Evaluation of the Potential of Deep Learning for Manufacturing Process Analytics

Master Thesis

of

Elias Jan Hagendorfer

Chair of Automation
Montanuniversität Leoben

Supervisor: O.Univ.-Prof. Dipl.-Ing. Dr.techn. Paul O'Leary

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Affidavit

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

Leoben, 15.01.2018
Ort, Datum

Unterschrift

Abstract

This thesis investigates the use of deep learning for the automatic identification of machine operations from multivariate time-series data emanating from sensors and actuators. Methods from deep learning and time-series analysis are reviewed with the aim of determining their suitability. A new approach is introduced to alleviate weaknesses in current approaches which include insufficient signal selection, requirement of large amount of training data or neglection of the physical nature of the system. It consists of: a preprocessing methodology based around stationarity tests, redundancy analysis and entropy measures; a deep learning algorithm classifying time series segments into operation categories; a process analytics framework dealing with operation length and frequency. The approach was applied successfully to several datasets from heavy machinery bulk handling systems.

Kurzfassung

Die vorliegende Arbeit untersucht die Anwendung von Deep Learning zur automatischen Identifikation von Maschinenoperationen aus von Sensoren und Aktoren stammenden multivariaten Zeitreihen. Methoden des Deep Learning sowie der Zeitreihenanalyse werden analysiert um ihre Tauglichkeit festzustellen. Ein neuer Ansatz wird vorgestellt um die Schwächen in aktuellen Ansätzen zu beheben, welche unter anderem unzureichende Signalauswahl, den hohen Bedarf an Trainingsdaten sowie die Vernachlässigung der physikalischen Natur des Systems umfassen. Dieser besteht aus: einer Methodik zur Signalvorverarbeitung rund um Stationaritätstests, Redundanzanalyse und Messung der Entropie; einem Deep Learning Algorithmus, der Segmente von Zeitreihen in verschiedene Operationskategorien klassifiziert; einem Konzept zur Prozessanalytik rund um Länge und Häufigkeit der Operationen. Der Ansatz wurde erfolgreich auf mehrere Datensätze von Schwermaschinen zum Schüttgutumschlag angewandt.

Contents

List of Figures

List of Tables

1 Introduction

The manufacturing industry has been undergoing drastic changes in recent years. Technology innovation has lead to a situation where all the elements of production are entirely connected - machines, workers, equipment, resources, processes or even costumers. The industrial world has been disrupted and a new key production factor has been moved to the center of attention: data. Though the increased availability of manufacturing data through advances in sensor technology and storage systems can be seen as progressive, the potential of data as a production factor can only be unleashed by gathering insights into the production process from it. This can only be achieved by designing suitable data analytics approaches. Though a variety of applications is possible, the most work is done in the area of predictive maintenance. The area of process analytics, referring to generating a holistic analysis of a machines operational behaviour, is not explored with the same intensity.
Data analytics methodologies exist in a variety of domains. Analytics in the manufacturing domain has to deal with a number of special requirements and challenges. This is especially true for advanced methods, such as machine learning algorithms, that are forming the core element of many analytics approaches. Their successful application is mostly data dependent and manufacturing data is providing a rough environment in this regard.

Firstly, it is required that used methods have to be able to process high dimensional data within a remarkable timeframe as well as uncover dependencies within a process and between different processes, being represented by correlation and causality. Furthermore, data quality (e.g. name) or data composition (e.g. availability of labelled data) have to be considered and a high amount of redundant or irrelevant data can lead to a lower performance [93]. A respectable try to sum it up in one sentence from [56]: problems in manufacturing are "data rich but information poor".
Another noteable feature of manufacturing data is resulting from its origin, as it is sensor data in most cases and therefore has a temporal component. Each measurement results in a specific value at a specific point in time and falls into the category of time series data. This data type comes with a number of characteristics. Summed up in [46], reaching from the need to have a certain ratio between the amount of data and system complexity and an explicit dependency on time to non stationarity and invariance.
A further requirement is that sensor data analytics needs to correctly include the inevitable causalities lying within the data through the underlying physical process [63].

In spite of all the difficuties, there are clear implications that machine learning methods are potentially filling a gap in such applications, being the high amount of successful examples in manufacturing and obvious intersections of its strengths with the requirements of the domain. Though a generalization to all algorithmus can not be made, ML is handling high dimensional datasets relatively well. Furthermore, the fact that ML is using past data to understand the problem allows proper handling of dynamic systems, as adjustments are made to changing behaviour that would be hard for a deterministic model. Some of the mentioned challenges and problems might have had a fostering effect on the rise of a novel machine learning technique. Deep Learning, a set of methods empowering models consisting of multiple layers learning representations from data with an increased level of abstraction [48], have shown great results on several important machine

learning problems and therefore attracted a considerable amount of attention. One of the reasons might be that classic machine learning approaches can not be applied to raw data easily [48]. Taking a classification task as an example, the algorithm is in need for so called "features" to tell him what characterises the data with respect to the classes.
This can be quite problematic, as the construction of such features (called "feature engineering") is a very time consuming step that often requires a priori knowledge and feature quality is crucial for the learning phase [1].

1.1 Motivation

Nature has been an inspiration for computer science several times, as with deep learning and the human brain. As humans are able to analyze sensory information by letting it propagate through a hierarchy of layers which basically learns representations from it without preprocessing [1]. As mentioned in [5], perception with regard to sensor data can be seen as assigning a meaning and a semantic representation to sensory observation. This seems to be an interesting starting point to lead over to the need for new data analytics approach.

Predictive maintenance, failure detection and process optimization require automated process analysis. A key enabler for all these is a human understandable, event based representation methodology which segments the production process of a machine into its steps. There are several existing techniques that are able to overcome individual parts of the challenge posed, but can not solve it in total. Concerning the selection of the most promising variables from a high number of sensor dimensions, concepts like stationarity, entropy or correlation have been considered before, but mostly individually. A comprehensive approach to select the optimal set of signals with respect to all of them has not been presented yet. Symbolic representations are able to represent sensor data and provide its information with reduced dimensionality, but require a priori concerning the data. Well established machine learning approaches are able to classify time series, but often require large sets of labelled training data. This again requires knowledge to enable the definition of classes (e.g. machine operations) and a manual labelling step. In addition, as complex machine operations are formed by a simple suboperations, the length of time series segments processed hugely matters. Current literature does not cover this adequately.

1.2 Goals

This situation motivates an evaluation of the potential of deep learning methods in manufacturing process analytics. This is carried out via analyzing the specific requirements of this application and matching it with what an algorithm can actually provide, including statistical analysis and preprocessing that can be combined with deep learning.
In concrete, the goal of this work is to provide a thorough literature review on time series preprocessing methods relevant for the specific features of the dataset as well as deep learning models applicable to this type of task. The lessons learned from this review will be used to select the most suitable methods and construct a novel approach for sensor data analytics from it.
In detail, the following shall be achieved:

- A discussion of the strengths and weaknesses as well as successful applications of

statistical preprocessing methods and deep learning models.

- The conception of a preprocessing methodology capable of signal inspection and selection. Increasing applicability and performance of machine learning algorithms as well data insights shall be achieved. The focus will be the removal of redundant, information poor or random data as well as normalization.
- A deep learning algorithm able to classify machine operations from time series data. An ideal combination of a deep architecture and a training algorithm is selected, whereby the feature extraction capabilites of deep methods are of special interest. The limited use of training data plays an important role.
- A framework for aggregating the results into useful information on the manufacturing process.

1.3 Organization

This work is structured as follows: Part 2 reviews statistical methods for preprocessing time series data and tries to come up with recommendations while Part 3 reviews different deep learning models including their architecture, training process and way of application. Part 4 introduces the analytics approach introduced in this work and Part 5 evaluates the approach on machine sensor data. Part 6 concludes this thesis and proposes possible directions for future work.

2 Statistical Preprocessing of Time Series Data

The purpose of the following section is to introduce a theoretical background for the preprocessing tasks considered necessary in the course of this work. This is relevant as a preprocessing algorithm will form an important part of the analytics approach. Reviewing different methods mainly follows the purpose of enabling and supporting an appropriate selection. The areas covered were defined by basic assumptions about the data and what statistical properties and methods can do to support analysis. Their main contribution is related to either improving performance of algorithms, guaranteeing model validity and reducing dimensionality.
For all the four preprocessing aspects considered in the following, it starts with identifying methods that are considered capable of performing the required tasks. Moreover, literature is reviewed for possible method selection or short-listing via uncovering strengths and weaknesses and draw possible conclusions for sensor data classification. The focus lies on possible successes of a method on similar problems, data sets or domain of applications. In the ideal case, works that provide a valid comparison can be referenced.
Three of the four aspects deal with eliminating or selecting signals with respect to key features while normalization is performing an actual transformation on the data. The latter is assuring a common scale which is beneficial for further steps. Stationarity assessment is verifying the presence of an important property which is a prerequisite for any model trained on the data to be valid. Redundancy analysis is attempting to identify time series that contain redundant information to reduce data dimensionality. Lastly, signals shall be compared with respect to their "interestingness", which means presence of patterns and probability of learn characteristics of the process form them. This idea is best covered by methods from the domain of time series complexity measurement [83].
It needs to be mentioned that although the specifics of multivariate time series data are of huge importance for the overall approach, the signals will be preprocessed individually. The capability of the methods to handle MTS is therefore not a factor in their evaluation.

2.1 Normalization

In the following, different possibilities for normalizing time series are presented, including their drawbacks and limitations. The purpose of this section is to select a suitable normalization technique for a manufacturing analytics approach. This is done via analyzing their essential properties and reviewing literature where comparisons have been made. The following aspects are considered especially important:

1. Handling of time series data properties (Stationarity, Volatility, Outliers, Distribution)

2. Effects on performance of machine learning algorithms

Simply explained, normalization takes given values of a specific attribute and then adjusts them so that they fit into a defined range. As formulated in [67], these techniques take a variable X and replace it by any function of it (square root, mean, divided by etc.). The authors also state that normalization becomes more important when data has attributes of varying units and with different scales, as data mining algorithms rely on comparing data points and this only works if they are proportioned. As mentioned, this argument is highly relevant in manufacturing as sensor systems lead to that exact situation: high variations in scale across different dimensions. As it will be pointed out in the course of this

section, time series data is fraught with some other perils when it comes to normalization. Two important properties of time series shall be outlined here for a better understanding, as they will be mentioned during explanation and discussion of normalization methods.

An often discussed feature in time series analysis is stationarity. Due to its complexity and relevance, it will be elaborated in a seperate section.

Another property is uniform and non-uniform volatility, respectively. In contrast to stationarity, volatility does lack a consistent statistical definition. Intuitively speaking, it is describing the degree of fluctuation within data. An illustration might help to imagine how a time series with non uniform volatility might look like (see Figure 1). According to the previously made definition, this time series is also non stationary, as variance is varying over time. Although it is true that volatility can not be computed without a solid definition, the idea behind it allows the intuition that time series collected by sensors in manufacturing will often have changes in volatility from one segment to another.

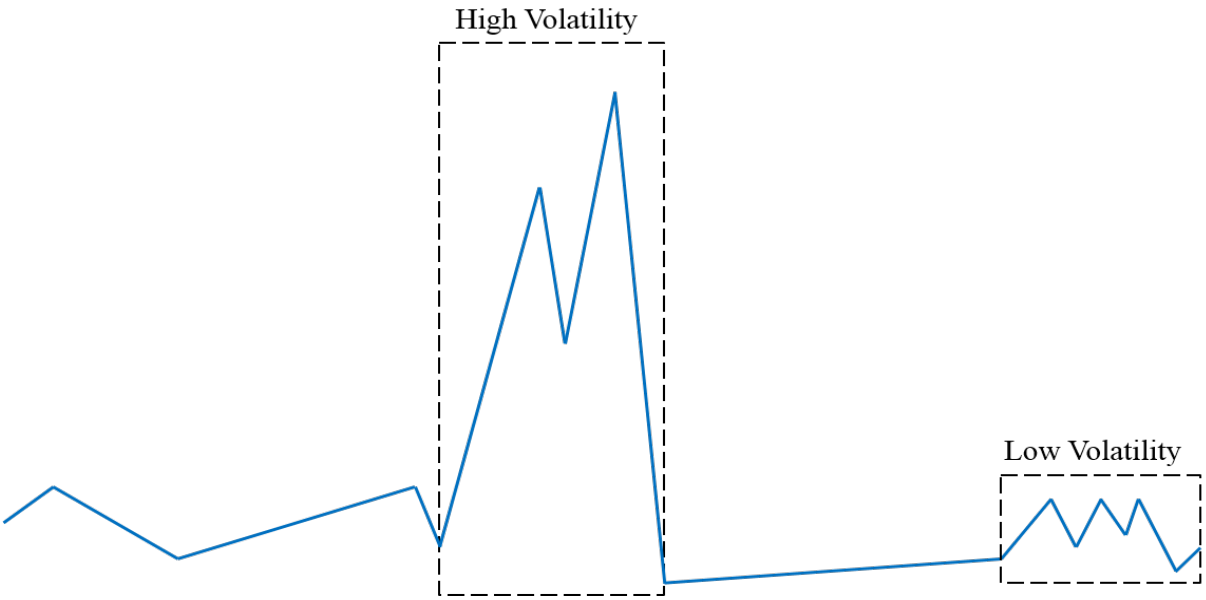


Figure 1: Time series with non-uniform volatility

Almost any work done in machine learning states that data quality is crucial for success and the importance of preprocessing is mutually agreed upon. Now, this alone does not provide a justification for data normalization.
Nayak et al. (2014) [73] mentioned a few positive effects of placing normalization techniques in front of machine learning, stating that scaling the input features is reducing bias within a neural network and that training time can be reduced as the training process can start in a similar range for each feature. In [70], the authors are reminding fellow researchers of the inevitability of normalization in time series analysis to produce meaningful results. An example from video analysis is provided, illustrating that small changes in scale or offset are significantly increasing the classification error if normalization is not used.
As a next step, the meaning of normalizing data in problems from an industrial environment shall be pointed out. Sola & Sevilla (1997)[82] are applying a multilayer perceptron to several variables measured in a nuclear power plant. Their experiments lead to two

interesting observations: Firstly, the error rates decrease with reduced difference in variation of variable. It is argued that this is caused by the fact that all variables are equally influencing the output error as they are in a similar range. Secondly, there is a threshold in the iteration number at which the error starts rising again, which seems to depend on the normalization method. This is explained by the over-fitting effect and the downsized searching space, which reduces the effort for the backpropagation algorithm.
In the following, a selection of normalization techniques shall be discussed. The three most common methods, frequently appearing in work on comparing them (such as [60]) are Min-Max normalization, Z-score normalization and decimal point normalization, which can be found in [26]. Median normalization and sigmoid normalization can be found in [73].

Min-Max Normalization.
Min-Max normalization is defined as:

$$x_n = \frac{x_0 - X_{min}}{X_{max} - X_{min}}. \tag{1}$$

Where the new value of variable x, x_n, is calculated from the current value x_0 and the minimum and maximum value for the variable in the dataset, X_{min} and X_{max}. The variable can be mapped into a defined range. While the above definition only works for the range $[0, 1]$, other ranges as $[-1, 1]$ can be achieved by using [73]:

$$x_n = x_{lb} + \frac{(x_{ub} - x_{lb})(x_0 - X_{min})}{X_{max} - X_{min}} \tag{2}$$

where x_{ub} and x_{lb} are the upper and lower bounds defining the range.
In [62], the authors state that the min-max normalization faces the problem of not knowing if future values are lying within the minimum and maximum values of a sample. They mention two possibilities to overcome this issue:

- Setting every out-of-bounds value to X_{min} or X_{max}. This might lead to a significant information loss, as lower and higher values can not be considered less important just because they are out of range for earlier data and might even be characteristic for the dataset. In addition, it might very well be the case that a lot of future data is out of bounds. This would lead to a high number of samples being set to exactly a boundary value which results in a concentration that is not existing before normalization. The negative effect of this issue on machine learning performance is validated in [82].
- Using sliding window technique (for example in [85]) normalizing a window of certain length on its own. The necessary assumption of uniform volatility might not always hold up.

Z-Score Normalization.
This method is computed as:

$$x_n = \frac{x_0 - \bar{X}}{\sigma_X}. \tag{3}$$

Here, the specific value x_n of a certain attribute X is normalized to x_n, while $\bar{X}$ and σ_X are the mean and the standard deviation of that attribute. It can be concluded intuitively and is mentioned correctly in [34] that this leads to data with zero mean and unit variance. In terms of properties of this method, the authors stated that z-score normalization is

not depending on knowing the overall minimum and maximum of the dataset and is able to reduce the effect of outliers. Both are valuable assets in normalizing time series sensor data. On the negative side, the method is limited to stationary environments where mean and variance do not change over time.

Decimal Scaling Normalization.
Decimal normalization is computed as:

$$x_n = \frac{x_0}{10^c} \tag{4}$$

whereby c must be chosen so that $max|X_n| < 1$. Decimal scaling has the already mentioned problem of being dependent on knowing maximum values (the number of decimal points moved depends on it).

Median Normalization.
Each sample is divided by the median, represented by $\widetilde{X}$, for all samples:

$$x_n = \frac{xi}{\widetilde{X}}. \tag{5}$$

As assessed by [73], this method benefits from the fact that the median is not affected by the magnitude of extreme deviations in the data, as the mean would be.

Sigmoid Normalization.
A sigmoid function has to be chosen to perform this normalization:

$$x_n = \frac{\mathrm{e}^x - \mathrm{e}^{-x}}{\mathrm{e}^x + \mathrm{e}^{-x}} \tag{6}$$

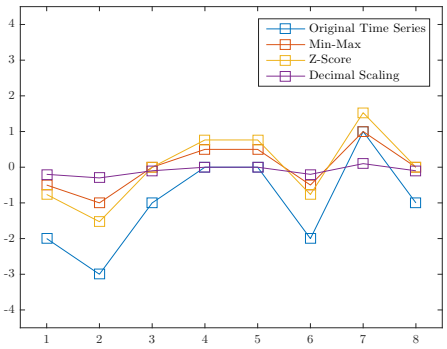
Sigmoid normalization does not depend on the distribution of the data [73].
Being interested in the interaction of normalization techniques and machine learning algorithms, [60] provide a study involving the three methods above. Being tested together with a least squares support vector machine (LS-SVM) on a prediction task, decimal point normalization is achieving the best results in terms of mean squared error and accuracy, followed by min-max and Z-Score. Unfortunately this is accompanied by the highest computational time, min-max having the lowest. The authors repeated the experiment with neural networks, receiving similar results.
A similar evaluation has been done by [81], who used normalization in the course of univariate time series forecasting with evolutionary ANN. Their results show that vector normalization led to the highest forecast accuracy on the test set, stated for the sake of completeness:

$$x_n = \frac{x_0}{\sqrt{\sum_{j=1}^{k} x_j^2}} \tag{7}$$

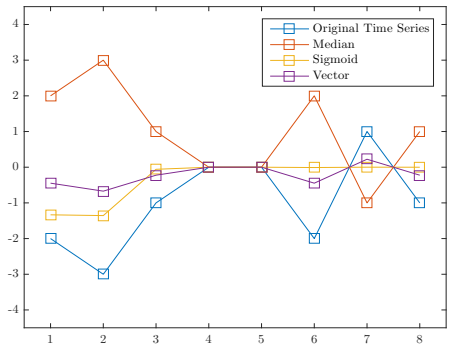
On the training and validation set, min-max normalization delivered the best result.
In [34], the authors show that the classification performance of a feed forward back propagation neural network is depending on the used normalization method. In their experiment, min-max normalization slightly outperforms Z-Score and Sigmoid normalization.
[73] provide an evaluation of normalization techniques on four different time series forecasting models based on artificial neural networks. On average, min-max, Z-score and decimal point normalization performed poorly while sigmoid normalization lead to a very

high forecasting accuracy.
In [16] min-max and Z-score normalization are evaluated. In terms of time series forecasting accuracy. The methods acted as a preprocessor for MLP and other algorithms, whereby min-max (linear scaling) had the better performance. More interesting is the fact that accuracy was varying with different scales results for linear scaling. Variation in post-normalization range can therefore be seen as a possible adjustment with respect to an ideal analytics approach.

Other conclusions on relevant method particularities can be drawn from a visualization of normalization results for a simple time series example (see Figure 2a and 2b).



(a) Example comparison of min-max, Z-Score and decimal point normalization



(b) Example comparison of median, sigmoid and vector normalization

Figure 2: Visual analysis of normalization methods

Further analysis is heavily relying on the presence of characteristic patterns within the time series. The possibility of identifying those with machine learning shall not be negatively affected by changing the shape of the series. It needs to be pointed out that though the time series look smoothed after normalization, the relative distances between individual data points have not changed. Looking at figure 2a, both methods are preserving the shape of the original time series, with Z-Score normalization coming a bit closer. Apart from that, it is notable that data points of value 0 are moved in both methods. This seems problematic in a sensor data environment, as zeros represent a special causality in the respective system (e.g. no activity).
This is different for the methods visualized in figure 2b, as zeros remain unchanged. The flattened shape which is mainly a visual effect as the values are set to a smaller range. This random example also reveals a possible effect of median normalization, which is inverting the signal in case of a median of -1.

Although this multi-perspective analysis revealed some interesting aspects on time series, it does not allow an unambiguous conclusion on a preferable method. That was to be expected, as this is obviously depending on data and algorithms and both factors are varying within the analysed studies. Still, it remains to be noted that min-max normalization preserves the shape of the original time series and is flexible on normalization ranges, which puts it on a short-list for practical applications.

2.2 Stationarity Assessment

As it has already been mentioned in the introduction, time series stationarity has clear relevance when performing analytics on it. In concrete, the effect of stationarity on applicability and validity of results of certain normalization techniques has been mentioned. In this part, the need to consider this property in the context of this work is explained on a broader basis. To present this explanation in an understandable manner, it is required to introduce the concept of stationarity first.
As a time series is a series of observations, there is always some kind of process generating this observations. A stationary time series is therefore originating from a stationary process, characterised through the fact that its stochastic properties are not changing over time [61]. In formal definitions, it is distinguished between [61]:

1. A strictly stationary process, meaning that given time points $t_1, ..., t_n$ and random variables $X(t_1), ..., X(t_n)$, the joint distribution of $X(t_1), ..., X(t_n)$ is the same as for $X(t_{1+\tau}), ..., X(t_{n+\tau})$. The definition holds for all n and τ and means that all moments of all degrees are the same throughout the entire process. Furthermore, the joint distribution of two variables cannot depend on their time instance, but only on the interval between them.

2. A process of second order or weak stationarity, limiting the definition to mean and variance (independent from t) and autocovariance (only depends on lag τ)

To expect that the assumption of strict stationarity holds for a dataset is asking a lot, especially if it is collected over a longer period of time. We may actually be seeking points in the data who are non stationary. These may be points in time where causal events occur. Most work on the analysis of time series stationarity with regard to practical applications is therefore focusing on weak stationarity. As the purpose of this section is to introduce stationarity as an interesting property for applying machine learning on sensor data time series, the term stationarity will be used for the latter.
To rephrase the above given definition, the vector $(x_1, ..., x_n)$ and its time shifted counterpart $(x_{1+k}, ..., x_{n+k})$ have the same mean vectors and covariance matrix, holding for every integer k and every positive integer n [14]. In [14], the author lists several elementary characteristics of strictly stationary time series from which some can be used here to further reinforce the distinction to weak stationarity:

1. Random variables from a strictly stationary time series are identically distributed

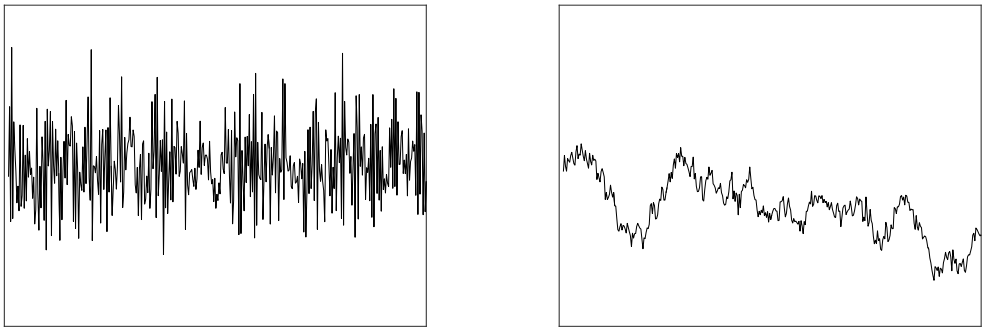
2. Strict stationarity is given for sequences which are *iid* (independent identically distributed).

3. A time series for which weak stationarity has been assessed is not guaranteed to be strictly stationary

For the third one, it is noted that the generalization of stationarity is possible the other way round. This trivial conclusion can be drawn from the fact that if all statistical properties are constant over time, this has to be true for mean, variance and autocorrelation, which are relevant for second order stationarity.

