FACHHOCHSCHULE NORDWESTSCHWEIZ

MASTER THESIS PROPOSAL

# Deep Learning for Anomaly Detection

*Author:*
Kevin SANER

*Supervisor:*
Prof. Dr. Thomas HANNE

*A thesis proposal submitted in fulfillment of the requirements for the degree of Master of Science in Business Information Systems*

*in the*

School of Business

November 28, 2021

# Declaration of Authorship

## Declaration of Authenticity

I the undersigned declare that all material presented in this paper is my own work or fully and specifically acknowledged wherever adapted from other sources. I understand that if at any time it is shown that I have significantly misrepresented material presented here, any degree or credits awarded to me on the basis of that material may be revoked. I declare that all statements and information contained herein are true, correct and accurate to the best of my knowledge and belief. This paper or part of it have not been published to date. It has thus not been made available to other interested parties or examination boards.

Olten, 21st of December 2021

Name:

## Statement of Authenticity

I, Kevin Saner, hereby confirm that this master thesis is my own work or fully and specifically acknowledged wherever adapted from other sources. I understand that if at any time it is shown that I have significantly misrepresented material presented here, any degree or credits awarded to me on the basis of that material may be revoked. I declare that all statements and information contained herein are true, correct and accurate to the best of my knowledge and belief.

Signed:

Date:

“*Your brain does not manufacture thoughts. Your thoughts shape neural networks.*”

Deepak Chopra

FACHHOCHSCHULE NORDWESTSCHWEIZ

# *Abstract*

School of Business

Master of Science in Business Information Systems

**Deep Learning for Anomaly Detection**

by Kevin SANER

In the age of Big Data, data analysis becomes ever more important. To analise the data, many researchers nowadays focus on articial intelligence. Artificial intelligence does not rely on labour-intensive feature engineering like the traditional machine learning or statistical models. Therefore the use of AI, such as neural networks, can save a lot of development time. Two widely used architectures of neural networks are the Convolutional Neural Networks and the Recurrent Neural Networks. A Convolutional Neural Network is generally used when a task is related to image recognition, whereas Recurrent Neural Networks are used for the prediction of time series. Recently an approach was proposed to analyze time series data with Convolutional Neural Networks. The strengths and weaknesses of this approach, however, are currently unknown and are further investigated in this paper. To examine the usefulness, the practically relevant use case of anomaly detection was chosen. Within the scope of this work, different approaches on anomaly detection, that employ convolutional or recurrent neural networks are investigated. Further, the influence of transfer learning on both architectures is examined. Since the architectures should be compared regarding their performance, ways to evaluate the performances are assessed. Finally, it is described, how the chosen framework of methodoloy is applied and how the further research is planned out.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **DNN** | **D**eep **N**eural **N**etwork |
| **CNN** | **C**onvolutional **N**eural **N**etwork |
| **RNN** | **R**ecurrent **N**eural **N**etwork |
| **GRU** | **G**ated **R**ecurrent **U**nit Network |
| **LSTM** | **L**ong **S**hort **T**erm **M**emory |
| **ROC** | **R**eceiver **O**perating **C**haracteristics |
| **AUC** | **A**rea **U**nder the **C**urve |
| **MAE** | **M**ean **A**bsolute **E**rror |

# Chapter 1

# Introduction

With the rise of the Internet of Things (IoT) and ever more sensors, gadgets and smart devices like smartwatches for fall detection or blood pressure monitoring, or fridges for temperature protective control in use, the amount of available data steadily increases (Alansari et al., 2018). Simultaneously, the possibilities to use the data to draw conclusions increases. This data is generally used to draw conclusions such as failure of a system or a medical issue, such as a heart attack. These events typically occur very rarely (Hauskrecht et al., 2007). However, when the number of instances of each class is approximately equal, most machine learning algorithms function best. Problems occur when the number of instances of one class greatly exceeds the number of instances of the other. This issue is very popular in practice, and it can be observed in a variety of fields such as fraud detection, medical diagnosis, oil spillage detection, facial recognition, and so on (Thabtah et al., 2020). The task of identifying the rare item, event or observation is often referred to as anomaly detection. Typically, the anomalous item translates to problems such as bank fraud or medical problems. Often, the anomaly does not adhere to the common statistical definition of an outlier. Therefore, many outlier detection methods (in particular unsupervised methods) fail on such data (Hodge and Austin, 2004)

A special discipline in anomaly detection is to find the anomaly in a time series. The anomaly detection problem for time series is usually formulated as finding outlier data points relative to some standard or usual signal. Time series anomaly detection plays a critical role in automated monitoring systems. It is an increasingly important topic today, because of its wider application in the context of the Internet of Things (IoT), especially in industrial environments. Popular techniques to find the anomalies are:

- Statistical Methods
- Support Vector Machines
- Clustering
- Density-based Techniques
- Neural Networks

Currently neural networks are regarded the cutting-edge research, although first approaches to use artificial neural networks exist since 1958 (Rosenblatt, 1958). They are popular in research for only about 15 to 20 years. Neural networks are well-suited for assisting people in solving complex problems in real-world situations. They can learn and model nonlinear and complex relationships between inputs and outputs; make generalizations and inferences; uncover hidden relationships, patterns, and predictions; and model highly volatile data (such as financial time series data) and variances required to predict rare events (such as fraud detection). Since neural networks do not require time intensive feature engineering, but instead learn the features themselves a lot of time can be saved setting up a model compared to traditional machine learning approaches. At the moment, there is a lot of research focussing on neural networks, which is why scientist generally expect further improvements on this kind of technology (e.g. (Braei and Wagner, 2020), (Thabtah et al., 2020), (Verner, 2019), (Wen and Keyes, 2019)). This promising outlook is why this research paper also purely focusses on Neural Networks for anomaly detection. Convolutional Neural Networks and Recurrent Neural Networks are two neural network topologies that are commonly employed. Convolutional Neural Networks are commonly utilized for image identification tasks, whereas Recurrent Neural Networks are used for time series prediction. Recently, a method for detecting anomalies in time series data with Convolutional Neural Networks was proposed. Therefore especially Recurrent Neural Networks and Convolutional Neural Networks are investigated and compared.

## 1.1 Definitions

Following, important terms in the context of anomaly detection using neural networks, for a better understanding of the problem statement, are elaborated and defined.

### 1.1.1 Univariate and Multivariate Data

Time series data investigation poses a special discipline. Generally anomalies in time series are harder to detect for traditional statistical models, since the possibility of long term dependencies exist. Time series data comes in different forms. It is distinguished between univariate, bivariate and multivariate data. Univariate involves the analysis of a single variable while bivariate and multivariate analysis examines two or more variables. One significant difference between time-series and other datasets is that the observations are dependent not only on the components, but also on the time features. Thus, time-series analysis and the statistical methods employed are largely distinct from methods employed for random variables, which assume independence and constant variance of the random variables. Time-series are important to data analysts in a variety of fields such as the economy, healthcare and medical research, trading, engineering, and geophysics. These data are used for forecasting and detecting anomalies.

#### Univariate Data

There is only one variable in this type of data. Because the information only deals with one variable that changes, univariate data analysis is the simplest type of analysis. It is not concerned with causes or relationships, and the primary goal of the analysis is to describe the data and identify patterns.

#### Multivariate Data

Multivariate data is defined as data that contains three or more variables. It's similar to bivariate, but there are more dependent variables meaning there is not only one variable that influences the observed behaviour (independent variable) but several. The methods for analysing this data are determined by the objectives to be met. Regression analysis, path analysis, factor analysis, and multivariate analysis of variance are some of the techniques. Data collected from several sensors installed in a car is an example of a multivariate time-series.

### 1.1.2 Neural Networks

An Artificial Neural Network (ANN) with several layers between the input and output layers, is known as a Deep Neural Network (DNN). Neural networks come in a variety of shapes and sizes, but they all have the same basic components: neurons, weights and functions. These components work in a similar way as human brains and can be trained just like any other machine learning algorithm.

#### Neuron

Artificial neurons represent the smallest building blocks of neural networks. A neuron usually receives separately weighted inputs which it sums. The sum is then passed through the activation function to calculate the output of the neuron. When training a neural network, the input weights are adjusted by the optimizer function (see section 1.1.2) to improve accuracy of the given task e.g. classification.

#### Layer

Often neural networks use multiple layers. These layers are then stacked on top of each other, creating a so called deep architecture, which is why, when training neural networks it is often referred to as deep learning. In neural networks three different kind of layers are distinguished. There are input, output and hidden layers. A layer can be described as a collection of neurons. All layers between the input and output layer are called hidden layers. In the input layer data is fed into the neural network. The output of the hidden layer is calculated by taking the weighted sums of input and passing it through the activation function. Typically, a more complex problem requires more hidden layers to accurately calculate the output. In the output layer the final result e.g. a classification is produced. Figure 1.1 shows how a simple Neural Network with just one hidden layer could look like.

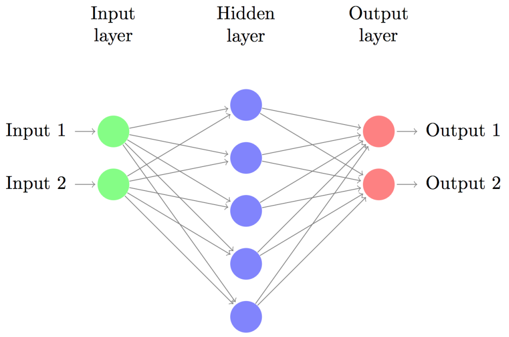


FIGURE 1.1: Input, Hidden and Output Layers (Britz, 2015)

#### Optimizers

Machine learning, and especially deep learning, has the idea of loss, that tells us how good the model is performing at the moment. The loss function is used the train the network so it performs better. Essentially, a way to decrease the loss needs to be found, because a lower loss indicates that the model is doing better. In mathematics, the process of minimizing an expression is called optimization. Optimizers are techniques or approaches that are used to adjust the characteristics of the neural network, such as weights, in order to minimise losses.

#### Supervised vs. Unsupervised Learning

Supervised learning refers to a task, where a machine learning algorithm learns on data that is labelled. A time series that contains an anomaly that is labelled as such would be an example of such a data set. In contrast, when a machine learning algorithm is applied to an unlabelled data set, it is called unsupervised learning. The goal of unsupervised learning is to find hidden patterns (clusters) in the data set.

## 1.2 Background

Following, it is explained how different kinds of neural networks work and what they are used for.

### 1.2.1 Neural Networks for Anomaly Detection

Out of the three most popular neural network architectures, convolutional neural networks (CNN), recurrent neural networks (RNN) and deep neural networks (DNN), RNNs are typically used for anomaly detection in time series [e.g (Malhotra et al., 2015), (Verner, 2019)]. RNNs have built-in memory and are therefore able to anticipate the next value in a time series based on current and past data. Classic or vanilla RNNs can theoretically keep track of arbitrary long-term dependencies in input sequences. There, however, is a computational issue: when using back-propagation to train a vanilla RNN, the back-propagated gradients can "vanish" or "explode" due to the computations involved in the process, which use finite-precision numbers. Because Long Short Term Memory (LSTM) units allow gradients to flow unchanged, RNNs using LSTM unit or Gated Recurrent units (GRU) partially solve the vanishing gradient problem and therefore drastically improve accuracy (Hochreiter and Schmidhuber, 1997).

Specially to mention in this context are LSTM (Long-Short Term Memory) and GRU (Gated Recurrent Units). Both achieved outstanding performance when used for tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or IDSs (intrusion detection systems) (Chung et al., 2014)

#### LSTM

LSTM was first proposed in 1997 by Schmidhuber and Hochreiter (Hochreiter and Schmidhuber, 1997). The initial version to the LSTM unit consisted of a cells, input and output gates. In 1999, the LSTM architecture was improved by introducing a forget gate and therefore allowing the LSTM to reset its own state (Gers, Schmidhuber, and Cummins, 2000). LSTM is used in a supervised training approach, that means it tries to predict a predefined state taking the past and the current state. If the predicted state differs from the expected state, the weights of the different gates are adjusted using an optimizer algorithm such as gradient descent. Figure 1.2 shows how the gates and the cell are arranged. The cell represents the memory of the LSTM. In simple words, the LSTM works as follows to predict a new value:

1. Forget Gate: Obsolete information is removed from the cell state.

2. Input Gate: New information is added to the cell state

3. Output Gate: The new information and the cell state are added to make the new prediction.

4. The new cell state is propagated to the next LSTM unit

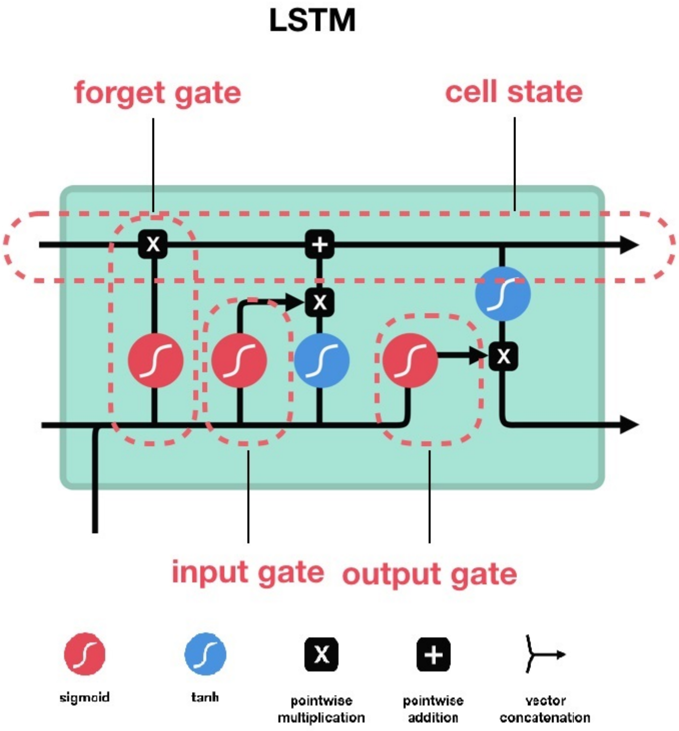
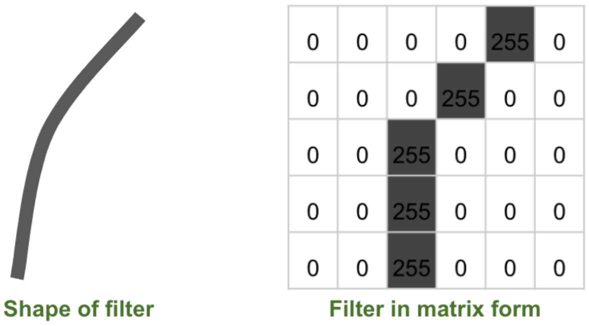


FIGURE 1.2: Gates and Cell of LSTM (Phi, 2018)

**CNN**

In contrast to RNNs Convolutional Neural Networks are generally used for image classification. CNNs work as feature extractors and are able to recognize patterns. CNNs use layers that are not fully connected, to reduce complexity (compare to 1.2.1). In a CNN, a set number of neurons forms a filter. These filters or kernels are the actual feature extractors. A filter may represent a line or pattern (see Figure 1.3) (LeCun et al., 1998).



| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 255 | 0 |
| 0 | 0 | 0 | 255 | 0 | 0 |
| 0 | 0 | 255 | 0 | 0 | 0 |
| 0 | 0 | 255 | 0 | 0 | 0 |
| 0 | 0 | 255 | 0 | 0 | 0 |

FIGURE 1.3: Example of a Filter used in CNN (Stureborg, 2019)

To detect whether, a feature is occurrent in a picture, the filter is gradually moved over the picture in so called strides. In every step (stride) the dot-product between the filter and the part of the picture is calculated. The results of the operations are stored in activation maps. The greater the dot-product the more alike are the filter and the section of the image. Training the network hereby refers to determining the shapes of these filters. Other typical features of a CNN are the pooling layers. The pooling layers reduce the amount of computation necessary. The most commonly used pooling technique is max-pooling and works as shown in Figure 1.4.

