CHALMERS
UNIVERSITY OF TECHNOLOGY

Fault Classification using Shapelets and Deep Machine Learning

Using shapelets to automatically annotate a large time series dataset in an effort to train Deep Neural Networks in fault classification on real sensor data from trucks

Master's thesis in *Systems, Control and Mechatronics* and *Algorithms, Language and Logic*

Oscar Olesen & Patrick Wadström

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:NN

Fault Classification using Shapelets and Deep Machine Learning

Using shapelets to automatically annotate a large time series dataset in an effort to train Deep Neural Networks in fault classification on real sensor data

Oscar Olesen
Patrick Wadström

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

Fault Classification using Shapelets and Deep Machine Learning
Using shapelets to automatically annotate a large time series dataset in an effort to train Deep Neural Networks in fault classification on real sensor data
Oscar Olesen & Patrick Wadström

Supervisors: Mamadou Diaby, Volvo AB, Marcus Ahlstrand, Volvo AB, Jinxiang Song, Chalmers Department of Electrical Engineering
Examiner: Guiseppe Durisi, Chalmers Department of Electrical Engineering

Master's Thesis 2020:NN
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Fault Classification using Shapelets and Deep Machine Learning
Using shapelets to automatically annotate a large time series dataset in an effort to train Deep Neural Networks in fault classification on real sensor data
Oscar Olesen & Patrick Wadström
Department of Electrical Engineering
Chalmers University of Technology

Abstract

At Volvo, a significant amount of time is spent analyzing fault reports from drivers. These fault reports consist of recordings from multitudes of sensors measuring every aspect of the trucks. To alleviate the manual labour of analyzing every incoming fault report for actual faults in their products, Volvo has requested an investigation in the possibility of producing a tool to identify common root faults.

Deep Neural Network (DNN) has shown ability to identify and classify data quickly. DNNs could potentially be a useful tool alleviating the manual labour of expensive engineers. However, the current performance of DNN is highly correlated with the amount of annotated data available. To avoid requiring large amounts of initial labour, this thesis investigates the possibility of annotating large amounts of data using only a few annotations.

The results show that it is possible to automatically annotate a multivariate time series dataset to train a DNN and the best network achieved an F1 score of 94.5%. This shows that this is a feasible solution and that it can be used to reduce the time engineers spend analyzing fault reports.

Keywords: shapelet, semi-supervised, neural networks, trucks, time series, fault classification

Acknowledgements

We would like to thank our supervisors at Volvo, Mamadou Diaby and Marcus Ahlstrand, for the support and guidance throughout the master thesis, without them this would not have been possible. We would also thank our boss Robin Karlsson and Volvo at large for allowing us this opportunity and for providing us with all the resources needed to realize this master thesis. Last but not least, we would like to thank our supervisor at Chalmers, Jinxiang Song for the academic guidance he provided.

Oscar Olesen & Patrick Wadström, Gothenburg, May 2020

Contents

List of Figures xi

List of Tables xiii

Glossary xv

1 Introduction 1
- 1.1 Goal . . . 1
- 1.2 Literature study . . . 2
- 1.3 Scope and limitations . . . 3
- 1.4 Problem definition . . . 4
- 1.5 Sustainability and ethical aspects . . . 4
- 1.6 Report outline . . . 5

2 Theory 7
- 2.1 Definitions . . . 7
  - 2.1.1 Basic definitions . . . 7
- 2.2 Shapelets . . . 8
  - 2.2.1 Supervised Shapelet Extraction Algorithm . . . 12
  - 2.2.2 Unsupervised Shapelet Extraction Algorithm . . . 13
- 2.3 Clustering . . . 16
  - 2.3.1 density-based Clustering . . . 18
  - 2.3.2 Clustering Assessment . . . 19
- 2.4 Deep Neural Networks . . . 21
  - 2.4.1 Convolutional Neural Networks . . . 22
  - 2.4.2 Residual Neural Networks . . . 22
  - 2.4.3 Classification performance measures . . . 23
- 2.5 CAN-bus Protocol . . . 24

3 Methods 25
- 3.1 Research environment . . . 25
- 3.2 The Dataset . . . 25
- 3.3 Pre-processing the fault reports . . . 26
  - 3.3.1 Automatic conversion of CAN-bus recordings . . . 26
  - 3.3.2 Signal reconstruction from CAN-bus recordings . . . 27
  - 3.3.3 Pre-processing system architecture . . . 28

3.4 Annotating the Dataset . . . 29
3.4.1 Extending Unsupervised Shapelets . . . 29
3.4.1.1 Multivariate Distance Measure . . . 29
3.4.1.2 Shapelet Scoring . . . 31
3.4.2 Shapelet Derivation Tree . . . 34
3.5 Clustering . . . 35
3.5.1 Clustering Evaluation . . . 36
3.6 Training the Neural Networks . . . 37
3.6.1 Training- and Validation-set split . . . 38
3.6.2 Convolutional Neural Network Architecture . . . 38
3.6.3 Residual Network Architecture . . . 39
3.7 GPU Acceleration . . . 39
3.8 Verification . . . 40
3.8.1 Toy Dataset . . . 40
3.8.2 External Datasets . . . 40
3.9 Benchmark . . . 41

4 Results & Discussion **43**
4.1 Benchmark results . . . 43
4.2 Shapelet extraction . . . 46
4.3 Clustering results . . . 48
4.4 Deep Neural Networks . . . 50
4.4.1 Final Network . . . 53
4.5 Verification . . . 55
4.5.1 Toy example . . . 55
4.5.2 External Datasets . . . 57
4.5.2.1 Epileptic Seizure Recognition . . . 57
4.5.2.2 GunPoint . . . 58

5 Conclusion **59**

Bibliography **61**

List of Figures

2.1 Visualization of three subsequences extracted from the sliding window process. 8
2.2 Comparison between a long and a short subsequence distance 9
2.3 Illustration of an orderline. The red circles represent time series from one class and the blue squares represent time series from another class. Low values on this orderline indicate a low subsequence distance. 10
2.4 Information gains of three different split strategies. Each vertical line is a split strategy, each circle is an object of class A and each square an object of class B. 11
2.5 A clustering feedback loop where the clustering evaluation is used to affect how every other step is done. 18
2.6 Visualisation of the density-based clustering process 19
2.7 Example of the convolution done by a filter in a Convolutional Neural Network (CNN) 22
2.8 Basic residual network block 23

3.1 Example of a signal reconstruction from a CAN bus recording. . . . 27
3.2 Pre-processing system architecture diagram 29
3.3 Visualization of the signal filter. The mean of each cluster is marked with an arrow of the same color as the cluster. With signals 0 and 1, the means are spread out across the orderline, thus being included in the filter. For signals 2 and 3, the yellow and purple means are close to each other, thus the signals gets filtered out 31
3.4 Visualization of an overlap between two clusters. 33
3.5 Illustration showing that considering every branch is necessary to find all features of a dataset. 35
3.6 Comparison of K-means clustering versus Ordering Points To Identify the Clustering Structure (OPTICS) clustering. The first row is clustering by K-means and the second row is clustering done by OPTICS 37
3.7 Network structures used in the thesis 38
3.8 Example of the toy dataset classes and signals. Every class has some induced shapes that would not be possible to achieve with a uniform noise ranging between $[-1, 1]$. 41

4.1 This is the orderline created by the baseline algorithms with the ground truth classes denoted by the different colors and black denoting a time series with unknown truth. 44
4.2 This is the same orderline as in figure 4.1 but here the colors denote the clustering labels with black denoting time series regarded as noise. 44
4.3 This is the shapelet space created by the unfiltered shapelet tree algorithm with the colors denoting ground truth classes. Again, black denotes time series with no known truth. 45
4.4 This figure shows the shapelet space created by the unfiltered shapelet tree algorithm with the colors denoting the clustering labels. Black denotes time series regarded as noise. 46
4.5 The first shapelet extracted by the shapelet tree algorithm. The top row shows the signals used in the shapelet, the second row shows the orderline created by each shapelet and the third row is the orderline of the shapelet. Signals with a red box indicates signals that has been filtered out by the signal filter and is not used to create the shapelet orderline. 47
4.6 The second shapelet extracted by the shapelet tree algorithm, this is the left side of the split point in Figure 4.5. 47
4.7 The second shapelet extracted by the shapelet tree algorithm, this is the right side of the split point in Figure 4.5. 48
4.8 This figure shows two views of the shapelet space created by the shapelets. The red, green and blue points denote the time series with predefined labels and the black points denote time series with unknown labels. 49
4.9 This figure shows the same two views of the same shapelet space as figure 4.8, but here the different colors denote the clustering labels assigned by OPTICS and black denotes points OPTICS regards as noise. The red and green colors labels seem to match the ground truth well, while the blue ground truth class has been split into three new clusters: blue, cyan and orange. 50
4.10 Example of the noisy nature of the training data. This plot is an extreme case and most of the time the training is more stable. 51
4.11 This figure shows that average training data over 50 runs when training the CNN with the Volvo dataset 52
4.12 This figure shows that average training data over 50 runs when training the Residual Network (ResNet) with the Volvo dataset 53
4.13 The best CNN network achieved when training. 54
4.14 The best ResNet network achieved when training. 55
4.15 This figure shows an example time series from each class in the toy dataset. 56
4.16 The first shapelet extracted when using the toy dataset 56
4.17 This shapelet is the second one extracted for the toy dataset and is the right branch of the shapelet tree. 57

List of Tables

3.1 Testing setups 41

4.1 Comparison of four different shapelet algorithms showing both clustering entropies as well as silhouette score. Setups 1 and 2 are the unsupervised versions and setups 3 and 4 are the semi-supervised shapelet tree versions. 43
4.2 Clustering scores when running the shapelet algorithms and OPTICS with the Volvo dataset 49
4.3 Peak score of all three performance metrics for both the CNN and ResNet 51
4.4 Peak Macro F1 scores for the CNN and ResNet along with the corresponding precision and recall. 53
4.5 Clustering scores when running the shapelet algorithms and OPTICS with the toy dataset 56
4.6 Clustering scores when running the shapelet algorithms and OPTICS with the ESR dataset with $minPts = 12$ 58
4.7 Clustering scores when running the shapelet algorithms and OPTICS with the Gun Point dataset with $minPts = 19$ 58

Glossary

AI Artificial Intelligence. 5

API Application Programming Interface. 26, 27

BLF Binary Logging Format. 26

CAN Controller Area Network. 24, 25, 27

CNN Convolutional Neural Network. xi–xiii, 3, 4, 21, 22, 38, 50–54

CSV Comma Separated Values. 26–28

DBSCAN Density-Based Spatial Clustering of Applications with Noise. 17, 19

DNN Deep Neural Network. v, 2–4, 7, 22, 37, 43, 54, 59, 60

ECU Electronic Control Unit. 1

ESR Epileptic Seizure Recognition. 40, 41, 57

GAP Global Average Pooling. 21

OPTICS Ordering Points To Identify the Clustering Structure. xi, xiii, 17, 19, 35–37, 44, 48, 49, 56, 58

ReLU Rectified Linear Unit. 21

ResNet Residual Network. xii, xiii, 3, 4, 21–23, 38, 39, 50, 51, 53–55

1

Introduction

Ever since the introduction of the Electronic Control Unit (ECU) in 1976 [1], vehicles of all kinds have slowly migrated from mechanical systems to electronic systems. In the modern truck, there exists a multitude of sensors and several kinds of ECUs. This has shifted truck development from a solely mechanical task to include a large portion of software development.

The software engineers developing control algorithms and other necessary software rely on fault reports as a critical method of feedback. Fault reports are created and transmitted when test drivers, who drive trucks with beta software during their normal workdays, detect something they consider a fault and push the report button. The reports contain a recording of signals from several ECUs, some may also contain audio and video, for a set amount of time before and after the push. However, due to being initiated by human drivers, reports may sometimes be for non-faulty scenarios or badly detailed, since the driver usually lack the expertise to provide useful descriptions, making the reports time consuming for engineers to analyze.

In an effort to shift the time spent from analyzing fault reports to innovating the future of trucks, this thesis aims to develop a classification tool aiding the engineers by providing them with an initial estimation of what faults a report might contain.

1.1 Goal

The aim of the thesis is to produce a tool aimed at creating an initial estimation of potential faults present in fault reports. The broad and very general goal of simply creating a tool has to be defined and scoped. To clearly motivate the final goal of this thesis, the backstory leading to the final problem definition is introduced.

At Volvo, there exists a large number of fault reports recorded from previous development efforts and reports for active software. These reports consist of recordings of signals from the system bus and in some cases audio and video. The audio and video sometimes provided often contain descriptions from the test drivers, however, much as Huo et al. [2] identified, reports are most useful when the reporter is an expert and may sometimes be detrimental otherwise. This was affirmed by the engineers who assisted us with this thesis. As such, the engineers analyzing these reports rely

heavily on the system bus recordings.

The faults in the system bus recordings are seldom identified by using a single signal but often multiple signals, and the faults identified may be represented in ways such as value change, slope and odd values, often in combinations. The shared nature of the signals is the dimension of time, which makes the signal data time series data.

As a part of the general goal is to produce a tool to increase efficiency, requiring a large effort in creating this tool may counteract the time saved using the tool. As such, creation of the tool must be done using a minimal amount of human effort.

In an effort to create a solution using minimal human effort, capable of analyzing a broad spectrum of signal behaviours, the scope was limited to searching for a machine learning solution. This choice was motivated by the low human effort required to train machine learning solutions with a proper dataset.

However, the current collection of reports at Volvo lacks any annotations. To be able to utilize existing supervised machine learning methods, annotations are needed. As such, machine learning solutions must either be unsupervised or the annotation process needs to be automated.

With these motivations clarified, our final goal became to create a solution for annotating a time series dataset with minimal human effort along with a fault classifier for time series data based on machine learning.

1.2 Literature study

There exists significant research into classification solutions using machine learning. In a review by Fawaz et al. [3], many solutions specific to time series data are compared and analyzed, especially showing the potential in utilizing existing DNNs to classify time series data. Based on this review, we concluded that the networks reviewed could easily be applied to the current context with the exception of needing supervised training.