Having introduced the concept of stationarity, the statistical properties are not sufficient to make the necessity of its consideration obvious. This can only be achieved by

linking the concept with relevant, practical issues.
Modelling approaches for univariate as well as multivariate time series often include a stationarity assumption, questioning its usability for non-stationary data and making stationarity testing necessary. Although this property is causing inconvenience, generalizing any model by relaxating the assumption of stationarity would increase its size and its number of parameters [35] reducing its ability to solve real world problems. A quite general interpretation of stationarity in the context of data modelling might be that in its absence, the accuracy of a model is time-dependent, which is of huge importance. In [61], an interesting example is brought up, explaining the problem with applying the problem to non stationary data generated by a process which is evolving over time. Considering $X_t = \sigma_t Z_t$ as a time series model. In case of non-stationarity σ_t could change with every time step. One has to estimate the quantity σ_t only using one single data point (at time t), which will usually be not very successful.
The same idea can be thought through with the help of an illustration (see Figure 3). The data was generated using an stationary AR(2) process and a non stationary unit root process. In the stationary example, it is visible that the mean and variance of any segment of the time series could be used to represent the entire series, whereas this would fail in the non stationary case.



(a) Time series fulfilling stationarity - AR(2) process

(b) Time series not fulfilling stationarity - unit root process

Figure 3: Concept of stationarity for time series data

With some adjustments, the previous arguments concerning models in general can be extended to machine learning algorithms and their issues with non stationary data. In detail, it is sufficient to rephrase the above ideas, as it has been done by [86], who are summing up the evolving challenges in two points: What has been learned by the algorithm from past data is not automatically useful for future samples. Furthermore, training data is not always a suitable preparation for the test phase, if the data is not iid and stationary. Ways of overcoming these problems might be to repeatedly train the model over again on a window of the dataset or combining different models, each focused on a certain part.

Lastly, the relevance of stationarity to manufacturing sensor data is pointed out. Though processes are frequent sources of manufacturing signals [88], several explanations for the occurrence of non stationary times series in this domain are given in [9]:

1. Changes in the data generating phenomena

2. Sensor malfunction

3. Modification of sensor location

An additional possibility that comes to mind is that wear and tear in machine components reflects as a trend in the respective measurements.

Two other statistical properties which are often mentioned in the same context as stationarity are ergodicity and randomness. They are dealt with here as connections between the concepts can be identified. As with stationarity, ergodicity and randomness are making a statement about the data generating process and are relevant for analytics. Interdependencies can be drawn formally and intuitively.
According to [89], an ergodic process is one where the existence of one of its realizations can be used to derive its statistical properties. As stationarity, ergodicity is independently defined per property, indicated by the terms mean ergodicity and covariance ergodicity. This can be enhanced to time series, done in [15]. The mean of a single time series realized from a mean stationary time series model is required to converge to the ensemble mean with increasing series length:

$$\lim_{n\to\infty} \frac{\sum X_t}{n} = \mu \tag{8}$$

Per this definition alone, the relevance of ergodicity in a time series and manufacturing context might not be obvious. Firstly, time series data is not independent, which is one of the characteristics making it challenging. Ergodicity intuitively stands for the independence of two relatively far away observations in the series. Furthermore, time series data from the economic area can be described as a single instance of a hypothetical model. This is contrasted to technical experiments, where each run can yield to varying results (caused by e.g. slightly changed starting conditions) and is represented by a time series in the ensemble [15].
The term hypothetical can be interpreted as a reference to economical data not being generated by an identifiable model. It is therefore impossible to state that two economical series are realizations of a common model. This is different with respect to an experimental setup.
Transferring this to sensor data analysis in a production environment, one would not see it as several runs but more as a continuously monitored process. This would lead to a long, single realization of the manufacturing process, not raising the question of ergodicity. This does not hold up if you consider factors like wear and tear of machine components, component failures or different ways of manual operation as different starting conditions for the process model, questioning ergodicity.
Contemplating this idea, the usage of machine learning, whether supervised or unsupervised, will make it necessary to split measurement data segments or training/evaluation/test sets. The concept is therefore worth considering in addition to stationarity, as an ergodic process is always stationary, but not necessarily in reverse [10]. In [18], the authors mention that non ergodicity is negatively affecting ML performance in a signal processing application.

Passing on to random time series, which are generated by an underlying random process. The latter is defined as a sequence of variables and can be thought of as a random vector with an infinite number of dimensions [24]. More specifically, random data is neither deterministic nor periodic [39]. Making a connection to the concept of stationarity,

the authors of [19] mention that those two properties are are independent to each other with respect to time series and that stationarity is referring to temporal relations in the data while randomness is not.

Testing

After introducing a theoretical background for important time series properties, the focus of the following is on testing procedures for evaluating the presence of them in actual time series data. Several important aspects of analytics making it necessary to differentiate according to bespoke features. Firstly, different models are needed for processing stationary/non-stationary data and conclusions and insights drawn from it might need to be interpreted differently. In other words, it makes no sense to let the same algorithm run over stationary and non-stationary data, as this inconsistency can affect validity of results. Another aspect lies in the use of testing procedures as a filter, which reduces amount and dimensionality to be processed. Lastly, extending this to the practical applications in manufacturing, irregularities like sensor failure can be discovered by a signal considered random. This statement can be made as a random time series will not have periodic elements, which should be immanent in this context as a production process is generally of a repetitive nature.
There are several of individually developed approaches in the literature for testing stationarity and randomness geared towards specific applications and datasets. As a novel preprocessing approach including testing methods is developed in the course of this work, only the foundational methods shall be outlined. Those are runs testing, unit root testing and autocorrelation function. Regarding ergodicity, one of the few practicable testing methods identifiable in current literature is mentioned.

Runs Test.
The runs test [13], also referred to as Wald-Wolfowitz runs test, is a non-parametric test which can be used to assess whether or not a dataset is originating from a random process. A run within a sequence is referring to a segment of equal adjacent elements. Originally, runs were series of increasing or decreasing values, as the probability of a certain value being larger or smaller than its predecessor is following a binomial distribution in a random dataset. It is therefore foundational for the test. To apply runs test, any sequence must be transformed into a two-valued representation.
In time series applications, the representation is usually generated by differentiating between values above and below the median of the series. For the actual test, the fact that the series is random is defined as the null hypothesis. It is rejected if:

$$Z > Z_{1-\frac{\alpha}{2}}. \tag{9}$$

Assuming a significance level of 5 percent, the hypothesis is rejected if the test statistic $Z > 1.96$. It is computed via:

$$Z = \frac{R - R_e}{\sigma_R}, \tag{10}$$

with R being the number of runs observed in the series. The expected number of runs, R_e and the standard deviation are defined as:

$$R_e = \frac{2n_1n_2}{n_1 + n_2} + 1, \tag{11}$$

and

$$\sigma_R^2 = \frac{2n_1n_2(2n_1n_2 - n_1 - n_2)}{(n_1 + n_2)^2(n_1 + n_2 - 1} \tag{12}$$

whereby n_1 being the number of runs above the median, n_2 the ones below.

The authors of [42] are using this test in the course of a selection process for seismic time series. In [68], Wald-Wolfowitz is used as one of several tests to perform sensor verification, which is an argument for its applicability in the course of this work. It can also be used with non-stationarity as the hypothesis, demonstrated within [37], which applies it in the domain of hyperspectral images.
The main advantages of the runs test is that it is non-parametric and distribution free. This means that the significance level holds for any distribution and the method is applicable to a case were only small samples are available [30].

Unit Root Test.
The principal functioning of a unit root test can be explained introducing an AR(1) time series model:

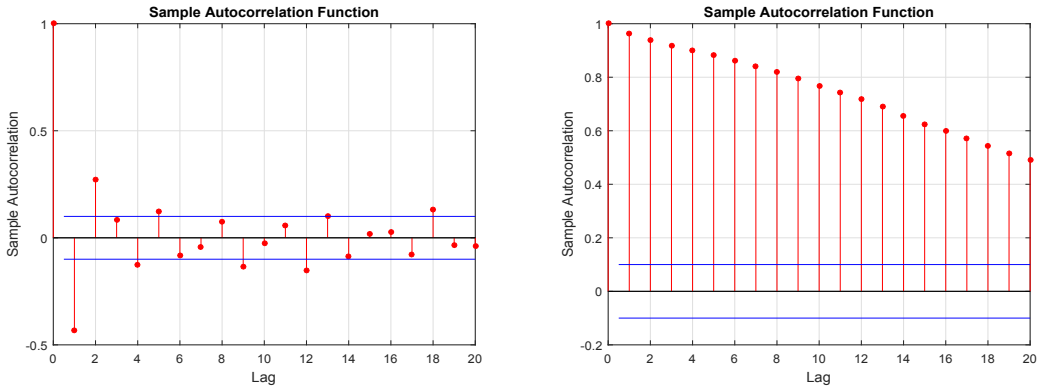
$$x(t) = \theta x(t-1) + \epsilon_t, \tag{13}$$

where $x(t)$ is a time series value of time t and ϵ_t is an error term.
Models used to describe processes consist of one or several single term expressions, where each of them corresponds to a root. A root of size 1 (unity) is a unit root which is one of the causes on nonstationarity and nonpredictability of time series. For equation (13), a unit root test evaluates θ. In case of $|\theta| = 1$, the test is accepting the null hypothesis (H_0) that the times series has a unit root. The alternative hypothesis would be that the time series is stationary in case of $\theta < 1$.
This procedure basically describes the Dickey-Fuller test [20]. As the complexity of the AR(1) process above is exceeded in most applications, the Augmented Dickey-Fuller(ADF) test [74] is far more used. It is able to capture a more complex model with components for trend and drift. This allows several versions of the test to be carried out. As an example, it is possible to test for trend stationarity specifically by restricting other coefficients to zero and neglecting the respective terms.
In [2], the authors compared several unit root and stationarity test and present recommendations for different time series lengths. The ADF test performs well on all samples, particularly as length increases, which is the usual case in sensor data analysis. In addition, they bring up the idea of supplementing a unit root test with the KPSS test [44] which tests the null hypothesis of stationarity against an alternative of unit root. Thinking that through, the two could be used to reassure results mutually. A rejected H_0 from ADF test and an accpeted one from KPSS test would vehemently imply stationarity. A failed rejection from both could indicate that the number of observations is insufficient.

Autocorrelation Function.
The autocorrelation function (ACF) [11, 80] allows the computation of a coefficient that describes the correlation of a time series with itself at a later point in time. For a time series x of length n at time t with a lag of k representing the delay, the ACF is given by:

$$r(k) = \frac{\sum_{t=k+1}^{n}(x_t - \bar{x})(x_{t-k} - \bar{x})}{\sum_{t=1}^{n}(x_t - \bar{x})^2}. \tag{14}$$

The result describes the correlation of future and past values. The use of this function for stationarity testing is relying on the development of autocorrelation with increasing lag. A fast approach of the ACF towards 0 is an indicator of a stationary time series [89], while a nonstationary time series will manifest in slow decay. This assessment can be assisted visually via a correlogram. Figure 4 visualizes the autocorrelation function for the exapmle time series from Figure 3.



(a) Autocorrelation function of a stationary AR(2) process

(b) Autocorrelation function of unit root process

Figure 4: Autocorrelation of stationary and non-stationary time series

Problems of this approach include that (non)stationarity is indicated, but not explicitly proven. Furthermore, the conclusions that can be drawn from the ACF visualization could as well be drawn by the series itself, as signs of nonstationarity such as mean shift, trend, seasonality or periodicity are visible.

Ergodicity Test.
Ergodicity tests are not as common as the ones for stationarity. In [89], a testing procedure composed of several components is presented. It is structured as follows:

1. Time series t is tested for stationarity via its autocorrelation function as it is a prerequisite for ergodicity
2. The sample mean value series M_t and the variance series of mean value $D(M_t)$ are calculated
3. $D(M_t)$ is simulated with RBF neural network (as it is nonlinear) and predict its trend with the time parameter tends towards infinity
4. Check the series for mean ergodicity (confer equation (8))

Details can be found in [89].

Transformation of Nonstationary Time Series

After nonstationarity has been detected by one of the proposed testing methods, one might still be interested in using the time series in further analysis. If a model is used that is unable to handle nonstationary data, we need a way to transform it into a stationary form. A simple, but often sufficient method is differencing [14]. It can be computed for

different orders, but as pointed out in [41], first-order differencing is enough if the data is free of seasonal effects:

$$y(t) = x(t-1) - x(t) = \nabla x(t-1) \tag{15}$$

with y being the new stationary series and x is the nonstationary form. Lei et al. (2015) [38] successfully removed the trend from chemical process viscosity data.

2.3 Redundancy Analysis

Multivariate Time Series (MTS) are multidimensional data. Amongst other challenges, the number of problems can reach a level where problems arise. The curse of dimensionality is an often used term in machine learning and other domains. Briefly explained, it describes that the amount of training data necessary for an ML task exponentially increases with data dimension.
Its specific relevance in time series analysis has been analyzed in [87]. The authors point out that the problem with high dimensional time series lies in the fact that the intuition behind many analytics approaches is based on low dimensional spaces. It is significantly harder to draw intuitive for problems in higher dimensional spaces. This problem is also affecting a deep learning algorithm, as it is a non-linear model which usually uses more parameters than inputs, leading to overfitting and numerical instability.
This leads to the concept of redundancy analysis. The idea here is to reduce the dimensions of MTS via removing or not considering single time series. The potential of significant reduction of dimensions is especially high if the MTS dataset consists of machine sensor signals. The same physical phenomenon, occurring in a part of the machine, is observed by several sensors, makes a certain degree of redundancy inevitable.

Cross Correlation.
Such an analysis can be carried out using the cross correlation function. A normalized version, able to handle varying amplitude of signals, is presented in [3]:

$$\bar{C}_{x_1x_2}(\tau) = \frac{C_{x_1x_2}(\tau)}{C_{x_1x_1}(0)C_{x_2x_2}(0)} \tag{16}$$

where τ represents a time offset and $C_{x_1x_1}$ is the autocorrelation of the first signal. The normalized cross correlation, $\bar{C}_{x_1x_2}(\tau)$, will lead to result between [-1,1]. Both boundary values mean identically shaped series, while 0 means uncorrelation. A possible drawback of this function arises when applied to autocorrelated signals, producing misleading results [28].

Having a set of multiple signals, cross correlation is applied pairwise, ending up with a cross correlation matrix. Such a matrix R is then:

$$R = \begin{bmatrix} C_{x_1x_1} & C_{x_1x_2} & \cdots & C_{x_1x_n} \\ \vdots & \vdots & \vdots & \vdots \\ C_{x_mx_1} & C_{x_mx_2} & \cdots & C_{x_mx_n} \end{bmatrix}. \tag{17}$$

A possible result of such a cross correlation coefficient matrix for an example of four signals would look like in Figure 5 below. The diagonal obviously contains now information.

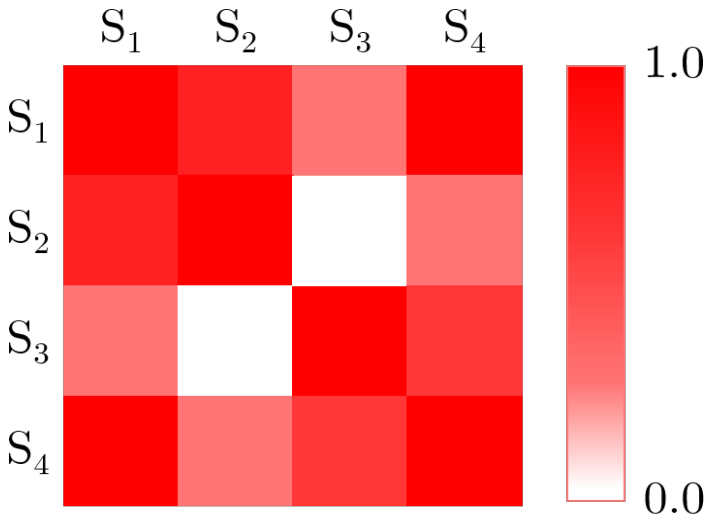


Figure 5: Exemplary cross correlation matrix

So far, only correlation information about the signals has been determined. For actually reducing data dimensionality, a concrete selection procedure needs to be defined. Possible examples include [55], who perform pairwise correlation and classify signals above a threshold as cancelled. Mutually correlated signals are grouped and only one representative of each group is considered further. Bacciu (2016) [3] uses this concept on features instead of signals. Pairwise cross correlation and identification of noisy features via autocorrelation are repeated until a final set of features is found.

Mutual Information.
Mutual Information (MI) [43] is based on the concept of Shannon Entropy [78]. It is capable of evaluating the dependencies of two random variables, whereby the amount of information on one variable that can be deducted from the other is quantified.
For the two random variables X and Y, mutual information is defined as:

$$MI(X,Y) = H(X) + H(Y) - H(X,Y), \tag{18}$$

where $H(X)$ and $H(Y)$ are the Shannon Entropies of the variables and $H(X,Y)$ is the joint entropy. With the definition of Shannon Entropy , this is equal to:

$$MI(X,Y) = \int_X \int_Y \rho_{X,Y}(x,y) \log \frac{\rho_{X,Y}(x,y)}{\rho_X(x)\rho_Y(y)} dxdy. \tag{19}$$

An important prerequisite for the computation is the estimation of $\rho X, Y$ (joint probability density function. Details on this are omitted here, but a promising method is k-nearest neighbour estimation [43].
MI has been used successfully in input selection for time series forecasting in [12] and [95] and for MTS feature selection in [22]. While correlation methods measure the degree of linear association, MI can detect nonlinear relations between two variables with no limitations concerning statistical moments. Though this definitely is a desirable feature considering possible nonlinear dependencies in a sensor dataset, problems can emerge in real world applications. MI results can be misleading and difficult to interpret, as will be shown in Chapter 5.

2.4 Time Series Complexity

The previously introduced allowed to identify a group of time series that are correlated to each other. If it is decided that only one of them needs to be analyzed, it is needed to determine which one. It is therefore of interest to measure the relative "interestingness" of a time series with respect to an application.
It is impossible to present a general definition of when data is interesting, as it is problem dependent and even in a defined setting, the characteristics of a dataset are often unknown. What can be said, at least for the problem of sensor data classification, is that data is interesting if it is able to describe the underlying process well and should be suspect to patterns.
Considering this, the field of time series complexity measures, especially entropy-based techniques, show potential to cover the problem. The concept of information entropy was introduced by [78] and describes the average amount of information in data generated by a stochastic process. A high amount of information thereby corresponds to unpredictability of the data, as e.g. a periodic signal contains less potential for new information arising. This is interpreted differently in applications like this one, as high predictability is beneficial for well founded process analysis.

It has to be noted that there are few applications of entropy-based complexity measures in a manufacturing context, but potential can definitely be spotted.

Approximate Entropy.
Approximate Entropy (ApEn) [66] is a technique which can be used to measure the predictability of a time series. Like other entropy based techniques, it is able to provide a quantification of the occurrence of patterns in the data.
To perform ApEn, one has to construct a sequence of vectors out of your time series, whereby each vector is formed via

$$x(i) = [s(i), s(i+1), ..., s(i+m-1)], \tag{20}$$

whereby s is the time series and m is the length of the runs that are compared. One should end up with the sequence $x(1), x(2), ..., x(N-m+1)$ with N being the length of the time series. Next, the following computation is done for each vector:

$$C_i^m(r) = \frac{(n_j | d[x(i), x(j)] \leq r)}{(N-m+1)}, \tag{21}$$

with

$$d[x(i), x(j)] = \max_{k=1,2,....,m} |s(i+k-1) - s(j+k-1)|, \tag{22}$$

whereby the distance function computes the maximum difference in scalar components of the vectors. Now, we define

$$C^m(r) = (N-m+1)^-1 \sum_{i=1}^{N-m+1} C_i^m(r). \tag{23}$$

The Approximate Entropy value of the time series is equal to

$$H_{ApEn}(m, r) = C^m(r) - C^{m+1}(r) \tag{24}$$

Approximate Entropy will deliver a lower value for time series showing recurring patterns which are therefore more predictable.
An issue with ApEn is that it has been developed and mainly used within the physiological domain. On the other hand, this is true for the majority of suchlike methods. In addition, physiological time series are also sensor measurements. They are equally subject to noise, can be monotonous a majority of the time with an unsteady occurring of events. Unfortunately, ApEn is also a subject to general drawbacks pointed out in [71]: heavy dependence on sequence length and a poor relative consistency. The latter is referring to different results with varying paramters when comparing two series. This actually is alarming, as the technique shall be used for a relative evaluation of different signals.

Sample Entropy.
Motivated by the weaknesses of ApEn, [71] developed another complexity measure called Sample Entropy (SampEn).
Again, a time series of length N is introduced and vectors of length m such as $X_m(i) = x_i, x_{i+1}, ..., x_{i+m-1}$. A filter criterion r is needed again, as well as a distance function d, which can be any common distance function. SampEn than counts the number of vector pairs that have a distance below r for all pairs of length m (represented by B) and length $m+1$ (represented by A). Sample Entropy is then computed as:

$$H_{SampEn} = -log\frac{A}{B}, \tag{25}$$

which will deliver a positive result, with lower values indicating higher predictability.
An important feature is the exclusion of self-comparisons regarding the vector pairs, meaning that $d[X_m(i), X_m(j)]$ is not computed for $i = j$. This is considered advantageous, as those comparisons influence the overall result and will create the impression that the time series is more regular than it actually is. In addition, there is less variance in the result with varying data length and implementation is easier.

Permutation Entropy.
Permutation Entropy (PE) was introduced by [4], is a complexity measure where the temporal order of values is considered.

For a time series $s = s_i, s_{i+1}, ..., s_N$ the permutation entropy is defined as

$$H_{perm}(n) = -\sum \rho(\pi) \log \rho(\pi). \tag{26}$$

In addition, it is possible and recommended to define permutation entropy per symbol of order n:

$$h_n = \frac{H(n)}{n-1}. \tag{27}$$

This is obviously asking for further explanation. PE is depending on the definition of a permutation order n. The summation in equation (26) is running over all permutations π of order n, equalling $n!$ possibilities.
Considering an example series with $N = 6$ elements and a chosen order $n = 2$, one can find 5 pairs of consecutive values. It is now distinguished between pairs where $s_t < s_{t+1}$, forming one permutation, and such where $s_t > s_{t+1}$ forming the other. Assuming that the pairs fall into the categories 3:2, permutation entropy is equal to

$$H(2) = -(\frac{3}{5}) \log \frac{3}{5}) - (\frac{2}{5}) \log \frac{2}{5}). \tag{28}$$

Further details, explanation and examples can be found in [72]. In their review, possible parameter choices are proposed as well. For n, a range of 3 to 8 is recommended for applications to physical systems.
The main advantages of this method include simplicity, less calculation effort as well as robustness and variance. In contrast to ApEn and SampEn, PE has been successfully applied to sensor data in the manufacturing domain. In [58], differences in permutation entropy of turbine monitoring signals are used to identify problematic working conditions. [69] have pointed out a connection of increased permutation entropy measures and failures in wind energy converters.

3 Deep Learning on Time Series Data

This chapter is reviewing different models from the deep learning domain. It will focus on introducing their basic structure as well as some more specific traits. A special focus lies on reviewing successful applications from the time series domain. Studies related to image data is also considered interesting if concepts can be applied to time series data.

3.1 Deep Belief Network

A Deep Belief Network (DBN) is a probabilistic, generative model introduced by Hinton et al. (2006) [31]. It has a deep architecture, meaning that the network consists of multiple layers whereby each consists of a restricted Boltzmann Machine (RBM).
An RBM itself has two layers, formed by visible units v and hidden units h. For both, there are bias vectors b and c. In addition, a weight matrix W connects visible and hidden layer, while intraconnections in the layers do not exist. To build a DBN, a number of RBMs are stacked on top of each other. The output of the bottom RBM is used as an input for one above. This continues until the top layer is reached. Figure 6 visualizes this structure.

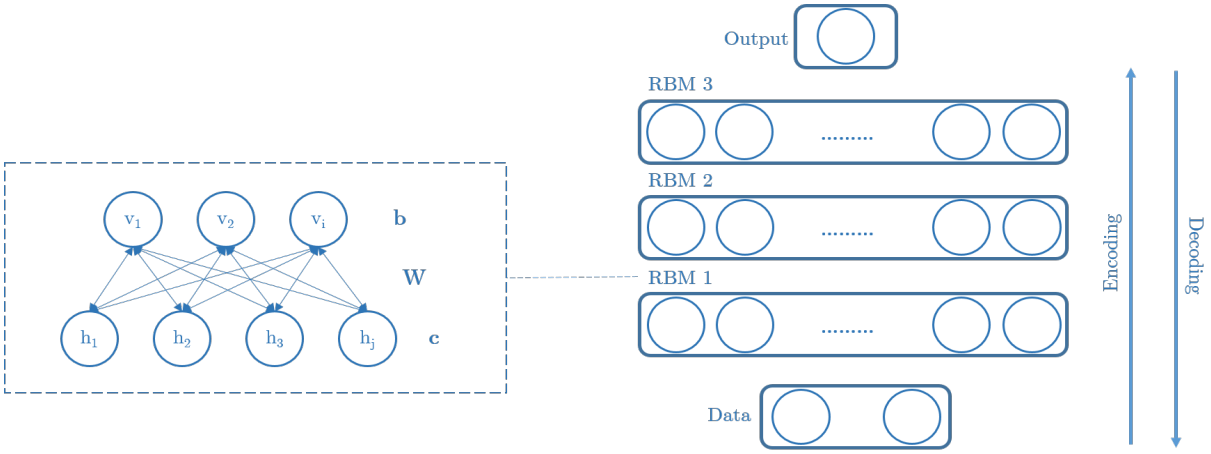


Figure 6: A Deep Belief Net is build by stacking Restricted Boltzmann Machines

The structure of a DBN is analog to the one of Multilayer Perceptron (MLP). The difference between the two is that a DBN uses a bias vector for the visible units. This is necessary for being able to reconstruct the input (called decoding). The reconstruction error is very important in the training process of a Deep Belief Net.