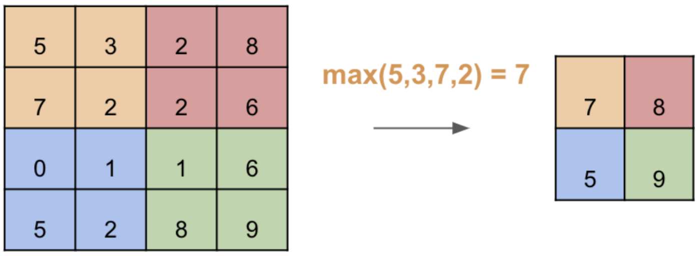


FIGURE 1.4: Example of max-pooling (Stureborg, 2019)

The idea of max-pooling is to only keep the maximum value of an activation map,. In the orange region 7 represents the maximum value, so it is kept while the other values are discarded (Stureborg, 2019). Using this technique the sharpest features are extracted.

In 2019, Wen and Keys proposed to use CNN also for anomaly detection in time series since it shares many common aspects with image segmentation. A univariate time series is therefore viewed as a one-dimensional image (Wen and Keyes, 2019).

## 1.3 Problem Statement

Defining a ground truth is a difficult aspects of time series anomaly detection. Determining when anomalous behaviour begins and ends in time series is a difficult task, as even human experts are likely to disagree in their assessments. Furthermore, there is the question of what constitutes a useful detection when detecting anomalies in time series. In the past, RNN have successfully been used for anomaly detection (e.g. [(Malhotra et al., 2015); (Fan et al., 2016)]. Therefore, designs for various use cases are well researched. RNN are well suited for the task, however, they take a long time to train due to the complexity of how a single unit is designed (see 1.2.1). In comparison CNN are not as complex and therefore, generally take less time to train. However, CNNs are generally used for image recognition and were only very recently used for anomaly detection in time series. It is therefore mostly unknown what designs are applicable for successful anomaly detection in time series data. While RNN are able to deal with multivariate data by design, a classical CNN requires design changes to be able to deal with multivariate data. Wen and Keys (2019) proposed to use a special kind of U-net, an improved version of a Fully Convolutional Neural Network (Ronneberger, Fischer, and Brox, 2015). Further, a CNN is not capable to analyse streaming data so it relies on segmentation of the data. These data segments are called snapshots. In order to not miss any data points, the frequency of taking these snapshots should be at least as high as the length of snapshot so that every time point is evaluated by the model at least once. However, for better performance it might prove beneficial to use a higher frequency which means every point is evaluated various times by the model (Wen and Keyes, 2019). The proposed design change and the fact that every point is evaluated multiple times, increases complexity and evaluation time and therefore counteract the architectural advantage of CNN compared to a RNN. When designing a neural network many parameters have to be chosen, this applies to both mentioned types of Neural Networks. For example, when designing a CNN, the number of layers, the activation function(s) of a single neuron and the optimizer function have to be chosen. Additionally, when using CNN for time series data the length and frequency of the snapshots have to be determined. Similarly, when designing a RNN also the number of layers and the optimizer function have to be determined. Because the basic building blocks of both networks types are very different it is difficult to fairly compare the complexity of two architecture approaches. Another important parameter which applies to both network types is the number of epochs for which the networks are trained. Through the epochs the training time is determined. In order to compare the two types of neural networks, two networks of similar complexity have to be designed. With equal training time the performance of both can be compared and evaluated. A RNN is therefore only set up as benchmark while the main goal of this research project is to clarify whether CNNs are really useful and propose an advantage over RNNs when applied on time series data for anomaly detection.

## 1.4 Thesis Statement

Convolutional Neural Networks are superior to Recurrent Neural Networks when looking for anomalies in time series data regarding training time and complexity.

### 1.4.1 Research Questions

- How does a CNN for univariate and multivariate data need to be designed for successful anomaly detection in time series data?
- What advantages and disadvantages arise when using a CNN compared to a RNN for anomaly detection in univariate and multivariate time series?
- What parameter settings are crucial for a fair performance comparison between RNN and CNN?

### 1.4.2 Research Objectives

Following the research objectives of this paper are defined.

1. Determine what design changes a CNN requires to detect anomalies in time series data.
2. Determine how the CNN should be designed for the comparison with a RNN
3. State the advantages and disadvantages of the chosen CNN architecture.
4. Define parameters which allow a fair comparison of CNN and RNN

### 1.4.3 Limitations

Recently there have been approaches that combine CNN and RNN into a hybrid network for tasks such as handwriting recognition or video-based emotion recognition (Dutta et al., 2018) (Fan et al., 2016). However, this paper only compares pure CNN and RNN, and does investigate a hybrid approach.

### 1.4.4 Significance

Until now, time series data was almost only approached with RNNs. This paper should answer the question whether CNNs propose a valid alternative and even propose some advantages over RNNs. The paper will answer the fundamental question whether research should channel efforts to further investigate CNNs for anomaly detection in time series data or whether no benefits can be discovered and research is better to focus on other areas.

# Chapter 2

# Related Literature

## 2.1 Anomaly or Outlier?

Generally, there is no agreement on how to distinguish between anomalies and outliers. The following often used citation claims equality of the term outliers and anomalies.

"*Outliers are also referred to as abnormalities, discordants, deviants, or anomalies in the data mining and statistics literature.*" - Aggarwal (2013)

By others, outliers are regarded as corruption in data, while anomalies are abnormal points with a particular pattern. In the context of this paper, only the term anomaly is used to refer to such irregular behaviour. It is hereby important to provide a clear definition for the concept of anomaly. This is critical since different meanings of abnormalities necessitate different detection methods. As a result, it is important to identify the key characteristics of anomalies and to use the description to highlight the boundaries. Following, two common definitions of anomalies:

"*Anomalies are patterns in data that do not conform to a well defined notion of normal behaviour.*" - Chandola et al. (2009)

Ord (1996), defines anomalies as follows:

"*An observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data.*"

Anomalies have two major features, according to both of these definitions:

- The anomalies' distribution deviates significantly from the data's overall distribution.
- Standard data points make up the vast majority of the dataset. The anomalies make up a very small portion of the overall dataset.

The development of anomaly detection methods is dependent on these two factors. The second property, in particular, prevents the employment of common classification methods that depend on balanced datasets.

### 2.1.1 Types of Anomalies

Anomalies come in a variety of shapes and sizes. Anomalies can be divided into three categories:

1. **Point Anomalies** - A point anomaly occurs when a single point deviates dramatically from the rest of the data. A point anomaly in a time series is, for example, a temperature peak in an otherwise, over time, steady temperature.

2. **Collective Anomalies** - Individual points may or may not be anomalous, but a series of points may be. A time series example could look as follows: A bank customer withdraws $500 from her account per weekday. Although withdrawing $500 every now and then is common for the consumer, a series of withdrawals is unusual.

3. **Contextual Anomalies** - Some points can appear natural in one context but be identified as anomalous in another: In Germany, a daily temperature of 35 degrees Celsius is considered natural in the summer, but the same temperature in the winter is considered unusual.

Knowing ahead of time what kind of anomaly the data might contain helps the data analyst choose the best detection process. Some methods for detecting point anomalies fail to detect collective or contextual anomalies entirely (Braei and Wagner, 2020).

## 2.2 Anomaly Detection on Univariate Time Series

First, anomaly detection on univariate time series using deep learning approaches is investigated to gain insight into the used architecture and the overall training and detection process.

### 2.2.1 LSTM

Because of their ability to retain long term memory, Long Short Term Memory (LSTM) networks have been shown to be especially useful for learning sequences containing longer term patterns. Malhotra et al. (2015) model normal behaviour with a predictor and then use the prediction errors to classify abnormal behaviour. This is especially useful in real-world anomaly detection scenarios where instances of normal behaviour are plentiful but instances of anomalous behaviour are rare. For prediction, multiple time steps into the future are forecasted to ensure that the networks capture the temporal structure of the chain. As a result, each point in the series has multiple corresponding expected values made at various points in the past, resulting in multiple error values. The likelihood of normal behaviour on the test data is calculated using the probability distribution of the errors produced when predicting on normal data. For this approach Malhotra et al. use a deep LSTM neural network. The proposed network architecture with two hidden layers is successfully applied on different univariate time series such Electrocardiograms (ECG), a valve time series and a power demand dataset. The proposed approach on univariate data could easily be adopted with multivariate data, where instead of a univariate in- and output, a multivariate data set is fed into the neural network to predict a multivariate output.

Especially to mention is the training approach used in both papers. To train the LSTM based Recurrent Network the data was divided into four sets. a non-anomalous training set, a non-anomalous validation set, a mixture of anomalous and non-anomalous validation set and test set, also consisting of a anomalous and non-anomalous sequences. This approach was chosen to deal with the rare class problem, which is typical in anomaly detection. Since non-anomalous data is plentiful, the model is trained in an unsupervised fashion, predicting the future but not able to classify if anomalous behaviour was present. The goal of this approach was to establish a baseline of non-anomalous behaviour with the training set. The learned model was then validated and tested with anomalous and non-anomalous data to evaluate its performance.

### 2.2.2 CNN

The use of CNNs for time-series analysis has received interest in recent years. Munir et al. (2019) forecast time-series and identify anomalies based on the prediction error using a CNN architecture called deep-learning based anomaly detection method (DeepAnT). DeepAnT uses an architecture that is divided into two modules. The first module is called "Time Series Predictor". The "Time Series Predictor" consists of a CNN. As the name of the module indicates, the CNN is responsible for predicting the future time stamps of a given time series, whereas the "Anomaly Detector" module is responsible for tagging given data point as anomalous. Figure 2.1 represents the architecture of the CNN-based predictor module. It consists of two convolutional layers, each followed by a max-pooling layer. As the last layer, however, a fully connected layer, where all neurons are connected to all neurons of the previous layer, is used. The last is responsible for predicting the next time step. The number of output nodes, hereby, corresponds to the number of predicted time steps. One output node means only the next time step into the future is predicted, whereas three output nodes would imply a sequence of three data points are predicted.

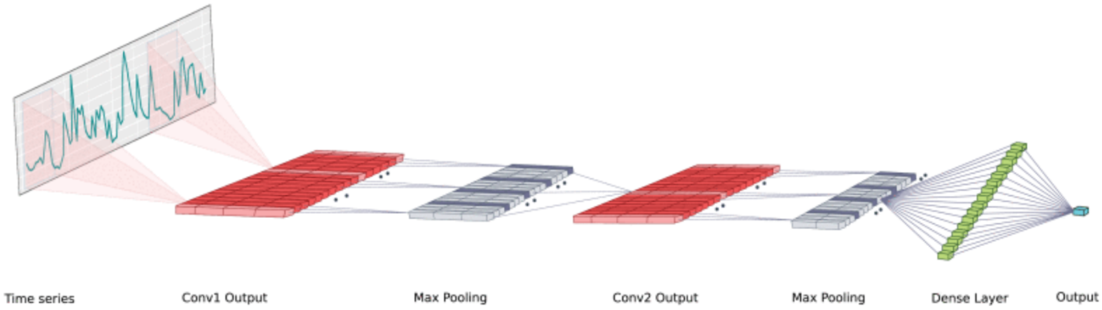


FIGURE 2.1: CNN Architecture for Time Series (Munir et al., 2019)

Once the next time steps are predicted, the values are then passed to the "Anomaly Detector". The detector module calculates the Euclidian distance between the predicted and the actual data point. This measure of discrepancy is used as anomaly score. A high anomaly score indicates a significant anomaly at a given time step. In order to classify the time samples as anomalous and non-anomalous, an anomaly threshold must be specified. The anomaly threshold classifies all data points that lie under the treshold, where the Euclidean distance of predicted and actual value is small, as normal. In contrast all points that exceed the threshold are classified as anomalous. Depending on how low the threshold is set, the sensitivity can be increased. Actually, a lot of anomaly detection algorithms require such a threshold.

### 2.2.3 Comparison

In an extensive study Braei and Wagner (2020) compared 20 different anomaly detection methods. The anomaly detection methods were divided into the three following categories.

- Statistical Methods

- Classical Machine Learning Methods
- Deep Learning Based Methods (Neural Networks)

#### Comparison of AUC

The above stated methods were applied on data containing point, collective and contextual anomalies. In order to compare the different approaches, AUC-Values [1] were used. The results showed that the statistical methods generally performed best on point and collective anomalies while deep learning approaches performed rather poorly. On a dataset that contained contextual anomalies, the situation reflected the exact opposite. Deep learning approaches clearly outperformed statistical methods. It was observed that deep learning approaches kept their ability to generalize while statistical methods overfitted on the data (Braei and Wagner, 2020).

#### Computation Time

The second parameter that was used to compare the different categories was training and inference time. Inference refers to the time used to classify the test data. Compared to statistical methods and classical machine learning, deep learning approaches once again performed rather poorly regarding training time. Looking at inference time deep learning approaches generally perform well, outperforming the other two categories. However, there are huge difference within the deep learning approaches. While CNNs have a very low inference time, and outperform most other algorithms, LSTMs have the highest inference time of all tested algorithms (Braei and Wagner, 2020).

## 2.3 Anomaly Detection on Multivariate Time Series

In this section, anomaly detection on multivariate time series is investigated. Once again, CNNs and LSTMs are examined on the state of the art application possibilities for anomaly detection or time series classification.

### 2.3.1 LSTM

First, relevant works that use LSTMs are investigated. LSTMs will be used to establish a benchmark.

#### Anomaly Detection using LSTM

In his dissertation, Alexander Verner (2019), compared different versions of LSTM Neural Networks against traditional machine learning algorithms such as Support Vector Machines (SVM) and Random Forests. The various algorithms were applied to data of sensors that measure blood glucose levels. To measure the blood glucose levels multiple sensors are used for an accurate result, thus a multivariate time series is the outcome. The data set used in Verner's work did not contain anomalies by default, but the anomalies were embedded into the data set artificially. Since the data was also labelled after the type of anomaly it contained, the learning approach can be classified as supervised. Thus, the goal of the study was to correctly classify which anomaly was present. The first approach used a deep or stacked LSTM, with 10 units in the first layer and 35 in the second. The

[1] An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. The AUC measures the entire two-dimensional area underneath the entire ROC curve. See section 2.4.1

resulting accuracy of only 0.4 was, however, rather disappointing. This poor performance was explained with an occurring information loss within the deep network. Next, a single layer architecture, with 100 units was proposed. With just a 10% increase the resulting accuracy was still unacceptable. In a third attempt, the neural network was enriched with an embedding layer. Using this technique, the neural network was able to detect the relationship between the frequencies of measurements that were closely positioned in the time-series. It achieved an accuracy of 98%. The final architecture then used bidirectional LSTMs (BLSTMs). According to Graves (2005), BLSTM outperform LSTM on supervised labelling tasks. A BLSTM can extract even more information from raw data by taking into account the relationship of each measurement to both previous (past) and subsequent (future) measurements in the input time-series. This enhancement resulted in a 99% accuracy. This impressive accuracy, however, comes at a cost. LSTMs are very time consuming and resource intensive to train. Verner used an Amazon Web Service computing instance with 24 CPUs, 4 GPUs and 128 GB of RAM but the training of an LSTM still took about 10 hours.

### 2.3.2 CNN

In this section CNNs for anomaly detection but also for classification of time series are investigated.

#### Classification of Time Series Data using CNN

In their research project Zheng et al. (2014) tried to beat the state of the art classification algorithm for time series, which is the k-Nearest Neighbor algorithm (k-NN). k-NN has been empirically shown to be extremely difficult to beat. The typical problem of the k-NN algorithm, however, is its computation time. Zheng et al. proposed to use their own developed architecture, which is called Multi-Channel Deep CNN (MC-DCNN). Each channel, hereby represents a CNN with convolutional and pooling layers. Typically channels in CNN are used to extract features from the different spectra of pictures. A coloured picture for example consists of three channels, red, green and blue. Each channel now works as feature extractor on just one colour. This feature of CNN is now used in time series classification. Every channel learns features independently using a single dimension of the multivariate time series as input. Another difference to image classification is that multivariate time series classification uses multiple 1D subsequences rather than 2D image pixels as data. Because CNN only learn features, no classification can be done. In order to classify, a CNN architecture is combined with a Multilayer Perceptron (MLP) that uses fully connected layers. Figure 2.2 shows the just described architecture. On the left, the different channels are shown, where every channel takes on its own univariate time series. With the denoted feature maps and pooling layers the features of the time series are learned. In the MLP on the right, finally, the classification of the time series is done. It is important to note that this architecture does not predict the next time steps in the series but instead a given time series is directly classified. The classification is hereby for example used to classify the physical activity depending on the heartrate and is not used for anomaly detection. In order to use the proposed architecture for anomaly detection however, only the output layer has to be changed.

