DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

# Energy-based Condition Monitoring for Industry 4.0

Leander Lauenburg

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

# Energy-based Condition Monitoring for Industry 4.0

# Energiebasierte Zustandsueberwachung fuer Industrie 4.0

| | |
|---|---|
| Author: | Leander Lauenburg |
| Supervisor: | Prof. Dr. Alexander Pretschner |
| Advisor: | Markus Duchon, PhD |
| Submission Date: | September 29th, 2018 |

# Affirmation

I confirm that this bachelor's thesis is my own work and that I have documented materials used.

Date: _____ Signature: _____

# Acknowledgments

First and foremost, I would like to thank Dr. Markus Duchon who was my supervisor for this thesis. I would like to express my sincere gratitude to you for sharing your expertise and advice. I also would like to thank the research institute fortiss GmbH for granting me access to its resources. Further, I would like to thank Hendrik Walzel and Stefan Profanter who helped me in recording the data used in this work.

# Abstract

Condition Monitoring (CM) is essential not only for recognizing system failure quickly and remotely, but also for optimizing energy usage in system processes. However, current systems for CM are highly domain-specific. This makes them expensive and non-reusable. This thesis explores the idea of using an Energy Monitoring System coupled with Convolutional Neural Network models to perform domain-agnostic, easily configurable Condition Monitoring. To this end, the energy consumption of two systems, a miniature industrial road and a robotic arm, is recorded for different system states and different models are trained on this data. A classifier model is used to recognize the states of the systems, and different auto-encoder models are used to detect abnormal system behavior. The classifiers perform exceptionally well reaching accuracies of over 99%, whereas the auto-encoders show promising state reconstruction and measurement error detection but are yet to be tested on the accuracy of detecting actual anomalous system behavior. Although work remains to be done, in particular with respect to enhancing the system for Predictive Maintenance (PM), the results of this thesis emphasize the potential for CM using energy based data.

# Contents

# List of Figures

# List of Tables

# List of Equations

# List Of Abbreviations

| | |
|---|---|
| **AC** | Alternating Current |
| **AI** | Artificial Intelligence |
| **AN** | Artificial Neuron |
| **ANN** | Artificial Neural Network |
| **Bp** | Backpropagation |
| **CL** | Convolutional Layer |
| **CM** | Condition Monitoring |
| **CNN** | Convolution Neural Network |
| **CPS** | Cyber-Physical System |
| **DB** | Data Base |
| **DC** | Direct Current |
| **DL** | Deep Learning |
| **EMS** | Energy Monitoring System |
| **FE** | Feature Engineering |
| **GD** | Gradient Decent |
| **HTTP** | Hypertext Transfer Protocol |
| **JSON** | JavaScript Object Notation |
| **KNN** | K-Nearest Neighbors |
| **ML** | Machine Learning |
| **M2M** | Machine to Machine |
| **PM** | Predictive Maintenance |
| **PL** | Pooling Layer |
| **SVM** | Support Vector Machine |
| **SPI** | Serial Peripheral Interface |
| **THD** | Total Harmonic Distortion |

# 1 Background and Motivation

## 1.1 Motivation

The awareness for energy consumption is changing constantly. This has confronted a wide range of industries with the task of becoming more energy efficient and appearing environmentally friendly.
Many large companies are already taking great efforts in this direction. One of these efforts is the detailed monitoring of energy consumption. However, nowadays this is mainly driven by the goal to fulfill ISO standards. These only require monitoring the usage and companies may then additionally do simple cost and activity accounting. However, actually using the accumulated data in real-time is being lost to the companies.

The ASCI Group, a subdivision of the fortiss GmbH [1], is a renowned research team coordinating and pushing different international projects with the goal of achieving a certain level of autonomy through smart grids. An important step for the realization of smart grids is the incorporation of mid-sized and small-sized companies as intelligent nodes into the power grid system.
But due to high costs, such energy management has so far been mainly reserved to large corporations.
Therefore, an affordable Energy Monitoring System (EMS) coupled with automation components are being developed and tested by the ASCI group in strong cooperation with other leading research organizations and industry partners.
This thesis is motivated by the idea of exploring the potential of high-resolution, real-time electricity data that is already being accumulated in the industry. This is enhanced by a simultaneous, investigation concerning the capability and limits of the in-house EMS developed by the ASCI Group in strong cooperation with industry partners.
To accomplish this goal, the accumulated data is used in CM and Predictive Maintenance (PM).
Additionally, the results of this thesis are intended to advance Machine to Machine (M2M) communication. In a fully automated factory, consisting only of intelligent devices, the machines do not only need the necessary intelligence but also meaningful data to be processed by this intelligence.
This entails that the machines have to be consistently informed about their surroundings, allowing them to dynamically adjust to change in a running process. The machines thereby evolve from the observed to the observer.

[1] http://fortiss.org

### Problem Statement

The complexity of the technical world is ever-increasing. An increase in complexity often translates into an increase in risk of system failure. The failure of a single component or subsystem cannot only paralyze the system but result in a chain of events damaging other parts of it or even the entire system. To minimize the occurrence of such incidents, companies have developed various methods for CM as well as PM, with the goal of increasing the planning horizon to better schedule downtime and maintenance, and identify error sources. But so far, these methods provide a low level of re-usability in other domains, making them highly time consuming and expensive. For this reason, there is a need for systems for CM and PM with multi-domain capabilities. The usage of high-resolution energy data (e.g. power consumption) for CM has the potential to create such a multi-domain solution by reducing the input space to mere energy based variables - these are, of course, domain agnostic. This approach additionally provides a non-intrusive way for data collection with a lightweight, modular and easy to install EMS.

### Research Question

In accordance with the problem statement the following research questions have been raised:

- Can a high-resolution EMS serve as a multi-domain solution for CM and PM?
- To what degree can such a model be generalized as to be easily applicable to a variety of systems?
- Does the system developed at fortiss GmbH meet the necessary requirements?
- Which parameter of the monitored energy data has the most significant impact on the model's performance as it relates to optimization?

## 1.2 Approach

Solving the problem statements starts with a detailed overview of the necessary theoretical background and an extensive literature review thereof. A programmatic enhancement of the EMS to fit the task of CM follows this. Additionally, an appropriate Data Base (DB) structure is set-up to handle the resulting data stream. In the next step, data based on defined use cases is recorded, cleaned, and labeled. It is then fed into various deep learning models to train them on recognizing different states of a machine. The models are then extended for PM using data discontinuity and error recognition.

## 1.3 Structure of the Thesis

Chapter 2 summarizes the current state of research and tools that lay the groundwork for this thesis. Chapter 3 introduces the hardware of the EMS, outlines the use cases, and presents the resulting data sets. This is followed by an overview over the methodologies in Chapter 4. Then, in Chapter 5 the model configurations and implementations are presented, followed by an explanation of the training processes and a presentation of the results. Finally,

Chapter 6 summarizes and analyses the results, discusses their meaning with respect to the initially posed research questions, and draws a conclusion. The Appendix consists of detailed overview over important concepts from the fields of electrical engineering, data science, and computer science as they relate to the concepts and ideas of CNNs.

# 2 State Of The Art

This chapter provides an overview of the current state of research that relates to this thesis, and the tools used. It is split into three sections:

- energy-based CM
- ML based approaches for CM with anomaly detection
- tooling for this thesis

## 2.1 Energy Based Condition Monitoring

In in their paper "Condition monitoring towards energy-efficient manufacturing: a review" Zhou et al. [1] discuss techniques and intelligent sensors for CM. For energy-based CM, they examine the idea of mapping the characteristic energy-consumption of machines to the energy-usage profile of the entire plant to "reduce idle power usage and increase energy efficiency"[1, Zhou et al., 2017, p.4]. Although their results and techniques are interesting for identifying excess supply and giving first insights into the usefulness of energy data, their approach is not applicable for continues CM. On the other hand, Marais, Van Schoor and Uren [2] argue for energy-based CM for entire process plants in their paper "An Energy-based approach to condition monitoring of industrial processes":

> "The use of energy mitigates the multi-domain monitoring problem, and could also provide a hierarchical view of the process plant's condition."[2, Marais et al., 2015, p.1]

They see the greatest potential of energy-based CM in replacing or enhancing methods of time-based, risked-based, and predictive maintenance. By focusing on the energy variables (which includes not just electrical, but all power based, measurable quantities such as heat flow), they list the following benefits:

- The input space becomes one-dimensional
- Hierarchical monitoring is made possible
- The approach is inherently multi-domain.

The hierarchical structure of energy-based CM is depicted in 2.1. Diagnostic algorithms and prognostic algorithms are often chosen to be in the ML domain, due to the domain's power in pattern recognition and state classification.

However, Zhou et al. outline the limitation of an energy-based monitoring approach to certain types of systems[1]. For example, machines with a pneumatic elements raise the problem that their electrical energy consumption runs acyclical with respect to their use of pressurized air. This is due to an accumulator within the cycle. Zhou et al. therefore believe it to be necessary to fall back on traditional condition monitoring sensors like torque and rotational speed sensors. Maier, Schriegel, and Niggemann [3] broach the issue of ever-increasing complexity in production plants. To be able to compete, production has to evolve with the product. The complexity and variance of products are continually evolving. Accordingly, production systems are becoming "modular, [...] parameterized and comprise a growing set of sensors." [3, Maier et al., 2016, p. 1] Updating and integrating new devices into the production chain is burdensome. Additionally, unaccounted for system faults do not result in system optimization, but rather entire system reboots without analyzing the cause for error. To solve this problem, Maier et al. discuss an approach which centers around the idea of elevating every subsystem to a Cyber-Physical System (CPS), creating their virtual counterparts sometimes referred to as their "digital twin". Using insights from Big Data, the machines are then supposed to...

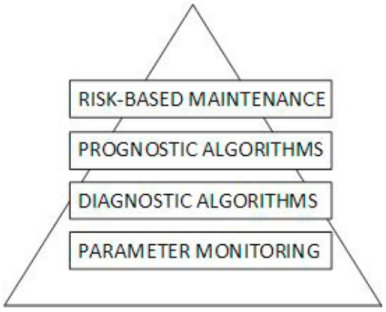



Figure 2.1: Hierarchical view of CM [2, Marias et al., 2015, p.773]

> ...detect deviations from normal workflow as an anomaly (e.g. condition monitoring), find root causes for such anomalies (e.g. diagnosis), calculate optimal parameters and predict the future system behavior (e.g. predictive maintenance, energy optimization). [3, Maier et al., 2016, p.474]

Maier et al. refer to a statement by Werner Vogels, Amazon's chief technology officer, from 2004 in which he stated that when it comes to data, more is always better:

> Every used kilowatt hour, from each produced screw to each car, even each switching of a proximity sensor and each change of a temperature sensor generates raw data that holds an enormous potential if it is stored and provided for intelligent analysis. [3, Maier et al., 2016, p.475]

However Maier et al. also see quite a few challenges in this approach. The challenges arise from acquisition, processing, and analysis of the data. A key challenge is the synchronization of from different data sources and the requirement for feature engineering stemming from the high dimensionality of the data.

## 2.2 ML based approaches for CM and anomaly detection

Marais et al.[2] praise the capability of Artificial Neural Networks (ANNs)s for pattern recognition and classification. This is underlined by the potential Maier et al.[3] see in an ML based approach for CM.

The paper "Feature Extraction and Classification of the Electric Current Signal of an Induction Motor for Condition Monitoring Purposes" gives first insights into an exclusively energy-based CM and PM approach. Gebbe et al. [4] analyze the current data of a conduction motor using a Machine Learning approach for condition-based maintenance. The data in the paper heavily relies on Feature Engineering so that the main focus is on feature extraction and classification.
Their concept is based on the fact that the most common fault of induction motors | broken rotor bars and broken rings | can be deduced from their electrical current consumption.
Gebbe et al. run several algorithms they group into three categories: neighbor-based classifiers, statistical classifiers, and rule-based classifiers. Only the Support Vector Machine (SVM) based classifier has a high accuracy of 93%. None of the other classifiers scores higher than 55%. This is likely due to the small size of the data set. Nonetheless, their results provides insights into analyzing time series data and Feature Engineering.

Shuyang Du et al. [5] analyze time series data using machine learning techniques as well. For the paper "Modeling approaches for time series forecasting and anomaly detection" they compare the performance of the three different techniques: K-Nearest Neighbors (KNN), Long Short-Term Memory based recurrent networks (LSTM), and Sequence to Sequence with Causal CNN (SeqtoSeq CNN) on forecasting the web traffic on Wikipedia webpages. [5] The Convolutional Neural Network shows the most promosing performance for their time series data, being able to recognize even low frequency patterns. They do point to one caveat, though:

> However, none of our models are able to capture spikes or outliers that arise from external sources. Enhancing the performance of the models will require augmenting our feature set from other sources such as news events and weather. [5, Shuyang Du et al., 2017,p. 5]

All in all Shuyang Du et al. raise compelling arguments for the use of CNNs.
For Maier et al. anomaly detection is at the core of CM and be an enabler for Predicitve Maintenance[3].
Kiran et al. define anomaly detection as follows:

> Anomaly detection is an unsupervised learning task where the goal is to identify abnormal patterns or motions in data that are by definition infrequent or rare events. [6, Kiran et al.]

The field of anomaly detection is dominated by ML and in particular Deep Learning (DL) approaches[6][7]. For multivariable data with more than two classes, anomaly detection is extremely data-driven [7]. Chalapathy, Menon and Chawla [8] give further insights into DL methods for anomaly detection:

> Deep learning methods for anomaly detection can be broadly classified into model architecture using autoencoders and hybrid models. [8, Chalapathy et al.]

Further, Chalapathy et al. state that "autoencoders utilize [the] magnitude of the residual vector (i.e. reconstruction error) for making anomaly assessments." [8, Chalapathy et al.]

and "hybrid models mainly use [an] autoencoder as [a] feature extractor, wherein the hidden layer representations are used as input to traditional anomaly detection algorithms... "[8, Chalapathy et al., 2018, p.1].
This matches the approaches of other papers, such as that of Kiran et al. with a stated goal of "[reviewing] the state-of-the-art deep learning based methods for video anomaly detection" [6].
In conclusion, the literature suggests autoencoder based DL approaches for anomaly detection.

## 2.3 Tooling

The preceding sections have suggest a ML approach. Considering the amount of data to be analyzed in conjunction with the research question "How strongly can such a model be generalized to be easily applicable to a variety of systems?", the insights from 2.2 suggest a DL approach. The best performance, in regard to latency, will be achieved by using a low-level language like C++ [9]. However, Wes McKenny [9] argues that for most applications the developers time is more valuable than the CPU time and recommends the use of the popular and dynamic programming language Python. Wes McKenny underlines this recommendation by pointing out the following benefits [9, Wes McKinney, 2012, p.2]:

- "Large and active scientific computing community"
- "An improved library support"
- "Pythons ease of integrating C, C++, and FORTRAN code"
- "Pythons suitability for not only research and prototyping but also building the production systems"

The extraordinary progress in AI and ML has resulted in some powerful frameworks like *TensorFlow* making DL easily accessible to a broader community of users.
TensorFlow is a library that simplifies the use of distributed numerical computation considerably allowing to compute large ANNs efficiently. [10] TensorFlow was created at Google and supports many of their large-scale Machine Learning applications[10]. Further simplifications to TensorFlow exist enabling rapid prototyping and experimenting. Keras is one such framework. It is a high-level API using TensorFlow, Microsoft Cognitive Toolkit or Theano (frameworks for DL) as a backend for its networks. It is an open source neural network library written in Python that provides an easy way to experiment and design DL models and its lead developer François Chollet says the following:

> "Keras, one of the most popular and fastest-growing deep- learning frameworks, is widely recommended as the best tool to get started with deep learning." [11, François Chollet,2018, p.xvi]

## 2.4 Chapter Summary

This chapter discusses the current state of research as it regards to energy-based Condition Monitoring and anomaly detection and the approaches most widely used. The limitations are also highlighted. As a result, energy-based monitoring is suggested to be used in conjunction with Convolutional Neural Networks. This is followed by a reasoning of the tooling, specifically the Python programming language and the Keras library for Artificial Neural Networks, used in this thesis which is based on best-practices and recommendations found in literature.

# 3 Hardware, Use Cases, and Data Sets

This chapter discusses the Energy Monitoring System developed by the ASCI Group in cooperation with industry partners. Then, the different use-cases for which data is recorded are presented along with the actual data.

## 3.1 Energy Monitoring System

The Energy Monitoring System by the ASCI Group which is used for this thesis is shown in 3.1.

### 3.1.1 Components

The system consists of a Raspberry Pi as a master unit and one to five B-3U4I-400 bricks as slave units. The B-3U4I-400 module was developed and manufactured by the IMACS GmbH [1] and offers three voltage inputs and three current inputs. Internally, the units calculate the effective voltage, effective current, power factor/phase shift, effective power, and harmonic distortion.[12]

[1] http://www.imacs-gmbh.de

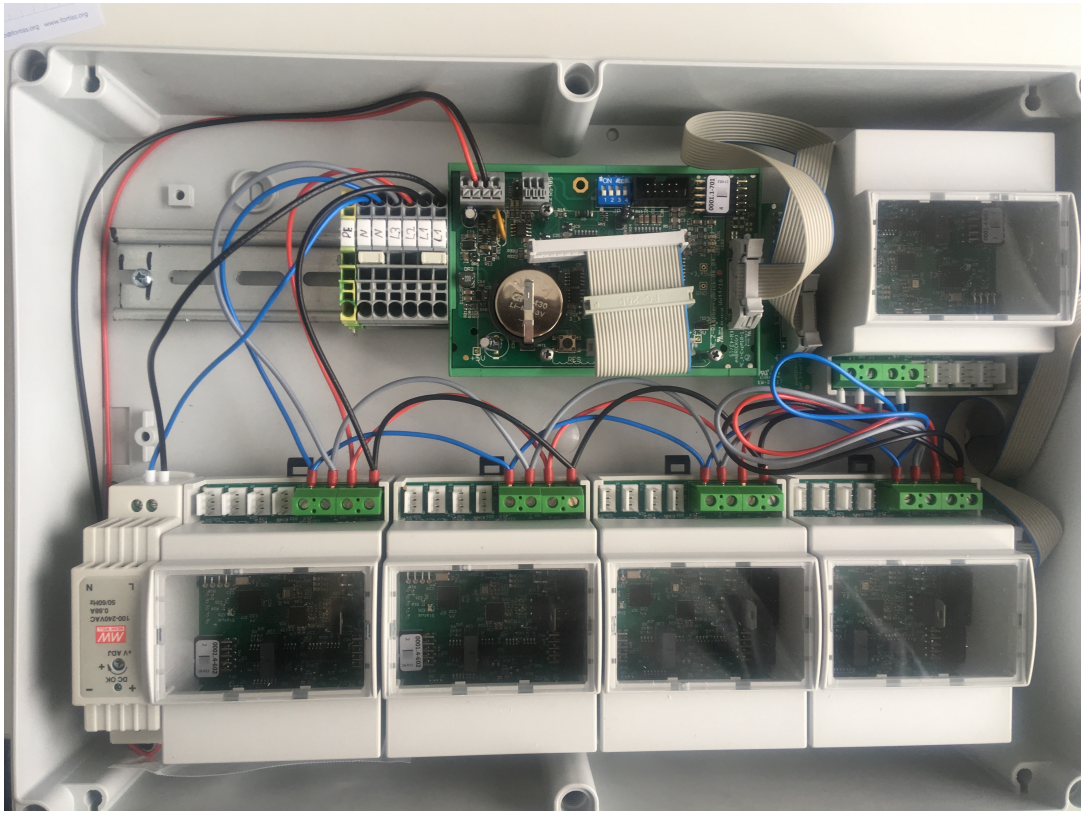


Figure 3.1: The EMS developed by the ASCI Group.
Each B-3U4I-400 brick is surrounded by a case. The Raspberry Pi (not shown) connects to the board not surrounded by a case.

A Serial Peripheral Interface (SPI) handles the communication between the master and the slaves.
A SPI is the realization of a full duplex, serial communication protocol. This means that two modules communicate through bit-wise, simultaneous, and bi-directional data streams. Furthermore, SPI is a synchronous protocol in which all participants share the same clock signal. Figure 3.2 depicts a SPI set-up with one master and three slaves. The slaves connect to the master via a serial clock signal (SCLK), a master-out-slave-in line (MOSI), a master-in-slave-out line (MISO) and a slave-select for every slave (CS1, CS2, CS3). This is necessary since the slaves are connected in series and concurrent activity by more than one slave would result in faulty information.

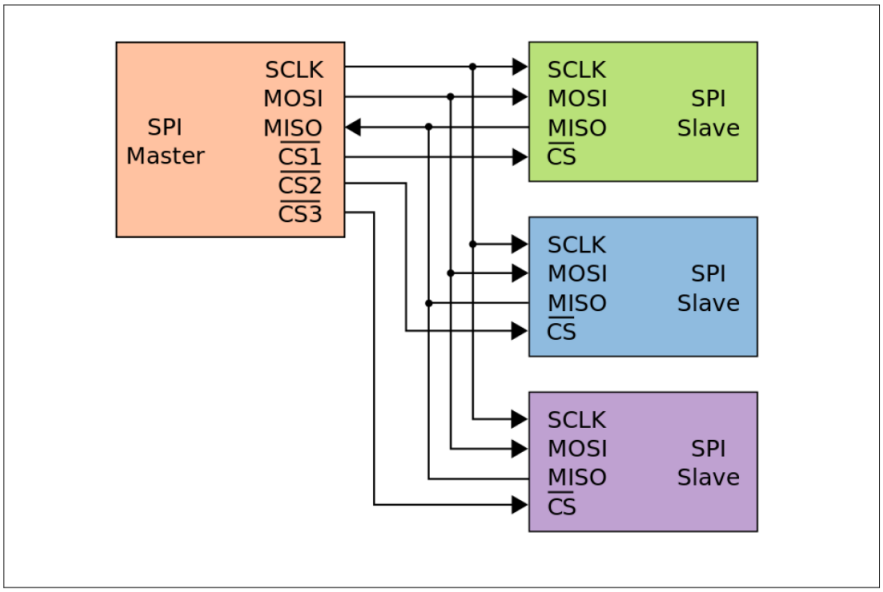



Figure 3.2: Serial Peripheral Interface with three slaves. The bar on top of the CS channels indicates that the slave-selects are active low.

[13, N. Tollervey, 2017, p.106]

### 3.1.2 Limitations

The master is only able to select and communicate with one slave at a time. Switching the line of communication by activating another slave takes around ten ms. Thus, for five slaves, the master needs 50 ms to complete "one round". However, if the slaves gather data slower than these 50 ms, the master is limited by this. On the other hand, if the slaves refresh at a quicker rate, the system is unable to process the accumulated data at the same speed as it is produced.
Communication with the back-end presents a further bottleneck. The EMS's software runs two threads. One thread handles the communication between the master and the slaves and the other the communication between the master and the back-end system. Both threads share access to a queue that is filled with the data from the bricks. The first thread pushes new data into the queue whenever it gets new data from this brick. The second thread loops through the queue processing one entry at a time and converting it to JavaScript Object Notation (JSON) before sending it to the back-end via an Hypertext Transfer Protocol (HTTP) POST. Extensive testing has shown that the highest frequency of POST requests the system is able to handle is around $\frac{1}{5}s = 200ms$.
The current implementation of the SPI is written in Java. Since Java isn't real-time, this results in the EMS not being real-time capable either. Additionally, the time stamp of the data is not added by the bricks themselves, but rather right before the POST request. Naturally, this leads to latency between the actual event and the recorded data of this event.

## 3.2 Use-Cases

The data used for training the networks in this thesis stem from the use-cases presented hereinafter.