As other neural networks, different activation functions exist for an RBM. Taking a bernoulli-bernoulli version as an example, the probability that any hidden unit h_j is activated given the visible vector v is defined as:

$$P(h_j|v) = \frac{1}{1 + exp^{b_j + \sum iW_{ij}v_i}} \tag{29}$$

$$P(v_i|h) = \frac{1}{1 + exp^{c_i + \sum jW_{ij}h_j}} \tag{30}$$

It basically works the other was round for any visible unit, v_i.
For both the visible and hidden vector, the energy function E and joint distribution p is

defined as:

$$E(v,h) = h^T W v + b^T h + c^T v \tag{31}$$

$$p(v,h) = \frac{1}{2} exp^{E(v,h)} \tag{32}$$

During training, the parameters W, b and c are adjusted in order to minimize the reconstruction error. The rule for parameter update is using contrastive divergence [31] to approximate the gradient of the log likelihood of v. Introducing a learning rate σ, the change of parameters within one step can be expressed as follows:

$$\Delta W \leftarrow \sigma(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1) \tag{33}$$

$$\Delta b \leftarrow \sigma(\langle h_j \rangle^0 - \langle h_j \rangle^1) \tag{34}$$

$$\Delta c \leftarrow \sigma(\langle v_i \rangle^0 - \langle v_i \rangle^1) \tag{35}$$

Via the introduction of their training algorithm, Hinton et al. (2006) triggered an increased usage of neural networks which is sometimes referred to as a renaissance. It is seen as one of the breakthroughs of deep learning. The training methodology proposed consists of an unsupervised part, pretraining each layer individually and a supervised part, finetuning the network with training data for the actual task. While the latter is kind of similar to other training algorithms, the unsupervised pretraining is an interesting feature that deserves further attention. Putting data through a neural network is like applying a nonlinear function to it. In the unsupervised training process, each layer is learning a nonlinear transformation of its input that captures the main variations in it. Pretrained layers build a foundation for supervised learning and generally increase performance.
Though this positive effect is observed, little is known about its causes. In [21], the authors elaborate on that question. As potential causes, they identify:

1. Network preconditioning: parameters are adjusted to a suitable range which leads to a better starting point for their optimization

2. The model is initialized in the parameter space in a way that a lower cost function value is attainable

3. Pretraining works as a special form of regularization, shifting bias in a useful direction and lowering variance

Their experiments proof several of the benefits of unsupervised pretraining. Firstly, an increase in depth is not recommended without pretraining because of poor results. Furthermore, better model generalization and higher feature quality can be expected. The presumed generalization effect was shown, though it deviates from classic regularization techniques as the effect is not fading away with more training data.

Variations of DBN

Modified versions of deep belief nets can be found in the literature. The most interesting examples are convolutional and conditional DBN.
A convolutional DBN [52] is a special realization of a DBN. It is constructed by stacking convolutional restricted Boltzmann machines (CRBM) on top of each other. To achieve a convolutional character, the hidden layer consists of groups and those are sharing all the weights between the visible and the hidden layer. To illustrate this concept, the energy

function of a CRBM with real valued visible units shall be outlined here. Lee et al. (2009) [53] apply it to single channel audio time series:

$$E(v,h) = \frac{1}{2}\sum_{i}^{n_v} v_i^2 - \sum_{k=1}^{k}\sum_{j=1}^{n_h}\sum_{r=1}^{n_w} h_j^k W_r^k v_{j+r-1} - \sum_{k=1}^{k} b_k \sum_{j=1}^{n_h} h_j^k - c\sum_{i}^{n_v} v_i. \tag{36}$$

Here, k is the number of groups in the hidden layer. The visible layer is n_v dimensional array of binary units. This equals to n_h and n_W for the hidden units and weights. The bias b_k is shared among each group and c is shared for visible units.
The authors show that good classification performance is achievable learning feature representations from unlabelled data.

A conditional DBN [6] consists of conditional RBM (cRBM). Those also consist of a hidden and a visible layer, but are extended with additional visible units which are directly connected. Those are used to provide further information to the network, e.g. past data.
Considering RBM equations, this extension is replacing the standard bias terms b and c with a dynamic bias:

$$\hat{b} = b + By \hat{c} = c + Cy \tag{37}$$

The weight matrices B and C represent the direct connections and vector y contains the additional (conditioning) data. Figure 7 illustrates the idea of cRBM.

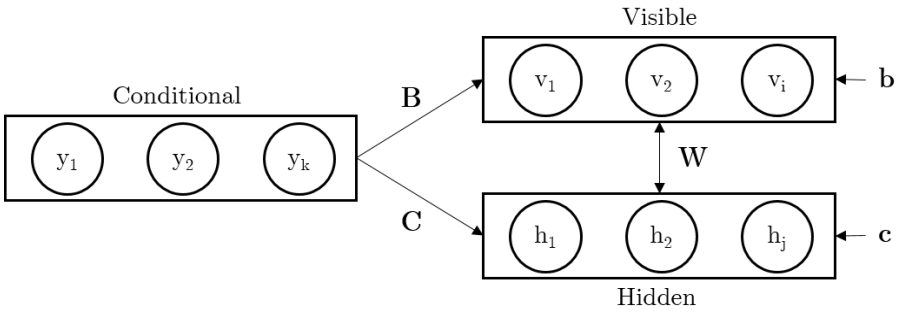


Figure 7: RBM can be extended with conditional units (cf. Battenberg & Wessel (2012)

Battenberg & Wessel (2012) [6] demonstrate the ability of conditional DBN in drum pattern analysis. The possibility to use the conditional units to take past data into consideration indicates potential for all tasks in the temporal domain.

3.1.1 Time Series Applications

In [45], the authors face a multivariate time series classification problem. Five different sleep stages shall be classified using four channels of measurement from the human body. They have been classified originally by a method requiring expert knowledge. The stages occur in epochs of 20 to 30 seconds and channels are segmented to 1 second windows.
Three approaches are compared, whereby two are relevant in the context of this work: a Deep Belief Net (DBN) approach based on handmade features and a raw DBN approach, learning features without the use of expert knowledge.

The feature DBN approach operates on 28 handmade features, which are relative power,

mean value, signal kurtosis, signal entropy, spectral mean and others for all signals. A DBN with two layers and 200 units per layer with a Softmax-classifier on top. The training process consists of a pretraining phase for each layer and a fine-tuning phase for the top layer. The raw DBN approach uses the same network structure. The training happens in an unsupervised manner, using a concatenated signal matrix as the visible layer, where $a - d$ are the channels, n is the number of samples per second and w is the window size.

$$V = \begin{bmatrix} a_1^n & b_1^n & c_1^n & d_1^n \\ a_{1+w}^{n+w} & b_{1+w}^{n+w} & c_{1+w}^{n+w} & d_{1+w}^{n+w} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{38}$$

This results in an input dimension of n x w x $numberofchannels$.
The approaches were compared with respect to classification accuracy and the confusion matrix. The raw DBN had good results on most subjects, but very poor ones on a small number. Although the feature based approach had better results overall, it should be stated that the features learned seem to have a good quality. They cover high and low frequencies as well as high and low amplitudes which is also true for the hand engineered features.
Some lessons learned from the discussion of results can be taken away:

- Setting an initial bias value for the hidden units can influence the result.
- Pretraining the DBN with a relatively large number of epochs is considered crucial for classification performance.
- Even size of different classes present in the training dataset is preferable
- The DBN is struggling on test datasets that are clearly different from training data
- It seems that correlations within the time series are not captured well enough when learning features from raw data. In a feature based approach, correlation can simply be added as an additional feature.
- Increasing the number of hidden units and/or layers has not increased classification accuracy, but simulation time.

In [36], DBN is used for fusing multifeature vibration signal data to improve detection of bearing faults. This is done as noisy vibration datasets are information poor and diagnosis is difficult.
The algorithm they use works as follows:

- Features are extracted from the samples as they are nonlinear. In detail, 14 features from the time domain are computed.
- A DBN is trained and finetuned using training data with the features computed above.
- Having target labelled data, the reconstruction error is computed using the output. According to it, the network structure is adjusted and the training process is repeated until the error is low enough.
- The ideal DBN structure is used for the actual fault diagnosis

This leads to a 42-12-12-4 network structure, whereby the number of input nodes result from 14 features for each sensor and the number of hidden units was determined by the algorithm above. No feature selection is applied. Their results show that DBN is more accurate and robust compared to other machine learning algorithms such as Support Vector Machine, K-Nearest-Neighbour and backpropagation neural network.

Another manufacturing application can be found in [29], where unsupervised learning is executed via a DBN to detect faults in a gear transmission chain. The diagnosis happens via classifying time series segments into one of 8 predefined health conditions. The interesting part is a comparison of DBN with multilayer back-propagation neural network (BPNN). Shortly, their approach works like outlined below:

1. **Time Series Segmentation:** Through apriori knowledge, it is considered relevant to segment the data according shaft cycles. Those are identified through axis crossings. The data in between the axis crossings is then interpolated using a cubic spine function.

2. **BPNN Setup:** For this approach, 15 features are preselected. 11 of those are from the time domain (mean, peak-to-peak, standard deviation, kurtosis etc.) and four are from the frequency domain (mean frequency, frequency center etc.). Not all of them, but a varying number is used during experiments. The selection is done via distance evaluation technique (DET). The network has three hidden layers with a structure of 100-50-10 and the output dimension is the number of health conditions.

3. **DBN Setup:** The RBM parameters are set up heuristically using a genetic algorithm and pretrained individually. A shallow (1000 hidden units) as well as a deep (1000-500-200) approach is evaluated.

In their experiment, deep belief nets significantly outperformed the BPNN approach. The latter is found to very unstable, meaning the results for different feature sets show high variations. The feature learning ability of DBN is proven by the fact that using those features with BPNN improves results compared to using predefined features. The authors reached up to 100 percent accuracy with their methodology.

3.2 Autoencoder Networks

Autoencoders [7] are neural networks learning data representations in an unsupervised manner. Consisting of one input and output layer and one or more hidden layers, the data is usually compressed into a lower dimensional form. This works through the combination of an encoder and a decoder. During encoding, the input is forwarded through the network and is thereby transformed:

$$z_1 = \Sigma_1(W_1 x + b_1), \tag{39}$$

where Σ_1 is the activation function, W_1 and b_1 are representing the encoder weights and biases. In the decoding step, the input is reconstructed from the representation. The indices are used as the parameters are different for the different layers. In a one-layer case, they are not needed. The reconstructed input is computed as:

$$x_r = \Sigma_2(W_2 z_1 + b_2) \tag{40}$$

The training of an autoencoder works via adjusting the parameters to minimize the reconstruction error:

$$E(x, x_r) = ||x - x||^2 \tag{41}$$

Another feature of autoencoders is that input and output layer have the same dimension. Their structure is shown in Multiple layers are referred to deep or stacked autoencoders. The encoded input can be a useful discriminative feature and can therefore be extracted and used for further analysis on this dataset. Apart from that, the stacked layers can also be finetuned and used as a classifier itself.

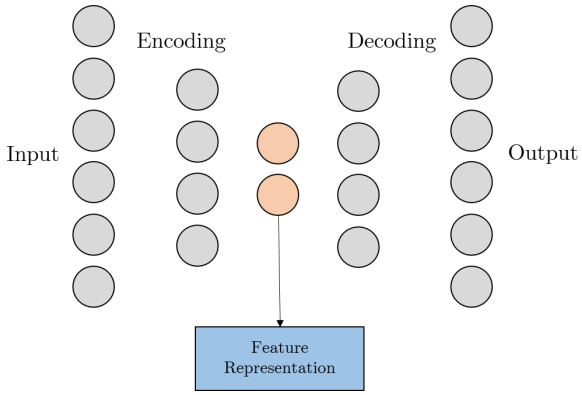


Figure 8: Structure of a stacked autoencoder

3.2.1 Time Series Applications

In [47], the authors improve on sparse autoencoders (SAE) by enhancing them with a cost-relevant feature. Their work is motivated by the fact that SAEs face problems when applied to noisy datasets. As this problem is based on the fact that a standard cost function treats the reconstruction error of each unit equal, a learnable parameter, called learning vector, is added to the cost function which allows the network to favour certain units and therefore increase their effect on parameter update. Two possible implementations are proposed:

- A fixed weighting vector set before training through a feature selection algorithm
- An adaptive weighting vector learned during training with other model parameters

The results show that SAEs enhanced with this element offer superior performance compared to standard versions on practical datasets.

In [25], Guenther et al. use a deep autoencoder to extract features from image data in the context of laser welding. Their implementation consists of the following parts:

- A region of interest is identified within the images as their dimensionality makes efficient training of an ANN impossible. In addition, subsampling is used for dimensionality reduction.
- The multilayer autoencoder consists of a relatively large number of neurons in the early layers (1024, 2048) as a more general representation should be learned. Overall, 16 features are extracted through 11 hidden layers.
- Their representation is compared to PCA, showing that while PCA has a lower reconstruction error, the autoencoder representation is superior for classification.

3.3 Convolutional Neural Networks

Convolutional neural networks (CNN) are another variant of deep learning algorithms. They fall into the category of feed-forward neural networks. Those, in contrast to recurrent networks, process information solely in the forward direction and their node connection graph is free of cycles. LeCun et al. (2010) present CNNs in [51] describing their multiple stage structure, where each input and output is represented by multiple arrays called feature maps. As in DBNs and other deep networks, CNNs consist of input, output and hidden layers, whereby the hidden layers perform operations on the feature map. Those operations are either convolution, pooling, processing via fully connected layer or normalization. A good introduction into the foundations of CNNs, being their training process and the different layer structures, can be found in [92]. The basics will be introduced in this section. Figure 9 visualizes the structure of a convolutional neural network.

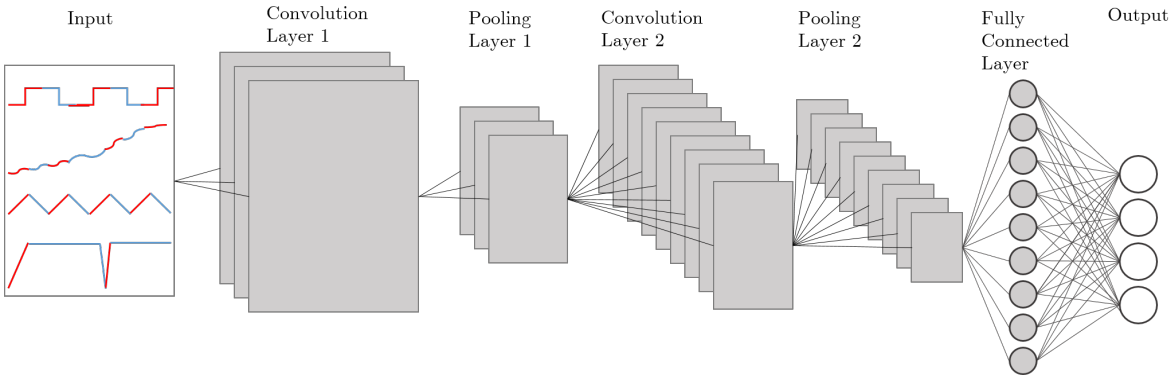


Figure 9: Principal structure of a Convolutional Neural Network (CNN)

When the input is put through the layers initially, the output result is not going to be very valuable because the weights (and other parameters) are initialized randomly. To quantify this situation, the loss function comes into play, represented by a loss layer in CNN structure. Realized through one of different possible functions, the loss quantifies the degree of accordance between data labels and network prediction. The network parameters are then optimized to reduce this loss. In CNN, this is realized via stochastic gradient descent (SGD) [49]. For parameters W, user defined learning rate η and cost function output z, the parameters are updated as:

$$W = W - \eta \frac{\partial z}{\partial W} \tag{42}$$

The term $\frac{\partial z}{\partial W}$ describes the rate of changes in z based on changes in W. A vector like that is called gradient, and SGD reduces the loss function value by optimizing the parameters in the opposite direction of the gradient. A common variant is to update the parameters based on a single sample, which shows fast conversion [49]. The gradient is searched for using error-backpropagation [50]. Now some details on the different layer types in a CNN are provided.

- **Convolutional Layer.** In the convolutional layer, a kernel operator is moved over the input. As both are represented numerically, a product will be computed and transferred into the output. This is done for every possible position the kernel can be shifted on the input, as visualized in Figure 10a. In most applications, multiple

kernels are used in every convolutional layer. What those kernels basically do is "highlighting" features of the input so they can be considered by the algorithm. So if the convolution layer is followed by an activation layer, the neuron activation will be made based on these features. A simple example would be the vertical or horizontal edges of an image, whereby more complex features, such as a specific combination of edges, can be learned by convolutional layers deeper in the network. The area of the input the kernel is currently located on is often called a receptive field.
A general formulation of the operation is given in [51]:

$$y_j = b_j + \sum_i k_{ij} * x_i \tag{43}$$

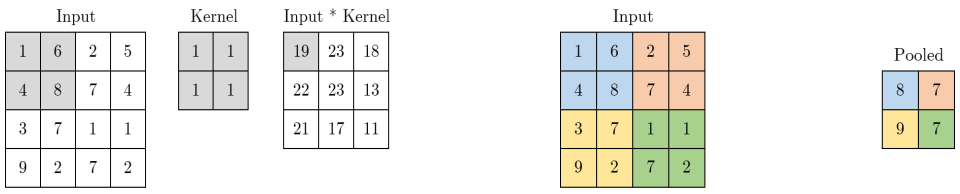
whereby y and x represent the feature map before and after the convolution, respectively. The kernel k is applied and a trainable bias b is used.

- **Pooling Layer.** Pooling layers are inserted in between in convolutional layers in a CNN. In general, they follow the purpose of reducing the feature maps representation size, reducing computational effort and prevent overfitting. This happens by using a nonlinear function to combine the input of a defined a group of units into one unit. Max-pooling is one of the most used variants. Scherer et al. (2010) [76] evaluate its performance impact compared to subsampling and show it is superior. With a pooling window size n x n and a window function u, max-pooling identifies the maximum value in a certain area of the input transfers it to the output. Figure 10b illustrates this principle, as well as the following equation:

$$x_j = max_{NxN}(x_i^{nxn} u(n,n)) \tag{44}$$

However, this comparison has been made on image data. In [17], max-pooling is used in successfully applying a CNN in time series classification. The authors report that a larger pooling size has the benefit of the layers only learning local features and a larger number of layers can be employed.

- **Fully-connected Layer.** Fully connected layers usually form the last stage in a CNN before the output is computed. All its neurons are connected to all activations in the previous layer. This layer uses the input from all convolutional and pooling layers and draws a conclusion with respect to the actual task.



(a) Convolution (b) Max-Pooling

Figure 10: Illustration of convolution and pooling

3.3.1 Time Series Applications

Although Time series are applications are rare, some lessons can be learned from them. Manufacturing oriented applications are mainly based around image recognition.

Encoding Time Series as Images

As stated, CNNs are mainly applied to images and time series applications are rare. While the few time series approaches are adjusting the model, the authors of [27] take a different path and transform a time series into an image before using it as an input for a CNN. Their methodology holds several interesting aspects in the context of this work. This is an outline of their work:

1. **Time Series Encoding.** A time series x is taken and a 2D phase space trajectory is computed. Each spot in that space is defined by $s_i = (x_i, x_{i+1})$. A recurrence plot (RP) is computed. Each index is defined as $R_{ij} = dist(s_i, s_j)$.

2. **Deep Architecture.** A CNN consisting of two feature learning stages is used, each having concolution, activation and pooling operators. ReLu is used as an activation function and maxpooling is in place.

3. **Training Process.** Training happens via a gradient based optimization method using backpropagation and stochastic gradient descent for parameter optimization.

Figure 11 illustrates the concept of recurrence plots (RP) for time series.

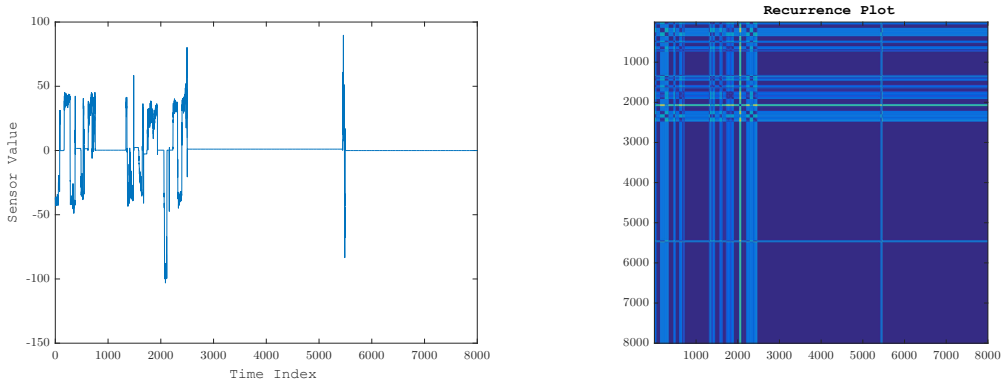


Figure 11: Exemplary sensor time series and respective recurrence plot

The authors reported that their results are superior to comparable studies, including the ones were were RPs are combined with other learning algorithms. Especially the concept of RP is considered interesting for applying deep networks to sensor data. Firstly, the use of RP is based on the fact that a time series can be characterized by its recurring behaviour, which is especially true for time series from a production environment. Furthermore, the transformation of time series into images broadens the amount of suitable existing concepts out there.

Unifying Feature Learning and Classification

In [94], Yang et al. (2015) use convolutional neural networks for human activity recognition (HAR). Their methodology, which capitalizes on the fact that a combination of unsupervised feature learning and supervised classification can enhance the discriminative power of features, is outlined below.

The multivariate time series dataset originating from several sensors placed on a subjects body is segmented using a sliding window approach. An instance processed by the CNN ends up to be matrix with the sample rate and the number of variables as dimensions. The CNN is constructed consisting of five sections:

- Section 1 and 2: A convolution layer and a ReLu layer is used. A max pooling layer finds the maximum feature map and a normalization layer is in place.
- Section 3: Just a convolution and ReLu layer, as convolution reduces the feature map dimension to one and pooling becomes unnecessary.
- Section 4: A fully connected layer unifies the feature map for all sensor dimensions, followed by a ReLu and optimization layer.
- Section 5: A fully connected layer maps features to output classes and a softmax-function provides probabilistic classification result and an entropy cost function quantifies the error using labelled examples.

The results show that CNN outperforms shallow methods such as kNN and SVM, which is explained by feature representation capturing the characteristics of the time series better. The fact that a deep belief net (DBN) is outperformed as well indicates superiority of supervised deep methods. The interesting features of this work are the problem similarity of HAR and machine operations classification as well as the reported success of unifying feature learning and classification in one model.

3.4 Long Short Term Memory Networks

One of the shortcomings of traditional neural networks is that past samples are not a factor in making a prediction for the current one. Although all prior samples are influencing the parameter adjustments, no special dependencies are considered between two examples. This has been the main motivation behind Recurrent Neural Networks (RNNs). A loop structure is allowing their output to have a partial influence on the following inputs. This feature makes them suitable for sequence data and they have been applied successfully to time series problems.
A lot of this success was based on Long Short Term Memory Networks (LSTMs) introduced in [32]. In practice, it is not clear and not consistent which past information is relevant for the current sample. Although the loop structure of RNNs would allow them to capture dependencies with different time spans between them, they perform poorly in experiments when required to learn the ones that are widely apart in time, as reported by [8]. LSTMs also have a chain-like structure, but in a more sophisticated form. Initially, they consisted of an input gate unit, protecting memory from irrelevant input and an output gate unit, protecting other units from irrelevant unit of the current one as well as a memory cell. The latter is a central linear unit receiving information from the output and the input [32]. The output and input activation, y_{out} and y_{in} work like:

$$y_{out} = f_{out}(net_{out}(t))y_{in} = f_{in}(net_{in}(t)) \tag{45}$$

where f_{out} and f_{in} are the activation functions and net_{out}, net_{in} are current units output and input. This initial structure has been adjusted several times. A practicable variant, which can be found in [33, 23] shall be outlined below. The structure is visualized in Figure 12.