Zheng et al. state that the used architecture was superior to the k-NN algorithm regarding accuracy. Further, the experiments show that deeper architecture are able to learn more robust high-level features. Further, the MC-DCNN architecture performs much faster than the k-NN algorithm, especially when a large dataset is present. The above described architecture could be adopted unchanged in order to classify anomalies.

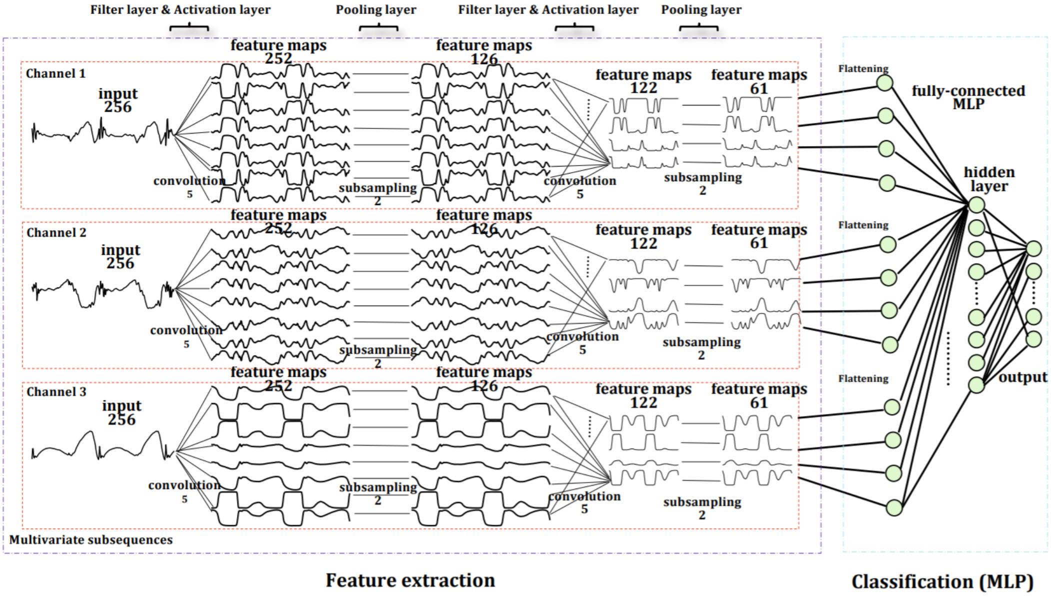


FIGURE 2.2: Architecture of the MC-DCNN (Zheng et al., 2014)

### U-Nets for Anomaly Detection

Wen and Keyes (2019) use a special type CNN architecture to detect anomalies. The architecture used is called U-Net. A U-Net consists of so called encoding and decoding layers. The encoding layers work like a standard CNN whereas the decoding layers are used for upsampling. Upsampling refers to restoring the previously condensed feature maps to its original size. In short, the encoding layers extract the most important features of the time series and decoding layers are using these features to assemble a new time series of the same dimensions as the original one. This encoder-decoder-architecture is often referred to as autoencoder architecture. The main weakness of autoencoders is that in the encoding part through downsampling information is permanently lost. To prevent this information loss, U-Nets introduce so called skip channels also called skip connections. A skip connection, as the name implies, is one that connects an earlier part of the network to a later part of the network and transfers data. The idea is simple: skip channels bring back missing knowledge from some earlier layers so that the network can be properly contextualized. This architecture was proven successful when applied to segmentation of neuronal structures in electron microscopic images in the original paper (Çiçek et al., 2016). In order to handle multivariate time series U-Nets also make use of multiple channels. Figure 2.3 shows the architecture of the above described U-Net. The left part of the picture shows the encoding layer with five convolutional layers. The encoding layer is followed by the decoding layer, in the right part of the picture. The decoding layer consists of 4 upsampling layers. This symmetric architecture is completed by the yellow lines, that represent the skip channels.

Finally, the model tries to classify what kind of anomaly the multivariate time series contained e.g a seasonal anomaly (contextual) or a point anomaly. Depending on whether the anomaly classes are mutually exclusive or not either a sigmoid or softmax activation is used as last layer activation function.

The proposed U-Net was tested on four scenarios: a univariate task with sufficient data, a multivariate task with sufficient data, a univariate task with insufficient data and transfer learning, and a multivariate task with insufficient data and transfer learning.

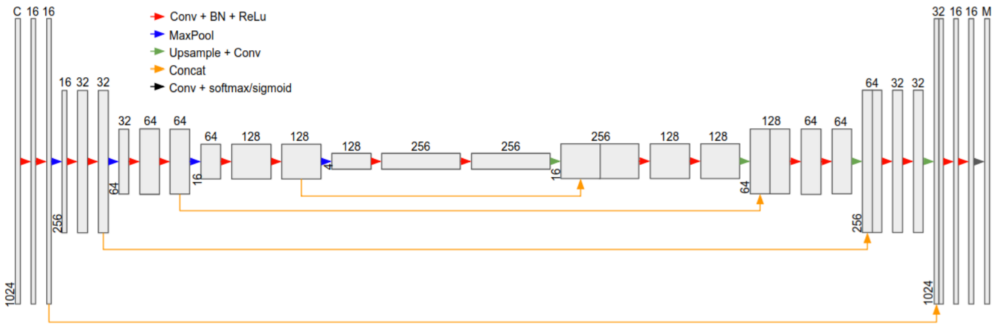


FIGURE 2.3: U-Net Architecture (Wen and Keyes, 2019)

For the univariate task the dodgers loop sensor data was used. It involves a 28-week time series of traffic on a ramp near Dodger Stadium with a 5-minute frequency. The goal is to spot unusual traffic patterns caused by sporting events. Out of 39 events, only three could not be detected. The missing detection were attributed to missing values in the data set. Missing values apparently also were the reason for some false positives.

For the multivariate task the gasoil heating loop data set was used. It contains 48 simulated control sequences for a gasoil plant heating loop that was hacked at one stage. There are 19 variables in all time series. A multivariate U-Net with 19 channels was trained. Out the 18, in the test present attacks, only one was missed. However, also 3 false alarms (false positives) were reported.

## 2.4 Hyperparameter Settings for fair comparison

In this section it is investigated how CNNs and RNNs can be compared fairly. Since the two's architectures fundamentally differ it is difficult to establish a baseline, where both networks are of similar complexity. In their work Braei and Wagner (2020) chose another approach. Instead of trying to build networks with similar complexity, they fine-tuned each evaluated neural network to its full potential and compared their scores as well as their training and inference time. To compare the scores, Braei and Wagner introduced an anomaly threshold (similar as in section 2.2.2). To find out the optimal threshold, it is varied to optimize sensitivity (true positive rate) and specificity (true negative rate). Varying the threshold and drawing the false positives against the true positives results in the so called Receiver Operating Characteristics Curve (ROC-Curve), which is used to find the optimal threshold for a certain model. How the ROC-Curve is used to compare different models is explained in the next section.

### 2.4.1 ROC and AUC

The receiver operating characteristic curve, ROC-Curve, and the associated metric area under the curve (AUC), together also called AUROC, which is the area under the ROC-Curve, are two metrics that are frequently used to compare models. This metric is highly useful, especially for detecting anomalies. The ROC-Curve depicts the relationship between the true positive rate and the false positive rate at various threshold values. The variables needed for the calculations are:

- TP: True Positives

- FN: False Negatives
- FP: False Positives
- TN: True Negatives

The true positive rate is defined as follows:

$$TPR = TP/(TP + FN)$$

The false positive rate is defines as:

$$FPR = FP/(FP + TN)$$

Typically, lowering the classification threshold, described in section 2.2.2, causes more items to be classified as positive, which increases both False Positives and True Positives. A typical ROC curve is depicted in Figure 2.4.

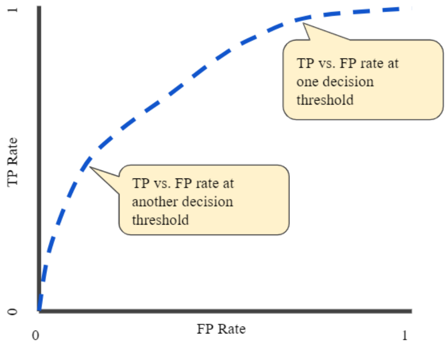


FIGURE 2.4: Example ROC-Curve (Google, 2021)

The area under the above shown curve, however, is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen. The AUC in anomaly detection expressed the likelihood that the measured algorithm assigns a random anomalous point in the time-series a higher anomaly score than a random normal point. As a result, AUC is a good way to compare different anomaly detection methods (Braei and Wagner, 2020).

### 2.4.2 F-Score

Another approach to compare models is to use the F-Score. The F-Score describes a model's accuracy on a dataset. It is commonly used to assess binary classification systems, which categorize examples as "positive" or "negative" or in the case of anomaly detection into "anomalous" and "non-anomalous". The F-score is a method of combining the model's precision and recall, per definition it is the harmonic mean of the precision and recall of the model. Precision and recall are calculated as follows from the true and false positive respectively negatives.

$$Precision = TP/(FP + TP)$$

$$Recall = TP/(TP + FN)$$

From precision and recall the F-Score, and in this specific case the F1-Score can be calculated as follows.

$$F1 = 2 * (Precision * Recall)/(Precision + Recall)$$

When applying F-Score, a perfect model has an score of 1. Therefore, different models can also easily compared.

### 2.4.3 Computation Time

Another approach to compare the performances is to compare the computation times. To compare the computation times Braei and Wagner (2020) looked at the average training and inference time of traditional machine learning models as well as statistical models and neural networks on univariate time series. Generally, neural networks are expected to invest most of their computation time in training and are able to outperform the traditional machine learning on inference time. However, the practical results showed that Recurrent Neural Networks (LSTM & GRU) not only needed a long time to train but also had a large inference time on certain data sets. With LSTM performing second best on the given data set, behind k-means clustering, it can be concluded that this comes with a trade-off. On the same data set a CNN was trained. The CNN achieved an AUC-Value of 0.818, compared to 0,84 of the LSTM, which is not much worse than LSTM. Looking at the training and inference time, the CNN is far superior. The LSTM takes about 1000 seconds for training and also inference. The CNN, however, trains for 50 seconds, with an inference time of under one second, making the CNN the better choice, when relying on small inference times.

## 2.5 Research Gap

Anomaly detection on time series data has been widely researched. There are comprehensive comparisons of statistical methods, traditional machine learning approaches and neural networks such as the works of Braei and Wagner (2020) or Verner (2019). Braei and Wager (2020), focus thereby only on univariate time series. Complementary, Verner (2019) only investigated multivariate time series. In their work, Braei and Wagner (2020) could show that CNN and RNN are able to achieve similar performances, while CNNs are superior in computation time. Verner (2019) faced a similar drawback in his research, where the training of the RNN took approximately 10 hours.

Since CNNs are only very recently investigated for anomaly detection in time series, Verner (2019) did not include them in his research. The approaches presented by Zheng et al. (2014) and Wen and Keyes (2019), however, show promising results, when using CNNs for time series classification respectively anomaly detection.

As reserach gap it has been identified that is currently unknown if CNNs are able to outperform RNNs when used in the field of anomaly detection. Braei and Wagner (2020) were already able to achieve good results when testing the approach on univariate time series. This work attempts to extend this knowledge to multivariate time series.

# Chapter 3

# Research Methodology

## 3.1 Introduction

This section describes which research approach was chosen and why. Further, it elaborates, how the research approach is implemented and what work is done in the corresponding sections.

## 3.2 Research Design

As research methodology experimental research was chosen. Experimental research typically focuses on systematically testing a hypothesis. It is often applied to research fields such as physics and chemistry but also psychology. In the this research project, the hypothesis to test is formulated as the thesis statement (see section 1.4.

Experimental research knows five process steps. These are:

- Awareness of the Problem
- Design of Experiments
- Experiments
- Evaluation
- Conclusion

In the following, it is outlined what will be done in the different process steps and how it is going to help answering the research question and test the thesis statement. Figure 3.1 illustrates the different steps.

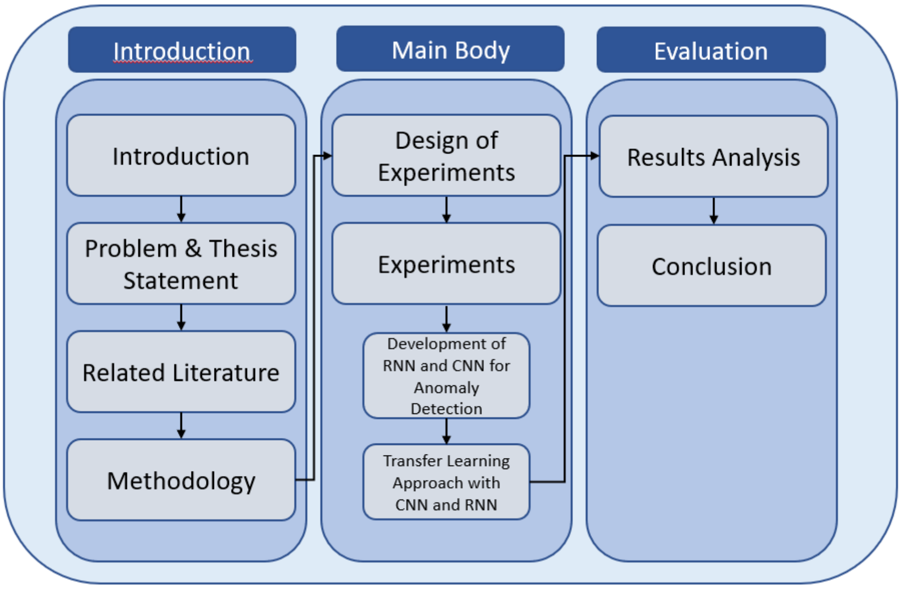


FIGURE 3.1: Thesis Map (**own**)

### 3.2.1 Awareness of the Problem - Literature Review

In this work, the literature review consists of two parts, background information presented in Section 1.2 and the part Related Literature presented as Chapter 2. The reason for this split is to give insight on how neural networks and especially the derived architectures such as CNN and RNN work. A basic understanding of these two architectures is required to comprehend the problem and thesis statement presented in Sections 1.3 and 1.4.

Further, the part Related Literature should deliver insights on how CNN and RNN are applied to detect anomalies. Where helpful, the publications investigated not only focus on detecting anomalies but also on related fields such as classification or prediction of time series. The investigated knowledge fields should serve as suggestions on how anomaly detection models can be set up. Next, the possibilities of transfer learning are examined. Since artificial intelligence experiences a boom in recent years, and neural networks are used more frequently, the possibilities offered by transfer learning will increase. Even more important, transfer learning is able to tackle one of the fundamental problems in anomaly detection. It enables to learn a performant model, even on a small data set.

At last, to be able to answer the research question it is investigated how the different architectures, RNN and CNN, can be compared and what measures are important.

### 3.2.2 Experimental Design

In this section, it is outlined how the experiments are designed. It is described how and why datasets are selected as well as which architecture principles are followed when designing the neural networks.

#### Data Selection

As described in Section 2.1, there are different kinds of anomalies, which are more or less difficult to detect. Section **??** Transfer Learning further shows that the relationship of the source domain and target domain data set is of significant importance for the quality of the results. These two facts indicate that the data sets used for the experiments need to be carefully selected. In the Section Design of Experiments suitable data sets are proposed. At least, three different data sets are selected to conduct experiments. One of these data sets represents the source domain for transfer learning and another one the target domain. To learn the generic features necessary for transfer learning the data set used as source domain needs to consist of a large amount of data.