### 3.2.1 Industrial Road

The first use-case is an industrial road for which the DL models are used to determine which section of the road is active and in what state the current process is. When combined with analytic methods the models recognize aberrations from the general process. The industrial road depicted in figure 3.3 is a miniature demonstrator with components made by festo GmbH [2]. The demonstrator consists of four sections each in charge of one of the following tasks (from left to right):

1. Retrieval of components from magazine
2. Sorting of components by height
3. Fixating lids to components
4. Sorting of components by material

The sections, in turn, consists of the following elements:

1. Magazine, rotating arm
2. Lift, conveyor belt, sensor for measuring the component height
3. Conveyor belt, gate, mechanism for attaching lids
4. Conveyor belt, sensor for material detection, sorting mechanism

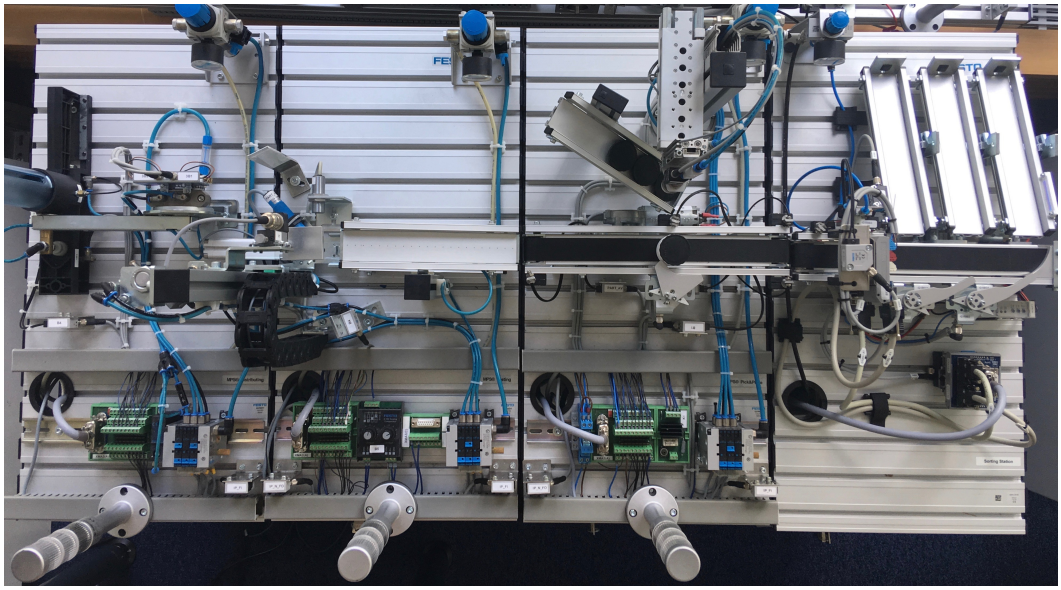


Figure 3.3: Festo demonstrator: Industrial road

[2] https://www.festo.com/

One of the main challenges for Condition Monitoring based on characteristic energy consumption is disaggregation: when monitoring the main power supply of a complex system with many independently operating consumers, the number of unique consumption patterns quickly becomes limitless. However, an industrial road represents a cyclic process. The possible number of consumption patterns is therefore significantly reduced.

### 3.2.2 Robot Movement

The second use-case is a robotic arm. Here, the EMS learns to distinguish between different movements of the arm. The EMS was used to record the characteristic consumption patterns for distinct movements of the arm of the e.Do robot depicted in Figure 3.4. The e.Do arm has six degrees of freedom. Since the robot has a power supply with an AC/DC converter which may distort the recorded data, both the AC and the DC consumption characteristics were recorded. For this reason the EMS was enhanced to not only work for Alternating Current (AC) but also for Direct Current (DC).


[e.Do, accessed: 18.08.2018, https://edo.cloud/edo-6-axes/]

Figure 3.4: e.Do

## 3.3 Data Sets

### 3.3.1 Data Set of the industrial road

In this thesis, the power consumption of the entire industrial road, as well as that for each of the road's sectors is recorded. The combination of these recordings makes it possible to label the accumulated power consumption data with the sectors that are active for any given time. This data is then used to teach the ANN with the goal that it learns to recognize which sectors are active by monitoring the accumulated power consumption.
The data was recorded for a period of 35 minutes in which each section underwent 120 cycles. Figure 3.5 depicts the power consumption of the single sections and the road as a whole. Figure 3.6 depicts the voltage level of the individual sections and the road as a whole. Due to outliers that distorted the data and a high average voltage with low variance, the level was cleaned.
Samples depicting the current, phase shift and Total Harmonic Distortion (THD) are attached in the appendix.

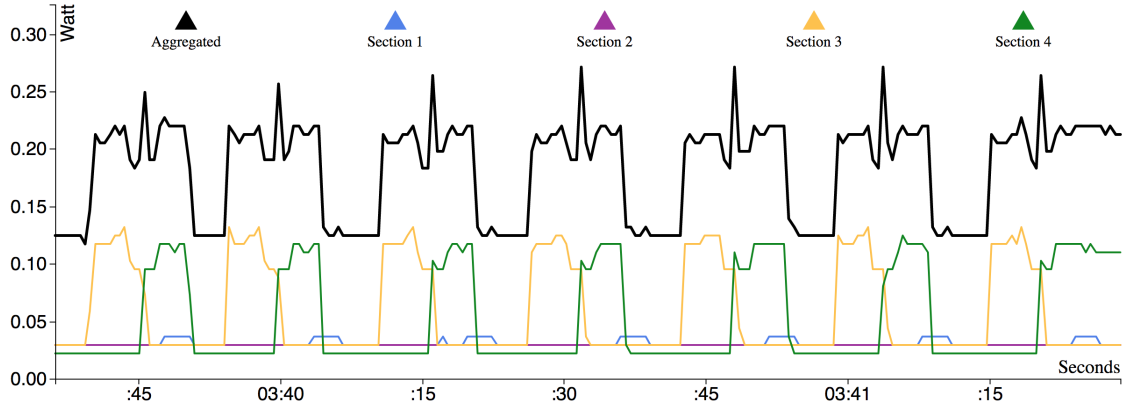



Figure 3.5: Power consumption of the whole demonstrator and the single sections. Aggregated: black, section 1: blue, section 2: purple, section 3: yellow, section 4: green

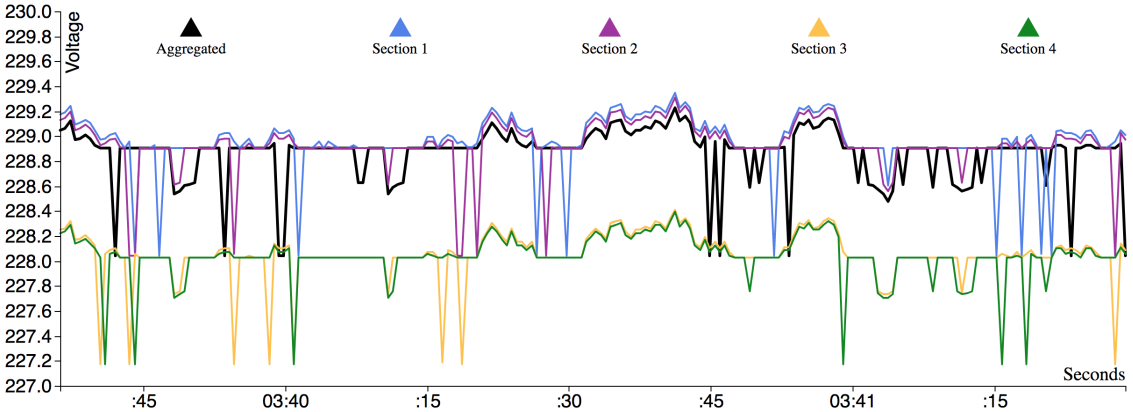



Figure 3.6: Voltage level of the whole demonstrator and the single sections. Aggregated: black, section 1: blue, section 2: purple, section 3: yellow, section 4: green

### 3.3.2 Data recordings of the movements of a robot

First, the characteristic consumption for the individual motors was recorded. Second, the characteristic consumption of three distinct movements each using two of the six engines was recorded. In a last session, the characteristic consumption of two distinct movements each using three of the six motors was recorded. Depending on its orientation every motor either performed a 90°bow or 180°rotation and repeated these movements with and without a weight for 50 times each.

Figure 3.7 depicts the phase shift introduced by the single motors.
Figure 3.8 depicts the current flow through the e.Do's single motors.

Samples depicting the power, voltage, and THD are attached to the appendix.

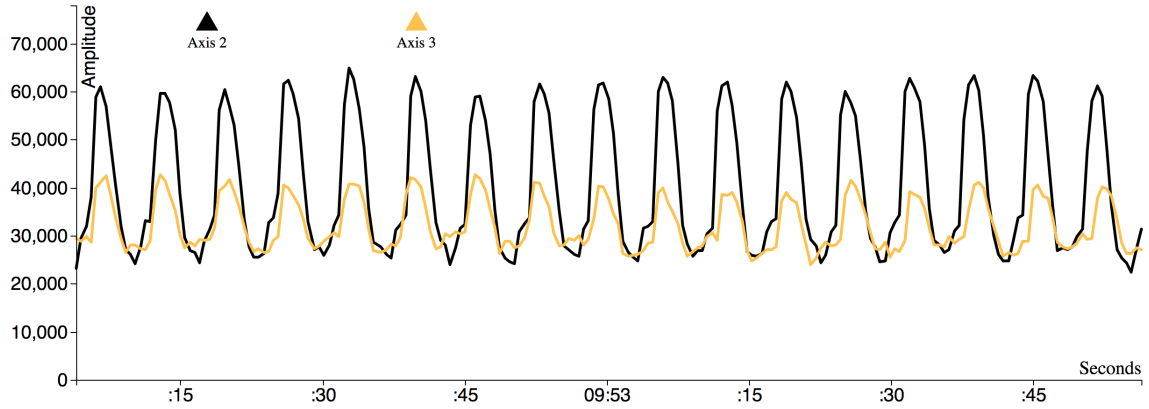



Figure 3.7: Phase shift introduced by second motor from the bottom and the third motor above it. Second motor: black, third motor: yellow

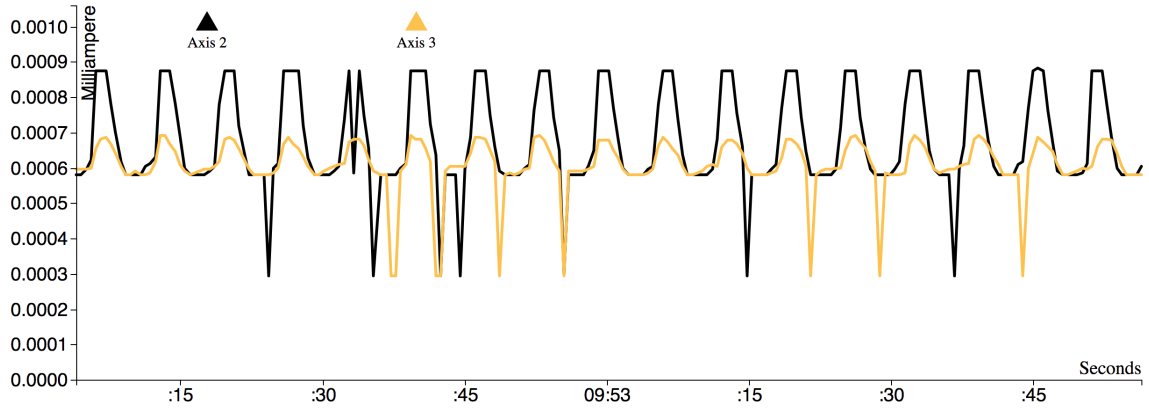



Figure 3.8: Current level of the second motor from the bottom and the third motor above it. Second motor: black, third motor: yellow

## 3.4 Chapter Summary

This chapter starts off by explaining the components of the Energy Monitoring System developed by the ASCI Group in cooperation with their industry partners. The hardware side of the system, the measurement bricks and computer responsible for running the software,

and the software are described in full detail. Additionally, the limitations of the system are discussed. This is followed by a description of the use-cases that the system monitors: a miniature industrial road and a robotic arm. Finally, the approach of recording the data is explained per use-case and the resulting data sets are presented.

# 4 Data Analysis and ANN Tuning

The first part of this chapter explains the choice of the model. The second part then describes the methods used for the preprocessing of the recorded data sets such as cleaning, sampling and labeling. Finally, the architecture and parameter tuning of one classifier and two auto-encoders is explained.

## 4.1 Model choice

The choice of the model is motivated by two ideas: first, to provide a multi-domain monitoring system powered by large amounts of multi-variable energy data, and second, to develop a system that can offer additional insights into the running system's processes. As suggested by Gebbe et al.[4], a ML approach seems appropriate, in which rules are not explicitly set but learned from the statistical structure of the input data. However, traditional ML approaches need Feature Engineering (FE) which makes the performance not only dependent on the operator's domain knowledge but also reduces the flexibility and reusability of the system.[10] Since the data is time series data, an additional step of discretization as shown by Gebbe et al. [4] makes FE even more cumbersome. Hence, omitting FE is key.
Marais et al.[2] suggest a DL approach for artificial neural network confronted with these challenges. This is further corroborated by the books "Hands-On Machine Learning with Scikit-Learn and TensorFlow" [10] and "Deep Learning with Python"[11]. The decision to use CNNs, a specific DL model is motivated by Shuyang Du et al. (2.2) as well as their proved performance in the field of voice recognition. The decision is also based on the increasing usage of CNN for time series applications within the data science community. [11]
The models are written in Python 3 with the help of the Keras library [1]. This library provides a user-friendly, yet powerful framework for creating neural networks and allows for fast experimentation and prototyping.

## 4.2 Data Analysis

Preprocessing data often represents the most time-consuming part of ML. This process can consist of multiple steps.

### 4.2.1 Data Cleaning

As explained in 7.1.3 of the appendix
Discontinuity during the measurement process for the training data can lead to outlier points. This is able to corrupt the DL models.

[1] https://keras.io

Therefore, it is crucial to find and remove defective data before training. Visualizing the data is one approach to recognizing discontinuities in the general patterns. As an example for this, Figure 4.1 depicts a sample of the recorded power consumption of one of e.Do's motors for a single movement that is repeated 50 times.
One can deduce several error sources from this plot. First, the motor needs some time to start before continuing into constant operation. Second, once in constant operation there exists a minimum and maximum and any values outside of these limits indicate a measurement error. Thus, the data-set needs to be truncated to exclude the data from the start-up of the motor and the wind-down and deal with the outliers. The outliers are dealt with by setting their value to the average value of the data set.

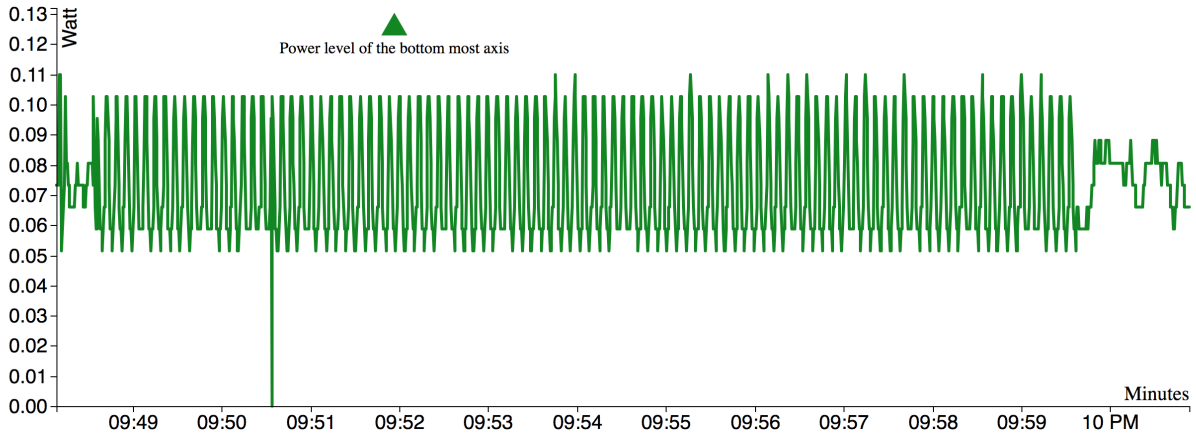


Figure 4.1: Example for discontinuity within the recorded data

### 4.2.2 Sampling the data

Since the data is time series data, it is necessary to split the data into appropriate chunks which can be fed to the DL models. To do this, we use a window function whose size and sampling rate needs to be determined.
The size of the window function depends on the data as well as the application. For long-term trends large window functions are appropriate, and vice versa for patterns with a higher frequency. Thus, for deterioration and predictive maintenance a large window function is needed, whereas live analysis of the data requires a small size. Note that for patterns with little noise and high stability, a window size is advisable with which neither the size is divisible without remainder by the signal frequency, nor the signal frequency is divisible by the size without remainder. Naturally, the larger amount of data resulting from smaller windows requires longer computational time. [5]
The sampling rate of the function depends more on the data at hand than on the specific application. Generally, the more data that is feed into a DL model, the more flexible and robust the model becomes. Thus, if the sample rate is lower than the signal frequency, the resulting model will be less robust. However, increasing the sample rates can lead to overfitting.
Due to the nature of the use-cases the windows for the industrial road are rather larger, the windows for the robotic arm as small as possible. Giving live feedback of the robot arm's movements and detecting anomalies needs to be as close to real-time as possible, while the activity of an entire section on an industrial road fills a broader time frame. For both cases

the sampling rate is $\frac{1}{0.5 \cdot w} Hz$ where $w$ denotes the window size.

### 4.2.3 Labeling the Data

The Machine Learning model in this thesis is a supervised model. This means that the model is given the classification scheme rather than finding such a classification through statistical considerations as is done for unsupervised learning | often used in data mining.[10] For the robot arm the labeling is straightforward: each distinct movement gets its own label. For the industrial road this process is more complicated because the aggregated data needs to be labeled so that the model learns to disaggregate it.

Each B-3U4I-400 brick can monitor three measuring points. In total, five measurements are needed: four individual sections, and the entire road. Thus, two bricks are needed. Each brick has an id and is mapped to a specific measuring point as shown in Table 4.1. The table also includes a timestamp of the mapping and a 'readingid' specifying the id of a set of bricks.

| timestamp | readingid | idclip1 | idclip2 | idclip3 | description1 | description2 | description3 |
|---|---|---|---|---|---|---|---|
| 1530711335737 | 10 | 100 | 200 | 300 | Aggregated | Section1 | Section2 |
| 1530711335737 | 10 | 400 | 500 | 600 | Section3 | Section4 | Null |

Table 4.1: Data base represetion of the mapping of the clamps

Table 4.2 gives an example to how the data is stored. The 'variableid' is a combination of the clip's id (the first digit) and the id (the last digit) of the measured variable. There are five variables: power, phase shift, current, voltage, and THD.

| timestamp | variableid | readingid | value |
|---|---|---|---|
| 1530708783291 | 101 | 10 | 16 |
| 1530708783291 | 201 | 10 | 4 |
| 1530708783291 | 301 | 10 | 4 |
| 1530708783291 | 102 | 10 | 43033 |
| 1530708783291 | 202 | 10 | 43812 |
| 1530708783291 | 302 | 10 | 42189 |
| 1530708783291 | 103 | 10 | 89 |
| 1530708783291 | 203 | 10 | 81 |
| 1530708783291 | 303 | 10 | 81 |
| 1530708783291 | 104 | 10 | 16683 |
| 1530708783291 | 204 | 10 | 16703 |
| 1530708783291 | 304 | 10 | 16703 |
| 1530708783291 | 105 | 10 | 04 |
| 1530708783291 | 205 | 10 | 54 |
| 1530708783291 | 305 | 10 | 38 |
| 1530708783291 | 401 | 10 | 4 |

Table 4.2: Snippet of the data stored in the data base

Analysis of the recorded data has resulted in the insight, that power consumption is the most useful characteristic. Therefore, this is used for labeling. Table 4.3 depicts an excerpt from the measured power consumption data. The ‘aggregated‘ column is the total consumption, ‘ti‘ denotes brick i, and the appended ‘01‘ stands for the power consumption variable. Every state of the system, e.g. section 1 & 3 are running, or none of the sections are running, gets its own label: in this case 13 and 0, respectively. A section is determined to be running whenever its power consumption crosses a certain threshold. The result of this labeling is the ‘label‘ column in the aforementioned table. Using the ‘timestamp‘ and ‘readingid‘ this label can now be appended to the entire aggregated data set (consisting of the five variables).

| timestamp | readingid | aggregated | t1_01 | t2_01 | t3_01 | t4_01 | label |
|---|---|---|---|---|---|---|---|
| 1530711364082 | 10 | 18 | 5 | 4 | 4 | 3 | 1 |
| 1530711364613 | 10 | 17 | 5 | 4 | 4 | 3 | 1 |
| 1530711365142 | 10 | 18 | 5 | 4 | 4 | 3 | 1 |
| 1530711365672 | 10 | 17 | 4 | 4 | 4 | 3 | 0 |
| 1530711366205 | 10 | 17 | 4 | 4 | 4 | 3 | 0 |
| 1530711366734 | 10 | 17 | 4 | 4 | 4 | 3 | 0 |
| 1530711367264 | 10 | 17 | 4 | 4 | 4 | 3 | 0 |
| 1530711367799 | 10 | 17 | 4 | 4 | 4 | 3 | 0 |
| 1530711368329 | 10 | 17 | 4 | 4 | 4 | 3 | 0 |
| 1530711368853 | 10 | 17 | 4 | 4 | 4 | 3 | 0 |
| 1530711374657 | 10 | 28 | 4 | 4 | 5 | 13 | 4 |
| 1530711375189 | 10 | 27 | 4 | 4 | 4 | 13 | 4 |
| 1530711375712 | 10 | 29 | 4 | 4 | 4 | 16 | 4 |
| 1530711376238 | 10 | 31 | 4 | 4 | 4 | 17 | 4 |

Table 4.3: Table containing the aggregated power and the power of each section plus an extra column denoting the label for each timestep

As can be seen in 3.5 section two does not have any power consumption. This is because all its parts are pneumatic, a problem discussed in Chapter 2. For this reason, the activity recognition only works for sections one, three, and four, resulting in eight labels.

Table 4.4 shows the label distribution for the entire dataset covering 4151 individual data points.

| | label | count |
|---|---|---|
| No section active | 0 | 1148 |
| Section one active | 1 | 391 |
| Section three active | 3 | 1291 |
| Section four active | 4 | 707 |
| Section one and three active | 13 | 3 |
| Section one and four active | 14 | 459 |
| Section three and four active | 34 | 142 |
| Section one, tree and four active | 134 | 10 |

Table 4.4: Occurrence of the different labels in the dataset of the festo demonstrator

Note the low counts 3 and 10 for labels 13 and 134, respectively. These counts are far fewer than the 120 cycles of the industrial road that were recorded. This means that sections 1 & 3 and section 1, 2, and 3 are never decisively simultaneously active. However, this may indicate that their start-up and wind-down times are close to each other resulting in occasional overlapping.
The data is then sampled and the sample is given the most frequently occurring label of the

individual data points.

### 4.2.4 Measuring Direct Current

For the e.Do both AC and DC are taken into account. The e.Do draws AC from the power socket but has an AC/DC converter and runs itself on DC. Since it is likely that the converter obscures the consumption characteristics, it is of interest to measure the DC consumption. However, the measurement clamps of the EMS can only measure AC. Nonetheless, because AC is DC changing its direction with a certain frequency, it is possible to use the clamps to measure changes in the electricity of DC. Fortunately, the information of these electrical spikes can be used for DL models, as they learn statistics and not physics, as can be seen in figure 4.2. The graph at the bottom depict the DC measurements of the second motor from the bottom and the third motor above it. Indeed, these appear to be more characteristic than the corresponding AC measurements.