Overall [3] is a good article, but we have one criticism. When presenting their "Fully convolutional neural network", they are vague about the fact that it has a fully connected layer at the end. Therefore we will refer to their "Fully convolutional neural network" as Convolutional Neural Network (CNN) in this thesis since we do not regard it as a *fully* convolutional neural network.

The potential of DNN is extended by its ability to perform inference on low power devices [4, 5, 6], showing the ability to transfer a trained network onto test trucks or create portable tools for engineers.

In a report by Längkvist et al. [7], the conclusion is drawn that better results for unsupervised feature learning for time series are produced by the modified deep learning models. However, they also conclude that there are still many challenges

in applying these solutions to time series data.

In a review by Bagnall [8], many time series data classification algorithms are compared. As he concludes, the performance of the algorithms is often related to the specific type of issue. This research however stands as a basis for the decision to aim at *shapelet* based solutions as we believe it is general enough to cover our signal variants.

In the area of *shapelets*, Keogh has been prominent as he introduced the concept of *shapelets* [9]. The method can be used as a way to group time series data by similarity. In another report, the *shapelet* concept is expanded to an unsupervised solution [10], utilizing clustering to provide good similarity identifications without any previous annotation. It was concluded that this method could potentially be applied as an annotation generator for the dataset provided, allowing the utilization of the high performing DNNs. However, the unsupervised method selects any useful similarity to separate data. As the data at Volvo contains many similarities that are present among several fault classes, the right similarities have to be chosen to successfully divide data into the defined fault classes.

Much of the existing research on *shapelets* has focused on the use of univariate time series. There exists research on multivariate solutions [11], however, these solutions also require variates to be correlated and of equal importance. Based on this we concluded that an effort in creating a new multivariate solution was needed before *shapelets* present a viable solution.

1.3 Scope and limitations

Based on the literature study, we concluded that the unsupervised methods found are either non-trivial to apply given our dataset and may yield unexpected results or will learn features freely and with high probability annotate the dataset in unwanted groupings.

To control the possible annotations (labels) but allow the machine to perform the annotations, it was concluded that a hybrid solution was required. An adaption of the unsupervised *shapelets* to control the selection of similarities could potentially yield good annotations. However, these algorithms are computationally heavy and the goal is to produce a usable tool, which requires it to finish within a reasonable time. In addition to the final runtime performance achievable by DNN and its potential to be transferred into embedded contexts such as portable tools or inside the trucks, it was decided that DNN would provide the final classifier provided as the tool created. The heavier *shapelet* based algorithm would provide annotations and allow for supervised training of DNNs.

In the review article by Fawaz et al. [3], different time series datasets and DNNs are tested. They show that their CNN (referred to as FCN in the review article) and Residual Network (ResNet) outperform the other networks. ResNets was introduced in 2015 by [12] and uses shortcuts between layers in the neural networks, to allow

for deeper networks than previously possible. This is explained further in Section 2.4.2. Based on this review, the CNN and ResNet were chosen to be evaluated as final classifiers. Two networks were chosen to avoid any mishaps with either network not performing as expected.

To allow for relevant findings, the research effort was dedicated to creating the hybrid approach using *shapelets*. No comparison or related implementations with other solutions will be performed or analyzed.

To conclude, the effort is focused on performing semi-supervised annotations of a time series dataset consisting of one dimensional signals with mixed correlations, and proving the concept works to create a tool based on DNN which generates estimations of faults in new reports.

1.4 Problem definition

To build a tool for classifying fault reports, two main problems were formulated:

1. Annotate a dataset of time series data using only a small effort based on the concept of *shapelets*.

2. Verify that the annotations created can train a DNN to perform well enough to be considered an aid to the engineers.

Annotating the dataset consists of the following two subproblems; expanding the *shapelet* concept to handle partially correlating multivariate data and how to control the similarities used to extract *shapelets* as to create relevant annotations.

The research questions this thesis aims to answer are therefore the following:

- Can clustering of unlabeled multidimensional time series be done to facilitate the training of deep neural networks using supervised training methods?

- Can a useful[1] level of precision and recall be achieved for detecting and classifying faults in a new time series using the trained deep neural networks?

1.5 Sustainability and ethical aspects

Trucks and buses are responsible for about a quarter of all CO_2 emissions from road transports in the EU, which is why EU forces vehicle manufacturers to decrease their CO_2 emissions. By 2025, emissions have to be reduced by 15%, and by 2030, emissions have to be reduced by 30%, compared to the EU average in the reference period 1 July 2019 – 30 June 2020 [13].

[1]Useful is defined as having more than 2/3 in precision and more than 2/5 recall. These are arbitrary figures based on the intuition that any initial help is important, but bad predictions may cost time and counteract the intention of the tool.

To meet these requirements, Volvo is working hard with sustainability and have several projects to reduce both the CO_2 and noise emissions. For example, Volvo has introduced fully electric trucks which both reduce local CO_2 emissions as well as noise emissions. Volvo has partnered with Samsung SDI to develop new battery packs as well as with Daimler to develop hydrogen fuel cells [14, 15]. To realize these products and to reduce emissions, there is a lot of work needed by skilled engineers. This is where we can contribute. By allowing engineers to spend less time troubleshooting new fault-reports, they can instead work on developing new technologies to reduce emissions.

The ethical part of this thesis work is not as clear cut. If the research we introduce is only used to analyze fault-reports to develop more environmental vehicles, we see no ethical problems. However, this research could potentially be used for annotating data containing personal information, which might become highly unethical. Since this is a public paper, we cannot stop companies or individuals using technology in unethical ways. That task is up to the legislators. Work is being done to make sure Artificial Intelligence (AI) is not being used unethically, for example, the OECD has ethics guidelines for using AI [16]. We do not perceive this research to have any direct impact on the ability to do unethical things.

1.6 Report outline

The report is outlined as follows, beginning with theory in Chapter 2, existing research on which this thesis is based is introduced. The following chapter, Chapter 3, describes the methods used and the algorithms produced in this thesis. Then follows Chapter 4, were results are combined with an in depth analysis and discussion. The thesis is concluded in Chapter 5.

2

Theory

2.1 Definitions

The theory chapter begins by introducing a set of basic definitions used throughout the paper. It is followed by the definitions used to provide the final definition of *shapelet*. The introduction of *shapelets* is followed by brief explanations of the supervised and unsupervised shapelet extraction algorithms. Thereafter follows an introduction and relevant explanation on clustering, continued by an introduction of DNNs. Lastly, the CAN-bus protocol used in the system bus of the trucks is briefly explained.

2.1.1 Basic definitions

This thesis is focused on time series data. To clarify, time series are defined as follows.

Definition 1. [*Time Series*] A time series $T = T_1, T_2, ..., T_N$ is an ordered set of equally spaced real values. The total number of real values is equal to the length of the time series.

To allow referencing to parts of time series, the definition of subsequence is provided as Definition 2.

Definition 2. [*Subsequence*] A subsequence S_i^l is a set of l equally spaced real values from a time series, T, that starts at position i.

Another concept related to time series is *Sliding Window*. It is a method for selecting every possible subsequence in a time series, visualized in Figure 2.1. The formal definition is given in Definition 3.

Definition 3. [*Sliding Window*] Given a time series T of length m, and a subsequence of length l, all possible subsequences can be extracted by sliding a window of size l across T and considering each subsequence S_i^l of T, where i is the starting position.

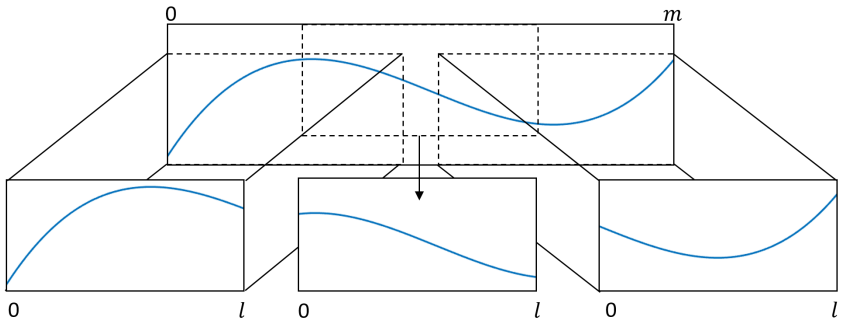


Figure 2.1: Visualization of three subsequences extracted from the sliding window process.

2.2 Shapelets

Shapelets were introduced by Keogh and Ye as a classification tool utilizing features specific to time series [9]. In short, shapelets are phase independent subsequences extracted from time series which are representative of its class. Informally, the intention with shapelets is to find distinctive shapes in time series which divide a relevant subset from the dataset. The algorithm provided by Keogh and Ye uses Euclidean distance as a similarity measurement and rank the shapelets by their ability to separate time series by their class. To create a phase independent solution, normalization is applied before comparing subsequences.

Using shapelets to classify time series allows for finding distinctive features amongst time series which could otherwise differ significantly since the distinctive features can be present at any position in the series. Below follows a short and compact explanation and some key definitions related to shapelets.

Before defining a shapelet, some prior definitions need to be established. We start with the definition of *subsequence distance* in Definition 4. This allows for measuring the difference between a subsequence and a potentially much longer time series.

Definition 4. [*Subsequence distance*] A distance function that takes a subsequence S and a time series T as input and returns a non-negative value d, which is the distance from T to S.

$$SubsequenceDist(T, S) = min(Dist(S, S')), \text{ for } S' \in \mathbf{S}_T^{|S|} \tag{2.1}$$

where $Dist$ refers to any vector distance measure and $\mathbf{S}_T^{|S|}$ represents the set of subsequences of length $|S|$ extracted from a time series T.

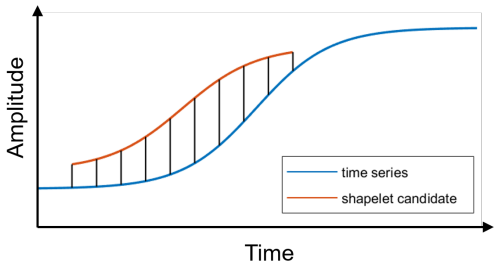
In the original paper [9], the $Dist$ measure is Euclidean norm, also known as $L^2 - norm$. The formal definition of Euclidean norm is given by Definition 5.

Definition 5. [*Euclidean norm*] The distance between two vectors, p and q, of

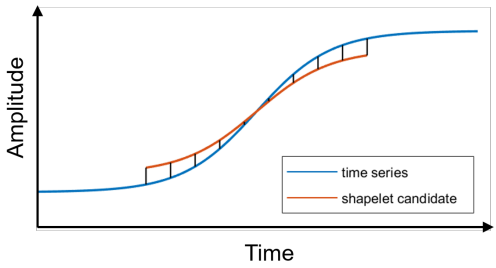
length n, is computed by

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{2.2}$$

Intuitively the *SubsequenceDist* is the smallest distance between the shapelet candidate S and any subsequence S' in the time series T. The difference between a long and short subsequence distance is shown in Figure 2.2, where 2.2a shows a long distance and 2.2b shows a short distance.



(a) An example of a long subsequence distance, giving a bad fit for a shapelet at the given location.



(b) An example of a short subsequence distance, giving a god fit for a shapelet at the given location.

Figure 2.2: Comparison between a long and a short subsequence distance

To give the comparison phase independence, normalization is done on both the shapelet candidate and the sequence compared. This is done by *Z-normalization*, also known as standard score. It is the same method used for normalization in the original paper [9]. *Z-normalization* normalizes sequences to zero mean and unit energy. This is done before comparison. The definition of *Z-Normalization* is given in Definition 6.

Definition 6. [*Z-Normalization*] A set of values S is normalized by,

$$x' = \frac{x - \mu}{\sigma}, \text{for every } x \in S \tag{2.3}$$

where, μ is the mean of the set and σ is the standard deviation of the set.

To allow for visualization of subsequence distances, the definition and description of an *orderline* is provided in Definition 7. This is a concept originating from the authors of shapelets [9].

Definition 7. [*Orderline*] The orderline is a vector of subsequence distances over a set of time series.

The orderline visually describes how similar a time series is to the shapelet, a low value indicates a low subsequence distance, meaning that the shapelet and subsequence best matching in the time series are similar. A high value indicates that the subsequence is dissimilar from the shapelet. This can be seen in Figure 2.3

Figure 2.3: Illustration of an orderline. The red circles represent time series from one class and the blue squares represent time series from another class. Low values on this orderline indicate a low subsequence distance.

as subsequences represented by dots and squares are placed on the orderline with different distances from the left. As such, the red dots to the left are more similar to the shapelet than the squares to the right. Since the best subsequence is derived by *SubsequenceDist*, this also represents the similarity considered for the time series.

The extraction of *shapelets* is based on ranking candidates by a measurement. This is defined using two definitions. The first definition is denoted as entropy by the original authors, the convention is maintained in this thesis. The definition of entropy is given in Definition 8. The entropy is used in the second definition of *Information Gain*, which is the measurement used to rank candidates. *Information Gain* is given in Definition 9.

The extraction of *shapelets* is based on ranking candidates by a measurement. This is defined in two parts. First, the notion of entropy in the context of shapelets is defined in Definition 8, then information gain, which utilizes entropy, is defined in Definition 9.

Definition 8. [*Entropy*] Consider a time series dataset D consisting of two classes, A and B. Given that the fraction of objects in class A is $p(A)$ and the fraction of objects in class B is $p(B)$, the entropy of D is

$$I(D) = -p(A)\log(p(A)) - p(B)\log(p(B)) \tag{2.4}$$

The entropy measure can informally be summarized as a measure that is small when the dataset only contains a single class and rises as objects from both classes are mixed.

Definition 9. [*Information Gain*] Given a certain split strategy sp which divides D into two subsets D_1 and D_2, the entropy before and after splitting is $I(D)$ and $\hat{I}(D)$. The information gain is given as:

$$Gain(sp) = I(D) - \hat{I}(D) \tag{2.5}$$

The *Information Gain* is a measurement for *entropy* gain obtained by dividing the dataset at a specific position on the orderline, referred to as *split strategy*. The concept is visualized in Figure 2.4. Note the high gain on the leftmost strategy as the right subset has proportionally more blue squares than red dots. As such the entropy is lowered on the right side (the remaining items in D), yielding a higher gain compared to the original set.