Further, the selected data sets have a direct impact on how the experiments have to be designed. Here the decision has to made, whether an unsupervised or supervised approach is applied. A supervised approach hereby requires a data set where the anomalies are labelled. In contrast, an unsupervised approach can investigate any time series data for anomalies but it is difficult to validate the achieved results.

#### Setup of Experiments

In the Section Setup of Experiments the general setup of the experiments is elaborated. Chapter 2 showed that there are various approaches on how to detect anomalies. Anomalies can either be directly classified by the neural network or detected via an anomaly threshold. This decision has to made based on the chosen data set, since only in a supervised approach a network can be designed to directly classify whether an anomaly is present. In Section 4.3 a suitable method to answer the research questions is proposed. In addition, the advantages of the proposed setup, especially with an outlook on how the achieved results can be compared, are elucidated. Further, in Section 4.3, all global parameters are defined. Global paramters are parameters that need to be defined manually rather than learned, and are valid for all experiments. An example of such a global parameter could be the optimizer function and its corresponding parameters.

### 3.2.3 Experiments

Finally, in the Section Experiments the proposed experimental setup is implemented. The experiments are only conducted on multivariate time series, since Braei and Wagner (2020) already issued a comprehensive study comparing different approaches for anomaly detection on univariate time series. Since transfer learning was not part of the referred study, the experiments on transfer learning can, however, also be done on univariate time series.

In the Chapter 5, the design of the neural networks is described. It includes the determination of the basic architecture, which is specific to the proposed data set, further, it also includes a lot of tuning work to figure out the best parameters for each neural network. To test the performance of the neural networks, also a baseline classifier is established. The baseline classifier is of a very simple nature. Inspired by the trivial null classifier [1] the established baseline will figure as the benchmark to beat for the deep learning approaches.

### 3.2.4 Evaluation - Results Analysis

The Section Results Analysis is dedicated to the examination of the previously achieved results. The results are compared using predefined metrics such as training time, inference time and F1-Score. Looking at these metrics helps to compare the neural network architectures and to determine whether CNN are actually superior to RNN when applied for anomaly detection. Comparing the neural networks to the baseline algorithm further gives some insight on how useful neural netoworks are in general for anomaly detection.

[1] The trivial null classifier always predicts the majority class and thus represents the minimum precision every useful model should surpass.

### 3.2.5 Conclusion

When comparing the neural networks on an anomaly detection task, it is expected that no architecture is overall superior. For example, the high accuracy of RNN comes with the drawback of long training and inference times whereas a CNN with possibly lower accuracy outperforms the RNN escpecially on inference time. Thus, the decision which architecture to use depends on the use case. Therefore, as conclusion a set of recommendations, that shows what architecture is best suited for a certain use case, is compiled.

# Chapter 4

# Experimental Design

In this Chapter, the datasets are selected, general design ideas are explained and established, global parameters are defined and at last the experiments and their goals are described.

## 4.1 Tools

The following Sections provide insight on what tools, such as software and hardware, is used to conduct the experiments.

### 4.1.1 Hardware

All experiments are conducted on a HP Probook equipped with a Intel Core i7 with 2.8GHz and 32GB of RAM installed. No additional graphics card was used.

### 4.1.2 Software

All necessary code is written in R (Version 4.1.0). To design the neural networks, the library Keras (Version 2.4.0) together with the Tensorflow (Version 2.5.0) backend is used.

## 4.2 Datasets

Following, it is motivated how and why the proposed data sets are used.

### 4.2.1 Problems of Existing Benchmarks

In order to find out, which neural network architecture is better suited for anomaly detection, first, suitable datasets have to be evaluated. Most of the papers on anomaly detection test on one of the popular benchmark datasets such as the ones created by Numenta, Yahoo, NASA, or Pei's Lab. These benchmark datasets are, however, declared as flawed by Wu and Keogh (2020). Wu and Keogh state that the benchmark datasets suffer from at least one of the following flaws:

1. **Triviality:** Surprisingly, a sizable proportion of the problems in the benchmark datasets are trivial to solve. Triviality is hereby defined as follows: An anomaly can be found with just one line of code.

2. **Unrealistic Density:** This flaw refers to too many anomalies in the dataset or at least in a certain region, whereas in a real world dataset the anomalous data points make up a portion of just above 0 percent.

3. **Mislabeled Ground Truth:** The data in all of the benchmark datasets appears to be mislabeled, with both false positives and false negatives. This is significant for a number of reasons. The majority of anomaly detectors work by computing statistics for each subsequence of some length. They may, however, place their computed label at the beginning, end, or middle of the subsequence. If caution is not exercised, an algorithm may be penalized for reporting a positive just to the left or right of a labeled region.

4. **Run-to-failure Bias:** Because many real-world systems are run-to-failure, there is often no data to the right of the last anomaly. Therefore, a naïve algorithm that labels the last point as an anomaly has a very good chance of being correct.

In their work, Wu and Keogh, introduced the UCR Time Series Anomaly Datasets as new benchmark, that avoids the problems listed above. However, at the start of this research project the datasets were not publicly available. Because the search for a dataset, that does not suffer from the above mentioned flaws, would be too time-consuming, the decision was taken to partly engineer own datasets.

### 4.2.2 Anomalies

The neural networks should be used to detect various types of anomalies, in order to test their ability to recognize them. Foorthuis (2021) compiled, in an extensive literature review, a study on the different types of anomalies. The anomalies were divided into different categories, of which foremost the quantitative multivariate aggregate anomalies are relevant for this research project, especially a) to f) (see figure 4.1). These types of anomalies typically occur in time series data, that is composed by sensor data. Examples of such data could be temperature measurements or Electrocardiograms.

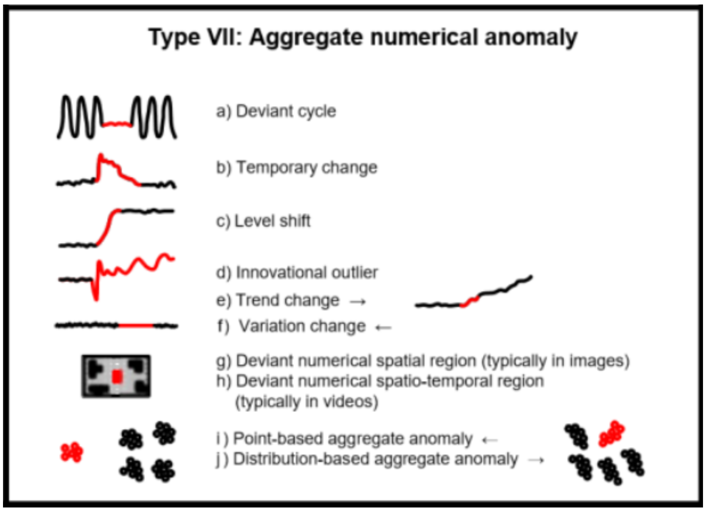

Type VII: Aggregate numerical anomaly
a) Deviant cycle
b) Temporary change
c) Level shift
d) Innovational outlier
e) Trend change →
f) Variation change ←
g) Deviant numerical spatial region (typically in images)
h) Deviant numerical spatio-temporal region (typically in videos)
i) Point-based aggregate anomaly ←
j) Distribution-based aggregate anomaly →

FIGURE 4.1: Quantitative Anomalies (Foorthuis, 2021)

### 4.2.3 Dataset Selection

In the following subsections, it is proposed how and why the datasets are selected for the different experiments.

**1. Dataset**

The dataset, which should be used for the first experiment will be of synthetic nature. It consists of various cyclic patterns. In a second step, the dataset is enriched with anomalies. This way, two dataset are produced. The dataset without anomalies is used for an unsupervised learning approach whereas the dataset with labelled anomalies is used for a supervised approach. Further information, on how the dataset is created, what it looks like and what the anomalies look like can be found in Section 5.1.1

**2. Dataset**

The second dataset, which is used for anomaly detection, should consist of real data. To make sure, that the requirements, mentioned in section 4.2.1 are met, the anomalies are embedded manually into the dataset.

**3. Dataset**

As third dataset, one of the existing benchmark datasets should be used. Despite their obvious flaws, it is still considered useful to validate the previously achieved results on an official benchmark. Further, this gives insights into the overall usefulness of the proposed neural network architectures.

### 4.2.4 Split of Datasets

With all datasets the classical train, validation and test dataset approach is chosen. For the supervised approach, the training data is enriched with anomalies, whereas for the unsupervised approach a "clean" dataset is used. The final evaluation is done on a test dataset, which is the same for all approaches.

## 4.3 Setup of Experiments

The following section explains how the different experiments are conducted in detail. When desinging the experiments, the focus is put on comparability rather than optimally tuned neural networks. Further, it is shown how the datasets and anomalies were engineered.

The following subsections give information on the chosen setups of the experiments that apply to all experiments. Further, global hyper-parameters are defined. Global hyper-parameters are defined manually and are the same for all experiments. Hyper-parameters, which do not fall in this category such as number of neurons in a layer, are specified in Chapter 5 in the section belonging to the respective experiment.

### 4.3.1 Supervised Learning

The supervised learning approach refers to training the neural network on a labelled dataset. The dataset used for training already has the anomalies embedded. The task of finding the anomalies can also be described as a classification task, where the neural networks classifies a sequence as normal or anomalous. In such a case binary crossentropy and a sigmoid activation function are used as loss function and last layer activation fuction. The aforementioned combination means that in the last layer a logistic regression is done, where a threshold is determined to classify the sequences into normal and anomalous.

Supervised classification tasks are used and function well when sufficient samples of all classes are available. In anomaly detection, this is generally not the case, as the anomalous is per definition underrepresented. In the literature (e.g. (Wen and Keyes, 2019)), however, the supervised approach is still successfully used. There are two explanation for this: First, as explained in Section 4.2.1, the density of anomalies is unrealistic and second, the anomalies are of the same kind and always look very similar, so a neural network is able to learn the pattern of the anomaly. In the experiments it is investigated, how the neural networks react when presented with anomalies that are similar to the ones in the training data, but also to previously unseen anomalies of the same kind (e.g. level shift). It is expected, that any neural network will fail when there are not enough similar anomalies in the training data.

### 4.3.2 Unsupervised Learning

The unsupervised learning approach refers to the training of a neural network on a dataset that is free of anomalies. The neural network merely learns the cyclic pattern of the data. When learning the pattern the loss function applied is Mean Absolute Error (MAE), so the learning task is a regression, which itself is supervised. The actual anomaly detection hereby is done in a second step. As proposed in section 2.2.2, the anomaly detector module, which is also used in this experiment, calculates the Euclidean distance between predicted and actual value, where a large value corresponds to an anomaly.

A unsupervised approach is the more reliable choice when trying to detect anomalies, because the model does not need to know what anomalies might look like. However, setting up this approach is more time consuming, especially when dealing with multivariate time series, as for every variable, a separate detector module and corresponding threshold needs to be set up. It is yet to be expected, that both neural network architectures, CNN and RNN, will outperform their supervised counterparts.

### 4.3.3 Neural Networks

The following Sections describe principles that will be used in the design of the neural networks. The described principles and their corresponding paramters are used in all experiments. Parameters that are specific to the experiment are specified in the corresponding section in the Chapter 5.

#### Normalization

Before the data is fed into the neural network it is normalized. Normalization ensures that the magnitude of the values that a feature assumes are more or less the same. Therefore, the mean of each time series is subtracted of each time series, and divided by the standard deviation. As normalization parameters, the mean and standard deviation was used for all, test, validation and training dataset.

#### Activation Function

When designing the neural networks, as activation function generally "ReLu" (Rectified Linear Unit) is used. The function is non-linear and basically just returns the input if it is bigger than 0 and otherwise 0. This function is widely used, because of its simplicity and generally yields good results with little computation expenses.

### Optimizer

As optimizer, the often used default choice in machine learning, ADAM (Adaptive Moment Estimation), with the proposed default values, is applied (Katanforoosh, Kunin, and Ma, 2019). ADAM updates the learning rate when training, making it faster than other optimizers such a Gradient Descent. On the downside, however, ADAM uses a lot memory for a given batch size and is found to generalize poorly in late stages of training.

### Batch Computing

Typically in machine learning, training examples are not used one after another or all at once, because updating the optimizer function after every example or only after the whole dataset is inefficient as it uses a lot of memory. To speed up the process, training data is fed to the machine learning model in batches. A batch is a collection of training examples. The size of these batches, however, also has an impact on the learning ability of the machine learning model. When using an ADAM optimizer, too large batches can have a negative impact on performance of the model (Krishnan, 2019).
When detecting anomlies, the batch size parameter is only used when training. As in a real world scenario every new measurement is analysed immediately since it mostly is crucial to detect the anomaly as early as possible. Analysing every datapoint, however, leads to a increase in computation time which explains the long inference times in the experiments.

## 4.3.4 Experiments

In the following, it is suggested how the experiments on the 3 different datasets are conducted.

### 1. Experiment

In a first experiment the learning abilities of RNN and CNN are compared. It is investigated how useful the architectures are in a supervised and in an unsupervised setup. This experiment gives general insight on which setups and approaches work under which conditions. As the anomalies embedded are similar in the training and test set, the supervised approaches should yield good results, whereas the unsupervised approaches, given the simplicity of the dataset, are not expected to miss any anomalies.

### 2. Experiment

The second experiment is conducted on a more challenging dataset. Also the embedded anomalies are of a more challenging nature. All approaches that have been found successful in Experiment 1, are investigated further on the new dataset. Since the dataset consists of more, partially dependent, variables, and more challanging cyclic patterns the architectures used in Experiment 1 have to be extended for example by adding additional layers.

### 3. Experiment

In the third experiment, a dataset that has already been used in the field of anomaly detection should be used. The neural network architecture types CNN and RNN are applied to the chosen dataset. First, this shows which approach is better suited and second, the achieved results can be compared to already existent results to verify the overall performance of the chosen anomaly detection methods.

### 4.3.5 Results

As results three metrics are reported. First and most important, is the F1-Score which gives insight on the models ability to recognize anomalies. Second and third, the training and inference time are reported.

# Chapter 5

# Experiments

## 5.1 Experiment 1

The first experiment was conducted on a fully synthetic dataset. Supervised and unsupervised learning approaches were used to detect the anomalies.

### 5.1.1 Dataset

The dataset, that was created for this first task consisted of five variables. The dataset was created under the assumption, that one measurement was drawn per hour on totally 2000 days, resulting in 48000 datapoints. The variables all follow a cyclic pattern shown in Figure 5.1 but are not dependent on each other.

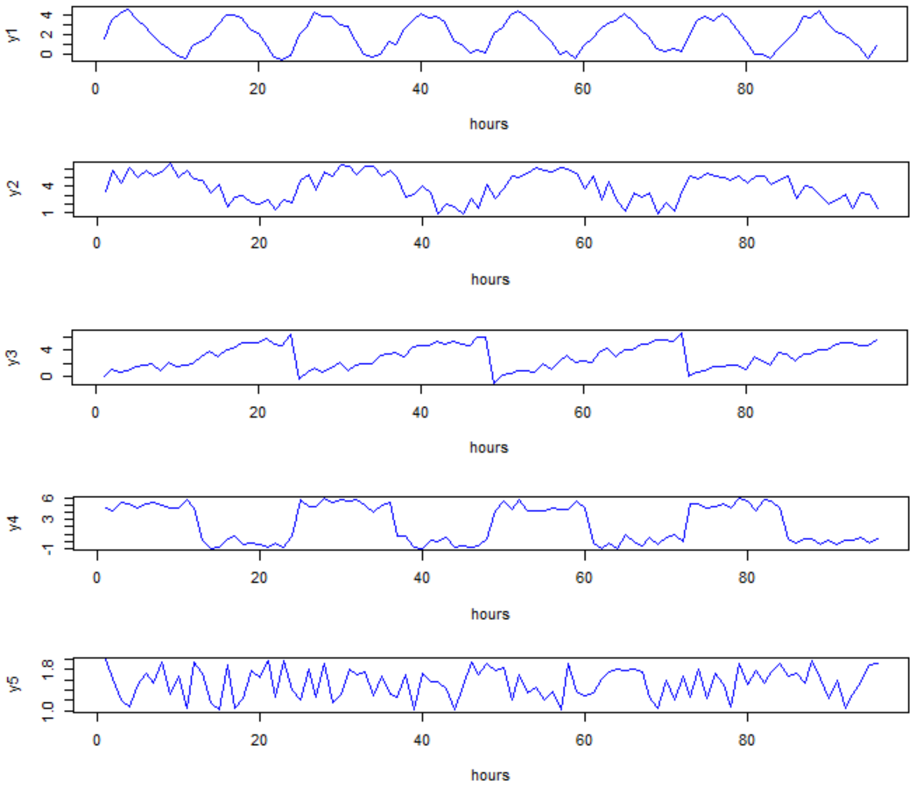


FIGURE 5.1: Synthetic Dataset (**own**)

In a second step the dataset was enriched with six different kinds of anomalies. The anomalies embedded into the dataset are of type deviant cycle, temporary change and level shift. Figure 5.2 shows examples of the embedded anomalies. The same kind of anomalies were embedded into the training and the test dataset.

