figures


2.1 Architecture of forecast based fault prediction system, adapted from Yang et al. [2018] . . . . . 6
2.2 Two types of time series classification methods . . . . . 11
2.3 Steps of SMOTE process, adapted from [Fawcett, 2016] . . . . . 13
2.4 Bagging with balanced classes, adapted from [Fawcett, 2016] . . . 14
2.5 Example of Precision recall-curve with Average Precision (AP) 0.88 16
2.6 Example of Receiver Operating Characteristic-curve . . . . . 17
2.7 Artificial Neural Network . . . . . 18
2.8 Autoencoder . . . . . 19
2.9 ANN with multiple hidden layers . . . . . 20
2.10 A convolution operation . . . . . 21
2.11 Squeeze-excitation-block, adapted from [Hu et al., 2018] . . . . . 21
2.12 Learned embedding representations of German states (reduced to 2D with t-SNE) [Guo and Berkhahn, 2016] . . . . . 24

3.1 Fault prediction procedure, adapted from [Zhang, 2018] . . . . . 32
3.2 Framework of data-driven fault prediction, adapted from [Yang et al., 2018] . . . . . 33
3.3 Monitoring results of pitch related faults [Yang et al., 2018] . . . . 33
3.4 Denoising autoencoder with temporal information, reproduced with permission from [Jiang et al., 2018] . . . . . 34
3.5 Learning embedding vectors [Chen et al., 2019] . . . . . 37
3.6 Overview of proposed framework, adapted from [Leahy et al., 2018] 39
3.7 Fully Convolutional Networks for Time Series Classification, adapted from [Wang et al., 2017b] . . . . . 43
3.8 Temporal Convolutional Networks [Bai et al., 2018] . . . . . 44
3.9 Residual connection in TCN [Bai et al., 2018] . . . . . 44

4.1 An overview of the proposed system . . . . . 48

4.2 An overview of the wind farm (Screenshot from Bazefield Software Platform) . . . 49
4.3 Distribution of faults per category . . . 51
4.4 Duration of allocations per category . . . 52
4.5 Number of allocations per month . . . 52
4.6 Illustration of sample generation with $w = 6$ . . . 58
4.7 Illustration of Alarm Control System . . . 60
4.8 Input Module with Embedding . . . 63
4.9 MLP Block Diagram . . . 65
4.10 CNN Block Diagram . . . 67
4.11 LSTM Block Diagram . . . 68
4.12 HMVTS Block Diagram . . . 69

5.1 F1 score for the fault categories on both PF windows. Note that the values for PF window = 6h is an order of magnitude smaller. . . 77
5.2 F1 score for the different models on both PF windows. Note that the values for PF window = 6h is an order of magnitude smaller. . 78
5.3 Model F1 rank on both PF windows . . . 79
5.4 AUC vs F1 score on test set . . . 80

1 Training history for MLP models on test set . . . 92
2 Training history for CNN models on test set . . . 93
3 Training history for LSTM models on test set . . . 94
4 Training history for HMVTS models on test set . . . 95

5 ROC curves for Random Forest models on test set . . . 98
6 ROC curves for MLP models on test set . . . 99
7 ROC curves for CNN models on test set . . . 100
8 ROC curves for LSTM models on test set . . . 101
9 ROC curves for HMVTS models on test set . . . 102

# List of Tables


2.1 Abbreviations used for evaluation metrics . . . . . 13
2.2 Confusion Matrix . . . . . 14

3.1 Search terms used for paper retrieval . . . . . 28
3.2 Inclusion and evaluation criteria . . . . . 29
3.3 Selected parameters from SCADA data. Table adapted from [Zhang and Wang, 2014] . . . . . 31
3.4 Summary of reviewed work . . . . . 40

4.1 Sample of SCADA data (obfuscated for data protection) . . . . . 50
4.2 Sample of available alarm data . . . . . 53
4.3 Result of labelling procedure . . . . . 54
4.4 Number of samples and labels in each set for the different fault categories . . . . . 56

5.1 Results RF . . . . . 74
5.2 Results MLP . . . . . 75
5.3 Results CNN . . . . . 75
5.4 Results LSTM . . . . . 76
5.5 Results HMVTS . . . . . 76
5.6 F1 score for the different models on each of the fault categories and PF windows. (Best score in bold) . . . . . 79

# Abbreviations

| | |
|---|---|
| **AE** | Autoencoder |
| **ANN** | Artificial Neural Network |
| **AUC** | Area under curve |
| **BN** | Batch Normalization |
| **CMS** | Condition Monitoring System |
| **CNN** | Convolutional Neural Network |
| **DAE** | Deep (or Denoising) Autoencoder |
| **DL** | Deep Learning |
| **DNN** | Deep Neural Network |
| **DTW** | Dynamic Time Warping |
| **EWMA** | Exponentially weighted moving average |
| **FCN** | Fully Convolutional Network |
| **FN** | False Negative |
| **FP** | False Positive |
| **LSTM** | Long-short term Memory |
| **MAPE** | Mean absolute percentage error |
| **ML** | Machine Learning |
| **MLP** | Multilayer Perceptron |
| **NLP** | Natural Language Processing |

**OM** Operations and Maintenance

**PCA** Principal Component Analysis

**PF** Pre-fault

**RF** Random Forest

**RMSE** Root Mean Squared Error

**RNN** Recurrent Neural Network

**ROC** Receiver Operating Characteristic

**RUL** Remaining Useful Lifetime

**ReLU** Rectified Linear Unit

**SCADA** Supervisory Control And Data Aquisition

**SDAPE** Standard deviation of average percentage errors

**SLR** Structured Literature Review

**SMOTE** Synthetic Minority Oversampling Technique

**SVM** Support Vector Machine

**TCN** Temporal Convolutional Networks

**TN** True Negative

**TP** True Positive

**TSC** Time Series Classification

**kNN** k-Nearest Neighbors

**t-SNE** t-Distributed Stochastic Neighbor Embedding

**tanh** Hyperbolic tangent

# Chapter 1

# Introduction

This chapter will provide the reader with an introduction to the thesis. First, I describe the motivation behind the thesis. Subsequently, the primary goal of the research, as well as the more specific research questions, will be presented. The main contributions and an overview of the structure of the thesis will form the final sections of this chapter.

## 1.1 Background and Motivation

Wind energy is a renewable energy source, whose installed capacity has drastically increased in recent years. Conti et al. [2016] argue that the total energy production from wind energy will increase by 5.7 percent each year on average until 2040. Wind turbines are commonly exposed to rough weather conditions and subject to frequent variations of operating conditions. Operations and maintenance costs may account for up to 30 per cent of the total cost of energy for offshore wind turbines [Tavner, 2012].

For offshore wind turbines, transportation of service personnel may be delayed several days because transportation is only feasible at certain transportation windows. The availability of these windows depend on several factors, such as the weather and availability of transportation vessel/helicopter. It may also take some time for spare parts to arrive. For these reasons, the idea of establishing reliable monitoring systems to assist in early fault detection is appealing. This may allow operators to take preventive actions, or to plan maintenance/repair in advance if a fault is unavoidable. The components of wind turbines with the highest fault rates are the rotor (especially the pitch system), transmission and power system [Pfaffel et al., 2017]. Several efforts have been made with the goal of predicting such faults in advance, and a wide range of machine learning

techniques have been tested in the process, with various degrees of success.

Fault prediction for wind turbines has been applied to mainly two types of data, data from supervisory control and data acquisition (SCADA) systems, and data from condition monitoring systems (CMS). CMSes are based on sensors that are installed for the sole purpose of monitoring the condition of a component or system. These sensors need to be installed separately. Wind turbine components have high reliability in general and operate at relatively low speeds, compared to components in other industries. This may offer an explanation of the low adaption of CMSes for wind turbines. In contrast, most wind turbines are equipped with a SCADA system by default, providing a great deal of information on the wind turbines operating conditions, sensors and status.

The SCADA system provides time-stamped measurements from the wind turbine sensors of different resolution. Since these data are already readily available for most turbines, the use of SCADA data for fault prediction is very appealing, and will be the basis of this thesis. It should be noted that there is no common taxonomy or naming convention of the SCADA signal names. This is all dependent upon the vendor of the system, and poses a potential obstacle towards generalization of research. For instance, some of the sensors that are emphasized in a paper may not necessarily be available for another wind turbine, and if they are available, they may not represent reality in the exact same way.

Traditional machine learning methods require some degree of manual feature engineering in order to capture inherent relationships between input variables. Additional feature engineering is required in order to capture relationships between data points at different timesteps. This introduces a need for domain knowledge in order to select and craft the most relevant features, without adding irrelevant ones. A brute force approach will increase the dimensionality of the samples, and make subsequent modelling more difficult.

Deep learning is a representation learning method that has turned out to be very good at discovering intricate structures in high-dimensional data, while circumventing the need for manual feature engineering [LeCun et al., 2015]. Representation learning means that it is able to learn multiple levels of representations, with slightly more complex, non-linear transformations at each level. With enough such transformations, very complex functions can be learned. Deep learning has recently achieved state-of-the-art results in in domains such as image recognition, speech recognition, translation, and other natural language processing applications.

## 1.2 Goals and Research Questions

In the previous section I explained the motivation for a fault prediction system based on SCADA data, and introduced the appeal of investigating the performance of deep learning architectures as the classifier component of such a system. This has led me to formulate the primary goal of the thesis as follows:

**Research goal** *Build a deep learning-based fault prediction system for offshore wind turbines based on SCADA data.*

We want to evaluate different deep learning models for the classifier component of the system. It is very interesting to find out how these models compare to the state-of-the-art in this domain. This leads to the following research questions:

**Research question 1** *How does deep learning methods perform in comparison to the state-of-the-art for predicting faults in wind turbines?*

**Research question 2** *Which of the investigated deep learning architectures are best suited as the classifier component in such a system?*

## 1.3 Research Method

In this project, software to implement the fault prediction system will be created, and empirical experiments will be designed and conducted to address the research questions. A generic system will be designed, and the deep learning methods as well as the state-of-the-art model will be used as the classifier component of the system and compared to each other. In that way, a fair comparison between methods may be performed. As a large dataset with known faults are provided, we are able to hold out a subset of the data (test data) from the models, and explicitly measure their ability to discriminate between normal operation and impending faults, and thus overall usefulness.

## 1.4 Thesis Structure

This first chapter of the report gives an introduction to the problem we are addressing in this thesis, and a motivation for why this problem needs to be solved. Chapter 2 provides some background theory needed to follow the rest of the thesis. Chapter 3 contains a survey of current literature and attempt to establish a state-of-the-art. Chapter 4 describes the architecture of our proposed system. Chapter 5 summarizes results of empirical experiments on a dataset provided by Equinor ASA. Finally, these results are discussed and evaluated in chapter 6.

# Chapter 2

# Background Theory

This chapter is intended to introduce the reader to the concepts that are necessary to understand the rest of the thesis. Please, note that this is not an in-depth explanation, as much as a brief foundation. In order to gain a deeper understanding of concepts introduced in this chapter, the referenced sources are a good starting point. I will start by presenting concepts related to the data that is used, as well as fault prediction in general. Next, I will give a brief introduction to time series classification, and the considerations that must be made when utilizing time series data for machine learning. The relevant evaluation metrics are also presented. To conclude the chapter, basic deep learning concepts and methods are introduced.

## 2.1 Supervisory control and data acquisition (SCADA) data

SCADA is a term common to a wide range of industrial processes, used to describe the system that gathers data collected from sensors related to the system. There are many different vendors that provide several kinds of SCADA systems. The data collected from these systems will often contain measurements along with a timestamp for each measurement, resulting in time series data.

## 2.2 Fault prediction methods

There exist different methods of predicting faults ahead of time. This section will group these methods in three categories, which will be referred to in later sections.

### 2.2.1 Forecast-based fault prediction

Forecast based fault prediction use one specific sensor of the SCADA data as target variable. Temperature sensors are commonly used, due to their correlation with certain types of faults. This sensor is then predicted using other sensors as input variables. It is critical that data indicating a fault is not included as training data, as the model is supposed to represent normal operating conditions. After the model is fitted to historical data, one can then compare the output of the fitted normal model to the actual values in real time. The residual errors between the expected output and actual output can then be used to indicate emerging faults. This may be considered a supervised learning method, since the model is fitted with a target variable, even though the target variable is not a direct indicator of a fault. Using this method, the need to train on actual historical faults are alleviated, as one builds a normal model, and detect deviations from this, rather than to try to model faults. If one does not have sufficient fault data readily available to train with, this method becomes especially attractive. An example of the architecture of a forecast-based fault prediction system can be seen in figure 2.1.

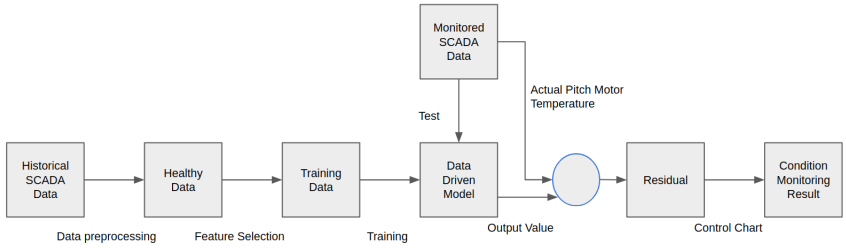



Figure 2.1: Architecture of forecast based fault prediction system, adapted from Yang et al. [2018]

### 2.2.2 Anomaly-based fault prediction

This methodology builds a model of normal behavior based on historical data as well. In contrast to forecast-based fault prediction, this method models the whole input signal, and seeks to detect anomalies based on the whole signal, rather than using specific sensors as a proxy for faults. If historical records of faults exist, these should be omitted from the training data, so that the normal behavior model is based on healthy data. The model is then trained to recreate

its own input as the output. For unseen data, anomalies may then be detected by computing the error between the expected output (which is the same as the input) and the actual output. This error may then be compared to the normal errors. If this error deviates from the normal, an anomaly has occurred. To determine what a deviation from the normal is, one needs to define a deviation threshold, for example three standard deviations for a normal distribution, which corresponds to ∼99.7 percent of the data being normal, and ∼0.3 percent being anomalous. A common challenge for anomaly-based methods is that most data that is different than what the model is trained on will represent an anomaly. This makes it difficult to filter out fault data from "non-fault, but new condition"-data.

### 2.2.3 Supervised fault prediction

To do supervised learning where the presence of a fault is the target variable, one needs explicit examples of faults. The number of faults per turbine per year is relatively low (from 10 to 50 based on observations on provided dataset). This number is extremely low compared to the number of samples from normal operation. This implies that the supervised approach to fault prediction is an imbalanced class problem, and must be handled with special caution. Several methods of handling this is described in section 2.6.

Another issue with supervised fault prediction, is the divergence in methodology for labelling of faults in the literature. Ideally, there would be a common way of labelling and storing fault data for all turbine monitoring systems. This is not the case however. In the dataset used for this project, there are fault records, called allocations. The fault allocations are a combination of automatically labelled faults from the SCADA system and manually labelled faults. There were also many stops that did not have a corresponding record in the allocation table. Additionally, there are many potential pitfalls, and different strategies that may be used when labelling data. One approach for labelling is to try to predict remaining useful lifetime (RUL). This would imply labelling a sample with the time until failure, making the problem a regression problem. This method is well suited when the failures belong to one common component. With several different components, the label, and thus also the targets for training a machine learning algorithm, would need to be the RUL of each of the possible components.

Another approach is to use a binary label, framing the fault prediction problem as a classification problem. With this approach, an important decision to make is whether one is trying to predict exactly when a fault occurs, or if the turbine is in a pre-fault state. The former would imply labelling only the sample before the fault occurs, while the latter would imply labelling all samples in a window, $m$, before the fault occurs as pre-fault samples. The predictions must be interpreted accordingly. If the samples are labelled as pre-fault data, the pre-

dictions would indicate the probability of being in a pre-fault state. In contrast, if the samples are the labelled where the actual fault occurs, we can make a prediction which will be interpreted as "the probability that a fault will occur at the next timestep", and the fault prediction horizon will be very short. The former will likely be of limited value, as actions to mitigate the fault would need to be taken before the next timestep.

Generally, one will also have to throw away all training samples where faults already have occurred, as it is not useful to predict a fault that has already occurred. All these considerations must be taken into account to ensure that the evaluation metrics will be fair, and will be useful in a real world system.

## 2.3 Capturing temporal features

The SCADA dataset consists of timestamped measurements that are temporally related. Still, out of the surveyed literature that takes a supervised approach to the fault prediction problem, only one paper makes use of some kind of time-related features from previous observations, and they just add the lagged value of one timestep of selected variables as additional features.

A possible reason for this, is that the number of features might grow very fast, and a problem known as "the curse of dimensionality" will arise [Domingos, 2012]. This problem will introduce the need for a dimensionality reduction method, for example Principal Component Analysis (PCA) Jolliffe [2011], in order to make use of most machine learning algorithms.

When considering the objective of predicting a fault based on this data, it is fair to assume that not only the sensor values at that specific time will be relevant in order to determine if a fault is imminent, but also the temporal features[1] of the time series up to that point (see Chapter 3). For illustration, consider an oil pressure sensor. It is likely that the trend of the readings from this sensor will be an important factor in order to predict a related fault. At the very least, it is not unlikely that capturing the trend will enable an earlier prediction of the fault. There are several ways of capturing such temporal information, as we will see in the following sections.

### 2.3.1 Adding manually crafted time-related features

One way to capture temporal information is to consider each observation by itself, but add manually crafted time-related features. An example would be to add rolling mean for different time windows, or to add the difference between current and past observations. In Fulcher and Jones [2014], 25 time-related features are

[1] In this context, temporal features are used to describe properties such as trend, time derivatives and time integrals over a single feature or combinations of features

added, and are shown to improve classification performance considerably. In order to select those features that are most relevant, domain knowledge is an advantage. The number of features can grow very large if one is not selective about which features are added.

### 2.3.2 Adding lagged timesteps as features

Another approach is to simply add previous observations as features, and let the model try to figure out possible relationships between the observations. However, this method also suffers from a quickly growing number of features. Consider the case that we would want to capture the development over the past day for all variables. With 10-minute-interval samples, this would mean $6 \cdot 24 = 144$ previous timesteps per feature, which would quickly lead to a large number of features, and increase the complexity of working with the dataset.

## 2.4 Domain specific feature engineering

With the incorporation of domain knowledge, several additional features might be crafted based on combinations and/or transformations of original features. Examples of potentially useful engineered features for the wind turbine diagnostics domain is the difference in temperature between front and rear bearing or the ratio between active power and available power. Hu et al. [2016] have done research on domain specific feature engineering for wind turbine diagnostics, and show that a prediction accuracy increase of 27% was gained by adding manually crafted features to the observed SCADA data. Feature engineering introduces the need for domain knowledge, to be able to craft relevant features. There are also frameworks for automated feature engineering. This has been described as "One of the holy grails of machine learning is to automate more and more of the feature engineering process" [Domingos, 2012]. One such framework is *featuretools*, based on the techniques described in Kanter and Veeramachaneni [2015]. A common drawback for these approaches is that they are computationally expensive, especially as the number of samples and original features grow large.

## 2.5 Time series classification

Time series classification is the process of classifying a *sequence* of observations as belonging to a specific category. This can be seen in contrast to the cases above, where each observation is classified independently. We introduce a parameter $n$, denoting the number of observations that together make up one sample. This parameter will be referred to as window size. For illustration, let us consider

the following individual observations, where each row is an observation, with the rightmost column representing the dependent variable that is produced by the previous $n$ observations:

$$\begin{bmatrix} x_1^0 & x_2^0 & x_3^0 & x_4^0 & x_5^0 & NaN \\ x_1^1 & x_2^1 & x_3^1 & x_4^1 & x_5^1 & NaN \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 & NaN \\ x_1^3 & x_2^3 & x_3^3 & x_4^3 & x_5^3 & y_0 \\ x_1^4 & x_2^4 & x_3^4 & x_4^4 & x_5^4 & y_1 \end{bmatrix}$$

NaN is used to denote "Not a number", and is used to indicate that there are no corresponding output for the n first observations, as they do not make up a whole example. Below is an example of how this would look like framed as a time series with a window size, n, of 3. This means that we will get (number of total observations-n) training examples, in this case, two. The training examples would look like this, with $\hat{x}$ representing the inputs, and $\hat{y}$ representing the output.

$$\hat{x}(0) = \begin{bmatrix} x_1^0 & x_2^0 & x_3^0 & x_4^0 & x_5^0 \\ x_1^1 & x_2^1 & x_3^1 & x_4^1 & x_5^1 \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 \end{bmatrix}, \hat{y}(0) = y_0$$

$$\hat{x}(1) = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & x_4^1 & x_5^1 \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 \\ x_1^3 & x_2^3 & x_3^3 & x_4^3 & x_5^3 \end{bmatrix}, \hat{y}(1) = y_1$$

The most common approaches for such time series classification will be discussed in the next sections.

### 2.5.1 Instance-based time series classification

In this approach, each time series (which might be multivariate) is classified by measuring the distance to other samples. One algorithm that uses this approach is the simple "lock step"-distance depicted in figure 2.2 A. Another more commonly used algorithm is Dynamic Time Warping (DTW), which can accommodate unaligned patterns in the time series, i.e. if a pattern is recognized at one point in the sequence, the same pattern can be used for recognizing an unseen sequence even if the pattern occurs at a different point in the sequence.

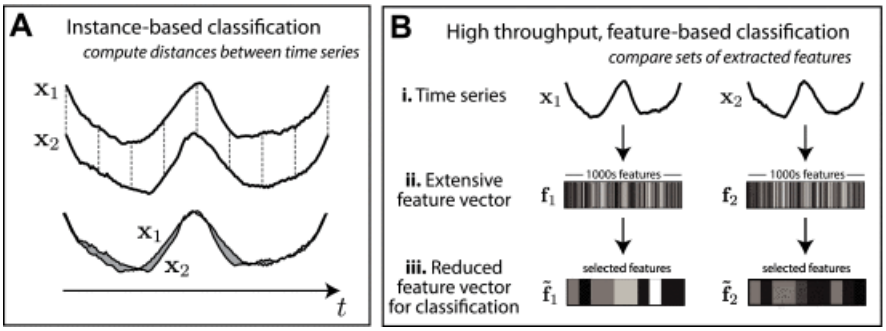



Figure 2.2: Two types of time series classification methods

### 2.5.2 Feature-based time series classification

In this case, a large number of features of the time series are extracted, and added as additional features. In order to only use the features with most explanatory power, some kind of feature selection is normally done before a machine learning algorithm is applied. There exists a wide variety of feature selection algorithms, which are typically divided into the following categories:

1. Filter methods. These methods remove the features that are least interesting based on general features, such as correlation with the dependent variable.

2. Wrapper methods. These evaluate a subset of the variables, and therefore allow for interactions between variables to be a part of the evaluation. This increases computation time compared to filter methods.

3. Embedded methods. These are models that incorporate some kind of feature selection internally, with a decision tree model as an example.

## 2.6 Class imbalance

Class imbalance is a term used to describe classification cases where the number of examples of each class are not evenly distributed. Supervised fault prediction represents a problem where we face this issue. This section will provide an overview of commonly used methods for handling the class imbalance problem. For a more thorough review, the reader is referred to Branco et al. [2016].

### 2.6.1 Undersampling majority class

This method aims to balance class distribution by randomly eliminating samples from the majority class. A major drawback of this method is that potentially useful data is thrown away.