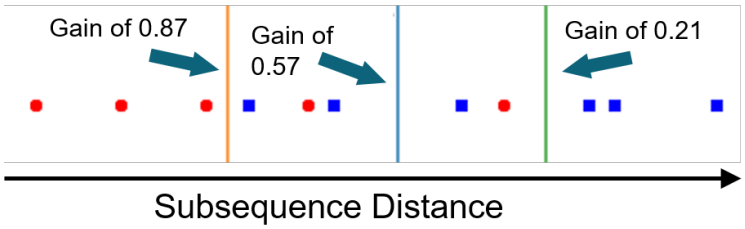


Figure 2.4: Information gains of three different split strategies. Each vertical line is a split strategy, each circle is an object of class A and each square an object of class B.

To allow finding the split strategy yielding the best subset divisions, the *Optimal Split Point* is defined as follows.

Definition 10. [*Optimal Split Point OSP*] Consider a time series dataset D consisting of two classes, A and B. For a shapelet candidate S, we choose some distance threshold d_{th} and split D into D_1 and D_2, such that for every time series object $T_{1,i}$ in D_1, $SubsequenceDist(T_{1,i}, S) < d_{th}$ and $T_{2,i}$ in D_2, $SubsequenceDist(T_{2,i}, S) \geq d_{th}$. An *Optimal Split Point* is a distance threshold for which

$$Gain(S, d_{OSP}(D, S)) \geq Gain(S, d'_{th})$$

for any other distance threshold d'_{th}.

With these definitions in place, the formal definition of a *shapelet* can be given.

Definition 11. [*Shapelet*] Given a time series dataset $\mathbf{D}$ which consists of two classes, A and B, $shapelet(\mathbf{D})$ is a subsequence such that, with its corresponding optimal split point,

$$Gain(shapelet(D), d_{OSP}(D, shapelet(D))) \geq Gain(S, d_{OSP}(D, S)) \tag{2.6}$$

for any other subsequence S.

Put simply, a shapelet is defined as the subsequence derived from the dataset which along with its associated *Optimal Split Strategy* provides the best separation of a class of data based on similarity.

The attentive reader realizes that a subset close to the left side of the orderline consisting of data points from the same class, and other points further to the right indicates that the shapelet itself represents a distinct feature for the leftmost subset.

To achieve invariance against offset and scale, the shapelets and subsequences are always Z-normalized before comparison. The reader should note that this means that we only compare curve shapes and not absolute values. It is the shape and not the location which affects the similarity measure.

2.2.1 Supervised Shapelet Extraction Algorithm

The shapelet algorithm described in the same paper introducing shapelets [9] is a brute-force algorithm for deriving shapelets based on ground truth and using them to classify a univariate multi-class dataset.

Algorithm 1 provides a pseudo code implementation of the original algorithm.

Algorithm 1 Supervised Shapelet Extraction

```
Require: MAXLEN: Max shapelet length
Require: MINLEN: Min shapelet length
Require: D: A dataset
Ensure: S: A shapelet
  candidates ← GenerateCandidates(D, MAXLEN, MINLEN)
  bsf_gain ← 0
  S ← Nothing
  for all C in candidates do
      distance_vector ← ∅
      for all T in D do
          distance ← SubsequenceDist(T, C)
          insert T into distance_vector by the key distance
      end for
      optimal_split_point ← OptimalSplitPoint(distance_vector)
      D_1 ← Every T in distance_vector where distance ≥ optimal_split_point
      gain ← I(D) − I(D_1)
      if gain > bsf_gain then
          bsf_gain ← gain
          S ← C
      end if
  end for
  return S
```

The algorithm starts by deriving every possible subsequence with lengths in the specified interval (given as parameters) by using a *sliding window* (definition 3) on the complete dataset. The algorithm continues by calculating the distance vector for every candidate, followed by finding the *Optimal Split Point* and finally the gain produced by using the shapelet. It then compares the candidate's gain against the best known gain and continually stores the best shapelet along with its gain.

The supervised algorithm creates a binary split between two classes. To allow for multi-class datasets, the algorithm is iteratively performed inside a decision tree [17] and considers one class to be a class in the dataset, and the second class every other class.

The multi-class version is shown as Algorithm 2. In short, it iterates the shapelet derivation while mapping one class as a class from the dataset, and the second class as all the other classes from the dataset combined. The algorithm keeps track of the

D_1 dataset of the best shapelet and assigns it for the next iteration. As such the dataset continually shrinks until it is empty with each shapelet being similar and extracting a single class (if possible).

Algorithm 2 Multi-Class Supervised Shapelet Extraction

```
Require: MAXLEN: Max shapelet length
Require: MINLEN: Min shapelet length
Require: D: A dataset
Ensure: S: A set of shapelets
  S ← ∅
  while D is not empty do
    candidates ← GenerateCandidates(D, MAXLEN, MINLEN)
    bsf_gain ← 0
    S' ← Nothing
    D' ← ∅
    for all C in candidates do
      distance_vector ← ∅
      for all T in D do
        distance ← SubsequenceDist(T, C)
        insert T into distance_vector by the key distance
      end for
      optimal_split_point ← OptimalSplitPoint(distance_vector)
      D_1 ← Every T in distance_vector where distance ≥ optimal_split_point
      gain ← I(D) − I(D_1)
      if gain > bsf_gain then
        bsf_gain ← gain
        S' ← C
        D' ← D_1
      end if
    end for
    D ← D'
    Append S' to S
  end while
  return S
```

Some important aspects to note! The algorithm is supervised; truth is continuously used to determine entropy and information gain. The algorithm is exhaustive; it tries all possibilities. It is univariate; it considers each time series as an object.

2.2.2 Unsupervised Shapelet Extraction Algorithm

The unsupervised shapelet algorithm is an extension of the supervised shapelet algorithm by Zakaria et al. [10]. The paper introduces a variant of a shapelet called an unsupervised-shapelet (shortened as u-shapelet). It does not require any ground truth in its derivation, which means that the entropy measure of the supervised algorithm is replaced with an unsupervised equivalent. In addition to modifying the

measure, the extraction algorithm has been modified to derive shapelet candidates from a single time series for each iteration, contrary to the supervised shapelet extraction algorithm. This makes it nonexhaustive, furthermore, it might not find the optimal solution.

Definition 12. [*U-shapelets*] An *unsupervised-shapelet* S is a subsequence of a time series T for which the *sdists* between S and the time series from a group D_A are much smaller than the *sdists* between S and the rest of the time series D_B in the dataset D.

$$sdist(S, D_A) \ll sdist(S, D_B) \tag{2.7}$$

The definition of an unsupervised shapelet refers to the measure *sdist*. This is the same measure referred to as *SubsequenceDist* in the supervised algorithm, provided by Definition 4. In the algorithms defined, we will refer to *sdist* as *SubsequenceDist* to maintain consistency with the supervised algorithm.

In the unsupervised shapelet algorithm, the entropy measure is replaced with a separation measure called *gap*. The separation measure can informally be described as measuring the gap between two sets of datapoints, called D_A and D_B. This separation measure is computed for each possible point of separation. The best point of separation is returned together with the score. The gap measure is defined in Definition 13.

Definition 13. [*Gap*] Given a dataset D, which is split into two subsets: D_A and D_B, where D_A is the left subset on the orderline and D_B is the right subset on the orderline, the gap is calculated by

$$gap = \mu_B - \sigma_B - (\mu_A + \sigma_A) \tag{2.8}$$

where μ_B and σ_B are the mean and standard deviation of D_B and μ_A, σ_A are the mean and standard deviation of D_A.

The maximum gap for a u-shapelet is derived similarly to the Optimal Split Point in the supervised algorithm. The derivation of the maximum gap is implemented in Algorithm 3. The algorithm checks the gap score of every division point between two distances in the distance vector, keeping track of the best and at what split point it occurred. There is an if statement to avoid pathological cases where few items are similar or dissimilar. This is limited by a constant k which is defined by the user.

The derivation of shapelets is shown in Algorithm 4. It works by checking all shapelet candidates from the first time series in the dataset, picking the one with the highest gap score. The dataset is then divided at the point given by $mean(dis_A) + std(dis_A)$. It is not divided by the split point yielding the highest gap score. The used split point, given by the previous equation, includes more items when the group separation is low since it considers the standard deviation of the group being

Algorithm 3 computeGap

Require: $\hat{s}$: Candidate u-shapelet; D: dataset
Ensure: maxGap: Maximum gap score; dt: split point

```
dis ← sorted(SubsequenceDist(ŝ, D))
for l ← interval 1 to |dis_s| − 1 do
    d ← (dis_s(l) + dis_s(l + 1))/2
    dis_A ← dis_s < d
    dis_B ← dis_s > d
    r ← |dis_A|/|dis_B|
    if 1/k < r < (1 − 1/k) then
        m_A ← mean(dis_A), m_B ← mean(dis_B)
        s_A ← std(dis_A), s_B ← std(dis_B)
        gap ← m_B − s_B − (m_A + s_A)
        if gap > maxGap then
            maxGap ← gap
            dt ← d
        end if
    end if
end for
return maxGap, dt
```

separated. Intuitively it can be seen as catching a cluster (everything inside the standard deviation starting from the mean).

Before the next iteration, the algorithm selects the next time series to act as a source for shapelet candidates as being the time series with the furthest distance from the current shapelet. The algorithm iterates until there is only a single time series being separated. This indicates the discriminatory power is depleted such that we can only identify a single time series with the shapelet derived.

Reference [10] not only introduces the *u-shapelet*, but the application for unsupervised classification using clustering techniques. To enable clustering based on shapelets, the algorithm returns the distance vector for each shapelet.

The unsupervised algorithm has a set of properties that differ from the supervised algorithm. Its performance can be hindered by the ordering of time series as the first time series of the dataset is used to derive the initial set of candidates. This issue was discovered and improved upon in a subsequent paper by the same authors [18]. The new separation measure (gap) only considers single dimensional cluster separation, meaning that if one shapelet is present in a subset of each class, the algorithm separates these subsets as a class and does not separate them further. This means that the unsupervised approach captures the most prominent similarities.

Algorithm 4 Unsupervised Shapelet Extraction

Require: MAXLEN: Max shapelet length
Require: MINLEN: Min shapelet length
Require: D: A dataset
Ensure: S: A set of shapelets

```
S ← ∅
ts ← D(1)
while True do
    cnt ← 0
    ŝ, gap, dt, dis ← ∅
    for sl ← MINLEN to MAXLEN do
        for i ← 1 to |ts| − sl + 1 do
            ŝ(cnt + 1) ← ts(i : i + sl − 1)
            [gap(cnt + 1), dt(cnt + 1)] ← computeGap(ŝ(cnt + 1), D)
        end for
    end for
    index1 ← max(gap)
    Append ŝ(index1) to S
    dis ← SubsequenceDist(ŝ(index1), D)
    dis_A ← dis < dt(index1)
    if |dis_A| == 1 then
        return S
    else
        index2 ← max(dis)
        ts ← D(index2)
        m ← mean(dis_A) + std(dis_A)
        D̂ ← every T in D where dis < m
        D ← D − D̂
    end if
end while
return S
```

2.3 Clustering

Clustering is an unsupervised classification task that utilizes large amounts of data to find patterns within the data in order to group them in a meaningful way [19]. This allows for annotation of data to be used in, for example, machine learning, which is the use of the derived annotations in this master thesis.

Before starting to describe what clustering is, there is a concept that needs to be introduced, *feature space*.

Definition 14. [*Feature space*] A feature space is an n-dimensional vector space where each sample is represented as a point. The dimensionality of this space is equal to the number of features used to represent the data.

In 1988, Jain and Dubes described a typical workflow when performing clustering, it includes the following steps [20]:

- Pattern representation, including feature extraction and selection.
- Definition of a pattern proximity measure.
- Grouping.
- Data abstraction (if needed).
- Assessment of the output (if needed).

Pattern representation refers to the number of available patterns, the number of classes and the number of features. *Feature extraction and selection* refers to the identification of available features in the data.

Pattern proximity refers to the distance measurement in the feature space. A common measurement here is the euclidean distance which is used in this thesis. But this can be replaced by any distance measure that fits the domain of the data.

Grouping is the central task in this clustering, where the features are used to group similar data points and annotate them. There are several examples of clustering algorithms such as K-means, Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and OPTICS. K-means partitions all samples into a predefined number of clusters, DBSCAN and OPTICS are density-based algorithms that partitions samples into clusters based on the density of points in an area. These algorithms will be explained further in Section 2.3.1.

Data abstraction refers to producing a description of the clusters, either as a way to summarize the data for some other machine step or to summarize the data to a human. This could for example be to show the features that are common within the cluster.

Assessment of the output is as it sounds, assessing how well the clustering performs. When having access to the ground truth labels, it is easy to compare the labels produced by the clustering algorithm. However, in this thesis, only a small subset of ground truth labels are available and the majority of data points will be unknown, complicating the task of assessing the output.

The workflow using these tasks, as suggested by Jain in [20], is shown as a flowchart in Figure 2.5. The assessment of the output is used as manual feedback to the other steps to improve the feature extraction and clustering itself [20].

In this thesis the *feature extraction* will be done via the use of shapelets.

The *pattern proximity* calculations will be done by comparing the time series to the shapelets, looking at the similarity between each time series and each shapelet. This produces a matrix that is used by a clustering algorithm to assign each time series

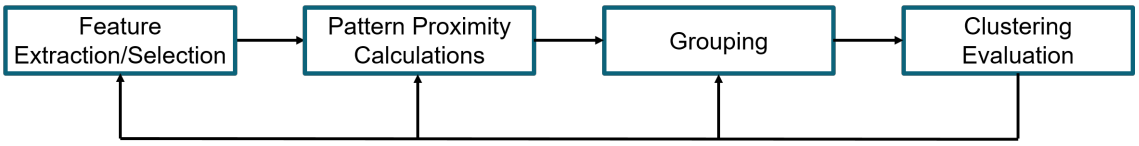


Figure 2.5: A clustering feedback loop where the clustering evaluation is used to affect how every other step is done.

a label.