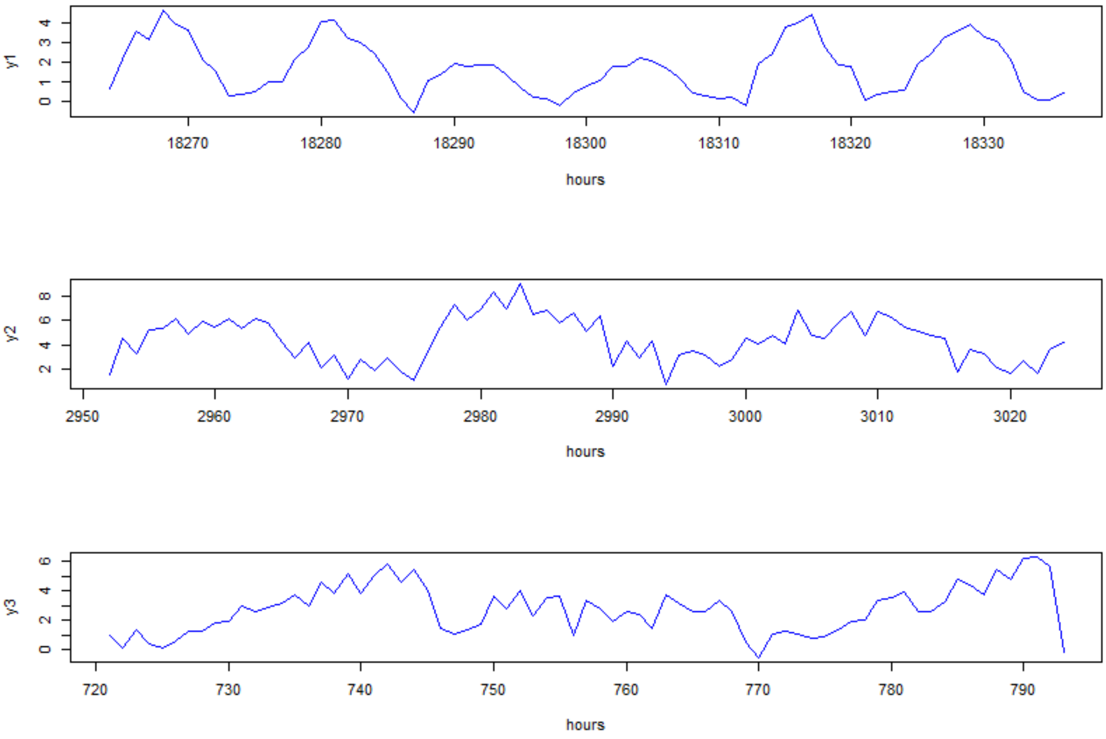


FIGURE 5.2: Synthetic Anomalies (**own**)

For the supervised learning approach the dataset was also labelled. In each case the whole day was marked as an anomaly. In total 30 anomalous days were embedded into the dataset, this corresponds to 1.5 percent of anomalous datapoints.

### 5.1.2 Neural Networks

In the first experiment, a CNN was tested against a RNN in a supervised and also in an unsupervised fashion. As RNN, the type GRU was chosen. As some first attempts showed, that it showed sufficient results with improved computation time compared to the more complex LSTM.

For the supervised learning approach, the chosen architecture can be seen in table 5.1. The architecture with two hidden layers, first presented in Section 2.2, served as example for the RNN. Supplementary, the CNN was built to match the number of layers used for the RNN, where each convolutional layer was followed by a max-pooling layer (explained in Section 1.2.1.

The architecture used for the unsupervised learning approach, displayed in table 5.2, looked very similar to the architecture of the supervised approach. The main difference between the two architectures was that the CNN was not built as a sequential model, because the used software does not support such an architecture. Instead of having one output layer, the CNN had to be designed with five parallel output layers, each predicting

TABLE 5.1: Configuration of Supervised Learning in Exp. 1

| | **Input** | **NN-Architecture** | **Output** |
|---|---|---|---|
| **CNN** | 120 past datapoints | 2 1D-Convolutional Layers<br>2 Max-Pooling Layers<br>1 Dense Layer | 1 Dense Layer<br>with Sigmoid Activation |
| **RNN** | 120 past datapoints | 2 GRU Layers<br>1 Dense Layer | 1 Dense Layer<br>with Sigmoid Activation |

one time series [1]. In comparison, the RNN has just one output layer with 5 neurons, each used to predict a time series.

TABLE 5.2: Configuration of Unupervised Learning in Exp. 1

| | **Input** | **NN-Architecture** | **Output** |
|---|---|---|---|
| **CNN** | 120 past datapoints | 2 1D-Convolutional Layers<br>2 Max-Pooling Layers | 5 Dense Layers with<br>1 regression outputs |
| **RNN** | 120 past datapoints | 2 GRU Layers | 1 Dense Layers with<br>5 regression outputs |

**Learning**

The learning on the data was done using a so called generator function, which iterates over the dataset in predefined steps. The generator function takes the following parameters:

- Lookback - How many data points are considered
- Step - How the data is sampled
- Delay - How many time steps in the future is the target
- Batch Size - The number of samples per batch

For this experiment, the parameter lookback was set to 120 data points in the past, which represent the last 5 days. The parameters step and delay were set to one and as batch size 128 was chosen. This setup can be described as follows: There are 128 ordered samples taken from the the whole dataset. Each of these samples consist of 120 data point of past data. With the step parameter set to one, there is not further subsampling done. With delay set to one, the task for the neural network is to predict the next data point in the future per sample for each variable in the data set. This task is done simultaneously for all samples of batch, which speeds up training.

[1] More detailed summaries of the models can be found in appendix A

### 5.1.3 Results of Experiment 1

Inference time and F1-Score are, for all models, calculated on the same test dataset. As the initial dataset, it consists of 48000 datapoints, with 30 anomalous days embedded. For the supervised approach using a RNN, no F1-Score is reported, as the model, just like the trivial null classifier, always predicted no anomaly. Since the anomalies span over a whole day, for all approaches, the results were averaged per day. Resulting in normal or anomalous days. The reported F1-Score was finally calculated on this data.

TABLE 5.3: Results of Experiment 1

| | **F1-Score** | **Training Time** | **Inference Time** |
|---|---|---|---|
| **CNN Supervised** | 0.991388 | 169s | 422s |
| **RNN Supervised** | - | 967s | 952s |
| **CNN Unsupervised** | 0.9989817 | 129s | 435s |
| **RNN Unsupervised** | 0.9979613 | 300s | 793s |

From the table 5.3, it can be seen that the supervised learning approach takes longer to train. This can be explained by the fact that a CNN must first learn the patterns and only then begins to recognise the anomalies. Further, it shows that despite promising results when training, the trivial null classifier achieves a higher F1-Score than the supervised CNN approach. The best score was achieved with a CNN applied in an unsupervised fashion. The CNN reported just one false negative and a false positive, compared to one false negative and two false positives of the RNN.

## 5.2 Experiment 2

In a second experiment, a real dataset was used as base. The anomalies were embedded manually. Since, the unsupervised approach with RNN in Experiment 1 proved useless it was not included in the second experiment. Since the anomalies in training and test set looked very similar as in Experiment 1, the supervised approach with the CNN was able to recognize some of the anomalies. In the second approach, it should be investigated if this is approach could be of any use, if the anomalies consist of a hitherto unknown pattern.

### 5.2.1 Dataset

The dataset used was derived from the "Appliances Energy Prediction Dataset" available on the UCI Machine Learning Repository. The dataset consists of 9 room temperatures (T1 to T9) and corresponding humidity levels, energy in use of ligth and appliances, two random variables for testing regression models as well as six variables containing weather information. The dataset consists of 19735 datapoints, where 6 datapoints are drawn per hour. Of the available variables only 10 variables were used for the anomaly detection task. The variables used are 5 room temperatures, energy use, outside temperature, air pressure and wind speed. The variables were selected because they show a dependency. For example, a high outside temperature and energy use result in high temperatures in the different rooms. Figure 5.3 shows an extract from the used dataset.

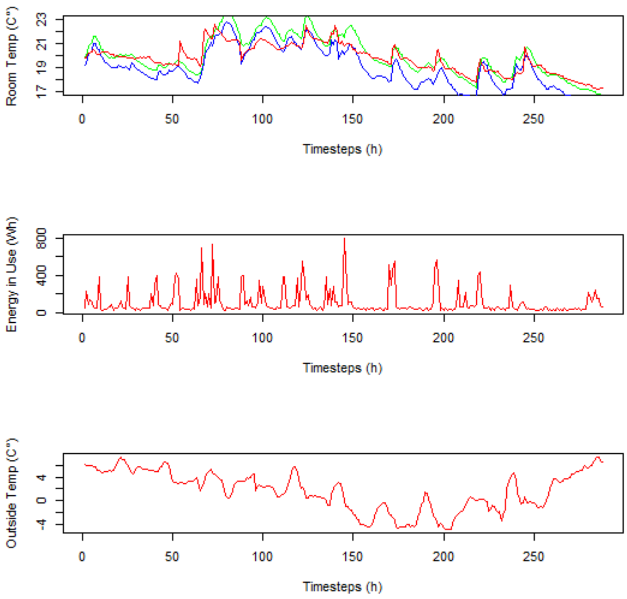


FIGURE 5.3: Appliances Energy Prediction Dataset (**Own or UCI???**)

#### Sampling

The data set contains datapoints measured between November and May. During this time period, it is observed, that the base temperature steadily rises. This poses the problem, that if the model is trained on data from November to March, with April as validation and May as test period, it is biased on the prevailing colder temperatures. To overcome this problem a special sampling technique was applied. In total six samples of the data set were created. A sample was created by randomly drawing one data point per hour over all datapoints. Four of these sample are then used for training, one for validation and one for testing. Using this sampling technique, the model is no longer biased on certain weather conditions but with the drawback, that the test set is not independent of the test and validation set.

### Anomalies

Again, the anomalies were embedded manually into the dataset. The anomalies are of type level shift, deviant cycles, variaton change and distribution-based aggregate anomaly. The anomalies were embedded into the room temperature variables (T1, T2, and T3) and the energy use variable. Figure 5.4 shows examples of the mentioned anomalies.

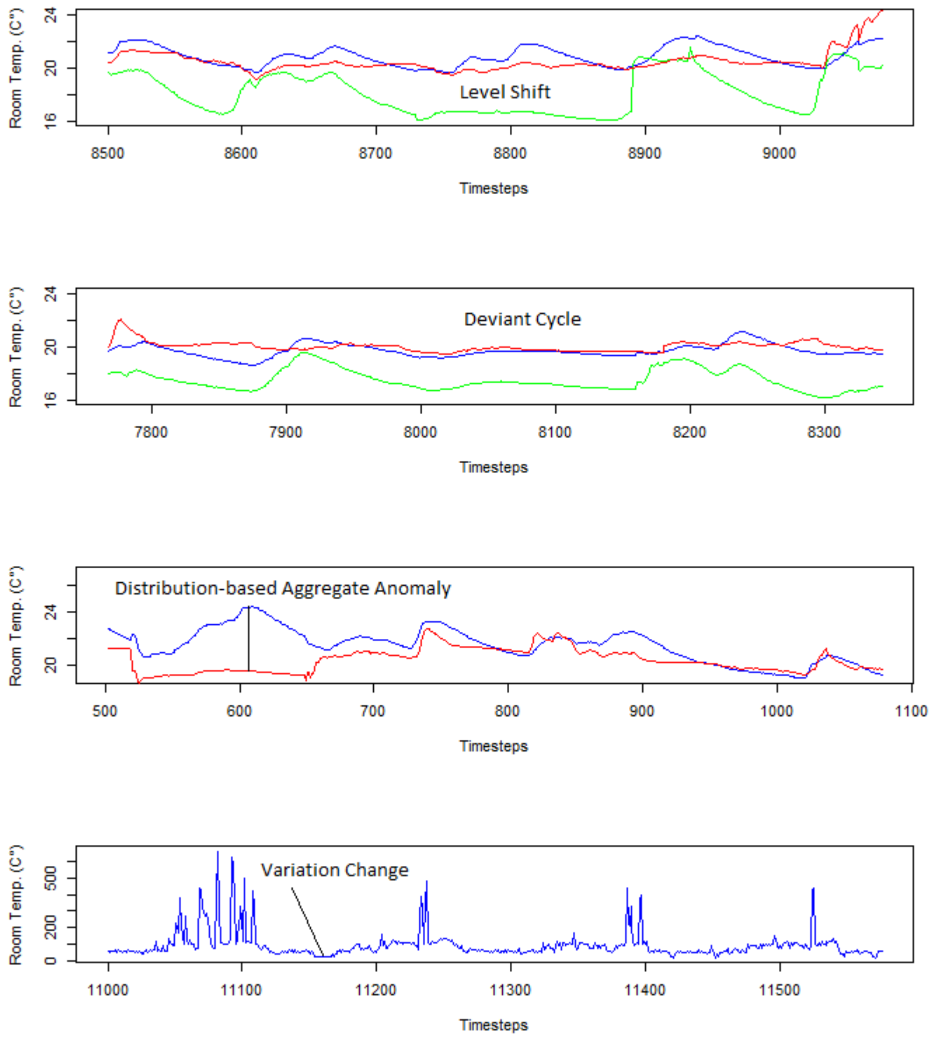


FIGURE 5.4: Examples of Embedded Anomalies (**Own**)

Since the test dataset only consists of around 3000 data points. It was used 3 times so that it does not have an unusual high density of anomalies. In the first instance, anomalies that only affected one variable were embedded. In the second instance, anomalies, that affected all 3 temperature variables, such as a deviant cycle, were embedded. On top, in the energy use variable, variation change anomalies were embedded. In the third instance, distribution based anomalies, which affected T1 and T2, were embedded.

### 5.2.2 Neural Networks

Since the RNN did not show sufficient results in Experiment 1 when given a classification task, it was excluded from Experiment 2. The CNN, however, which had probably learned the patterns of the anomalies, was tested again. Since the anomalies in the training and test set are not as similar anymore as in Experiment 1, it is expected that the CNN classifier performs very poorly.

TABLE 5.4: Configuration of Unupervised Learning in Exp. 2

| | Input | NN-Architecture | Output |
|---|---|---|---|
| **CNN** | 288 past datapoints | 3 1D-Convolutional Layers<br>1 Max-Pooling Layers | 4 Dense Layers with<br>1 regression outputs |
| **RNN** | 288 past datapoints | 3 LSTM Layers | 1 Dense Layers with<br>4 regression outputs |

Because of the complexity of the dataset, in the second experiment it was decided to use LSTM units instead of GRU in the RNN architecture. LSTM units generally provide more accurate results but use more memory and therefore more computation time (Lendave, 2021). The designed neural network architecture consisted of 3 sequential layers of LSTM layers and one dense layer with four neurons.