### 2.6.2 Random oversampling of minority class

This method aims to balance class distribution by randomly duplicating samples from the minority class. A major drawback of this method is that it is very likely to lead to overfitting.[2]

### 2.6.3 Oversampling synthetic examples of minority class

Several methods exist that seek to generate synthetic examples of the minority class. The most commonly used is SMOTE (Synthetic Minority Oversampling TEchnique) [Chawla et al., 2002], that generates new samples by interpolating between samples drawn from the minority class. The method is well illustrated by figure 2.3.

Several improvements and variations of SMOTE have been proposed in recent years. These will not be covered in detail here.

### 2.6.4 Algorithm-based handling of class imbalance

Some ensemble learning algorithms are well suited to handle the imbalance problem due to their ability to learn from several weak classifiers. An example is the use of bagging of decision tree classifiers, where the sample used by each decision tree is balanced, as can be seen in figure 2.4.

### 2.6.5 Cost-sensitive learning

Another approach for handling the imbalanced class problem is to incorporate different cost for misclassification of each class. To illustrate, we can use the example of a dataset with 1% positive samples, and 99% negative samples. The cost of misclassifying a positive example can then be set to 99 times that of misclassifying a negative example. In this way, a classifier will be forced to learn to classify the positive class, in order to avoid a large cost.

[2] Overfitting is a term used to describe the situation where the noise in the training data is modelled, rather than the underlying patterns, which leads to poor generalization to unseen data.

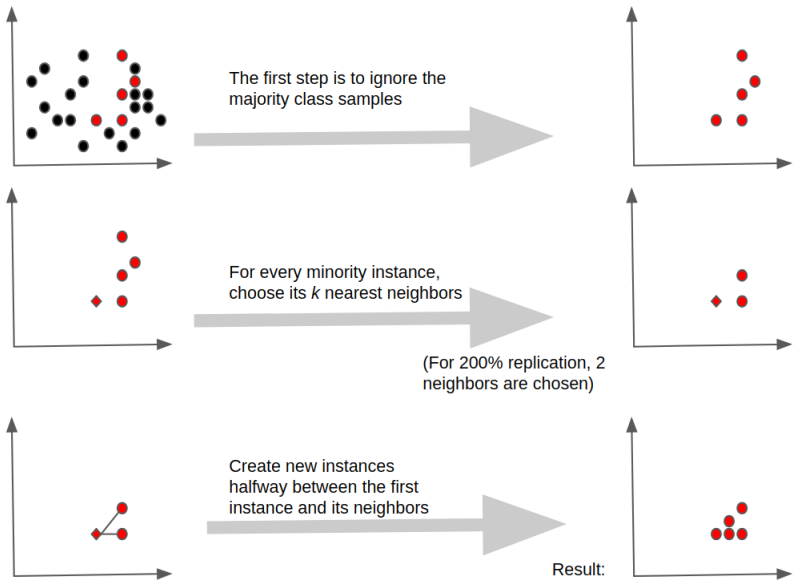



Figure 2.3: Steps of SMOTE process, adapted from [Fawcett, 2016]

## 2.7 Evaluation metrics

This section will provide some insights into commonly used evaluation metrics for binary classification. We make use of the common abbreviations listed in table 2.1.

Table 2.1: Abbreviations used for evaluation metrics

| Abbr. | Meaning |
|---|---|
| TP | True Positive |
| TN | True Negative |
| FP | False Positive |
| FN | False Negative |

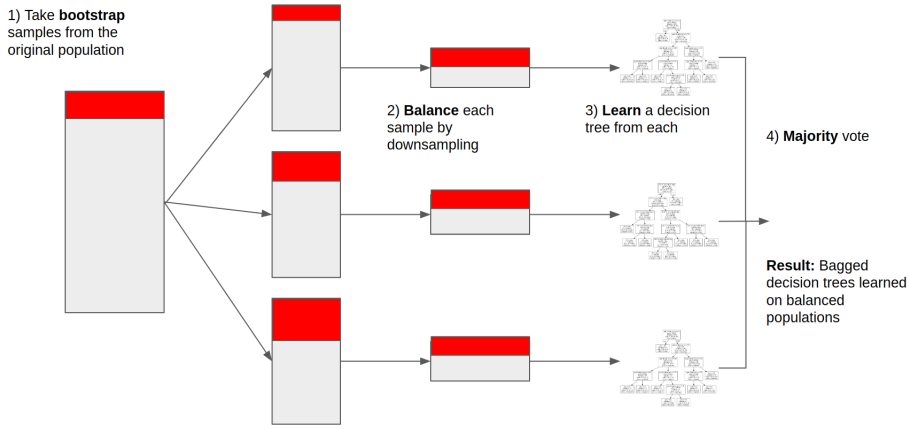



Figure 2.4: Bagging with balanced classes, adapted from [Fawcett, 2016]

### 2.7.1 Confusion matrix

A confusion matrix is a matrix used to visualize the predictions from a classifier against the actual labels.

Table 2.2: Confusion Matrix

| | Actual Positives | Actual Negatives |
|---|---|---|
| Predicted Positives | TP | FP |
| Predicted Negatives | FN | TN |

Several other metrics can be derived from the values of the confusion matrix, as we will see in the next sections.

### 2.7.2 Accuracy

This is a metric that describes what fraction of the classifications that are correct.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

This metric is not well suited with imbalanced classes. To illustrate, we can think of the example where we have only 1% of the samples that are positive. A classifier that classifies everything as negative, will then score a 99% accuracy, even though it is not particularly useful.

### 2.7.3 Precision

Precision is a metric that displays how many of the positively classified samples that actually were positive.

$Precision = \frac{TP}{TP+FP}$

Also known as sensitivity.

### 2.7.4 Recall

Recall is a metric that displays how many of the total positive samples that were classified as positive.
$Recall = \frac{TP}{TP+FN}$

Also known as specificity.

### 2.7.5 F score

$F$ score is a metric that combines precision and recall to give a score based on both. $F_1$ score is the special case of the F score that computes the harmonic mean of precision and recall, i.e. both are given equal weight. Often, one might want to give more weight to either precision or recall. To illustrate, we can think of an example related to the fault prediction domain. If it is more important that all alarms are detected than reducing the number of false alarms, we want to put more weight to recall. In the opposite case, if we want to give more weight that our predicted faults are actual faults, and reducing the number of false alarms (and also the number of faults detected), we can give more weight to precision.

$F_1 = 2 * \frac{precision*recall}{precision+recall}$.

The general case has a parameter $\beta$, which can be adjusted to give more weight to either precision ($\beta < 1$) or recall ($\beta > 1$).

$F_\beta = (1 + \beta^2)\frac{precision*recall}{(\beta^2*precision)+recall}$.

### 2.7.6 Precision recall curve

This is a plot that illustrates the trade-off between precision and recall, and may provide insight into the performance of the classifier at different levels. We can see that as recall increases, i.e. a larger fraction of the true positives are accurately classified, the precision will gradually decrease. An example of a precision recall curve can be seen in figure 2.5.

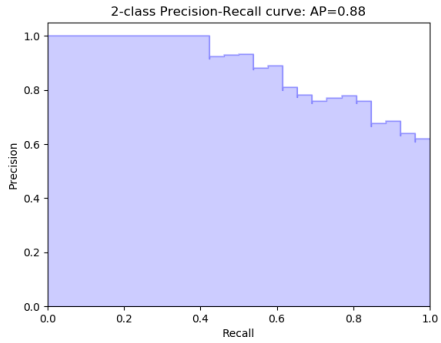



Figure 2.5: Example of Precision recall-curve with Average Precision (AP) 0.88

### 2.7.7 ROC-curve

The receiver operating characteristic curve (ROC) [Fawcett, 2006] is a plot designed to illustrate the diagnostic ability of a binary classifier. It is created by plotting the true positive rate (TPR) vs. the false positive rate (FPR) at different threshold settings.
TPR is given by
$TPR = \frac{TP}{TP+FN}$, while

FPR is given by
$FPR = \frac{FP}{FP+TN}$

If the ROC curve has a steep curve at the beginning it means that many of the predictions with highest probabilities are correct. The performance of random chance will form a diagonal line in the ROC-curve. An example of a ROC-curve can be seen in figure 2.6.

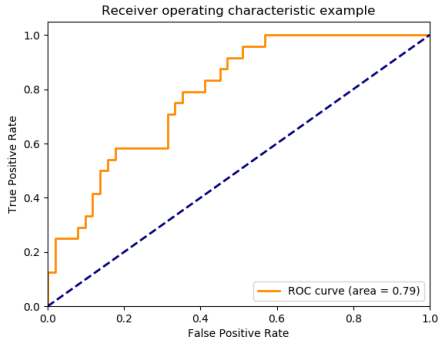



Figure 2.6: Example of Receiver Operating Characteristic-curve

An important point is that the ROC curve may lead to an overly optimistic picture of the classifier's skill in the case of very imbalanced classes. The reason for this is the inclusion of the number of true negative samples in the FPR. [Davis and Goadrich, 2006].

### 2.7.8 AUC

Area under curve (AUC) is a metric used in combination with the ROC-curve, where the area under the ROC-curve is calculated. An AUC of 0.5 corresponds to the expected performance of a random classifier, and an AUC of 1 describes a perfect classifier.

## 2.8 Artificial Neural Networks (ANN)

An artificial neural network (ANN) is a computational model inspired by the human brain. The key concept is that the hidden units in the network has parameters called weights and biases. The output of a given node is determined by its input, bias, weight and the activation function. The activation function is typically a non-linear mathematical function, with sigmoid, the hyperbolic tangent (tanh) and rectified linear unit (ReLU) [Nair and Hinton, 2010] being common examples. The weights and biases are iteratively adjusted to optimize a loss function. This loss function is calculated based on the predictions given by the network and the expected output. The derivative of the loss function is calculated and propagated back through the network, in a process called backpropagation. The parameters are thus adjusted according to how much they contributed to

the error. This optimization method is called gradient descent. An illustration of an ANN can be seen in figure 2.7.

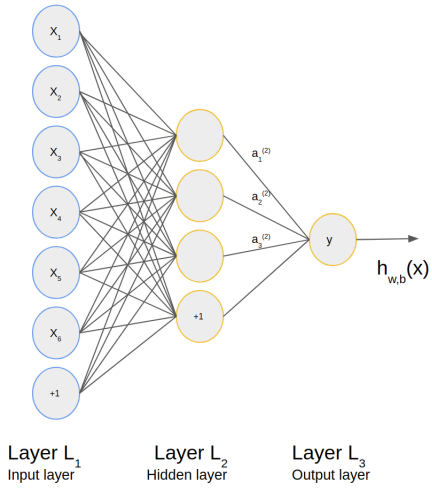



Figure 2.7: Artificial Neural Network

In the figure, $x_i$ represent the i'th input variable, $a_i^l$ is the activation for the i'th unit at layer l, and the circles labeled "+1" are the bias units. $h_{W,b}(x)$ is the network's hypothesis, or output, given the input, weights and biases.

The most common way of training neural networks today is by training the network iteratively with *batches* of training samples, where the weights and biases are updated after each batch. When all the samples in the training set have been used for training once, we say that the network has been trained for one *epoch*. An important hyperparameter to choose before training is the *learning rate*. The learning rate controls how much the weights are updated on each iteration. Several optimizer algorithms have been suggested to dynamically adjust the learning rate during training. One such optimizer, that is commonly used, is the Adam optimizer introduced by Kingma and Ba [2014]. Adam uses moving averages of the weights and biases instead of absolute updates on each iteration. This has been shown to increase performance of neural networks.

## 2.9 Autoencoder

An autoencoder is a special case of ANN, which is trained to recreate the same output as it is fed in. In this way, the autoencoder automatically learns fea-

tures of unlabeled data. This can be useful in a number of ways, for example as a dimensionality reduction method, which in turn enables a more effective distance calculation between samples. Distance calculation may be used to produce clusters with samples in proximity to each other. Autoencoders have been successfully used for anomaly detection [Sakurada and Yairi, 2014]. This is done by comparing the output produced by the autoencoder with the expected output (which also is the input). If the error between the two is higher than previous errors, we can say that the sample is anomalous as it differs to some the degree from the samples that the autoencoder previously has been trained on. A basic example of an autoencoder is shown in figure 2.8.

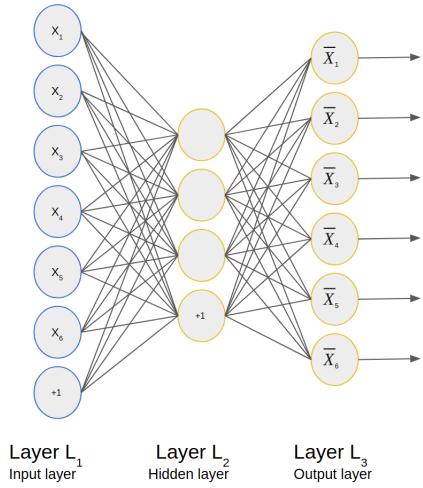



Figure 2.8: Autoencoder

## 2.10 Deep Learning

Deep learning is a term used to describe Artificial Neural Networks with several hidden layers. Multiple layers enable the neural network to learn increasingly complex representations of the data. Deep learning has dramatically improved the state-of-the-art of several problem domains in recent years [LeCun et al., 2015]. A visualization of an ANN with multiple hidden layers can be seen in 2.9.

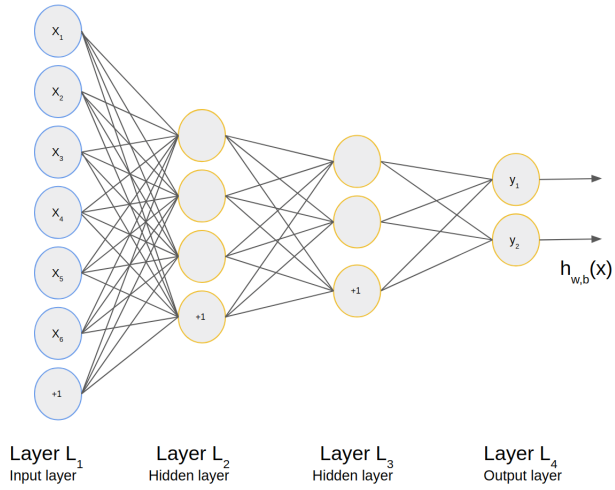



Figure 2.9: ANN with multiple hidden layers

## 2.11 Convolutional Neural Networks (CNN)

Convolutional neural networks are very similar to ordinary neural networks described in the previous section. The difference is that convolutional neural networks are designed to process data that come in the form of multiple arrays, such as 1D for signals, sequences and language, 2D for images or audio spectrograms and 3D for video or volumetric images [LeCun et al., 2015]. Note that images may also be 3D if they consists of multiple *channels*, as is the case with a typical color image, with the pixel value of each of the 3 channels (RGB) represents a 2D matrix, which become 3D when stacked together. Convolution is a mathematical operation that is best explained by the visual explanation in figure 2.10, where a *kernel* (the 3x3 dark blue matrix) slides over input data (the 5x5 blue matrix), and a matrix multiplication is done to produce an *output* (the green 3x3 matrix). The subscripted numbers are the values of the kernel matrix. The kernel's parameters (weights) are shared between all its inputs. This reduces the number of total parameters, and enable CNNs to learn features independent of location. This is best illustrated by considering images, where a feature (for example the face of a cat) can be learned no matter where in the image it occurs.

Dumoulin and Visin [2016] wrote an excellent paper with visual explanations of different convolutions and their parameters, which is recommended for readers interested in a deeper understanding of this topic.

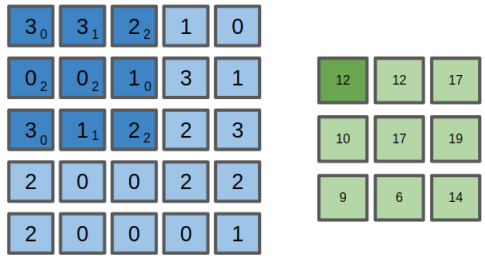



Figure 2.10: A convolution operation

### 2.11.1 Squeeze-and-excitation networks

Introduced by Hu et al. [2018], squeeze-and-excitation networks are an architectural unit of a CNN that explicitly models the relationship and interdependencies between channels. They have been shown to increase performance of CNN's without introducing significant computational cost.

The components of a squeeze-and-excitation block can be seen in 2.11

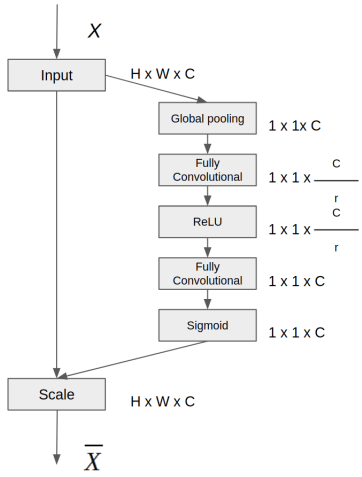



Figure 2.11: Squeeze-excitation-block, adapted from [Hu et al., 2018]

The *squeeze* part is performed by squeezing global information into a chan-

nel descriptor by using global average pooling to generate channel-wise statistics. This information is then exploited by the *excite*-block, which is capable of learning a non-linear and non-mutually exclusive interaction between channels (multiple channels should be emphasized). Two fully connected layers are wrapped around the non-linear component and act as a dimensionality-reduction and dimensionality-increasing layer, respectively.

## 2.12 Recurrent Neural Networks (RNN)

Recurrent neural networks are networks designed especially for sequence modelling, which were introduced in the 1980's by Hopfield [1982]. They have one or more feedback connections, which means that parameters can be shared among different parts of the architecture. This enables the network to extract important information from a sequence regardless of which position the information is located at. This is normally done by storing the output at one part of the sequence, and using this output to update weights at an earlier part of the network. Basic RNN architectures are notoriously difficult to train, as described in detail in Pascanu et al. [2013]. Several improvements have been suggested to remedy this, and two of the most prominent ones will be described in the next sections.

### 2.12.1 Long Short-Term Memory (LSTM) Networks

LSTMs were proposed in Gers et al. [1999] to make RNNs simpler to train. In addition to the cell/unit itself, an LSTM-unit consists of three gates, the input gate, the output gate, and the forget gate. The input gate decides which input are accepted, the output gate controls the output, and the forget gate controls what should be forgotten by the unit. These small control mechanisms allow the LSTM to forget irrelevant information, and make use of the information that is considered most relevant across several timesteps.

### 2.12.2 Gated Recurrent Unit (GRU)

This is another RNN-variation that is quite similar to the LSTM. GRU uses two kind of gates, the update gate and the reset gate. The update gates determine how much of the hidden state should be kept, and perform approximately the same task as the input and forget gate of the LSTM. In contrast to the LSTM, which contains both hidden state and cell state, the GRU only keeps the hidden state.

## 2.13 Entity embedding of categorical variables

The concept of using embedding vectors to represent categorical variables is known from the Natural Language Processing (NLP) domain, where words are represented with an embedding vector instead of a constant value. The embedding vector is initiated with random weights and learned similar to the weights of a hidden layer in a neural network. This method enables the network to learn that some words may yield a similar output most of the time, and their embedding vector values will be closer to each other in a euclidean space than words that do not yield similar output. Mikolov et al. [2013] demonstrated the effectiveness of learning vector representations of words. The same concept may also be applied to other categorical variables, and will be particularly useful if we expect some of the possible values that a categorical variable may take to be more similar than others. Consider using day of week as a categorical variable in the context of predicting sales. We would probably expect Wednesday and Thursday to be more similar than Wednesday and Sunday. This would likely be learned by using an embedding for the categorical variable. In Guo and Berkhahn [2016], the authors demonstrate the effectiveness of encoding a categorical variable into an embedding vector by using the method in a Kaggle Competition, with the goal of predicting future sales for Rossmann Stores[3] and achieved an impressive 3rd place with relatively little feature engineering. One of the categorical variables for which the method was applied to, was which German state the stores were located in. After performing a dimensionality reduction of the learned embeddings with the t-SNE algorithm [Maaten and Hinton, 2008], the state representations were plotted in 2D. This plot resembled the German map surprisingly well.

## 2.14 Regularization techniques

An important problem in machine learning is to make an algorithm that will generalize to unseen data, and not just perform well on the training data. Regularization is in Goodfellow et al. [2016] defined as any modification made to a learning algorithm with the intention of reducing its generalization error but not its training error. There are many options for regularization in deep learning, and this section will briefly describe the most common ones related to this thesis.

### 2.14.1 Early stopping

When training neural networks with a sufficient capacity (number of parameters) to overfit to the training data, we often observe that performance on training data

[3] https://www.kaggle.com/c/rossmann-store-sales

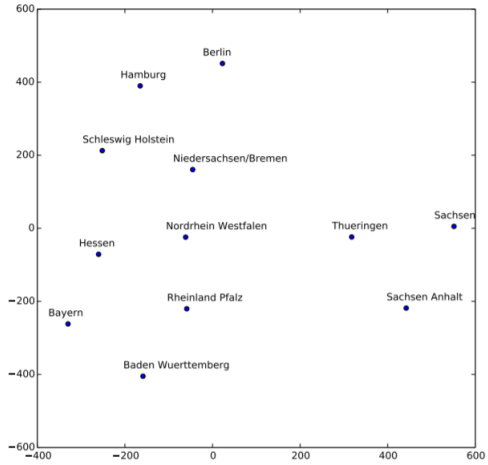



Figure 2.12: Learned embedding representations of German states (reduced to 2D with t-SNE) [Guo and Berkhahn, 2016]

decreases over time, while performance on validation data starts to increase when overfitting occurs. By employing early stopping, the algorithm keeps track of the performance on the validation set as well as the learned weights, after each epoch of training. After training has ended, the weights of the network are restored to the weights that gave the best performance on the validation data. In Goodfellow et al. [2016], early stopping is recommended almost universally.

### 2.14.2 Dropout

Dropout was introduced by Srivastava et al. [2014], and provides a simple and powerful approach to regularizing neural networks. It works by randomly selecting a subset of the hidden units that are blocked from learning at each weight update. This forces features to be learned across several pathways of the network. Dropout is computationally cheap and has been shown to provide effective regularization.

### 2.14.3 Weight regularization

This method adds a penalty to the loss function based on the magnitude of the weights. This encourages the network to keep the weights small. Commonly used methods are L1-regularization, which penalizes the absolute magnitude of the weights, while L2-regularization penalizes the squared magnitude of the weights.

An alternative is to combine L1 and L2-regularization.

### 2.14.4 Activity regularization

This technique is similar to weight regularization, except the model is penalized during training based on the magnitude of the activations. The same regularization methods may be used.

### 2.14.5 Batch normalization

Batch normalization was introduced by Ioffe and Szegedy [2015], and addresses the problem of coordinating simultaneous weight updates across multiple layers. This is achieved by standardizing the activation between layers per mini batch that the network is trained on. This has been shown to reduce training time, and has potential to act as a regularization method according to Goodfellow et al. [2016].

## 2.15 Chapter summary

This chapter has introduced the theoretical concepts necessary for the reader to follow the rest of the thesis, with concepts from both the wind turbine-specific domain, time series classification, machine learning in general, as well as deep learning-specific concepts. It should be noted that several of these domains are highly active at the present time, and new concepts that could prove relevant to this thesis might have been introduced by the time the thesis is finalized.