The *grouping* itself will be done by using a density-based clustering algorithm, which is described in Section 2.3.1.

The *assessment of the output* will be performed with three separate assessments. There will be two clustering evaluation indexes combined with a human judge to determine how well the clustering performed. This is explained further in Section 2.3.2

2.3.1 density-based Clustering

density-based clustering group points together based on how tightly packed they are in the feature space, meaning that it can cluster points regardless of the shape of the cluster. Density based clustering can also create an arbitrary number of clusters as the goal is to group points in regions with similar density [21].

Another property of density-based clustering algorithms is their ability to classify samples in the feature space as noise, meaning that the algorithm does not consider certain samples as part of any cluster [21, 22].

To explain how the density-based algorithms work, a simplified description is provided below. The description intends to capture the main concepts of density based clustering relevant to this thesis.

These algorithms define clusters by classifying the points into three categories; *core points*, *border points* and *noise*. To place points in one of these three categories the algorithms also use a concept called the *eps neighborhood of a point*.

The *eps neighborhood of a point* is simply how many other points exist within a sphere of radius *eps* around the point. This is then used to classify each point.

A *core point* is defined by having at least *minPts* number of points in its *eps neighborhood*, where *minPts* is a user-defined parameter.

A *border point* is a point which does not have *minPts* in its *eps neighborhood*, but is in the *eps neighborhood* of a *core point*.

Finally, a point is classified as *noise* if it does not have *minPts* points in its *eps neighborhood* and is not in the *eps neighborhood* of a core point [21].

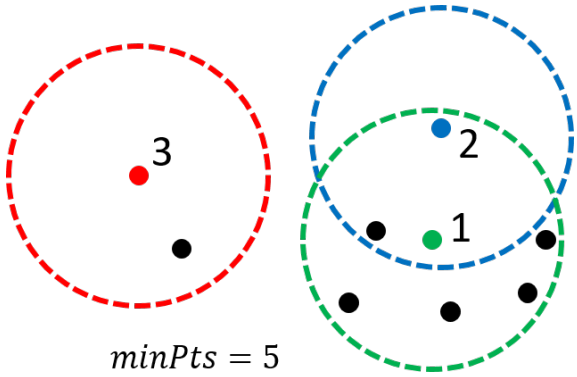


Figure 2.6: Visualisation of the density-based clustering process

Figure 2.6 visualizes the concepts described above. In the figure, the assignment of points is visualized with *minPts* set to 5. Starting with the green point (nr 1) it has 6 points in its *eps neighborhood* and is therefore a *core point*. The blue point (nr 2) does not have 5 or more points in its *eps neighborhood* but is in the *eps neighborhood* of a core point and is therefore a *border point*. Lastly, the red point (nr 3) does not have at least 5 points in its *eps neighborhood* and is not in the *eps neighborhood* of a *core point*, it is therefore labeled as noise.

DBSCAN is the original density-based clustering algorithm introduced in 1996 and OPTICS is an extension to DBSCAN which was introduced in 1999. The main difference is OPTICS's ability to define clusters of different densities while DBSCAN has a density threshold where everything denser than the threshold is considered to be clusters [21, 22].

2.3.2 Clustering Assessment

In [23], a number of clustering evaluation techniques are presented. The evaluation techniques presented are categorized into external indexes and internal indexes. External indexes refer to indexes that utilize the ground truth labels of the data and internal indexes refer to indexes that score the clustering structure, without regard to the ground truth labels.

The authors do not, however, come to any conclusion about which techniques are the best, but state that a human judge probably should be used in conjunction with the indexes [23].

The evaluation techniques used in this thesis are *Clustering Entropy* which is an external index and the *Silhouette Coefficient* which is an internal index.

Clustering Entropy is divided into two indexes where a low entropy indicates a good clustering performance [24]. The two indexes are:

- Cluster Entropy which reflects how homogeneous each cluster is with regard to the ground truth classes.

- Class Entropy which reflect how well each ground truth class has been distributed into the same cluster.

Both these indexes have drawbacks that affect how well they reflect the clustering performance, but when used in conjunction they balance the weakness of the other. For *Cluster Entropy*, the weakness is that assigning a new cluster to each point in the feature space would result in each cluster having perfect homogeneity. This is balanced out by the fact that *Class Entropy* will be high in that case, since it penalizes distributing a ground truth class across several clusters.

The weakness with *Class Entropy* is that assigning every point in the feature space to the same cluster would result in a perfect score. This is balanced out by *Cluster Entropy* since the single cluster would have low homogeneity with regard to the ground truth classes.

We move on to the mathematical expression of *Cluster Entropy*. For each cluster c_i the entropy is calculated by

$$Ec_i = -\sum_j \frac{n(l_j, c_i)}{n(c_i)} \log \frac{n(l_j, c_i)}{n(c_i)} \tag{2.9}$$

where $n(l_j, c_i)$ is the number of points with the ground truth label l_j in cluster c_i and $n(c_i)$ is the number of points in cluster c_i with a ground truth label. The overall *Cluster Entropy* is then calculated by a weighted sum of the individual cluster entropies:

$$Ec = \frac{1}{\sum_i n(c_i)} \sum_i n(c_i) Ec_i \tag{2.10}$$

We now continue with the mathematical expression of *Class Entropy*. For each class l_j, a single class entropy El_j is calculated by:

$$El_j = -\sum_i \frac{n(l_j, c_i)}{n(l_j)} \log \frac{n(l_j, c_i)}{n(l_j)} \tag{2.11}$$

where $n(l_j, c_i)$ is the number of points with the ground truth label l_j in a cluster, c_i and $n(l_j)$ is the number of points with the ground truth label l_j. The overall *Class Entropy* is the calculated by a weighted sum of the individual class entropies:

$$Ec = \frac{1}{\sum_j n(l_j)} \sum_i n(l_j) El_j \tag{2.12}$$

With these two entropies, it is possible to evaluate the overall *Clustering Entropy*, it could be combined into a single index, but since they both give different information we have chosen not to.

The second index, *Silhouette Coefficient*, does not utilize the ground truth labels and therefore evaluates the structure of the feature space. The *Silhouette Coefficient* is bounded between -1 and 1 where -1 indicates incorrect clustering. Scores around

0 indicate overlapping clusters and scores close to 1 indicates a highly dense and separated clusters. One drawback with this score is that it is usually low for non-convex clusters, meaning that clusters generated by density-based clustering might receive a low score since they are not necessarily convex clusters [25, 26].

The score for a single sample is defined as:

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)} \tag{2.13}$$

where b_i is the mean distance between a sample and all other point in the same cluster and a_i is the mean distance between a sample and all other points in the next nearest cluster. The score for all samples in the clustering is then calculated as the mean of all scores:

$$s = \frac{1}{n(s_i)} \sum_i s_i \tag{2.14}$$

With these two scores, it is possible to both evaluate how the ground truth labels are distributed throughout the clusters and to evaluate how well separated the clusters are. Together with a human judge it should be possible to evaluate the clustering in a good way.

2.4 Deep Neural Networks

The clustering described in previous sections produces annotations for each time series that can be used to train a neural network such as a CNN or a ResNet, which are the two architectures used in this thesis.

Both networks share a few units which do not include trainable parameters but are nonetheless important to the network performance.

- Rectified Linear Unit (ReLU) is used as an activation function in neural networks. If the input is less than zero, the output is zero and if the input is larger than zero the output is the same as the input [27].
- Global Average Pooling (GAP), this unit averages all values along all but one dimension of the data, reducing the dimensions of the data.
- Softmax, this unit is often used in neural networks as the last operation to obtain a probability distribution over the classes [28].

Below a short description of the two network architectures highlighting their respective properties follows.

2.4.1 Convolutional Neural Networks

In this section, the basic properties of CNNs are described. The information presented here comes from the book *Deep Learning Book* written by Goodfellow, Bengio and Courville [28].

CNNs are especially effective for processing data that has a known grid-like topology, where a scrambling of this grid will destroy the information. Some examples of data with this structure are images that can be seen as a 2-D grid of information or time series which can be seen as a 1-D grid. However, if an image has several colors the image becomes a 3-D grid where the third dimensions include the colors. In digital images, a mix of red, green and blue are often used. This third dimension, which will be referred to as channels throughout this report, can in contrast to the other two be scrambled without loss of information.

The convolutions done by a CNN is not the same operation as the convolution used in pure math and certain fields of engineering. This convolution can be seen as applying and sliding a matrix (called filter in CNNs) over the input as shown in Figure (2.7). This convolution allows the network to find specific features, regardless of their location in the data. For example, a filter looking for vertical edges in a picture can be applied to every location and the output from this filter would have large values at the positions where edges are present. [28].

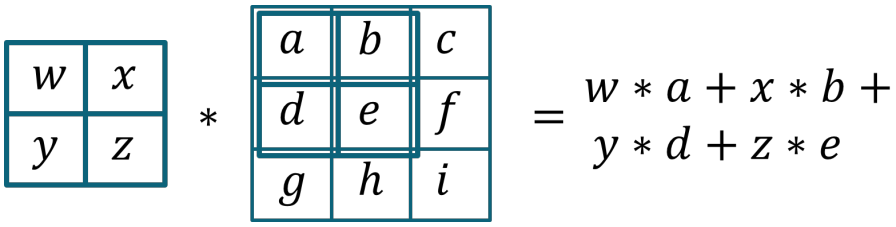


Figure 2.7: Example of the convolution done by a filter in a CNN

2.4.2 Residual Neural Networks

Residual neural networks are network architectures that have shortcuts across two or more layers. This network architecture was introduced in 2015 by [12]. One of the main problems the authors were trying to solve was the degradation problem with DNNs.

In essence, the degradation problem is that when increasing the number of layers in a network, both the training and validation errors can increase throughout the whole training process compared to a shallower network. To solve this problem He et al. [12] proposed a residual learning scheme where shortcut connections are used to bypass two or more layers and then recombine with the output, see Figure 2.8.

The authors that introduced ResNet hypothesize that it is easier to learn the residual in this way than to learn the function of the feature space and therefore this architecture solves the degradation problem. However, they can not show that

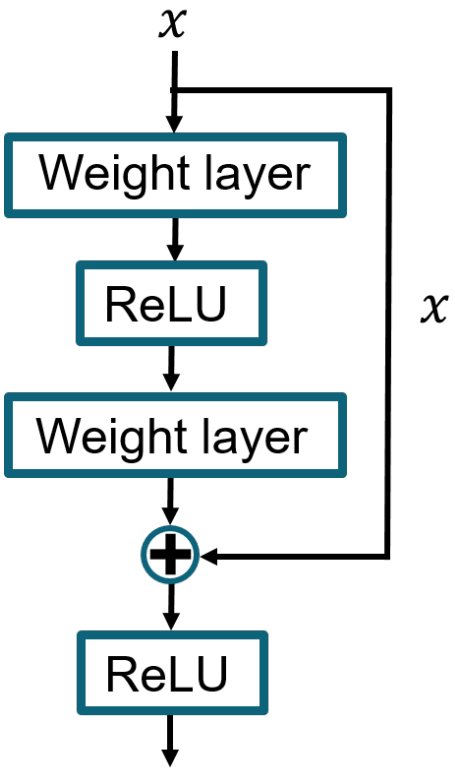


Figure 2.8: Basic residual network block

their hypothesis is correct, only that the degradation problem can be solved with ResNets. When training with the ImageNet dataset, they used a 152 layer deep residual model which achieved a test error of 3.57%, winning the Image Net Large Scale Visual Recognition Challenge (ILSVRC) in 2015 [12].

2.4.3 Classification performance measures

To measure the performance of the neural networks, the metrics *precision* and *recall* will be used. These two metrics are often used in machine learning applications and give a good measure of how well a network performs. *Precision* is defined as:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{2.15}$$

and *recall* is defined as:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{2.16}$$

Intuitively, *precision* describes how accurate a model is when it does a single classification and *recall* describes how well a model classifies the ground truth positives.

2.5 CAN-bus Protocol

Controller Area Network (CAN) is a data bus initially created for usage in vehicles [29]. The standard consists of a link-layer description as well as a physical layer description. In essence, this means that CAN standard covers both the hardware and software protocol for transmitting bytes across the network. This section will not go into depth on the inner workings of the CAN-bus but rather highlight the relevant properties.

CAN is asynchronous and re-transmissions are used to provide transmission success. This results in the measurements reported not always being delivered in equal time steps or instantaneously. The report intervals are determined by the nodes themselves and may differ between nodes, this means that different signals may be updated at different speeds. However, the standard implies a common re-transmission speed and thus the delays introduced are often considered negligible.

Each message on the bus is received by all nodes and only a single node can transmit at any time. The protocol uses frame identifiers to target devices and for devices to determine if they should act on the frames. This makes it possible to record the bus by connecting a node that records every frame. However, since each frame has its own identifier and structure, external information is needed to interpret it properly. The external information required for this thesis is provided by Volvo and is considered confidential.

3

Methods

The methods chapter contains sections describing each methodology in order of usage. It starts by describing the research environment and pre-processing, it then continues with the process of annotation, clustering, acceleration of the algorithms and verification of the final product.

In Section 2.3 we used the concept of *feature space*. However, in the rest of the report, the feature space will always be built using shapelet orderlines. To separate the notion of a feature in a broader sense, we will now refer to the feature space as the *shapelet space*.

3.1 Research environment

The thesis was performed using several existing tools. Primary tools consist of Jupyter Notebooks, Python programming language and the PyTorch machine learning framework. In addition to this primary set of tools, VISION was used to perform some initial inspection of CAN-bus recordings before the pre-processing steps were implemented and verified.

The Python programming language was extended with several external libraries. Some significant libraries include scikit-learn, matplotlib and python-can. The library scikit-learn provided clustering implementations, matplotlib provided plotting abilities and python-can assists in developing a CAN-bus recording interpreter.