The CNN was designed as follows, three layers of one-dimensional convolutional layers were used. After the first layer, a max-pooling layer was added, to reduce the feature space and improve computation time. Since the CNN is expected to predict a time series, the feature space was not further reduced through max-pooling layers, since it would negatively affect the accuracy.

#### Learning

The generator function for this experiment was configured as follows:

- Lookback - 288 past data points: 12 days in the past
- Step - 1: no further subsampling is done
- Delay - 1: the next time step in the future is predicted
- Batch Size - 128 samples per batch

First experiments were done only using data from T1, T2, T3, energy use and outside temperature to predict T1 to T3 and appliances energy use. However, the achieved results were not satisfactory. The results were not accurate enough to predict the anomalies. It was theorized, that the training data did not contain enough information to reliably predict the future, therefore additional variables were added. Adding more variables to predict the future resulted in a better MAE. Finally, the following variables were used to make predictions: T1 to T5, energy use of appliances and ligths, outside temperature, air pressure and windspeed. As the anomlies, however, were only embedded into 4 out of the 10 variables, only the affected 4 were predicted.

### 5.2.3 Results of Experiment 2

Because the dataset was much more challenging, the reported scores are clearly inferior to Experiment 1. In total there were 13 anomalies to be detected in the test sets. The difference in F1-Score between the CNN and the RNN results from one anomaly which was not recognized by the CNN. However, training the RNN takes over 30 times longer than the CNN.

TABLE 5.5: Results of Experiment 2

| | **F1-Score** | **Training Time** | **Inference Time** |
|---|---|---|---|
| **CNN Unsupervised** | 0.666666 | 172s | 24s* |
| **RNN Unsupervised** | 0.727272 | 5420s | 93s* |

* the inference time reported is on just one instance of the test dataset

The above reported results show the overall performance. Since the dataset was more complex than in the first experiment, the results are investigated further to show how the different architecture performed on the various anomalies and how accurate the predictions were on the different variables. First of all, the average error of the predicted variables are investigated further. The below results are reported from the first test set.

TABLE 5.6: MAE of Predictions

| | **CNN** | **RNN** |
|---|---|---|
| **T1** | 0.2712066° C | 0.1741582° C |
| **T2** | 0.5236905° C | 0.9009765° C |
| **T3** | 0.3210329° C | 0.7244096° C |
| **Appliances** | 29.64146 Wh | 72.91079 Wh |

From Table 5.6, it can be seen that the CNN outperforms the RNN on all variables except T1. The RNN, however, performs exceptionally well on T1. Comparing these results to the performance of the RNN on the first test set shows that only the anomalies that were embedded in T1 are detected. A similar picture emerges in the second test set, since the anomalies affect all 3 variables, a deflection should be visible on all three reported anomaly scores. However, the anomalies detected by the RNN were only visible in the anomaly score reported off T1. While the CNN is not as accurate on any of the variables as the RNN is on T1, this is how the CNN was outperformed. Out off the six anomalies embedded in the second test set, the CNN only recognized one, while the RNN recognized three out of six. On the third test set, both architecture types were able to recognize all anomalies.

Further, as expected, both architecture types were not able to find the variation change anomalies embedded in the variable appliances. The anomaly was constructed as when too many times the exact same value is reported. This anomaly type is with the necessary domain knowledge relatively easy to find.

## 5.3 Experiment 3

In the third experiment, a dataset that was dedicated to anomaly detection tasks was used. The achieved results can therefore be verified on previously achieved results.

### 5.3.1 Dataset

The dataset was introduced by Filonov et al. (Filonov, Lavrentyev, and Vorontsov, 2016). The data was created by modelling a gasoil plant heating loop The GHL (Gasoil Heating Loop) model has three reservoirs: a receiving tank (RT), a heating tank (HT), and a collection tank (CT). The technological challenge is to heat gasoil in RT to 60 degrees Celsius, resulting in a viscosity sufficient for transmission to CT. In the model, the heating is done in stages. A fraction of gasoil is heated to 60 degrees Celsius in HT and then pumped back into RT to relax for a while. This technique is done until the RT temperature reaches 60 degrees Celsius. After that, the RT is poured into the CT. Following that, RT is refilled from an infinite source.

The training dataset consists of 19 variables and around 1.5 million datapoints with no anomalies, the most important, according to Filonov et al. (Filonov, Lavrentyev, and Vorontsov, 2016) are shown in Figure 5.5. The sensors for RT level, RT temperature, and HT temperature are the first three variables. The remaining two variables relate to control signals for turning on/off the gasoil supply and turning on/off the heater.

For testing the trained models, Filonov et al. (2016) provide 48 test sets, which are further decribed in Section 5.3.1

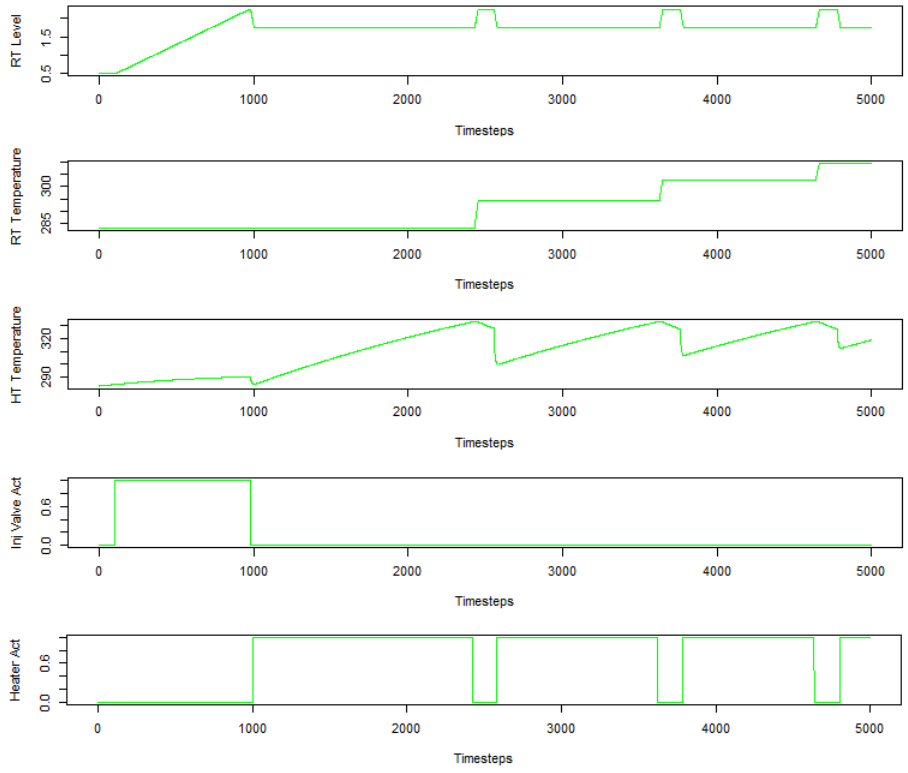


FIGURE 5.5: Examples of Embedded Anomalies (**Own**)

### Sampling

The dataset intended for training contains 1.5 million data points, on top all 48 available test sets contain around 200’000 data points. The training of the neural networks and the evaluation of the test sets would therefore take a lot of time. Inspired by the work of Wen and Keyes (2019), who took snapshots of length 50’000 and then downsampled them to a length of 1’024, the data should also be downsampled to save time. Downsampling refers to reducing the number of data points in a time series. There are different techniques to achieve downsampling. The technique applied in this case is called decimation. Since the data contains many repeating data points, which have the same value, only every 10th datapoint was kept, which corresponds to a decimation by a factor of 10. This greatly reduces the computation time, since the training dataset now contains only 150’000 data points while a test set contains 20’000 data points. On the downside, using this technique to downsample information is lost, which possibly effects the results of the chosen anomaly detection methods. Figure 5.6 shows how the variable RT Level behaves on 5’000 data points pre- and post-downsampling.

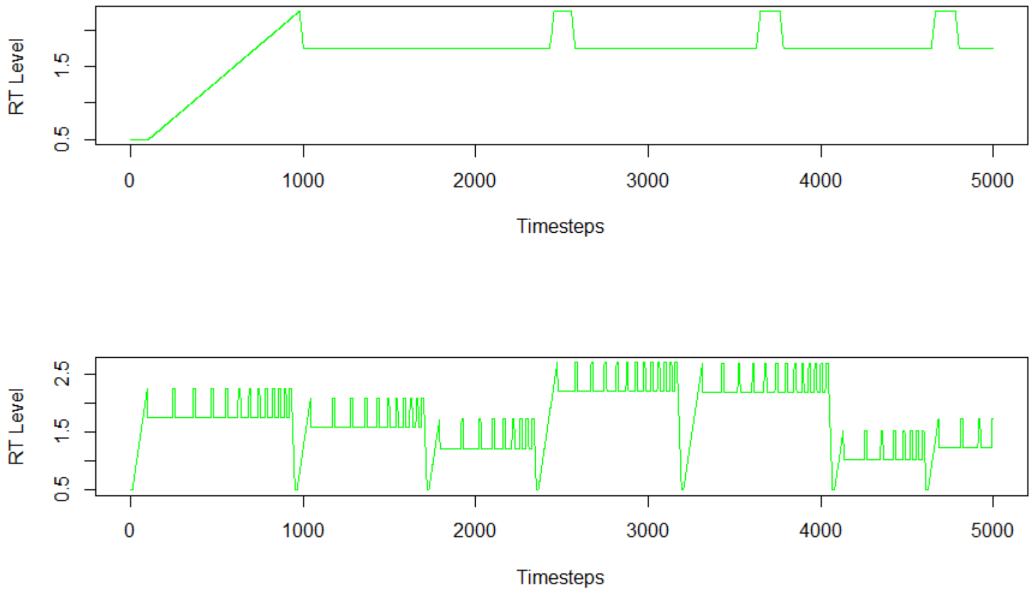


FIGURE 5.6: Effect of Downsampling on RT Level (**Own**)

### Anomalies

As anomalies cyber attacks were introduced. The anomalies each concern the variables RT Level and HT Temperature, which had been changed without authorisation. By setting the value for RT Level or HT Temperature to different levels and varying the the time of attack 48 test sets were created. Each test set contains only one anomaly of either type. Figure 5.7 and Figure 5.8 give an example of the anomalies look like. In red, the variable Danger, which is the control variable that indicates the anomaly, is shown [2].

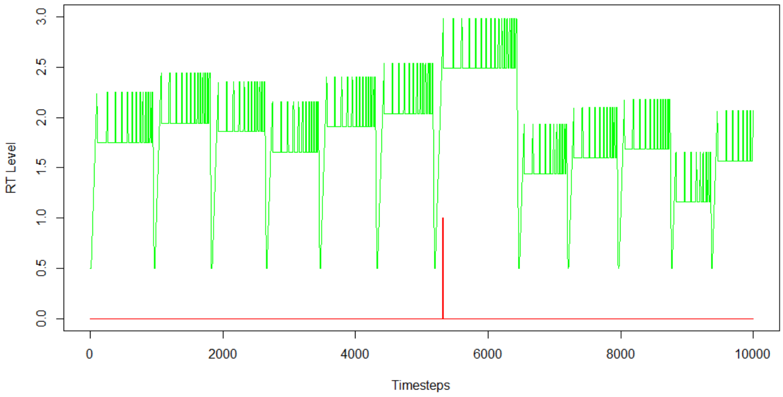


FIGURE 5.7: Attack on max-RT-level set point (**Own**)

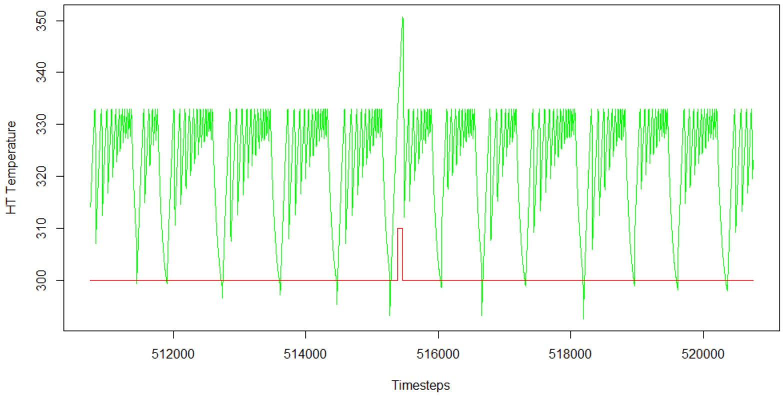


FIGURE 5.8: Unauthorized change of max HT temperature (**Own**)

Since the anomalies affect only the two above shown variables, the task for the neural networks was defined to predict the two above mentioned variables. Out of the past information of all 19 variables a neural network is expected to reliably predict the two monitored variables.

[2] In Figure 5.8 the variable Danger was artificially lifted to a base level of 300 instead of 0 and multiplied by 10 for better visibility

### 5.3.2 Neural Networks

As described in Section 5.3.1, the provided training dataset does not contain any anomalies, therefore only unsupervised training approaches are used. Wen and Keyes (2019) used the GHL dataset, for a supervised approach, but used part of the test set for training, therefore making it not possible to compare the results to works of Filonov et al. (2016).

In Experiment 3, the chosen neural network architecture is again a LSTM. Since, Filonov et al. (2016) use a LSTM network in their experiment, in order to be able to compare the experiments the same neural network architecture was chosen. However, Filonov et al. only report in their work, that two LSTM layers and a dropout layer were used. The exact configuration of the LSTM layers is not evident. Therefore, the number of units per layers was experimentally determined. Finally, 32 units and 24 units were used in the first and in the second layer respectively. The dropout layer, which was added to reduce overfitting, was configured with a dropout rate of 0.1, which was adapted for this experiment.

TABLE 5.7: Configuration of Unupervised Learning in Exp. 3

| | **Input** | **NN-Architecture** | **Output** |
|---|---|---|---|
| **CNN** | 100 past datapoints | 3 1D-Convolutional Layers<br>2 Dropout Layers | 2 Dense Layers with<br>1 regression outputs |
| **RNN** | 100 past datapoints | 2 LSTM Layers<br>1 Dropout Layer | 1 Dense Layers with<br>2 regression outputs |

Some early experiments with a two-layered CNN yielded in poor results. Therefore, an additional convolutional layer was added. Max-pooling layers were initially included in Experiment 2 but were found to have a negative impact on the ability to accurately predict the required time series, so they were removed entirely in Experiment 3. Since the CNN now also had a tendency to overfit, two dropout layers were added.

#### Learning

The main challange when training the neural network was to determine an appropriate lookback. Initial experiments were carried out with 1000 - 5000 past data points that were considered. Although, the computation time increased significantly, the reported results were not satisfactory. Looking at the work of Wen and Keyes (2019), who trained their network on snapshots of length 300 and applied a similarly rigorous sampling technique, the lookback paramter was set accordingly. Experiments with a lookback of between 300 and 500 past data points, improved not only the reported MAE, but also had a positive effect on computation time. While the results for the lookbacks between 300 and 500 were not significantly different, it was determined to use 300 because it resulted in the shortest computation time. The generator function for this experiment was, therefore, configured as follows:

- Lookback - 300 past data points: 1000 seconds in the past
- Step - 1: no further subsampling is done
- Delay - 1: the next time step in the future is predicted
- Batch Size - 128 samples per batch

### 5.3.3 Results of Experiment 3

As in the previously described experiments, the anomaly score was calculated as the difference between predicted value and actual value. Figure 5.9 shows that calculating the different results in a graph with many peaks where the anomaly is not clearly visible. Filonov et al. (2016) experienced the same problem and applied a exponential rolling mean on the calculated difference to smooth the outliers. The same technique is used, to smooth the difference calculated in Experiment 3, although only a simple rolling mean is calculated. The lower graph of Figure 5.9 shows the time series after the rolling mean was applied together with the Danger control variable in red and the applied threshold in blue.