# Chapter 3

# Related work and motivation

This chapter surveys related work in the literature relevant to this project. The fields of related work are divided in two main categories. The first category is the domain-specific field of using SCADA data for fault prediction in wind turbines. The second category is the more general field of deep learning for multivariate time series classification. I argue that a review of both categories of work is important in order to gain a high-level overview of all research that is relevant to this project. The first category of related work will be reviewed by following a structured literature review (SLR), while the more general domain will be reviewed by using two recent review papers as starting point, with the most relevant referenced papers added as needed.

The aim of this chapter is to establish the current state-of the-art of the specific field of failure prediction in wind turbines based on SCADA data, and extract relevant insights from the general field of deep learning for time series classification. Together, these findings will form a motivation for the design of an improved deep learning-based system for failure prediction in wind turbines based on SCADA data, which will be presented in the next chapter.

## 3.1 Structured literature review protocol

In order to extract the most relevant information for the research questions in this thesis, a Structured Literature Review (SLR) was performed. This involves following a set of predefined steps to retrieve the most relevant related work, and extract the most useful information from these studies. The advantages of an

Table 3.1: Search terms used for paper retrieval

| Group 1 | Group 2 | Group 3 | Group 4 |
|---|---|---|---|
| Failure prediction | machine learning | wind turbine | SCADA |
| Fault prediction | neural network | | |
| Predictive maintenance | deep learning | | |

SLR include reproducability, avoiding bias, and to gain a broad overview of the field.

This section will describe the SLR protocol used to review literature for this thesis. This protocol was developed and used by the author in the process of extracting information from previous work related to this thesis. The SLR protocol was developed based on materials and template provided by Anders Kofod-Petersen, Adjunct Professor at the Department of Computer and Information Science at the Norwegian University of Science and Technology.

### 3.1.1 Search engines

The following search engines were used to search for relevant literature. One search in a domain-specific journal (Renewable Energy) is added along with two well-known search engines.

- IEEE
- Google Scholar
- Science Direct (with selected Journal: Renewable Energy)

### 3.1.2 Search process

The search terms used to generate the search queries can be seen in table 3.1.

The queries are generated by combining one term (T) from each group (G).

([G1,T1] OR [G1,T2] OR [G1,T3] OR [G1,T4]) AND ([G2,T1] OR [G2,T2] OR [G2,T3]) AND [G3,T1] AND [G4,T1].

From these queries, 494 papers were identified.

### 3.1.3 Research questions (for search)

The following questions are identified and sought to be addressed through our literature review:

Table 3.2: Inclusion and evaluation criteria

| Criteria identification | Criteria |
|---|---|
| IC1 | The study's main concern is fault prediction in wind turbines based on SCADA data. |
| IC2 | The study is not a review paper, and presents empirical results on real-world data. |
| IC3 | The study use SCADA data. |
| IC4 | The study describes a complete solution for fault prediction |
| QC1 | There is a clear statement of the aim of the research. |
| QC2 | The study is put into context of other studies and research. |

**Research question 1** *What are the existing solutions for predicting faults in wind turbines based on SCADA data?*

**Research question 2** *How do the different methods found by addressing RQ1 compare to each other.*

**Research question 3** *What is the strength of the evidence in support of the different solutions?*

**Research question 4** *What are the implications of these findings that must be considered when designing an improved system based on deep learning for fault predictions in wind turbines based on SCADA data.*

### 3.1.4 Inclusion and evaluation criteria

In the table below, the criteria for selecting papers included for review are presented. Papers were first filtered by title, then by reading abstracts, and finally by reading the full text.

A total of 9 papers were included after these criteria was applied.

### 3.1.5 Data extraction

The following data points were extracted from each of the papers reviewed.

1. Name of author(s)
2. Title

3. Study identifier
4. Year of publication
5. Number of samples
6. Number of features
7. Approach
8. Method (Best if several)
9. Does the study capture time features?
10. Evaluation and results
11. Are false alarms evaluated?

These categories help form a taxonomy which we can use to compare important characteristics of the reviewed papers, and will be helpful to present the findings of the reviews in a structured manner.

## 3.2 Fault prediction in wind turbines from SCADA data

This section will present the results of the SLR for the field of fault prediction for wind turbines based on SCADA data. The section is divided into one subsection for each of the papers that were selected for review. A brief summary of the reviewed papers will be presented at the end of the section.

### 3.2.1 Wind Turbine Fault Detection Based on SCADA Data Analysis Using ANN

In this paper, Zhang and Wang [2014] used SCADA data collected from a turbine at a wind farm located on Hundhammerfjellet, owned by NTE - Nord-Trøndelag Elektrisitetsverk. The turbine is directly-driven, which means that power is generated directly from the main shaft, with no gearbox in between. A forecast-based fault prediction methodology is adopted. The target variable is the turbine rear bearing temperature. The authors argue that this is a good proxy to predict faults related to bearing overheating. A straightforward threshold check to flag temperatures exceeding pre-set levels is already applied, but the goal in this paper is to detect such overheating in advance, so that damage-reducing actions may be taken. Data was collected from one turbine from April 22, 2009 to July

| Model output | Input |
|---|---|
| Rear bearing temperature | Rear bearing temperature (t-1) |
| | Active power output (t) |
| | Nacelle temperature |
| | Turbine speed (t) |

Table 3.3: Selected parameters from SCADA data. Table adapted from [Zhang and Wang, 2014]

21, 2009. This amounts to roughly 13000 data points. The variables used in this paper can be seen in table 3.3. This is only a small subset of all variables available in the SCADA data

An ANN was used to model the bearing temperature. The architecture of the ANN was found by cross-validation, and three hidden layers with 5, 10 and 1 hidden units respectively were used for the final model. They trained the network for 1000 epochs. Since this is a forecast-based fault prediction method, predicted output is compared with actual output, and the difference is used as a deviation metric, which may trigger a fault prediction if pre-set thresholds are exceeded. The ANN is trained to the point where a RMSE of 0.2 degrees Celsius is achieved. Testing on one actual fault shows that their method is able to detect a fault in the main bearing 10 days in advance, but the number of false positives are not mentioned.

### 3.2.2 Automatic Fault Prediction of Wind Turbine Main Bearing Based on SCADA Data and Artificial Neural Network

In this paper, Zhang [2018] describes the methodology used to automatically predict incipient faults by Kongsberg Digital AS in their EmPower® system. The methodology is very similar to [Zhang and Wang, 2014], with some differences. The rear bearing temperature is used as target variable in this work as well, but the methodology is reusable for modelling other sensors that are known to act as a proxy for faults. Only the differences between this and the previous work will be highlighted in this section. First, the t-1 lagged variable is included for Active power, Nacelle temperature, and Turbine speed. The combined procedure is illustrated in figure 3.1 below.

A sliding-window approach is proposed to determine whether an alarm should be generated. In this way, transient deviations from the expected output will not result in an alarm, and subsequently, the number of false alarms is reduced. The authors show that they are able to detect faults from "some days to 2 months

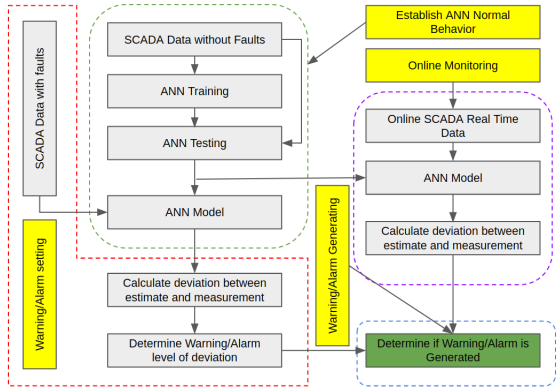



Figure 3.1: Fault prediction procedure, adapted from [Zhang, 2018]

before fault", but conclude that it would have been necessary to study more fault cases to be able to give a more exact statistical evaluation of the system.

### 3.2.3 A Data-Driven Approach for Condition Monitoring of Wind Turbine Pitch Systems

In this paper, Yang et al. [2018] also employs a forecast-based fault prediction methodology, where the pitch motor temperature is used as the proxy for faults. The input data initially include 15 variables, but this is reduced to five variables through a feature selection process. The architecture of the proposed system can be seen in figure 3.2.

The authors make sure that the normal model is trained on healthy data only, through removal of abnormal data in a preprocessing step. They explored several machine learning models in this paper, and found Support Vector Regression (SVR) Drucker et al. [1997] to be the best performing algorithm, and an ANN with three layers resulting in the second best results. No details about the number of hidden units or architecture of the ANN were provided. Another contribution of this work is the proposal of an exponentially weighted moving average (EWMA) control chart that is used to determine whether an alarm should be generated. This method gives more weight to recent deviations from expected values than more distant deviations.

The authors demonstrate that their model is able to detect pitch-related faults 38 to 162 hours ahead of time in 8 specific cases, as shown in 3.3.

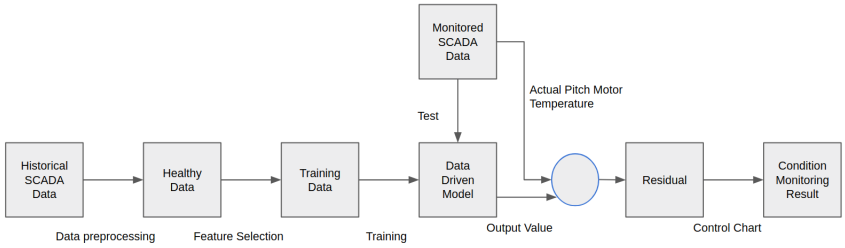



Figure 3.2: Framework of data-driven fault prediction, adapted from [Yang et al., 2018]

| Case | Alarm Time | | Time Ahead |
|---|---|---|---|
| | Proposed Model | Supervisory Control and Data Acquisition (SCADA) Alarm System | |
| 1 | 23 August, 2016 23:30 | 30 August, 2016 2:20 | 146.8 h |
| 2 | 19 December, 2016 17:00 | 21 December, 2016 17:30 | 48.5 h |
| 3 | 25 January, 2017 15:50 | 28 January, 2017 12:00 | 68.2 h |
| 4 | 7 April, 2017 15:10 | 14 April, 2017 9:20 | 162.2 h |
| 5 | 26 April, 2017 16:10 | 29 April, 2017 16:30 | 72.3 h |
| 6 | 20 June, 2017 7:10 | 21 June, 2017 21:20 | 38.2 h |
| 7 | 20 June, 2017 14:00 | 22 June, 2017 3:20 | 37.3 h |
| 8 | 24 June, 2017 00:10 | 26 June, 2017 7:10 | 55.0 h |

Figure 3.3: Monitoring results of pitch related faults [Yang et al., 2018]

### 3.2.4 Wind Turbine Fault Detection Using Denoising Autoencoder with Temporal Information

As the title suggests, Jiang et al. [2018] has in this work adopted the objective of detecting faults that are already present. Even though the objective differs somewhat from fault prediction, the main goal is the same: to reduce operations and maintenance (O&M) costs. This approach aims to do so by detecting smaller, non-critical faults, that otherwise may have gone undetected. The underlying assumption is that such faults may result in more critical faults if they are not detected and not acted upon.

This paper makes no assumption about which specific sensors may be proxies for faults. Instead, they use an unsupervised learning approach, where they build a reconstruction model on all sensors available. The reconstruction model is based on a deep autoencoder, which is trained to be able to reconstruct its own input. The input data is a sliding window of the SCADA data, so that temporal

variations both within an individual sensor, as well as between sensors, can be captured.

Another key aspect of this approach is the technique used to make the autoencoder resistant to noise in input data. This is achieved through randomly dropping out a subset of the input data, with the objective of training the model to reconstruct the original data, even if the input data is noisy, or have missing values. The authors introduce the acronym DAE (Denoising Autoencoder)[1]. The proposed sliding window-denoising autoencoder (SW-DAE) is compared to both PCA, AE, DAE and DPCA models. The authors use ROC-curve and AUC as the metric for comparison with other models. It is shown that the sliding window approach yields an increase in model performance, and the SW-DAE has the best performance of the evaluated models on two fault cases.

A diagram of the SW-DAE is shown in figure 3.4

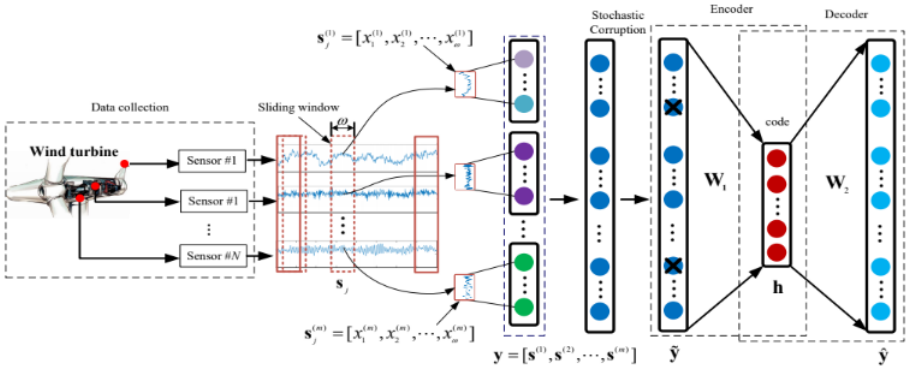



Figure 3.4: Denoising autoencoder with temporal information, reproduced with permission from [Jiang et al., 2018]

Here, we can see that the time series resulting from a sliding window over the different sensors are used as input to an autoencoder. Some noise are added to these time series, and the resulting signal is sent through an autoencoder, which need to learn a low-dimensional representation of the input signal in order to recreate its input.

[1]Not to be confused with Deep Autoencoder, which is also referred to by the same acronym

### 3.2.5 Wind Turbine Gearbox Failure Identification With Deep Neural Networks

In this paper, Wang et al. [2017a] used a forecast-based approach as well. In this paper the pitch lubricant pressure was used as proxy for gearbox related faults. Only three input variables are used to model the pitch lubricant pressure: The gearbox oil temperature, the power output and the shaft temperature. No lagged variables are included to capture the temporal differences in the signal.

The authors compared six different machine learning algorithms, DNN, k-Nearest Neighbors (kNN), LASSO-regression, Ridge-regression, SVM and Neural Networks (NN). The metrics used for comparison were MAPE (mean absolute percentage error) and SDAPE (standard deviation of average percentage errors). The Deep Neural Network consisted of three hidden layers, and used dropout with a dropout probability of 0.5. The hyperbolic tangent (tanh) was used as activation function. The NN considered consisted of only one hidden layer. The number of hidden units were selected through cross-validation, with the maximum number set to 100.

They trained one model for each of six different wind farms in China. Data from a total of 92 different wind turbines were used. Five actual gearbox faults occurred in the period of the data that was collected. The DNN method resulted in the lowest prediction errors of the evaluated methods.

Similar to Yang et al. [2018], an EWMA control chart was employed to determine when an actual alarm should be raised. A window size of one week was used as input. The authors show that they are able to detect faults up to two days in advance, without generating false alarms.

### 3.2.6 The Prediction and Diagnosis of Wind Turbine Faults

In this paper, Kusiak and Li [2011] employ a supervised learning strategy, where fault/status data are combined with SCADA data. They suggest a three-level prediction approach, where the first level is to predict whether a fault will occur. The second level is to predict the category of the fault, and the third level is to predict a specific fault. The data used in this study is sampled to 5-minute intervals, and is made up of data collected from four different wind turbines over a period of three months.

To create a balanced dataset, the authors downsample the number of negative training examples to match the number of faults. This results in a total training dataset size of only 1300 for level 1 and 2, and 168 for level 3. Only two input variables are used, the wind speed and power output. No further explanation was given as to why only these were used, as they note that many more variables are available. The authors evaluate the use of different prediction periods, meaning the number of timesteps before a fault that was used as input to the algorithm.

The number of timesteps evaluated range from 0 to 12, with 12 timesteps ahead resulting in 60 minutes of input data. The evaluation metrics include accuracy, sensitivity and specificity. However, the usefulness of these metrics is not apparent, as the distribution of faults in the downsampled dataset does not match the original data. The downsampled data was also used for evaluation. The authors also note that the lack of labelled fault data is a limitation of this study.

### 3.2.7 Diagnostic Models for Wind Turbine Gearbox Components Using SCADA Time Series Data

Orozco et al. [2018] describe an initial dataset of 948GB, but this was subsequently resampled to 10-minute intervals. The objective of the study is to model temperature of turbine components, adopting a forecast-based approach. The authors chose to use only two input variables, the ambient temperature and the power output. They also randomly downsample the dataset to 5000 samples before training their models. No explanation is given as to why this was done. The algorithms evaluated include linear regression, multivariate polynomial regression, random forest, and neural network. The two linear models resulted in lowest RMSE. The authors propose a rule-based statistical evaluation to flag a fault, and show that the proposed algorithm is able to detect faults. The study does not include any metric or evaluation to take the number of false alarms given into consideration.

### 3.2.8 Learning Deep Representation of Imbalanced SCADA Data for Fault Detection of Wind Turbines

In this paper, Chen et al. [2019], aim to predict blades icing accretion faults based on SCADA data. They attempt to mitigate the class imbalance issue of training a deep neural network by reframing the task to learn an embedding of a data sample, rather than train a discriminative classifier directly on the labels. Their target is to learn this embedding, and exploit this embedding vector through performing a k-nearest neighbor model for classification. An important point is that the samples that are used to train their network are not only one single SCADA data point, but a triplet of points, consisting of:

1. $X_a$ : An anchor point.

2. $X_p$ : A data point of same class as anchor point.

3. $X_n$ : A data point of opposite class as anchor point.

The cost function of the neural network is then configured to minimize the distance between the anchor point and the positive sample, and maximize the distance between the anchor point and the negative sample, as illustrated in 3.5.

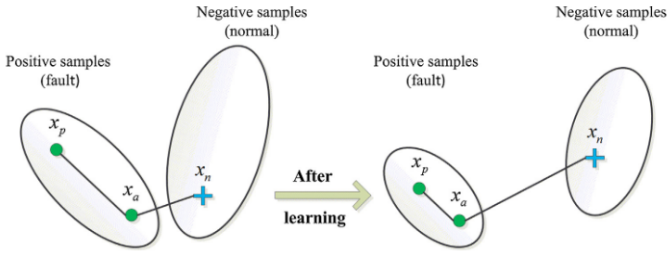



Figure 3.5: Learning embedding vectors [Chen et al., 2019]

The network architecture used is a combination of CNN and fully connected layers. The dataset is collected from 3 different wind turbines over a two-month-period, with 7 second resolution. This is substantially higher resolution than other works in the literature. The authors do not mention either the number of faults or labelling procedure used, which makes it difficult to draw insights from the empirical evaluation. Specifically, it would be useful to know if their labels only denote samples after an icing fault has already occurred.

### 3.2.9 Comparative Analysis of Neural network and regression based condition monitoring approaches for wind turbine fault detection

This study, performed by Schlechtingen and Santos [2011], compares linear regression and neural networks for creating normal models of the following fault proxy variables: power output, generator bearing temperature, generator stator temperature, generator slip ring temperature, shaft speed, gearbox oil sump temperature, gearbox bearing temperature and nacelle temperature. The purpose of these models is to be used in a forecast-based prediction fault scenario.

The study evaluated both what they call an autoregressive neural network, where one timestep lagged values are included as input variables too, and a neural network with only the signals at present time as input variables. The performance varied for different scenarios. Both neural network-based approaches were found to result in lower prediction errors than linear regression methods. No quantified evaluation was done with regards to the number of false alarms generated.

### 3.2.10 A Robust Prescriptive Framework and Performance Metric for Diagnosing and Predicting Wind Turbine Faults Based on SCADA and Alarms Data with Case Study

In this paper, Leahy et al. [2018] adopted a supervised learning approach. Even more, they propose a novel framework to label faults based on SCADA data and alarms data. This is an important contribution towards more unified research efforts in this domain. In their own words, three goals are being targeted in this paper:

1. Build a system which can automatically identify turbine stoppages and identify their high-level root cause, based solely on alarms and 10-minute SCADA data as inputs, with no manual cross-referencing of maintenance logs or correspondence from technicians needed, and apply this to an existing dataset.
2. Use this dataset to predict specific types of wind turbine faults, using classification techniques and evaluate classification performance using appropriate metrics.
3. Evaluate the effectiveness of the classifier as a field-deployed system using a novel alarms-based system.

The overview of the framework can be seen in figure 3.6. The authors highlight an important part of the labelling phase, where three different labels are applied initially, pre-fault (PF), non-fault (NF) and fault (F). The F label is set during a stoppage. The PF label is set for samples that precede a fault by less than $n$ timesteps, where $n$ is the number of timesteps before a fault the model is trained to predict it.

In this paper, several classification algorithms were evaluated. Both SVM (with linear, polynomial and Gaussian kernel), decision trees, logistic regression, and a random forest model was evaluated, with the random forest model yielding the best results. Hence, a random forest classifier was also the final classifier used generate predictions in their proposed framework.

### 3.2.11 Summary and state-of-the-art

Related research in the area of fault prediction in wind turbines based on SCADA data has been presented. This section summarizes the chapter and reviews the related work in light of our research questions. This provides a motivation for the architecture of the proposed system.

Several issues make it difficult to establish a clearly defined state-of-the-art for fault prediction in wind turbines from SCADA data. There has been a lot of

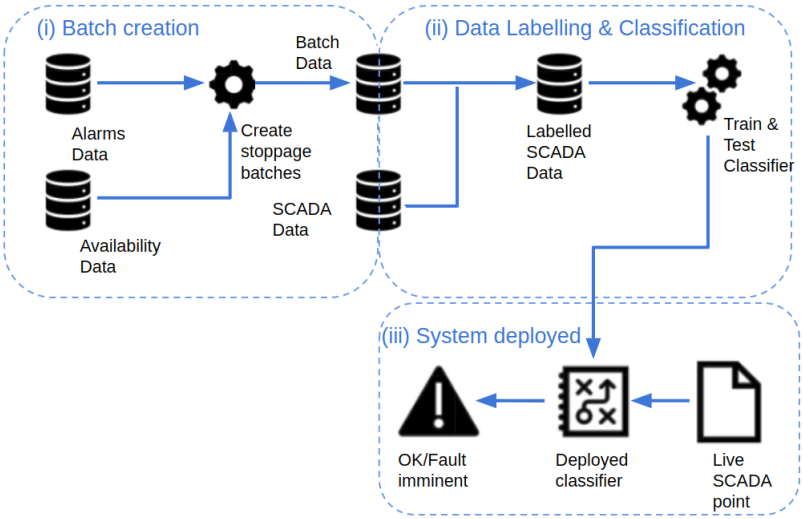



Figure 3.6: Overview of proposed framework, adapted from [Leahy et al., 2018]

effort in this area, but due to the diversity of the methodologies as well as large variations in datasets, the knowledge transfer between research efforts is limited.

Other fields, such as image recognition Deng et al. [2009] and language modelling Mikolov et al. [2011], have reaped huge benefits from establishing well-known benchmark datasets. Such an initiative would likely benefit the researchers working on fault prediction in wind turbines based on SCADA data as well. One of the challenges that might present itself in this case is the lack of common equipment and vendor systems, which makes the standardization of such a dataset difficult. For the papers that have been reviewed for this project, there are also large variations in number of samples, number of features and evaluation methodology. The variations across some of these taxonomies are distilled and presented in table 3.4.