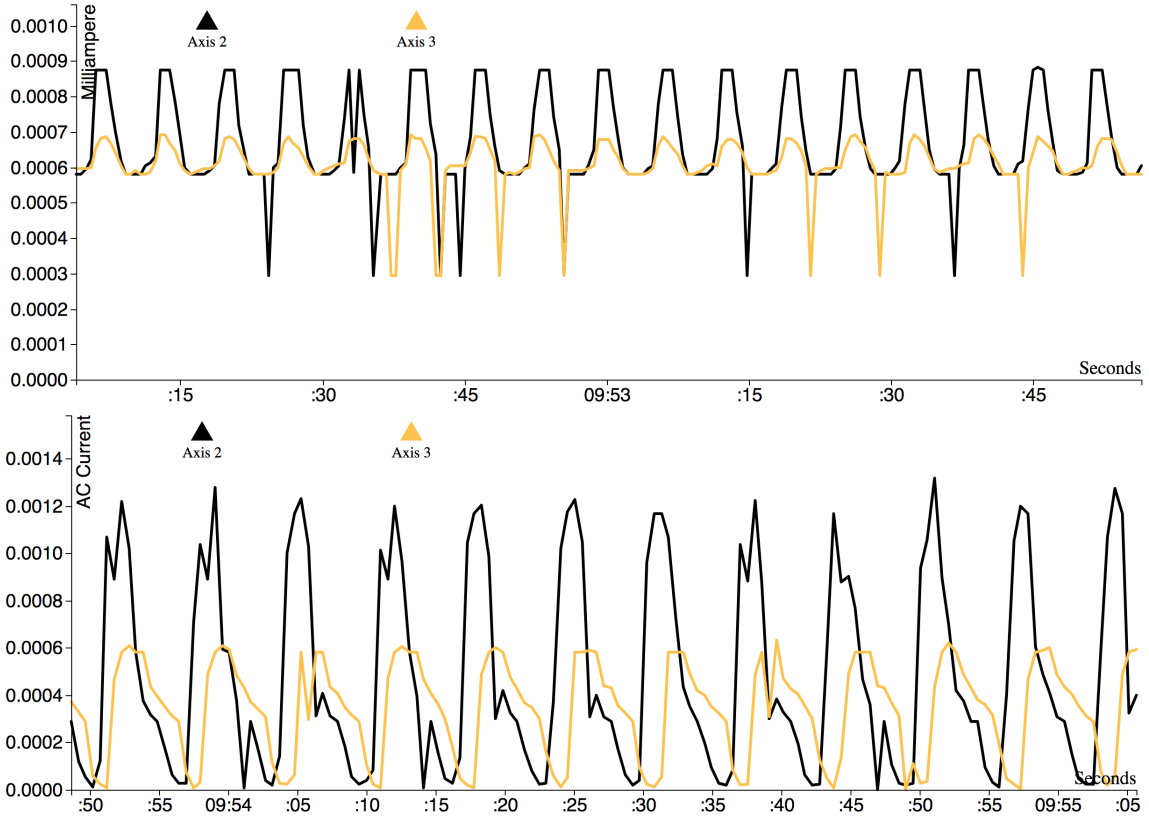



Figure 4.2: Comparing AC measurements to DC measurements of the e.Do's second and third motor (top: AC first motor (black), AC second motor (yellow); bottom: DC first motor (black), DC second motor (yellow)

Since DC doesn't have any phase shift or THD, and the voltage is measured independent of the current through additional sensors, the model is fed the following seven distinct values:

- Power (AC*voltage)
- Phase shift

- Alternating Current
- Voltage
- Total Harmonic Distortion
- Power (DC*voltage)
- Direct Current

### 4.2.5 One-hot encoding

Hot encoding is a process which converts the labels of the data to a format that can easily be handled by a ML model.[11]

For example, one-hot encoding the e.Do's labeling results in a single axis tensor with six dimensions which is equal to an array with six entrees.
Table 4.5 displays an example of one-hot encoding of the labels for a single motor being active.

| Label | One-hot encoding |
|---|---|
| MotorOne | [1,0,0,0,0,0] |
| MotorTwo | [0,1,0,0,0,0] |
| MotorThree | [0,0,1,0,0,0] |
| MotorFour | [0,0,0,1,0,0] |
| MotorFive | [0,0,0,0,1,0] |
| MotorSix | [0,0,0,0,0,1] |

Table 4.5: One-hot encoding example

### 4.2.6 Normalization and Standardization

The recorded values fluctuates strongly and experience large ranges. Such heterogeneous value distribution can cause gradient descent algorithms to do large gradient updates, preventing the network from converging.[11] Therefore, it is necessary to decrease the range and make the values more homogeneous before training the network.
There are two strategies to achieve this: normalization and standardization. Normalized data results in a range of $[0, 1]$ where the new value is its relative size compared to the original range. This is done through the following equation: 4.1.

$$z_i = \frac{x - min(x)}{max(x) - min(x)} \tag{4.1}$$

On the other hand, standardization subtracts the mean value from each data point and divides the results by the variance, as depicted in equation 4.2. Standardized data has a zero mean but is not bound to a specific range.

$$z_i = \frac{x - \bar{x}}{\sigma} \tag{4.2}$$

Normalization is often the preferred method. However, if the goal is to detect outliers, and the network model allows values outside of the normal range, standardization is the strategy to be followed [10]. In this case both of the aforementioned cases are met, which is why this thesis makes us of standardization.

## 4.3 The Classifier

Note: The fundamental idea and concept behind deep learning, classifiers and in particular CNNs are presented in length in 7.1.4 of the Appendix.
A classifier is a particular kind of ANN trained to *classify* data into different categories. The classifier developed in this work is used to identify the different movements of the robot arm as well as the different states of the industrial road as it relates to CM.

### 4.3.1 Configuring the Classifier

The following sections describe the parameter configuration of the classifier made from a CNN. The Appendix goes into more detail in explaining the individual parameters.

#### Setting the parameters for a Convolutional Layer

The following list shows the different parameters used to initialize one-dimensional layers in Keras that are used in this thesis:

- filters
- kernel_size
- strides
- padding
- data_format
- dilation_rate
- activation
- use_bias

'filters' defines the number of filters, or depth, of the Convolutional Layers. The number of features that a layer extracts depends on its depth. Hence, a greater depth results in more fine-grained feature extraction. However, this, of course, comes at a cost in performance. Therefore, an appropriate number of filters is a crucial choice. In the presented use-cases the number of filters were chosen based on the dimensions of the input data, adding at least one filter per dimension. This is done so that no information is lost in the process.

The second parameter, the ‘kernel_size‘, is the number of input neurons per layer neuron, also known as the receptive field for a one-dimensional CNN. The optimal size depends on the data, but it varies between one and the number of input neurons, or sample size. In general, a small kernel size w.r.t. the range of possible sizes results in a more fine-grained analysis whereas the opposite is the case for a large kernel size. The kernel size for the robotic arm is therefore smaller than for the industrial road. For one, because the robot data has a smaller sample size, and secondly, because the former system should recognize higher frequency patterns than the road.

The stride size determines the overlap in input neurons per kernel.
The dilation rate dilates the receptive field without changing its size, allowing larger pattern recognition without the need for greater computational power or a loss in input resolution. [5]. This is done by skipping values with a certain step size. However, for the use-cases at hand the patterns that should be recognized are fairly short term w.r.t. the sample, or input, size. Consequently, the dilation rate is set to one. In order to avoid mapping errors between the different dimensions of the sample size and the kernel size, padding is used. One such method, *zero padding*, introduces additional rows and columns to achieve a functioning mapping in between in the receptive field and the previous layer. This can be seen in Figure (7.8). For time series data it is important that the padding does not violate the temporal order, i.e., the output only depends on the past and present, and not the future. To achieve this in Keras, padding is set to "casual".
The ‘data_format‘ parameter is a setting which influences the data structure of the layer and depends on the network library Keras is using. For TensorFlow, this is set to "channels_last".
The choice of the activation function is important for translating the input activity to output activity of ANN. While sigmoid functions have been widely used the past, rectifier functions are becoming more and more popular.[10] These are zero when the input is less than or equal to zero, and a linear function when the input is greater than zero. The rectifier function is depicted in 4.3.

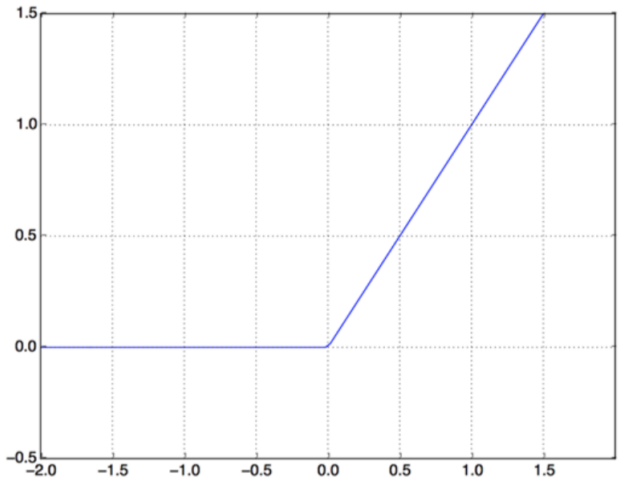



Figure 4.3: Rectifier function
[11, Francois Chollet, 2018, p.71]

The output value is further determined by the connection strength in between the layers and some bias $b$. For a multi-dimensional input this results in the following tensor operation:

$$output = max(0, dot(W, input) + b) \tag{4.3}$$

[11, Francois chollet, 2018, p.47]

The weights matrix W has the shape (input_dimension, #units ).
The rectifier function has an advantage over a sigmoid function in that it does not bound output values to be between 0 and 1 so that large changes in input values are actually reflected in large changes in the output. Secondly, it reduces the number of calculations because input values less than zero result in an output of zero, not in an output approaching zero as would be the case for a sigmoid activation function.[10]
However, the rectifier function also comes with some problems. The most prevalent one is the so-called "Dying ReLU problem".[10] If, for example, the bias is at one point set to a large negative value such that $w * x + b <= 0$, the output function always return zero, failing to re-active neurons as the gradient of zero is also zero. An approach to solving this problem is termed "LeakyReLU" and calculated as follows:

$$output = max(0.01 * dot(W, input) + b, dot(W, input) + b) \tag{4.4}$$

[11, Francois chollet, 2018, p.47]

The LeakyReLU function allows for a small positive gradient for an inactive neuron making the recovery of an otherwise dead neuron possible.

### Pooling

In CNN pooling layers are used to reduce the information gathered by the receptive field layers. For instance, max pooling takes the values of the field and returns only the maximum value, whereas average pooling returns the average value. Relevant literature suggests the usage of max pooling [10][11], which is the reason why it is used in this work. The resulting pooling layer is configured as to have a stride size in between 0.25 and 0.5 times the sample size.

### Batch Normalization

Normalization of the input data is an important step in ensuring that the model converges. This is especially important, as the values from the rectifier function can take on any positive number. This normalization is done between each convolutional and pooling layer. Batch normalization is explained in more detail in the Appendix.

### Dense Layer, Flattening and Soft Max

As explained in the Appendix it is necessary to flatten the output of the final convolutional layer into a single channel layer before feeding it to the following dense layers. Dense layer are layers in which each neuron is connected to all input neurons. These are used for the actual classification task. Theoretically, one dense layer can be enough to map the extracted features to a class or label. However, in practice more than one such layer is applied. This

is done so that the combination of unstable lower-level features present in the first dense layer translate to more robust high-level features more indicative of a certain class. Based on examples from "Hands-on Machine Learning with Scikit-Learn & TensorFlow"[10] the number of dense layers for the robot arm and industrial road was chosen to be between one and three, respectively. The number of neurons in a dense layer determines the complexity and variability of feature extraction. However, finding an appropriate number is done via trial and error, as researchers have yet to find an optimization strategy. However, the amount should be smaller than the input layer dimension and larger than the output layer dimension, otherwise a bottleneck is created.[10]
The activation function for the dense layers is the same as that for the convolutional layers, except for the final layer which shouldn't output continues values but rather mutually exclusive labels. However, such a strategy can result in a similar case as the aforementioned ReLU problem. So instead of classifying the output state as a specific class, the output is assigned probabilities for all classes that sum to 1. A function that satisfies these requirements is the $softmax$ activation function which is a generalization of the logistic function further discussed in Appendix 7.1.4. $softmax$ maps a vector of an arbitrary size to a vector of a predefined dimension with entries in the range of $(0, 1]$ so that their sum equals one.[10][11]

#### Dropout

To avoid overfitting the model to the data, the *dropout* method is used. By inserting a dropout layer in between two other layers, the output of a random set of neurons is set to zero. The size of this set is specified. Consequently, the neurons of the succeeding layer are forced to take into account the value of all connected neurons because they cannot rely on just a few selected inputs. The optimal dropout rate (i.e. the ratio of the neurons that are set to zero over the total neurons) once more depends on the application and data. Again, finding the best dropout rate is a process of trial and error [11]. For the presented use-cases the dropout rates were set to be rather small at 10 to 35 %.

#### Epoch

The training data is split into a specified number of batches. The entire collection of batches make up an epoch. For every training iteration consisting of one epoch the model is fed the entirety of batches. However, the order the batches appear in can change. Finding a reasonable amount of epochs that train the model without overfitting it w.r.t. to the training set is important. The approach for this thesis is to stop training once performance stops increasing.

### 4.3.2 Architecture

The following figure depicts a model of the architecture of the classifiers.

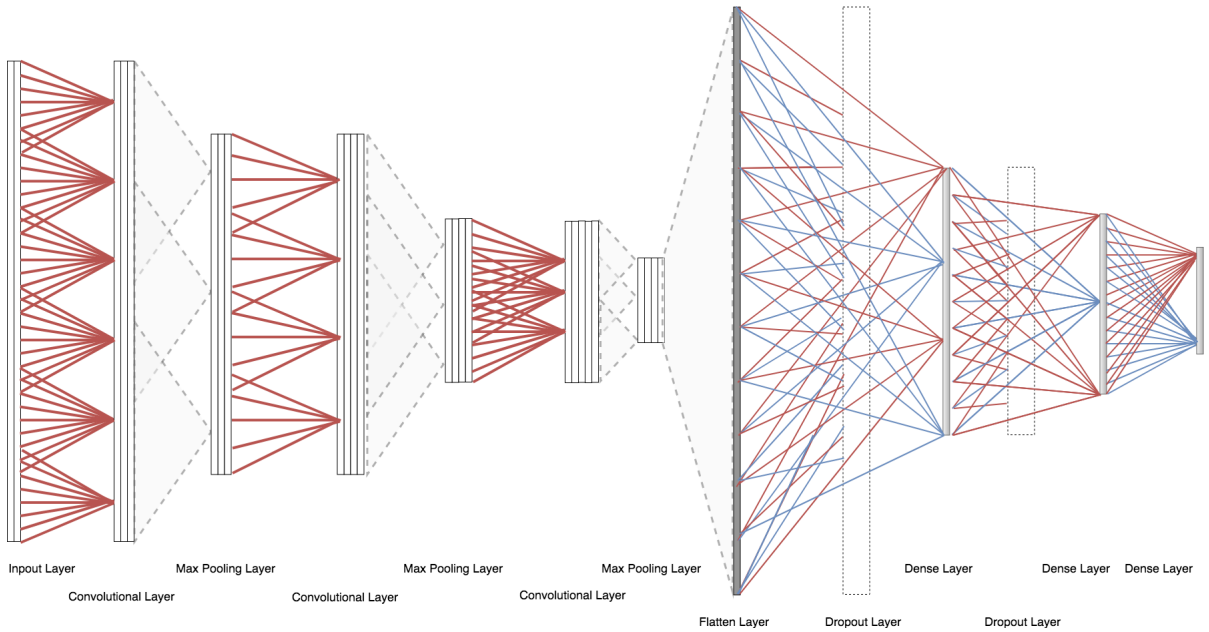



Figure 4.4: Architecture of the classifier

## 4.4 Anomaly Detection

For CM and PM it is crucial to be able to detect anomalies within the recorded data. A sudden jump in the average value of a variable can, for example, indicate a malfunction of a running process. Additionally, detecting long-term drifts in the variables is necessary for recognizing wear. Hence, anomaly detection is crucial for PM.
Although anomaly detection can be done with a variety of approaches, DL is especially useful for auto-encoders as explained in 2.2 of Chapter 2.
As explained in Appendix 7.1.6 an auto-encoder consists of two parts: an encoder and a decoder. The encoder typically decreases the input dimension and extracts the most meaningful representations of the training data, while the decoder learns to reconstruct the input as closely as possible based on the encoder's output. When confronted with input data not found in the training samples the decoder cannot reconstruct the encoder's input to the base accuracy which can be used to identify anomalies. Another benefit of auto-encoders are their capability to learn incrementally, enabling them to update their weights continuously from new data. [10] The simplest auto-encoders consist of three to five dense layers. However, for image reconstruction and generation CNN-based auto-encoders are frequently used [10]. These are also capable of long-term pattern recognition [5]. For these reasons, the implemented auto-encoders use convolutional and a mixture of convolutional and dense layers.

### 4.4.1 Configuring the Auto-encoder

In several points the auto-encoders are configured identically to the classifiers: they use the same ReLu activation function, take advantage of batch normalization, and are configured with similar dropout rates. The major differences to the classifiers is the output layer and

the arrangement and dimensions of the layers. The architectures are described in more detail in Appendix 4.4.2.

### Parameter configuration of Convolutional Layers

The considerations concerning the configuration of the parameters for the convolutional layers are almost identical to those for the classifiers. However, the number of filters per layer depends on the data compression of the model which will be explained shortly.

### Output Layer

In contrast to the classifier the output layer of an auto-encoder doesn't classify a given input state, but reconstructs this state. Therefore, if the input data is multi-dimensional this same shape is necessary for the output layer. Therefore, the output layer has to be a CL and cannot be a dense layer.

### Feature Extraction and Sample Reconstruction

The encoders' task is to compress the input data to a latent space representation with reduced dimensionality. One sample is comprised of two dimensions: the different variables and the different times. Thus, compression can be achieved by reduction in either one or both of these dimensions.
The channel dimensionality, i.e. the variables, can be altered through the number of feature maps/filters per convolutional layer or a flattening layer. The amount of time steps per sample can be decreased though pooling layers or the number of units per dense layer, for example. If the compression is to be achieved via a reduction in number of channels, the number of filters per CL needs to be reduced as the layer gets closer to the compressed state. However, to enable meaningful feature extraction the first layer should be equipped with a vast amount of filters. The exact parameter configuration is a result of trial and error.
When using a purely convolutional auto-encoder, decoding the compressed data is done through so-called transpose convolution in which the available data is merely up-sampled.[10] For one-dimensional layers this can only be done indirectly in Keras as is shown in the following code:

```
numOfRows    = 4
numOfColumns = 8

#Input:[[1,6,3,2][7,3,1,0][4,2,8,1][4,0,2,1]]
inputTest = Input(batch_shape=(None,numOfRows,numOfColumns))

maxPoolingLayer = MaxPooling1D(2, padding = 'same')(inputTest)
#Outputs: [[1,6,3,2],[4,2,8,1]]

upSamplingLayer = UpSampling1D(2)(maxPoolingLayer)
#Outputs: [[1,6,3,2],[1,6,3,2],[4,2,8,1],[4,2,8,1]]
```

The up-samling process by mere value duplication doesn't produce any new information.For this reason a second approach with a combination of convolutional, dense, and reshape layers was developed.
In contrast to the max pooling layer of the pure CL auto-encoder, the second auto-encoder flattens the two-axis input tensor of shape $(n, m)$ to a single-axis tensor of shape $(n*m)$. The neurons of the dense layer that brings the input into its final compressed state are connected all-to-all to the output units from the flattened layer. Additionally, by actively reducing the input dimension, the features reconstructed by the decoding part, i.e. the output, are based on the entirety of the compressed state. Up-sampling the data is now done not by simple duplication but by interpolation based on the weights of the individual neuronal connections. Naturally, the down-side of the second auto-encoder is its increased computing time.

### 4.4.2 Architecture

Figures 4.5 and 4.6 depict the resulting architectures for the auto-encoders.

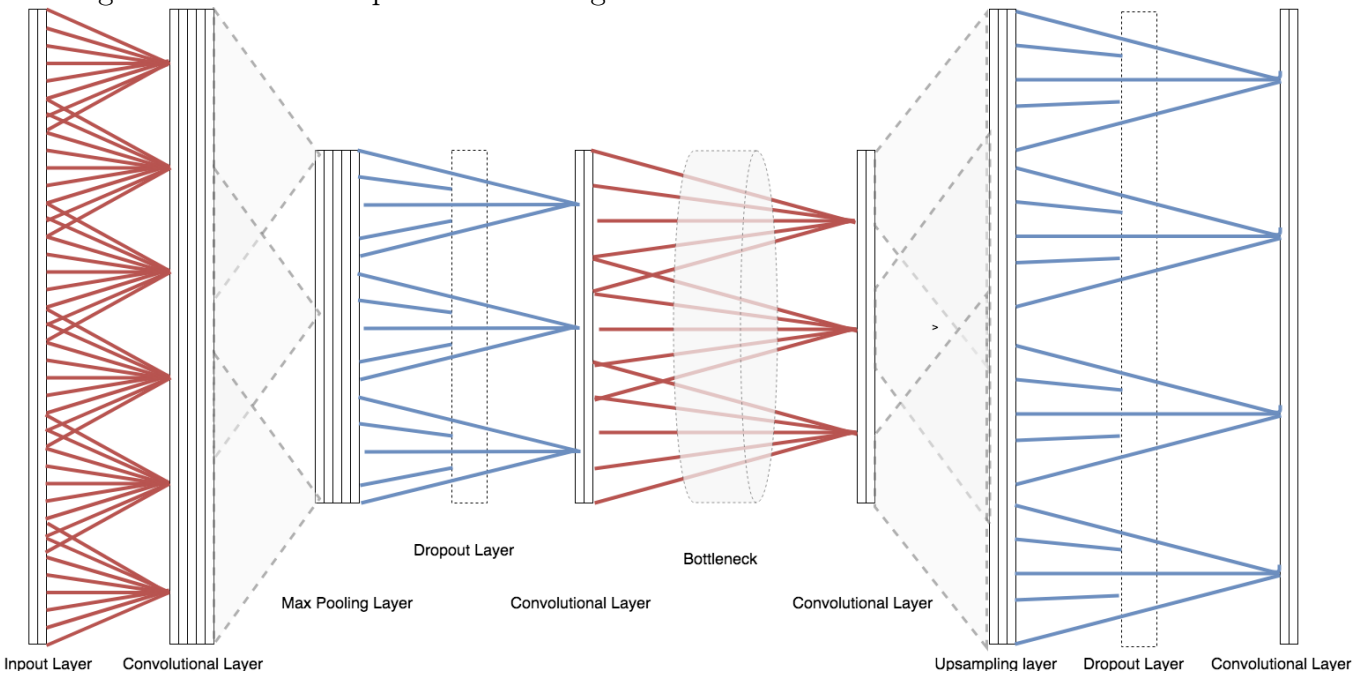



Figure 4.5: Architecture of the pure convolutional autoencoder

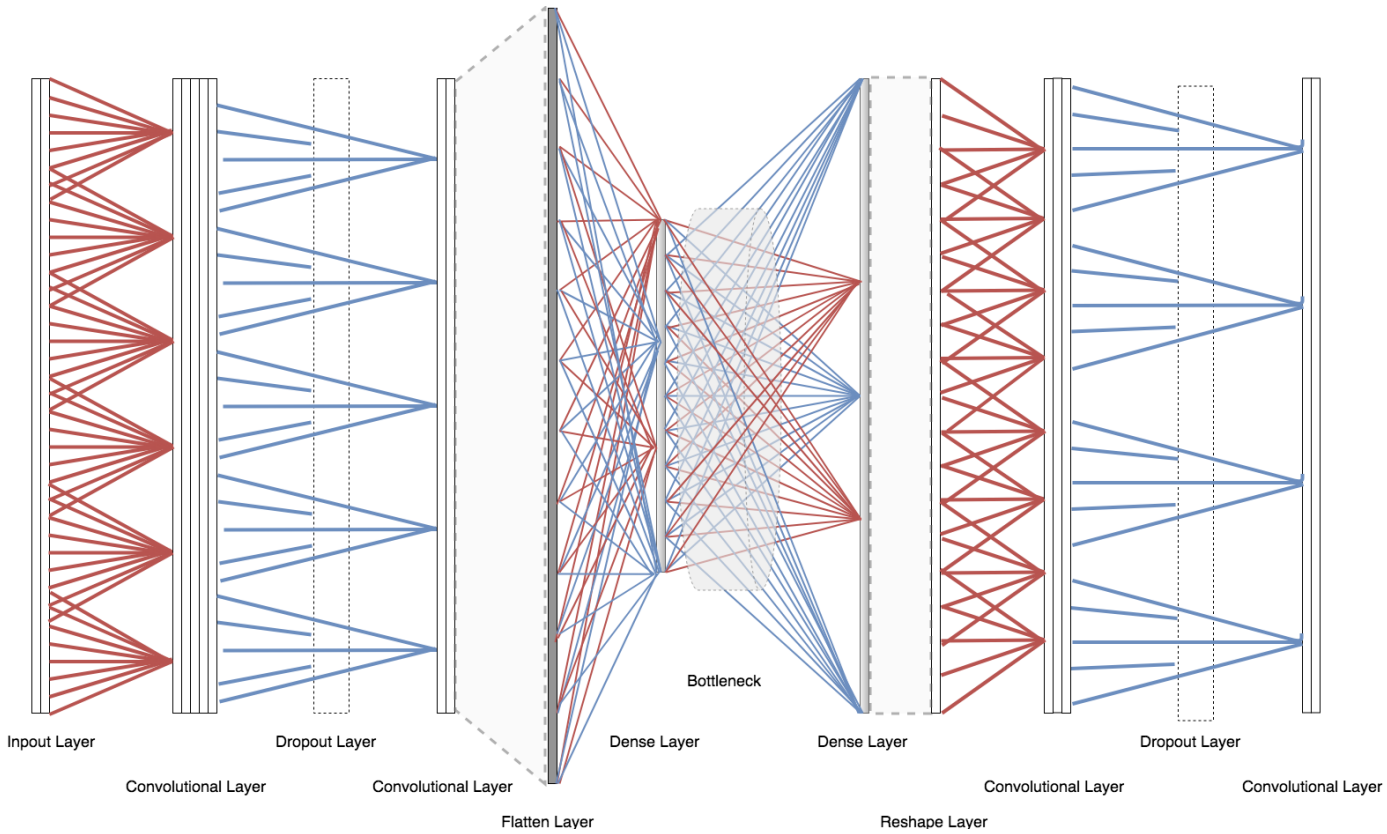



Figure 4.6: Architecture of the hybrid autoencoder

## 4.5 Chapter Summary

This chapter is headed by a listing of reasons why Convolutional Neural Networks are used for the application at hand and how this is implemented. What follows is an overview of the different preprocessing techniques that the raw data undergoes before being fed to the models. Finally, an explanation of the models' parameter configuration and topological architecture for both the classifier and the auto-encoders is presented.