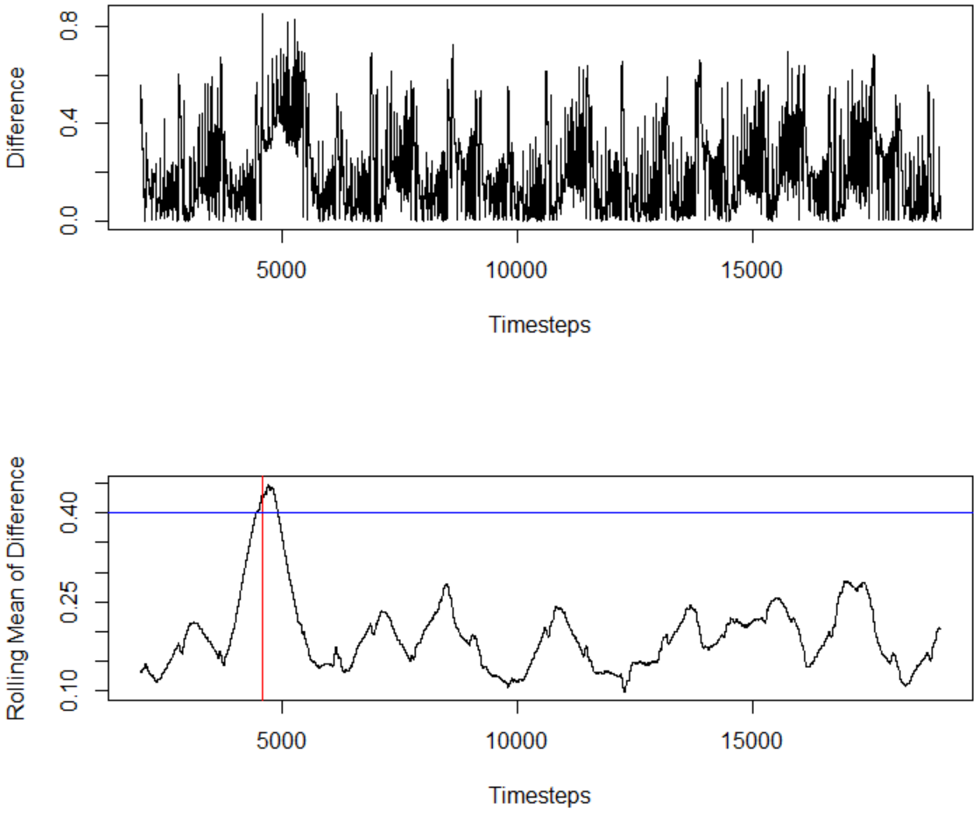


FIGURE 5.9: Effect of Rolling Mean Calculation (**Own**)

Notable, when looking at the predictions on the test sets, is that every set had a ramp up behaviour where the predicted and actual did not match causing an anomaly at the start of all test sets. This behaviour was observed with both, the CNN and the RNN model, leading to the conclusion that this behaviour was created when the test set was generated. It was assumed, that this behaviour does not occur during operation. The anomalies caused by the ramp up phase were therefore not counted when calculating the final result. Further, when calculating the result, a series of anomalies that occur in quick succession are counted as just one anomaly as this refers to one of the identified flaws of the existing benchmarks (see Section 4.2.1). However, because of the down-sampling of the data and the application of a rolling mean to the predicted result, it would also no longer be possible to recognize the separate peaks caused by a series of anomalies. Since it is unknown whether Filonov et. al (2016) used the same approach, it is possible that better F1-Scores are achieved this way, but actually the task was simplified.
Although Wen and Keyes (2019) also used the GHL data for their experiment, the results cannot be compared as they used parts of the test set to train their model, meaning they could not evaluate their model on the whole test set.

TABLE 5.8: Results of Experiment 3

| | **F1-Score** | **Training Time** | **Inference Time** |
|---|---|---|---|
| **CNN Unsupervised** | 0.8696 | 142s | 185s* |
| **RNN Unsupervised** | 0.9053 | 467s | 530s* |
| **RNN Unsupervised**** | 0.872 | - | - |

* the inference time reported is on just one instance of the test dataset
** Benchmark achieved by (Filonov, Lavrentyev, and Vorontsov, 2016)

# Chapter 6

# Results Analysis

In this chapter, the results of all experiments are compared and discussed to evaluate which approach is suitable in what scenario.

The experiments quickly showed that when trying to detect anomalies in time series with deep learning approaches, the decribed unsupervised approach is the preferred choice. As supervised approaches need to assure, that the anomalies that occur when the model is in operation are the same as in the training set. Further, a training set that contains enough anomalies must exist. Only under these limiting conditions a supervised model can reliably be deployed. When these requirements are fullfilled a Convolutional Neural Network might be a good choice. Experiment 1 showed that a CNN, in comparison to a RNN, is capable of recognizing the anomalies. The same conclusion was reached by Wen and Keyes (2019), who successfully applied a special version of a CNN (U-Net: Section 2.3.2) to the GHL dataset.

In contrast to the supervised approach the unsupervised approach generally proved to be more stable. As it needs no knowledge of the nature of the anomalies. The only requirement for this approach is a sufficiently large training set to learn the normal behaviour. Both neural network types, CNN and RNN, showed similar abilities when learning the normal behaviour. This means that CNN, that are usually applied for tasks such image recognition, are similarly suitable to predict time series, which is normally the domain of RNN. Looking at the F1-Scores of Experiments 1 to 3, shows that CNN can be useful alternative to RNN when detecting anomalies. Although, the F1-Score of the CNN is often marginally lower than the F1-Score of the compared RNN, the CNN has another big advantage. When tracking anomalies, it is often a neccessary to detect them as quickly as possible. Looking at the inference times of the various models, it can be seen that the CNN model is up to three times faster than the RNN model. This advantage of speed can be of paramount importance in a real world scenario, as it can decide if delicate components of a system take damage or not.

The CNN is not only faster at inference but also in training. As there many hyperparameters, such as batch size, lookback, layers etc., to be determined for an optimal model performance, the training time becomes important. When using a machine learning algorithm, a model is trained again and again to find the best configuration. So the training time of a model decides how fast a well performing model can be developed. Looking at the training times achieved in this work, it can be seen that the CNN learns much faster. This might have two reasons. First, when the number of parameters in the CNN and RNN models are compared, it can be observed, that a CNN generally has less parameters (see Appendix A). When training a neural network, after every batch, it needs to be calculated how much these parameters need to be adjusted. This makes it more and more computationally expensive the more parameters there are. Second, it seems to be good strategy to look at a time series as a pattern. Where the RNN calculated the influence of the past on the future, the CNN tries to recognise patterns in the data. This approach not only worked when the time series followed a cyclic pattern as Experiment 1 and 3 but also in the case of the weather data in Experiment 2.

Ultimately, as it is often the case in data science, the choice whether to use RNN or CNN is a tradeoff, in this particular case, between speed and quality of the result. A tradeoff, which might, however, be mitigated, in the favor of the CNN, by using more advanced versions of the CNN such as the U-Net.

# Chapter 7

# Conclusion

This final chapter seeks to answer the research questions proposed in Section 1.4.1. Further, the results of this work are discussed in regard to the thesis statement proposed in Section 1.4

## 7.1 Answering the Research Questions

**How does a CNN for univariate and multivariate data need to be designed for successful anomaly detection in time series data?** Munir (2019) showed that for univariate time series data 1-dimensional convolutional layers can successfully be used. Wen and Keyes (2019) also use 1-dimensional convolutional layers for their proposed architecture type, the U-Net. The further design of the CNN for anomaly detection is dependent on the given task. If a supervised approach is deemed suitable, the CNN is defined as a classifier deciding whether a given sequence is anomalous or not. However, this approach is reserved for only a few special use cases where the shapes of the anomalies are known and unchanging. If it is unknown how the anomalies look like, the CNN has to be designed to predict the future of a time series. Given this so-called regression task, it is preferable to omit the max-pooling layers, that are normally part of CNN architecture, because they impair a CNN's ability to reliably predict a time series.

**What advantages and disadvantages arise when using a CNN compared to a RNN for anomaly detection in univariate and multivariate time series?** The main advantage of a CNN over an RNN, such as a GRU- or LSTM-neural network, is its shorter computation time. The research revealed that utilizing a CNN reduces the amount of time it takes to train and infer. This is especially useful in the use case of anomaly detection as it is crucial to recognize the anomalies as fast as possible.

The second big advantage of a CNN is its ability to recognize anomalies. Given a case where the anomalous behaviour always follows the same pattern as in Experiment 1, a CNN can reliably be trained to recognize this pattern. A real world scenario for this use case be the frequency analysis of a ball bearing (Mais, 2002).

In the scope of this work no major disadvantages were found when applying CNN to detect anomalies.

**What hyper-parameter settings are crucial for a fair performance comparison between RNN and CNN?**

## 7.2 Comment on Thesis Statement

The experiments conducted in this work did not show that Convolutional Neural Networks are overall superior to Recurrent Neural Networks.

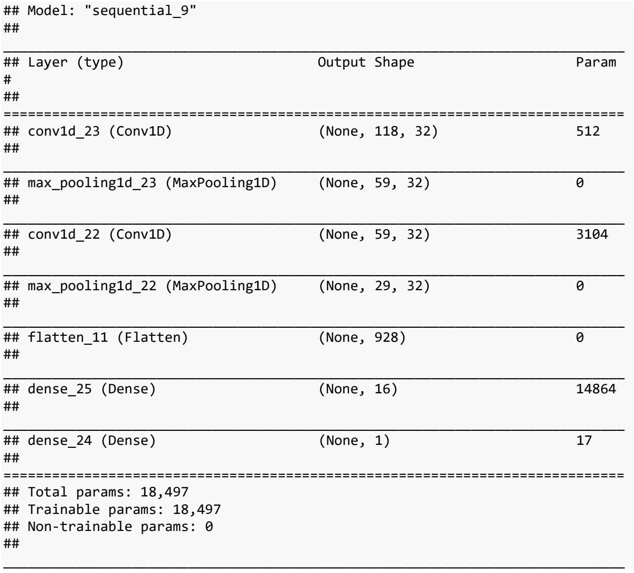
The second advantage of the CNN is its shorter inference time.

# Appendix A

# Model Summaries

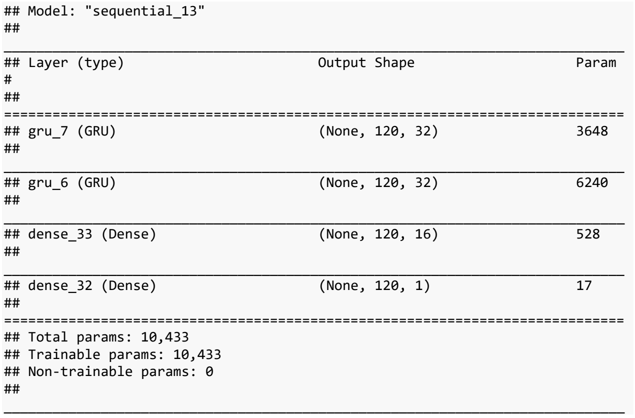
## A.1 Summary of Models of Experiment 1

### A.1.1 Supervised Learning



```
## Model: "sequential_9"
##
_____________________________________________________________________________
## Layer (type)                        Output Shape                    Param
#
##
=============================================================================
## conv1d_23 (Conv1D)                  (None, 118, 32)                 512
##
_____________________________________________________________________________
## max_pooling1d_23 (MaxPooling1D)     (None, 59, 32)                  0
##
_____________________________________________________________________________
## conv1d_22 (Conv1D)                  (None, 59, 32)                  3104
##
_____________________________________________________________________________
## max_pooling1d_22 (MaxPooling1D)     (None, 29, 32)                  0
##
_____________________________________________________________________________
## flatten_11 (Flatten)                (None, 928)                     0
##
_____________________________________________________________________________
## dense_25 (Dense)                    (None, 16)                      14864
##
_____________________________________________________________________________
## dense_24 (Dense)                    (None, 1)                       17
##
=============================================================================
## Total params: 18,497
## Trainable params: 18,497
## Non-trainable params: 0
##
_____________________________________________________________________________
```

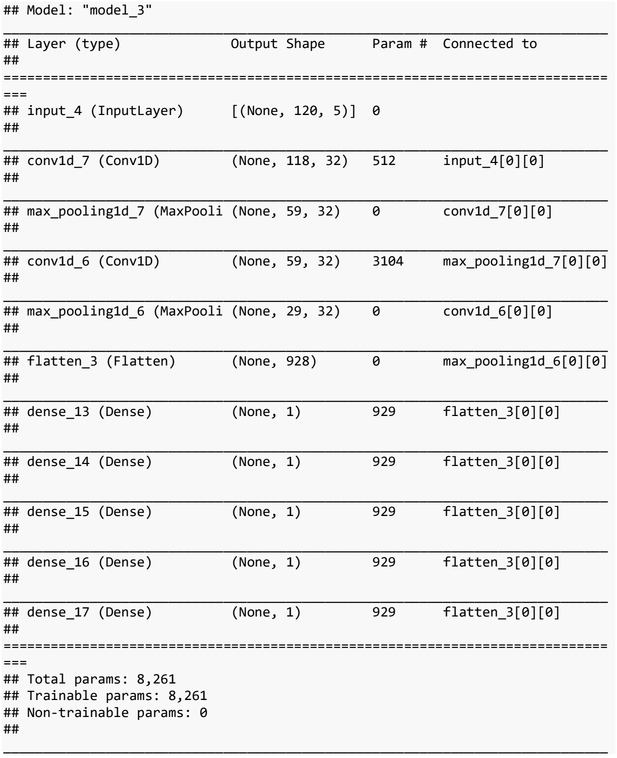
FIGURE A.1: Summary of CNN (**own**)



```
## Model: "sequential_13"
##
_____________________________________________________________________________
## Layer (type)                        Output Shape                    Param
#
##
=============================================================================
## gru_7 (GRU)                         (None, 120, 32)                 3648
##
_____________________________________________________________________________
## gru_6 (GRU)                         (None, 120, 32)                 6240
##
_____________________________________________________________________________
## dense_33 (Dense)                    (None, 120, 16)                 528
##
_____________________________________________________________________________
## dense_32 (Dense)                    (None, 120, 1)                  17
##
=============================================================================
## Total params: 10,433
## Trainable params: 10,433
## Non-trainable params: 0
##
_____________________________________________________________________________
```

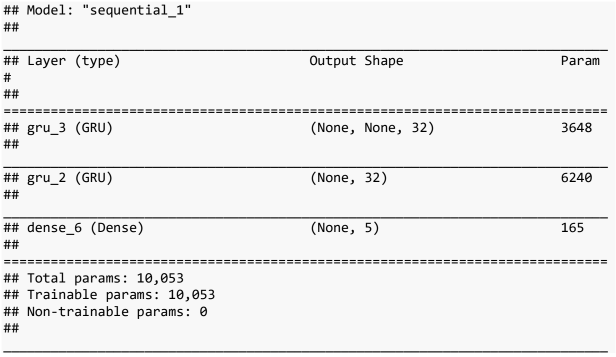
FIGURE A.2: Summary of GRU (**own**)

### A.1.2 Unsupervised Learning



```
## Model: "model_3"
_____________________________________________________________________________
## Layer (type)              Output Shape      Param #  Connected to
##
=============================================================================
===
## input_4 (InputLayer)      [(None, 120, 5)]  0
##
_____________________________________________________________________________
## conv1d_7 (Conv1D)         (None, 118, 32)   512      input_4[0][0]
##
_____________________________________________________________________________
## max_pooling1d_7 (MaxPooli (None, 59, 32)    0        conv1d_7[0][0]
##
_____________________________________________________________________________
## conv1d_6 (Conv1D)         (None, 59, 32)    3104     max_pooling1d_7[0][0]
##
_____________________________________________________________________________
## max_pooling1d_6 (MaxPooli (None, 29, 32)    0        conv1d_6[0][0]
##
_____________________________________________________________________________
## flatten_3 (Flatten)       (None, 928)       0        max_pooling1d_6[0][0]
##
_____________________________________________________________________________
## dense_13 (Dense)          (None, 1)         929      flatten_3[0][0]
##
_____________________________________________________________________________
## dense_14 (Dense)          (None, 1)         929      flatten_3[0][0]
##
_____________________________________________________________________________
## dense_15 (Dense)          (None, 1)         929      flatten_3[0][0]
##
_____________________________________________________________________________
## dense_16 (Dense)          (None, 1)         929      flatten_3[0][0]
##
_____________________________________________________________________________
## dense_17 (Dense)          (None, 1)         929      flatten_3[0][0]
##
=============================================================================
===
## Total params: 8,261
## Trainable params: 8,261
## Non-trainable params: 0
##
_____________________________________________________________________________
```