Only three of the reviewed papers adopt a supervised fault prediction approach. It is likely that an amount of high quality fault labels have been difficult to obtain, especially considering the relatively small number of samples for the majority of the work. Out of the papers that used a supervised learning approach, the largest dataset used is 6 months' worth of data collected from 11 wind turbines in the paper by Leahy et al. [2018]. Still, the number of stoppages per stop category was only in the range of hundreds. The authors of that study

Table 3.4: Summary of reviewed work

| Paper | Number of samples | Number of features | Approach | Method (best if several) | Capture time features | Evaluation and results | Evaluate false alarms |
|---|---|---|---|---|---|---|---|
| Wind turbine fault detection based on SCADA data analysis using ANN Zhang and Wang [2014] | 13000 | 4 | Forecast-based prediction of rear bearing temperature | ANN | No | Testing on one fault, predicting 10 days in advance. | No |
| A Data-Driven Approach for Condition Monitoring of Wind Turbine Pitch Systems Yang et al. [2018] | 10562 | 5 | Forecast-based prediction of pitch motor temperature | SVR | No | Testing on 8 faults, predicting 38-162 hours in advance | No |
| Wind Turbine Fault Detection Using Denoising Autoencoder with Temporal Information Jiang et al. [2018] | 5000 | 10 | Anomaly-based fault detection | Sliding-window Denoising Autoencoder | Yes | AUC better than other anomaly-based baselines. | No |
| Wind Turbine Gearbox Failure Identification With Deep Neural Networks Wang et al. [2017a] | 3642-76500 per wind farm (6 wind farms modelled separately) | 3 | Forecast-based prediction of pitch lubricant pressure | ANN | No | Show that five actual faults are accurately predicted | Yes, but only for 87 samples. These do not generate false alarm. |
| The Prediction and Diagnosis of Wind Turbine Faults Kusiak and Li [2011] | Originally 25920, undersampled to 1300 for 2 fault levels, and 168 for the last one. | Originally "over 60", 2 used | Supervised fault prediction | CART | No | Evaluate on 7 fault categories with 12-55 faults each. | Yes, compute recall, but on an undersampled dataset |
| Diagnostic Models for Wind Turbine Gearbox Components Using SCADA Time Series Data Orozco et al. [2018] | 5000 | 3 | Forecast-based prediction of bearing temperature | Multivariate polynomial regression | No | Evaluate on assumed faults derived from temperature spike followed by stop. | No |
| Comparative analysis of neural network and regression based condition monitoring approaches for wind turbine fault detection Schlechtingen and Santos [2011] | Not disclosed | 5 | Forecast-based prediction of several sensor measurements | ANN | Yes, one lagged timestep. | Show that anomalies are detected before six known faults. | No |
| Learning deep representation of imbalanced SCADA data for fault detection of wind turbines Chen et al. [2019] | 2-months (7s resolution) | 22 | Learning deep representation for classification | ANN+kNN | Show improved detection on blades icing accretion fault compared to NBM | Yes, for some specific features. | Yes, but not useful due to unknown labelling procedure |
| A robust prescriptive framework and performance metric for diagnosing and predicting wind turbine faults based on SCADA and alarms data with case study Leahy et al. [2018] | Originally 300000, but they mention that they do random undersampling before training | 11 (but used 2D PCA before training) | Supervised fault prediction | Random Forest | No | Evaluate several hold-out sets with fault data from one turbine as hold-out set each time, resulting in 7-66 faults. | Yes, compute recall and number of false alarms for corresponding precision on each hold-out set |
| This thesis | 640k-3.4M, for each of the fault categories (see table 4.4) | 167 | Supervised fault prediction | Several deep learning architectures | Yes | Evaluate on unseen test sets for multiple fault categories | Yes, present F-score for actual alarms on test set |

also emphasize a desire for the verification on a larger dataset. The benefit of more data is considered to be even more present when considering the use of deep learning methods, where performance only increases with the amount of data. It is also interesting to note that the largest number of features used in the reviewed papers are 22, with the majority using 5 or less features. This seems a bit odd considering that SCADA data from wind turbines typically provide a wide variety of sensor measurements. Possible explanations may be that only the most important features are chosen based on domain expertise, or that a reduced number of features are chosen to reduce complexity of training the machine learning algorithms. It is likely to assume that improved performance could be achieved by including all available features. This is especially true if deep learning methods are to be used, since they are well suited for modelling high-dimensional data. Another finding is that only 3 of the reviewed papers capture temporal features. Those that do, also employ this in quite a limited fashion, by only adding the lagged timesteps for a few selected features. As discussed in section 2.3, it is likely that being able to include temporal features will lead to increased fault prediction performance.

Leahy et al. [2018] is considered to be the most relevant for this project due to its relatively large dataset, a robust framework for labelling stoppages, as well as a relatively large subset of features included (8 features plus a few derived features). This paper also avoids some pitfalls seen in other papers related to resampling and evaluation metrics. It is important that the evaluation metrics evaluate the amount of actual faults detected as well as false alarms given. If the system is trained on samples labelled in the PF-period, it is necessary to include a way of translating the metrics on these predictions back to the actual faults and false alarms.

For reasons discussed above, we will consider the system proposed here by Leahy et al., and their use of a random forest model, to be the state-of-the-art of fault prediction in wind turbines based on SCADA data, and their method will be adapted to our dataset and compared to our proposed system.

## 3.3 Deep learning for Multivariate Time Series Classification

This section will provide a review of deep learning for multivariate time series classification. The papers in this section have not been selected through a SLR, but identified by starting with two recent, relevant review papers. The intention of this section is primarily for inspiration as to which DL architectures that might work best for our kind of problem. I acknowledge that empirical experiments will be necessary to evaluate any hypotheses that might arise from this section.

### 3.3.1 Deep learning for time series classification: a review

In Ismail Fawaz et al. [2018], the authors provide a review of the field, along with an empirical evaluation of several DL architectures on a series of datasets. The datasets are gathered by Bagnall et al., in an effort to promote research by enabling comparison of methodologies across a wide range of datasets from different domains and with different characteristics.

The authors compare 9 DL architectures from the deep learning for time series literature. Their results show that the ResNet architecture originally proposed as a computer vision model in [He et al., 2016], and adapted for time series classification by Wang et al. is the best performing methodology across the 12 datasets, with an average rank of 1.62 out of 9 methodologies on the datasets with most training samples ( $>$799). However, the results from this review are not particularly useful for the specific task of this system, as our dataset is 2-3 orders of magnitude larger. Of the datasets used in the review, only 2 had more that 1000 training samples, and the largest one had 6600 training samples. Also, the ResNet architecture is DL architecture designed for image recognition with elaborate details specific to this task, and has a large number of parameters.

### 3.3.2 Time series classification from scratch with deep neural networks: A strong baseline

In Wang et al. [2017b], a simple end-to-end baseline architecture for time series classification is proposed, based on Fully Convolutional Networks (FCN). The fully convolutional architecture is an architecture which has previously achieved state-of-the-art results on semantic segmentation of images [Long et al., 2015]. In Wang et al. [2017b], this architecture is adapted for time series and shown to perform well on datasets from Chen et al. [2015]. Note that the authors evaluate the architecture on one-dimensional time series only, and necessary adaptions will have to be made in order to apply the same architecture to multivariate time series, especially in order to capture interactions between individual features (which can be thought of representing one individual time series).

This architecture is shown in figure 3.7. The numbers at the bottom of a block is the number of filters in the block. For each block, three 1D kernels of size [3,5,8] are used for the convolution operation. BN is an acronym for batch normalization as introduced in section 2.14.5. This is shown to act as a regularizer and improve performance of deep neural networks [Ioffe and Szegedy, 2015].

### 3.3.3 An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

Bai et al. [2018] popularize the term Temporal Convolutional Network to describe

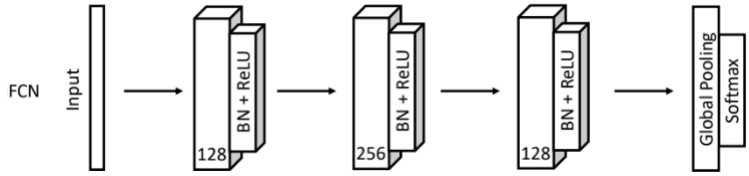



Figure 3.7: Fully Convolutional Networks for Time Series Classification, adapted from [Wang et al., 2017b]

a generic sequence modelling architecture based on convolutional layers, rather than recurrent ones. In Bai et al. [2018], the authors show through an extensive empirical evaluation that TCNs are able to outperform canonical recurrent networks, such as LSTMs, on several challenging sequence modelling tasks. In addition, they discuss some of the pros and cons for TCNs vs. recurrent architectures. TCNs have the advantage of much more efficient training, and a flexible receptive field size. Recurrent networks have the advantage that they only need input of a current observation to make a prediction, while a TCN needs input equal to the length of the receptive field to make a prediction.

The architecture is characterized by two properties: 1) The convolutions are causal, meaning there is no information leakage from future to the past. 2) The architecture can take a sequence of any length and map to a sequence of any length. The basis of this architecture is a FCN, with the added constraint of causal convolutions. The authors also show how this architecture can be adapted to sequences of arbitrary length, through the use of dilated convolutions, in order to enable an exponentially large receptive field. Figure 3.8 illustrates how the dilated convolutions are applied to every $d$ observations(s) of the input. This can be seen in contrast to standard convolutional kernels, where the filters are applied to a contiguous sequence of the inputs.

In addition, a residual connection is proposed, with the purpose of stabilizing possibly deep networks, by allowing layers to learn the identity mapping[2]. In figure 3.9 the identity mapping is illustrated with the green line. This has shown to benefit very deep networks.

### 3.3.4 Section summary

This section has reviewed the most recent literature in the field of deep learning for time series classification. A limitation of the review is the size of the datasets used for most empirical evaluations. This makes it difficult to draw di-

[2]The identity mapping is simply the same output as input

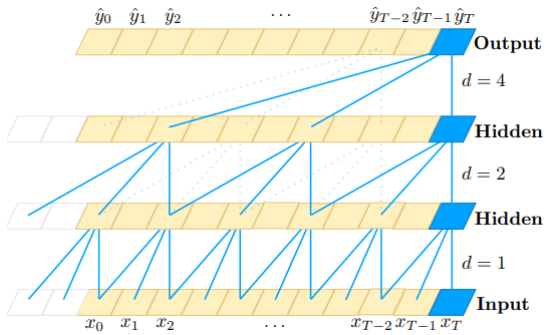



Figure 3.8: Temporal Convolutional Networks [Bai et al., 2018]

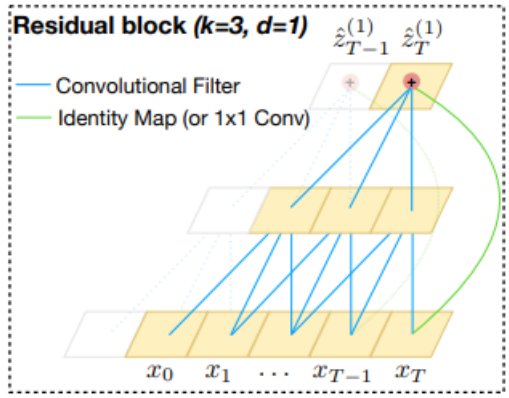



Figure 3.9: Residual connection in TCN [Bai et al., 2018]

rect conclusions about which DL architectures that will be most suitable for our problem. Nevertheless, we see that DL has shown great promise for time series classification in general. Both recurrent and convolutional neural networks, and in particular architectures that combine the two have been successful in the field. These insights will be built upon for our purpose of designing a DL system for fault prediction in wind turbines based on SCADA data.

## 3.4 Chapter Summary

This chapter has reviewed the literature both in the specific field of fault prediction in wind turbines based on SCADA data, as well as deep learning for multivariate time series classification. The domain-specific field is characterized by spread efforts, diverging methods for labelling, and a lack of standardized benchmarks. Additionally, the reviewed works have not utilized the full set of sensors available from SCADA data, the datasets have been of rather limited size, and temporal features has not been given particular attention, as most work has treated one SCADA observation individually, without adding engineered temporal features.

The deep learning for time series classification field has reaped some benefits from common benchmark datasets. These are, however, not directly comparable to the dataset for this project, especially with regards to size. These datasets do not pose the imbalanced class problem that we face for this project. Some deep learning architectures that have shown good performance across a broad range of datasets have been investigated, and are used for inspiration and reference when designing the prototypes that are presented in chapter 4, and evaluated in chapter 6.

# Chapter 4

# Architecture

This chapter will describe the architecture of the proposed system for predicting faults on wind turbines. First, an overview of the system will be presented, before I will elaborate on the details of each component in the following sections. The aim of this chapter is to provide the reader with an understanding of how the system is designed in order to produce a prediction for an impending fault. The proposed system consists of the following main components:

- Labelling algorithm for labelling data points.
- Data cleaning and preprocessing.
- Splitting in training, validation and test set.
- A classifier component. Several deep learning classifiers are evaluated against a random forest classifier.
- An alarm control system that converts raw probability predictions into the binary event of an alarm.
- An evaluation component that enables fair comparison of different classifier component on a metric that is closely related with real-world performance.

A visual overview of the system can be seen in figure 4.1.

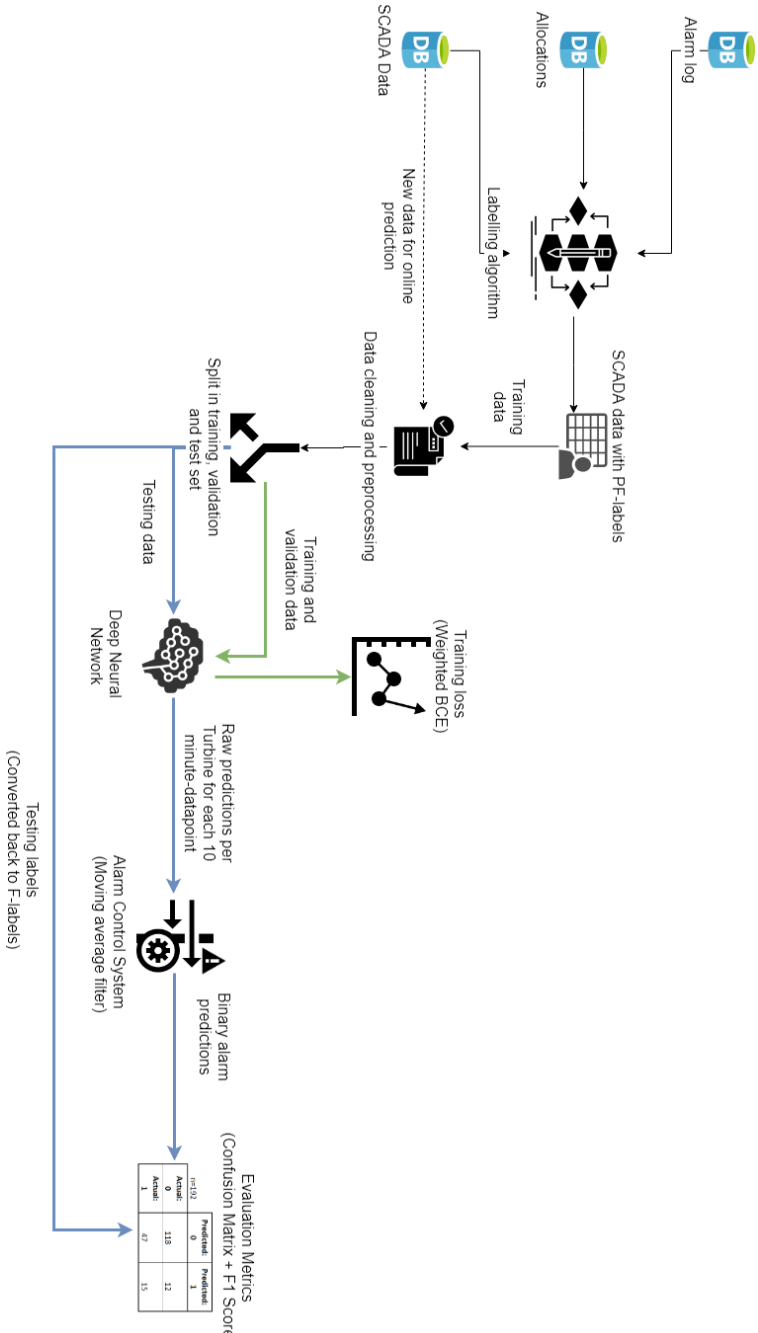



Figure 4.1: An overview of the proposed system

## 4.1 Data Sources

For this research project, the dataset that has been made available is collected from 67 wind turbines located at one of Equinor's offshore wind farms outside the coast of the UK. The collection period is from 1.7.2017-31.12.2018. The size of the dataset that is made available to this project exceeds those that are used in all the reviewed papers, both with regards to length, number of turbines, sensors included and number of faults. As the performance of deep learning methods improves with more data, this is considered a great advantage.

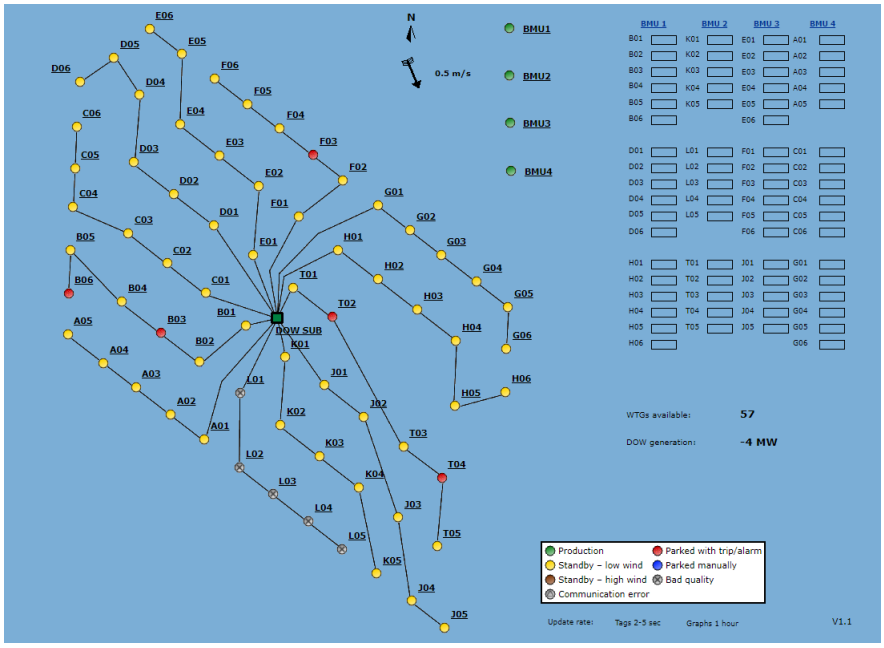



Figure 4.2: An overview of the wind farm (Screenshot from Bazefield Software Platform)

In figure 4.2, we can see how the 67 wind turbines that form the wind farm are organized. The letter next to each turbine indicates to which string the turbine is attached. The number indicates the turbine's position on the string starting from closest. We can see that some strings have 5 turbines attached, and some have 6.

### 4.1.1 SCADA data

The SCADA data from each of the 67 turbines is resampled to 10-minute intervals, which translates to 5.28M data points over the collection period in total. The resampling means that some information will be lost, but in order to remedy this, both the average, maximum, minimum, and standard deviation is included for each 10-minute interval for each sensor. This enables us to capture some of the signal lost due to the downsampling. The mean and standard deviation may not be as useful for sensor measurements that are not normally distributed.

Below, in table 4.1 is an example of the dataset for one turbine, with only a few selected variables for the sake of illustration. In total, 167 variables are used.

| TimeStamp | ActivePower_avg | AmbieTmp_avg | BladeAngleA_avg |
|---|---|---|---|
| 2017-07-01 00:00:00 | 3034.4880 | NaN | -3.493407 |
| 2017-07-01 00:10:00 | 4033.7280 | NaN | -3.425983 |
| 2017-07-01 00:20:00 | 4557.7417 | 12.811111 | -2.856430 |
| 2017-07-01 00:30:00 | 5496.5920 | 13.790909 | -1.077675 |
| 2017-07-01 00:40:00 | 2129.2964 | NaN | 0.499621 |

Table 4.1: Sample of SCADA data (obfuscated for data protection)

### 4.1.2 Allocation data

This data source contains records of wind turbine outage along with a manually labelled fault allocation category. This category represent the subsystem for which the fault is allocated to. These categories are of particular interest to us, as these are of a granularity that is considered to be suitable as targets. Using the fault categories will allow for many more examples of each fault category compared to using specific alarms as targets. This consideration is based upon the assumption that the majority of faults belonging to a category will share some characteristics. The human-in-the-loop allocation of these data may be seen as both an advantage and a disadvantage. In general, the data is assumed to be of good quality, although there are also some examples of similar outages being allocated to different fault categories. To alleviate this possible noise in the labels, we combine the allocation data with the automatically generated alarm data described in the next section.

There are several of the allocation categories that represent manual stoppages, planned maintenance and other stoppages that are part of normal operations. Fortunately, domain experts from Equinor were helpful in identifying which of these categories that it would be useful to know about in advance. Stoppages belonging to these categories will form our fault dataset. The identified categories were:

- M076 Blade Adjustment System
- M079 Control And Protection System
- M104 Yaw System
- M005 Out of Electrical Specification
- M099 TS-Converter System

There were in total 3075 faults over the collection period, and the distribution over categories can be seen in figure 4.3.

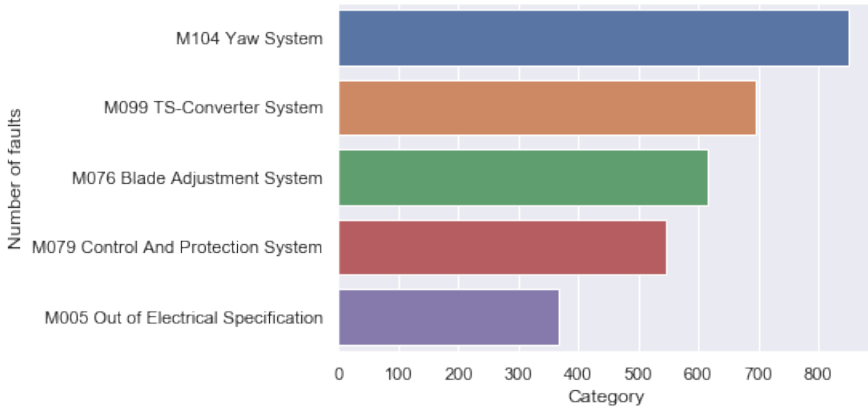



Figure 4.3: Distribution of faults per category

For this project, each of the fault categories will be modelled independently. The reasoning is that different fault categories will have different characteristics, which make some of them more susceptible for being predicted in advance than others. Modelling each category will provide the opportunity to test which categories that benefit from a longer period of PF-labels, and which categories that don't.