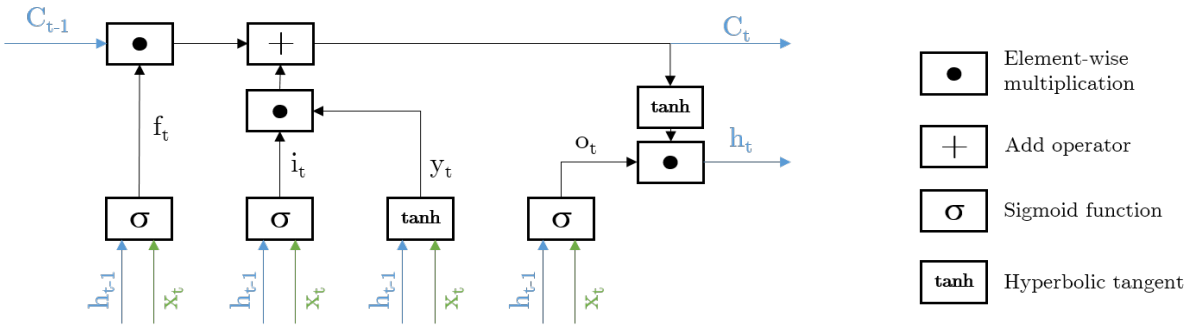


Figure 12: Structure of a LSTM network

An LSTM in this form consists of three high level elements:

1. Forget gate f: This layer decides how much of the previously stored information (memory) needs to be retained. It is realized by a sigmoid layer who takes the previous hidden state h_{t-1} and the current input, x_t into account.

2. Input gate i: Here, the amount of current information to be maintained is determined. This is done in a similar manner compared to the forget gate. In addition, a hyperbolic tangent layer who creates a temporary cell state y [33]. This two will be used combined for a state update.

3. Output gate o: A sigmoid layer decision what part of the cell state is outputted. The new hidden state is determined via putting it through another tanh layer.

A LSTM consists of gates, which are controlling how much information is forgotten or remembered. In addition, they deal with the problem of vanishing gradients [33]. For steering the information flow, sigmoid functions and hyperbolic tangent functions are used. The sigmoid produces values between 0 and 1, where 0 means that all the information is forgotten. The information in the network is represented via the cell state C, the hidden state h and a temporal cell state y. The hidden state is the one actually put out. The vector x_t represents the current input data at time stamp t.

3.4.1 Time Series Applications

The work of [91] deals with detecting anomalous behaviour in time series data from the Large Hidden Collider (LHC). In detail, the authors use voltage data streamed from superconducting magnets, which form a critical part in the system. The anomalous event of interest is a so called quench, describing an incident where a superconducting cable starts to conduct normally. It is decided to model the data with LSTMs. This decision is influenced by the existence of quench precursors, meaning that past events are influencing the likelihood of a quench with different probabilities. As stated in this section, recurrent networks are capable of modelling this dependencies.
With a logging database accessible with years of magnet data, several data selection decisions need to be made. As quenches are relatively rare within this huge database, a group of magnets with the highest quench frequencies are selected. Another crucial factor is the window length, as all relevant event precursors should be considered with the amount of data staying reasonably low. The authors choose a trade-off between those two factors with a window size of 24 hours before the quench.

The deep learning structure employed consists of an input layer, four LSTM hidden layers, one feed-forward hidden layer and an output layer. For the hidden layers a different number of cells is evaluated together with variations in other parameters. Two of the four LSTM layers are using a dropout rate of 20 percent. Dropout is referring to the practice of leaving some hidden units during testing, but not during training. Root mean squared error (RMSE) and mean percent error (MPE) are used to evaluate accuracy. The signals are normalized between 0 and 1 and the dataset is split between training and testing in a 70:30 ratio. The results show that the setup perfroms well and a very low RMSE is achieved. The accuracy declines when predictions are made more steps ahead.

Mehdiyev et al. (2017) [57] use an LSTM autoencoder network to extract features from multivariate sensor data from steel manufacturing. The authors are trying to measure the quality of semifinished products to choose further production stages. The surface defects (if any), caused by the chemical properties of steel, are classified based on the learned feature representations by a deep feed-forward neural network. The dataset includes data from 89 sensors measuring process variables in different positions.
The results when directly applied show high average accuracy. Though the results on samples without defect were high, the approach was not able to assign the parts with a defect to the correct production step. To improve that, a sensitive learning approach was used, penalizing misclassification based on financial implications. Although the overall accuracy went down, defect classification improved as they have higher cost effect.

4 An Approach for Manufacturing Process Analytics

4.1 Preprocessing Approach

In the course of gathering valuable insights from sensor data, a novel preprocessing approach for multivariate time series data is introduced. Right away, it is important to put the term *novel* into perspective.
Calling a set of computations *Preprocessing* is just done inconsistently to some degree. Even if this very broad term is limited to techniques applied before machine learning, what is summarized under this description is just varying a lot. When the methods used here are looked up in the work of other authors, they are often referenced as feature selection. This mainly, but not always means selecting characteristics computed from the data that describe it best. In fewer cases, similar to this approach, the term signal selection is used. Though technically different, those two terms are synonymous in a way, as different signals measured from a piece of machinery are features of the process, where some might be more characteristic than others.
It is still considered valid to label the introduced approach novel, as the exact same or at least clearly similar combination of methods can not be found in the literature. Their selection has been made based on specific requirements of the application in this work and from reviewing them in a relevant way in Chapter 2.

Generally, a physical system is described by a number of variables. What we basically want to end up with is a subset of variables offering the same information. This is achieved by the preprocessing structure visualized in Figure 13 testing and comparing the data with different regards. After selecting the (sub)system that should be analyzed and a basic data validity check, the data is normalized to a common scale and null signals are removed. A stationarity test is performed and used as a filter for nonstationary signals. Signals are grouped via their redundancy to each other. The ones that are considered the most relevant with respect to information content are selected to finish the process.
Details on each step are following below. Some special traits of the approach shall be mentioned. The preprocessing is actually contributing to the process analysis itself, which is quite unusual. The signals dismissed for low entropy and nonstationarity are not dropped entirely but are further analyzed with the same methods. If it is revealed that only particular segments of the data are unpredictable, an intersting event in the process might already be discovered, as this can be indicative for non regular system behaviour. The separate analysis of those channels is considered a good path to take as partial selection of a signal is not possible. When passing an MTS dataset to a learning algorithm , the the timeframe has to be the same for all dimensions. It is therefore recommended to perform the approach over the entire dataset, select the respective signals and then continue with any necessary separation (e.g. into training and test data) afterwards. Other operations o the data, such as smoothing, representations or feature extraction will be covered in the learning approach as it is inseparably linked to it.

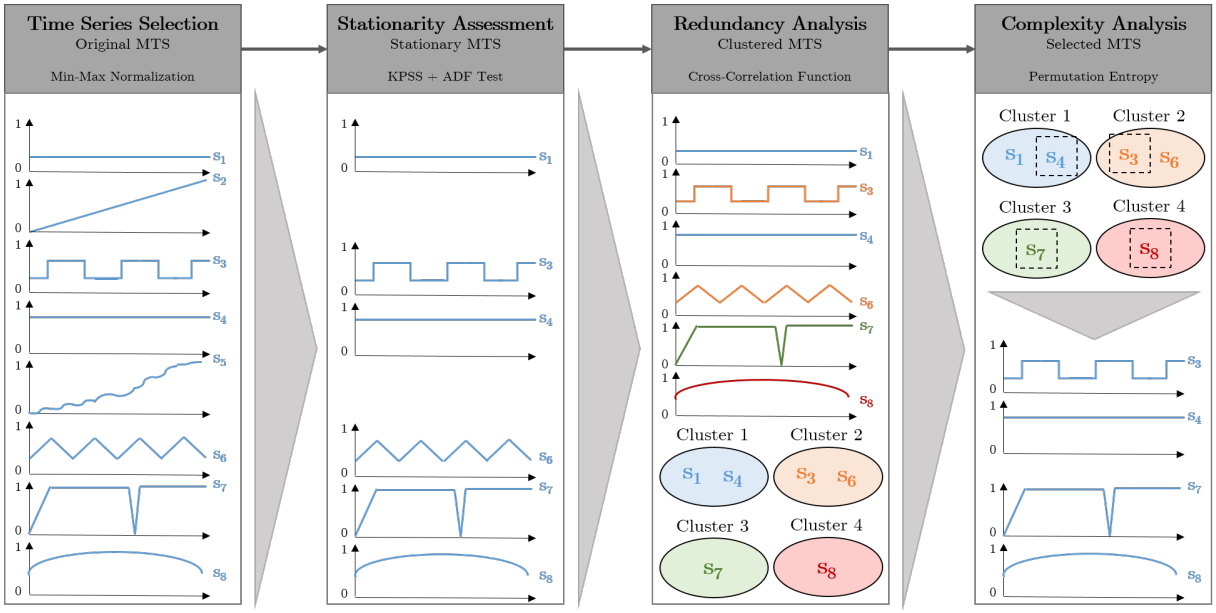


Figure 13: Approach for preprocessing sensor signals

In addition to the selective functionality, the approach is also able to perform further data investigation based on the result during selection. This is done in a segmental manner, meaning that results of the overall series are compared with those of smaller segments. Whenever a segment differs considerably from the series, it is considered anomalous behaviour. Figure 14 illustrates how this feature is embedded in the preprocessing approach.

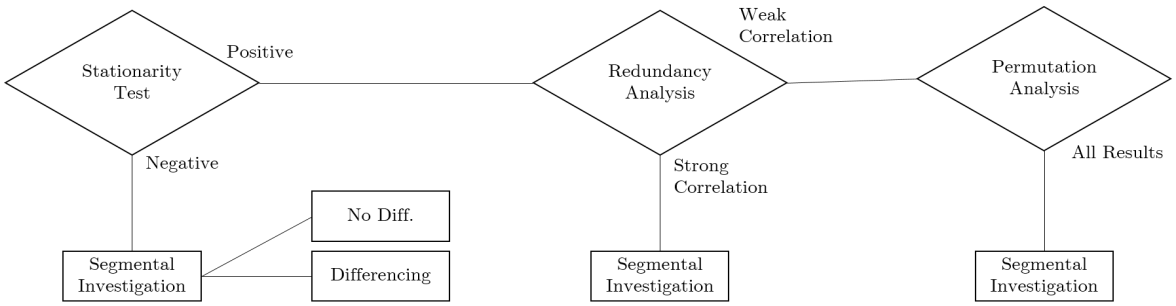


Figure 14: Investigative analysis in the course of preprocessing

Normalization

The scale shown at the illustrative time series shall indicate that normalization has been performed as an initial step. In the method review in Section 2.1, Min-Max normalization and sigmoid normalization showed some desirable features. Because of the scalability, Min-Max seems favourable. In addition, its main drawback of not knowing if future values fit into initial minimal and maximal values as it is applied to the entire dataset at once. If it is planned to use the approach on data streams or with in the course of real time applications, this choice has to be reconsidered. As normalization is the preprocessing step with the least computational effort, trials with other techniques are carried out sporadically to check the compatibility with the learning algorithm. Overall, Min-Max has worked on the presented tasks.

Algorithm 1 Normalization

Dataset S, Signal s, Scale $upBound, lowBound$

for each s in S **do**
    $max_s \leftarrow max(s)$
    $min_s \leftarrow min(s)$
    if $min_s, max_s = 0$ **then**
        $Break$
        Remove s from S
    else
        $s_n \leftarrow lowBound + (upBound - lowBound)(s - min_s))/(max_s - min_s)$
        Replace s by s_n

Stationarity Assessment

The stationarity assessment is designed as a filter method, removing signals classified as nonstationary from the further steps. The runs-test has caused some difficulties when applied to sensor data, as the majority of channels were classified non-stationary. Though a concrete cause for this behaviour can not be identified and comparable applications in the literature are rare, it seems evident that the datasets are not suitable for this technique. Either, the consideration of sequences of values above and below the mean only is not enough to capture complex processes. Another explanation might be that the large values of null values causes the results.
Autocorrelation function (ACF) delivers reasonable results in an efficient manner. On the other hand, issues with automatic execution arise as the convenient use of is visual inspection. The automation is complicated by the fact that the drop of the ACF, which is an indicator for stationarity, happens at a different lag every time depending on the nature of the process. Considering this, a combination of Augmented Dickey-Fuller Test (ADF), testing the unit root hypothesis, and KPSS-test, testing the stationarity hypothesis remains as a decent option. Though both tests are reported as error prone to some degree, the combination leads to a mutual reassurance and will be enough to deliver a valid result.The announced further investigation of nonstationary signals is mainly the same algorithm on segmented data.

Algorithm 2 Stationarity Assessment

Dataset S, Signal s, Test Hypothesis hyp_{KPSS}, hyp_{ADF}

for each s in S **do**
    $hyp_{KPSS} \leftarrow kpss(s)$
    $hyp_{ADF} \leftarrow adf(s)$
    if $hyp_{KPSS} = 0$ AND $hyp_{ADF} = 1$ **then**
        Break
    else if $hyp_{KPSS} = 1$ AND $hyp_{ADF} = 0$ **then**
        remove s from S
    else
        Conflicting Test Result

Redundancy Analysis

The task of identifying redundancies is well executed by cross-correlation introduced in Section 2.3. As stated there, the only problem with it is that it produces misleading results in the presence of autocorrelation in the series. As this part of the selection process strongly influences the end result, this needs to be avoided.
The redundancy analysis happens via assigning a pair of signals into the same cluster if their cross correlation exceeds a threshold. Whenever another signals is highly correlated with one already assigned in a cluster, it joins there. In case of problematic results, the analysis can be done by replacing cross-correlation by mutual information (MI). As the entire process is supposed to be as automated as possible, the detection of such a result to be implemented as well. As the entire process is supposed to be as automated as possible, the detection of such a result to be implemented as well. It is decided to do this via checking *intracluster consistency*. Whenever any pair within a cluster shows a correlation coefficient below threshold, re-evaluation is needed. The algorithms below cover both.

Algorithm 3 Redundancy Analysis

Dataset S, Signal s, Correlation Matrix X, Cluster c

```
for i = 1 to S.size do
    for j = 1 to S.size do
        X_ij ← crosscorr(S_i, S_j)
        c_i ← S_i.getCluster
        c_j ← S_j.getCluster
        if c_i! = 0 then
            if c_j = c_i then
                c_j ← c_i
            else if c_j! = 0 then
                c_i ← c_j
            else
                c_i ← newCluster
                c_j ← c_i
```

Algorithm 4 Consistency Check

Cluster Set C, Cluster c, Correlation Matrix X, Threshold t

```
for each c in C do
    Load X for c
    for i = 1 to S.size do
        for j = 1 to S.size do
            if X_ij < t then
                Conflicting Result
```

Complexity Analysis

From all the redundant information within each cluster, we want to make sure that the one that is picked actually is the most interesting one. More specifically, We want to maximize

the probability of being able to identify patterns from the signals which are finally passed on to a learning algorithm. This idea is represented in time series complexity. The review in Section 2.4 revealed permutation entropy (PE) as the best choice, due to its positive traits and successful application on machine sensor data. The algorithm below selects the most complex signal from every cluster.

Algorithm 5 Complexity Analysis

Cluster Set C, Cluster c, Signal s, Selection S_{final}, Complexity h

```
for each c in C do
    for each s in c do
        h ← permutationEntropy(s)
        if h > max_h then
            max_h ← h
            s_max ← s
    S_final.add(s_max)
```

4.2 Machine Learning Approach

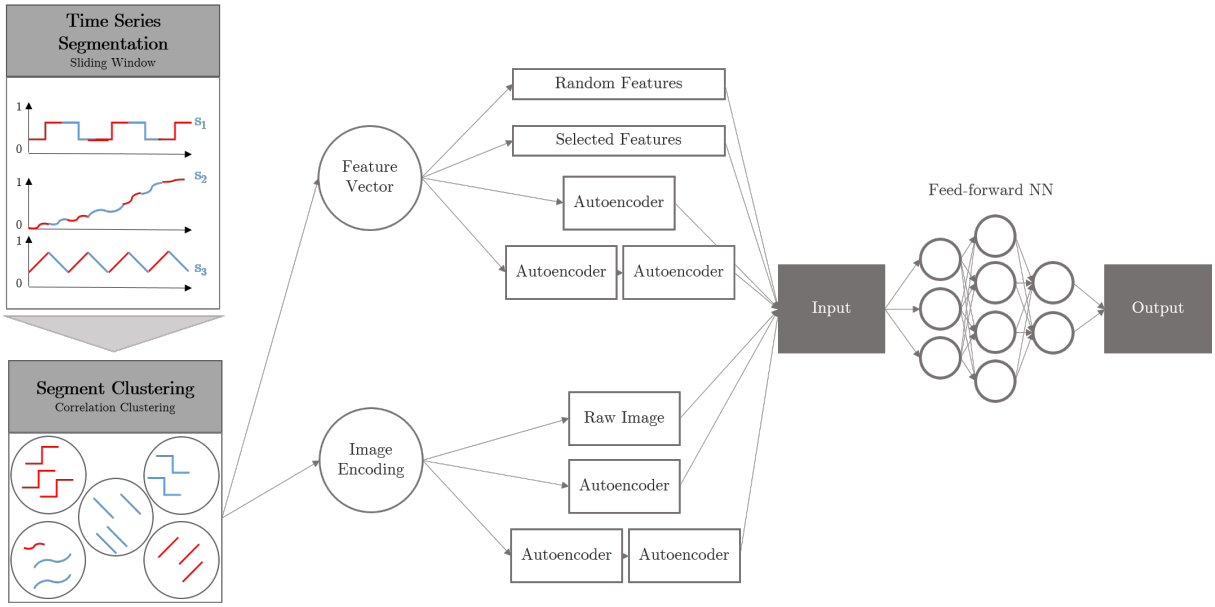


Figure 15: Approach for classifying machine operations from sensor signals

In the following, the experimental setup for analyzing the potential of (deep) machine learning for machine state classification from time series segments shall be defined (illustrated in Figure 15). Instead of providing a set-in-stone approach, the analytics pipeline is constructed in a way that allows experimentation and comparison of different concepts to be able to make recommendations for future work in this field. In detail, the following assumptions should be clarified:

1. **Representations:** The time series segments need a representation that can be handed to a neural network. It is evaluated if a set of features containing information related to the physical process, such as differentiations, can outperform one

without those. Furthermore, the idea of time series image encoding is evaluated as an alternative to feature representations. For this purpose, recurrence plots, proposed in a similar context by [27] are used.

2. **Representation Learning:** It shall be evaluated if methods from deep learning, more specific autoencoders, are able to learn a proper representation from time series in an unsupervised manner. By proper, we mean that when used for classification, the achieved accuracy can be seen competitive to other representations.

3. **Multiple Network Layers:** Although no classic deep learning algorithm is in use, the power of representation learning using multiple layers is evaluated. This is done by training a second autoencoder using the features from the first one as an input. The 2-layer features showing higher quality would proof the ability of capturing a more complex structure.

It shall be noted that for all experiments, the amount of training data is kept small, as labelled data is usually a limited resource in time series problems and therefore is a realistic condition. Below, the main components of the experimental setup shall be outlined.

Segmentation

After defining a machine part and timespan that shall be analyzed, one signal is selected using the preprocessing methodology introduced in this section. To have segments to classify, the entire time series is split up using a simple sliding window approach. Its only parameter is the window size w. It is not possible to provide a reference point for the choice of w as this is data and problem dependent. In general, if one wants to classify machine operations, the length should be chosen in a way that a segment is long enough to capture them. The larger the window size, the more complex operations can be classified.

Clustering

Although unsupervised learning will be a part of this approach, the classification step requires labelled examples for training as well as classifier testing. For that reason, a way to cluster the computed segments is needed and an algorithm performing this needs to be developed. Before the computations can be performed, two parameter choices have to be made manually. Firstly, the number of clusters has to be determined. Via visual inspection of the signal, one has to identify the main parts it consists of. The number of clusters will define the number of classes in the classification problem. Note that this approach can only be recommended for time series that show recurring behaviour, which is mainly true for manufacturing data. If used on datasets with random behaviour, the choice of clusters will be very difficult. The same argument applies to the second parameter, the cluster centroids. Those are representative segments for each cluster. They should be chosen in a way that they show high correlation to all segments belonging to this cluster, as cross-correlation is employed as a similarity measure in the algorithm. This, again, can only be done via visual inspection.
The algorithm then performs:

1. Perform cross-correlation between segment s_i and each cluster centroid $c_1 - c_j$.

2. Choose the cluster index with maximum correlation as the label.

3. Repeat for all segments.

Figure 16 provides an illustration.

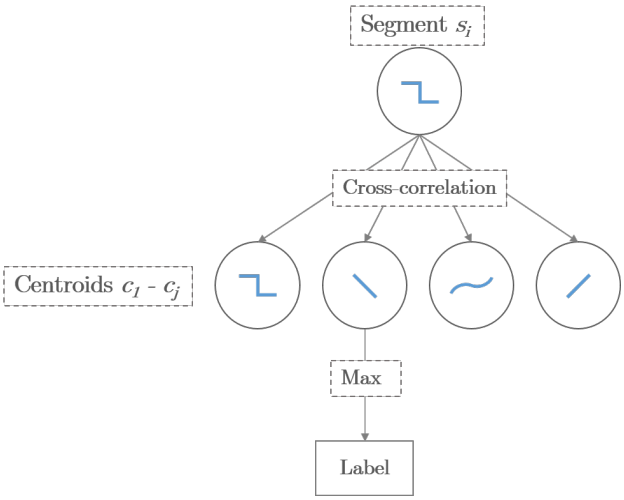


Figure 16: Segment clustering methodology

Time Series Features

A time series segment will be presented to the classification in a represented form. A common approach to this is to represent the data by a feature vector, which is a set of variables being discriminative with respect to the classification problem. As a reference point, a set of features will be picked from prior, comparable studies such as [84, 36], where several possibilities are listed. The set is picked randomly and restricted to the time domain.

1. Maximum value, x_{max}.
2. Minimum value, x_{min}.
3. Standard deviation,
$$x_{std} = (\frac{1}{n-1}\sum_{t=1}^{n}(x_t - \bar{x})^2)^{\frac{1}{2}}. \tag{46}$$
4. Skewness, describing the symmetry of the probability density function of the time series amplitude,
$$x_{skew} = \frac{\sum i = 1^n(x_i - \bar{x})^3)}{(n-1)x_{std}^3}. \tag{47}$$
5. Kurtosis, describing the peak degree of the probability density function,
$$x_{kur} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^4}{(n-1)x_{std}^4}. \tag{48}$$
6. Mean value, $\bar{x}$

To evaluate if physical information can enhance classification, differential features are used in the second feature set. Three of the six random features are replaced, the other three stay the same to make the comparison as accurate as possible. Please note that this is by no means an effort to select the optimal features or the optimal number of features. The feature sets are used experimentally to show the superiority of the added features:

1. First differentiation, $x_{d1} = \frac{x}{dx}$.
2. Second differentiation, $x_{d2} = \frac{x_{d1}}{dx}$.
3. Average permutation entropy, $\bar{x}_{pe}$.
4. Skewness
5. Kurtosis
6. Mean

Permutation entropy, described in Section 2, is also used as it provided useful insights in preprocessing that could possibly improve classification.

Recurrence Plots

Recurrence plots been briefly introduced in Section 3. Shortly, they offer a visualization of the recurring elements in a process. They are chosen as it seems reasonable to to use a representation based on recurrence for classifying segments that are supposed to represent frequent pattern. An additional motivation comes from the successful application in [27]. Alternative time series image encoding methodologies can be found in [90].

Autoencoders

Autoencoders are introduced in Section 3 and their basics shall not be repeated. They are used to extract features from time series segments, and the segments will be used directly as the input. To allow a comparison with selected features, the extracted feature vector should also have a dimension of six. That means that the autoencoder reduces the input dimension to a compressed representation. Figure 17 illustrates this process.