FIGURE A.3: Summary of CNN (**own**)



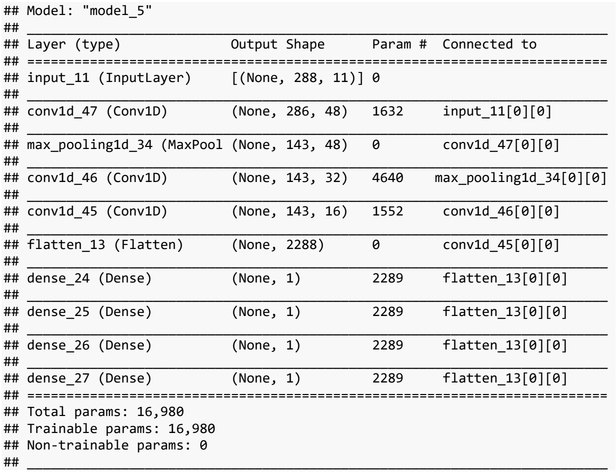
```
## Model: "sequential_1"
##
_____________________________________________________________________________
## Layer (type)                        Output Shape                    Param
#
##
=============================================================================
## gru_3 (GRU)                         (None, None, 32)                3648
##
_____________________________________________________________________________
## gru_2 (GRU)                         (None, 32)                      6240
##
_____________________________________________________________________________
## dense_6 (Dense)                     (None, 5)                       165
##
=============================================================================
## Total params: 10,053
## Trainable params: 10,053
## Non-trainable params: 0
##
_____________________________________________________________________________
```

FIGURE A.4: Summary of GRU (**own**)

## A.2 Summary of Models of Experiment 2

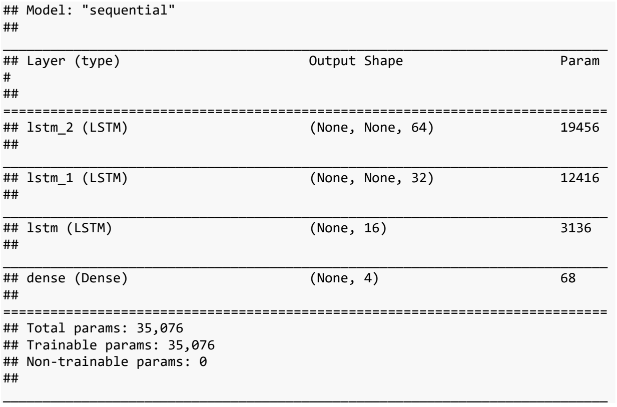
### A.2.1 Supervised Learning

### A.2.2 Unsupervised Learning



```
## Model: "model_5"
## __________________________________________________________________________
## Layer (type)              Output Shape      Param #  Connected to
## ==========================================================================
## input_11 (InputLayer)     [(None, 288, 11)] 0
## __________________________________________________________________________
## conv1d_47 (Conv1D)        (None, 286, 48)   1632     input_11[0][0]
## __________________________________________________________________________
## max_pooling1d_34 (MaxPool (None, 143, 48)   0        conv1d_47[0][0]
## __________________________________________________________________________
## conv1d_46 (Conv1D)        (None, 143, 32)   4640    max_pooling1d_34[0][0]
## __________________________________________________________________________
## conv1d_45 (Conv1D)        (None, 143, 16)   1552     conv1d_46[0][0]
## __________________________________________________________________________
## flatten_13 (Flatten)      (None, 2288)      0        conv1d_45[0][0]
## __________________________________________________________________________
## dense_24 (Dense)          (None, 1)         2289     flatten_13[0][0]
## __________________________________________________________________________
## dense_25 (Dense)          (None, 1)         2289     flatten_13[0][0]
## __________________________________________________________________________
## dense_26 (Dense)          (None, 1)         2289     flatten_13[0][0]
## __________________________________________________________________________
## dense_27 (Dense)          (None, 1)         2289     flatten_13[0][0]
## ==========================================================================
## Total params: 16,980
## Trainable params: 16,980
## Non-trainable params: 0
## __________________________________________________________________________
```

FIGURE A.5: Summary of CNN (**own**)

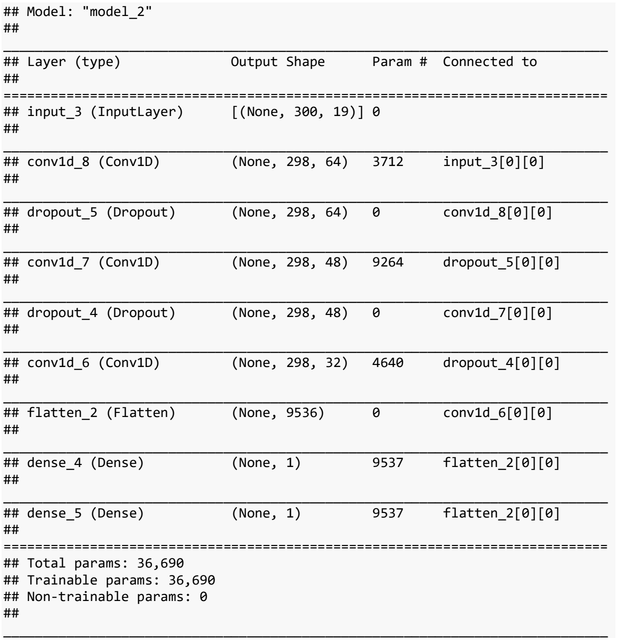


```
## Model: "sequential"
##
_____________________________________________________________________________
## Layer (type)                        Output Shape                    Param
#
##
=============================================================================
## lstm_2 (LSTM)                       (None, None, 64)                19456
##
_____________________________________________________________________________
## lstm_1 (LSTM)                       (None, None, 32)                12416
##
_____________________________________________________________________________
## lstm (LSTM)                         (None, 16)                      3136
##
_____________________________________________________________________________
## dense (Dense)                       (None, 4)                       68
##
=============================================================================
## Total params: 35,076
## Trainable params: 35,076
## Non-trainable params: 0
##
_____________________________________________________________________________
```

FIGURE A.6: Summary of LSTM (**own**)

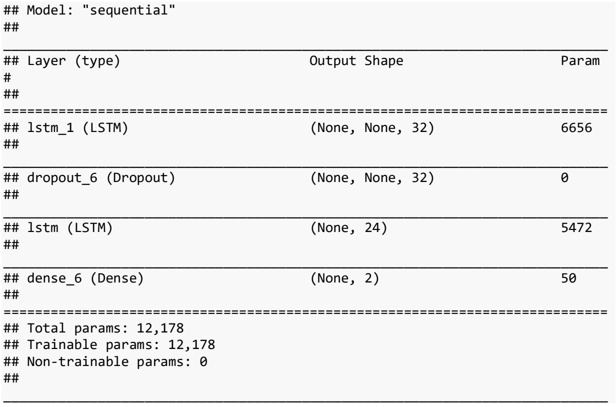
# A.3 Summary of Models of Experiment 3

## A.3.1 Unsupervised Learning



```
## Model: "model_2"
##
_____________________________________________________________________________
## Layer (type)              Output Shape      Param #  Connected to
##
=============================================================================
## input_3 (InputLayer)      [(None, 300, 19)] 0
##
_____________________________________________________________________________
## conv1d_8 (Conv1D)         (None, 298, 64)   3712     input_3[0][0]
##
_____________________________________________________________________________
## dropout_5 (Dropout)       (None, 298, 64)   0        conv1d_8[0][0]
##
_____________________________________________________________________________
## conv1d_7 (Conv1D)         (None, 298, 48)   9264     dropout_5[0][0]
##
_____________________________________________________________________________
## dropout_4 (Dropout)       (None, 298, 48)   0        conv1d_7[0][0]
##
_____________________________________________________________________________
## conv1d_6 (Conv1D)         (None, 298, 32)   4640     dropout_4[0][0]
##
_____________________________________________________________________________
## flatten_2 (Flatten)       (None, 9536)      0        conv1d_6[0][0]
##
_____________________________________________________________________________
## dense_4 (Dense)           (None, 1)         9537     flatten_2[0][0]
##
_____________________________________________________________________________
## dense_5 (Dense)           (None, 1)         9537     flatten_2[0][0]
##
=============================================================================
## Total params: 36,690
## Trainable params: 36,690
## Non-trainable params: 0
##
_____________________________________________________________________________
```

FIGURE A.7: Summary of CNN (**own**)



```
## Model: "sequential"
##
_____________________________________________________________________________
## Layer (type)                        Output Shape                     Param
#
##
=============================================================================
## lstm_1 (LSTM)                       (None, None, 32)                 6656
##
_____________________________________________________________________________
## dropout_6 (Dropout)                 (None, None, 32)                 0
##
_____________________________________________________________________________
## lstm (LSTM)                         (None, 24)                       5472
##
_____________________________________________________________________________
## dense_6 (Dense)                     (None, 2)                        50
##
=============================================================================
## Total params: 12,178
## Trainable params: 12,178
## Non-trainable params: 0
##
_____________________________________________________________________________
```

FIGURE A.8: Summary of LSTM (**own**)

# Bibliography

Aggarwal, Charu C. (2013). *Outlier analysis.* Vol. 9781461463, pp. 1–446. ISBN: 9781461463962. DOI: 10.1007/978-1-4614-6396-2.

Alansari, Zainab et al. (2018). “The rise of Internet of Things (IoT) in big healthcare data: Review and open research issues”. In: vol. 564, pp. 675–685. ISBN: 9789811068744. DOI: 10.1007/978-981-10-6875-1_66.

Arindam Chandola Varun, Banerjee and Vipin Kumar (2009). “Anomaly detection: A survey”. In: *ACM computing surveys (CSUR)* 41 (3), pp. 1–58.

Braei, Mohammad and Sebastian Wagner (2020). *Anomaly detection in univariate time-series: A survey on the state-of-the-art.*

Britz, Denny (Sept. 2015). *Implementing a Neural Network from Scratch in Python – An Introduction.* URL: http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/.

Chung, Junyoung et al. (Dec. 2014). “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: *NIPS 2014 Deep Learning and Representation Learning Workshop.* URL: http://arxiv.org/abs/1412.3555.

Dutta, Kartik et al. (2018). “Improving CNN-RNN hybrid networks for handwriting recognition”. In: vol. 2018-Augus, pp. 80–85. ISBN: 9781538658758. DOI: 10.1109/ICFHR-2018.2018.00023.

Fan, Yin et al. (2016). “Video-Based emotion recognition using CNN-RNN and C3D hybrid networks”. In: pp. 445–450. ISBN: 9781450345569. DOI: 10.1145/2993148.2997632.

Filonov, Pavel, Andrey Lavrentyev, and Artem Vorontsov (Dec. 2016). “Multivariate Industrial Time Series with Cyber-Attack Simulation: Fault Detection Using an LSTM-based Predictive Data Model”. In.

Foorthuis, Ralph (Oct. 2021). “On the nature and types of anomalies: a review of deviations in data”. In: *International Journal of Data Science and Analytics* 12 (4). ISSN: 2364-415X. DOI: 10.1007/s41060-021-00265-1.

Gers, Felix A., Jürgen Schmidhuber, and Fred Cummins (2000). “Learning to forget: Continual prediction with LSTM”. In: *Neural Computation* 12 (10), pp. 2451–2471. ISSN: 08997667. DOI: 10.1162/089976600300015015.

Google (2021). *ROC and AUC.* URL: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

Graves, Alex, Santiago Fernández, and Jürgen Schmidhuber (2005). “Bidirectional LSTM networks for improved phoneme classification and recognition”. In: vol. 3697 LNCS. DOI: 10.1007/11550907_126.

Hauskrecht, Milos et al. (2007). “Evidence-based anomaly detection in clinical domains.” In: *AMIA ... Annual Symposium proceedings / AMIA Symposium. AMIA Symposium*, pp. 319–323. ISSN: 15594076.

Hochreiter, Sepp and Jürgen Schmidhuber (Nov. 1997). “Long Short-Term Memory”. In: *Neural Computation* 9 (8), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://direct.mit.edu/neco/article/9/8/1735-1780/6109.

Hodge, Victoria J. and Jim Austin (2004). *A survey of outlier detection methodologies.* DOI: 10.1023/B:AIRE.0000045502.10941.a9.

Çiçek, Özgün et al. (2016). "3D U-net: Learning dense volumetric segmentation from sparse annotation". In: vol. 9901 LNCS. DOI: 10.1007/978-3-319-46723-8_49.

Katanforoosh, Kian, Daniel Kunin, and Jiaju Ma (2019). *Parameter optimization in neural networks*. URL: https://www.deeplearning.ai/ai-notes/optimization/.

Krishnan, Bipin (Nov. 2019). *WHEN and WHY are batches used in machine learning ?* URL: https://medium.com/analytics-vidhya/when-and-why-are-batches-used-in-machine-learning-acda4eb00763.

LeCun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86 (11), pp. 2278–2323. ISSN: 00189219. DOI: 10.1109/5.726791.

Lendave, Vijaysinh (Aug. 2021). *LSTM Vs GRU in Recurrent Neural Network: A Comparative Study*. URL: https://analyticsindiamag.com/lstm-vs-gru-in-recurrent-neural-network-a-comparative-study/.

Mais, Jason (May 2002). *Spectrum Analysis*. SKF USA Inc., pp. 1–32. URL: file:///C:/Users/Kevin/AppData/Local/Temp/0901d1968024acef-CM5118-EN-Spectrum-Analysis_tcm_12-113997.pdf.

Malhotra, Pankaj et al. (2015). "Long Short Term Memory networks for anomaly detection in time series". In: pp. 89–94. ISBN: 9782875870148.

Munir, Mohsin et al. (2019). "DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series". In: *IEEE Access* 7, pp. 1991–2005. ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2886457.

Ord, Keith (1996). "Outliers in statistical data : V. Barnett and T. Lewis, 1994, 3rd edition, (John Wiley and Sons, Chichester), 584 pp., [UK pound]55.00, ISBN 0-471-93094-6". In: *International Journal of Forecasting* 12 (1).

Phi, Michael (Sept. 2018). *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. URL: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: vol. 9351, pp. 234–241. ISBN: 9783319245737. DOI: 10.1007/978-3-319-24574-4_28.

Rosenblatt, F. (1958). "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65 (6). ISSN: 1939-1471. DOI: 10.1037/h0042519.

Stureborg, Rich (Jan. 2019). *Conv Nets for dummies*. URL: https://towardsdatascience.com/conv-nets-for-dummies-a-bottom-up-approach-c1b754fb14d6.

Thabtah, Fadi et al. (2020). "Data imbalance in classification: Experimental evaluation". In: *Information Sciences* 513, pp. 429–441. ISSN: 00200255. DOI: 10.1016/j.ins.2019.11.004.

Verner, Alexander (2019). "LSTM Networks for Detection and Classification of Anomalies in Raw Sensor Data". Nova Southeastern University, pp. 1–131. URL: https://nsuworks.nova.edu/gscis_etd/1074..

Wen, Tailai and Roy Keyes (2019). *Time Series Anomaly Detection Using Convolutional Neural Networks and Transfer Learning*.

Wu, Renjie and Eamonn J. Keogh (2020). "Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress". In.

Zheng, Yi et al. (2014). "Time series classification using multi-channels deep convolutional neural networks". In: vol. 8485 LNCS, pp. 298–310. ISBN: 9783319080093. DOI: 10.1007/978-3-319-08010-9_33.