It is also useful to investigate their contribution to downtime, as different faults will likely take longer to fix than others. The total downtime duration caused by each fault category is shown in figure 4.4

It is also useful to examine the development of allocations has been over time. This can be seen in figure 4.5

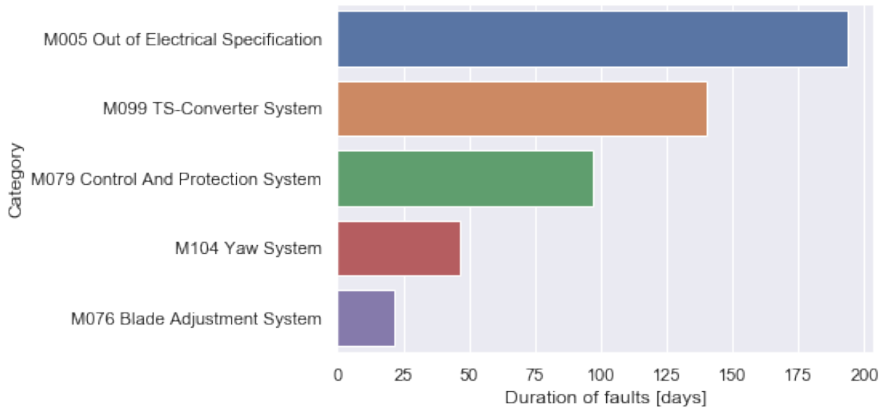



Figure 4.4: Duration of allocations per category

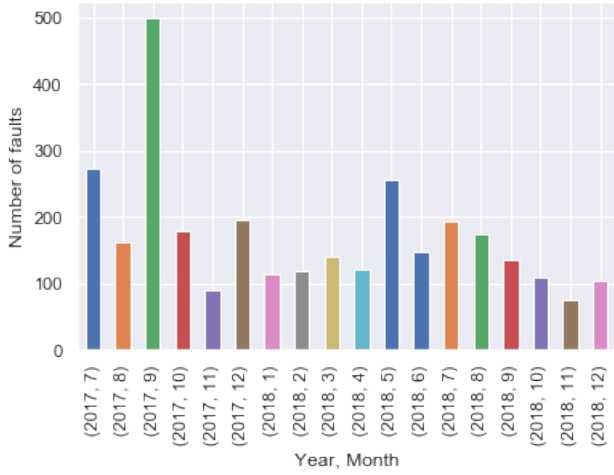



Figure 4.5: Number of allocations per month

### 4.1.3 Alarm data

This data source, in contrast to the allocation data, is not human-verified, but is a result of automatic alarm mechanisms in the SCADA system. These are of a relatively more detailed nature than the allocation data, and operates with "Alarm Code" as identifier. An allocation will typically consist of a sequence of

alarms which constitute an "alarm batch", and is allocated to a fault category. The exact nature of this data varies between manufacturers, but the structure is similar in most cases. Each alarm instance in the data made available for this project are categorized into in one of the following categories:

1. Status
2. Warning
3. Alarm

Our real interest lies in predicting the unexpected stoppages of the wind turbine. There are many reasons for a turbine to stop, and many stoppages are part of normal operation. In addition, a stoppage is generally preceded by a sequence of multiple alarms. This makes the procedure to label a SCADA point as an unexpected stoppage or not a bit intricate: In general, the first alarm of a fault allocation is considered the root cause alarm, and is used to map the stoppage to a corresponding category deterministically. All faults belonging to categories that are associated with intended stoppages due to maintenance or curtailment are excluded.

A sample of the alarm data made available can be seen in table 4.2.

| Turbine | Description | Start | End | Type | Category |
|---|---|---|---|---|---|
| DOW-L05 | Low lower frequency exceeded | 2017-07-11 09:01:59 | 2017-07-11 10:22:03 | Alarm | M005 Out of Electrical Specification |
| DOW-K01 | Gridvolt less than lower limit 1 | 2017-08-27 17:13:53 | 2017-08-27 17:25:57 | Alarm | M005 Out of Electrical Specification |
| DOW-E06 | Manual stop - owner | 2017-12-07 10:05:22 | 2017-12-07 16:45:27 | Alarm | M009 Requested Shutdown |
| DOW-F02 | Profinet SWTCC2 guard signal error | 2018-01-15 20:58:52 | 2018-01-15 21:01:51 | Alarm | M079 Control And Protection System |
| DOW-A03 | Hub: No feedback MTS-sens B | 2017-12-12 15:24:40 | 2017-12-12 15:46:12 | Alarm | Unknown |

Table 4.2: Sample of available alarm data

## 4.2 Labelling Algorithm

Considering that I have been given access to a dataset with relatively large number of faults that have been both allocated by humans and detected by an alarm system, a supervised approach has been chosen for this thesis. This implies that we need an algorithm for labelling. The procedure for labelling data points is a critical step to build a useful fault prediction system. As discussed in chapter 2, there are many ways of doing this. Based on the literature review, the method used for this project will be to combine stoppages from the alarm data with corresponding manual allocations, adapting the labelling algorithm from Leahy et al. [2018] to the provided dataset.

Our initial goal is to label each 10-minute SCADA point as to whether or not a fault occurs in the next 10-minute period. The algorithm can be split into the following steps:

1. Separating the alarms into batches belonging to one stoppage.
2. Connect each stoppage to a known subsystem, which has an associated stop category.
3. Label all the 10-minute SCADA points within the duration of each stoppage with the stop category.
4. Select a pre-fault period, $w_1$, and a minimum prediction-period, $w_2$, for each stop with a start time, $t_s$, and end time, $t_e$. Label according to:

$$y_t = \begin{cases} F, & \text{if } t_s - w_2 \leq t \leq t_e \\ PF, & \text{if } t_s - w_1 \leq t \leq t_s - w_2 \\ NF, & \text{otherwise} \end{cases}$$

   Where F is the label of the samples where the fault has occurred, PF is the pre-fault observations, and NF is normal, healthy data.

All the samples with F-labels will be removed from our dataset, as we do not want to train our algorithm to detect faults that already have occurred. The samples with PF labels will be the target that our machine learning algorithms will be trained to detect. To illustrate, we can look at 4.3. Here the labels for two different windows for one fault is shown. The actual fault lasted from 2017-07-22 16:53:17 to 2017-07-22 17:32:17. We can see that Label_1_M005 indicates the label with $w_1 = 10min$ and $w_2 = 0min$. Label_6_M005 indicates the label with $w_1 = 60min$ and $w_2 = 0min$. Note that the samples in the F period are removed.

| | TimeStamp | Label_1_M005 | Label_6_M005 |
|---|---|---|---|
| 3132 | 2017-07-22 16:00:00 | 0.0 | 1.0 |
| 3133 | 2017-07-22 16:10:00 | 0.0 | 1.0 |
| 3134 | 2017-07-22 16:20:00 | 0.0 | 1.0 |
| 3135 | 2017-07-22 16:30:00 | 0.0 | 1.0 |
| 3136 | 2017-07-22 16:40:00 | 0.0 | 1.0 |
| 3137 | 2017-07-22 16:50:00 | 1.0 | 1.0 |

Table 4.3: Result of labelling procedure

## 4.3 Training, Validation and Test Split

In order to ensure that that machine learning models will generalize well to unseen data, we usually split the dataset in to training set, validation set and

test set. The training data is used to train the model, and the model parameters that results in best performance on the validation set is used. Several different validation set may also be used to determine the best parameters. In order to provide an unbiased evaluation, we need to evaluate on a set that is not used for either training or (hyper)parameter selection. This is the test set, or holdout set.

The selection of training, validation and test sets is a crucial aspect of the system and there are several pitfalls that must be avoided in order to produce results that are not misleading, and will generalize well to unseen data. In order to determine how we should split our dataset, the following criteria are formulated:

1. The training, validation and test set should be consecutive in time and non-overlapping
2. We must be able to define the percentage of faults to be used for each set.
3. The sets should be stratified, ie. contain the same fraction of fault samples.
4. The PF-labels generated from the same fault should all be in the same split.

One could argue that the turbine's behavior is not affected by its behavior in the distant past, and that it would not introduce leakage to use validation data from an earlier point in time than the training data. However, consider the case where a new kind of fault develops over time. If we were to split disregarding time, we would model these faults a lot better, but this is not something we would be able to do in a real setting, as the new faults would not be possible to train/validate on. Splitting of the sets with regards to time will likely not lead to as good metrics as random split independent of time. However, I feel that it is better to err on the side of strictly avoiding a potential leakage from future data. This will ensure that I convey a more realistic expectation of what performance might be expected by the system on unseen data.

Another option would be to use one or several of the turbines as validation and test set. This was considered, but it is possible that there are some global features that are common for all turbines, which would lead to information leakage in this case as well. This would also make it impossible to take advantage of the learned embedding vectors for a turbine, which we hope will lead to improved performance.

The reasoning for criteria no. 3 is to enable a fair comparison of metrics across the different sets, as well as ensuring that the class_weights that are used for the training set will also be valid for the validation and test sets. In a production setting, these weights would need to be updated as the fraction of fault samples would drift. This criteria is achieved by randomly downsampling the negative class for the sets with the lowest fraction of labelled samples, so that the fraction of labelled samples are equal.

| Fault category | PF window | Samples Train | Samples Val | Samples Test | N labels Train | N labels Val | N labels Test |
|---|---|---|---|---|---|---|---|
| M005 | 1h | 640612 | 138020 | 138020 | 1411 | 304 | 304 |
| M005 | 6h | 639517 | 138798 | 138798 | 6621 | 1422 | 1437 |
| M076 | 1h | 2653547 | 568830 | 568830 | 1782 | 382 | 382 |
| M076 | 6h | 2557778 | 548228 | 548228 | 10964 | 2351 | 2350 |
| M079 | 1h | 2481854 | 531913 | 531913 | 2039 | 437 | 437 |
| M079 | 6h | 2627990 | 563269 | 563269 | 11720 | 2512 | 2512 |
| M099 | 1h | 3392612 | 729253 | 727387 | 1819 | 390 | 391 |
| M099 | 6h | 3390869 | 726837 | 726837 | 9783 | 2097 | 2097 |
| M104 | 1h | 2057866 | 451446 | 449022 | 3396 | 745 | 741 |
| M104 | 6h | 2078744 | 451245 | 446468 | 19150 | 4157 | 4113 |

Table 4.4: Number of samples and labels in each set for the different fault categories

We choose to use 70% of the faults for the training set, 15% for the validation set, and 15% for the test set. The resulting number of samples and labels for each set can be seen in table 4.4.

## 4.4 Data Cleaning and Preprocessing

This section describes the transformations that our data need to undergo before being used as input data to train our classifiers.

### 4.4.1 Handling Missing Data

Features with more than 99% missing data are excluded. For the remaining columns, missing values are imputed with -1, and a binary indicator column is added in order to enable the model to use the presence of a missing value as a feature of its own. We note that more effort could be put into this step by analyzing the characteristics of each column, and try more elaborate techniques for imputing missing data. Some of the possible options include filling with last known value, filling with mean or median, or filling with values from nearest neighbors. This requires more domain specific information about the sensors and how their measurements are collected, and is not implemented in the scope of this project. Implementing more advanced imputation strategies would also result in increased complexity of the system. We adopt the premise that deep learning models are able to learn the representations of missing values by its own through an indicator feature.

### 4.4.2 Feature Scaling

Before the input data is fed into the neural network we want to scale the values to ensure input features are in the same range. We to this by removing the mean

and scaling to unit variance for all numeric features. For the categorical features, we want to use an integer representation as input to the embedding layer. It is important to note that scaling is part of the preprocessing, and that the scaling parameters should only be derived from the training data to avoid leakage of information from the validation data. After the scaling parameters are derived from the training data, they may be applied to the validation and test data. It should be noted that this scaling is best suitable for features that are normal distributed, and will yield subpar results for features that are heavily skewed and/or have extreme values.

### 4.4.3 Feature Engineering

One of the main advantages with deep learning is that the method is well suited to learn relevant features on its own, hence reducing the effort needed for explicit feature engineering. This does not mean that we would not benefit for explicitly engineering features as they may facilitate better learning. Anecdotally, this has the potential to improve performance of deep learning models substantially. This will, however, not be the focus for this project, as feature engineering requires domain expertise, and that an interesting part of the project is to see if the deep learning models are able to extract these features on their own.

### 4.4.4 Categorical Variable Encoding

For the categorical variables "Turbine" and "String", we will encode the categories as integers. These will be used as input to an embedding layer, as discussed in section. 4.7.4.

### 4.4.5 Sample Generation

This section will explain the procedure for generating training samples from the labelled data. Our labelled data contains 67 data points (one for each turbine) for each 10-minute intervals (unless the point has been removed according to 4.2).

When we generate training samples we want to generate samples from a sliding window of data points, while making sure that the elements of the sliding window all originate from the same turbine. The window length is configurable, and denoted $w$. This means that we can generate $n - (w \cdot g)$ samples, where $n$ is number of original data points, $w$ is the sliding window length, $g$ is the number of turbine groups [1], given that $w$ is smaller than the number of data points for all turbine groups. This condition is satisfied for all experiments in this project,

[1] The term turbine group is used to describe the subset of data points originating from one common turbine

as the maximum $w$ that will be explored is 36, corresponding to 6 hours, and all turbine groups have a lot more samples than this.

The categorical features that we want to feed into an embedding layer need to be split into its own array. This means that one training sample should consist of a tuple consisting of (sample, targets), where the sample is a list of arrays, with one array for each of the categorical columns, and one array for all the continuous input columns. With two categorical features and 167 continuous features, a sample will be a list containing three arrays with dimensionality $(w, 1)$, $(w, 1)$, and $(w, 167)$. The corresponding target will be a binary 0 or 1 depending on whether the corresponding target has a PF label. This is illustrated in figure 4.6, where an actual fault occurred at 2018-12-31 23:57:12.

| | TimeStamp | Turbine | Col_1 | Col_2 | Label | |
|---|---|---|---|---|---|---|
| Last sample for turbine B01 | 2018-12-31 23:00:00 | B01 | - | - | 0 | |
| | 2018-12-31 23:10:00 | B01 | - | - | 0 | |
| | 2018-12-31 23:20:00 | B01 | - | - | 0 | |
| | 2018-12-31 23:30:00 | B01 | - | - | 0 | |
| | 2018-12-31 23:40:00 | B01 | - | - | 0 | |
| | 2018-12-31 23:50:00 | B01 | - | - | 1 | Last target for turbine B01 |
| First sample for turbine B02 | 2017-01-07 12:00:00 | B02 | - | - | 0 | |
| | 2017-01-07 12:10:00 | B02 | - | - | 0 | |
| | 2017-01-07 12:20:00 | B02 | - | - | 0 | |
| | 2017-01-07 12:30:00 | B02 | - | - | 0 | |
| | 2017-01-07 12:40:00 | B02 | - | - | 0 | |
| | 2017-01-07 12:50:00 | B02 | - | - | 0 | First target for turbine B02 |

Figure 4.6: Illustration of sample generation with $w = 6$

In the figure above, note that the PF window is only one timestep (10-minutes). For a bigger PF window, more samples before the fault occurred would be labelled.

When generating samples and targets for training the neural network, we want to be able to generate batches on the fly, rather than to generate all samples in advance, to avoid memory issues. We could also generate all samples and save them to disk in advance, but loading from disk during training suffers from disk read speed limitations. It will also speed up the training process to take advantage of multiprocessing for the batch generation. A generator conforming to the requirements in this section was implemented.

## 4.5 Alarm Control System

Our output from the models represent one predicted probability of the turbine being in a PF state per turbine per 10-minute sample. These predictions may

be noisy, and both Yang et al. [2018] and Wang et al. [2017a] have shown that smoothing predictions before giving an actual alarm might improve performance of the whole system, by reducing the number of false alarms. This means that high probabilities of an occurring fault must be sustained for a period before the predicted probabilities result in an actual alarm. In order to give more weight to the most recent predictions, we use an exponentially weighted moving average (EWMA) to calculate the filtered prediction probability. In general a weighted moving average is calculated as

$$y_t = \frac{\sum_{i=0}^{t} w_i x_{t-i}}{\sum_{i=0}^{t} w_i} \tag{4.1}$$

where $x_t$ is the input, $y_t$ is the result and the $w_i$ are the weights. We use the variant where weights are not calculated recursively, and each weight is given by $w_i = (1-\alpha)^i$. This gives

$$y_t = \frac{x_t + (1-\alpha)x_{t-1} + (1-\alpha)^2 x_{t-2} + ... + (1-\alpha)^t x_0}{1 + (1-\alpha) + (1-\alpha)^2 + ... + (1-\alpha)^t} \tag{4.2}$$

where $\alpha$ is the smoothing factor. In our case, we calculate $\alpha$ by providing a desired span, $s$. $\alpha$ is then given by $\alpha = \frac{2}{s+1}$.

In figure 4.7, we illustrate this control system by displaying an actual fault, the pre-fault period with labelled samples, the raw predictions, the filtered predictions (as calculated by EWMA with $s = 6$), and whether an alarm is given or not. The binary event of a generated alarm, $A(t)$ is given according to the following rule:

$$A(t) = \begin{cases} 1, & \text{if } y(t) \geq b \\ 0, & \text{otherwise} \end{cases}$$

Here, $b$ is a threshold that optimizes the desired F score on the validation set. In our case, we use the F1 score. This is chosen in absence of knowledge about the desired weight from a business perspective. Depending on the cost of investigating a false alarm and the potential savings by preventing a failure, the weight of precision/recall should be adjusted. If the cost of a false alarm and the savings of avoiding a certain fault were available, one might even optimize this threshold directly for monetary value.

In figure 4.7, the threshold $b$ is set to 0.6.

## 4.6 Evaluation metric

This section will detail how the different algorithms will be evaluated. Note that this is a different metric than that of which the different algorithms are optimized

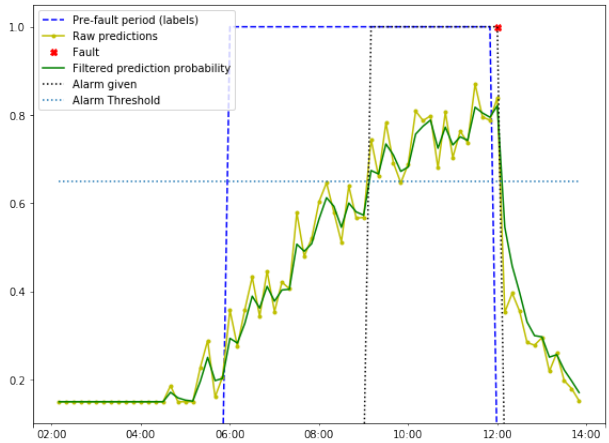



Figure 4.7: Illustration of Alarm Control System

for. The optimization metric differs between algorithms, but we need to establish a common evaluation metric for the algorithms that most accurately relates to performance in a real-world setting. What we are really interested in is how many of the alarms we are able to detect, and how many false alarms we have to accept to achieve this detection.

Once a binary prediction is made by the alarm control system from the previous section, we want to use common metrics of a binary classifier to evaluate the usefulness of the classifier. Before we can do this, it is important to relabel the true labels back so that only the observations where an actual fault first occurred are labelled (The $F$ label). This will allow us to get an interpretable metric, where we can extract the number of faults detected, as well as the total number of false alarms. We will define a positively identified fault as a case for which there was given an alarm within a window of $2 \cdot w_1$ in advance, i.e., twice the pre-fault period. The reasoning for this is that even though the classifier was trained on pre-fault states that occurred only in a window $w_1$ before an actual fault, some pre-fault states may occur even earlier before they manifest into a fault, but the classifier will still correctly recognize the pre-fault state. Due to the small number of faults, it is considered less likely that this will reduce the fairness of the evaluation by detecting faults by chance than the opposite case, where a positive detection is not counted just because it came too early (which should be considered good). More elaborate methods for determining this window size is subject to future work.

## 4.7 Deep Learning Models

This section will describe the architecture of the deep learning models that will be evaluated for this thesis. The architectures are based on the premise that we want to investigate and compare different generic architectures. Ideally, we would like to compare models with similar capacity, or number of parameters. This proved difficult in practice, due to the fact that the number of parameters changes differently with regards to the input dimensionality, caused by different PF windows. We note that models with an even larger number of parameters and more aggressive/intricate regularization techniques might perform even better, and is suggested for future work. The implementations are kept rather simple for all the models, as we want to get an impression of the performance of generic architectures first, and form a starting point which might be improved upon and specialized further in future work.

### 4.7.1 Output layer

As our problem deals with binary classification, all deep learning models will have a final output layer with one unit and a sigmoid activation function, which forces the output to have a value between 0 and 1.

### 4.7.2 Loss function

In order to handle the extremely imbalanced classes for this problem, we implement a cost-sensitive learning approach for the neural networks, where the cost of misclassifying an example is adjusted to match the class distribution. This is emphasized as a critical component to enable the use of deep learning on such an imbalanced dataset.

To describe the weighted binary crossentropy loss function, we first introduce the unweighted binary crossentropy.

Consider the unweighted binary cross entropy, given by

$$H(y, p) = -y \log(p) + (1 - y) \log(1 - p) \tag{4.3}$$

With meaning of variables as described below:

- H(y,p): The binary cross-entropy cost of predicting p on a sample with label y.
- y: Actual label.
- p: predicted probability of sample being in the positive class.

For the weighted case, we simply introduce a weight for each class, $W(y)$, where the weight is the number of total samples, $n_{tot}$, divided by the number of samples with given label, $n_y$:

$$W_y = \frac{n_{tot}}{n_y} \tag{4.4}$$

This factor can subsequently be used to modify the equation 4.3 to:

$$H_{weighted}(y, p) = -W_0 \cdot (y \log(p)) - W_1 \cdot (1 - y) \log(1 - p)) \tag{4.5}$$

This will be used as the loss function for all deep learning models. It is important that the weight will be calculated based on our actual faults for each fault category, and not the number of PF-labels. This was done initially, but led to overfitting early in training. As an alternative method for handling class imbalance, random undersampling of the majority class to achieve balanced classes was also prototyped. This resulted in a classifier with predictions no better than simple chance. A possible explanation for this was probably the small amount of data this method utilized.

It should be noted that a loss function that attempted to optimize F1 score directly by introducing a differentiable approximation of the F1 score was also prototyped. The reasoning behind the attempt was that this loss function would transfer better to our final evaluation metric, as it was observed that neither this loss function nor the AUC-metric always correlated very well with the final evaluation metric. This direction was abandoned, as the F1 score that would be optimized in training was based on the PF labels, rather than the actual faults. This resulted in the models "specializing" on detecting a number of samples that were all related to the same original faults, rather than modelling all of the actual faults as we would like to. It appeared that optimizing for the approximated F1 score for each batch led to the loss function getting stuck in a local minimum early in training. This loss function did thus not transfer better to our final evaluation metric than the weighted binary crossentropy as described above.

### 4.7.3 Diagram notation

This is a short explanation of the notation used in the diagrams below. The names of each box (the part before the colon) is just a unique name for each layer, while the part after the colon denotes the type of layer used for each box. These types corresponds with the keras class of the layer that is used. For more in depth explanation of each layer class, the reader is referred to the keras documentation.[2]

[2]https://keras.io

### 4.7.4 Common Embedding Module

To enable the network to learn embedding representations of the categorical variables, we need to initialize an embedding vector that will be learned during training. This will be implemented for all models. We need to specify the dimensionality, $d$ of this embedding vector for each of the categorical variables. This is chosen based on heuristic advise from the fast.ai course [Howard, 2018], which can be seen in equation 4.6.

$$d = min(\lfloor \frac{c+1}{2} \rfloor, 50) \tag{4.6}$$

where $c$ is the cardinality, or number of unique values for for the variable.

The output of the embedding layers are subsequently concatenated with the numeric features. For our case, we have two categorical features, "Turbine" and "String", for which we apply embedding layers. These two are concatenated with all the numeric features. This is illustrated in 4.8.