# 5 Results

This chapter presents the results of the DL models for the respective use-cases. First, the outcomes of the classifiers are discussed which is followed by that of the auto-encoders.

## 5.1 Performance of the Classifiers

The classifier's task is it to learn to distinguish different states of a process and label the samples appropriately.

### 5.1.1 Industrial Road

The underlying input data consists of measurements of the power, phase shift, current, voltage, and THD of the main power supply of the industrial road demonstrator. The classifier is supposed to determine which combination of sections are predominantly active for a given input sample. Theoretically, eight different states are possible resulting in eight different labels. However, a cycle of the industrial road does not necessarily include all combinations resulting in the model "knowing" less than eight states. The size of the window function further influences the number of labels, as a smaller size results in more fine-grained labeling and vice versa for a larger size. For a window function that is approx. 8s long, the classifier learns five labels, for a 5-6s window it knows six different states, and for a window size of 3-4s seven of the eight possible labels exist in the classifier's view. Too small sizes may result in labels of states that don't actually represent a cyclic process, but the rare occurrence of these do not impact the model's performance.
Table 5.1 shows a sample of the standardized and formatted data.

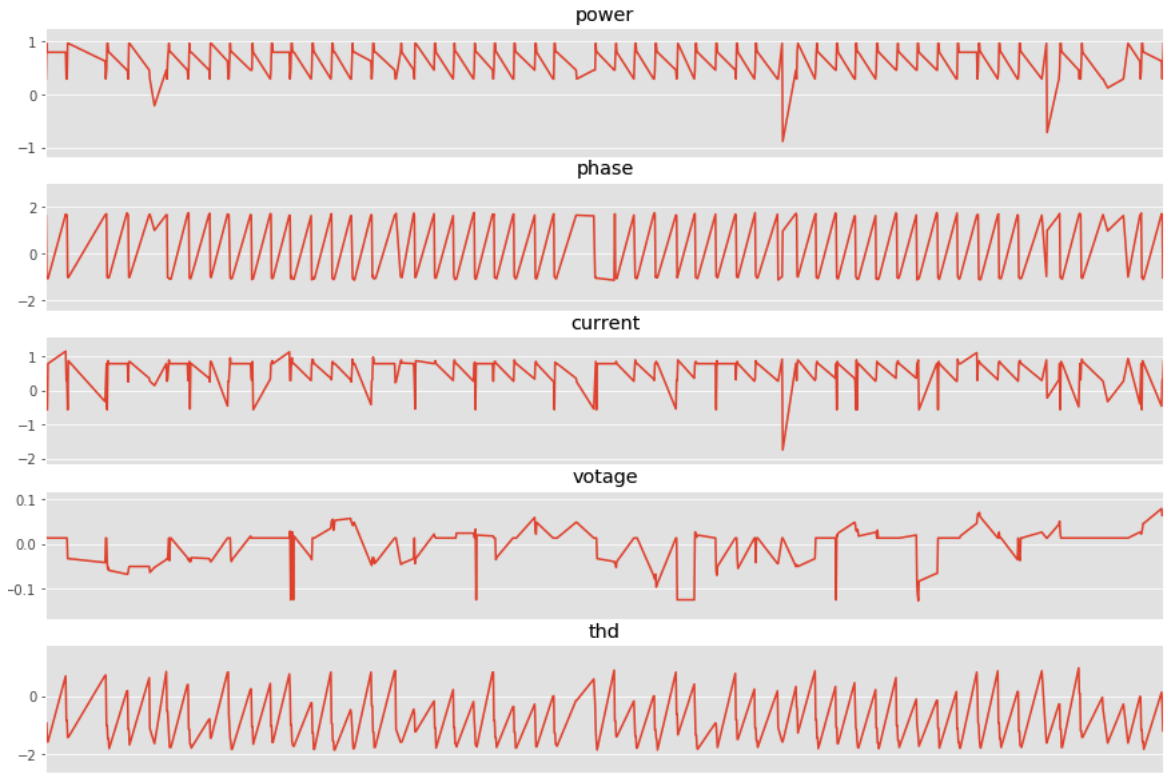



Figure 5.1: Snippets of the festo demonstrator's standardized data

A 60/40 ratio of training vs testing data was used in this work.
Table 5.2 depicts the development of the accuracy and the loss over five training epochs. Every epoch depicts the average accuracy and loss, a measure for model robustness, during training and testing. As can be seen the accuracy converges quicker than the loss. The 24 data point data corresponds to the 8s window, the 12 data point data to the 5-6s window, and the 3-4s window translates to 8 data points.

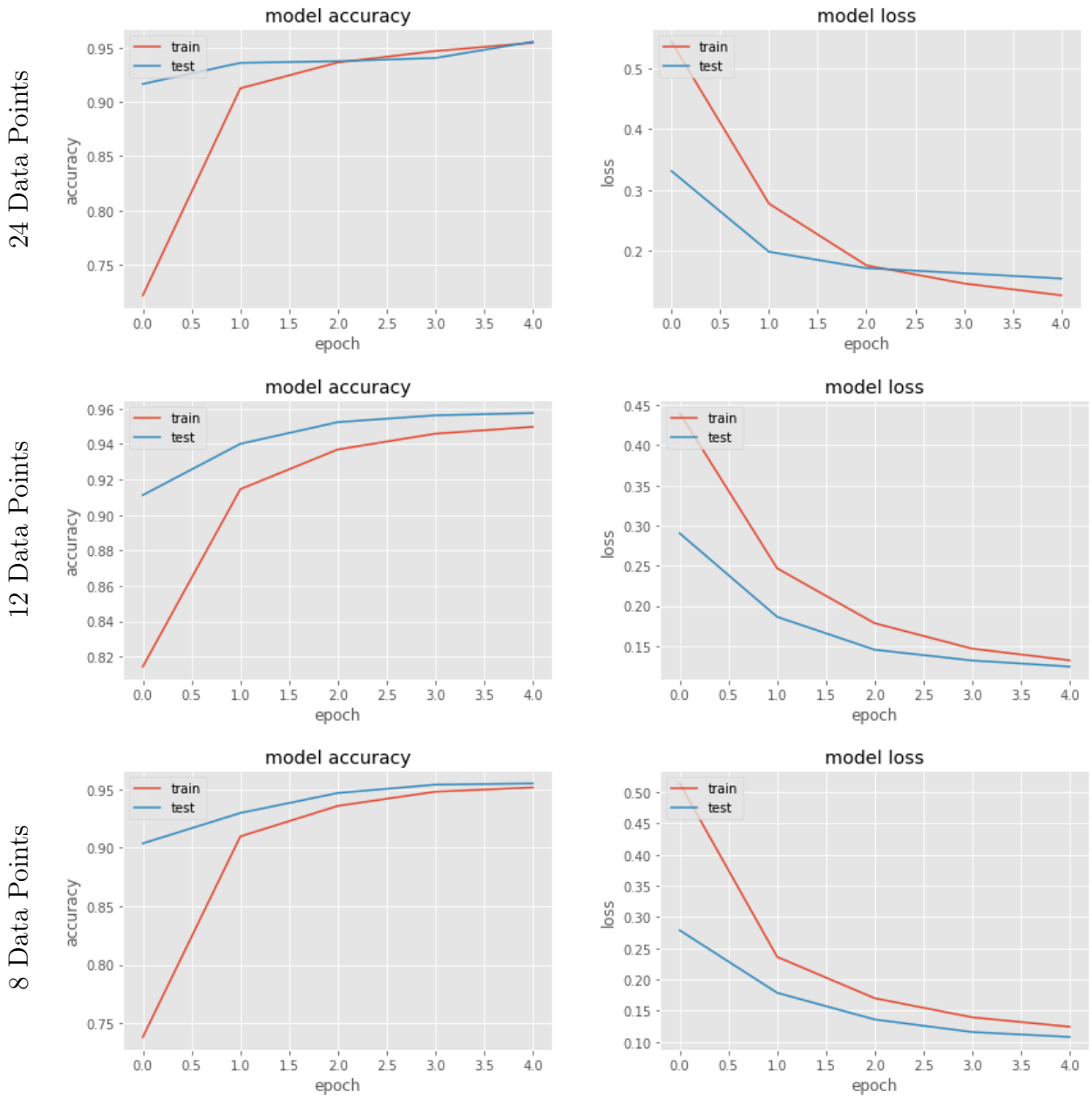



Figure 5.2: Accuracy and loss for five epochs for the data of the festo demonstrator for the training data

Figure 5.3 depicts the development of the loss and accuracy for the three cases over a duration of twenty epochs.

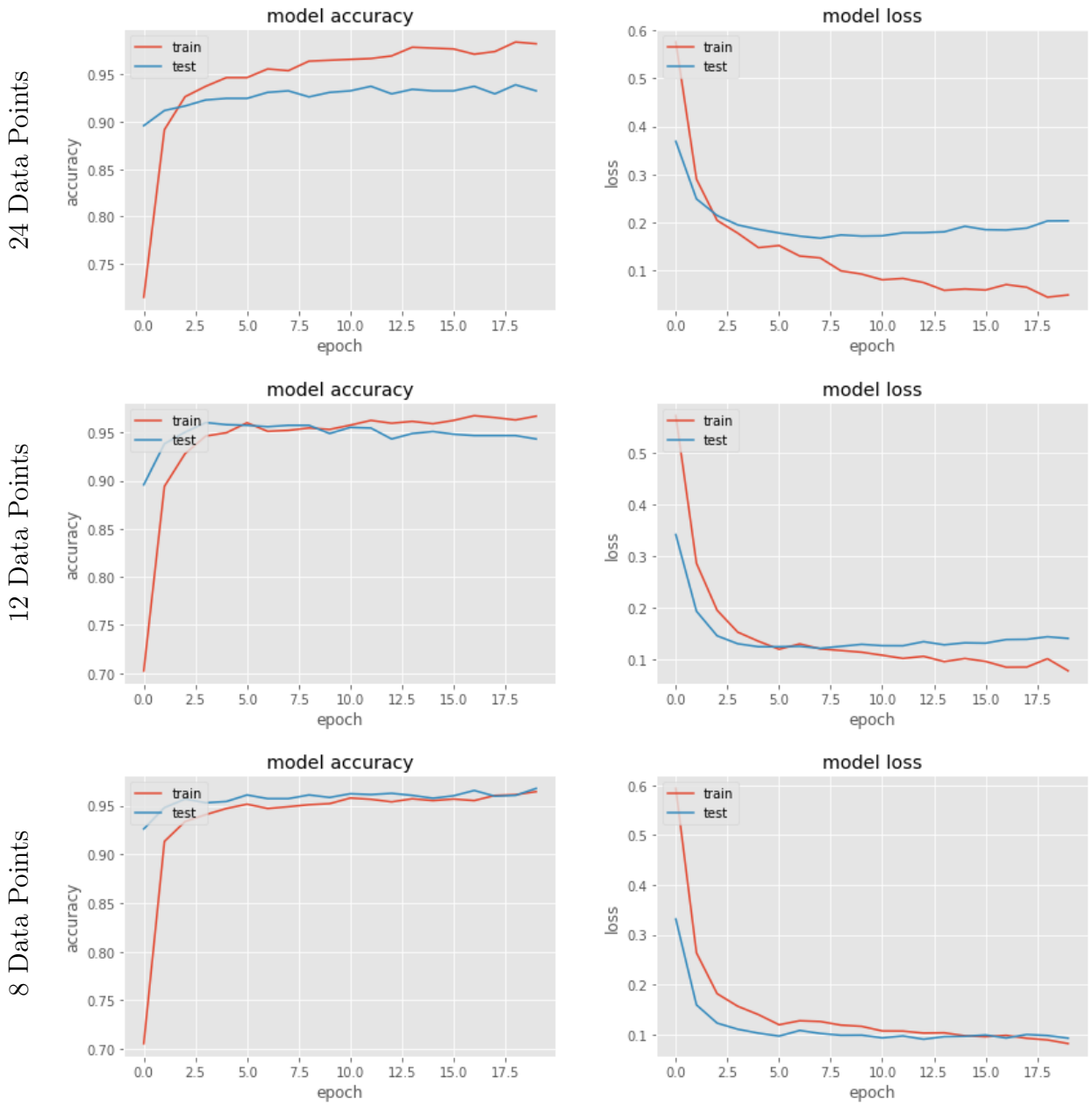



Figure 5.3: Accuracy and loss for 20 epochs for the data of the festo demonstrator

For the smallest sized window an increase in number of epochs increases the model's performance, bringing down the model's base error to a mere 3.5%. In contrast, the performance for the larger window sizes decreases. The difference in accuracy and loss between the training and test data suggest overfitting of the model.
Experiments determined the highest accuracy for all three window sizes with an average of approximately 32 filters per convolutional layer and 32 neurons per dense layer. For larger values, the performance increase is negligible and less robust.

### 5.1.2 Robotic arm

Three different data sets were recorded for this use-case:

- The consumption characteristics of singular motor activity for the six motors
- The consumption characteristics of three distinct movements each powered by a set of two motors
- The consumption characteristics of two distinct movements each stemming from the interplay of three motors

Due to the higher frequency of this data, the window functions are smaller compared to those of the industrial road.

#### Single motors

The data feed has seven channels, one for each measurement of the AC and DC of the e.Do. Each motor activity is assigned a label, resulting in six distinct labels. As in the previous use-case, the training-test split was chosen to be 60-40. The batch size was set to encompass 20 samples and training was conducted for five epochs. Several window sizes were used. The first includes 14 data points, the second eight, and the last window size is configured to measure four data points. Table 5.4 depicts the accuracy and loss for the different window sizes.

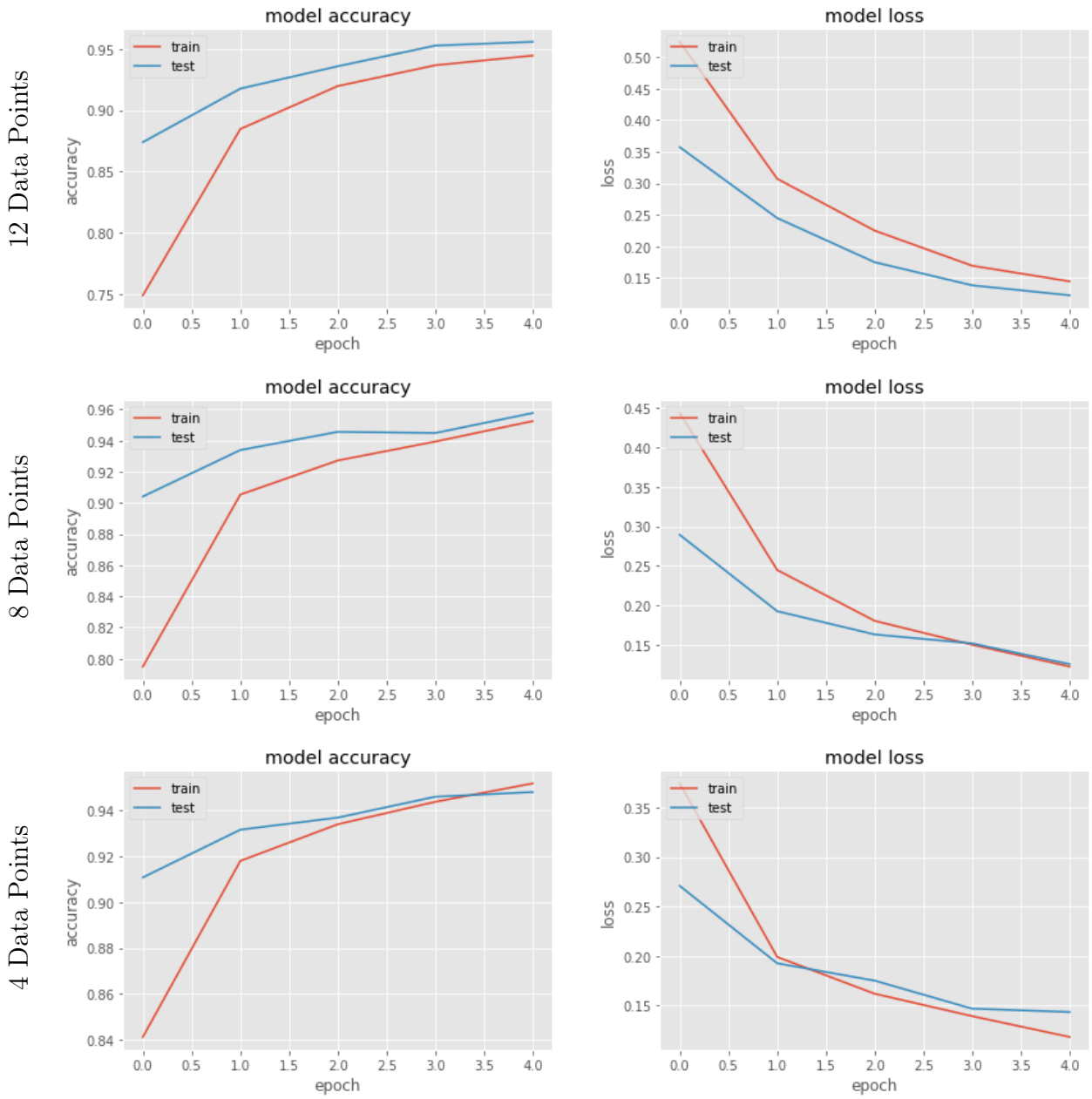



Figure 5.4: Accuracy and loss for 5 epochs for the data of the e.Do

Due to the higher number of channels, the best performance was achieved, in all three cases, with an average of about 64 filters per CL and 64 neurons per dense layer. The final accuracy is approximately 95% in all three cases.

The gradient of the loss, as well as the accuracy, indicates that a these haven't yet converged and a higher number of epochs would achieve even better results. This proves to be true. Training the model for 30 epochs results in final accuracies of approximately 98% as can be seen from Figure 5.5.

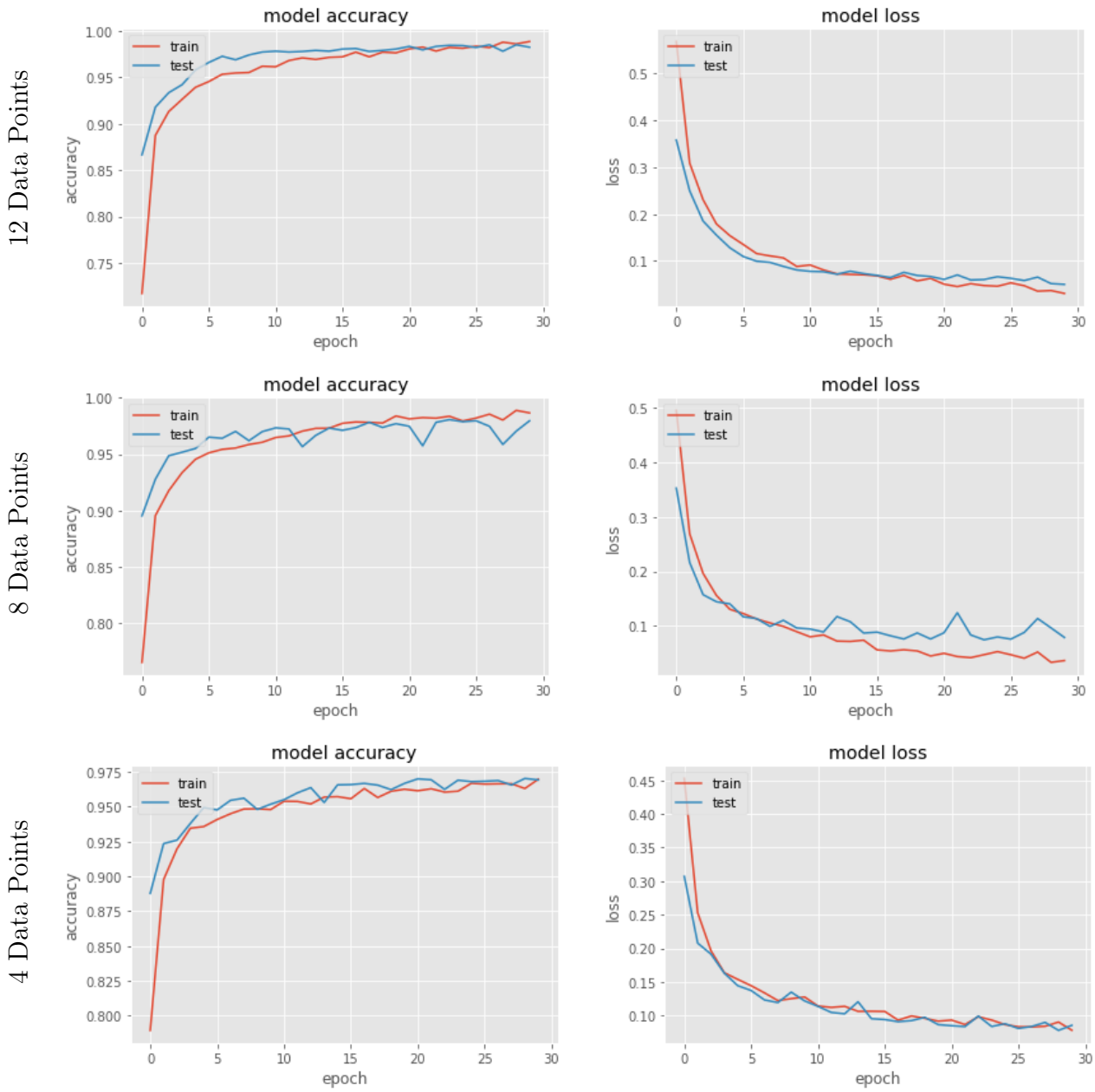



Figure 5.5: Accuracy and loss for 30 epochs for the data of the e.Do

A further increase in the number of epochs without adjusting the model's parameters resulted in unstable learning suggesting overfitting.

### Two motors

The second data set is composed of three distinct movements, where each movement uses two of the six motors. The movements are cyclic and undergo the following deviation before returning to their starting positions. The motors are enumerated from bottom to top.

- 90° rotation of motor 1, 90° bow of motor 2

- 90° rotation of motor 4, 90° bow of motor 3
- 90° rotation of motor 6, 90° bow of motor 5

Each movement was repeated 50 times with and 50 times without an attached weight. This data set was only evaluated with a window size comprising 12 data points, as the previous data set showed that the window size had little impact on the results. The average number of filters per CL was decreased to 48 filters. Again, each dense layer has 64 units/neurons. For 20 epochs with a batch size of 20 an accuracy of nearly 99 percent was achieved with a baseline error of 1.3 percent. This can be taken from Figure 5.1.

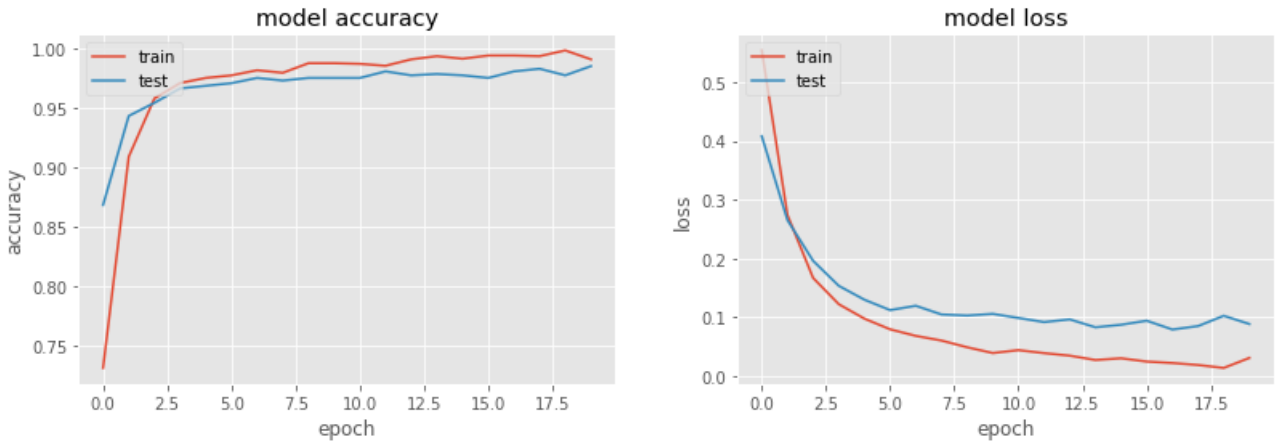



Table 5.1: Accuracy and loss for 20 epochs for the e.Do's second dataset

### Three motors

The third data-set is composed of two distinct movements, each using three of the six motors. Once again, the motors are enumerated from bottom to top. The following depicts the deviation from the arm's starting point. One movement consists of this deviation and a return to the initial position.