The CAN-bus recordings were stored prior to this thesis and have been extracted from internal systems. The dataset will be explained in the following section.

3.2 The Dataset

The dataset constructed during this thesis consists of recordings from the CAN-bus of trucks. The recordings are extracted from fault reports which are generated by test drivers. The trucks equipped with test software include a way for test drivers to report faults. The fault recordings include a preset time of data recorded prior to the initiation of a report, along with the recording created during the fault report. This means recordings may shift in length but are at least of a preset amount of

time.

Since the fault reports are initiated by test drivers, some may not include faults as the drivers might be wrong. The reports originate from trucks with varying configurations, meaning that each recording requires a different setup to be processed.

The dataset is constructed to be multi-variate. Each time series now contain several signals that share the same time-axis. Signals have been chosen by domain experts and correlate to those used by professionals analyzing fault reports from which these recordings originate. The signal selection has been limited to those existing across all configurations of trucks. Since the on-board computer in the vehicles does some automatic signal translations, many signals exist across different configurations albeit hardware being different. Along with these criteria, the signal selection has focused on keeping the number of signals reasonable to allow for human investigation. Checking several dozens of signals for correlation and determine relevance is a workload unfeasible for this thesis. Therefore 9 signals have been chosen, based on trial and error of which signals seem to work best.

The signals chosen have been omitted in the report due to confidentiality. However, they typically represent combined sensor measurements of the physical world affecting the truck. It should be noted that not all signals provide information which is relevant to all faults, therefore, some signals might only be relevant for a subset of the faults we are attempting to classify.

3.3 Pre-processing the fault reports

The pre-processing of the data originating from the fault reports includes translating proprietary formats and reconstructing the sampled signals to evenly spaced discrete time series. Both steps will be explained in the following subsections.

3.3.1 Automatic conversion of CAN-bus recordings

The CAN-bus is recorded in proprietary binary formats that require external tools to allow translation. Due to a large number of individual files and configuration requirements, this has been automated to save time. This includes reading proprietary file formats and derivation of the correct configuration for each file.

The primary format of recordings from external test environments are in Binary Logging Format (BLF) which has been reverse-engineered and implemented in the library python-can.

The other format used by engineers in the internal test environments is the recording format used by the VISION software. Conversion of this format has not been reverse-engineered and requires the VISION software to be used. Fortunately, VISION has an external Application Programming Interface (API) which allows external programs to request conversion of specific files to other formats. One format VISION supports as a target for conversion was Comma Separated Values (CSV) encoding,

which is readable using Pythons native CSV module. It should be noted that this still requires the software to be correctly configured, something which can not be done via external API's. Hence this part may require some manual intervention.

To perform conversion of both mentioned formats, databases containing message translations are required. These databases consist of format specifiers, message identifiers and frame identifiers used to decipher the binary chunks contained in the recording formats. These translative databases are specific to each truck configuration and on-board computer.

The proprietary formats are converted into a python list object which is used in the following section.

3.3.2 Signal reconstruction from CAN-bus recordings

The processing of CAN recordings involves translation from a record of momentary value changes to evenly spaced discrete time series. Since different signals in the CAN have different update frequencies, values can arrive at different time steps. Before receiving a signal, there is no value available. To avoid missing values or issues with zero-hold being interpreted by shapelets, the signal reconstruction waits for values for all selected signals before starting the reconstructed signal. Thus, some values can be dropped and not included in the reconstructed signal.

When the reconstruction has begun, the values are sampled using zero-order hold, meaning that if no new value has arrived the previous is used. An example of the complete reconstruction is shown in Figure 3.1 where three signals with different update frequencies are shown before and after reconstruction. Looking at the orange signal in Figure 3.1, the value changes become a bit delayed after the reconstruction. The matrix produced by the reconstruction is saved to disk using CSV.

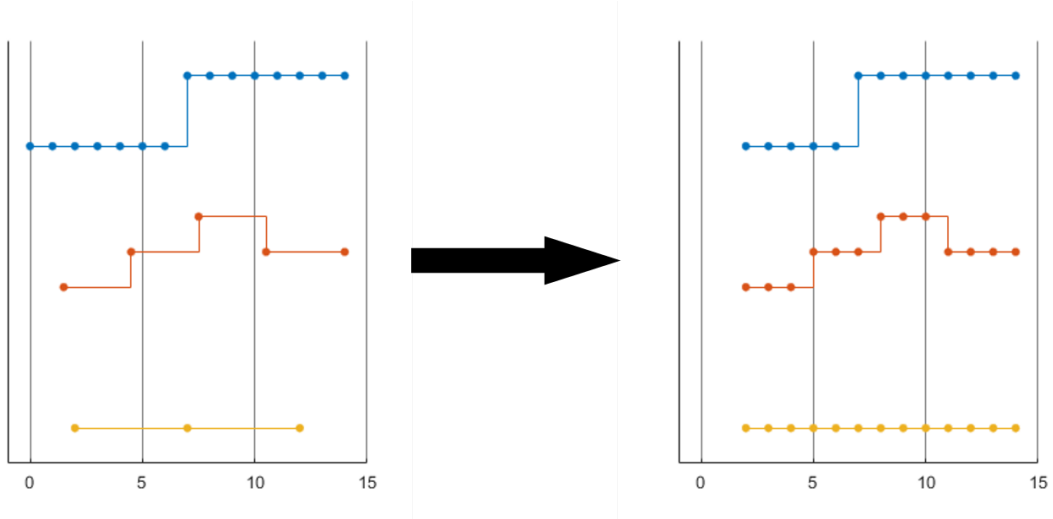


Figure 3.1: Example of a signal reconstruction from a CAN bus recording.

The implemented algorithms expect time series to have equal lengths. This requires

the reconstructed signals to be sliced accordingly. A majority of the fault reports extracted contain recordings that are more than 20 seconds and usually around 25 seconds. The minimum length of 20 seconds was chosen as the length to slice recordings by. This is motivated by aiming for as many data points as possible.

After being sliced, the newly constructed dataset is stored using PyTorch Tensors in a binary format provided by PyTorch load and save functionality. This enables the dataset to quickly be used without performing signal reconstruction or conversion of recordings.

In summary, the signal reconstruction requires three parameters: which signals to reconstruct, the sampling frequency and the length of the slices. Because of confidentiality with Volvo, we are not able to share which signals are being used. The two other parameters chosen, sampling frequency and slice length are 5 Hz and 100 samples (20 seconds) respectively.

The selection of signals is based on what domain experts use when manually searching for the faults that this thesis aims to find. This means that all signals should have relevance for at least on fault, but may not be relevant for all faults.

The values dropped by the signal reconstruction before all signals gain values could result in a loss of valuable information. However, the selected signals have update frequencies that occur within a span of a few seconds and faults are typically present closer to the center of the recordings.

When reconstructing a signal, the sampling frequency of 5 Hz could offset a value change by at most $200ms$ in the reconstructed time series, compared to the raw CAN-log. It is also possible that several value changes occur within the sampling period, potentially causing a loss of information. However, this is considered negligible since most signals do not have that high update frequency and the multiple value change is likely to be a value close to the sampled value. E.g. consider a reconstructed sine wave, the value changes occurring between sampling are intermediary values leading up to the sampled value. Since the shapelet algorithms compute normalized distances, the effect is negligible.

3.3.3 Pre-processing system architecture

The pre-processing is done in several steps with two types of intermediate formats to facilitate rapid experimentation and development. An architectural overview can be seen in Figure 3.2. The first step translates proprietary recording formats using translative databases into CSV without any information loss. The second step selects specific signals, reconstructs and samples them with a given frequency. The third step slices and converts the sampled signals into PyTorch Tensors which are saved to file using PyTorch's load and save functionality.

By having the intermediate CSV files without data loss, signal choice and sampling frequency can be modified without having to wait for the heavier task of translating

proprietary formats. By storing the sampled signals as PyTorch Tensors, the dataset is directly ready to be used without any pre-processing. This allows the creation of multiple variants of the dataset which can be quickly exchanged during development and investigation.

Figure 3.2: Pre-processing system architecture diagram

3.4 Annotating the Dataset

A significant task of this master thesis was to derive a method for annotating a large dataset with minimum manual effort. The dataset is mostly unlabeled and consists of multivariate time series. The approach taken is to extract shapelets from the time series and use them to find similar time series. By finding the relevant features defining a class, the time series of that class would return a distinct similarity measurement effectively separating them from time series of other classes. Combining the similarity measures between different shapelets allows us to formulate the problem as multi-dimensional clustering and utilize existing techniques to annotate each group of time series.

The final shapelet derivation algorithm can be seen as an extension on both the Unsupervised Shapelets introduced by Zakira et al.[10] and Shapelets by Keogh et al. [9]. In short, the fully unsupervised shapelets was extended with an ability to utilize a partial ground truth when ranking shapelets to control which similarities are considered. An ability inspired by the *Information Gain* measurement introduced in [9]. The shapelet derivation was also modified to enable multi-variate time series. Both abilities will be further expanded upon in the following subsections.

3.4.1 Extending Unsupervised Shapelets

The unsupervised algorithm introduced by Zakira et al.[10] was developed for classification of univariate time series. This thesis uses multivariate time series, thus the algorithm must be expanded. This expansion involves two parts, extending the distance measure to handle multivariate time series, presented in Section 3.4.1.1, extending and modifying the scoring to allow a unified score, presented in Section 3.4.1.2.

3.4.1.1 Multivariate Distance Measure

The univariate distance measure *SubsequenceDist* (see Definition 4), computes the distance between the current shapelet and the given time series using the Euclidean norm. To handle a multi-variate case, the distance measure is redefined to select relevant variates, compute the Euclidean norm between each variate and combine the distances to a normalized single value distance measure.

To select relevant variates, the signals are analyzed by using the ground truth of a subset of time series, making this a semi-supervised algorithm. The selection function is denoted as a signal filter as it filters out non-relevant signals for each shapelet.

Algorithm 5 Signal filter derivation

Require: C_c: A list of classes
Require: T_t: An index map mapping t in the dataset to c in the list of classes
Require: $D_{t,s,u}$: The matrix of distances between a shapelet and all subsequences
Ensure: F_s: A boolean map of signal contributions
  $X_{t,s}$ = infinity
  $S_s = 0.0$
  F_s = False
  for all $t, s, u \in D$ **do**
    if $X_{t,s} > D_{t,s,u}$ **then**
      $X_{t,s} = D_{t,s,u}$
    end if
  end for
  for all $c \in C$ **do**
    $M_s = 0.0$
    for all s in X **do**
      $M_s = mean(\{X_{t,s} \| T_t = c\})$
    end for
    $S_s = std(M_c)$
  end for
  for all $s \in S$ **do**
    if $S_s \geq mean(S)$ **then**
      F_s = True
    end if
  end for
  return F_s

The signal filter is given by Algorithm 5. It consists of three loops. The first loop computes the minimum distance of each individual signal in each time series and stores the value in X. The second loop computes the standard deviation of the mean of minimum distances for each signal. The third loop marks signals for inclusion. The inclusion is determined by considering the previously computed standard deviation of means of the signals. The threshold for including a signal is given by the mean of all deviations in S.

To clarify this algorithm, a visualization is provided in Figure 3.3. The intuition is that the average minimum distance for a signal should deviate between classes for it to be considered useful. If the average minimum distance for a specific signal does not deviate, then the signal does not contribute to separating items from different classes.

It is important to note that subsequences giving the minimum distance for each

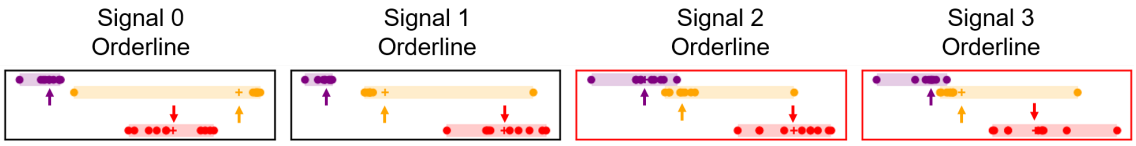


Figure 3.3: Visualization of the signal filter. The mean of each cluster is marked with an arrow of the same color as the cluster. With signals 0 and 1, the means are spread out across the orderline, thus being included in the filter. For signals 2 and 3, the yellow and purple means are close to each other, thus the signals gets filtered out

signal may not share the same starting point in the same time series. This means that each signal is unconstrained from the performance of other signals when being considered for inclusion. In other words, each signal is considered as if it was the only signal available. The joint performance of the signals marked for inclusion is not considered.

The inclusion threshold based on the mean of standard deviations has the drawback of always filtering out a signal, even if all signals are contributing. In the case of no contributing signals, the threshold will be equal to the value of all items and all signals will be marked for inclusion. However, the algorithm is intentionally created with a dynamic threshold not to filter out all signals on bad shapelet candidates. As the Volvo dataset is the target dataset, and it contains several signals which are only useful for separating a subset of classes, it is considered acceptable that at least one signal is always filtered.

After the filter is derived and applied, only the remaining signals are considered. The minimal subsequence distance is found by summing the remaining signals and extracting the index of the minimal subsequence. Different from the filtering, this means that the subsequences from each signal are synchronized in time. The minimal subsequence is normalized by the square root of the shapelet length to avoid scoring bias based on shapelet lengths. The complete sequence can be seen in Algorithm 6.

3.4.1.2 Shapelet Scoring

The measures *Information Gain* and *Gap* both aim to find the optimal split strategy on the distance vector. *Information Gain* is aimed at the fully supervised scenario and *Gap* at the fully unsupervised scenario. To gain some control over the similarities chosen a new measure is derived, denoted as *Custom Score*. The *Custom Score* is more tailored towards the Volvo dataset and aims to score by utilizing the partial truth.