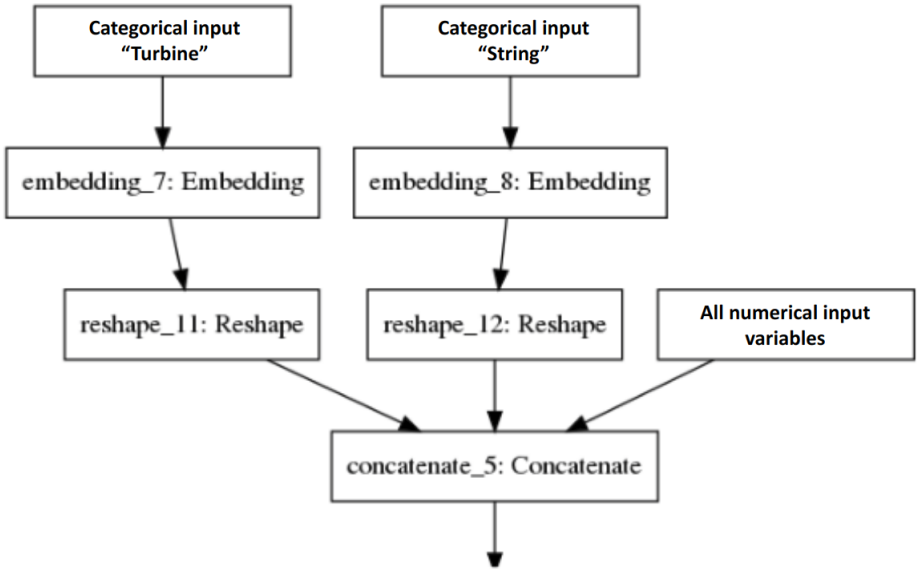



Figure 4.8: Input Module with Embedding

The resulting input module will be used for all deep learning architectures in this chapter.

### 4.7.5 MLP Architecture

The MLP implemented in this project is simply 3 layers of fully connected layers, with dropout applied between the layers. The input is flattened, so that all the features for all timesteps are fed into a dense layer. We can see that since all features from all timesteps are flattened, the size of the network will depend heavily on the number of timesteps.

The architecture is shown in figure 4.9.

### 4.7.6 CNN Architecture

The CNN architecture that we will use for this project consist of a single 1-dimensional convolutional layer, which will apply convolutions to the temporal dimension. We also adopt the squeeze-excite block introduced in section 2.11.1 in an attempt to better model feature interdependencies. A fully connected layer is also added to every timestep of the input to enable learning independent of timesteps. The architecture is shown in figure 4.10

### 4.7.7 LSTM Architecture

This model consists of two stacked LSTM cells. These are configured to return the output for the whole sequence. A prototype for which only the last state was returned was also tested, with poor results. A time distributed fully connected layer is added here as well. The architecture of the LSTM-network is shown in figure 4.11.

### 4.7.8 Hybrid Multivariate Time Series Network (HMVTS) Architecture

This model is a combination of both the MLP, CNN, and LSTM models described in the previous subsections. The respective vectors of these submodels before the top layer are concatenated together, with a small dense layer on top, before the common output layer is applied. The hope is that this architecture will combine the benefits of each of the submodels and perform even better. This model has a larger capacity and number of parameters than the submodels, and is thus more likely to overfit. To mitigate this risk, this model employs a bit more rigorous regularization than the other networks.

The architecture is shown in figure 4.12

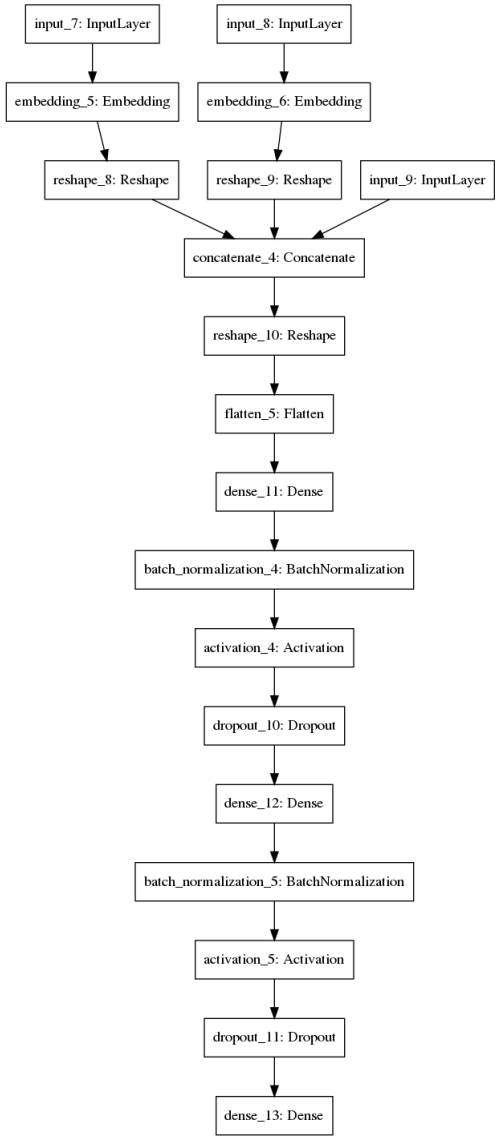



Figure 4.9: MLP Block Diagram

### 4.7.9 Regularization Methods

With the small number of faults we have, overfitting is a large challenge for this problem. Several regularization methods are applied to the deep learning models in order to facilitate generalization. All the regularization methods described in section 2.14 are employed to address this issue, and listed below.

- Early Stopping
- Dropout
- Weight Regularization
- Activity Regularization
- Batch Normalization

In general, increased performance on the validation data was observed for each of the techniques added, although no experiments to quantify this was performed.

## 4.8 Chapter summary

In this chapter, the architecture of the proposed system has been presented. First, an overview of the main components was provided, before each component was described in more detail. Several instances of the system described in this chapter will be empirically evaluated in the next chapter. The main focus will be to examine the effect of the different classifier components described in this chapter.

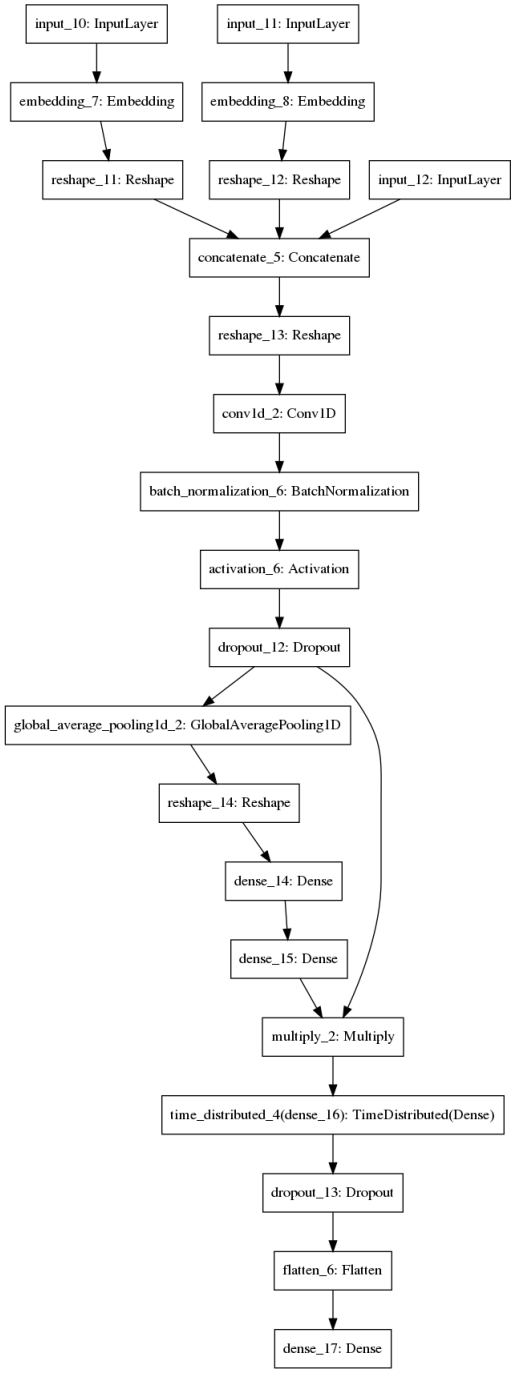



Figure 4.10: CNN Block Diagram

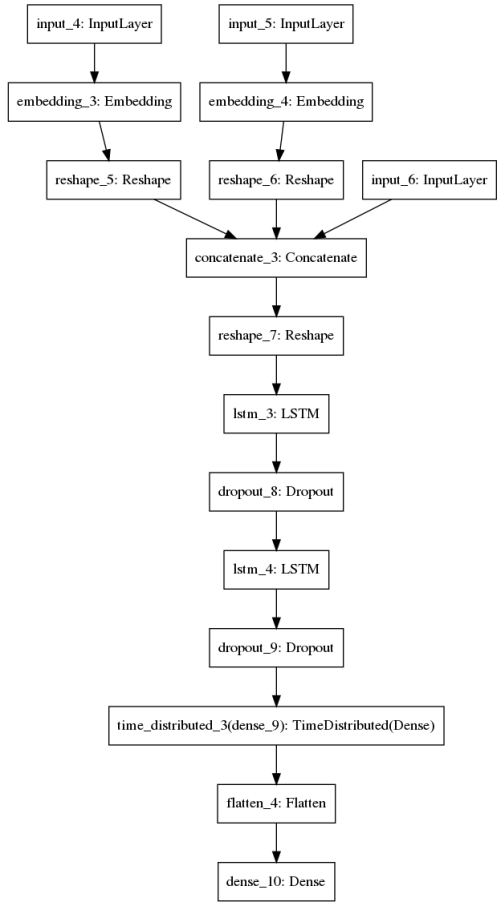



Figure 4.11: LSTM Block Diagram

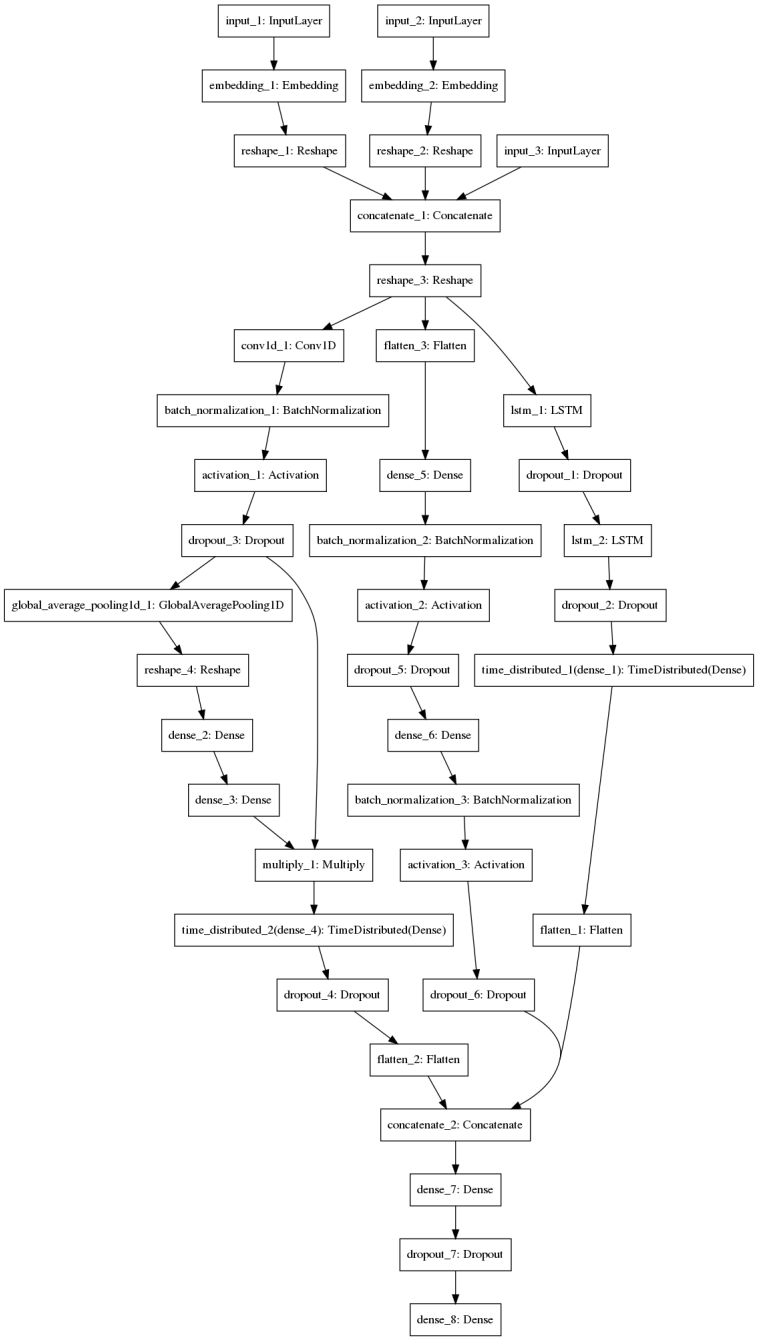



Figure 4.12: HMVTS Block Diagram

# Chapter 5

# Experiments and Technology

This chapter will describe experiments that have been done on the dataset provided by Equinor. All experiments have been conducted using the Python programming language with additional libraries as described in this chapter.

## 5.1 Hardware

The experiments were performed on a computer with AMD Ryzen 1700X 8-core CPU, NVIDIA GeForce GTX 1080 Ti GPU, and 64GB RAM. The OS on the computer was Ubuntu 18.04.

## 5.2 Software

This section lists the software tools and libraries used to perform the experiments. A short introduction to each library is included. For further documentation, the reader is referred to the reference documentation of each library. The source code for the whole project is available upon request. The code is considered too lengthy to include in an appendix, and Equinor ASA does not wish to make the software publicly available until it has been screened for sensitive information.

### 5.2.1 Pandas

Pandas [McKinney, 2010] is a versatile open source library for data analysis and data wrangling. Most of the data analysis and preprocessing for this project have

been accomplished through the use of Pandas, which has been a valuable tool in this process.

### 5.2.2 Scikit-learn

Scikit-learn [Pedregosa et al., 2011] is an open-source machine learning library for the Python programming language. It provides an easy-to-use API for a broad range of preprocessing algorithms, machine learning algorithms, and evaluation metrics. In this project, scikit-learn has been used for some preprocessing, the baseline algorithm (Random Forest), as well as several evaluation metrics.

### 5.2.3 TensorFlow

TensorFlow [Abadi et al., 2015] is an interface for expressing and executing machine learning algorithms, with support for execution on distributed, heterogenous systems, including GPUs. Neural network computations are much more efficient to process on GPUs than on CPUs, due to the GPU's superior ability to perform parallel calculations. In this project, TensorFlow is used as the backend computation library for Keras, with some minor custom TensorFlow functions used for evaluation.

### 5.2.4 Keras

Keras [Chollet et al., 2015] is an API for creating and training neural networks. Keras has support for multiple backends, where TensorFlow is one of the available backends. Keras provides a simpler API than TensorFlow, with the opportunity to add specific TensorFlow components seamlessly, if needed. A nice thing about Keras, is that the neural network models created in Keras support a scikit-learn API, so that these models can be used in a similar manner as scikit-learn models in a full pipeline.

### 5.2.5 MLFlow

MLFlow [2019] is an open source platform for handling the machine learning lifecycle. It facilitates structured logging of parameters, results and artifacts from ML experiments, and has been very helpful in organizing the experimentation part. It should be noted that MLFlow is in a beta release, and may be subject to breaking changes.

## 5.3 Hyperparameters

### 5.3.1 Baseline model: Random Forest

For the random forest, I did a grid search to find the best combination of the following hyperparameter options:

- max_depth: [1, 3, 5, 7, 10]
- max_features: [5, 8, 10, 20]

The model that provided the best results on the validation set was used for evaluation on the test set.

### 5.3.2 Neural Networks

The hyperparameters were selected by heuristics supported by initial trial and error. If severe overfitting was observed during initial trials, the number of parameters for the networks were reduced and/or regularization increased. No extensive hyperparameter search was performed for the deep learning models due to time and resource constraints. This is something that will be subject to future work. The following hyperparameters are the same for all models:

- optimizer: Adam [Kingma and Ba, 2014].
- batch_size: 256.
- learning_rate: 0.0001.
- loss_function: binary_crossentropy (with modified costs). See equation 4.3.
- early_stopping_metric: F1 score for actual faults on validation set.
- early_stopping_patience: 3 epochs.
- epochs: 20. (But stops before if no improvement in F1 score for actual faults on validation set for 3 epochs).
- dropout_fraction: 0.5. (HMVTS: 0.6)
- weight_regularizer: L2(0.001). (HMVTS: L2(0.005).)
- activity_regularizer: L2(0.001). (HMVTS: L2(0.005).)

A callback is implemented for the deep learning models to restore the weights from the epoch that yields the best F1 score on the actual faults in the validation set after training is finished. The structure of the neural networks should also be considered hyperparameters, but these were described in the corresponding section of chapter 4.

## 5.4 Results

In total, 50 experiments were performed, on 5 different fault categories, with two PF window lengths for each fault category, for each of the 5 model types (RF, MLP, CNN, LSTM and HMVTS). This means that a lot of data is collected. The corresponding ROC curves and training history (for the neural networks) for the individual models are presented in appendices A and B for brevity.

This section will provide a detailed walk-through of the results. These results are considered to be the key findings of this thesis, and all results are thus included for completeness. It should be noted that these results contain a lot of information, and that the aggregated results and findings will be summarized in the next section.

### 5.4.1 Random Forest

Results can be seen in table 5.1

| Fault category | PF window | n_faults | faults detected | missed faults | false alarms | tn | threshold | precision | recall | val auc | test auc | f1 score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M005 | 1h | 55 | 25 | 30 | 7470 | 130495 | 0.50 | 0.003 | 0.455 | 0.947 | 0.842 | 0.007 |
| M076 | 1h | 75 | 2 | 73 | 238 | 568517 | 0.84 | 0.008 | 0.027 | 0.923 | 0.891 | 0.013 |
| M079 | 1h | 73 | 18 | 55 | 6114 | 525726 | 0.50 | 0.003 | 0.247 | 0.992 | 0.715 | 0.006 |
| M099 | 1h | 71 | 4 | 67 | 198 | 727118 | 0.85 | 0.020 | 0.056 | 0.932 | 0.878 | 0.029 |
| M104 | 1h | 126 | 5 | 121 | 55 | 448841 | 0.76 | 0.083 | 0.040 | 0.883 | 0.737 | 0.054 |
| M005 | 6h | 64 | 7 | 57 | 4720 | 134014 | 0.79 | 0.001 | 0.109 | 0.971 | 0.638 | 0.003 |
| M076 | 6h | 83 | 15 | 68 | 15448 | 532697 | 0.97 | 0.001 | 0.181 | 0.781 | 0.702 | 0.002 |
| M079 | 6h | 71 | 13 | 58 | 35631 | 527567 | 0.98 | 0.000 | 0.183 | 0.856 | 0.592 | 0.001 |
| M099 | 6h | 69 | 17 | 52 | 96662 | 630106 | 0.92 | 0.000 | 0.246 | 0.804 | 0.585 | 0.000 |
| M104 | 6h | 127 | 56 | 71 | 101455 | 344886 | 0.97 | 0.001 | 0.441 | 0.809 | 0.672 | 0.001 |

Table 5.1: Results RF

Remember that the RF model uses only one data point to make its prediction. Still, the labels are the same as for the other models. It is obvious that the RF model performs worse when the PF window is 6h than for 1h. This indicates that the amount of signal may be less in the samples that are positively labelled in the 6h PF window, and that we are training on very noisy labels. To elaborate, it is likely that labeling all data points in a 6h PF window means that we are introducing a lot of positive examples that do not actually contain information that has any correlation with the actual fault.

### 5.4.2 MLP

Results can be seen in table 5.2.

| parameters | Fault category | PF window | n_faults | faults detected | missed faults | false alarms | tn | threshold | precision | recall | train auc | val auc | test auc | f1 score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 42604 | M005 | 1h | 54 | 10 | 44 | 617 | 136545 | 0.73 | 0.016 | 0.185 | 0.979 | 0.933 | 0.723 | 0.029 |
| 42604 | M076 | 1h | 75 | 4 | 71 | 279 | 567966 | 0.89 | 0.014 | 0.053 | 0.914 | 0.982 | 0.815 | 0.022 |
| 42604 | M079 | 1h | 73 | 3 | 70 | 765 | 530618 | 0.75 | 0.004 | 0.041 | 0.817 | 0.893 | 0.560 | 0.007 |
| 42604 | M099 | 1h | 71 | 7 | 64 | 74 | 726383 | 0.89 | 0.086 | 0.099 | 0.883 | 0.930 | 0.837 | 0.092 |
| 42604 | M104 | 1h | 126 | 6 | 120 | 96 | 448290 | 0.66 | 0.059 | 0.048 | 0.867 | 0.911 | 0.682 | 0.053 |
| 240364 | M005 | 6h | 63 | 6 | 57 | 1855 | 134274 | 0.94 | 0.003 | 0.095 | 0.892 | 0.612 | 0.592 | 0.006 |
| 240364 | M076 | 6h | 83 | 10 | 73 | 7807 | 537902 | 0.98 | 0.001 | 0.120 | 0.742 | 0.772 | 0.712 | 0.003 |
| 240364 | M079 | 6h | 71 | 1 | 70 | 791 | 559778 | 0.96 | 0.001 | 0.014 | 0.699 | 0.685 | 0.546 | 0.002 |
| 240364 | M099 | 6h | 69 | 3 | 66 | 1040 | 722859 | 0.99 | 0.003 | 0.043 | 0.764 | 0.681 | 0.684 | 0.005 |
| 240364 | M104 | 6h | 127 | 71 | 56 | 96070 | 347707 | 0.88 | 0.001 | 0.559 | 0.851 | 0.728 | 0.715 | 0.001 |

Table 5.2: Results MLP

We note that the MLP model with 6h PF window becomes a rather large model with respect to number of parameters.

### 5.4.3 CNN

Results can be seen in table 5.3.

| parameters | Fault category | PF window | n_faults | faults detected | missed faults | false alarms | tn | threshold | precision | recall | train auc | val auc | test auc | f1 score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 22644 | M005 | 1h | 54 | 1 | 53 | 467 | 136695 | 0.55 | 0.002 | 0.019 | 0.536 | 0.675 | 0.494 | 0.004 |
| 22644 | M076 | 1h | 75 | 15 | 60 | 658 | 567587 | 0.54 | 0.022 | 0.200 | 0.654 | 0.681 | 0.562 | 0.040 |
| 22644 | M079 | 1h | 73 | 27 | 46 | 1794 | 529589 | 0.53 | 0.015 | 0.370 | 0.614 | 0.741 | 0.553 | 0.029 |
| 22644 | M099 | 1h | 71 | 21 | 50 | 378 | 726079 | 0.61 | 0.053 | 0.296 | 0.77 | 0.851 | 0.775 | 0.089 |
| 22644 | M104 | 1h | 126 | 31 | 95 | 489 | 447897 | 0.62 | 0.060 | 0.246 | 0.793 | 0.933 | 0.715 | 0.096 |
| 22884 | M005 | 6h | 63 | 1 | 62 | 907 | 135222 | 0.63 | 0.001 | 0.016 | 0.602 | 0.724 | 0.474 | 0.002 |
| 22884 | M076 | 6h | 83 | 18 | 65 | 8462 | 537247 | 0.77 | 0.002 | 0.217 | 0.61 | 0.669 | 0.596 | 0.004 |
| 22884 | M079 | 6h | 71 | 1 | 70 | 7019 | 553550 | 0.94 | 0.000 | 0.014 | 0.553 | 0.381 | 0.429 | 0.000 |
| 22884 | M099 | 6h | 69 | 69 | 0 | 723783 | 116 | 0.62 | 0.000 | 1.000 | 0.519 | 0.516 | 0.513 | 0.000 |
| 22884 | M104 | 6h | 127 | 18 | 109 | 19801 | 423976 | 0.98 | 0.001 | 0.142 | 0.836 | 0.790 | 0.753 | 0.002 |

Table 5.3: Results CNN

The number of parameters for the CNN model only changes slightly for the different PF windows. This raises the question that maybe the number of parameters for this model was too small to capture the signal of the labels with 6h PF window.