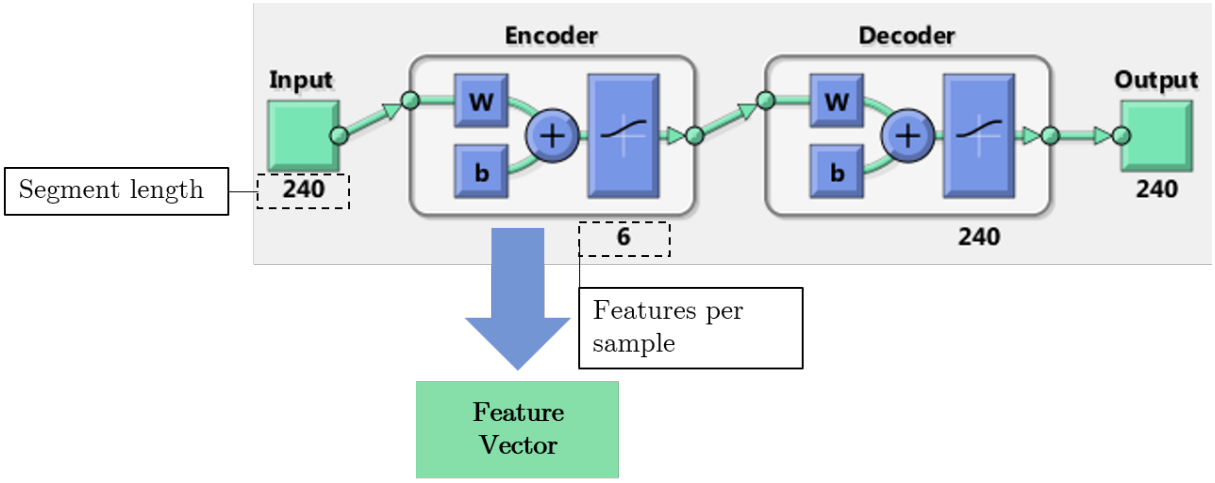


Figure 17: Feature extraction from autoencoder

During its training, the encoder transforms the input into the representation and the decoder processes it back to into the output, being time series data. The features learnt during this process can be extracted from the encoding layer. For the 2-layer approach, the feature representation is achieved in two steps. The features extracted from the first encoder are used as an input for the second one. Therefore, a larger dimension for the

hidden layer can be selected in the first autoencoder. A visualization of this principle can be seen in 18.

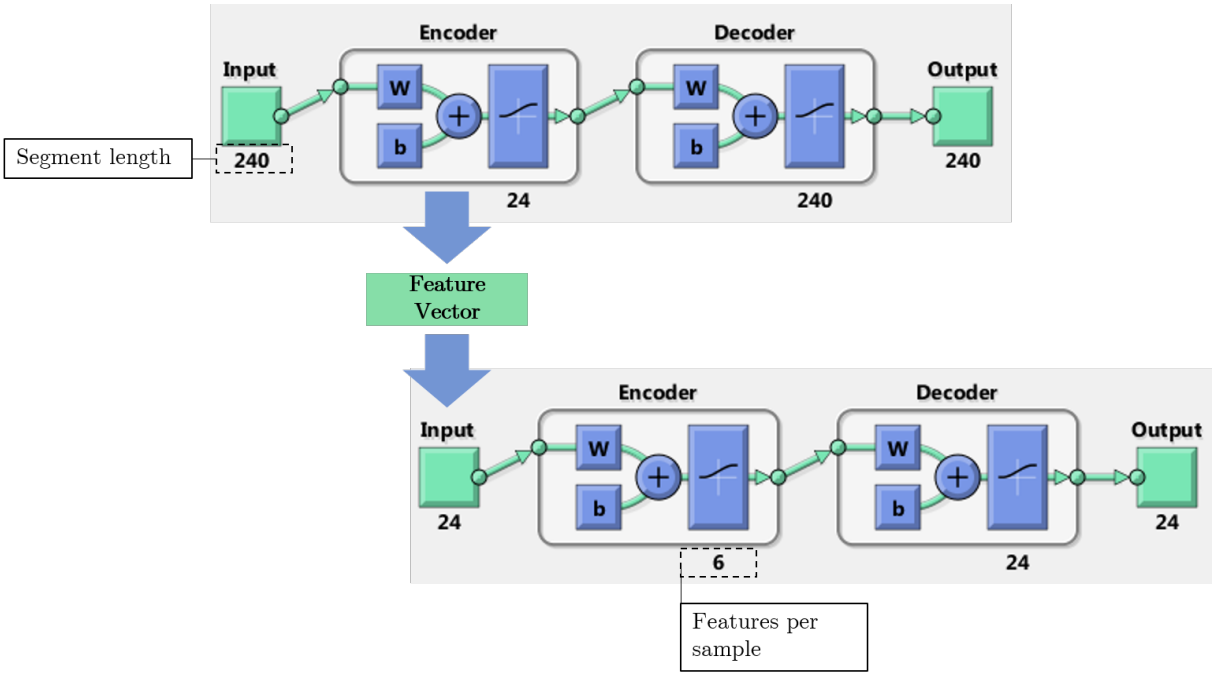


Figure 18: Two-step feature extraction through autoencoders

To achieve the best possible result, several parameters of the sparse autoencoder have to be adjusted. The best selection will be found during experimentation (Section 5) evaluating different combinations of the main parameters, shortly outlined below.

- **Error Transfer Function.** Defines the activation function for the autoencoder units. This has to be defined for both the encoding and the decoding part. Examples include the logistic sigmoid function,

$$f(z) = \frac{1}{1 + \mathrm{e}^{-z}}, \tag{49}$$

  or a linear transfer function,

$$f(z) = z. \tag{50}$$

- **Loss function for training.** This function measures how close the reconstructed output is to the input. For sparse autoencoders, the mean squared error can be used with additional terms for weight and sparsity

$$E = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} (x_{kn} - \hat{x}_{kn})^2 + \lambda \omega_{weights} + \beta \omega_{sparsity} \tag{51}$$

- **Weight regularization.** Defines how strong L2 regularization will influence the weights
- **Sparsity regularization.** Defines a sparsity constraint on the hidden layer output.
- **Training algorithm.** Defines the procedure of propagating data through the autoencoder and performing weight updates. An often used variant is scaled conjugate gradient descent [59].

The experiments are carried out using the *Autoencoder* class of the *MATLAB Neural Network Toolbox*.

Feed Forward Neural Network

To perform a comparison on the different variants of our methodology, a supervised classification has to be performed using the computed features. This is done using a classic feed forward neural network [77]. Concerning parameter optimization, it should be noted that similar conditions shall be simulated to make feature quality comparable. Slight optimizations can still be made if improvement can be spotted. The network takes the representation as an input and provides a classification result. The example in Figure shows the setup for a threelabel classification problem taking recurrence plots as an input.

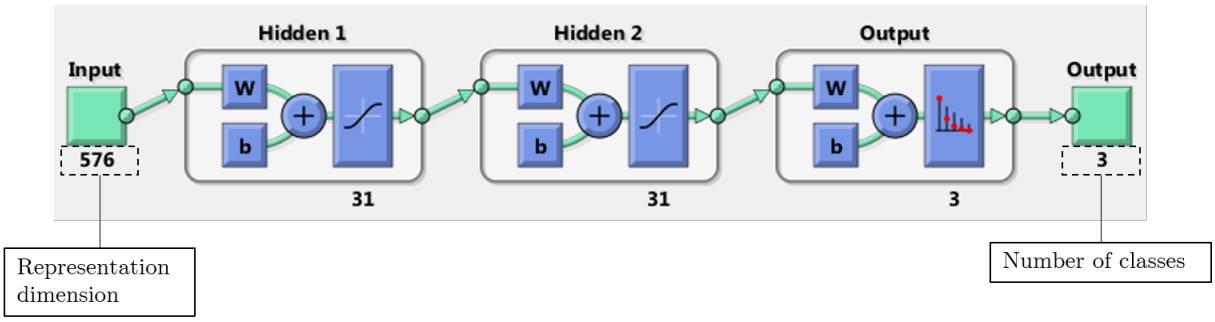


Figure 19: Feed forward neural network performing classification using feature representation

5 Experimental Evaluation

5.1 Data Sources

Before the results of the actual analytics operations on the data are shown, it is important to introduce the datasets used and explain some of their crucial features. This is necessary, as the way of processing it is clearly influenced by those properties and this information will be helpful in understanding why specific steps were chosen.
The analytical methods developed within this work are evaluated on datasets measured by sensor networks on three different manufacturing sites. The respective machines are large size bulk handling systems. More specifically, we are dealing with three bucket wheel reclaimers, whereby one of them is executed as a ship loading system. Each of them is equipped with several sensors. As they are transmitting simultaneously, the result is a multidimensional dataset, paraphrased multivariate time series. Some of the issues with this type of data have already been discussed.
The main differences between the three datasets are the number of sensors installed on the machine and therefore the time series dimensionality, as well as the amount of data available. In addition, the sensors can be grouped according to the machine parts they are installed on. This matters, as signals from the same group will usually show more mutual information and stronger correlation. This groups of course are varying from machine to machine as well.
Figure 20 provides an overview.

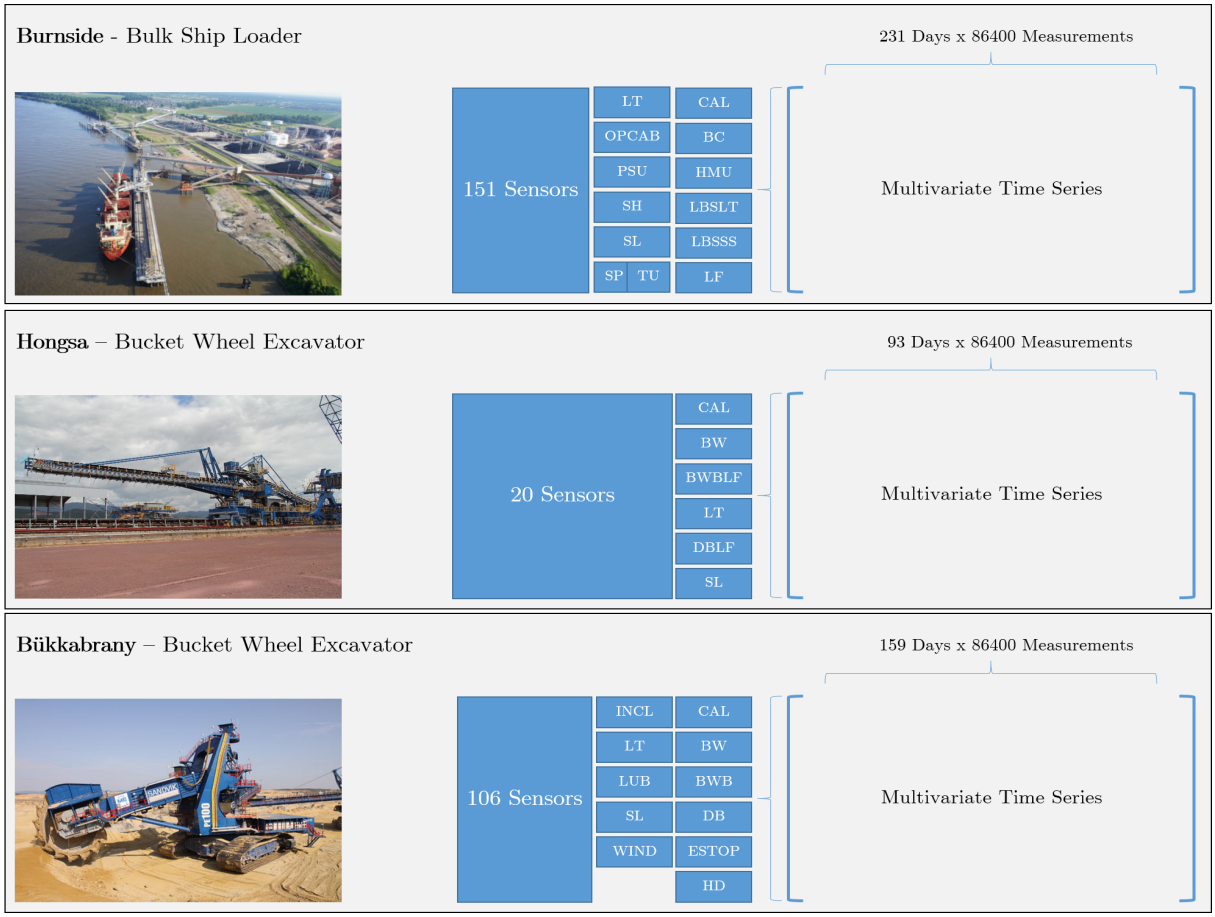


Figure 20: The three sites subject to analysis

As stated, several important properties of the acquisition process are presented. A more complete description can be found in [64].

- **Measurement.** Each sensor is transmitting with a sampling period of one second, therefore at a frequency of one Hertz. This leads to 86400 data points per sensor per day. The monitoring is continuous.
- **Batch Data.** For analysis, the data is available in daily batches. This creates a task that is clearly different from working with data streams and performing real time computations.
- **Data Structure.** To achieve a unique identification, each measurement is treated as a data object described by three attributes: *Timestamp, Channel Tag* and *Sensor Value*. Appended to each other, these data elements form a time series.
- **Interconnectability.** By checking the first and last values in the daily batches for consistency, seamless interconnectability is assured. Datasets of arbitrary length can be created.
- **Sensors and Actors.** Data is collected from sensors and actors to maximize insights into system behaviour.

Furthermore, the data provides several challenges that were discovered during investigation.

- **Null Signals.** The number of measurements of value 0 is high. This is mainly caused by inactivity of the entire machine or parts of it. Signals with value 0 during the entire period to analyze are less of a problem as they can be easily identified and removed. Relatively long periods of inactivity interrupted by shorter active operations are complicating analysis, as we end up with a long time series containing little information.
- **File Size.** The size of the daily datasets varies a lot. This is caused by inactive periods and incomplete transfer. It can be dealt with by not considering days for which the dataset has a size below a threshold. This is easy to include into an automated approach but bears the risk of information loss.
- **Change in Sensor Network.** Sensor removal, replacement or subsequent installation lead to inconsistency in the multivariate time series.

5.2 Data Quality and Forming an MTS Dataset

To form a time series dataset to analyze, a set of sensors and a timeframe have to be selected. As stated, the data is segmented into daily portions. As long periods of machine inactivity can occur and these are only increasing the size of datasets without providing valuable information, any possibility to exclude those segments is welcome. A simple one can be realized by considering the size of the daily datasets. Their size significantly varies over time due to the varying degree of machine activity. This is revealed by an analysis visualized in Figure 21. Although being aware of the possible information loss, it was decided that days with not at least a third of average size will not be considered and flagged invalid. Intuitively, the probability of this periods containing crucial information is just not high enough to justify a negative influence on the results of all the rest.

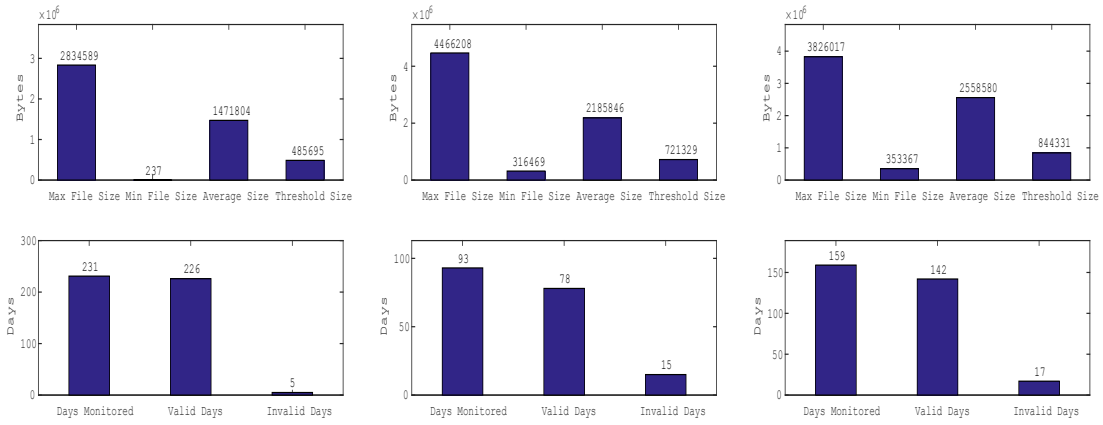


Figure 21: Varying sizes in daily sensor datasets

As the desired timeframe for analysis can exceed a day, it is necessary to concatenate data to form a larger MTS dataset, as illustrated in Figure 22.
As said, connectivity between day i and day $i+1$ is pre-assured. This, however, is not the case if datasets are left out following the size criteria defined previously. As this can not be avoided, day i then has to be connected to day $i+k$ with missing data in between.

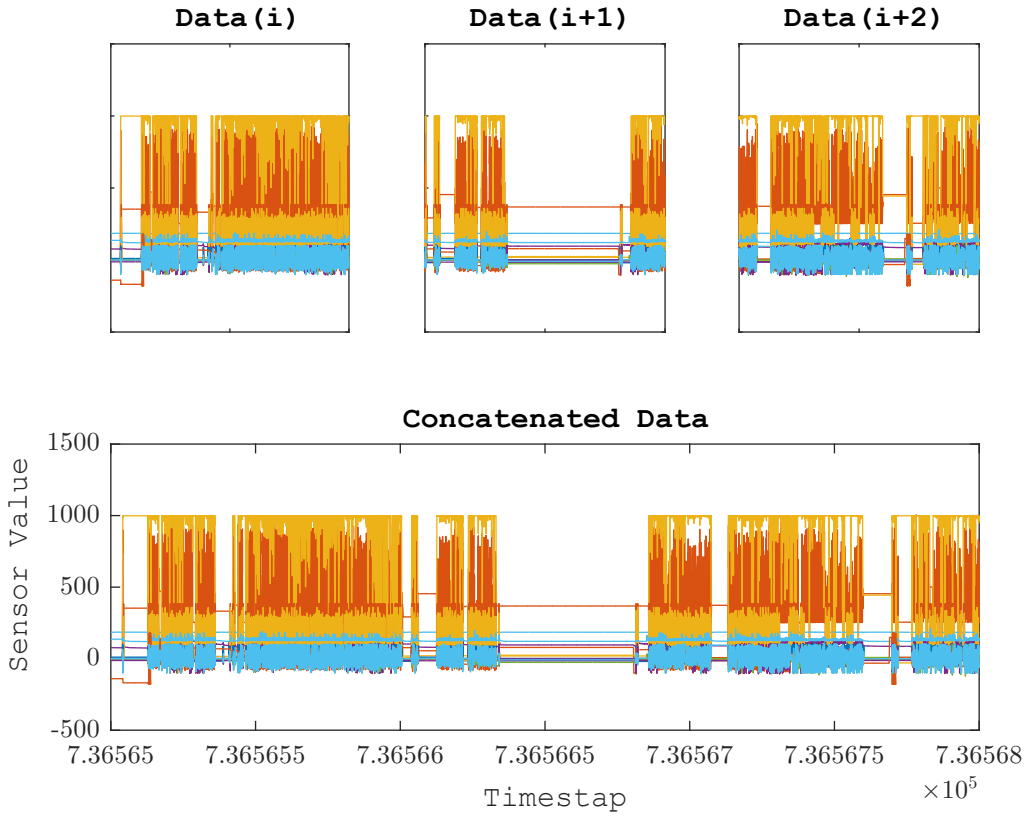


Figure 22: Dataset formed by concatenation

5.3 Time Series Selection

As a next step, an ideal set of time series is searched via the the algorithms introduced in section 4.1. The methods will be evaluated on the *Hongsa* dataset recorded from 23rd to

26th August 2016. After having 21 dimensions originally, after removing the timestamp, the calibration signal and one null signal, 18 remain. Considering four days with 86400 measurements each, we end up with a $345600x18$ multivariate time series dataset.

Stationarity is assessed with a combination of Augmented Dickey-Fuller test and KPSS test to avoid failures and create as much clear results as possible. During experimental evaluation, the effect of smoothing on stationarity assessments has been experienced. Without smoothing or with smoothing with a small window size, a majority of the signals was tested without a clear result. When applying a Savitzky-Golay filter [75] of cubic order and with a window size of 1000, the number of contradictive results drops significantly. Although this is not recommended, it seems to be necessary to achieve a stationarity test result. Please note that no information is lost as the original data can used for further steps The result is presented in Table 1.
The assessment classified 15 out of 18 signals as nonstationary. For them, the ADF test fails to reject the null hypothesis of a unit root and the KPSS does reject H_0 of trend stationarity. Three signals lead to unclear results, as both tests reject the hypothesis. Such a result is not uncommon on a sensor dataset. In [54, 41], nonstationary results

Sensor Name *Tag*	**Result ADF** H_0	**Result KPSS** H_0	**Stationarity Result** *Res*
bw-current-ValueY	0	1	Nonstationary
bw-speed-ValueY	0	1	Nonstationary
bwb-cyl1-pist-side-ValueY	0	1	Nonstationary
bwb-cyl2-pist-side-ValueY	1	1	No Result
bwb-cyl-angke-ValueY	0	1	Nonstationary
crw-current-m1-ValueY	0	1	Nonstationary
crw-current-m2-ValueY	0	1	Nonstationary
dsc-cyl-angle-ValueY	0	1	Nonstationary
dsc-cyl-pist-side-ValueY	0	1	Nonstationary
dsc-cyl-rod-side-ValueY	0	1	Nonstationary
slew-current-m1-ValueY	0	1	Nonstationary
slew-current-m2-ValueY	0	1	Nonstationary
slew-pos-ValueY	1	1	No Result
slew-speed-m1-ValueY	0	1	Nonstationary
slew-speed-m2-ValueY	1	1	No Result
slew-torque-m1-ValueY	0	1	Nonstationary
slew-torque-m2-ValueY	0	1	Nonstationary

Table 1: Results of stationarity assessment after smoothing with large window size.

were documented on sensor data. The reason might be that in larger signals of machine data, cyclic behaviour due to recurring operations becomes more likely. The presence of a single unit root in one signals is enough to produce a nonstationary results. Interpreting the smoothing effect on the results, noisy datasets are obviously harder to test for stationarity. Nevertheless, it is unlikely that the noise itself is classified as nonstationary. A better explanation is that in the presence of noise, it is more difficult to proof a unit root. For the signals with contradicting results, the obvious guess is that they are nonstationary, just not caused by a unit root. All the results were reassured with a third

test, the Phillips-Perron test [65]. Facing a result like that, it is decided not to exclude any signals from further steps, as there is no justification for that. Instead, the signals are transformed to stationary ones by applying differencing, introduced in Section 2.2.

Without having achieved a dimensionality reduction so far, redundancy analysis is the next step. The cross-correlation between every pair of the 18 signals is computed, lead ing to a correlation matrix (Figure 23). The goal now is to define a threshold and assign pairs with a cross-correlation coefficient above the threshold to the same cluster. Details can be found in section 4.1.
It is important to note that negative coefficients below the inverted threshold are also considered. Negatively correlated signals have simultaneous behaviour, just in the opposite direction, which is considered relevant in this context.

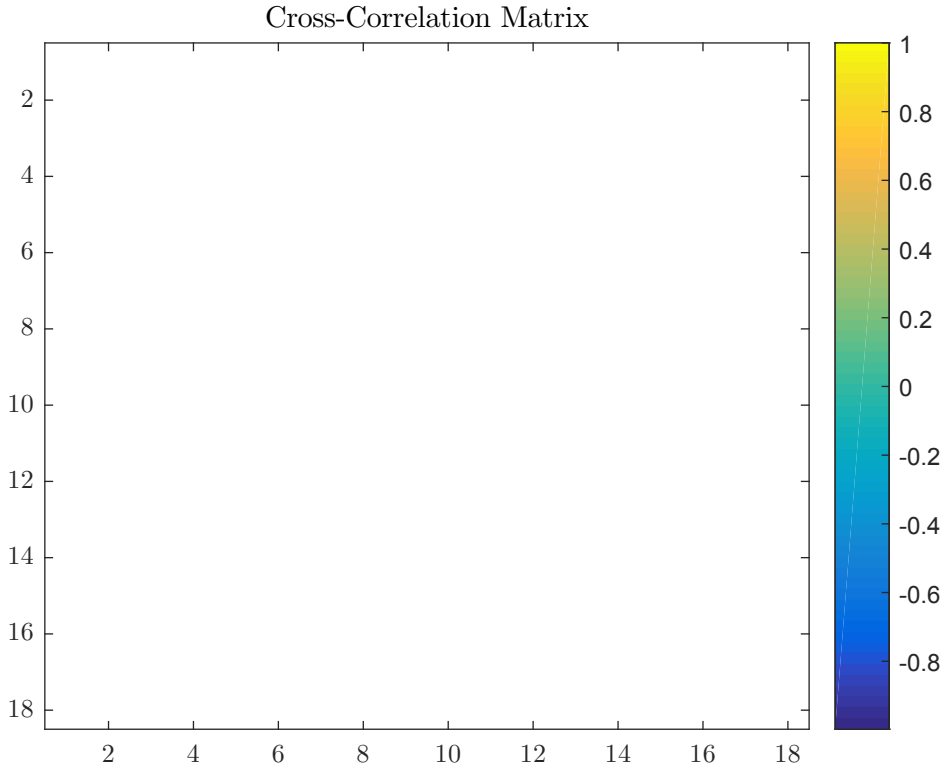
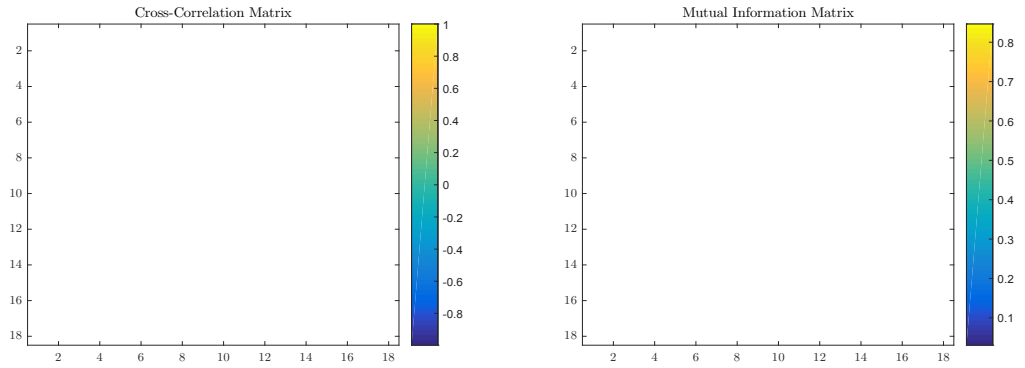


Figure 23: Cross-Correlation matrix for Hongsa dataset

As stated in section 2.3, cross-correlation (CC) can produce misleading results. For that reason, it can be interesting at the mutual information (MI) matrix to have a reference point. Unfortunately, the highest values in the CC matrix are not equally high in the MI matrix in all cases. This can be caused by CC focusing in linear dependencies while MI captures nonlinear relation. As linear dependencies are interesting in this context and there is no indication to doubt the results, they are used for further steps.