The *Custom Score* measure bases its measurement on three sub-measures. Shared between the sub-measures is the use of intervals. Where interval means the range between the minimum and maximum distance measure of time series of the class. This means that for each known class of objects, the interval between the leftmost and rightmost item on the orderline is derived. These intervals are then used to

Algorithm 6 Multivariate Distance Measure

Require: S: A shapelet candidate
Require: C_c: A list of classes
Require: T_t: An index map mapping t in the dataset to c in the list of classes
Require: D: A 3-dimensional array of all subsequences i for all signals s and all time series t
Ensure: d: A distance matrix containing the distance measure between the shapelet and each signal in each time series.

```
dis ← []
for all t in D(:,:,:) do
    for all s in D(t,:,:) do
        for all i in D(t,s,:) do
            dis(t,s,i) ← euclideanNorm(zNorm(S_s), zNorm(D(t,s,i)))
        end for
    end for
end for
signals ← derive_signal_filter(C, T, dis)
d ← []
for all t in dis do
    avg_dis ← []
    for all s in dis(t,:,:) do
        for all i in dis(t,s,:) do
            if signals(s) is True then
                avg_dis(i) ← avg_dis(i) + dis(t,s,i)
            end if
        end for
        d(t,s) ← dis(t,s,argmin(avg_dis))
    end for
end for
return d/sqrt(|S|)
```

compute the sub-measures. The sub-measures consists of the amount of overlap amongst intervals, the length of intervals in relation to the distance vector (denoted as tightness) and the length of the interval with the lowest starting position in relation to the distance vector (denoted as leftmost tightness).

The sub-measure measuring overlap is defined by Definition 15 and is computed with Algorithm 7. In Figure 3.4 the overlap measure is visualized.

Definition 15. [*Overlap*] Overlap is defined as the sum of segments of the distance vector which are covered by more than one interval defined by objects in known classes.

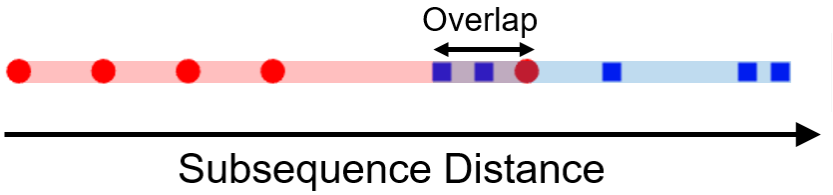


Figure 3.4: Visualization of an overlap between two clusters.

Algorithm 7 Computing the overlap measure

Require: C: A list of classes
Require: D: A list of distances for each item with a known truth
Ensure: The largest overlap in relation to the size of a class
  largest = 0.0
  for all i in C **do**
    $largest_i = 0.0$
    for all j in C **do**
      if $i \neq j$ and $\min(D_j) < \max(D_i)$ and $\max(D_j) > \min(D_i)$ **then**
        upper = $\min(\max(D_j), \max(D_i))$
        lower = $\max(\min(D_j), \min(D_i))$
        overlap = (upper - lower) / $(\max(D_i) - \min(D_i))$
        if overlap $> largest_i$ **then**
          $largest_i$ = overlap
        end if
      end if
    end for
    largest += $largest_i$
  end for
  return $largest/\|C\|$

The tightness sub-measure is described in Definition 16. The intuition is that short intervals where at least two intervals are placed far apart on the distance vector should be rewarded and long intervals as well as short intervals placed close together should be punished.

Definition 16. [*Tightness*] We define tightness as the length of all intervals given by the minimum and maximum subsequence distance of each class, divided by the length of the distance vector. n represents the number of known classes, D_i represents the subset of the distance vector containing items known to be of class i and D represents the distance vector.

$$T = \min(1, \max(0, \frac{\sum_{i=1}^{n} max(D_i) - min(D_i)}{max(D) - min(D)})) \tag{3.1}$$

The reader should note that the tightness sub-measure does not take into account overlapping intervals and can yield a score higher than one. Which is why the score

is limited to the range zero to one. E.g. if three intervals are completely overlapping, the score, without being limited, would be $(1 + 1 + 1)/1 = 3$.

The leftmost tightness is similar to the tightness measure described previously, but this sub-measure only consider the leftmost interval. The leftmost interval denoting the interval of the class which contains the time series with the minimum subsequence distance. The intuition is that the interval closest to 0 on the orderline is the similarity feature and thus the tightness measures how well the time series of the same class share the feature. The leftmost tightness measure is defined in Definition 17.

Definition 17. [*Leftmost Tightness*] We define leftmost tightness as the interval of the class containing the time series with the minimum subsequence distance divided by the length of the distance vector.

$$LT = \min(1, \max(0, \frac{max(D_{argmin(D)}) - min(D)}{max(D) - min(D)})) \tag{3.2}$$

These three sub-measures are weighted into the final measure called *Custom Score*, which ranges from zero to one, see (3.3). These sub-measurements are weighted into a single value using hyperparameters which can be tailored to the dataset at hand. The hyperparameters are named α, β and γ, they should be non-negative and their sum should be equal to 1 to produce a correct value. In this thesis, the hyperparameters has been chosen as: $\alpha = 0.7$, $\beta = 0.2$ and $\gamma = 0.1$.

$$CustomScore = \alpha * Tightness + \beta * Overlap + \gamma * LeftmostTightness \tag{3.3}$$

3.4.2 Shapelet Derivation Tree

The unsupervised algorithm, Algorithm 4, separates and removes the most similar group of data iteratively. This requires each shapelet to be a distinct feature for each class and **not** be present across classes. If it is, the objects from the other classes are likely to be separated along with the intended class.

This limits the ability to separate data since there must exist a distinct feature for each class. This is visualized in Figure 3.5 where the separated left side contains two classes and the right side contains only one. The unsupervised algorithm would not separate the red and green classes.

To free the algorithm from these limitations, the algorithm was extended to consider both sides of the separation separately. The extension is to continue the derivation on each resulting subset in a recursive fashion until a stopping criterion is met. The stopping criterion was based on the subset with known truth to determine if there is only one known label left. If so, the algorithm finishes. This is stricter than the unsupervised algorithm which stops when only a single object is separated as a group.

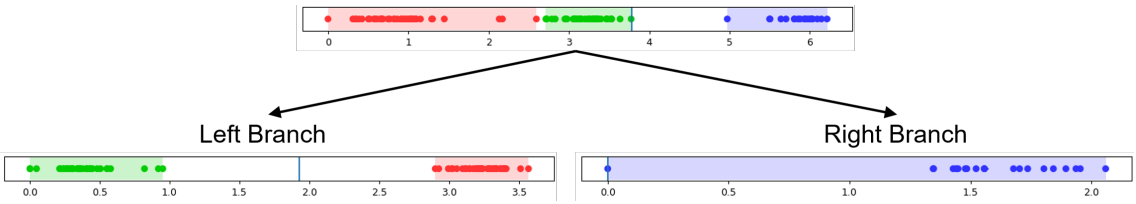


Figure 3.5: Illustration showing that considering every branch is necessary to find all features of a dataset.

However, the paper defining u-shapelets [10] also concludes the algorithm always extracts to many shapelets. Since the aim of this thesis is to control similarities used to separate data, it is sensible to stop when the ability to control the selection is lost. I.e. the subset with known truth is fully separated.

Algorithm 8 tree: The recursive Shapelet Tree derivation function

Require: D: A dataset
Ensure: The highest scoring label for the provided dataset
  if number of known classes in $D \leq 1$ **then**
    return Nothing
  end if
  candidates = derive_candidates(D)
  best_candidate_score = $-\infty$
  best_candidate_splitpoint = Nothing
  best_candidate_dis = Nothing
  best_candidate = Nothing
  for all c in candidates **do**
    dis = compute_distance_matrix(c, D)
    score, splitpoint = compute_score(dis)
    if score > best_candidate_score **then**
      best_candidate_score = score
      best_candidate = c
      best_candidate_splitpoint = splitpoint
      best_candidate_dis = dis
    end if
  end for
  left_set = D where best_candidate_dis < best_candidate_splitpoint
  right_set = D where best_candidate_dis > best_candidate_splitpoint
  left_branch = tree(left_set)
  right_branch = tree(right_set)
  return best_candidate, left_branch, right_branch

3.5 Clustering

Annotating the time series is done by using a clustering algorithm. In this thesis, we opted to use the density-based clustering algorithm OPTICS. While the OPTICS

algorithm is quite complex to implement, it is straight forward to use with the scikit python library. The user is only required to provide two inputs: a proximity measure and *minPts*. The proximity measure in this thesis is the distance matrix created by the shapelets. The parameter *minPts* defines how many points are required to be in an area for the area to be considered a cluster. More in-depth information about OPTICS is provided in Section 2.3.1.

The reasons for choosing the density-based algorithm OPTICS comes down to three properties that were deemed desirable, namely the ability to:

- assign an arbitrary number of clusters,
- detect clusters of arbitrary shape and density,
- label points in the shapelet space as noise.

Since the dataset consisted of fault reports at Volvo, there were bound to be more error classes in the data than we had ground truth data for. This motivates the need for the first ability.

With the ability to detect clusters of arbitrary shape and density, no assumptions had to be made about how the time series would be arranged in the shapelet space, motivating the second ability.

Lastly, the ability to label noise was necessary since it is not uncommon that an error report at Volvo describes something that is not actually an error. It might just be that the driver reporting an error, want a different behaviour.

The properties described above stand in contrast to other clustering methods such as K-means, which tries to minimize the variance within each cluster around some central point. This results in K-means dividing the whole feature space into Voronoi cells, forcing every point into a cluster. K-means also require the user to input the number of clusters to create and thus the user has to know how many clusters the data should contain [30].

In Figure 3.6, a comparison between K-means and OPTICS is visualized. For example, the two moons dataset (the first column of the figure) is correctly labeled with OPTICS, while K-means splits them in an incorrect way. The figure was generated using modified code from the scikit-learns webpage about different clustering algorithms [31, 26].

3.5.1 Clustering Evaluation

In this thesis, a subset of the data has predefined ground truth labels and to use the all accessible information, we decided to use the predefined labels in the clustering evaluation which is done with a measure called *Clustering Entropy*. Along with using the ground truth labels we also use a measure called *Silhouette Score* and visual inspection of the clustering.

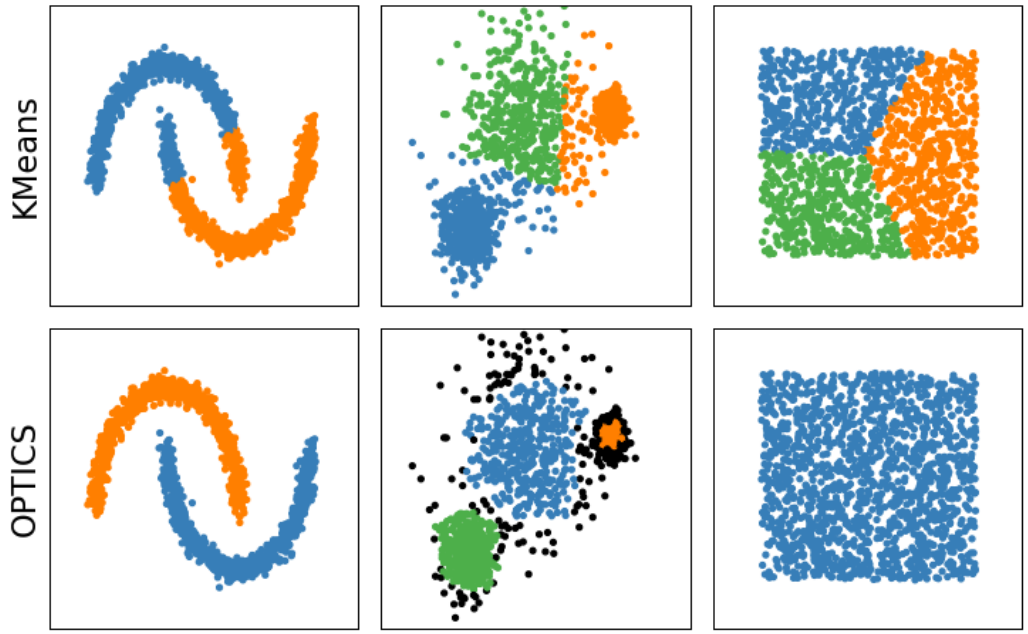


Figure 3.6: Comparison of K-means clustering versus OPTICS clustering. The first row is clustering by K-means and the second row is clustering done by OPTICS

In summary, the clustering entropy reveals how well the ground truth labels are distributed within the different clusters. An entropy of zero would mean that each ground truth class has its own cluster and that they are not distributed across several clusters. A silhouette coefficient near 1 indicates that the clusters are dense, convex clusters that are well separated from each other.

However, there are problems with both scores and that is why a visual inspection of the clustering is used as well. The drawback with clustering entropy is that it requires ground truth classes, therefore it does evaluate how the unlabeled time series are distributed in the shapelet space. The drawback with silhouette score is that it gives a low score for non-convex clusters (such as the two moons dataset in Figure 3.6) even if they are well separated.

Visual inspection can compensate for the drawbacks of the two indexes, but even this has its drawbacks, especially when the shapelet space is high-dimensional. With a shapelet space of 4 or more dimensions, it becomes hard to judge the clustering by means of visualization.

In conclusion, evaluating clustering is hard without access to all ground truth labels [23, 24], but combining several methods can help compensate for the drawbacks of each method.

3.6 Training the Neural Networks

To train the DNNs used in this thesis, the labels produced from the shapelet and clustering algorithms are used. The two network used are visualized in Figure 3.7

and come from [3]. The learning rates used when training the neural networks were derived empirically to be 0.001 and 0.0007 for the CNN and ResNet respectively.