### 5.4.4 LSTM

Results can be seen in table 5.4.

As can be seen from the table, the LSTM network learned to classify almost all examples as PF labels for three of the experiments ((M005, 1h), (M076, 1h), and (M099, 1h)). This naturally is a very poor classifier. A possibly explanation is that the LSTM is more sensitive to the large weight updates caused by our

| parameters | Fault category | PF window | n_faults | faults detected | missed faults | false alarms | tn | threshold | precision | recall | train auc | val auc | test auc | f1 score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 33844 | M005 | 1h | 54 | 54 | 0 | 137151 | 11 | 0.00 | 0.000 | 1.000 | 0.5 | 0.502 | 0.500 | 0.001 |
| 33844 | M076 | 1h | 75 | 75 | 0 | 568234 | 11 | 0.00 | 0.000 | 1.000 | 0.5 | 0.491 | 0.495 | 0.000 |
| 33844 | M079 | 1h | 73 | 6 | 67 | 129 | 531254 | 0.65 | 0.044 | 0.082 | 0.753 | 0.860 | 0.592 | 0.058 |
| 33844 | M099 | 1h | 71 | 71 | 0 | 726446 | 11 | 0.00 | 0.000 | 1.000 | 0.5 | 0.500 | 0.500 | 0.000 |
| 33844 | M104 | 1h | 126 | 16 | 110 | 284 | 448102 | 0.65 | 0.053 | 0.127 | 0.87 | 0.897 | 0.691 | 0.075 |
| 34084 | M005 | 6h | 63 | 6 | 57 | 6920 | 129209 | 0.72 | 0.001 | 0.095 | 0.54 | 0.552 | 0.570 | 0.002 |
| 34084 | M076 | 6h | 83 | 8 | 75 | 13923 | 531786 | 0.93 | 0.001 | 0.096 | 0.84 | 0.664 | 0.630 | 0.001 |
| 34084 | M079 | 6h | 71 | 1 | 70 | 806 | 559763 | 0.76 | 0.001 | 0.014 | 0.691 | 0.625 | 0.529 | 0.002 |
| 34084 | M099 | 6h | 69 | 16 | 53 | 64754 | 659145 | 0.73 | 0.000 | 0.232 | 0.829 | 0.666 | 0.683 | 0.000 |
| 34084 | M104 | 6h | 127 | 64 | 63 | 46058 | 397719 | 0.75 | 0.001 | 0.504 | 0.613 | 0.637 | 0.529 | 0.003 |

Table 5.4: Results LSTM

weighted loss function, and may be exposed to the exploding gradient problem [Graves, 2013]. One option that might be useful to explore in order to mitigate this problem, is to clip the gradient. When additional trials were performed with different random seeds, the results were different, indicating that the weight initialization and the order the samples are fed to the network matter. An interesting result is the LSTM's performance for (M079 1h), for which it beats all the other models with a rather large margin (F1 score of 0.058 vs 0.029 for the 2nd best model). Still, the number of faults detected in this case was only 6, so it is not possible to make definitive conclusions from this result.

### 5.4.5 HMVTS

Results can be seen in table 5.5 .

| parameters | Fault category | PF window | n_faults | faults detected | missed faults | false alarms | tn | threshold | precision | recall | train auc | val auc | test auc | f1 score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 94492 | M005 | 1h | 54 | 2 | 52 | 457 | 136705 | 0.53 | 0.004 | 0.037 | 0.909 | 0.931 | 0.705 | 0.008 |
| 94492 | M076 | 1h | 75 | 2 | 73 | 1 | 568244 | 0.88 | 0.667 | 0.027 | 0.9 | 0.989 | 0.809 | 0.051 |
| 94492 | M079 | 1h | 73 | 3 | 70 | 319 | 531064 | 0.54 | 0.009 | 0.041 | 0.719 | 0.785 | 0.559 | 0.015 |
| 94492 | M099 | 1h | 71 | 19 | 52 | 559 | 725898 | 0.72 | 0.033 | 0.268 | 0.902 | 0.969 | 0.840 | 0.059 |
| 94492 | M104 | 1h | 126 | 18 | 108 | 483 | 447903 | 0.60 | 0.036 | 0.143 | 0.874 | 0.972 | 0.805 | 0.057 |
| 294172 | M005 | 6h | 63 | 63 | 0 | 136058 | 71 | 0.00 | 0.000 | 1.000 | 0.587 | 0.600 | 0.493 | 0.001 |
| 294172 | M076 | 6h | 83 | 10 | 73 | 3200 | 542509 | 0.94 | 0.003 | 0.120 | 0.732 | 0.730 | 0.666 | 0.006 |
| 294172 | M079 | 6h | 71 | 1 | 70 | 203 | 560366 | 0.97 | 0.005 | 0.014 | 0.698 | 0.731 | 0.602 | 0.007 |
| 294172 | M099 | 6h | 69 | 2 | 67 | 625 | 723274 | 0.94 | 0.003 | 0.029 | 0.777 | 0.692 | 0.686 | 0.006 |
| 294172 | M104 | 6h | 127 | 1 | 126 | 327 | 443450 | 0.99 | 0.003 | 0.008 | 0.83 | 0.721 | 0.772 | 0.004 |

Table 5.5: Results HMVTS

The HMVTS seem to suffer from the same exploding gradient problem as discussed in the previous section on one of the experiments (M005, 6h).

## 5.5 Summary of results

This section will aggregate and summarize the most interesting results.

### 5.5.1 PF window comparison

It is immediately obvious that the final evaluation metrics become worse across the board with a PF window of 6h compared to 1h. The differences are an order of magnitude lower F1 score. This is to be expected, as we know that we are introducing more noisy labels. At the same time, if we are able to model this signal, the predictions might prove more useful anyway, as the available time window to mitigate the fault is substantially larger.

### 5.5.2 Fault category comparison

It is interesting to compare the results of different fault categories. This will provide insight into which of the faults for which the pre-fault SCADA data contain the most signal, and thus is most suited to be predicted in advance. We not that two categories, M104 Yaw System and M099 TS-converter system, seem to be the fault categories best suited for modelling. This is not valid for the 6h PF window, where the category M076 Blade System is the one that seem to be best suited for modelling, but the difference between the categories are smaller for experiments with 6h PF window.

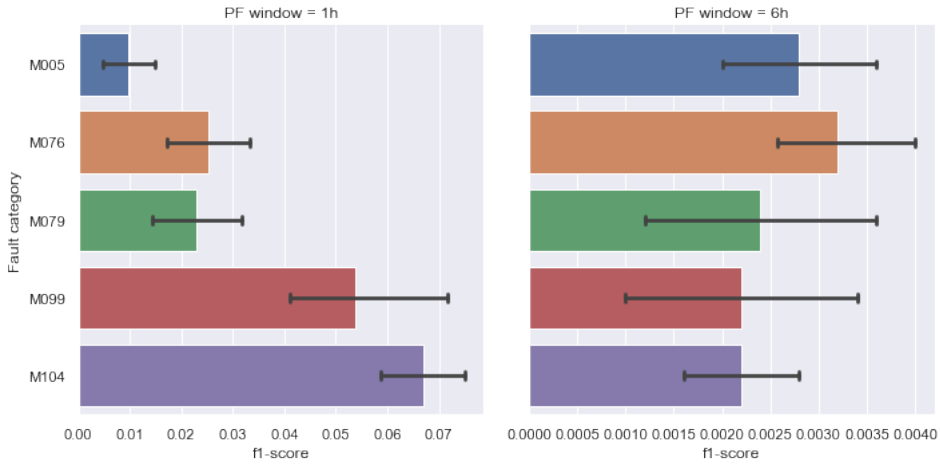



Figure 5.1: F1 score for the fault categories on both PF windows. Note that the values for PF window = 6h is an order of magnitude smaller.

### 5.5.3 Model comparison

In table 5.6, we can see the comparison of models for each of the experiments.

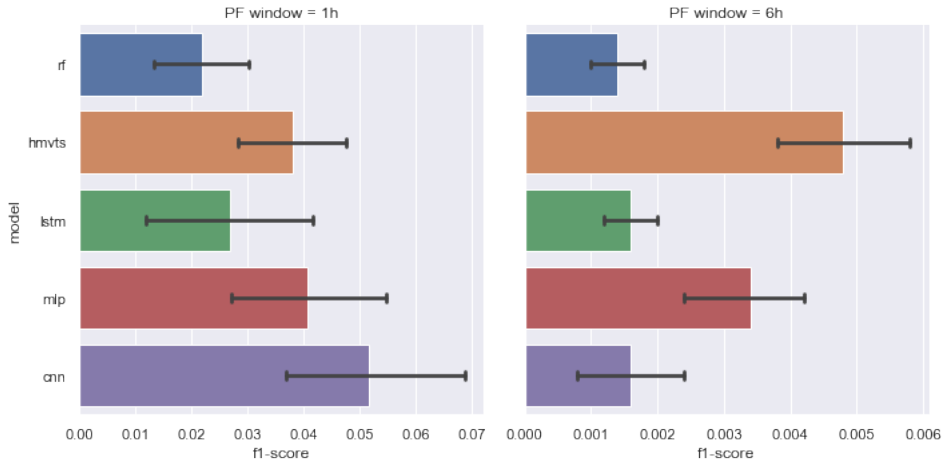



Figure 5.2: F1 score for the different models on both PF windows. Note that the values for PF window = 6h is an order of magnitude smaller.

The rather simple CNN is the strongest performing model for the 1h PF window experiments. This is an interesting result, and supports the conclusions of [Wang et al., 2017b].

It is interesting to note that the two models with the largest number of parameters are the best performers on the 6h PF window experiments. This may indicate that these models with larger capacity are better suited to capture the noisy signal in these labels.

If we would have been forced to choose only one model to be implemented as classifier for all categories, we could compare the mean rank of the models for both PF windows.

In figure 5.3, we can see that the random forest model was the worst performing model for both PF windows. The best performing models were the CNN, and HMVTS, for PF windows 1h and 6h respectively. The importance of this is limited though, as it is more likely that the models will be chosen individually, and only implemented for the combinations of fault category and PF window where the F1 score is good enough to support the business case.

### 5.5.4 Metric comparison

A significant discrepancy between the F1 and AUC metric was observed. This might be explained by the different model characteristics, as well as the possibly over-optimistic illusion given by the ROC-AUC metric as discussed in 2.7.7. An

| | model | | | | |
|---|---|---|---|---|---|
| Fault category, PF window | cnn | hmvts | lstm | mlp | rf |
| M005 1h | 0.004 | 0.008 | 0.001 | **0.029** | 0.007 |
| M005 6h | 0.002 | 0.001 | 0.002 | **0.006** | 0.003 |
| M076 1h | 0.040 | **0.051** | 0.000 | 0.022 | 0.013 |
| M076 6h | 0.004 | **0.006** | 0.001 | 0.003 | 0.002 |
| M079 1h | 0.029 | 0.015 | **0.058** | 0.007 | 0.006 |
| M079 6h | 0.000 | **0.007** | 0.002 | 0.002 | 0.001 |
| M099 1h | 0.089 | 0.059 | 0.000 | **0.092** | 0.029 |
| M099 6h | 0.000 | **0.006** | 0.000 | 0.005 | 0.000 |
| M104 1h | **0.096** | 0.057 | 0.075 | 0.053 | 0.054 |
| M104 6h | 0.002 | **0.004** | 0.003 | 0.001 | 0.001 |

Table 5.6: F1 score for the different models on each of the fault categories and PF windows. (Best score in bold)

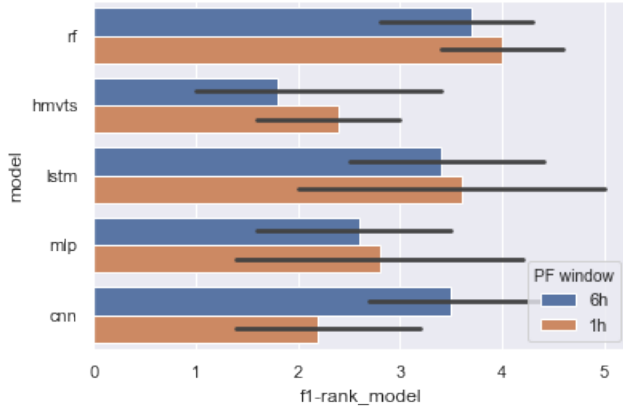



Figure 5.3: Model F1 rank on both PF windows

additional explanation is that some models might be better suited to model a broader range of faults, while some are more likely to specialize on a few faults with more pronounced characteristics, and modelling all labels belonging to these faults. The latter will yield a poor F1 score on the actual faults, but might still give a good AUC.

In figure 5.4, we can see the AUC vs F1 for the different models.

If these two metrics were directly correlated, we would observe a diagonal line to which the points could be fitted. This is not the case. We can see that the RF

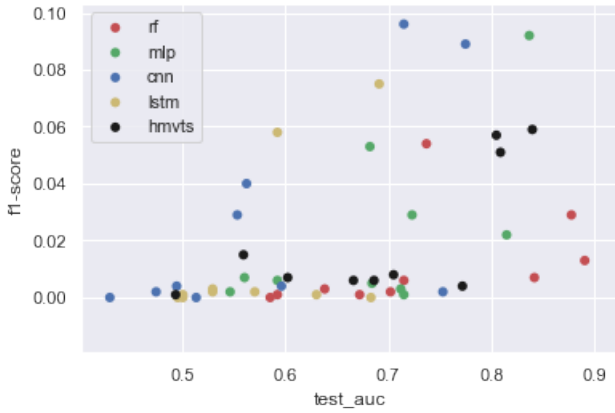



Figure 5.4: AUC vs F1 score on test set

model (red) for three experiments achieves a really good AUC score, but that this does not translate to a good F1 score on the actual labels. It is likely that this is because the RF model is able to "detect" many of the labels belonging to the same fault(s), while missing others completely. The opposite is observed for at least two of the experiments for the CNN-model, where the AUC score is not very high, but the F1 scores are much better.

### 5.5.5 Practical implications

As we can see, there is no model that performs consistently well across all experiments. The recommended consideration to be made when evaluating the usefulness of this system in practice, is to find a specific fault category for which an early warning system can detect a enough errors without too many false alarms with a sufficiently large pre-warning time (PF window).

Depending on the business case, the percentage of errors detected need not necessarily be high to be useful. Also, one can tolerate a high false alarm rate if there exists an efficient and inexpensive procedure to investigate alarms.

The challenge is thus to make a connection from the results presented in this thesis to what fault categories, and for which PF windows, such a system might bring the most value. Costs of operating such a fault prediction system should also be taken into consideration for this evaluation.

# Chapter 6

# Evaluation and Conclusion

This chapter will begin by providing a summary of the thesis in section 6.1, before the research goals are evaluated in section 6.2. Section 6.3 discusses the implications of the results, as well as some of the limitations. Section 6.4 summarizes the research contributions of this thesis, before section 6.5 concludes the chapter by describing potential directions that could be subject to future work in this area of research.

## 6.1 Thesis Summary

This thesis has investigated the application of a fault prediction system for offshore wind turbines based on SCADA data. Some background theory needed to follow the rest of the thesis was presented in chapter 2. Related work was surveyed in chapter 3, both for the specific domain of fault prediction in wind turbines based on SCADA data, as well as the more general domain of utilizing deep learning for multivariate time series classification. Based upon the findings of chapter 3, the architecture of the system and its components were presented in chapter 4. After the system was presented, a series of experiments were performed to evaluate the performance of different deep learning models as the classifier component of the system, as well as a model representing the state-of-the-art based on the literature review. Finally, this chapter summarizes and discusses the findings of the thesis.

## 6.2 Goal Evaluation

The primary goal of this project was to build a deep learning-based system for fault prediction in offshore wind turbines based on SCADA data:

**Research goal** Build a deep learning-based fault prediction system for offshore wind turbines based on SCADA data.

Such a system has been implemented, providing a system and framework for effectively and fairly comparing different classifier components of the system. The system was built making use of best practices from both related work in the domain, machine learning, time series classification and deep learning in general.

In addition to the system itself, the derived research questions that we wanted to address was:

**Research question 1** *How do deep learning methods perform in comparison to the state-of-the-art for predicting faults in wind turbines?*

**Research question 2** *Which of the investigated deep learning architectures are best suited as the classifier component in such a system?*

As shown in figure 5.3 and table 5.6, we can see that the random forest model that was implemented to represent the state-of-the-art approach, is the worst performing model on average in our experiments, and that the deep learning methods definitely show potential to outperform random forest models as part of such a system.

When it comes to comparison of the different deep learning architectures, there are large variations in performance across fault categories and PF windows. Hence, it is not possible to draw a definite conclusion as to which of the deep learning architectures are best suited in general. The question of which model performs best comes down to which type of fault that we want to model and which PF window is suitable (and valuable from a business perspective) for this fault category.

## 6.3 Discussion

It was observed that both the LSTM and HMVTS model failed to fit the data on some of the experiments. This could likely have been mitigated by employing gradient clipping to avoid extremely large gradients. Implementing this could help improve the performance of these models. This explanation did not present itself until late in the thesis work, and was thus not implemented, but must be considered as part of future work.

In the previous section, we observed that the F1 score on the actual faults was significantly worse with the 6h PF window than the 1h PF window. This raises an important issue that was not considered very carefully when choosing the strategy for this work; The PF windows should ideally be based on the characteristics of each specific fault type. If there is a mismatch between the signal and the labels in the data, we introduce a lot of noise to our targets, which will affect results negatively. The challenge, of course, is that the characteristics of the faults are not known explicitly in advance. It is very likely that the potential noise introduced by adopting the labelling procedure in this thesis might drastically reduce performance of the system if the PF window chosen does not correspond to the actual signal of an impending fault. In order to investigate this, further experiments on several PF windows could be performed.

An important thing to keep in mind is that the results of the deep learning models cannot be attributed to the architecture type only, but that the capacity (number of parameters) and regularization for each of the models also has a large effect on the performance. If we wanted to be able to draw more general conclusions about performance of architecture types, these factors would need to be controlled and accounted for to provide a fair comparison.

Overall, the results presented can be summarized as promising, and there are some signal that we are able to model. The results differ significantly between fault categories.

When considering whether the results of this thesis justify a deployment of the system, several factors need to be considered. Most importantly, an estimate of the cost of an undetected fault, as well as the cost of investigating a false alarm, should be taken into account. These estimates could be used in an analysis together with the evaluation metrics in this thesis, to estimate whether deployment of the system is profitable.

## 6.4 Research contributions

The main research contribution of this thesis is the design and implementation of a modular fault prediction system for offshore wind turbines based on SCADA data. The methodology used for the different components can be used, adapted and built upon for application to similar problems. Another key research contribution is the results presented that show that deep learning models that take an input sequence are well suited for time series classification problems, and they demonstrate improved results compared to a random forest model which only considers a single data point. The final research contribution is the insight that the domain of fault prediction is rather complex, and that different faults have their own characteristics, which makes one algorithm better suited to model them than others. As a consequence, no algorithm suits all fault characteristics.

## 6.5 Future Work

The work in this thesis might serve as a starting point for future work in multiple directions. Pursuing the same direction as this work, some of the possible options that could be explored to address the limitations discussed in section 6.3 are to investigate more fine-grained PF windows. For example 2h and 3h. These are likely to give better results than 6h, and might make a stronger business case due to a prolonged warning period compared to 1h. One option is also to try to facilitate better results by adding manually engineered features to the data. This is likely to improve the performance, although adding to the complexity of the system.

A possible option to reduce generalization error of this system could be to investigate the use of ensemble methods. This would involve using the predictions from several of the algorithms to make a final prediction that possibly could generalize better than the individual algorithms. Again, this would add quite a bit of complexity to the system.

As always in the field of deep learning, there are also endless opportunities for exploring architectures and hyperparameter search, and an exciting option would be to try to use evolutionary methods to discover the best deep learning architectures for the problems in this thesis. It should be noted that this direction would likely introduce a considerable computational complexity.

An option that is a bit on the side of the work done in this thesis is to investigate the possible reframing of the problem from a classification to a regression problem. This would mean labelling the samples with "time to next fault", in order to try to predict remaining useful lifetime (RUL).

Another direction that could be investigated is to take a normal behavior modelling approach, in order to detect anomalies from normal operations. Work in this direction would have to emphasize whether such a system is able to detect faults without introducing too many false alarms for data that is just different from previous data.

# Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. (2018). The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Branco, P., Torgo, L., and Ribeiro, R. P. (2016). A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys (CSUR)*, 49(2):31.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

Chen, L., Xu, G., Zhang, Q., and Zhang, X. (2019). Learning deep representation of imbalanced scada data for fault detection of wind turbines. *Measurement*.

Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., and Batista, G. (2015). The ucr time series classification archive. `www.cs.ucr.edu/~eamonn/time_series_data/`.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Conti, J., Holtberg, P., Diefenderfer, J., LaRose, A., Turnure, J. T., and Westfall, L. (2016). International energy outlook 2016 with projections to 2040. Technical report, USDOE Energy Information Administration (EIA), Washington, DC (United States). Office of Energy Analysis.

Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.

Drucker, H., Burges, C. J., Kaufman, L., Smola, A. J., and Vapnik, V. (1997). Support vector regression machines. In *Advances in neural information processing systems*, pages 155–161.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *ArXiv e-prints*.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.

Fawcett, T. (2016). Learning from imbalanced classes. `https://www.svds.com/learning-imbalanced-classes/`. Accessed: 2018-11-18.

Fulcher, B. D. and Jones, N. S. (2014). Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3026–3037.

Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

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

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Guo, C. and Berkhahn, F. (2016). Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

Howard, J. (2018). Rossmann kaggle competition, fast.ai course. `https://github.com/fastai/fastai/blob/master/courses/dl1/lesson3-rossman.ipynb`, Last accessed on 2019-03-27.

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141.

Hu, R. L., Leahy, K., Konstantakopoulos, I. C., Auslander, D. M., Spanos, C. J., and Agogino, A. M. (2016). Using domain knowledge features for wind turbine diagnostics. In *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*, pages 300–307. IEEE.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., and Muller, P.-A. (2018). Deep learning for time series classification: a review. *ArXiv*.

Jiang, G., Xie, P., He, H., and Yan, J. (2018). Wind turbine fault detection using a denoising autoencoder with temporal information. *IEEE/ASME Transactions on Mechatronics*, 23(1):89–100.

Jolliffe, I. (2011). Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer.

Kanter, J. M. and Veeramachaneni, K. (2015). Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Paris, France, October 19-21, 2015*, pages 1–10. IEEE.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kusiak, A. and Li, W. (2011). The prediction and diagnosis of wind turbine faults. *Renewable Energy*, 36(1):16–23.

Leahy, K., Gallagher, C., O'Donovan, P., Bruton, K., and O'Sullivan, D. (2018). A robust prescriptive framework and performance metric for diagnosing and predicting wind turbine faults based on scada and alarms data with case study. *Energies*, 11(7):1738.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

McKinney, W. (2010). pandas: a foundational python library for data analysis and statistics.