(a) Cross-Correlation matrix for Hongsa dataset

(b) Mutual Information matrix for Hongsa dataset

Figure 24: Comparison of Cross-Correlation and Mutual Information results

For this dataset, three redundancy clusters are identified and visualized in Figure 25.

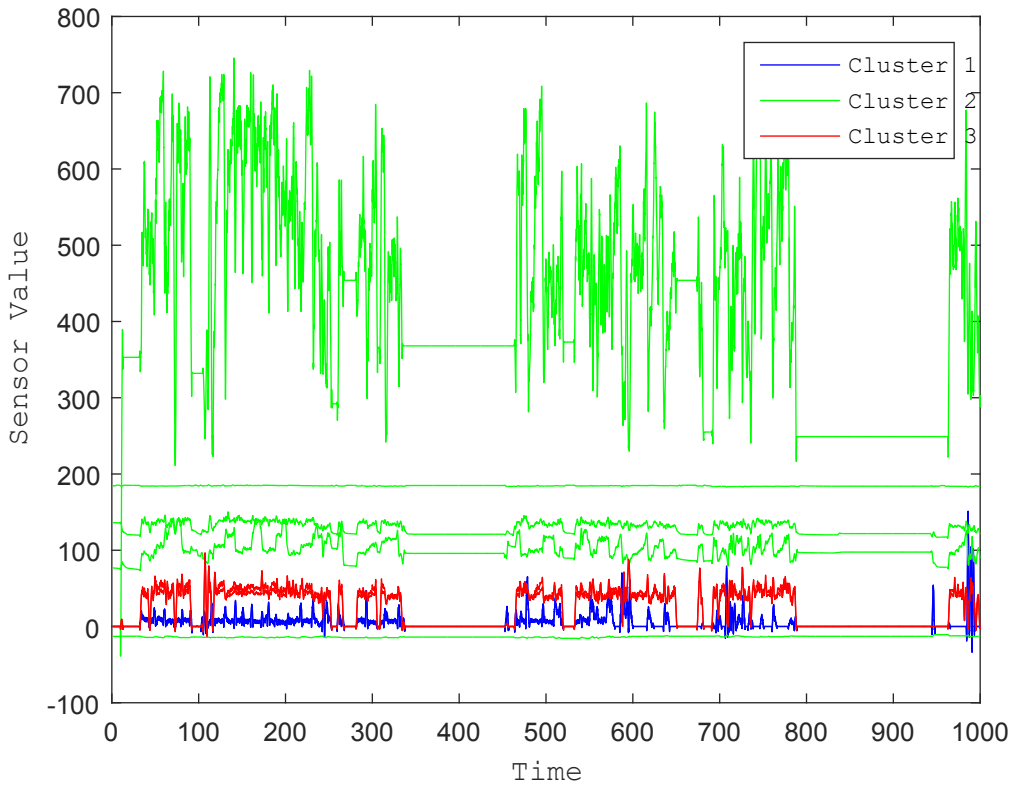


Figure 25: Three clusters were identified in the redundancy analysis

Now, the actual dimensionality reduction is achieved by selecting only one signal from each cluster. This is done by performing a robust version of permutation entropy (PE) [40]. A lower value indicates a more predictable, less random signal which is more promising for further analytics. Figure 26 shows how PE looks for two signals being in the same cluster, while Table 2 contains the results for all time series with high correlation including the selection.

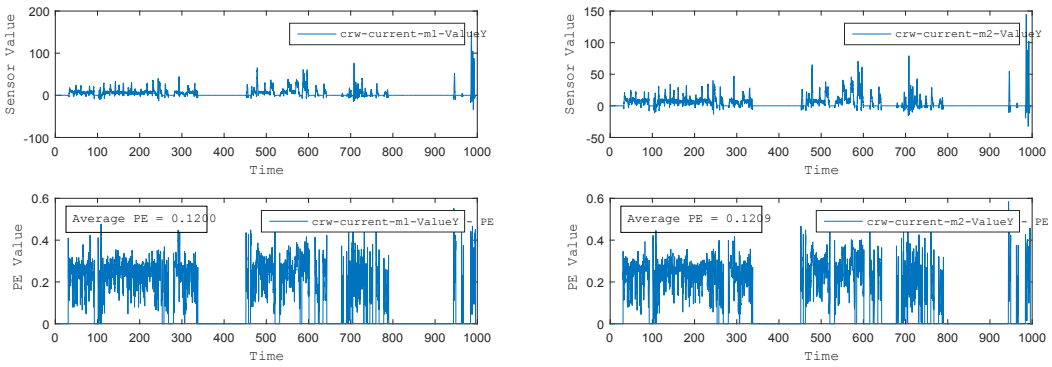


Figure 26: The left signal is selected from the cluster due to lower permutation entropy

As we can see here, the PE value allows to pick a representative channel for each cluster. A closer investigation of the results allows some conclusions on the suitability of permutation entropy for this task. Especially for clusters 2 and 3, the results lie within a very close range. This bears the risk that in case of computation inaccuracies or an unstable algorithm, the evaluation could end up as a random guess. The obvious explanation seems to be that a signals cross-correlation is correlated to its permutaion entropy. This is somewhat contradicted by both definitions, as correlation measures linear dependencies over the length of the time series while entropy refers to the predictability of every part based on previous ones. In addition, the results of cluster 1 show a much higher intracluster variance. Even in the very close case of cluster 2, PE selected to smoother and probably more predictable channel. Overall, the combined use of cross-correlation and permutation entropy can be considered useful and should not be omitted out of fear of performing redundant computations. The authors of [79] also reported a success as they developed a methodology to analyze time series coupling dependencies connecting ideas from both concepts. With regards to cross-correlation, the results of cluster 1 might suggest to increase the correlation threshold to only cluster signals with high similarities. This leads to 13 signals remaining and therefore a significant reduction of dimensions.

Sensor Name *Tag*	**Cluster** *C*	**Permutation Entropy** *PE*	**Selected** *y/n*
bw-current-ValueY	1	0.2572	no
bwb-cyl1-pist-side-ValueY	1	0.0891	yes
bwb-cyl2-pist-side-ValueY	1	0	no
dsc-cyl-angle-ValueY	1	0	no
dsc-cyl-rod-side-ValueY	1	0.0948	no
crw-current-m1-ValueY	2	0.1200	yes
crw-current-m2-ValueY	2	0.1209	no
slew-current-m1-ValueY	3	0.2524	yes
slew-current-m2-ValueY	3	0.2639	no

Table 2: Results of entropy-based selection from cross correlation clusters .

Considering that 15 signals were evaluated at the start and stationarity results could not be used for selection, this appears as a strong dimensionality reduction. As only 9 out

of the 18 signals showed high enough correlation to be assigned to a cluster, lowering the correlation threshold would lead to an even higher rate, but bears the risk of information loss. In general, it can be said that such methods seem to be highly data dependent and in need of adjustments.

After the signal selection abilities of the proposed method have been presented, the segmental investigation feature shall be evaluated. For the three main concepts used in preprocessing, being stationarity, cross-correlation and permutation entropy, the respective computations are broken down to time series segments. The investigative part thereby deals with the identification of anomalous segments. For a segment to be considered anomalous, it needs to have a value in one of the three categories that strongly differs from the rest of the signal. In the stationarity domain, no interesting results could be identified. After all channels were tested nonstationary, the initial guess was that those results were caused by a small number of nonstationary segments that can now be identified and investigated. However, it turned out that the majority of segments showed nonstationary characteristics and the ones that did not showed low informational content. Regarding cross-correlation, some anomalous events have been identified . In Figure 27 left, two channels showing high correlation are presented. Nevertheless, the highlighted timespan had a significantly lower coefficient when correlated with its parallel counterpart. Performing a visual inspection as a follow up on this result, it can be seen that the constant line interrupting the slewing is at a different level. Although it can not be clarified, this indicates that the operator has stopped the machine at a maximum slewing torque, which has only happened once in the entire timespan evaluated and can not be considered standard operation. The highlighted segment in Figure 27 right might not seem as anomalous, as it shows a decrease in temperature when motor activity stops. A closer look reveals that it is labelled because not all temperature decreases come with such an abrupt downshut.
Based on those results, it can be stated that correlation measures are suitable for anomaly detection in this manner, as rare differences between normally similar system behaviour will often be interesting events.

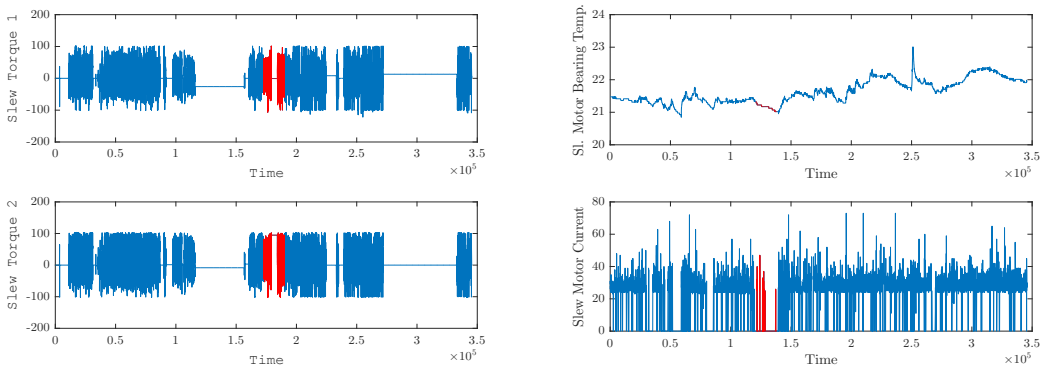


Figure 27: Anomalies detected via cross correlation segmental investigation

In Figure 28, a motor activity signal has been checked for segments with a relatively low permutation entropy. This resulted in the identification of different motor behaviours. This is especially interesting, as the signal pattern seems to change after this event, which could indicate a break between two different machine tasks. This shows that permutation entropy can be used in searching anomalies in sensor data from technical systems, which

has also been demonstrated in [58]. However, it should be pointed out that the execution of several experiments has created the impression that PE tends to identify segments with low activity. Therefore, it should be questioned if less expensive computations, such as variance, could lead to the same results.

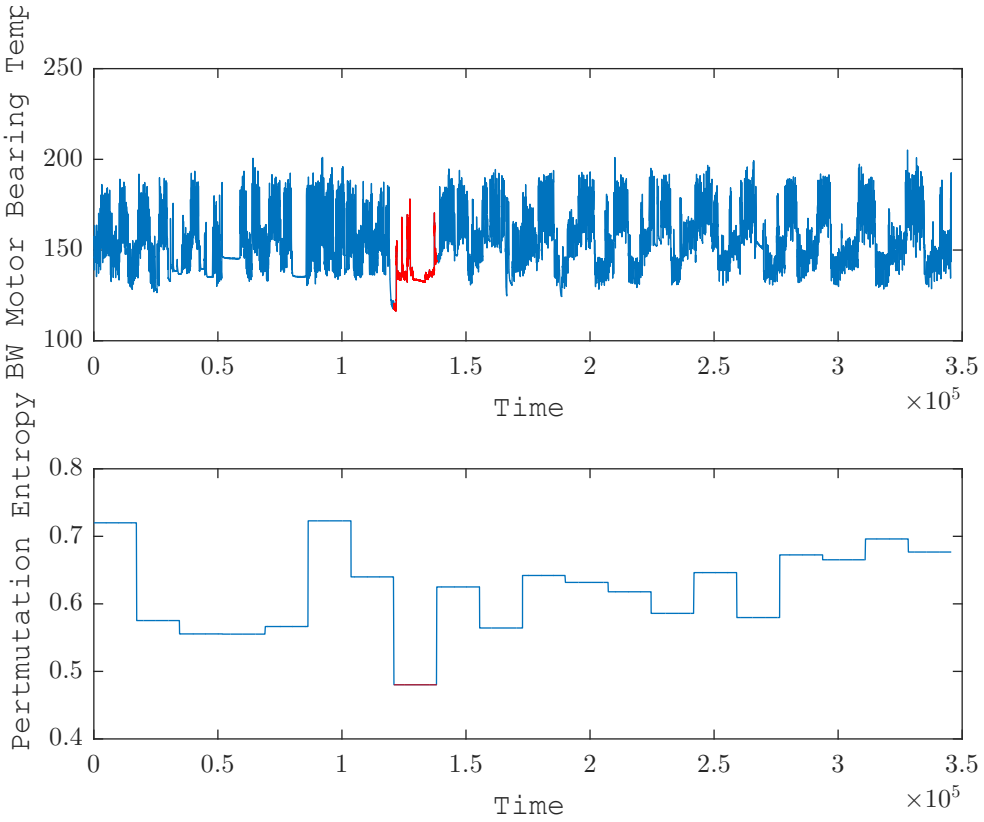


Figure 28: Lower permutation entropy of segment indicates anomaly

As large amounts of nonstationary data has to be processed, differencing (Section 2.2) provides a way to transform those into a stationary representation. It is, however, not an obligatory part of the preprocessing approach, and will only be used if poor machine learning results give a reason to do so. Figure 29 provides an example.

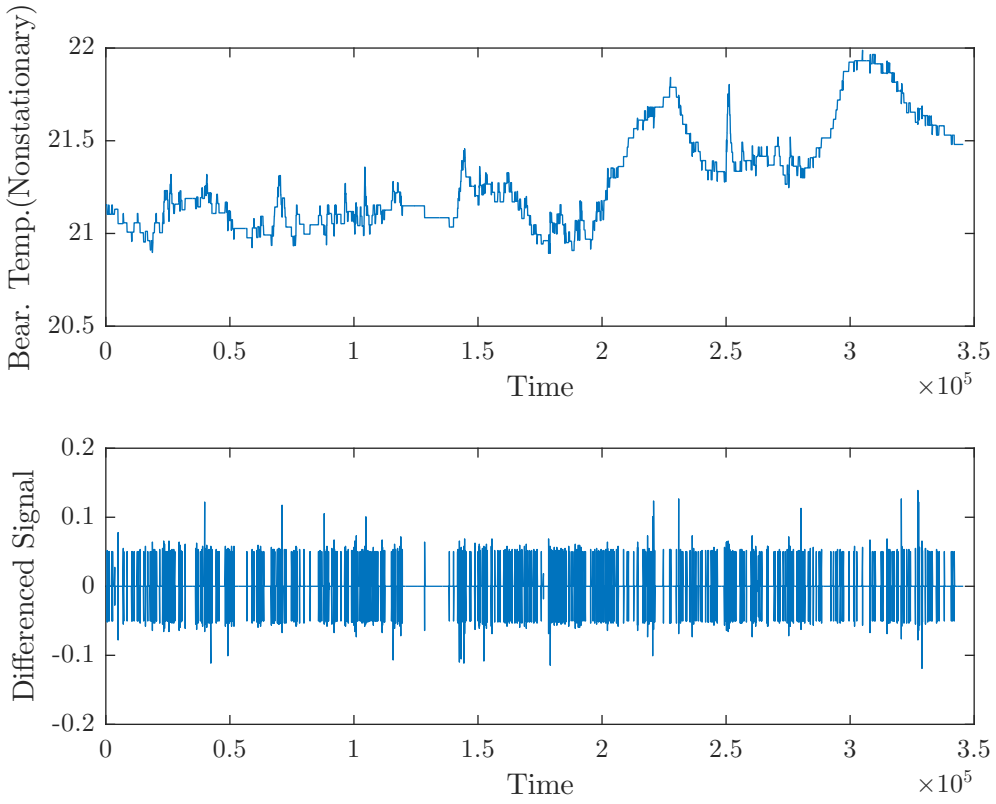


Figure 29: Differencing of nonstationary Signal

5.4 Classifying time series segments with neural networks

Now, the results produced by the proposed machine learning methodology shall be presented to answer the research questions introduced in Section 4.2. To do this, it is required to select a dataset. Through performing the developed selection algorithm on a specified set of machine groups, the two signals visualized in Figure 30 are selected.
Although the overall approach is able to handle a multivariate time series dataset, the machine learning part is geared towards analyzing a univariate series. It is decided to continue with the hydraulic pressure signal, as it offers interesting characteristics for analysis. Furthermore, a timespan for the analysis has to be determined. As already stated, a relatively short period is selected, as generating training data leads to a high effort and can be prone to error. Below, the key information on the data is outlined:

- **Sensor.** The respective time series is generated by a hydraulic pressure sensor installed on a discharged boom. It is part of the *Buekkabrany* dataset.
- **Length.** A time period of four days is chosen four analysis. This leads to a time series consisting of 345600 data points. A period with a relatively high activity rate is chosen, as data from non-activity days normally provide no information.
- **Range.** The signal has a symmetric value range, meaning that $x_{max} \sim -x_{min}$.

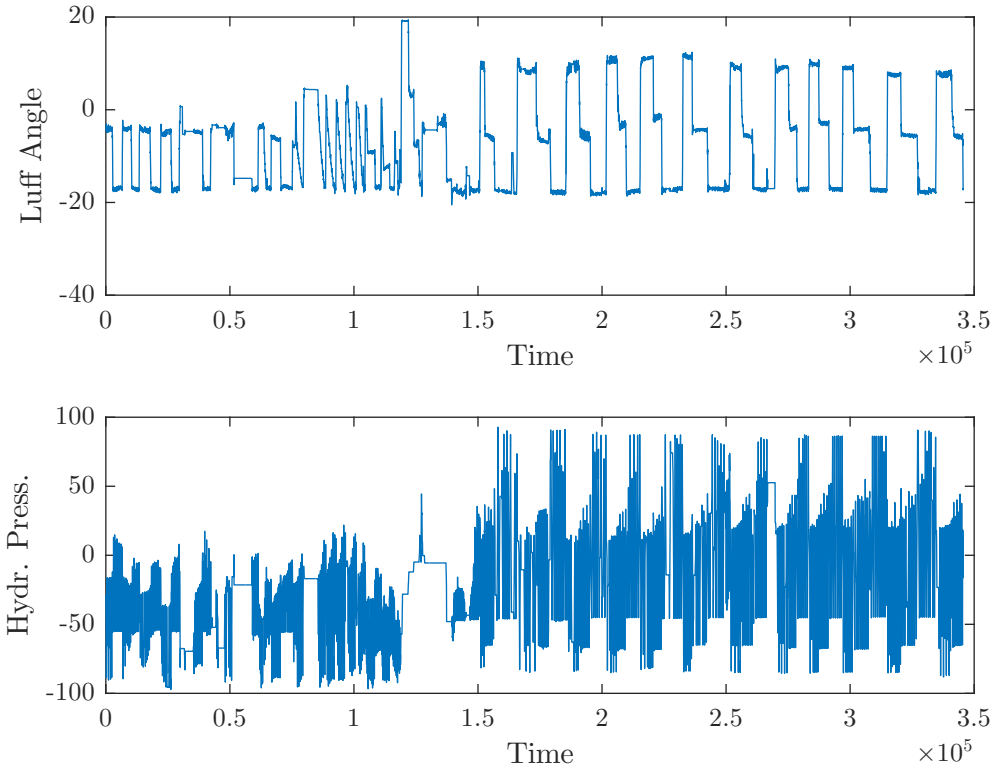


Figure 30: Two signals selected for classification

In order to be able to perform classification, it is necessary to identify classes that the dataset consists of. Before any computations can be done, we are required to look at the dataset and draw a conclusion on what characteristic events occur frequently, as only those are suitable class definitions. Considering the specifics of the pressure time series, it seems reasonable to base the classification around the signal direction. The following classes are considered the most valuable for machine process analysis:

1. Decreasing Pressure
2. Increasing Pressure
3. Constant Pressure
4. Positive Turn (decrease to increase)
5. Negative Turn (increase to decrease)

Note that the hydraulic pressure will depend on the load that is handled by the machine. It should be pointed out that the process analysis performed on this data will not go into too much detail on the technical specifications of the machine, but rather attempts to analyze the data with respect to different machine states on an abstract level. This concepts can be generalized to different processes, as long as the classes are chosen in a useful manner.
As the desired classes are now defined, all parts of the time series have to be labelled correctly. To achieve this, it is crucial two defined two parameters of the segmentation and clustering method introduced in section 4.1, being the length of the segments and

the cluster centroids. A proper manual choice of the segment length is important as the defined operations will only be recognizable in the right timespan. Referring to this concrete example, if segments are too short, it is less likely to capture a turn segment. If they are too long, it is not possible to capture an individual increasing/decreasing event, as multiple of them will occur in a long segment. It is determined manually that a segment length of four minutes, leading to 240 data points per segment are ideal.
Now, proper cluster centroids have to be picked. They should be a clear instance of their respective class to minimize clustering error. The chosen ones can be seen in Figure 31.

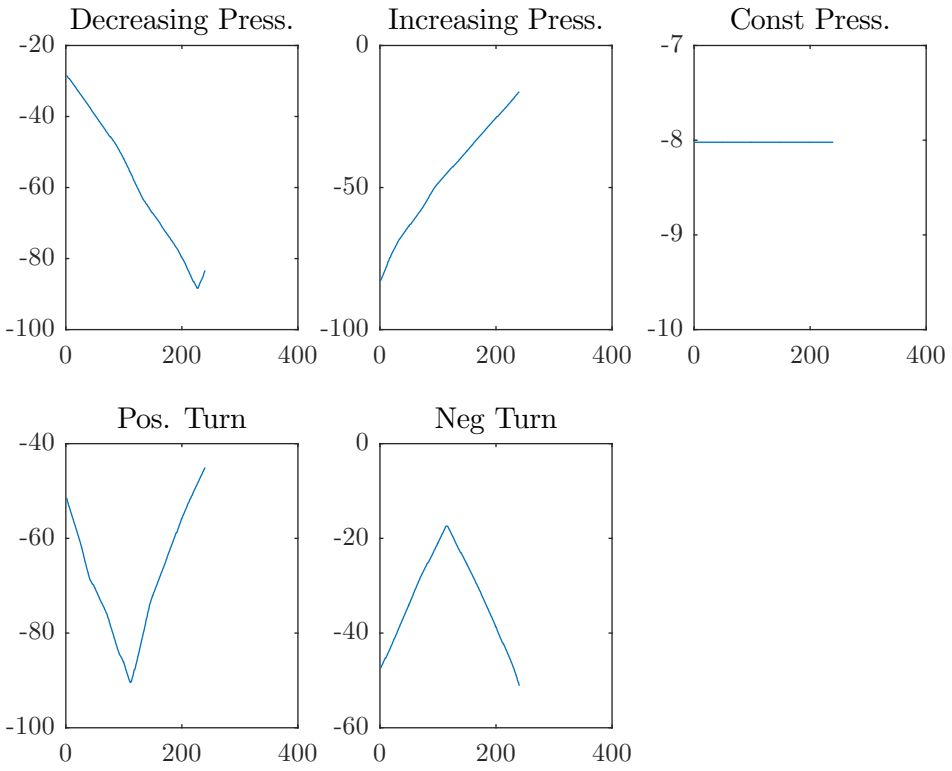


Figure 31: Cluster centroids for the five segment categories respective to signal and window size

The clustering algorithm now assigns a label to each segment. Table 3 shows the distribution of classes along the dataset. While an equal distribution will be desired in most classification tasks, such as digit classification, it is hard to achieve when labelling time series data automatically, as infrequent occurrences will be inevitable. In addition, it is pointed out that in case of long sequences of the same event, such as long inactive periods, it is advisable to shuffle the dataset before training, as this can harm classification accuracy. It has to be done in a way that indices are stored to avoid throwing away important process information. Furthermore, it should only be done if temporal relations between the samples are ignored. The analysis continues with performing experiments using a feed-forward neural network classifier. Network parameters are optimized using a trial-and-error approach. The best results are achieved using two hidden layers of size 31. The learning rate is set to 0.012, with the increase and decrease ratio being 1.08 and 0.7, respectively. Gradient descent with adaptive learning rate and momentum training is used. The data is divided in a 70/15/15 ratio into training, validation and test set.