1. 180° rotation of motor 1, 90° bow of motor 3, 90° bow of motor 5
2. 90° bow of motor 2, 180° rotation of motor 4, 180° rotation of motor 6

Again, the movements were repeated 50 times with and 50 times without a weight attached to the arm. Here an average of 64 filters per CL and 64 units per dense layer with a training length of twelve epochs resulted in the best results. The accuracy achieves values of up to 99.3%, as can be seen in Figure 5.2.

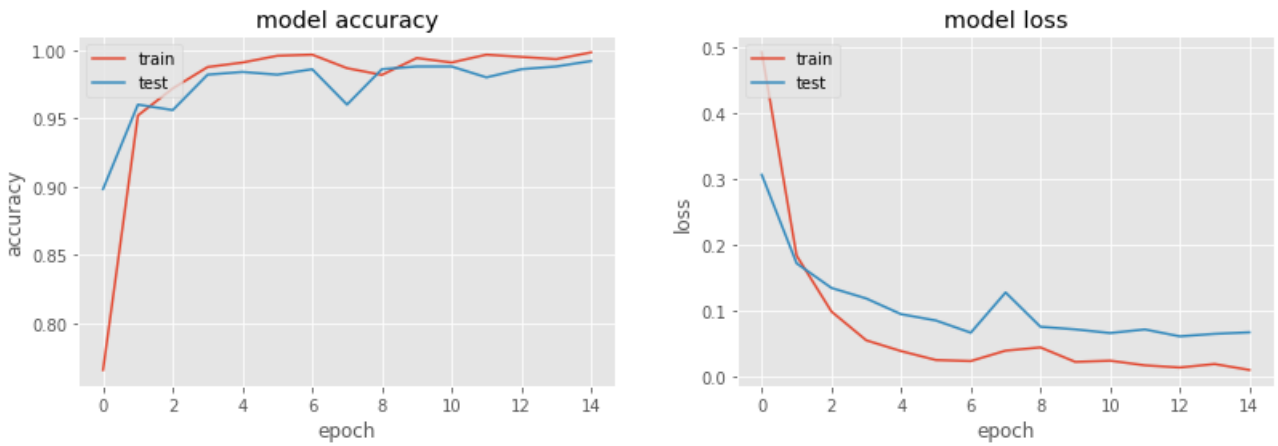



Table 5.2: Accuracy and loss for 20 epochs for the e.Do's third dataset

## 5.2 Performance of the Auto-encoders

The auto-encoders learn to compress input data and reconstruct it based on this compressed representation. Hence, if the input shows anomalies these are detected by recognizing a higher than usual reconstruction error of the decoded data. For evaluating an auto-encoder's performance a different approach is necessary. First, the reconstruction errors are calculated, and a threshold for the error is set to distinguish between a 'normal' error and an error stemming from an anomalous input. Then the performance is measured by how well the model can identify outliers and the associated loss function.

The following subsections evaluate the two different architectures of the auto-encoders, in which one is only made up of convolutional layers, and the other is a combination of these and fully connected, or dense, layers.
The evaluation is carried out using two of the three data sets from the robotic arm use-case:

1. e.Do: Singular motor movements
2. e.Do: Three distinct movements from two motors each

To detect abrupt changes the window size was chosen to be rather small and include eight values per sample with a frequency of approximately 0.5 Hz. Training is done over 150 epochs, the batch size is 100 samples.

### Fully Convolutional Autoencoder

The change in loss over time of this auto-encoder is depicted in Figure 5.3.

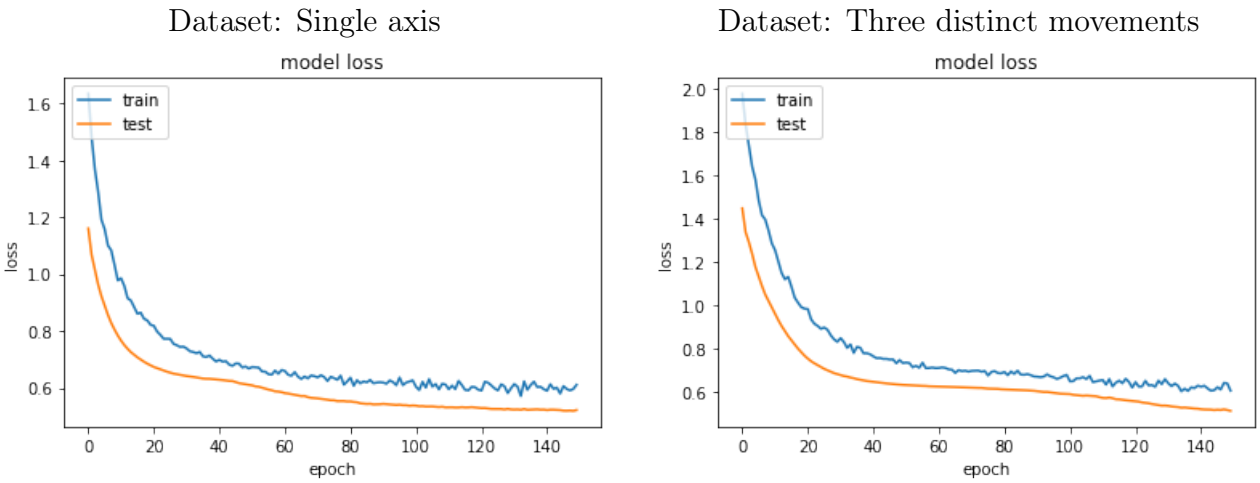



Table 5.3: Loss of the purly convolutional autoencoder (original dataset, 150 epocha): Left dataset of the single axis, right dataset of the three distinct movements

The results are nearly identical for both of the data sets. Therefore, only one set is examined further. Figure 5.6 shows the auto-encoder's reconstructed data for a single sample.

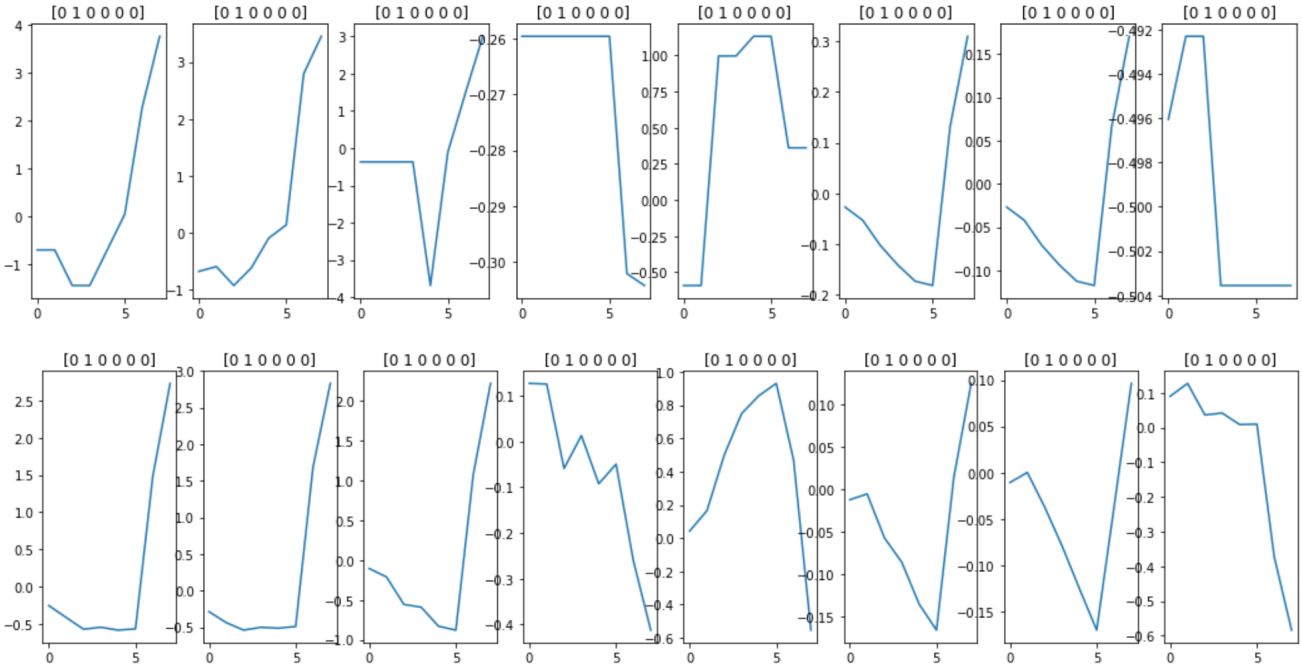



Figure 5.6: Original sample (top) compared to the reconstructed sample (bottom) for the purely convolutional autoencoder (original dataset)

The average reconstruction error is 3.2. The reconstruction error threshold for distinguishing anomalies from regular samples was set so as to separate the upper 10% of values from the lower 90%. Figure 5.7 shows the reconstruction error per sample and the calculated threshold. There are several sample with extremely high errors.

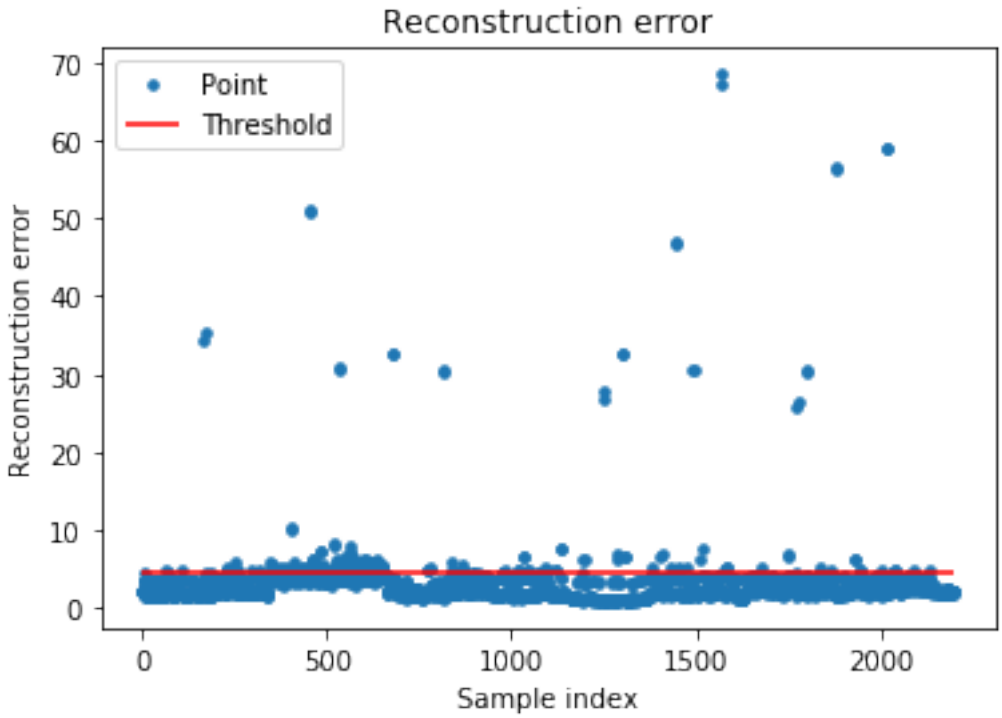



Figure 5.7: Reconstruction errors for the original dataset by the purly convolutional autoencoder

Figure 5.8 shows an example of an outlier sample. The spikes (6th and 7th plot in the Figure) are the measurements for DC current and power and far higher than the standardized average. For this reason the model has a hard time reconstructing the sample resulting in a large error.

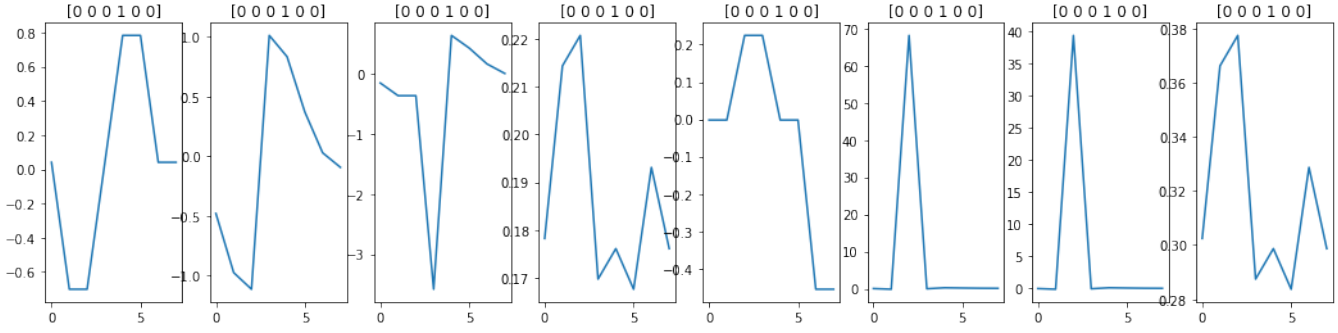



Figure 5.8: Singular anomaly (spike)

An attempt to improve the model's performance is done by cleaning the data from samples with error above the threshold and training the model with the resulting data. The resulting loss, considerably less than for the previous iteration, can be taken from Figure 5.9.

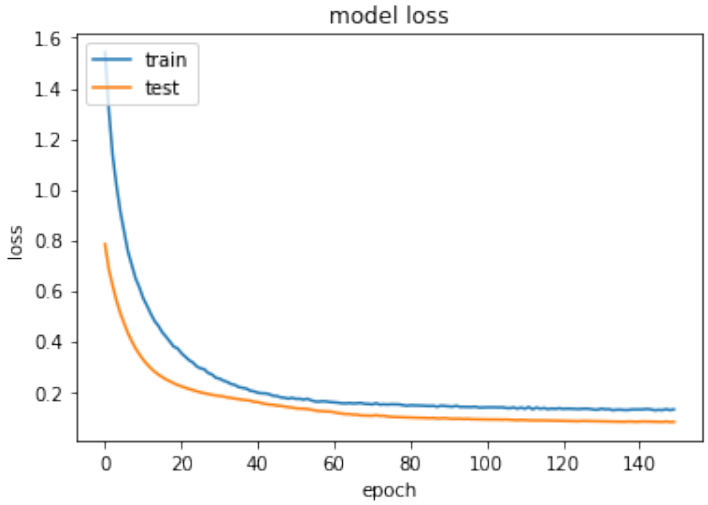



Figure 5.9: Loss of the purely convolutional autoencoder (cleand dataset, 150 epochs)

Figure 5.10 shows an example of the improved sample reconstruction (as compared to Figure 5.6.

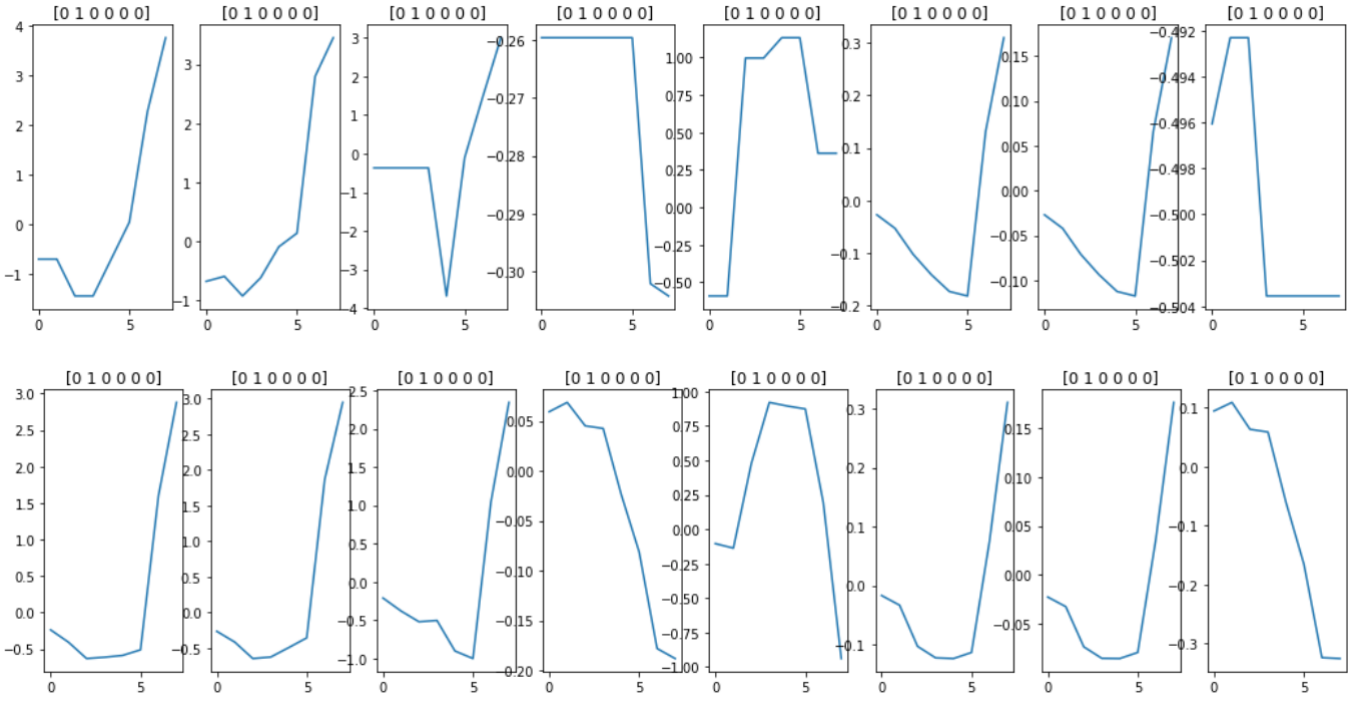


Figure 5.10: Original sample (top) compared to the reconstructed sample (bottom) after cleaning the data for the purely convolutional autoencoder (cleaned dataset)

Figure 5.11 shows the reconstruction error distribution for the model trained on cleaned data. The samples were order by label, i.e. the first $floor(\frac{2000}{6}) = 333$ samples are of the first class, the next 333 samples of the second class, and so forth. These section can be clearly seen in the Figure, indicating that the auto-encoder has class-dependent performance. This may be due to a difference in complexity of the movements.

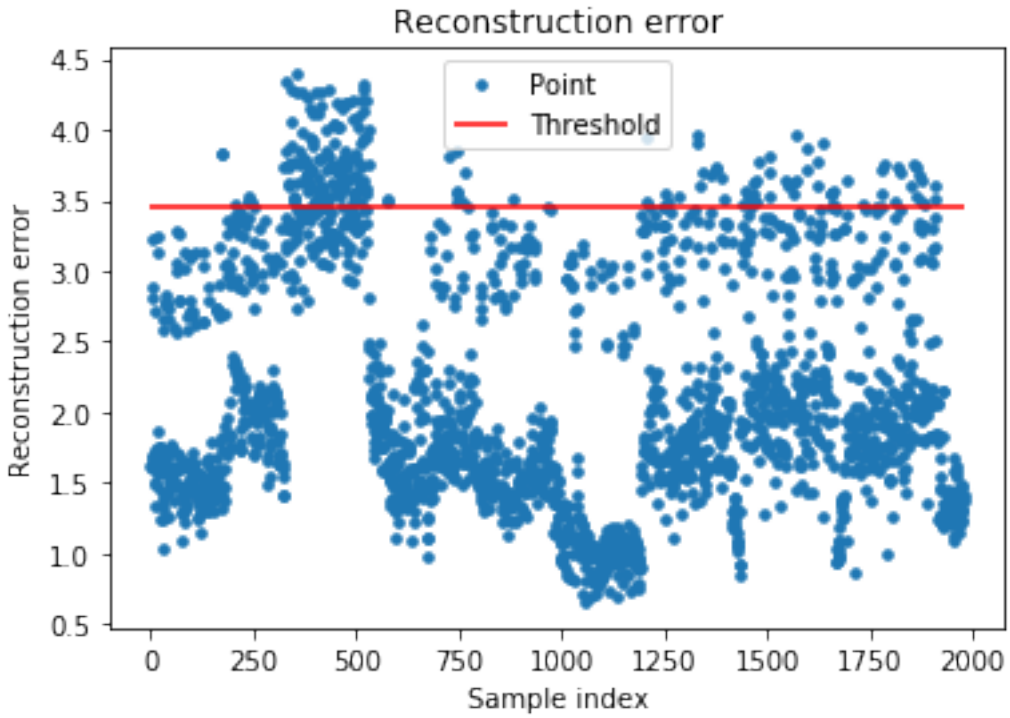



Figure 5.11: Reconstruction error for the cleaned dataset by the fully convolutional autoencoder

In total, the average reconstruction error of the adjusted model for the cleaned data set is 2.1 whereas the average error for the original data set is 2.9 | 0.3 less than that of the original model. Additionally, the adjusted auto-encoder also results in outliers from the original data having greater reconstruction error. This is shown in Figure 5.12.

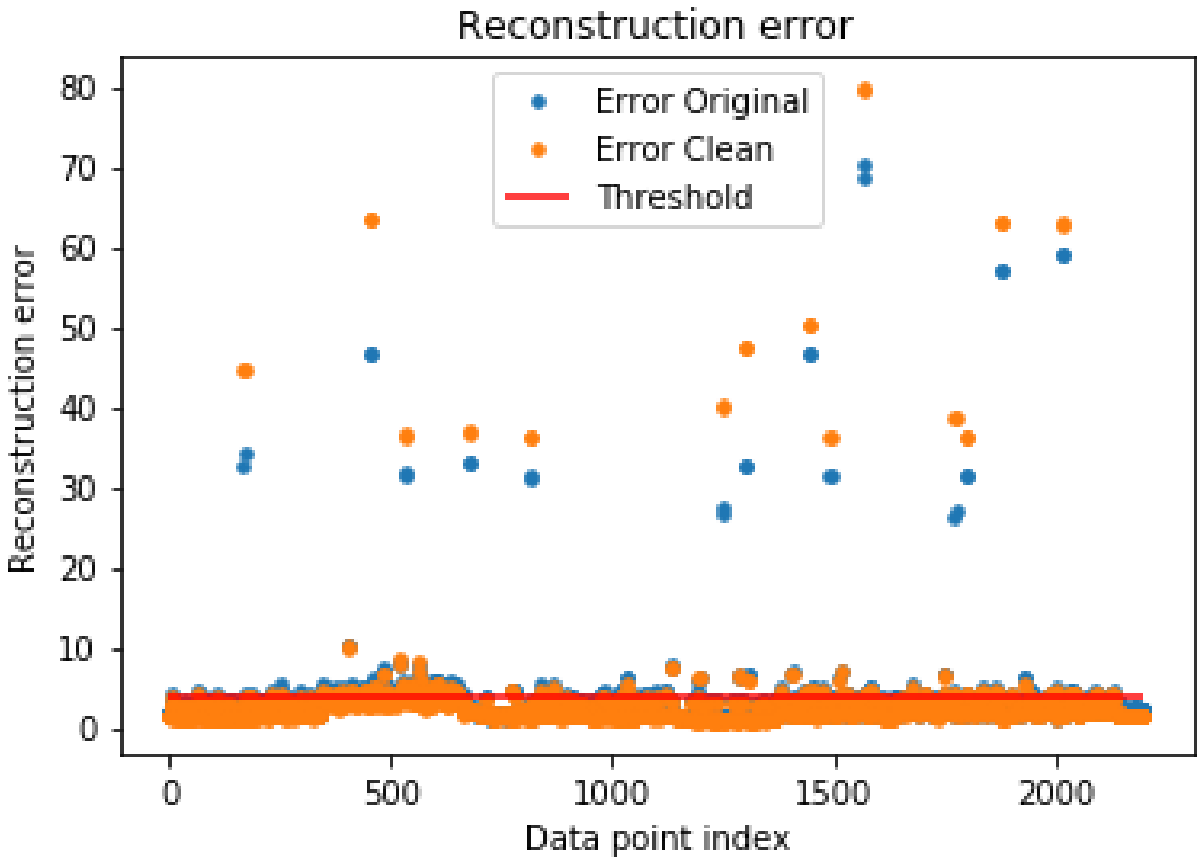



Figure 5.12: Convolutional Model: Reconstruction errors by the model learned on the cleaned data (orange) vs. that of the model learned on the orginal data (blue)

### Hybrid Autoencoder

The hybrid auto-encoder encodes the input data using two CL, one droptout layer, and one dense layer. The layers of the decoder are a mirror image of the encoder's. Before bringing the input into its compressed form it is flattened. In order to reconstruct the original dimensionality a reshaping layer is needed in the decoder. Figure 5.13 depicts the loss of the model when training on the original data set of the single motors.