Mikolov, T., Deoras, A., Kombrink, S., Burget, L., and Černockỳ, J. (2011). Empirical evaluation and combination of advanced language modeling techniques. In *Twelfth Annual Conference of the International Speech Communication Association*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Orozco, R., Sheng, S., and Phillips, C. (2018). Diagnostic models for wind turbine gearbox components using scada time series data. In *2018 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 1–9. IEEE.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pfaffel, S., Faulstich, S., and Rohrig, K. (2017). Performance and reliability of wind turbines: A review. *Energies*, 10(11):1904.

Sakurada, M. and Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, page 4. ACM.

Schlechtingen, M. and Santos, I. F. (2011). Comparative analysis of neural network and regression based condition monitoring approaches for wind turbine fault detection. *Mechanical systems and signal processing*, 25(5):1849–1875.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Tavner, P. (2012). Offshore wind turbines: reliability. *Availability and Maintenance, The Institution of Engineering and Technology, London, UK*.

Unknown (2019). Mlflow.org. `https:/mlflow.org`, Last accessed on 2019-04-12.

Wang, L., Zhang, Z., Long, H., Xu, J., and Liu, R. (2017a). Wind turbine gearbox failure identification with deep neural networks. *IEEE Transactions on Industrial Informatics*, 13(3):1360–1368.

Wang, Z., Yan, W., and Oates, T. (2017b). Time series classification from scratch with deep neural networks: A strong baseline. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 1578–1585. IEEE.

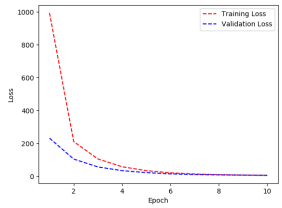
Yang, C., Qian, Z., Pei, Y., and Wei, L. (2018). A data-driven approach for condition monitoring of wind turbine pitch systems. *Energies*, 11(8):2142.

Zhang, Z. (2018). Automatic fault prediction of wind turbine main bearing based on scada data and artificial neural network. *Open Journal of Applied Sciences*, 8(06):211.

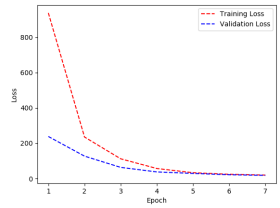
Zhang, Z.-Y. and Wang, K.-S. (2014). Wind turbine fault detection based on scada data analysis using ann. *Advances in Manufacturing*, 2(1):70–78.

This chapter contains the full results for all the experiments. BLABLA

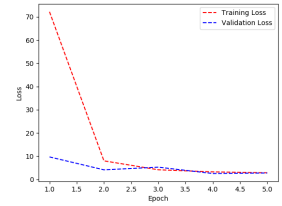
# A Training history



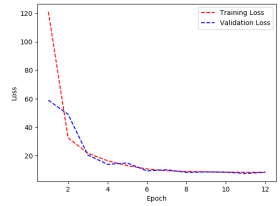

(a) M005, window=1h



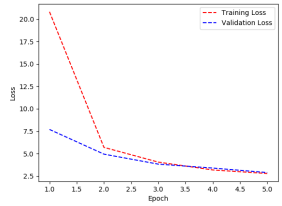

(b) M005, window=6h



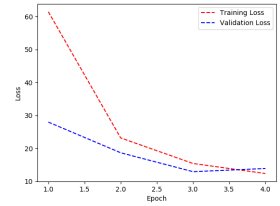

(c) M076, window=1h



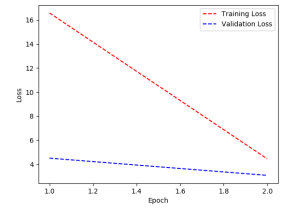

(d) M076, window=6h



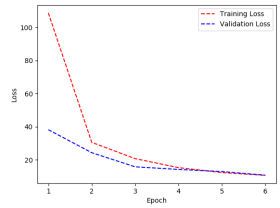

(e) M079, window=1h



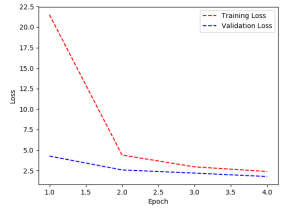

(f) M079, window=6h



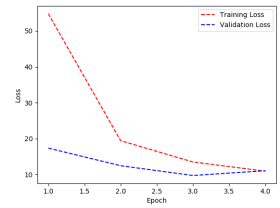

(g) M099, window=1h




(h) M099, window=6h

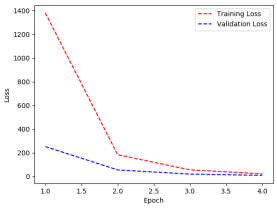



(i) M104, window=1h

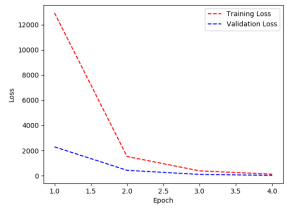



(j) M104, window=6h

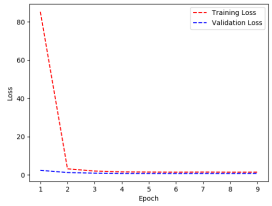
Figure 1: Training history for MLP models on test set



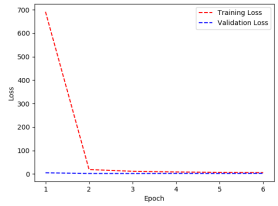

(a) M005, window=1h



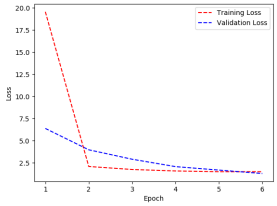

(b) M005, window=6h



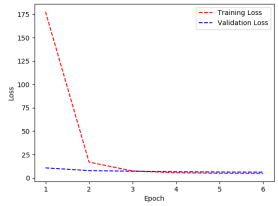

(c) M076, window=1h



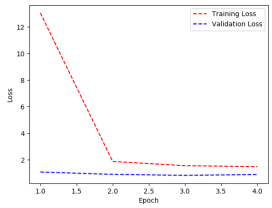

(d) M076, window=6h



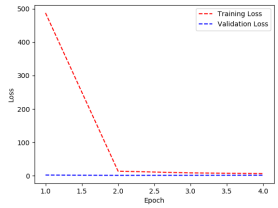

(e) M079, window=1h



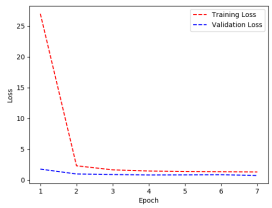

(f) M079, window=6h



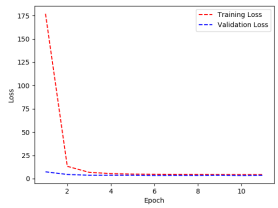

(g) M099, window=1h




(h) M099, window=6h

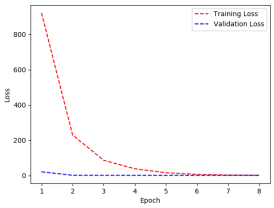



(i) M104, window=1h

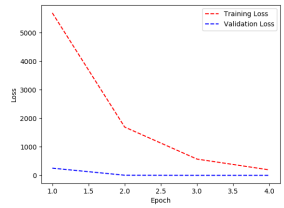



(j) M104, window=6h

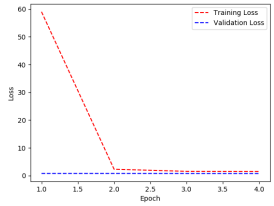
Figure 2: Training history for CNN models on test set



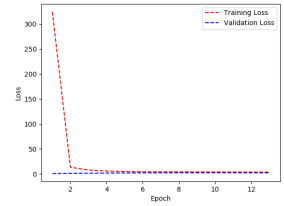

(a) M005, window=1h



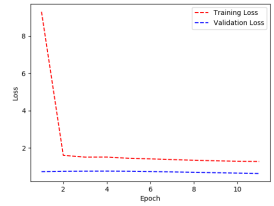

(b) M005, window=6h



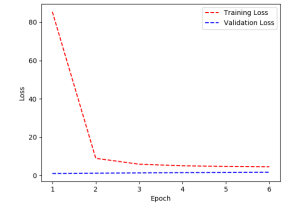

(c) M076, window=1h



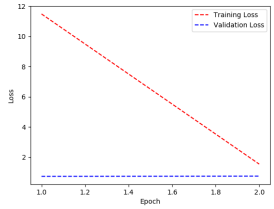

(d) M076, window=6h



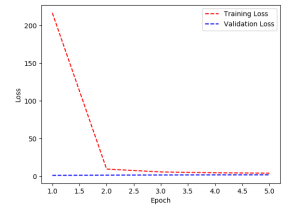

(e) M079, window=1h



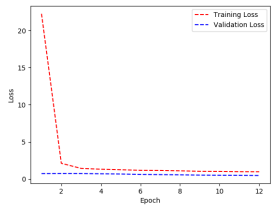

(f) M079, window=6h



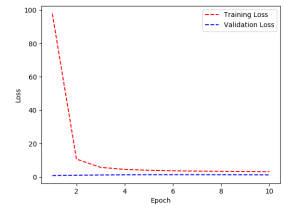

(g) M099, window=1h




(h) M099, window=6h

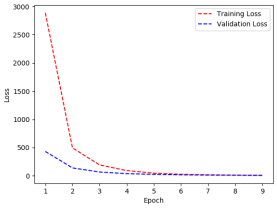



(i) M104, window=1h

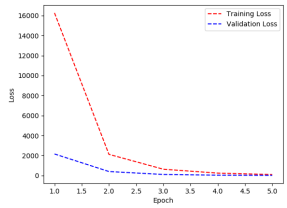



(j) M104, window=6h

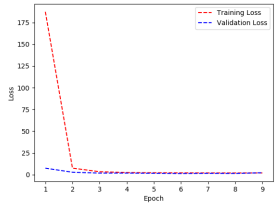
Figure 3: Training history for LSTM models on test set



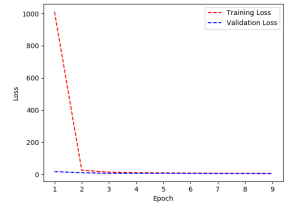

(a) M005, window=1h



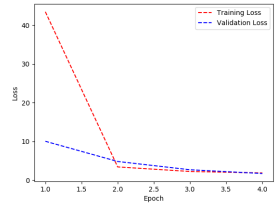

(b) M005, window=6h



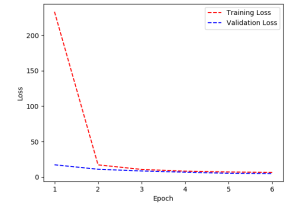

(c) M076, window=1h



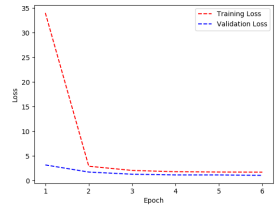

(d) M076, window=6h



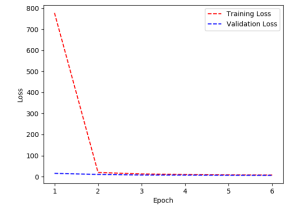

(e) M079, window=1h



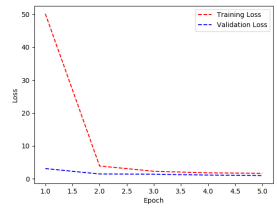

(f) M079, window=6h



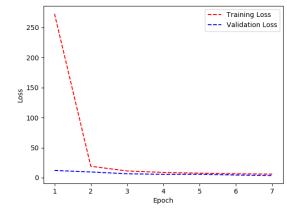

(g) M099, window=1h




(h) M099, window=6h



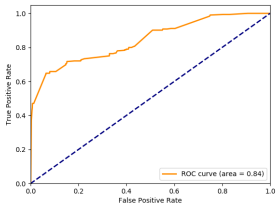

(i) M104, window=1h



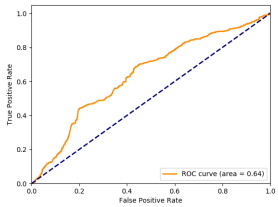

(j) M104, window=6h

Figure 4: Training history for HMVTS models on test set

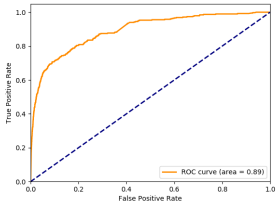
# B ROC curves on test set



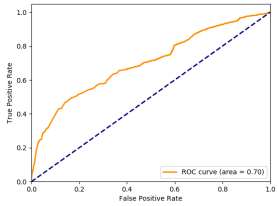

(a) M005, window=1h



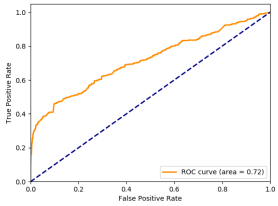

(b) M005, window=6h



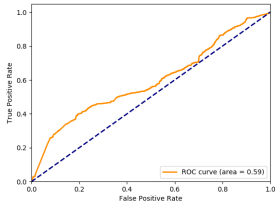

(c) M076, window=1h



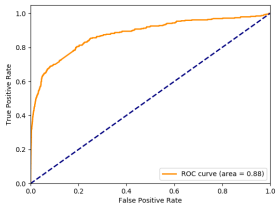

(d) M076, window=6h



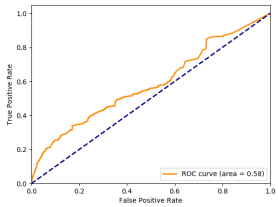

(e) M079, window=1h



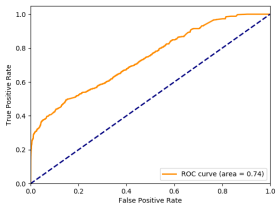

(f) M079, window=6h



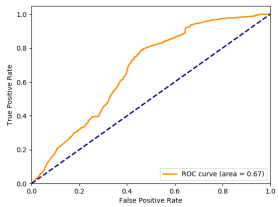

(g) M099, window=1h




(h) M099, window=6h

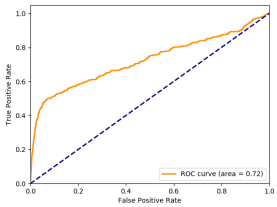



(i) M104, window=1h

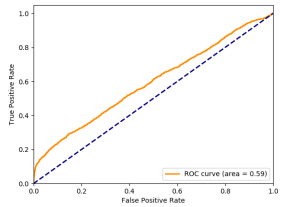



(j) M104, window=6h

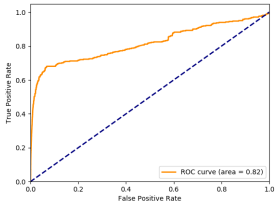
Figure 5: ROC curves for Random Forest models on test set



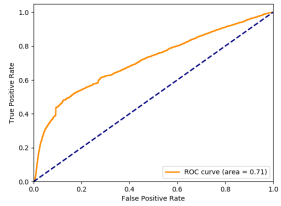

(a) M005, window=1h



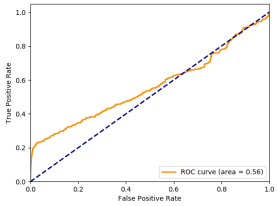

(b) M005, window=6h



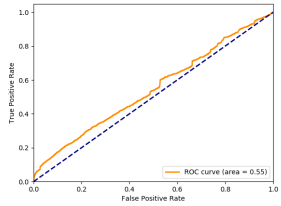

(c) M076, window=1h



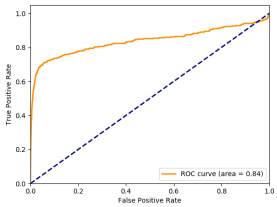

(d) M076, window=6h



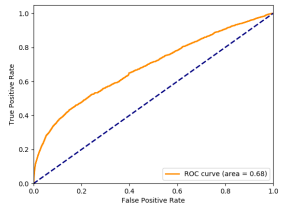

(e) M079, window=1h



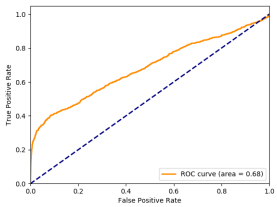

(f) M079, window=6h



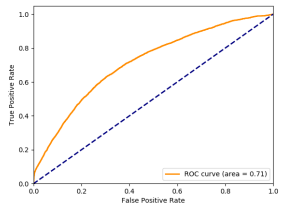

(g) M099, window=1h




(h) M099, window=6h

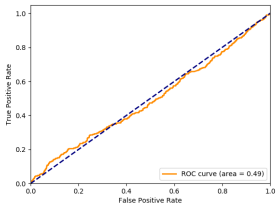



(i) M104, window=1h

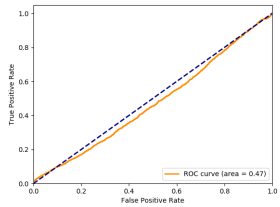



(j) M104, window=6h

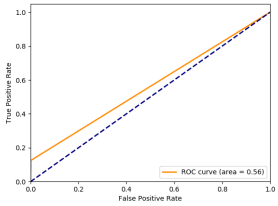
Figure 6: ROC curves for MLP models on test set



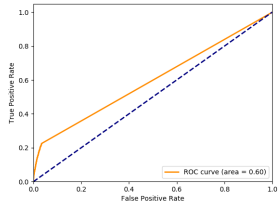

(a) M005, window=1h



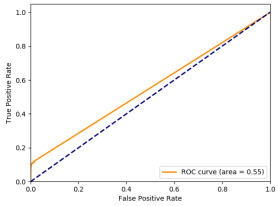

(b) M005, window=6h



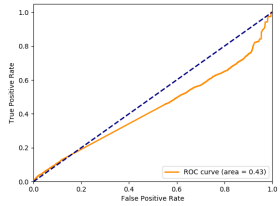

(c) M076, window=1h



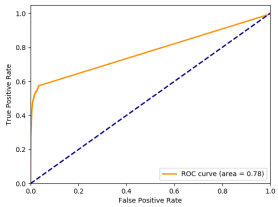

(d) M076, window=6h



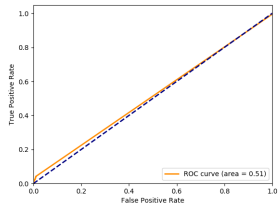

(e) M079, window=1h



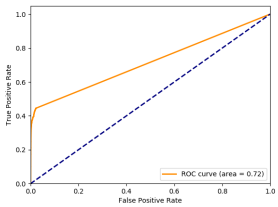

(f) M079, window=6h



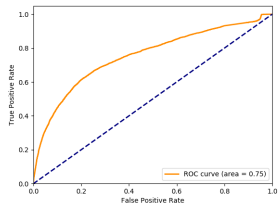

(g) M099, window=1h




(h) M099, window=6h

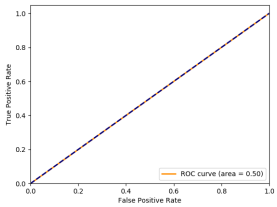



(i) M104, window=1h

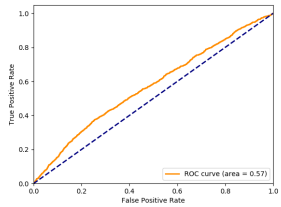



(j) M104, window=6h

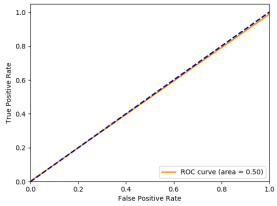
Figure 7: ROC curves for CNN models on test set



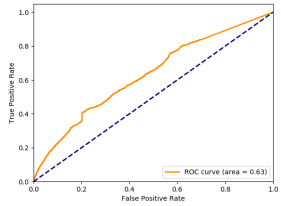

(a) M005, window=1h



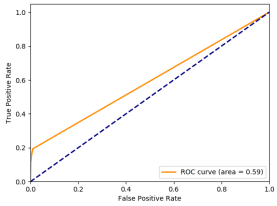

(b) M005, window=6h



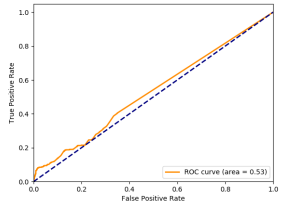

(c) M076, window=1h



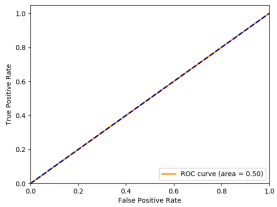

(d) M076, window=6h



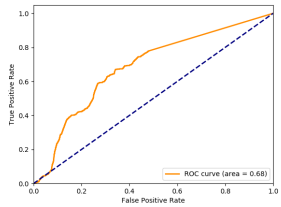

(e) M079, window=1h



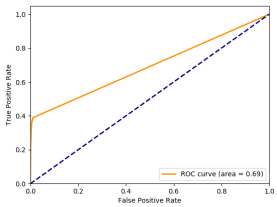

(f) M079, window=6h



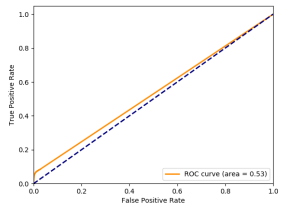

(g) M099, window=1h




(h) M099, window=6h

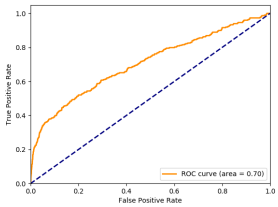



(i) M104, window=1h

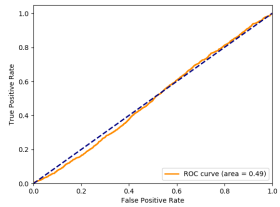



(j) M104, window=6h

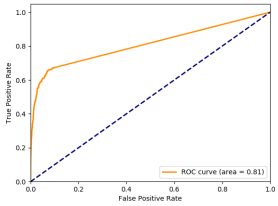
Figure 8: ROC curves for LSTM models on test set



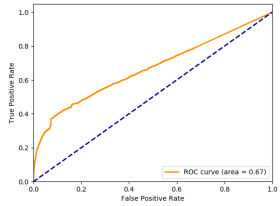

(a) M005, window=1h



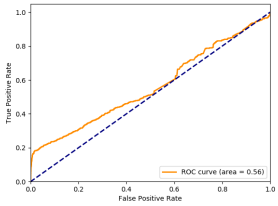

(b) M005, window=6h



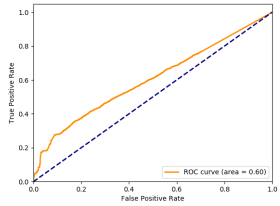

(c) M076, window=1h



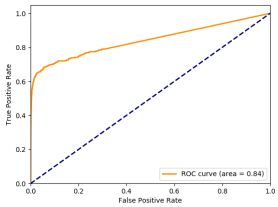

(d) M076, window=6h



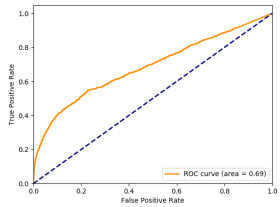

(e) M079, window=1h



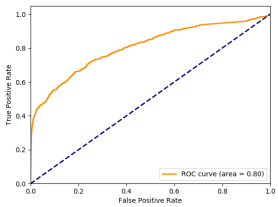

(f) M079, window=6h



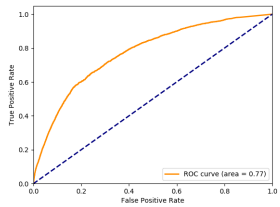

(g) M099, window=1h




(h) M099, window=6h




(i) M104, window=1h




(j) M104, window=6h

Figure 9: ROC curves for HMVTS models on test set