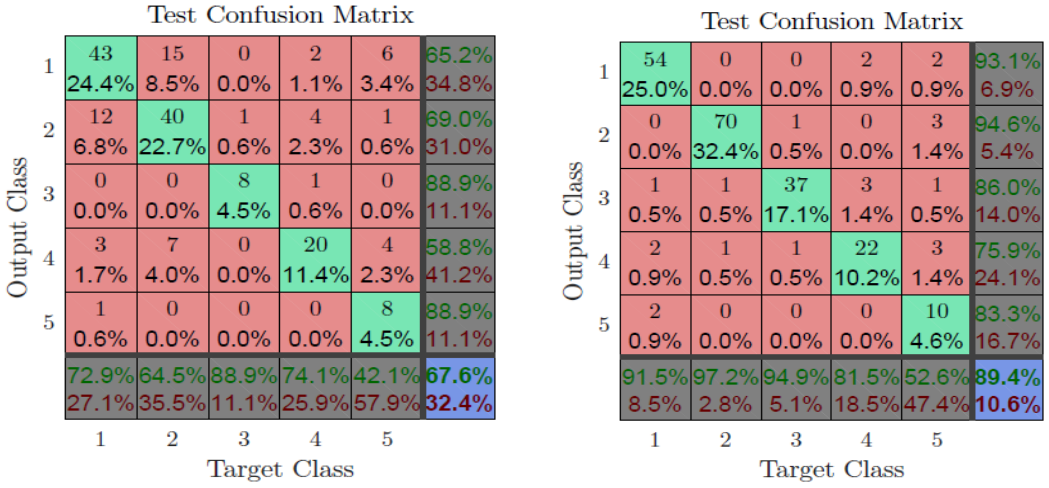
Segment Label	Number of Segments
Decreasing Pressure	421
Increasing Pressure	434
Constant Pressure	291
Positive Turn	172
Negative Turn	122

Table 3: Number of segments per defined event

For the autoencoders, we use two separate approaches. When a single autoencoder is used, the input features are directly transformed into a representation of the desired dimension. When the features are learned using two autoencoders, the first one uses a larger number of hidden layers. The parameters used for both are:

- Transfer function (encoder and decoder): *logistic sigmoid function*
- Maximum number of training epochs: *400*
- Loss function: *mean squared error with sparse adjusted*
- L2 weight regularization: *0.0009*
- Sparsity regularization: *1*
- Training algorithm: *scaled conjugate gradient descent*

Figure 32 shows the confusion matrix for the random feature set (left) and the selected features containing differential values and entropy measures (see section 4.2 for details). The results show a way better accuracy for the selected features. The misclassifications indicate that the first difference vector is the main reason for this success. When random features are used, the classifier seems to encounter difficulties distinguishing between increasing and decreasing segments, as a feature able to capture the signal direction seems to be missing.

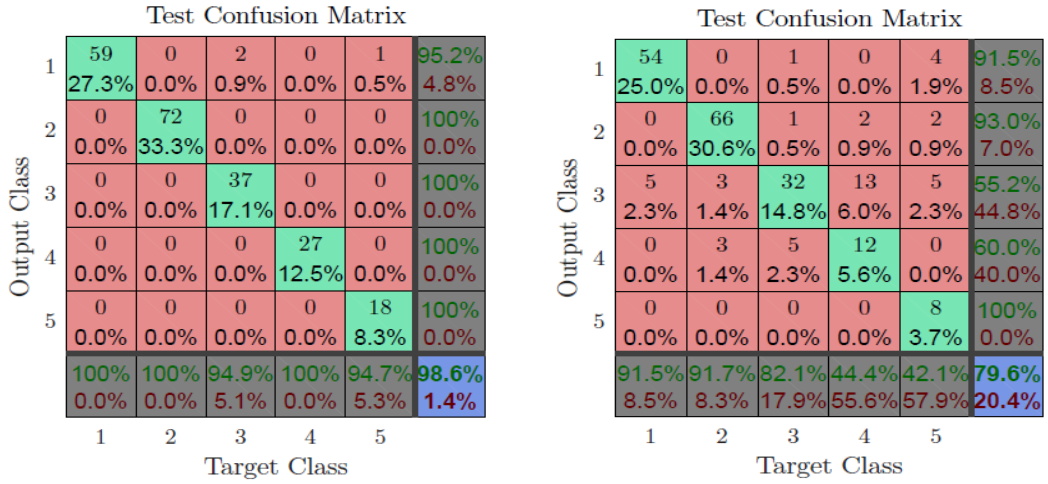


(a) Confusion matrix using random feature set

(b) Confusion matrix using selected features

Figure 32: Classification accuracy - selected features outperform random features

As a next step, the experiment is repeated with the feature vectors learned by autoencoders (33. The representation learned by a single sparse autoencoder leads to an excellent, almost perfect classification result, which emphasizes the representational power of autoencoders. It has to be noted that such a result could only be achieved by learning the features from a dataset with comparable feature composition. When data from lower activity periods is used, the performance is clearly worse, as discriminative features can not be extracted. When the two layer approach is used, the results deteriorate. Although the classification still works well for some classes, such as negative turn, the network now seems unable to identify the constant segments. As more abstract, higher level features are learned when multiple layers are used, they might not be suitable for something as simple as a straight line.



(a) Confusion matrix using features extracted by single autoencoder

(b) Confusion matrix using features extracted by two-layer autoencoder

Figure 33: Classification accuracy - results deteriorate with a second autoencoder

Now, the representative ability of recurrence plots needs to evaluated experimentally. Figure 34 shows the recurrence plots for the cluster centroids. Right away, a major flaw of this concept can be spotted. The plots of the classes 1 and 2 as well as 4 and 5 look almost identical, as recurrence plots represent the signal slope, but make no difference regarding its direction. This problem has not been encountered by other authors such as [27], as they deal with longer and more complex time series.

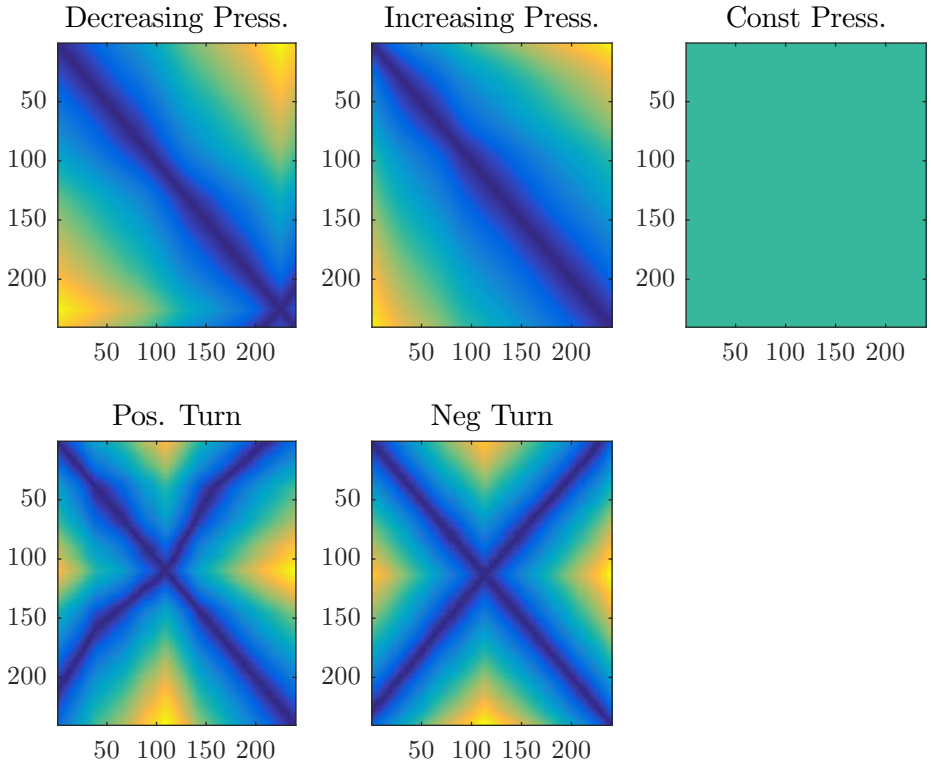
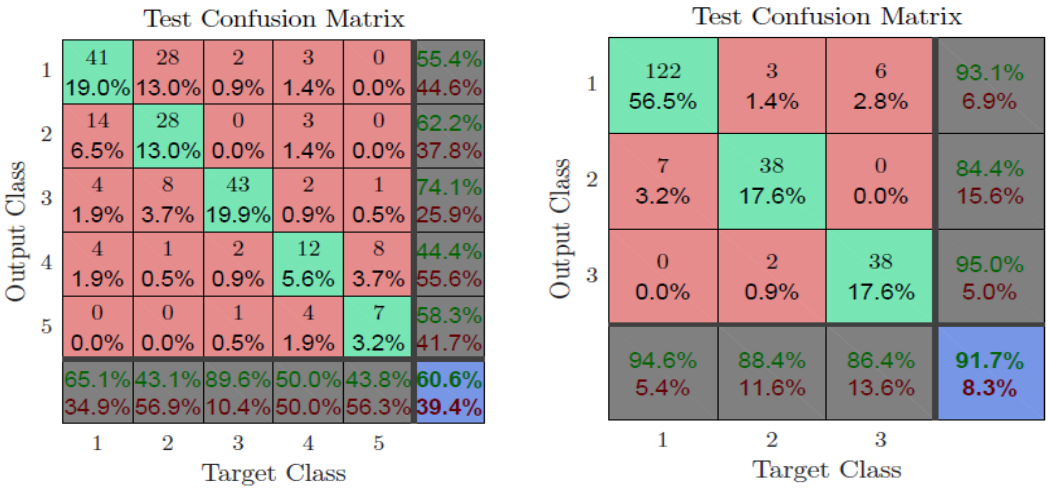


Figure 34: Recurrence plots for cluster centroids

This problem manifests itself in the classification results (Figure 35). The mentioned classes are misclassified to a high degree. To show that classifying recurrence plots does work in general, the problem is reduced to three classes. Now, increasing/decreasing as well as positive turn/negative turn share one label. In this setup, the accuracy rises from 60,6 to 91,7 percent.



(a) Confusion matrix using recurrence plots for five classes

(b) Confusion matrix using recurrence plots for three classes

Figure 35: Classification accuracy - recurrence plots fail to represent five classes

Finally, the feature extracting ability of autoencoders is tested on the image representation. As the features for a sample have to be provided to the network as a row vector, the recurrence plots have to be transformed into vectors and are stacked on top of each other, as visualized in Figure 36.

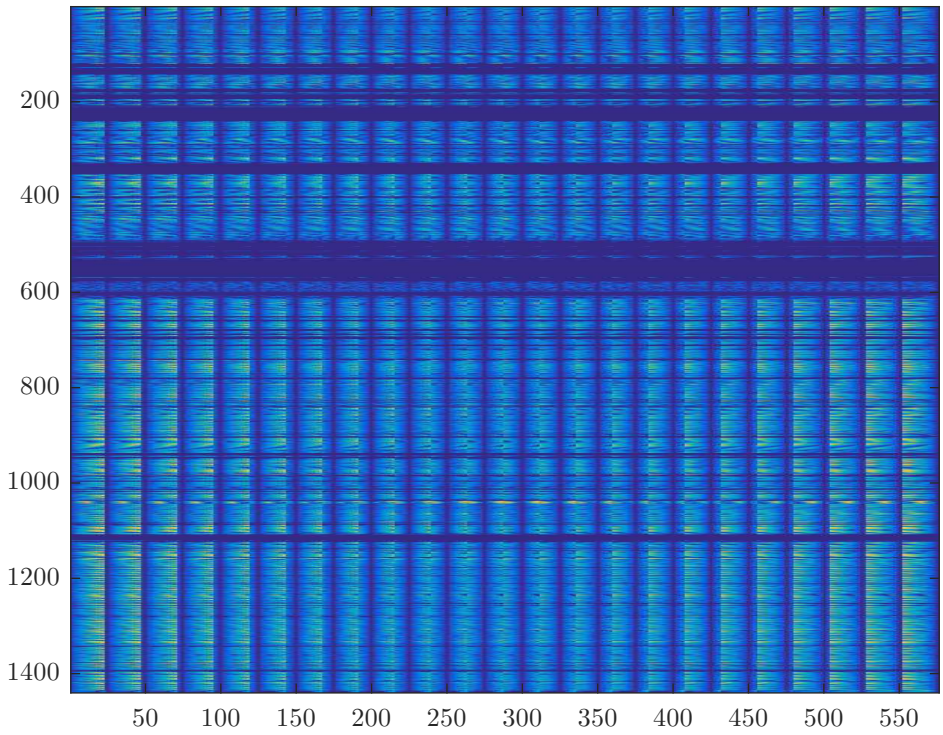
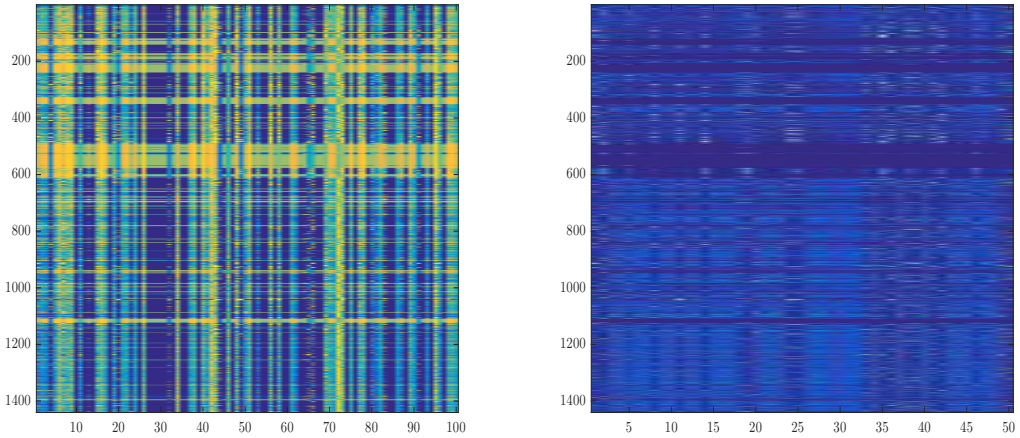


Figure 36: Recurrence plots aligned in stacked vectors for classification

Figure 37 presents a visualization of features extracted by the autoencoders. While one layer seems to learn a representation with a sharper realization of the edges in the dataset, the two layer approach produces a less accentuated version.

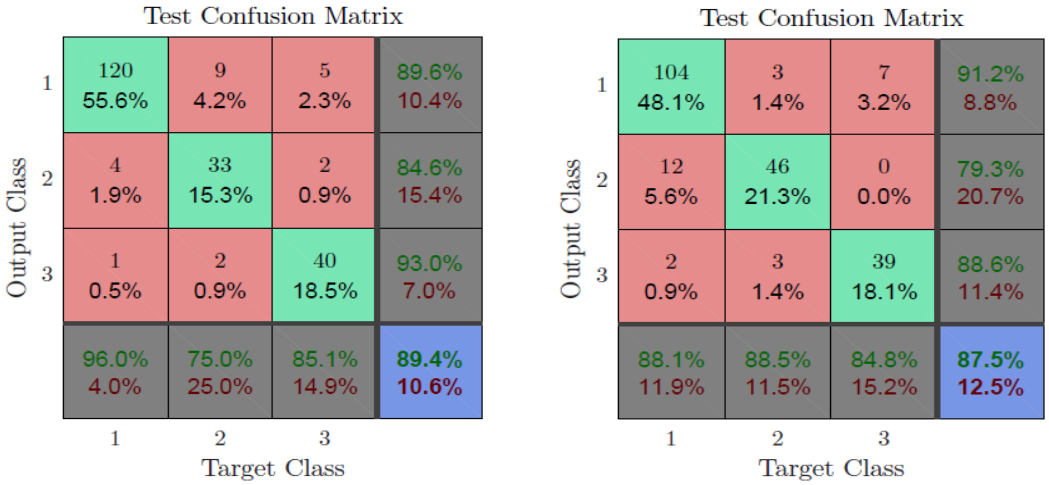


(a) Features extracted from RP representation by single autoencoder

(b) Features extracted from RP representation by second autoencoder

Figure 37: Visualization of features extracted by autoencoder networks through unsupervised learning

Those figures have more of an illustrative purpose, as feature quality can only be investigated through the classification experiment. For the three-class problem, both approaches fail to improve on the original result (Figure 38).



(a) Confusion matrix using single autoencoder features learnt from recurrence plots

(b) Confusion matrix using two-layer autoencoder features learnt from recurrence plots

Figure 38: Classification accuracy - no improvement through representation learning

A summary of the classification result is provided in Table This classification provides useful information on the machines operating process, as frequencies of certain machine states are known. This can be helpful in estimating maintenance intervals or creating key performance indicators. An example of a possible report generated from a classification result

Experimental Setup	**5-class Accuracy**	**3-class Accuracy**
Random Features	67,6%	-
Selected Features	89,4%	-
1-layer AE	98,6%	-
2-layer AE	79,6%	-
RP Representation	60,6%	91,7%
1-layer AE (RP)	-	89,4%
2-layer AE (RP)	-	87,5%

Table 4: Summarized classification results

Machine State	Occurrence	Time in State	Percentage
Pressure Decrease	59	236 min	28%
Pressure Increase	72	288 min	33%
Constant Pressure	37	148 min	17%
Positive Turn	27	108 min	12,5%
Negative Turn	18	72 min	8%

Figure 39: Example of process information included in classification result

Note that the results from Table 4 are from the test dataset, previously unknown to the algorithm. For a better understanding, some exemplary confusion matrices for the entire training process are provided. In general, the test error should be above the training error in most cases.