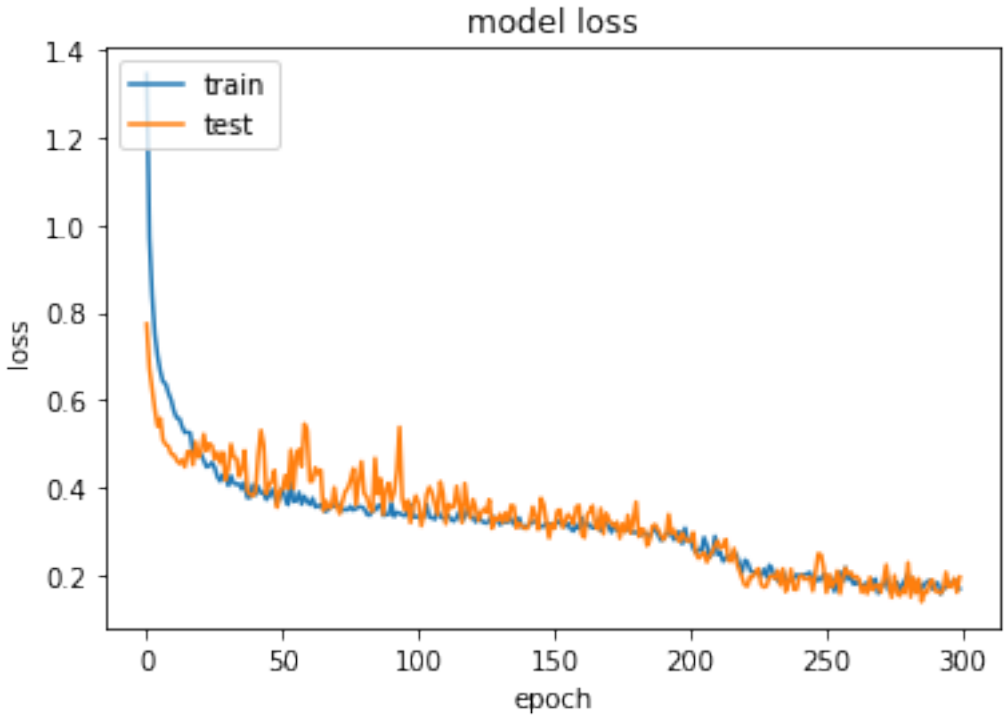



Figure 5.13: Loss for the hybrid autoencoder (original data set, 300 epochs)

Figure 5.14 shows the example input sample in its original and reconstructed form.

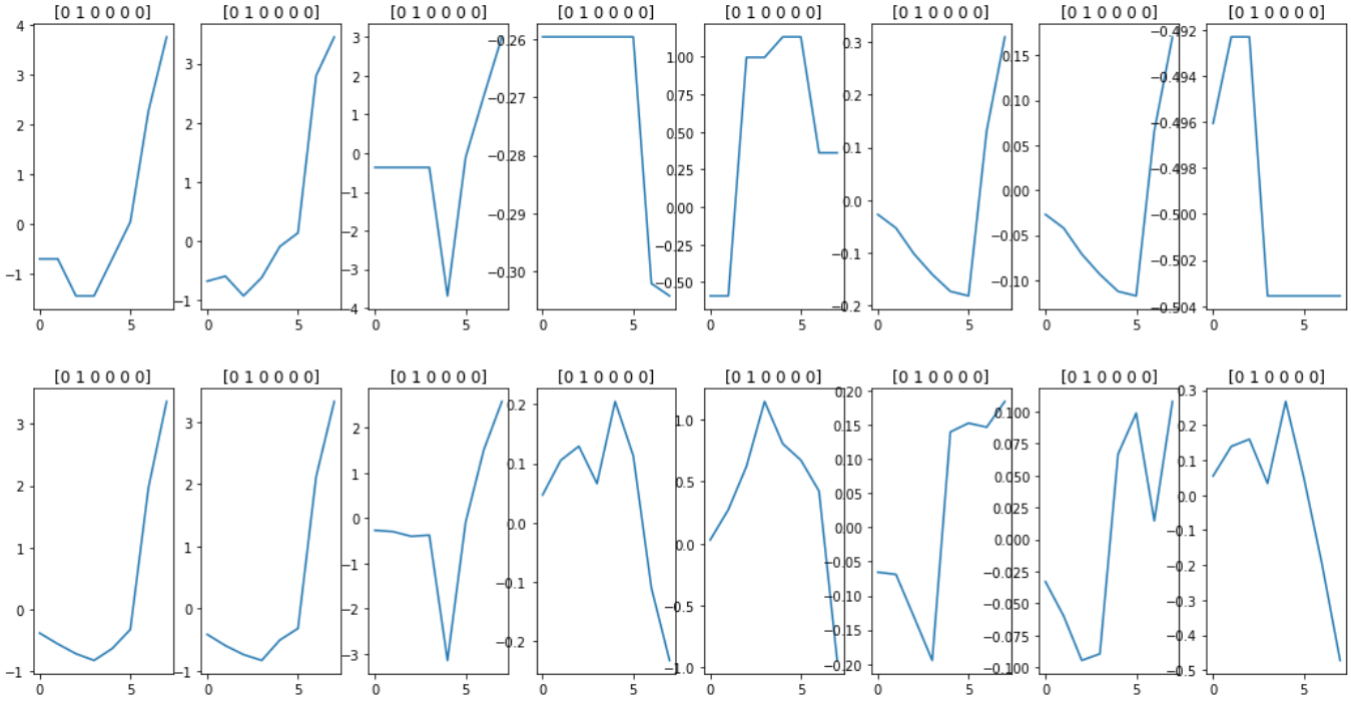



Figure 5.14: Original sample (top) compared to the reconstructed sample (bottom) for the hybrid autoencoder (original dataset)

The average reconstruction error for the hybrid model trained on the original, uncleaned set

is 2.4. The threshold is set in the same way as for the first auto-encoder. Figure 5.15 depicts the resulting reconstruction error per sample.

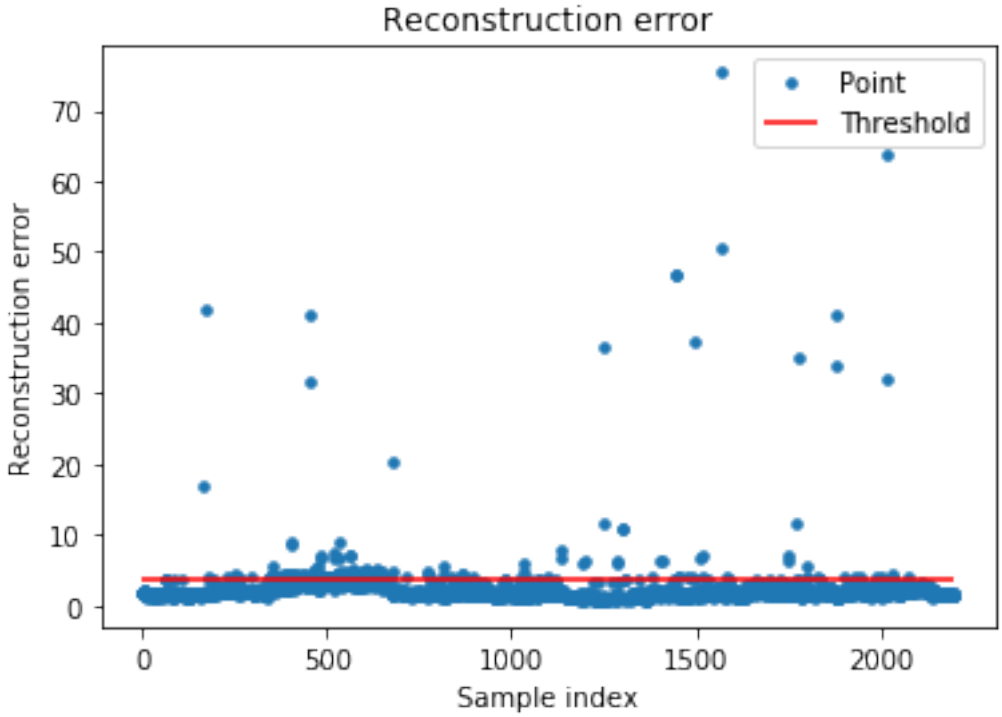



Figure 5.15: Reconstruction error for the orignial dataset by the hybrid convolutional autoencoder

As for the previous auto-encoder, training is repeated with the cleaned data set. The resulting loss curve is plotted in Figure 5.16.

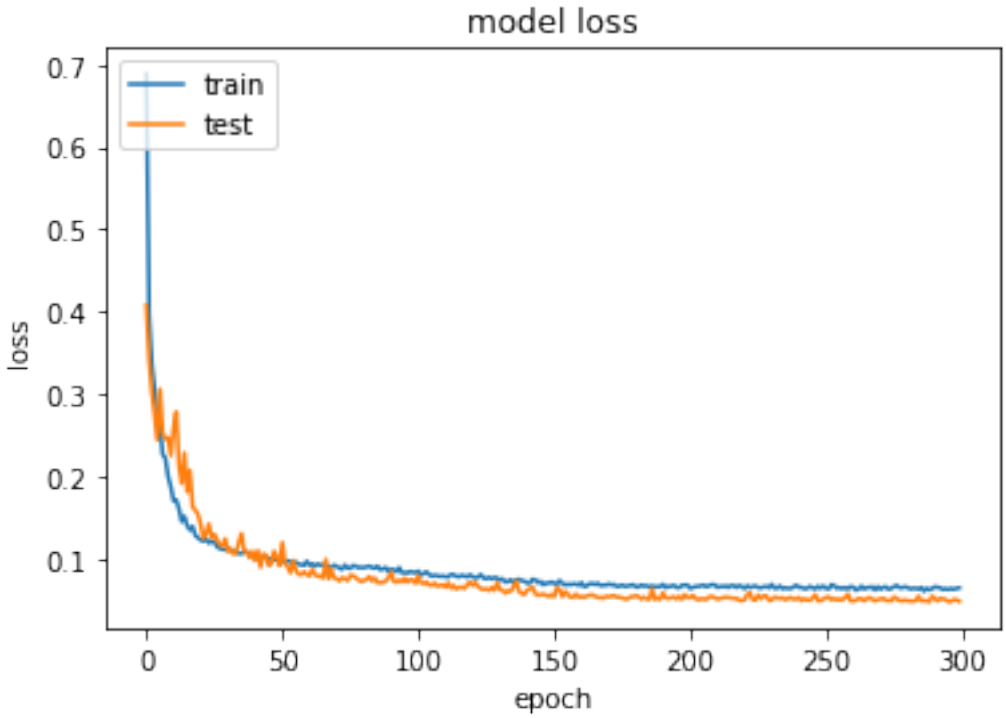



Figure 5.16: Loss of the hybrid autoencoder (cleaned data set, 300 epochs)

In Figure 5.17 the example sample is reconstructed by the adjusted hybrid model trained on the cleaned data set.

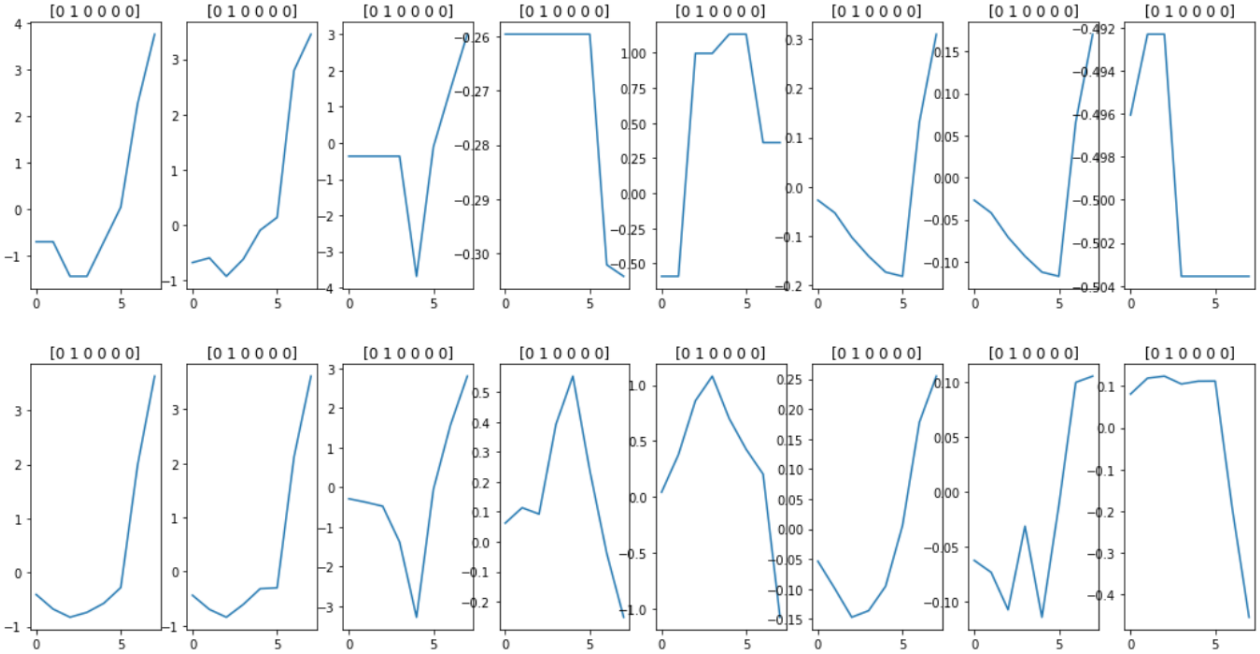



Figure 5.17: Original sample (top) compared to the reconstructed sample (bottom) for the hybrid autoencoder (clean dataset)

The resulting average errors for the adjusted, hybrid model for the original and cleaned data

sets are 2.4 and 1.6, respectively.
Figure 5.18 depicts the reconstruction errors of the cleaned data ordered by label. The model can be seen to also be less class-dependent, although the differences in average error by class aren't completely gone.

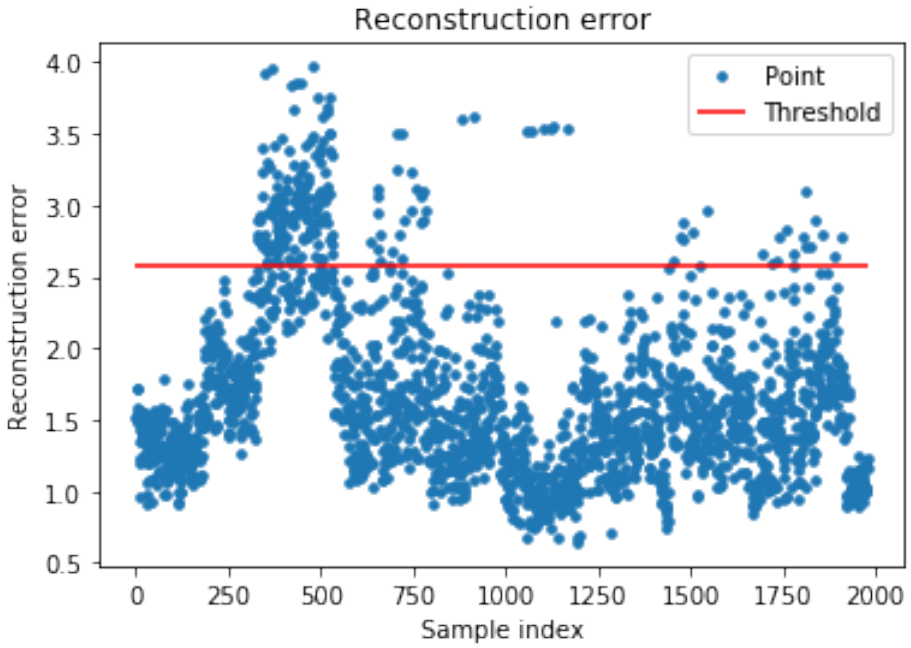



Figure 5.18: Reconstruction errors of the cleaned data by the hybrid convolutional autoencoder

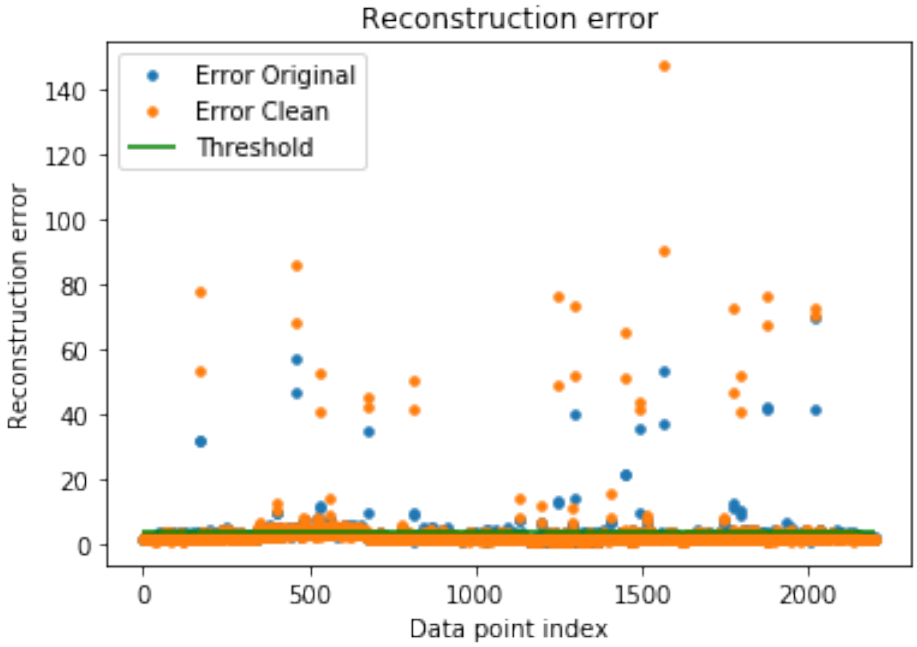



Figure 5.19: Hybrid Model: Reconstruction errors by the model learned on the cleaned data (orange) vs. that of the model learned on the orginal data (blue)

## 5.3 Chapter Summary

First, the results chapter discusses the performance of the classifier for both the industrial road and the robotic arm. Then the results of the two auto-encoder's used for anomaly detection, and a refinement of these, are analyzed and discussed with respect to the robot data.

# 6 Discussion And Outlook

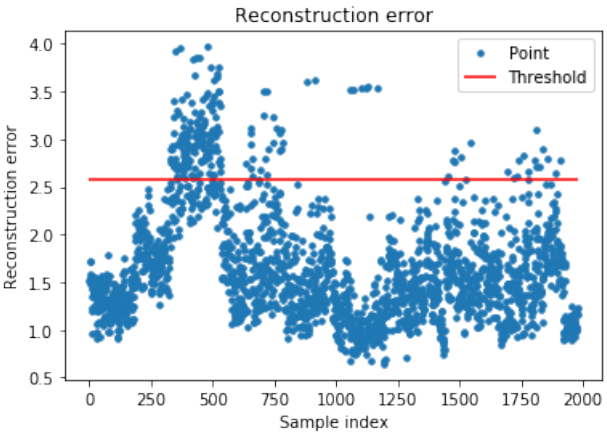
This work aims to overcome the multi-domain problem that current monitoring systems have through the use of an EMS. This is done using a DL approach and the results prove promising for two reasonably different use-cases. The learned models can recognize the various states of the systems and detect anomalies. The weakness of this approach is inherent in its nature: machines that do not run on electricity alone, such as pneumatic systems, cannot be monitored. This was demonstrated through the presented miniature industrial road in which the state of one section of the road could not be monitored. Hence, the question whether a high-resolution energy-data based model can serve as a multi-domain solution is positive but comes with a caveat. Nonetheless, if the monitored system is electrified, the developed models can be used for CM. Whether this can be extended to PM remains an open research question.

A second question was how strongly the system can be generalized to enable out-of-the-box use. Indeed, the developed approach simplified many aspects in particular by eradicating the need for feature engineering. Hence, no domain knowledge is needed for installation. Adding the limited amount of variables that can be measured per system further standardizes the models to work regardless of the use-case. However, the limitations entailed in limited variable scope have to be kept in mind as well.The only adjustments that have to be made on a case-to-case basis are adjusting the window function to fit the nature of the underlying data and adjusting the labeling process. The non-intrusive nature of data collection adds to the scalability of the system which further increases its applicability. Future extension of the model to include PMs may prove that domain knowledge is needed for this.

Although physically nonsensical, incorporating DC measurements by a AC sensor for the robotic arm use-case proved to increase the model's performance through its statistical value. Upgrading the EMS so that it can actually measure DC through Hall-effect sensors could further improve the performance and range of applicability. Another limitation is the system's current resolution. This could be improved by programming the data collection using a lower-level language capable of near real-time, and writing to a local instead of a remote DB.

Choosing convolutional ANNs resulted in effective classifiers. However, for the auto-encoders a hybrid network architecture resulted in superior performance as can be taken from Figure 6.1: the average reconstruction error is significantly lower, and the error distribution is more homogeneous. Consequently, the hybrid auto-encoder generalizes more effectively than the pure convolutional auto-encoder. However, this first step in improving the auto-encoder requires further investigation.

Hybrid model




Convolutional model

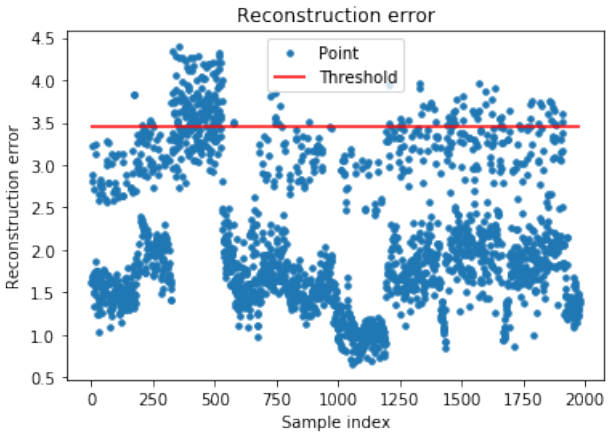



Table 6.1: Reconstruction errors (clean data): hybrid model (left) and purly convolutional model (right)

## 6.1 Outlook

Overall, the EMS in conjunction with the DL models developed in this thesis is an encouraging step toward domain agnostic CM. An important next step is answering the question "Can the system be used for PM?". So far, the results seem promising. Fairly minor changes to the auto-encoders could lead to first successes: for example, by monitoring the change in average reconstruction error over time could provide insights for maintenance decision. More elaborate adjustments are necessary for recognizing discontinuities in lower frequency patterns. The proposed approach of increasing the dilation rate may prove to be useful to this extent. Further, to be able to recognize discontinuities in patterns of low frequency more serious changes are necessary. Lastly, making the system real-time capable would also require appropriate infrastructure for the DL model.

# 7 Appendix

## 7.1 Basic Concepts

For a comprehensive understanding of this thesis' methodology and discussion, the following concepts and vocabulary should be familiar.

### 7.1.1 General Processes and Techniques

#### Condition monitoring

Condition Monitoring (CM) is the process of consistently monitoring one or more parameters of technical devices and machinery. The collected information is used for the identification and evaluation of running processes. [1]

#### Predictive maintenance

Predictive Maintenance (PM) uses the in the course of CM acquired data in combination with a wide variety of data analysis approaches to predict possible component failure respectively necessary maintenance. [14]

### 7.1.2 Electrical Basics

The used Energy Monitoring System (EMS) (3.1) is able to measure voltage and current as well as calculating effective voltage, effective current, power factor/phase shift, effective power and harmonic distortion. To be able to understand the analyzed data and the conclusion drawn from them it's important to understand the basic terms and concepts of electrical engineering.

#### Voltage

Voltage describes the potential energy of electrons between two points and the measuring unit of volt. Depending on the energy source voltage can either be constant (i.e. direct) or alternating. Since most consumers are built for alternating current and therefore alternating voltage the power supply system is based on it. But not every consumer uses alternating voltage. A flashlight for example uses constant voltage since a battery provides only direct current.
Most applications are covered in the range of low voltage 120V to ultra high voltage of 765,000V. The typical electrical socket provides 240V. Since many consumers are not in the need of the fully voltage provided they have build in transformers. The Voltage can be calculated by multiplying the current with the resistance. [15]

### Current

Electrical current is measured in the unit ampere and describes the flow of electrons in a conductor. A current within a circuit is induced by voltage. The number of electrons in a circuit stays constant. The flow of the electrons is the conversion of the potential energy of the voltage into kinetic energy. The current can be calculated dividing the voltage by the resistance.[15]

**Alternating Current** (AC) describes an electrical current which changes its direction periodically. This results in a sinusoidal waveform of the current when plotted against time. The phase of a wave describes the current position in a sequence of a periodic process. [15] In every day life AC is either single-phased or three-phased.
**Single-phased AC** is the result of a single current/wave. Generally it is used by the electrical consumers in a domestic household and therefore is standardized provided by the power outlet.
**Three-phased AC** is the result of three single phased alternating currents with a phase difference of 120 degrees each. Benefits of three-phased alternating current are e.g. that the phase currents tend to cancel each other out in the cases of linear balanced loads and its great suitability for long distance transport. The whole energy grid is built on three-phased AC and every household connection to the grid is three phased. Certain consumers in a household like cooking devices typically use three-phased alternating current due to higher efficiency and the ability to cope with load peaks.[15]

### Power

Electrical power is the rate of electrical energy transferred through a system per time unit. It is product of voltage and current. Power can be divided in active, reactive and apparent power.