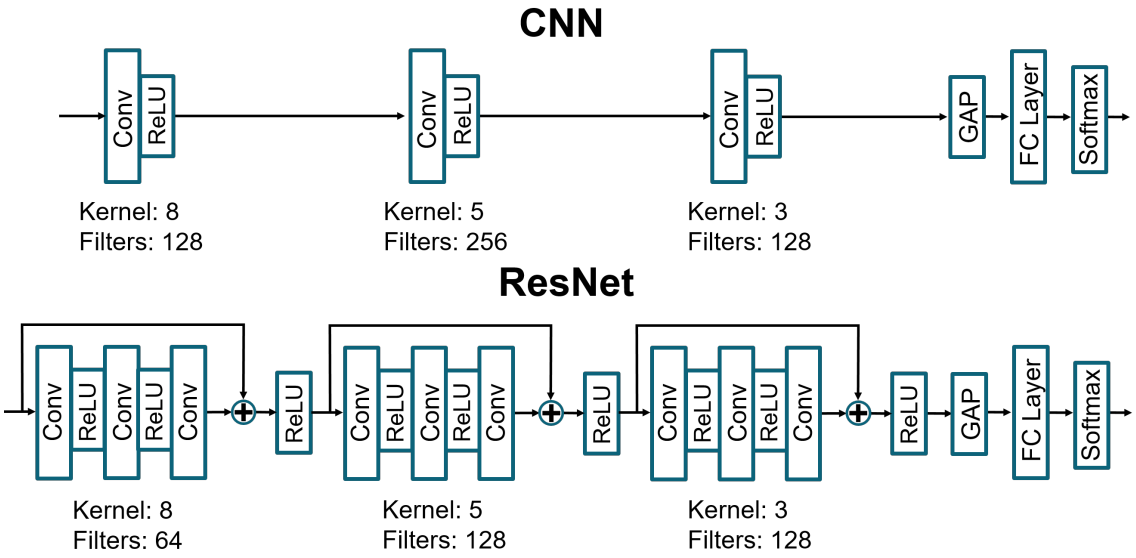


Figure 3.7: Network structures used in the thesis

3.6.1 Training- and Validation-set split

When splitting the dataset into training and validation sets, the split ratio is 85% training data and 15% validation data. Normally, the split is done randomly, but the dataset we have has a total of almost 100 time series. Since the dataset is so small, there is a risk that not every class is represented in the validation set. To deal with this, the dataset split is not totally random, instead, we split the classes into separate lists and randomly select 3 samples from each class to go into the validation set.

The dataset split has a low proportion of validation data against testing data, but when testing a higher validation ratio, the training suffered and the performance for both training and validation decreased. Therefore we decided to go with this split since it gave the best training and validation performance.

3.6.2 Convolutional Neural Network Architecture

The CNN used in this thesis has 4 layers in total where the three first are convolutional layers and the last layer is a fully connected layer. The convolutional layers have 1-D kernels of $\{8, 5, 3\}$ with stride 1 and have each $\{128, 256, 128\}$ filters. They are all batch normalized and followed by a ReLU unit.

Between the last convolutional layer and the fully connected layer there is a global average pooling unit which averages each incoming channel into a single value, reducing the amount of data into the fully connected layer. The whole network is then finished off with a softmax unit to provide a probability distribution as output [32].

3.6.3 Residual Network Architecture

Compared to the CNN, the ResNet has a very deep structure, having 10 layers in total. As mentioned in Section 2.4.2, this depth can be used without the model overfitting because of the shortcuts the ResNet structures use.

The ResNet organizes its layers into blocks with three convolutional layers, each batch normalized and followed by ReLU units. The input to the first layer of each block shortcuts to the output of the third layer and runs through the last ReLU unit as shown in Figure 3.7. There are three blocks with kernels $\{8, 5, 3\}$ and number of filters $\{64, 128, 128\}$. The network is finished in the same way as the CNN, with a Global Average Pooling layer and a fully connected layer, finished of by a softmax unit [32].

3.7 GPU Acceleration

The algorithms used in this thesis rely on performing the same set operations on a large set of data. This allows for massive amounts of parallelism as there is minimal branching, something GPU's are intended for. The two most optimizable operations were computing the Euclidean distance and performing Z-normalization. These operations make up the core of the operations when discovering shapelets.

In [18], the authors focus on converting the existing u-shapelet discovery algorithm to an anytime algorithm, in order to accelerate and improve the shapelet discovery. This new anytime algorithm tries to estimate which time series in the dataset is most likely to contain a good shapelet candidate, in contrast to the original u-shapelet algorithm which extracted the shapelet from the first time series in the dataset. For this thesis, the additional work added by this algorithm and the analysis required to verify its usefulness was not justified. Instead minor adaptations on the implementation of existing algorithms were implemented to make use of the hardware at hand to perform the thesis work in a feasible amount of time.

The adaptations applied to the implementation did not change the algorithm. Instead of performing each normalization or distance measure in a loop, the dataset and candidates were structured to make full use of the memory available and all operations are executed in parallel on the GPU.

The typical approach to parallelizing operations was as follows, start by moving a minimal amount of data to the GPU memory (this is a slow operation), expand the data by repeating, folding and unfolding dimensions to place the data in a single vector. Perform an operation and reshape the result back to a relevant shape and then move the data back to the CPU environment.

These accelerated implementations enabled processing of all datasets in the order of minutes on typical high-end laptops. This is considered feasible and thus no more optimization efforts were performed.

3.8 Verification

To verify that the algorithms work as intended and are applicable to other datasets than the Volvo dataset, additional datasets have been included for verification. The first dataset is a constructed dataset called toy. Two external datasets have also been included, one containing accelerometer data from an Epileptic Seizure Recognition study, and the known univariate GunPoint dataset used in several shapelet papers [10, 9]. The reason for choosing these datasets for verification is to evaluate the algorithms in different ways.

The toy dataset enables showcasing and evaluation of the properties of the algorithms. It also helps conveying the intuition and behaviour of the algorithms in this thesis. The external datasets are included to test the performance and applicability to other types of datasets than the dataset from Volvo.

3.8.1 Toy Dataset

The toy dataset contains 5 classes and 4 signals. The reason for including 5 classes is to demonstrate the shapelet algorithm's ability to handle multi-class problems. Including 4 signals enables the demonstration of the signal filter, showing that the signal filter can ignore certain signals that do not contribute any information to the separation of classes.

All signals are based on uniform noise in the range $[-1, 1]$. The first two signals have induced shapes that define the corresponding class. The induced shapes, along with all signals and classes in the toy dataset can be seen in Figure 3.8. The last two signals do not have any induced shapes and should therefore always be filtered out by the signal filter as they consist of only noise.

The shapelet algorithms should be able to separate these classes. However, classes 1 & 2 are quite similar since both have an upward slope in signal 1 and a downward slope in signal 2. This means that the shapelet algorithm is likely to put them quite close in the feature space.

3.8.2 External Datasets

The external datasets, Epileptic Seizure Recognition (ESR), by Villar et al. [33], and GunPoint, from the UCR archive [34], are two sets which depict different problems. The ESR dataset contains 3-axis accelerometer data from a study on epileptic seizures recording with the intention of creating a dataset for seizure recognition. The dataset consists of five labels, one depicting an actual epileptic seizure and the other four depicting different types of activity that could be confused with seizures.

The GunPoint dataset contains a single signal that depicts the X-axis position of the centroid of a hand. There exist two classes, one where the hand pulls a gun out of a holster and aims, then putting the gun back, and one where the hand does not pull the gun from the holster but aims using the hand as a fictive gun.

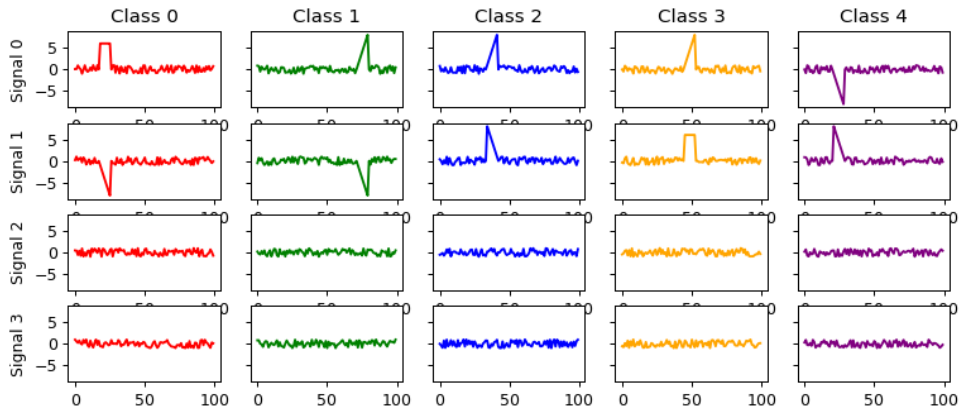


Figure 3.8: Example of the toy dataset classes and signals. Every class has some induced shapes that would not be possible to achieve with a uniform noise ranging between $[-1, 1]$.

The noteworthy differences between these two datasets are that the ESR dataset contains several signals and classes, while the GunPoint dataset contains only one signal and two classes.

3.9 Benchmark

In order to test if our algorithms improved the clustering performance, compared to the unsupervised algorithm, we have set up a test with 4 different algorithm-setups. The setups are shown in Table 3.1.

Table 3.1: Testing setups

Setup	Shapelet Algorithm	Signal Filter	Score measurement
Setup 1	Unsupervised	Disabled	Gap
Setup 2	Unsupervised	Enabled	Gap
Setup 3	Shapelet Tree	Disabled	Custom Score
Setup 4	Shapelet Tree	Enabled	Custom Score

Setup 1 is the unsupervised algorithm, Algorithm 4, with one adjustment. The unsupervised algorithm was developed to be used on univariate time series. So to deal with this limitation, all orderlines are summed to a univariate signal.

Setup 2 adds the signal filter before summing the signals in the unsupervised algorithm to evaluate what impact removing non-contributing signals have.

Setup 3 is the shapelet tree derivation, defined in Algorithm 8, in combination with the *Custom Score*, defined in (3.3).

Finally, setup 4 is setup 3 with the signal filter enabled.

4

Results & Discussion

In this chapter, the results are presented and discussed, highlighting the strengths and weaknesses of our approach. First, the results from evaluating the algorithms according to the benchmark are presented. Thereafter follows an analysis of the shapelets extracted using the produced algorithm, as well as an analysis of the clustering using said shapelets. After shapelet extraction and clustering follows the results from training DNNs, along with an associated analysis on their performance and ability to classify fault-reports. Lastly, the results from the comparison using external datasets are presented as a means to verify the algorithm.

4.1 Benchmark results

In this section, a comparison between the original shapelet algorithm and the algorithm produced is presented. The scores for performing clustering using the different benchmarks are found in Table 4.1. The dataset used for evaluation is the Volvo dataset, as this is the aim of the thesis.

The results show an improvement of using the algorithm produced against the naive extensions of the unsupervised algorithm, Algorithm 4, using summed signals and a univariate approach. However, the unsupervised algorithm is performing remarkably well on its own.

The primary improvement of the modifications is the ability to produce perfectly homogeneous clusters with respect to ground truth classes. This is in line with the goals of the thesis, to assert control over the similarities chosen.

Table 4.1: Comparison of four different shapelet algorithms showing both clustering entropies as well as silhouette score. Setups 1 and 2 are the unsupervised versions and setups 3 and 4 are the semi-supervised shapelet tree versions.

Setup	Cluster Entropy	Class Entropy	Silhouette Score
Setup 1	0.1300	0.6890	0.1876
Setup 2	0.1300	0.6890	0.1876
Setup 3	0.0000	0.3818	-0.0064
Setup 4	0.0000	0.4423	0.2391

The results also show that the signal filter does not affect the unsupervised algorithm, it derived the same shapelets and produced the same clustering. An in-depth analysis of this is missing, however, we speculate that the signals marked as non-contributing by the filter place the same time series around the same location on the orderline and hence yield no significant offsets or shifts on the summed orderline.

The unsupervised setups only extract a single shapelet, which is shown in Figure 4.1, with the colors denoting different ground truth classes. To make it easier to see where all time series are placed on the orderline, each class is put on a separate y-level, this will be the case with all orderlines in the results chapter.

In Figure 4.2 the orderline from the clustering is shown. Here the colors denote the different clusters OPTICS has defined. Comparing the two figures, it can be seen that the ground truth classes become mixed, this is why the cluster entropy is not zero. This behaviour is the motivation behind the shapelet tree derivation algorithm, Algorithm 8, which would continue derivation on the left side of the split point.

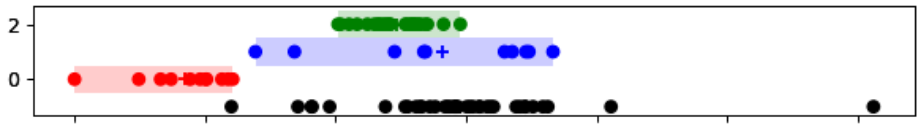


Figure 4.1: This is the orderline created by the baseline algorithms with the ground truth classes denoted by the different colors and black denoting a time series with unknown truth.

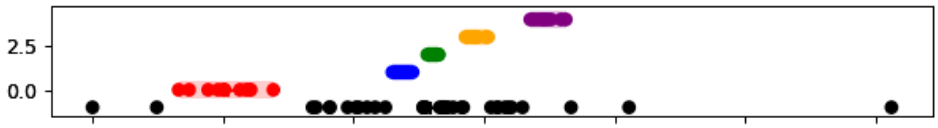


Figure 4.2: This is the same orderline as in figure 4.1 but here the colors denote the clustering labels with black denoting time series regarded as noise.

For the setups using the produced algorithm, the signal filter has a visible effect. The results in Table 4.1 may show small differences. However, the silhouette score close to zero is an indicator that something might be wrong.

To inspect the results from the shapelet tree closer, the labels from the clustering algorithm as well as the shapelet space plots are visually analyzed. In this section, an analysis of the unfiltered results is done and in the following section, a more in-depth analysis will be done for the filtered results.

In Figure 4.3 the shapelet space from the unfiltered setup using shapelet tree derivation is shown, with the colors denoting different ground truth classes. The green and blue classes have a slight overlap while the red class is well separated from the others, but is a bit dispersed. Looking at Figure 4.4, both the blue and green classes

are labeled as noise. This is not a good result since it means that the neural network will not be able to classify any of these faults when deployed, as it is trained to regard them as noise.