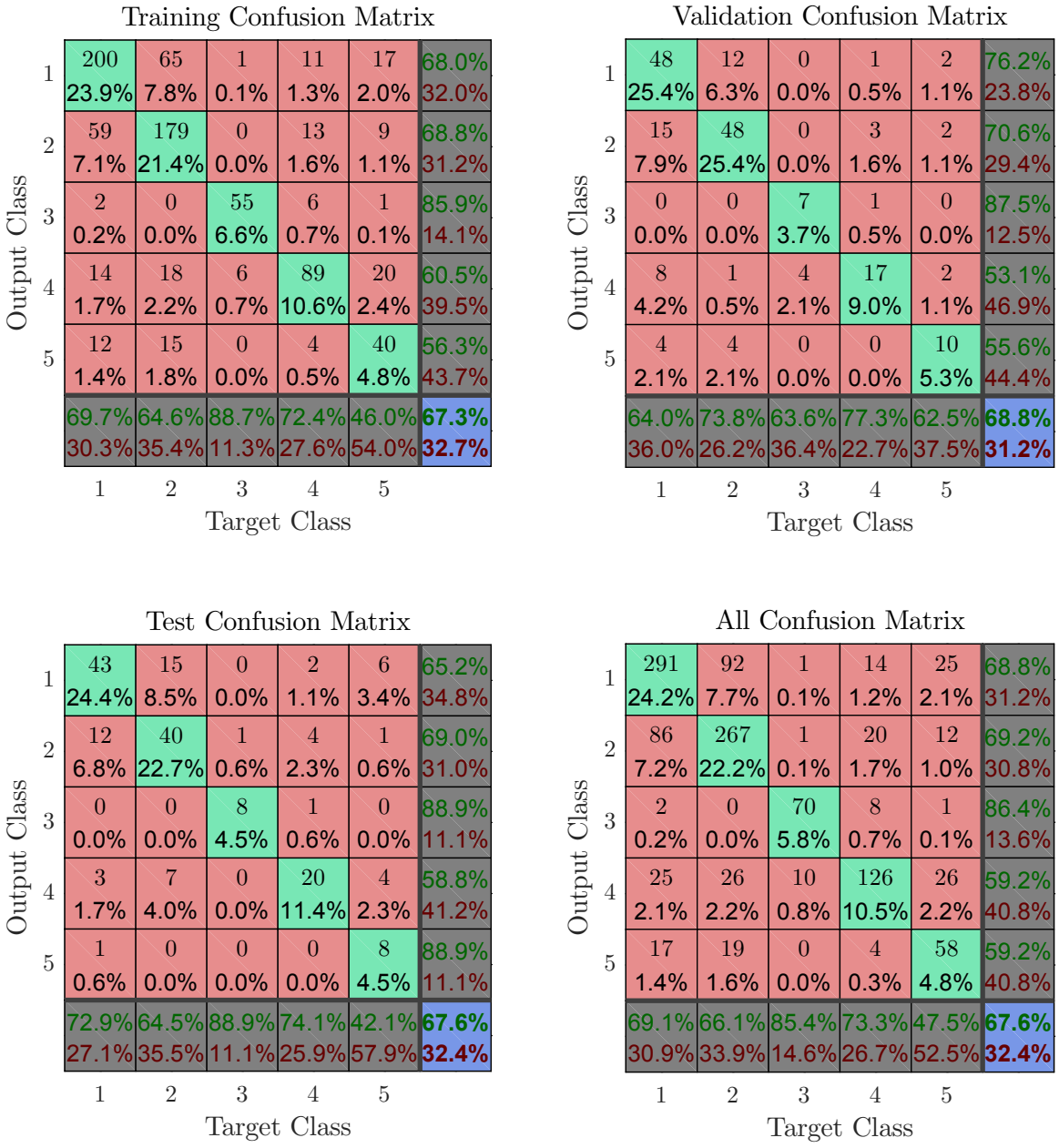


Figure 40: Confusion Matrix - Random Features

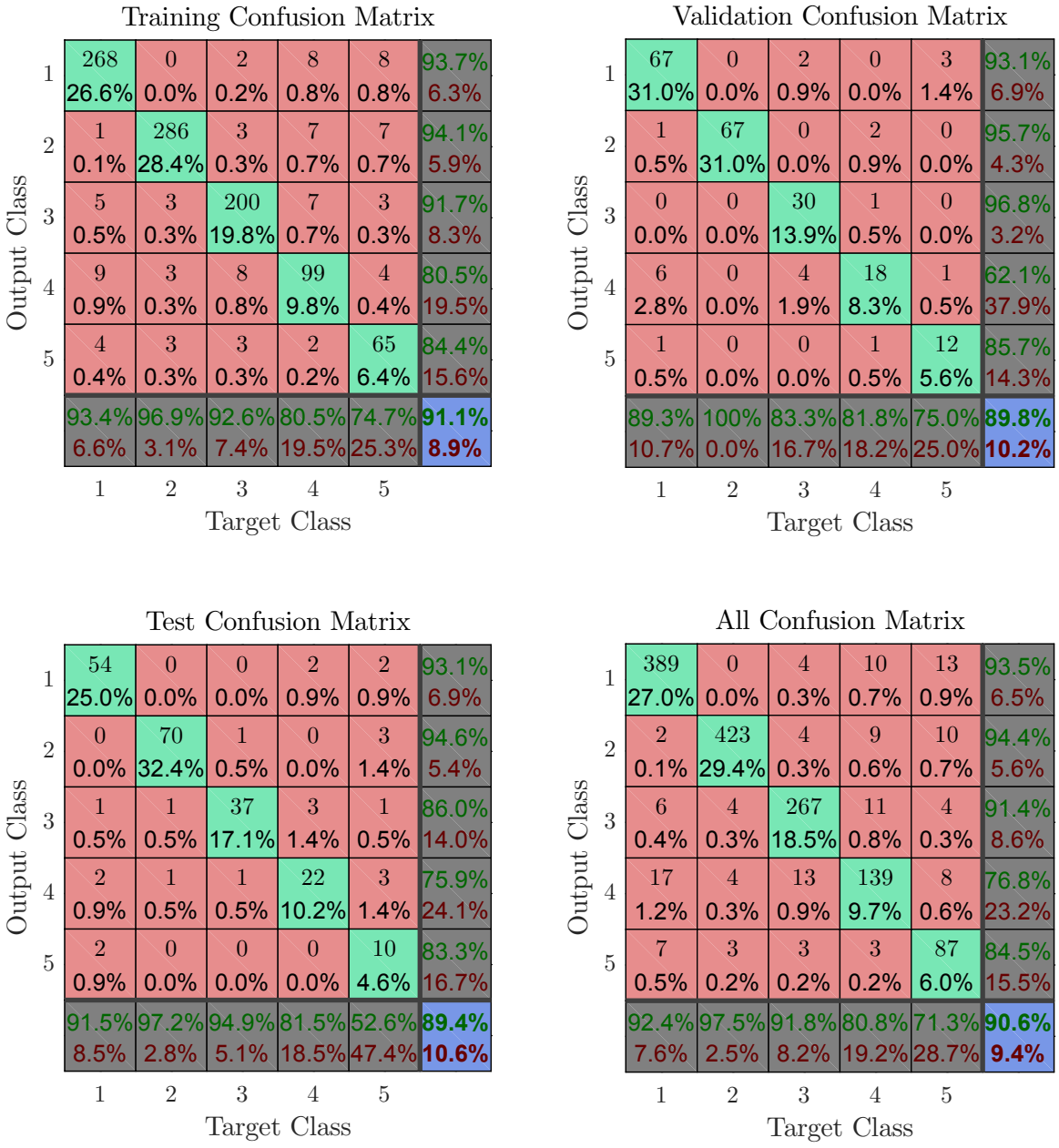


Figure 41: Confusion Matrix - Selected Features

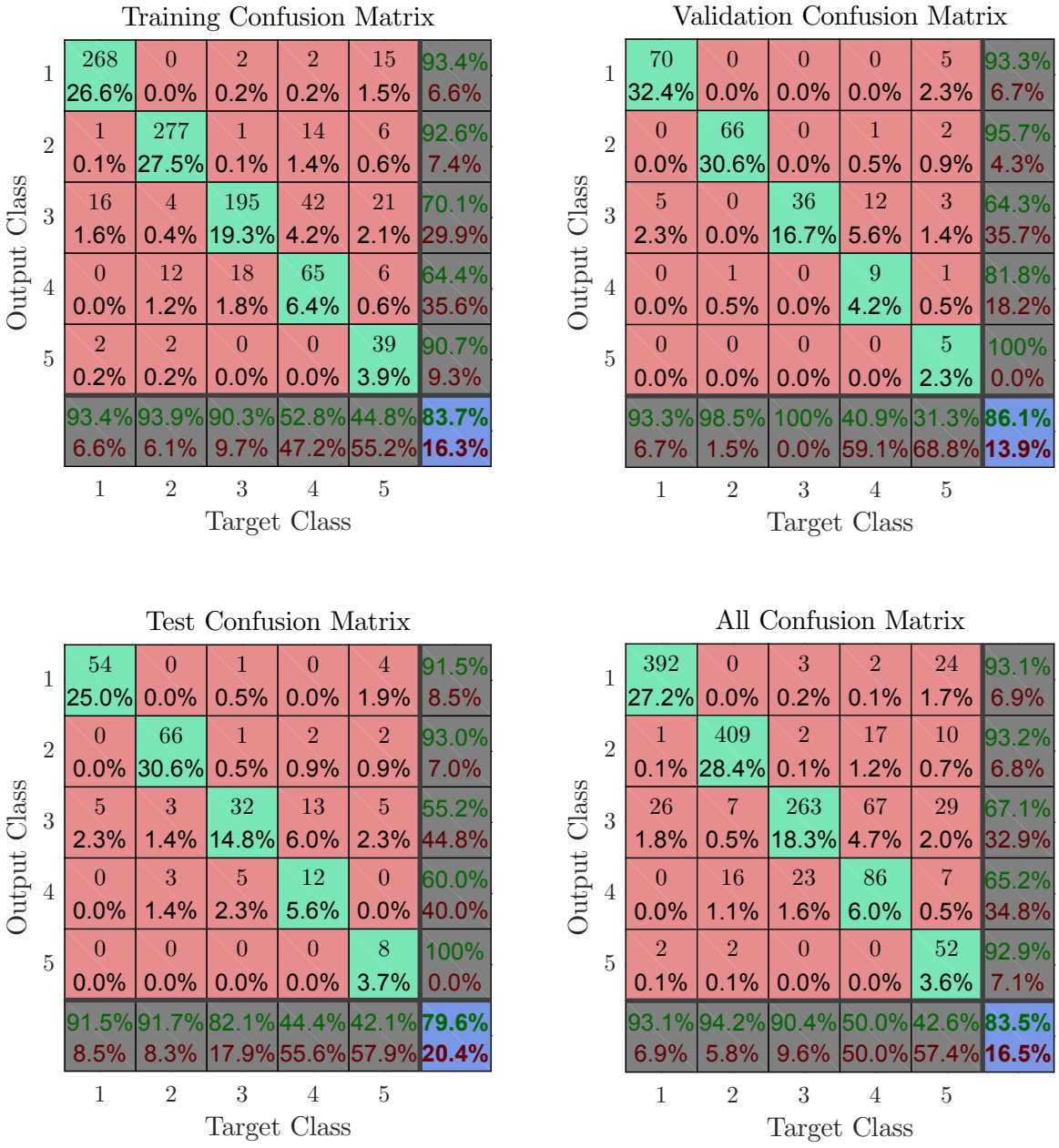


Figure 42: Confusion Matrix - 2-layer AE

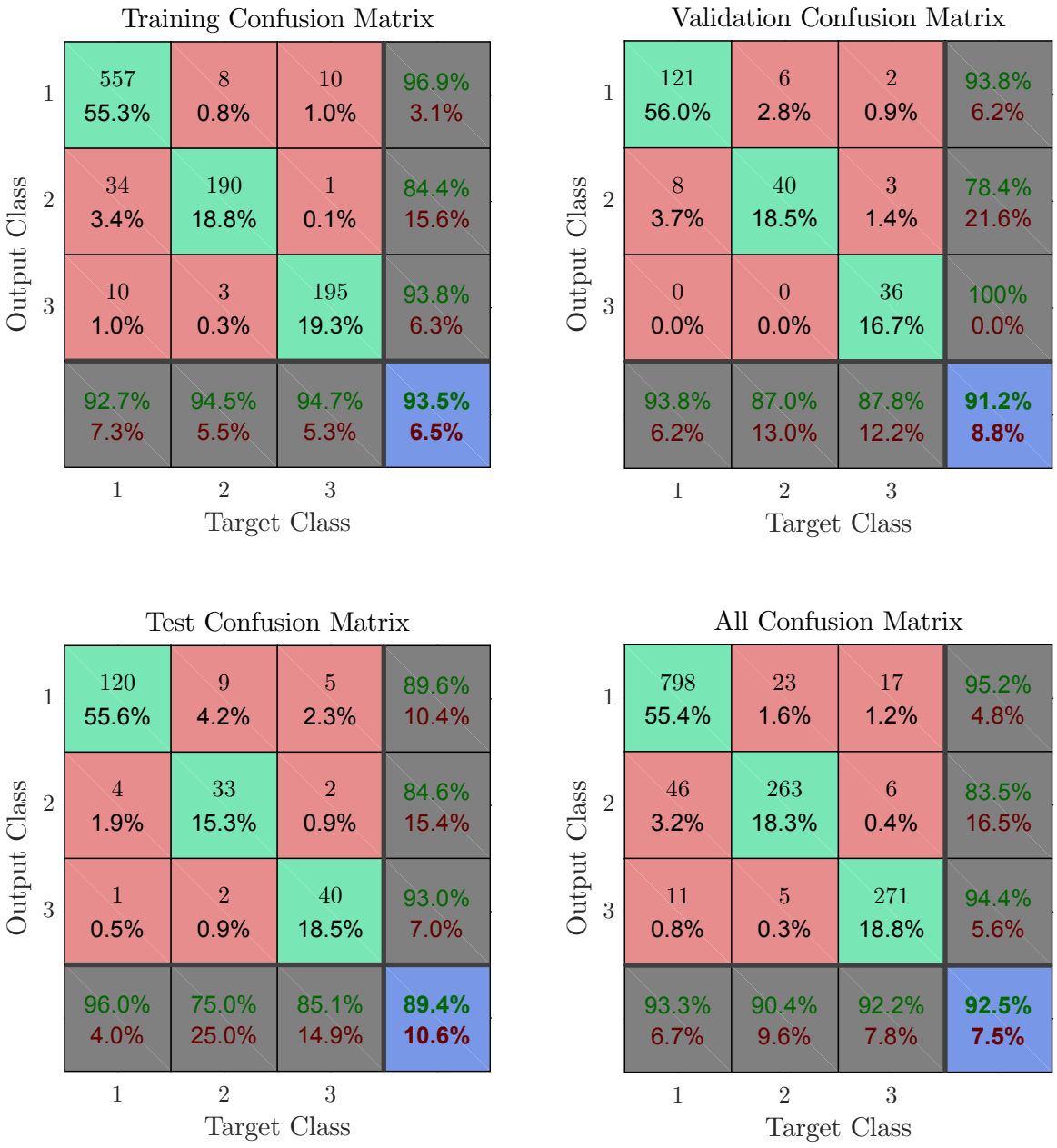


Figure 43: Confusion Matrix - 1-layer AE on RP

6 Conclusions and Future Work

This Thesis is evaluating the potential of deep learning techniques in the context of analysing manufacturing processes. In concrete, their contribution to the classification of time series segments originating from machine sensor data has been evaluated. As a prerequisite, research on relevant statistical concepts from time series analysis has been done leading to a preprocessing methodology based around stationarity, entropy and correlation measures. The results on the datasets show that cross-correlation as well as permutation entropy can be considered useful in analysing redundancies and information content of time series. Regarding stationarity, it can be concluded that the widely used definition of second-order stationarity should be questioned in the context of sensor data. Throughout analysis, none of the signals was tested as stationary when a certain length was exceed, meaning that standard testing procedures are not helpful as a finite sensor signal is very likely to have at least one segment being nonstationary in relation to the rest.
For classification, a selected time series was segmented using a sliding window approach and a small amount of segments was labelled using a clustering algorithm. This was done for short and simple and more complex segments. In the course of this, the concept of recurrence plots was tested as an alternative to feature vectors. Although advantageous for some classes, fundamental flaws were discovered, as the plots are not sensitive enough for signal direction and failed being distinguishable for such examples. As this is a very important feature in data from technical systems, their use is limited to specific cases.
For the actual learning task, autoencoder networks were tested for their ability to extract features from time series. The learnt features were compared to ones manually computed, whereby a random selection as well as a selection geared towards physical processes is used. The latter contains features such as differentials and outperformed the random selection. It is concluded that this is caused by the fact that differentials are more discriminative towards the physical nature of the process than the majority of classic time series features. The features extracted by an autoencoder were superior to both, showing the usefulness of unsupervised feature learning in sensor data classification. Depending on the dataset, an accuracy of up to 98 percent was reached. A performance effect when extracting features using two stacked autoencoders could not be shown. This is attributed to the fact that the segment structure is not complex enough for a multilayer representation to be beneficial. The actual classification was performed using a classic feed-forward neural network with two hidden layers. In addition, it can be shown that the classification result contains useful information on the machine operation process, such as frequency of certain operations or time spent on certain tasks.

Regarding future work, the evaluation of recurrent neural networks, especially long-short term memory networks might be a valuable contribution, as this type is able to capture temporal relations which are ignored in a segment based approach. Furthermore, a combined use of machine learning and analytical methods for sensor data, such as linear differential operators, can be considered. Such methods can help a learning algorithm to capture a higher level representation of the underlying physical process, such as a differential equation. This should be defined as a long term goal in manufacturing analytics.

References

[1] I. Arel, D. C. Rose, and T. P. Karnowski. Research frontier: Deep machine learning–a new frontier in artificial intelligence research. *Comp. Intell. Mag.*, 5(4):13–18, Nov. 2010.

[2] M. Arltova and D. Fedorova. Selection of unit root test on the basis of length of the time series and value of ar(1) parameter. *STATISTIKA*, 96(3):47–64, 2016.

[3] D. Bacciu. Unsupervised feature selection for sensor time-series in pervasive computing applications. *Neural Computing and Applications*, 27(5):1077–1091, 2016.

[4] C. Bandt and B. Pompe. Permutation entropy: A natural complexity measure for time series. *Phys. Rev. Lett.*, 88:174102, Apr 2002.

[5] P. Barnaghi, F. Ganz, C. Henson, and A. Sheth. Computing perception from sensor data. oct 2012.

[6] E. Battenberg and D. Wessel. Analyzing drum patterns using conditional deep belief networks. *ISMIR 2012*, 2012.

[7] Y. Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, Jan. 2009.

[8] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, Mar. 1994.

[9] G. Bontempi and Y. ael Le Borgne. An adaptive modular approach to the mining of sensor network data. In *in Proc. SIAM International Conference on Data Miningg*, 2005.

[10] D. A. S. Boris P. Bezruchko. *Extracting Knowledge From Time Series*. Springer-Verlag Berlin Heidelberg, 2010.

[11] G. E. P. Box and G. Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[12] M. Bozic, M. Stojanovic, Z. Stajic, and N. Floranovic. Mutual information-based inputs selection for electric load time series forecasting. *Entropy*, 15(3):926–942, 2013.

[13] J. V. Bradley. *Distribution-free statistical tests*. Prentice-Hall, 1968.

[14] P. J. Brockwell and R. A. Davis. *Introduction to time series and forecasting*. Springer-Verlag Inc, Berlin; New York, 2002.

[15] P. S. Cowpertwait. *Introductory Time Series With R*. Springer, 2009.

[16] S. F. Crone, J. Guajardo, and R. Weber. The impact of preprocessing on support vector regression and neural networks in time series prediction, 2006.

[17] Z. Cui, W. Chen, and Y. Chen. Multi-scale convolutional neural networks for time series classification. *CoRR*, abs/1603.06995, 2016.

[18] C. A.-M. Davila, J.C. and M. Zaremba. Wearable sensor data classification for human activity recognition based on an iterative learning framework. *Sensors 2017*, 17:1287, 2017.

[19] T. R. DeMark. *The New Science of Technical Analysis*. John Wiley & Sons, 1994.

[20] D. A. Dickey and W. A. Fuller. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association*, 74(366a):427–431, 1979.

[21] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660, Mar. 2010.

[22] L. Fang, H. Zhao, P. Wang, M. Yu, J. Yan, W. Cheng, and P. Chen. Feature selection method based on mutual information and class separability for dimension reduction in multidimensional time series for clinical data. *Biomedical Signal Processing and Control*, 21(Complete):82–89, 2015.

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

[24] R. M. Gray and L. D. Davisson. *An Introduction to Statistical Signal Processing*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[25] J. Günther, H. Shen, and K. Diepold. Neural networks for fast sensor data processing in laser welding. In *Jahreskolloquium - Bildverarbeitung in der Automation*, Lemgo, Germany, Nov 2014.

[26] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[27] N. Hatami, Y. Gavet, and J. Debayle. Classification of time-series images using deep convolutional neural networks. *CoRR*, abs/1710.00886, 2017.

[28] L. Haugh. Checking the independence of two covariance-stationary time series: A univariate residual cross-correlation approach. *Journal of the American Statistical Association*, 71 (354):378–85, 1976.

[29] J. He, S. Yang, and C. Gan. Unsupervised fault diagnosis of a gear transmission chain using a deep belief network. *Sensors*, 17(7):1564, 2017.

[30] W. C. Heschl. An investigation of the power of the wald-wolfowitz, two sample, runs test. Master's thesis, Naval Postgraduate School, 1972.

[31] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[32] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[33] D. Ienco, R. Gaetano, C. Dupaquier, and P. Maurel. Land cover classification via multi-temporal spatial data by recurrent neural networks. *CoRR*, abs/1704.04055, 2017.

[34] T. Jayalakshmi and S. A. Statistical normalization and back propagation for classification. *International Journal of Computer Theory and Engineering*, 3, 02 2011.

[35] C. Jentsch and S. S. Rao. A test for second order stationarity of a multivariate time series. *Journal of Econometrics*, 185(1):124–161, mar 2015.

[36] Y. L. Jie Tao and D. Yang. Bearing fault diagnosis based on deep belief network and multisensor information fusion. *Shock and Vibration*, 2016, 2016.

[37] C. G. Jihao Yin and J. Sun. Verification of statistical properties for hyperspectral images: Heteroscedasticity and stationarity. *Digital Information Processing and Communications*, 1(1), 2013.

[38] L. Jin, S. Wang, and H. Wang. A new non-parametric stationarity test of time series in the time domain. *Journal of the Royal Statistical Society Series B*, 77(5):893–922, 2015.

[39] A. G. P. Julius S. BENDAT, Loren D. Enochson. Tests for randomness, stationarity, normality and comparison of spectra. Technical report, Defense Technical Information Center, 1965.

[40] K. Keller, A. M. Unakafov, and V. A. Unakafova. Ordinal patterns, entropy, and eeg. *Entropy*, 16(12):6212–6239, 2014.

[41] C. S. Kiyoung Yang. On the stationarity of multivariate time series for correlation-based data analysis. *Data Mining, Fifth IEEE International Conference on*, 2005.

[42] R. Köhler A., Ohrnberger M. and C. S. F. Unsupervised feature selection for pattern search in seismic time series. *J. Mach. Learn. Res., Workshop and Conference Proceedings: New challenges for feature selection in data mining and knowledge discovery*, 2008.

[43] A. Kraskov, H. Stögbauer, and P. Grassberger. Estimating mutual information. *Phys. Rev. E*, 69:066138, Jun 2004.

[44] D. Kwiatkowski, P. Phillips, and P. Schmidt. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? Cowles Foundation Discussion Papers 979, Cowles Foundation for Research in Economics, Yale University, 1991.

[45] M. Längkvist, L. Karlsson, and A. Loutfi. Sleep stage classification using unsupervised feature learning. *Adv. Artif. Neu. Sys.*, 2012:5:5–5:5, Jan. 2012.

[46] M. Längkvist, L. Karlsson, and A. Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, pages 11–24, 2014.

[47] M. Längkvist and A. Loutfi. Learning feature representations with a cost-relevant sparse autoencoder. *International Journal of Neural Systems*, 25(01):1450034, 2015. PMID: 25515941.

[48] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, may 2015.

[49] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[50] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag.

[51] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *ISCAS*, pages 253–256. IEEE, 2010.

[52] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA, 2009. ACM.

[53] H. Lee, Y. Largman, P. Pham, and A. Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Proceedings of the 22Nd International Conference on Neural Information Processing Systems*, NIPS'09, pages 1096–1104, USA, 2009. Curran Associates Inc.

[54] S. W. Li Liang. A fault tolerant model for multi-sensor measurement. *Chinese Journal of Aeronautics*, 2015.

[55] R. Liu, S. Xu, C. Fang, Y. wen Liu, Y. L. Murphey, and D. S. Kochhar. Statistical modeling and signal selection in multivariate time series pattern classification. In *ICPR*, 2012.

[56] S. C. Y. Lu. Machine learning approaches to knowledge synthesis and integration tasks for advanced engineering. *Comput. Ind.*, 15(1-2):105–120, Nov. 1990.

[57] N. Mehdiyev, J. Lahann, A. Emrich, D. Enke, P. Fettke, and P. Loos. Time series classification using deep learning for process planning: A case from the process industry. *Procedia Computer Science*, 114(Supplement C):242 – 249, 2017.

[58] G. R. S. Molinas Cabrera, Maria Marta;Kulia. On wind turbine failure detection from measurements of phase currents: a permutation entropy approach. Technical report, Norwegian University of Science and Technology, 2013.

[59] M. F. Moller. A scaled conjugate gradient algorithm for fast supervised learning. *NEURAL NETWORKS*, 6(4):525–533, 1993.

[60] Z. Mustaffa and Y. Yusof. A comparison of normalization techniques in predicting dengue outbreak. *International Conference on Business and Economics Research, vol. 1, IACSIT Press*, 2011.

[61] G. Nason. *Stationary and non-stationary time series*, pages 129 – 142. The Geological Society, 2006.

[62] E. S. Ogasawara, L. C. Martinez, D. de Oliveira, G. Zimbrao, G. L. Pappa, and M. Mattoso. Adaptive normalization: A novel data normalization approach for non-stationary time series. In *IJCNN*, pages 1–8. IEEE, 2010.

[63] P. O'Leary, M. Harker, and C. Gugg. Sensor-data analytics in cyber physical systems - from husserl to data mining. In *Proceedings of the 4th International Conference on Sensor Networks*, pages 176–181, 2015.

[64] P. O'Leary and R. Ritt. Time series and time sequence analytics for the condition monitoring of large-scale bulk material handling machinery. Technical report, Institute for Automation, University of Leoben, 2016.

[65] P. C. Phillips and P. Perron. Testing for a Unit Root in Time Series Regression. Cowles Foundation Discussion Papers 795R, Cowles Foundation for Research in Economics, Yale University, 1986.

[66] S. M. Pincus. Approximate entropy as a measure of system complexity. *Proc Natl Acad Sci U S A.*, 88(6):2297–301., 1991.

[67] S. V. Poucke. Normalization methods in time series of platelet function assays. *Medicine*, 95(28):e4188, 2016.

[68] M. Quintana-Suárez, D. Sánchez-RodrÃŋguez, I. Alonso-González, and J. Alonso-Hernández. A low cost wireless acoustic sensor for ambient assisted living systems. *Applied Sciences*, 7:877, 2017.

[69] e. a. R. Yan. Permutation entropy: A nonlinear statistical measure for status characterization of rotary machines. *Mech. Syst. Signal Process*, 2011.

[70] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 262–270, New York, NY, USA, 2012. ACM.

[71] J. S. Richman and J. R. Moorman. Physiological time-series analysis using approximate entropy and sample entropy. *American Journal of Physiology - Heart and Circulatory Physiology*, 278(6):H2039–H2049, 2000.

[72] M. Riedl, A. Müller, and N. Wessel. Practical considerations of permutation entropy. *The European Physical Journal Special Topics*, 222(2):249–262, Jun 2013.

[73] B. B. M. S. C. Nayak and H. S. Behera. Impact of data normalization on stock index forecasting. *International Journal of Computer Information Systems and Industrial Management Applications*, 2014.

[74] S. E. Said and D. A. Dickey. Testing for unit roots in autoregressive-moving average models of unknown order. *Biometrika*, 71:599–607, 1984.

[75] A. Savitzky and M. J. E. Golay. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Anal. Chem.*, 36(8):1627–1639, July 1964.

[76] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III*, ICANN'10, pages 92–101, Berlin, Heidelberg, 2010. Springer-Verlag.

[77] W. F. Schmidt, M. A. Kraaijveld, and R. P. W. Duin. Feedforward neural networks with random weights. In *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems*, pages 1–4, Aug 1992.

[78] C. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.

[79] W. Shi, P. Shang, J. Xia, and C.-H. Yeh. The coupling analysis between stock market indices based on permutation measures. *Physica A: Statistical Mechanics and its Applications*, 447(Complete):222–231, 2016.

[80] R. H. Shumway and D. S. Stoffer. *Time Series Analysis and Its Applications (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[81] H. S. B. Sibarama Panigrahi, Y. Karali. Normalize time series and forecast using evolutionary neural network. *International Journal of Engineering Research & Technology (IJERT)*, 2013.

[82] J. Sola and J. Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science*, 44(3):1464–1468, jun 1997.

[83] L. Tang, H. Lv, F. Yang, and L. Yu. Complexity testing techniques for time series data: A comprehensive literature review. *Chaos, Solitons & Fractals*, 81(Part A):117 – 135, 2015.

[84] V. T. Tran, F. Althobiani, and A. Ball. An approach to fault diagnosis of reciprocating compressor valves using teager-kaiser energy operator and deep belief networks. *Expert Syst. Appl.*, 41(9):4113–4122, July 2014.

[85] M. Vafaeipour, O. Rahbari, M. A. Rosen, F. Fazelpour, and P. Ansarirad. Application of sliding window technique for prediction of wind velocity time series. *International Journal of Energy and Environmental Engineering*, 5(2-3), may 2014.

[86] M. van Heeswijk, Y. Miche, T. Lindh-Knuutila, P. A. J. Hilbers, T. Honkela, E. Oja, and A. Lendasse. Adaptive ensemble models of extreme learning machines for time series prediction. pages 305–314, 2009.

[87] M. Verleysen and D. François. The curse of dimensionality in data mining and time series prediction. In J. Cabestany, A. Prieto, and F. S. Hernández, editors, *IWANN*, volume 3512 of *Lecture Notes in Computer Science*, pages 758–770. Springer, 2005.

[88] G. Wallach, L. Faivishevsky, and A. Armon. Change and anomaly detection framework for internet of things data streams. *ICML 2016 Anomaly Detection Workshop*, 2016.

[89] H. Wang, C. Wang, Y. Zhao, X. Lin, and C. Yu. Toward a practical approach for ergodicity analysis. *Nonlinear Processes in Geophysics Discussions*, 2:1425–1446, 2015.

[90] Z. Wang and T. Oates. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 01 2015.

[91] M. Wielgosz, A. Skoczeń, and M. Mertik. Using LSTM recurrent neural networks for detecting anomalous behavior of LHC superconducting magnets. *Nuclear Inst. and Methods in Physics Research, A*, 867:40–50, 2017.

[92] J. Wu. Introduction to convolutional neural networks. Technical report, LAMDA Group, National Key Lab for Novel Software Technology Nanjing University, China, 2017.

[93] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben. Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1):23–45, jan 2016.

[94] J. B. Yang, M. N. Nguyen, P. P. San, X. L. Li, and S. Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 3995–4001. AAAI Press, 2015.

[95] M. M. R. Yousefi, M. Mirmomeni, and C. Lucas. Input variables selection using mutual information for neuro fuzzy modeling with the application to time series forecasting. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2007, Celebrating 20 years of neural networks, Orlando, Florida, USA, August 12-17, 2007*, pages 1121–1126, 2007.