**Active power** describes the power that is actually consumed or utilized by an AC circuit, resulting in a unidirectional energy flow.

$$Active\ power = voltage * current\ in\ phase\ with\ the\ voltage [15] \tag{7.1}$$

**Reactive power** is the power stored within the system. The voltage and current are 90 degrees out of phase resulting in the power flowing back and forth. Due to the energy flow always returning back to the source there is no net energy flow.

$$Reactive\ power = voltage * current\ out\ of\ phase\ with\ the\ voltage [15] \tag{7.2}$$

**Apparent power** is the result of the root mean square of the active and reactive power i.e. voltage and current. It is the power which has to be taken in account when modelling and running a power system, since although the reactive power technically is not consumed it still has to be provided.

$$Apparent\ power = voltage * current [15] \tag{7.3}$$

#### Power factor / Phase shift

The power factor is defined as the ration between real power and apparent power, describing how much of the provided power is actually put to use. A power factor of one is the result of a circuit containing purely restive loads i.e. active power is equal to apparent power, since the voltage and current reverse their polarity at the same time. For a power factor less than one the circuit contains inductive or capacity loads in need of reactive power. For inductive loads the current lags behind the voltage for up to 90 degrees and for conductive loads the current leads the voltage by up to 90 degrees. [15]
In an electric power system, a load with a low power factor draws more current than a load with a high power factor for the same amount of useful power transferred. Higher currents increase a systems energy loss and require e.g. larger wires. [15]

#### Total Harmonic Distortion

Non-linear loads or non-ideal devices create harmonics which disturb the sinusoidal wave form of AC power. The THD is the sum of the power of all harmonic distortions in ratio to the power of the fundamental frequency.

### 7.1.3 Machine Learning

ML is a subfield of Artificial Intelligence (AI) as shown in figure 7.1. The idea of ML is to train a program instead of explicitly programming it. The ML system learns on a provided data set to solve a specified task by coming up with rules based on statistical structure found within a training data set.[11]
In the context of computer science "learning" for a ML process can be defined as follows:

> A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. [10, Tom Mitchell, 1997 , p.20]

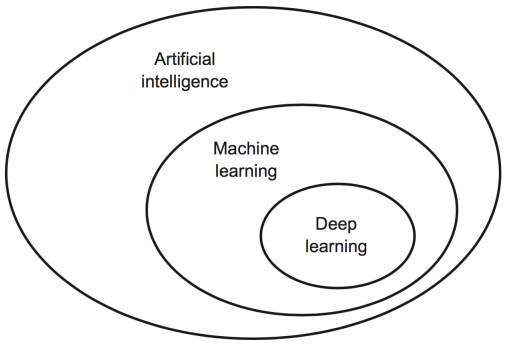



Figure 7.1: The relationship of artificial intelligence, machine learning, and deep learning [11, Francois Chollet, 2018, p.4]

ML projects can be categorized into two model types depending on the type of algorithm used. Model-based algorithms tune parameters of a model to fit the model to the training set.

Instance based algorithms learn the instances of a training set and use similarity measures to evaluate new instances. [11]

#### Data Cleaning

Discontinuities in the data during the measuring process can lead to outliers within a data set. Furthermore data instances can miss certain features. It is therefore necessary to prefilter the data to remove misleading instances and to fill or drop the instances or features with missing values. [10]

#### Feature Selection and Feature Engineering

FE represents the most crucial part of ML. For a ML algorithm to be able to make predictions or to categorize a new instance, based on the training set it learned on, the training set must contain a sufficient number of relevant features. It is therefore the task of the operator to select the most useful features and drop those which hold no relevant information (feature selection). Additionally he has to discretise continuous features, aggregate features to reduce input space and produce more useful ones as well as new features based on conclusions drawn from the data set (Feature Engineering).[10]

#### Training and testing data-set

Every ML algorithm lives on data. The more data is fed to the model the more sophisticated it becomes. However the performance of such models can only be evaluated through data which the model has not known so far. When training a model the data therefore always is split into a training data-set the model learns on and a testing data set to evaluate the model's performance after being trained. A typical rate used for splitting the data is 70/30, i.e. 70% training data-set and 30% testing data-set.

#### Batch

A batch is a fraction of a data-set. Batch learning means training a ML algorithm by feeding it one fraction (batch) of the data at the time. Models able to learn in batches need less computing resources and memory space. [10]

### 7.1.4 Deep Learning

DL is a sub-field of ML (Fig.: 7.1). It distinguishes itself from ML particularly through automating the process of FE, breaking a so far extensive multistage pipeline process down to a simple end-to-end model. [11] DL describes a learning process in which successive layers transform data instances into increasingly informative and meaningful representations. A layer's inner weights parameterize the transformation performed by a layer. The optimizer is continuously adjusting the inner weights via backpropagation following the result of the loss function. Training the layers (adjusting the weights) is typically done by a model called neural network.
Deep in the context of DL refers to the number of stacked layers.

Figure 7.2 shows the transformation of the image of a digit through successive layers of a deep network.

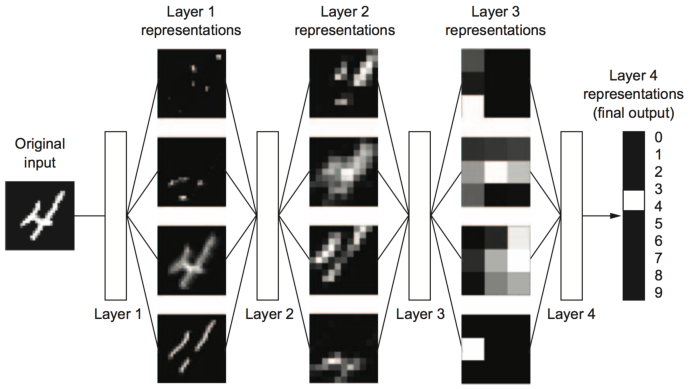



Figure 7.2: Inner representations of data by a CNN
[11, Francois Chollet, 2018, p.9]

The learning phase of a DL algorithm can be split into three significant steps, as depicted in 7.3 . First, the layers weights transform the input. The weights of a layer represent the values of the parameters of the mathematical conversions performed on the input. Secondly, the loss function evaluates the output. The loss function measures the performance of the network by comparing the network's prediction to the expected result. Thirdly the optimizer implements the so-called backpropagation algorithm to update the weights of the network's layers based on the loss function's feedback. [11]

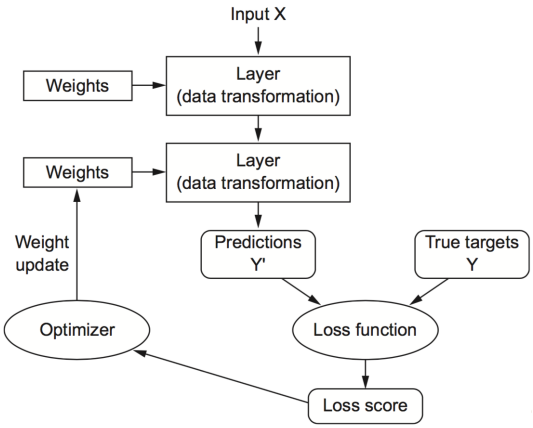



Figure 7.3: Mode of Operation of a deep learning algorithm
[11, Francois Chollet, 2018, p.10]

### Artificial Neural Networks

An Artificial Neural Network is typically the model of choice to train the layers that make up a deep learning model. ANNs get their names from remotely resembling the structure of the brain - at least in accordance to todays understanding of the brain. An ANN has an input

layer, one or more hidden layers and an output layer that represents the result: the input data transformed and mapped to some desired representation. Every layer is composed of several Artificial Neurons (ANs) which transform the data based on the weight assigned to them by the optimizer. A mesh of edges is interconnecting the layers. Each edge connects a pair of AN, transferring the output from a neuron $a$ to neuron $b$. In this case, $a$ is referred to as the predecessor and $b$ is called the successor. Whether a neuron's $a$ signal reaches through to neuron $b$ is in many implementations of ANNs dependent on a threshold - defined by the activation function - which the aggregated signal has to cross. *Fully connected layers* or *dense layers* are layers connecting all of their ANs to every AN of the previous layer. [16]

### Bias

For most implementations of ANNs the layers have a bias term additionally to the ANs' weights. The bias term is unaffected by the layers input but can be adjusted by the back-propagation algorithm which adds to the flexibility of the model.

### Gradient

A gradient is the derivative of a tensor or rather the generalization of a derivative of a function taking multidimensional input (tensors) by combining the parietal derivatives of the function's input variables. It is an operation performed on a scalar field which returns a vector field. The gradient at a certain point $P$ is always orthogonal to the level set at $P$ (it points in the direction of the greatest rate of increase) while the gradient's magnitude describes the slope of the level field at $P$. Function 7.4 shows the gradient of a scalar field of dimension $\phi(x; y; z)$. [17]

$$grad\ \phi = \frac{\partial \phi}{\partial x} * \vec{e}_x + \frac{\partial \phi}{\partial y} * \vec{e}_y + \frac{\partial \phi}{\partial z} * \vec{e}_z = \begin{pmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \\ \frac{\partial \phi}{\partial z} \end{pmatrix} \tag{7.4}$$

[17, Lothar Papula, 1997, p.61]

### Optimizer

The optimizer implements the backpropagation algorithm and updates the ANN's weights in accordance to the calculation of the loss-function via a form of Gradient Decent (GD).

### Backpropagation

The Backpropagation (Bp) algorithm represents the central algorithm of DL. Bp is based on the chain rule 7.5 which tells us that functions of the form f(g(x)) can be derived.

$$f(g(x)) = f'(g(x)) * g'(x) \tag{7.5}$$

Since the chain of operations of an ANN is differentiable via the chain rule, it is possible to map the parameters of an ANN to the loss functions gradient.

Propagating first forward through the ANN calculating the dot product of the input signals with the ANs weights and the bias terms and then applying an activation function we receive the ANN's chain of operations on the input. Then propagating backward through the ANN carrying the error terms the Bp algorithm calculates the contribution of the single parameters of each layer in the loss value.[10][11]

### Gradient Descent

How the optimizer adjusts the neuron's weights - the actual learning of the network- depends on the gradient descent of the loss. With the backpropagation algorithm computing the gradient of the loss with regard to the network's coefficients, it is possible to decrease the loss by just moving the coefficients in the opposite direction of the gradient.
The equation 7.6 depicts a single step performed by gradient descent. Theta ($\theta$) stands for the model's parameter and the nabla symbol ($\nabla$) represents the gradient operation on the loss function, depending on $\theta$. Eta ($\eta$) is called the learning rate, defining the pace with which the model is adjusting its parameters. [11]

$$\theta_{n+1} = \theta - \eta * \nabla_\theta F(\theta) \tag{7.6}$$

[10, Aurelien Geron, 2017, p.161]

### Activation Function

The activation function of a neuron decides if a neuron is to be activated or how strongly the neuron is to be activated. On the one hand, this depends on the input, and on the other hand on the shape of the activation function. The input of the activation function equals the weighted sum of input from the previous layer plus the bias.
A linear activation function or no activation function at all results in the output of the ANN always being a linear function of the input no matter how many layers the network has. It is therefore of great importance to choose an appropriate activation function to enable the model to reach nontrivial solutions.
Historically the first activation function used was a step function which does nothing else than defining a threshold for activating or disabling a neuron. For problems with more than two classes, a step function as activation function quickly meets it's limit - if more than one neuron of the output layer function fires the result is obscure. Today data scientists use a variety of different activation functions, which allow a finer granularity. Most commonly are functions of the sigmoid class. Sigmoid functions get their name from their characteristic "S"-shaped curve. The probably best-known sigmoid function is the logistic function depicted in figure 7.4.

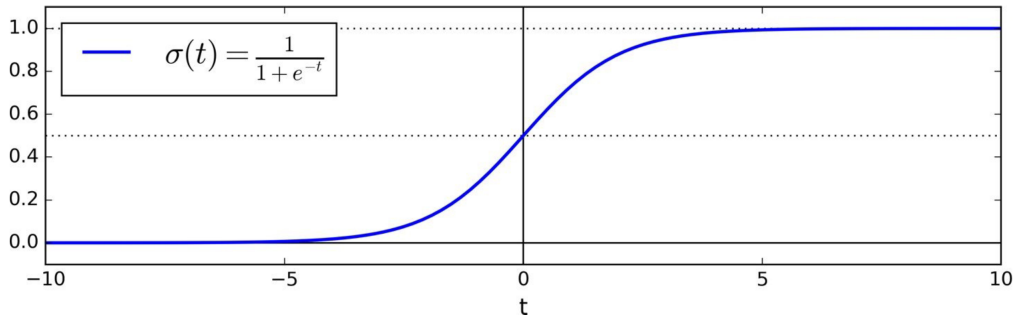



Figure 7.4: Logistic Function
[10, Aurelien Geron, 2017, p.187]

### Tensors

ML systems use as their fundamental mathematical structure tensors. Data feed to a DL model can have different dimensions. The dimensions do not only depend on the type of the data but also on its representation. The shape of the tensor defines the dimension of the data it contains. A tensor is a generalization of matrices - matrices being 2D tensors - for an arbitrary number of dimensions. Dimensions, in the context of tensors, are called axes. The number of variables per axis defines the axis' dimension. A 5D tensor has five axes with an arbitrary number of entries per axis or rather dimensions per axis. The *Shape* of a tensor describes its number of dimensions per axis, e.g. a tensor with shape (4,4,4) has three axes with four dimensions each.[10]

Typically time series data is stored in 3D tensors (a tensor with three axes). Vector data - 2D tensor with shape (timesteps, features) - combined with the number of samples adding up to a shape: (samples, timesteps, features).

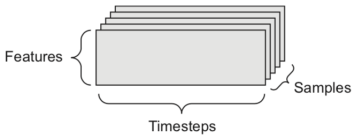



Figure 7.5: Tensor with three axes and shape: (samples, timesteps, features)
[11, Francois Chollet, 2018 , p.35]

Depending on the framework a tensor storing image data has typically four axes with a shape: (samples, height, width, channels) or (samples, channels, height, width). For example for a color image, the number of channels is three.

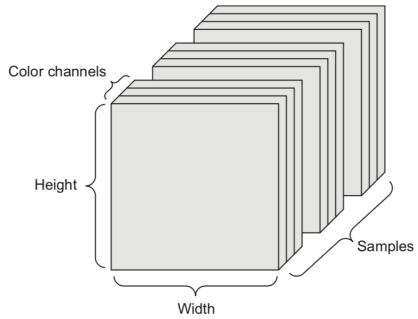



Figure 7.6: Tensor with four axes and shape: (samples, channels, heigth, width)
[11, Francois Chollet, 2018, p.36]

### 7.1.5 Convolution Neural Networks

CNNs are a particular class of ANNs. Although mainly used for computer vision, CNNs are highly effective for voice recognition or natural language processing. Research conducted by David H. Hubel and Torsten Wiesel in 1958-1959 gave great insides into the structure of the visual cortex creating the fundament for modern CNNs.[10] They discovered that many neurons which make up the visual cortex have a small local receptive field. A receptive field is the area of action an individual sensory neuron reacts to. The sum of the local receptive fields - which partly overlap - make up the whole visual field (in the case of the visual cortex). Additionally, they realized that while two neurons can have the same receptive field one may only react to images of horizontal lines where the other reacts only to vertical lines. Moreover, David H. Hubel and Torsten Wiesel recognized the hierarchical structure within the visual cortex of high-level neurons with larger receptive fields reacting to the more complex patterns of the combined output of several low level-neurons. Hence David H. Hubel's and Torsten Wiesel's research provided a concept for a compelling architecture for complex visual pattern recognition.[18] What makes CNNs so successful in today's industry is the use of partially connected layers. The traditional deep neural networks - which uses fully connected layers - quickly reach their limit when increasing image size or the number of variables.

> For example, a $100 \times 100$ image has 10,000 pixels, and if the first layer has just 1,000 neurons (which already severely restricts the amount of information transmitted to the next layer), this means a total of 10 million connections already on just the first layer. CNNs solve this problem using partially connected layers.
> [10, Aurelien Geron, 2017, p.472]

Convolution refers to a mathematical operation. The operation takes two functions, mirrors one of them and slides it over the other step by step calculating the resulting functions of the overlapping parts to produce a third function by thieir sum. Hence convolution in the sense of CNNs refers to splitting the input data into many step by step transformations were overlapping parts equal the local receptive fields of the single neurons.

### Structure of CNNs

At 7.1.4 of the review chapter the first abstraction of an ANN was shown to visualize the transformation of the data taking place within a deep learning model. For the parameter tuning of a CNN a far more extensive understanding of the different components of a CNN (as shown in 7.7) is needed.

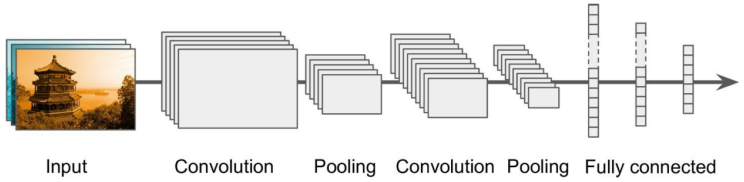



Figure 7.7: Typical CNN architecture
[10, Geron Aurelien, 2017, p.486]

### Convolutional Layer

Convolutional Layers (CL) represent the central piece of CNNs. The neurons of a CL only have connections to the data within their receptive field. Neurons of the input layer see just a fragment of the input data while neurons of following CLs - being connected to a limited number of the previous layer neurons- only see fragments of the transformed data of the previous CL . Every CL leads to a more meaningful transformation of the data. Low-level features extracted by the first hidden layers are merged to higher-level features by every additional CL.

The size of the receptive field of a neuron or rather the number of edges connected accordingly is a critical factor for performance and cost of a CNN. In a 2D CL of $i$ rows and $j$ columns - as shown in 7.8 - the neuron $n$ at $n_{ij}$ with a respected field of width $f_w$ and height $f_h$ is connected to the neurons in rows $i + f_h - 1$ and columns $j + f_w - 1$ of the previous layer. Usually, padding has to be added to fit the scale of the sum of the receptive fields of the layer neurons, to the scale of the previous layer. Most commonly rows and columns of zeros are being added to the previous layer, a process called *zero padding*.

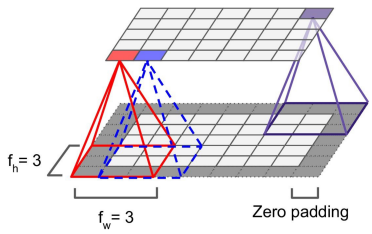



Figure 7.8: Receptive fields of a CL neurons and zero padding
[10, Geron Aurelien, 2017, p.375]

It is also possible to reduce dimension and cost by linking a large input-layer to a considerably smaller layer by introducing an offset between the neurons receptive fields. This is achieved by connecting the neurons in rows $i * s_h$ to rows $i + s_h + f_h - 1$ and the neurons of columns

$j * S_w$ to the neurons of columns $i + s_w + f_w - 1$ of the previous layer. The offset $s$ is called *stride*.

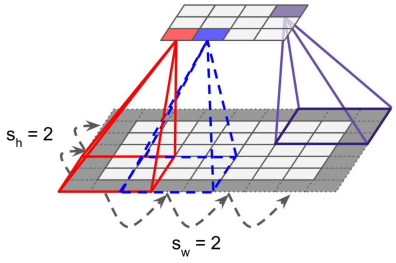



Figure 7.9: Dimensionality reduction through introducing strides
[10, Geron Aurelien, 2017, p.276]

The weight assigned to a neuron assigned by the optimizer via backpropagation (7.1.4) is effectively a filter for specific features. *Filter* or *convolution kernels* are the matrix operation of the receptive field of a neuron on the input data. Figure 7.10 shows the effect of two different filters. On the left side, the image was feed to a network whose every neuron had a vertical filter. A vertical filter means that the matrix representing the receptive field is full of zeros except for one column which is filled up with ones. The right picture was feed to a network whose every neuron had a horizontal filter. Horizontal filters respectively have zeros everywhere except for one row consisting of ones. Every neuron of a layer using the same filter leads to a representation of the input called *feature map*.

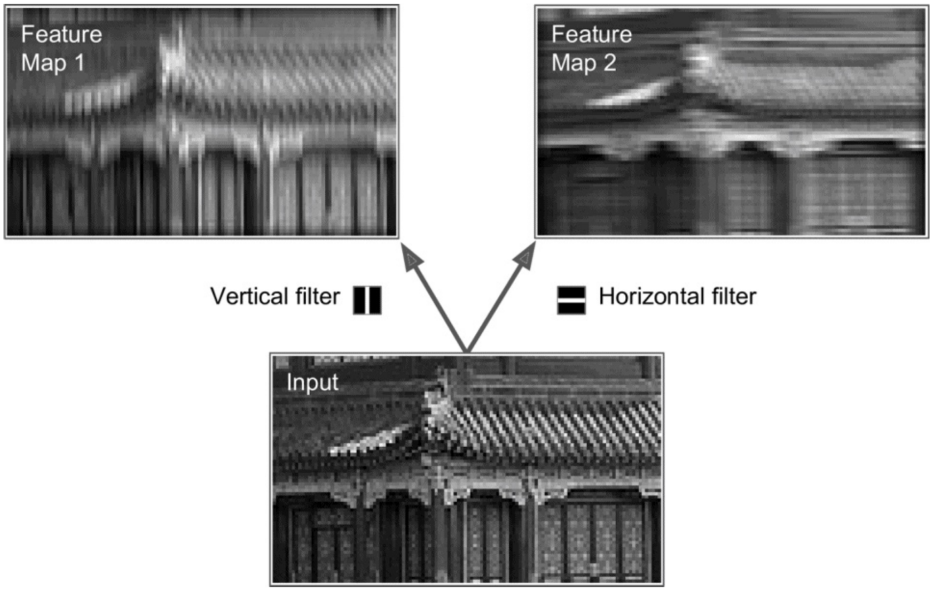



Figure 7.10: Feature map extraction through the filter of weighted neurons
[10, Geron Aurelien, 2017, 477]

During the training process the CNN searches for the most useful filters, combining these into intricate patterns. Effectively this means that a CL simultaneously applies a set of feature maps to the data to be able to recognize intricate patterns by extracting multiple features. Therefore, more accurately a CL is a 3D construct of multiple layers of feature maps applied by a 2D layer of neurons as depicted in figure 7.11.

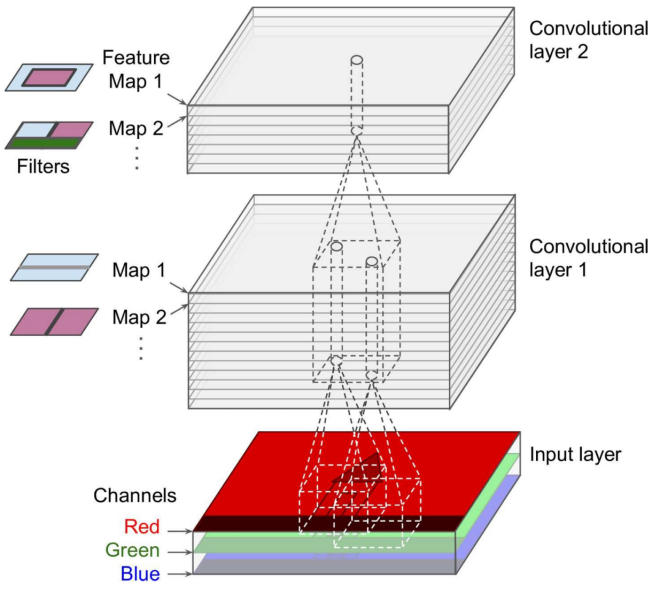



Figure 7.11: Multiple feature maps of CLs applied to an image with three channels [10, Geron Aurelien, 2017, p.479]

Equation 7.7 mathematically summarizes the previous explained. It calculates the output of a neuron for a single feature map of a single CL.

$$Zi,j,k = b_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k'=1}^{f_n} x_{i',j',k'} * w_{u,v,k',k} \quad with \begin{cases} i'=u.s_h+f_h-1 \\ j'=v.s_w+f_w-1 \end{cases} \tag{7.7}$$

[10, Geron Aurelien, 2017, p.480]

- $f_{n'}$: # of feature maps of previous layer (or # channels of input)
- $f_h$: height of receptive field
- $f_w$: width of receptive field
- $s_h$: vertical stride
- $s_w$: horizontal stride
- $u$: row of receptive field
- $v$: column of receptive field
- $k$: feature map of the processing CL
- $k$: feature map of the previous layer

$\boldsymbol{z_{i,j,k}}$ is the output of the neuron in the row $i$, column $j$ and feature map $k$ of the $l's$ convolutional layer.

$\boldsymbol{b_k}$ is the bias term of feature map $k$ in CL layer $l$

.

$\boldsymbol{x_{i',j',k'}}$ is the output of the neuron in the row $i'$, column $j'$ and feature map $k'$ of the $l-1's$ convolutional layer

.

$\boldsymbol{w_{u,v,k',k}}$ is the weight of the connection between the neuron of the feature map $k$ of layer $l$ and the input of feature map $k'$ of layer $l-1$.

### Pooling Layer

A Pooling Layer (PL) is an additional layer added to an ANN to reduce computational load by reducing the image size through subsampling the input. Similar to CLs the neurons of a PL have connections to the input within their receptive fields defined by size, stride and padding type. However, instead of applying weights to the input a PL's neurons aggregate the input. The most common aggregation function used by pooling layers are *max* or *mean* aggregation. A neuron implementing the mean aggregation function outputs the input divided by the number of neurons that make up its receptive field. Neurons implementing max aggregation output the highest value of the neurons within the receptive field as depicted in figure 7.12. The pooling layer is applied to every channel independently which is why the output depths stays the same.

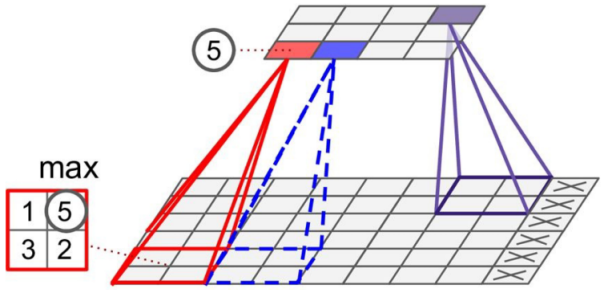



Figure 7.12: Max pooling with a $2 \times 2$ kernel, a stride of 2 and no padding
[10, Geron Aurelien, 2017, p.484]

### Dense Layer

A dense layer or fully connected layer is the building layer of a classical ANN. A neuron of a dense layer has connections to every unit of the previous layer. Within CNNs it is typically used as the final layer, performing the actual classification. The dense layer's task is to do the high-level reasoning by mapping the input of the previous layer to the desired output by reducing the inputs depths.

### Flattening

While the size of the data becomes considerably reduced when propagated through the network - justifying the use of a dense layer at the end of a CNN - it simultaneously gets deeper and deeper thanks to the CLs stacking ever more feature maps. The dense layers

dilatation rate = 0

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & 1 \\ 4 & 3 & 3 \end{bmatrix}$$

dilatation rate = 2

$$\begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 & 0 & 0 & 3 \end{bmatrix}$$

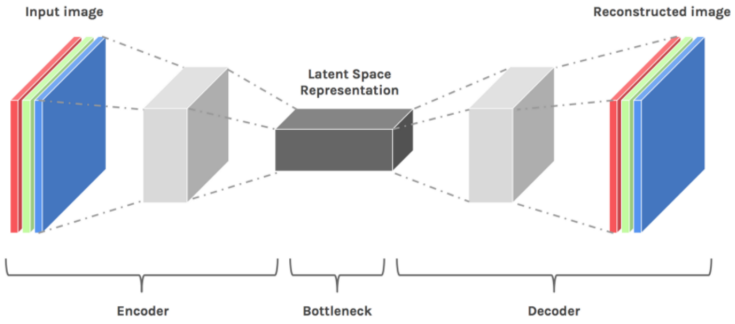
which function as classifiers need individual features for the classification. Therefore it is necessary to convert the output of the convolutional part of the model to a 1D feature vector. A 2D CL with 32 filters (number of feature maps) would return for every sample of a picture with three channels, a height and a width of 8 pixels and a feature tensor of shape: (32,8,8). The flattening application would then flatten the tensor containing the samples from (samples,32,8,8) to (samples, 2048). Thus returning a 1D array of 2048 features per sample.

#### Dilation rate

Defining a dilation rate greater zero is equal to putting a padding of zeros in between the 2D tensor (matrix) representation of a neuron's weight in its receptive field. The following example depicts a $3x3$ weight matrix with (right) and without (left) a dilation rate.
A dilation rate greater zero allows a neuron to have a greater receptive field while preserving input resolution and computational efficiency. It enables a network to better recognize patterns with low frequency. [5]

### 7.1.6 Autoencoder

Autoencoders are a particular type of self-supervised ANNs meaning that they are given as input and as required output the same set of data.
Figure 7.13 depicts the typical structure of an autoencoder. The encoder takes the input and compresses it to a low-dimensional latent space represented by the output of the layer with the smallest number of neurons. The decoder takes the encoded data as input and converts it back to the original input shape.[6]




Source: https://hackernoon.com/latent-space-visualization-deep-learning-bits-2-bd09a46920df, accessed: 03.09.2018

Figure 7.13: Architecture of an autoencoder

By introducing a bottleneck between encoder and decoder, the autoencoder is unable to map the input to the output directly. Instead the autoencoder functions as powerful feature extractor mapping the input data to the low-dimensional latent space by learning efficient representations of the input. [10]

## 7.2 Data Visualizations

### 7.2.1 Data Set of the industrial road

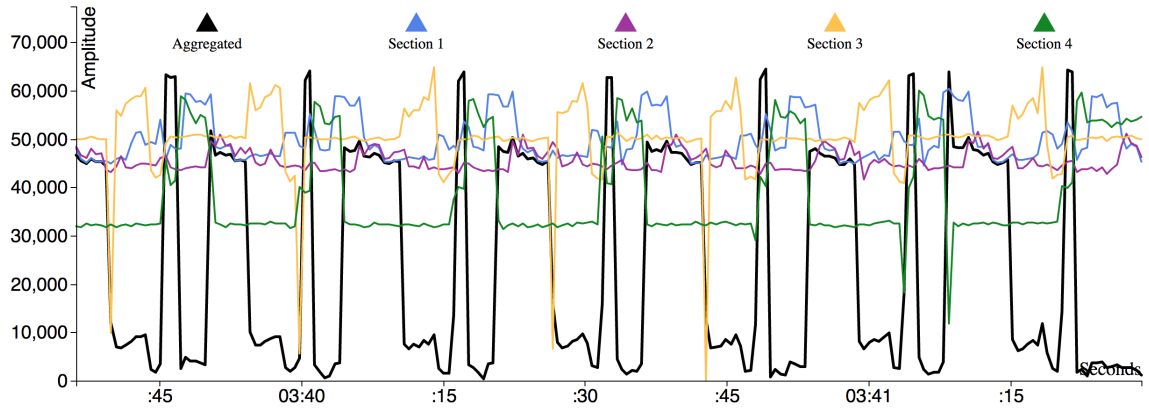



Figure 7.14: Phase shift introduced by the whole demonstrator and the single sections. Aggregated: black, section 1: blue, section 2: purple, section 3: yellow, section 4: green

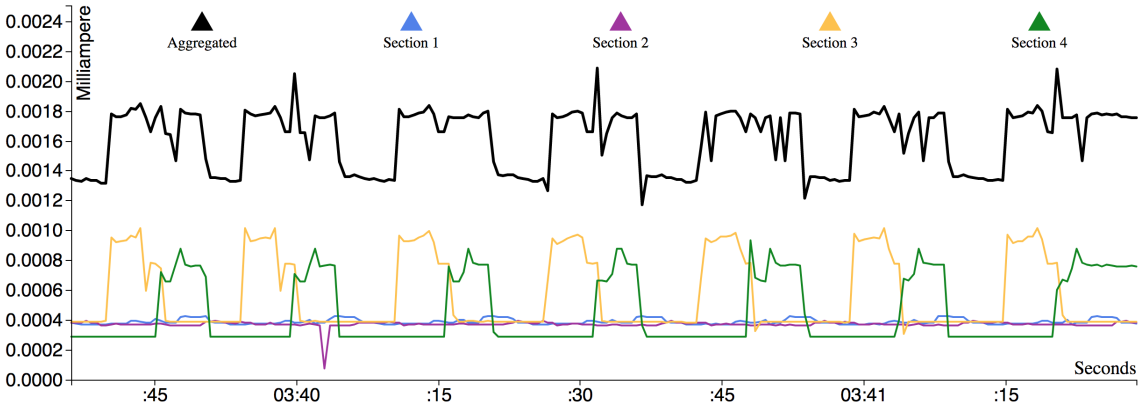



Figure 7.15: Current level of the whole demonstrator and the single sections. Aggregated: black, section 1: blue, section 2: purple, section 3: yellow, section 4: green

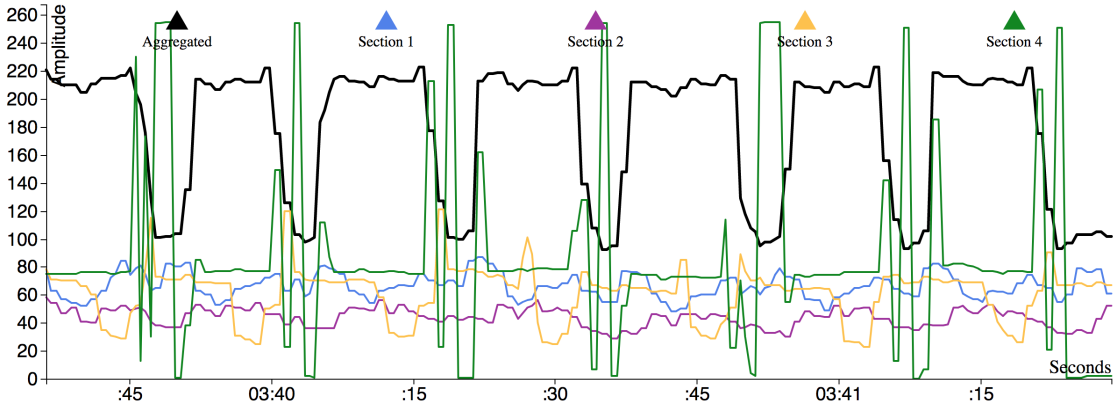



Figure 7.16: THD introduced by the whole demonstrator and the single sections. Aggregated: black, section 1: blue, section 2: purple, section 3: yellow, section 4: green

### 7.2.2 Data Set of the robotic arm

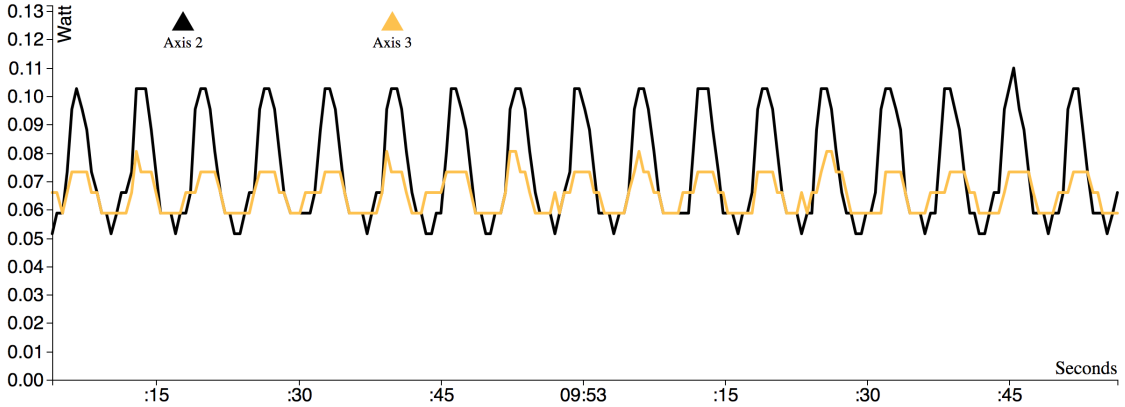



Figure 7.17: Power consumption of the whole demonstrator and the single sections. Aggregated: black, section 1: blue, section 2: purple, section 3: yellow, section 4: green

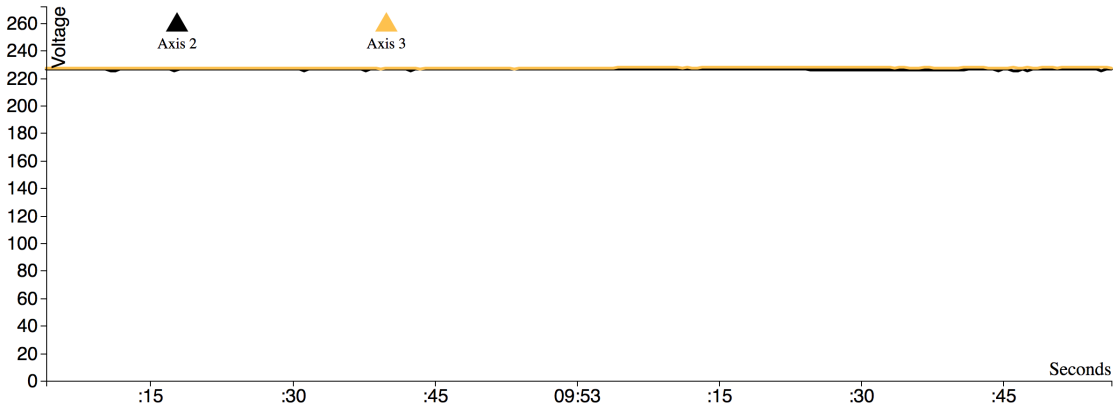



Figure 7.18: Voltage level of the second motor from the bottom and the third motor above it. Second motor: black, Third motor: yellow

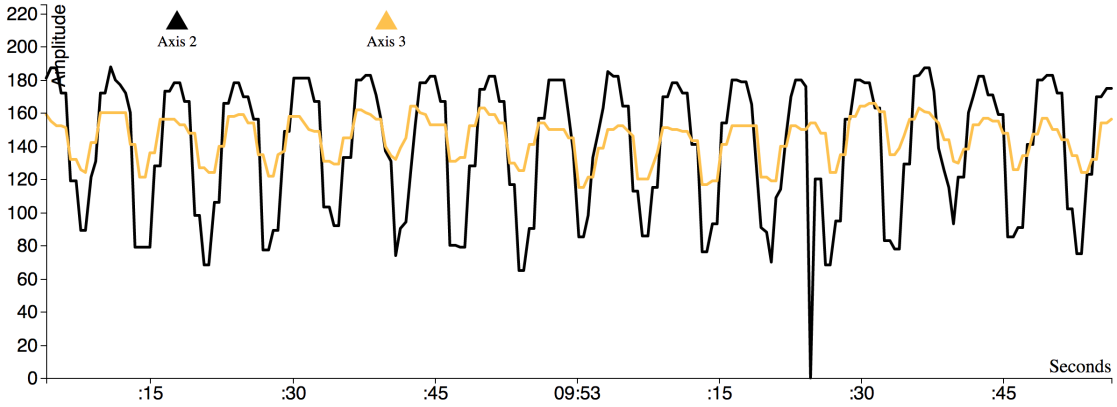



Figure 7.19: THD introduced by the second motor from the bottom and the third motor above it. Second motor: black, Third motor: yellow

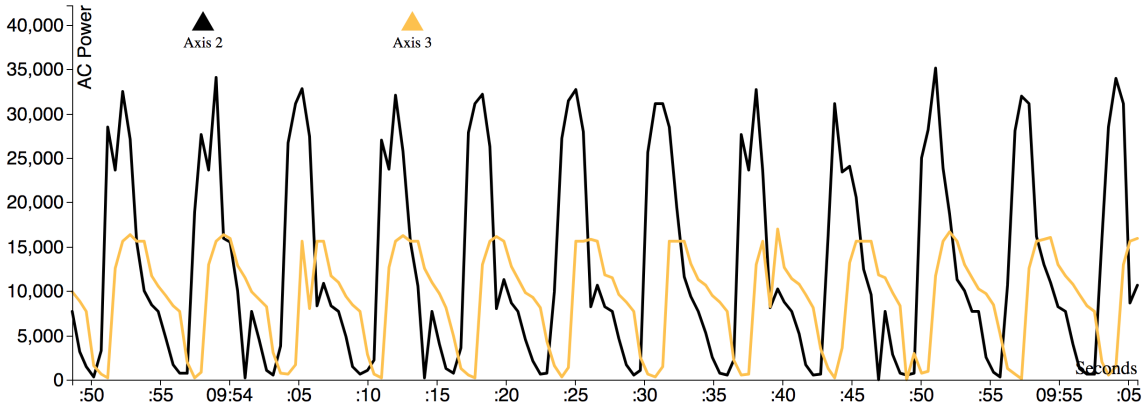



Figure 7.20: AC power consumption by the second motor from the bottom and the third motor above it. Second motor: black, third motor: yellow

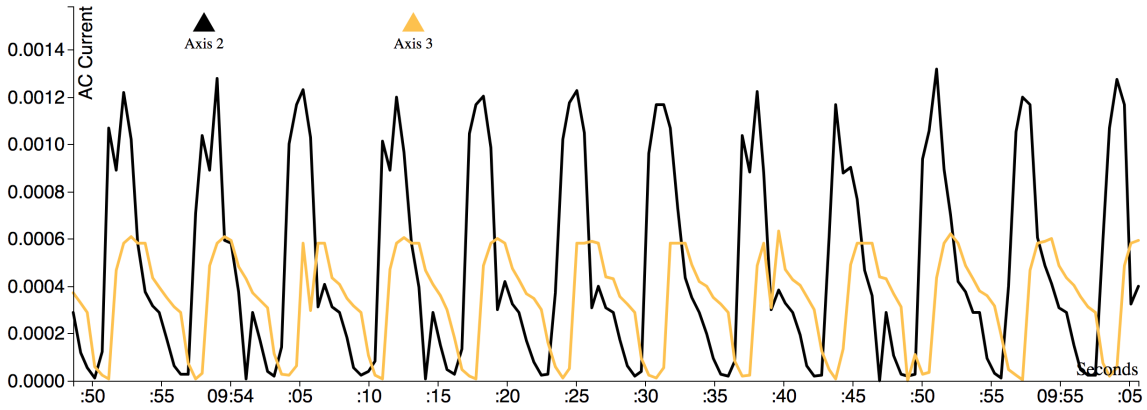



Figure 7.21: AC current as measured by the current sensors and introduced by the second motor from the bottom and the third motor above it. Second motor: black, third motor: yellow

# Bibliography


[1] Zhou, Zude ; Yao, Bitao ; Xu, Wenjun ; Wang, Lihui: Condition monitoring towards energy-efficient manufacturing: a review. In: International Journal of Advanced Manufacturing Technology 91 (2017), Nr. 9-12, 3395–3415. http://dx.doi.org/10.1007/s00170-017-0014-x. – DOI 10.1007/s00170–017–0014–x. – ISSN 14333015

[2] Marais, Henri-Jean ; Van Schoor, George ; Uren, Kenneth R.: ScienceDirect An Energy-based approach to condition monitoring of industrial processes. In: IFAC-PapersOnLine 4821 (2015), 772–777. http://dx.doi.org/10.1016/j.ifacol.2015.09.620. – DOI 10.1016/j.ifacol.2015.09.620

[3] In: Maier, Alexander ; Schriegel, Sebastian ; Niggemann, Oliver: Big Data and Machine Learning for the Smart Factory - Solutions for Condition Monitoring, Diagnosis and Optimization. Cham : Springer International Publishing, 2016. – ISBN 978–3–319–42559–7, 473–485

[4] Gebbe, Christian ; Tran, Christin ; Lingenfelser, Florian ; Glasschröder, Johannes ; Reinhart, Gunther: Feature Extraction and Classification of the Electric Current Signal of an Induction Motor for Condition Monitoring Purposes. In: Applied Mechanics and Materials 856 (2016), 244–251. http://dx.doi.org/10.4028/www.scientific.net/AMM.856.244. – DOI 10.4028/www.scientific.net/AMM.856.244. – ISBN 978–3–0357–1057–1

[5] Du, Shuyavng ; Pandey, Madhulima ; Xing, Cuiqun: Modeling approaches for time series forecasting and anomaly detection. http://cs229.stanford.edu/proj2017/final-reports/5244275.pdf. Version: 2017

[6] Kiran, B. ; Thomas, Dilip ; Parakkal, Ranjith ; Kiran, B. R. ; Thomas, Dilip M. ; Parakkal, Ranjith: An Overview of Deep Learning Based Methods for Unsupervised and Semi-Supervised Anomaly Detection in Videos. In: Journal of Imaging 4 (2018), feb, Nr. 2, 36. http://dx.doi.org/10.3390/jimaging4020036. – DOI 10.3390/jimaging4020036. – ISSN 2313–433X

[7] Görnitz, Nico ; Kloft, Marius ; Rieck, Konrad ; Brefeld, Ulf: Toward Supervised Anomaly Detection Marius Kloft / TU Berlin, TU Darmstadt, UNI Goettingen. Version: 2013. http://dx.doi.org/10.1613/jair.3623. 2013. – Forschungsbericht. – 235–262 S.

[8] Chalapathy, Raghavendra ; Menon, Krishna ; Chawla, Sanjay: Anomaly Detection using One-Class Neural Networks. In: CoRR abs/1802.0 (2018), 9. http://arxiv.org/abs/1802.06360

[9] McKinney, W ; Steele, Julie (Hrsg.) ; Blanchette, Meghan (Hrsg.): Python for data analysis. First Edit. Sebastopol : O'Reilly Media, 2012. – 466 S. – ISBN 978–1–449–31979–3

[10] Géron, Aurélien ; Tache, Nicole (Hrsg.) ; Adams, Nicholas (Hrsg.): Hands-on Machine Learning with Scikit-Learn& TensorFlow. First Edit. Sebastopol : O'Reilly Media, 2017. – 572 S. – ISBN 9781491962299

[11] Chollet, Francois ; Arritola, Toni (Hrsg.) ; Gaines, Jerry (Hrsg.) ; Dragosavljevic, Aleksandar (Hrsg.) ; Taylor, Tiffany (Hrsg.): Deep Learning with Python. First Edit. Shelter Island : Manning Publications Co., 2018. – 384 S. – ISBN 9781617294433

[12] Stahl, Karsten: emBRICK Products. https://www.embrick.de/. Version: 2018

[13] Tollervey, Nicholas H.: Programming with MicroPhython. In: Conant, Susan (Hrsg.) ; Jeff, Bleiel (Hrsg.): Programming with MicroPhython. First Edit. Sebastopol : O'Reilly, 2017. – ISBN 9781491972731, Kapitel GPIO - SPI, S. 105–108

[14] Jeschke, Sabrina ; Brecher, Christian ; Song, Houbin ; Rawat, Danda B.: Industrial Internet of Things Cybermanufacturing Systems. Version: 2016. http://dx.doi.org/10.1007/978-3-319-42559-7. In: Industrial Internet of Things Cybermanufacturing Systems. 2016. – DOI 10.1007/978–3–319–42559–7. – ISBN 978–3–319–42558–0, Kapitel 2, S. 709

[15] Wilfried Plassmann, Detlef Schulz, Dieter Conrads, Egon Döring , Peter Döring, Heribert G.: Handbuch der Elektrotechnik. Version: Fifth Edit, 2016. http://dx.doi.org/10.1007/978-3-658-07049-6. In: D, Reinhard (Hrsg.): Handbuch der Elektrotechnik. Fifth Edit. Wiesbaden : Teubner Verlag, 2016. – DOI 10.1007/978–3–658–07049–6. – ISBN 978–3–658–07048–9, Kapitel Grundlagen, 231–295

[16] Schmidhuber, J: Deep Learning. In: Scholarpedia 10 (2015), Nr. 11, 32832. http://dx.doi.org/10.4249/scholarpedia.32832. – DOI 10.4249/scholarpedia.32832

[17] Papula, Lothar: Mathematik für Ingenieure und Naturwissenschaftler Band 3. In: 7 (Hrsg.): Mathematik für Ingenieure und Naturwissenschaftler Band 3. Springer Vieweg, 1997. – ISBN 978–3–528–14937–6, Kapitel Gradient e, S. 61–65

[18] Hubel, D H.: Single unit activity in striate cortex of unrestrained cats. In: The Journal of Physiology 147 (1959), Nr. 2, 226–238. http://dx.doi.org/10.1113/jphysiol.1959.sp006238. – DOI 10.1113/jphysiol.1959.sp006238. – ISBN 0022–3751 (Print) 0022–3751 (Linking)