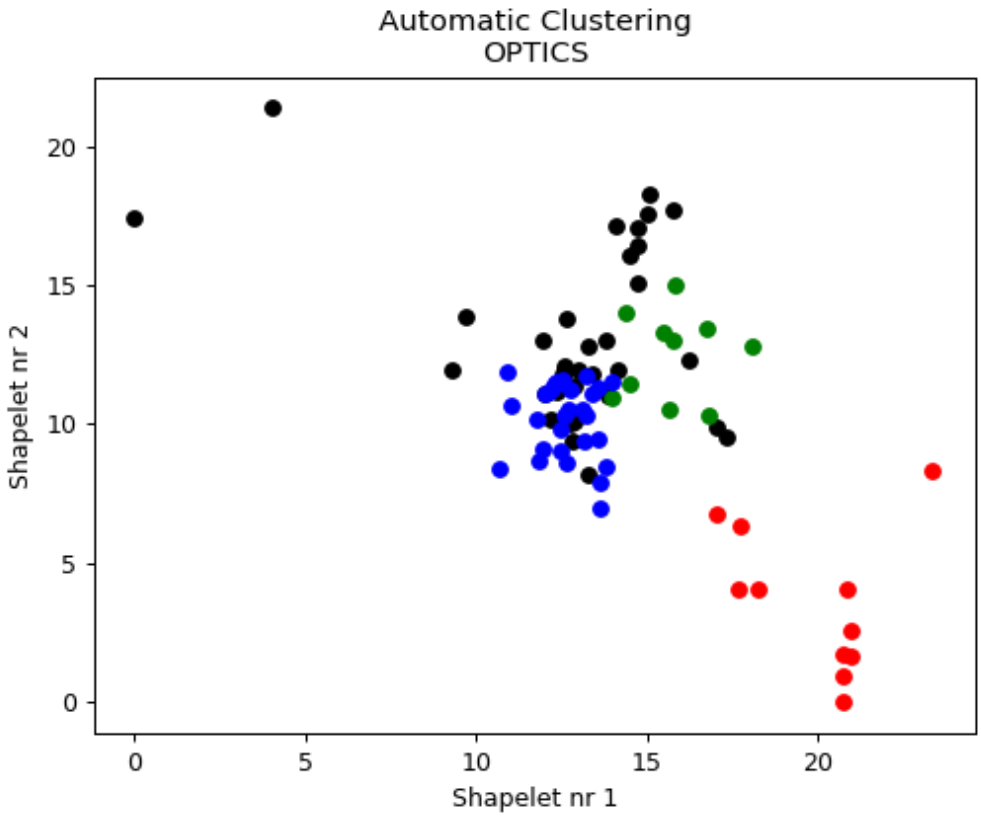


Figure 4.3: This is the shapelet space created by the unfiltered shapelet tree algorithm with the colors denoting ground truth classes. Again, black denotes time series with no known truth.

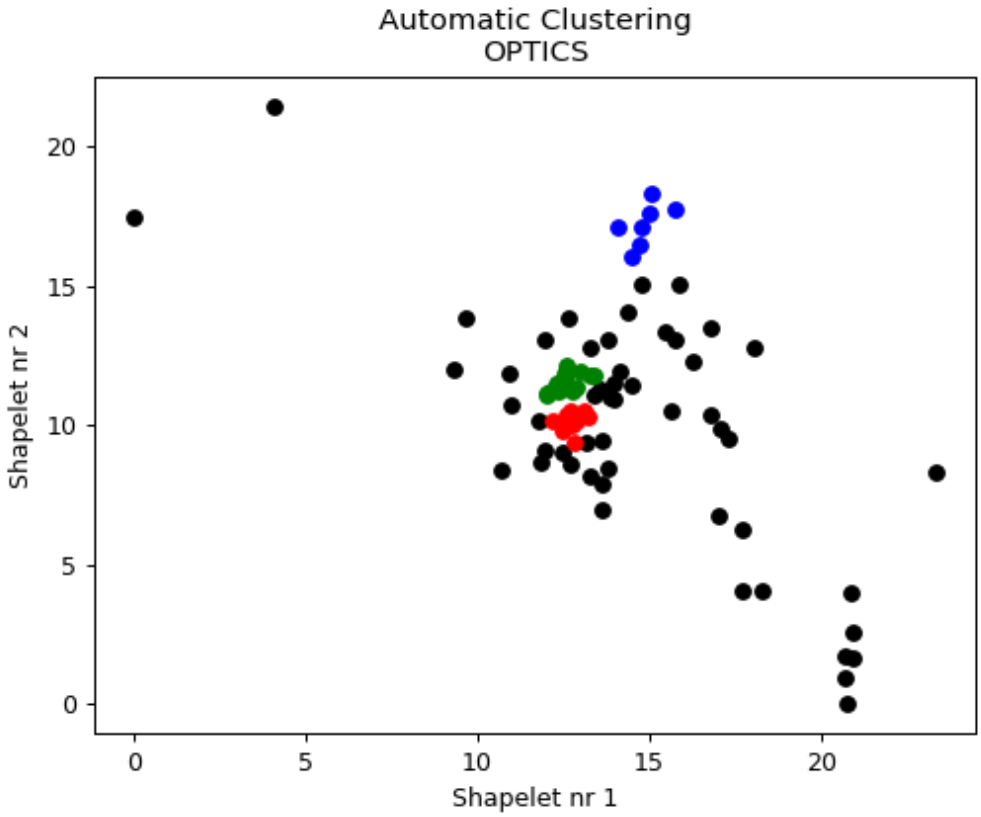


Figure 4.4: This figure shows the shapelet space created by the unfiltered shapelet tree algorithm with the colors denoting the clustering labels. Black denotes time series regarded as noise.

4.2 Shapelet extraction

In this section, the results of the shapelet extraction from the Volvo dataset is presented. Starting with the first extracted shapelet, shown in Figure 4.5. The shapelet separates the different classes in an acceptable way. However, there are some overlaps for all three classes potentially making it hard to separate the classes for the clustering algorithm in a later stage. The lack of clusters close to zero suggests that this shapelet does not represent a defining feature of any class. With these issues, it is not considered to be a good shapelet.

In spite of these issues, the shapelet does follow the properties of the scoring function, albeit not as expected. Tightness which has a weight of 70%, rewards a shapelet that does not spread the ground truth class intervals across the orderline, which the shapelet in Figure 4.5 does not. Overlap which has a weight of 20%, rewards a shapelet that contains as little overlap of ground truth class intervals as possible. Again, the shapelet in Figure 4.5 only contains a small fraction of overlap. The left-most tightness sub-measure, which has the lowest weight of 10% rewards a shapelet containing a tight leftmost ground truth class interval. However, the tightness is compared to the width of the orderline, not the other intervals, with this in mind,

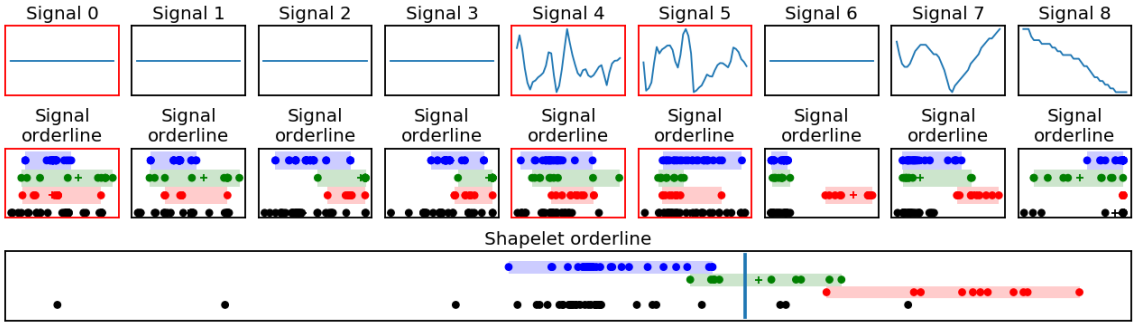


Figure 4.5: The first shapelet extracted by the shapelet tree algorithm. The top row shows the signals used in the shapelet, the second row shows the orderline created by each shapelet and the third row is the orderline of the shapelet. Signals with a red box indicates signals that has been filtered out by the signal filter and is not used to create the shapelet orderline.

the shapelet in Figure 4.5 does have a relatively tight cluster.

This shows that the shapelet does indeed follow the scoring as defined. It also reveals the innate problem of finding a general weighting of the sub-measures. Each dataset might require a tuned weighting to perform acceptably. A shapelet is regarded as good if it produces an orderline with a tight interval near zero and the rest of the intervals separated with no overlap. A tight interval close to zero would indicate a good distinctive feature being present in that class. The overlap limitation is used to find shapelets which not only separates a single interval, but several intervals on the orderline as this could potentially provide good clustering using fewer shapelets.

The second shapelet extracted, which is shown in Figure 4.6, has no overlap between the two existing ground truth classes. However, the blue class closest to zero is spread out across a large part of the orderline, again suggesting that this shapelet does not represent a defining feature of any class. This might not be a significant issue for clustering as the most important property is the classes being separated in such a way that they can easily be labeled with the same label.

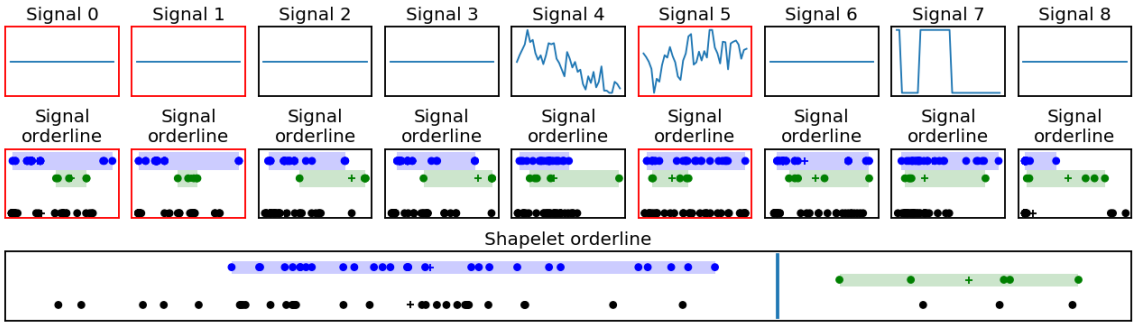


Figure 4.6: The second shapelet extracted by the shapelet tree algorithm, this is the left side of the split point in Figure 4.5.

The third and final shapelet extracted is shown in Figure 4.7. This shapelet separates the classes well, with the cluster to the left being tight and far away from the right

cluster. But even here the left cluster not close to zero, suggesting that the shapelet does not represent a distinctive feature.

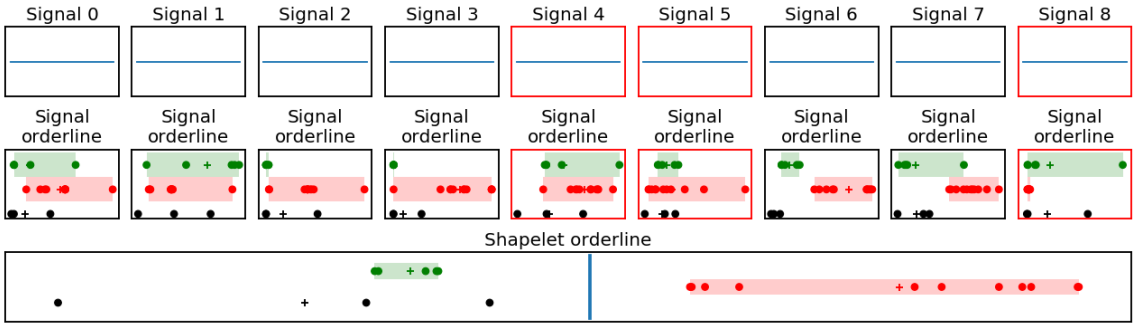


Figure 4.7: The second shapelet extracted by the shapelet tree algorithm, this is the right side of the split point in Figure 4.5.

When developing the shapelet algorithms, we expected that the shapelets would be able to represent distinctive features of each class. This would have been seen in the orderlines as one class interval being tight and placed close to zero. However, this was not the case for any of the shapelets presented in this section.

While we have no results showing why the algorithms have problems with representing distinctive features, we speculate that it has to do with the predefined fault classes. The faults exhibit similar, but not identical, patterns, with some signals being significantly different between cases. We suspect this from the class represented in blue in Figures 4.5 and 4.6. The definition of this class is general and can be presented as a group of faults. The two other classes appear to be well defined in the sense that they contain shared similarities across all time series. It is unexpected that the shapelet algorithm fails at finding distinct features for these. One possible solution could be to redefine the class labels when using shapelets, making the class definitions associated with a similarity, and then combine labels at a later stage. Another possible solution might be to use a different set of signals, potentially revealing important information we might have missed.

4.3 Clustering results

To annotate the dataset using the shapelets extracted, clustering is used. The OPTICS clustering algorithm, presented in Section 3.5, is used to perform the clustering of the shapelet space created by the distance vectors (represented by orderlines) of the extracted shapelets. The results are presented in Table 4.2 and analyzed later in this section.

The parameters used when running OPTICS are given by the default parameters of scikit learn, with the exception of $minPts$ which is set to 7. The clustering results are presented in Table 4.2. The cluster entropy of 0 indicates that the clusters are perfectly homogeneous with respect to the ground truth. The high class entropy

represents a high spread of ground truth classes into different clusters. However, this may not necessarily bad. With the previous insight of classes not representing similarities, we suspected that classes may be separated into several clusters and thus several clusters may represent the same class.

Table 4.2: Clustering scores when running the shapelet algorithms and OPTICS with the Volvo dataset

Cluster Entropy	Class Entropy	Silhouette Score
0.0000	0.4423	0.2391

To verify the reason as to why class entropy is high, the shapelet space is visually inspected. Luckily, the shapelet space becomes 3-dimensional as there are 3 shapelets extracted, allowing for a 3D plot. The space is visualized in Figure 4.8. The colors denote ground truth classes and in Figure 4.9, the colors denote the labels produced by clustering. Comparing the figures shows the red and green clusters from Figure 4.8 as being intact in Figure 4.9. However, the blue cluster becomes divided into three clusters along with a lot of unlabeled data. The division of the blue cluster into several clusters is the reason for the high class entropy, however, as previously discussed, this is not a problem as the clusters are still homogeneous.

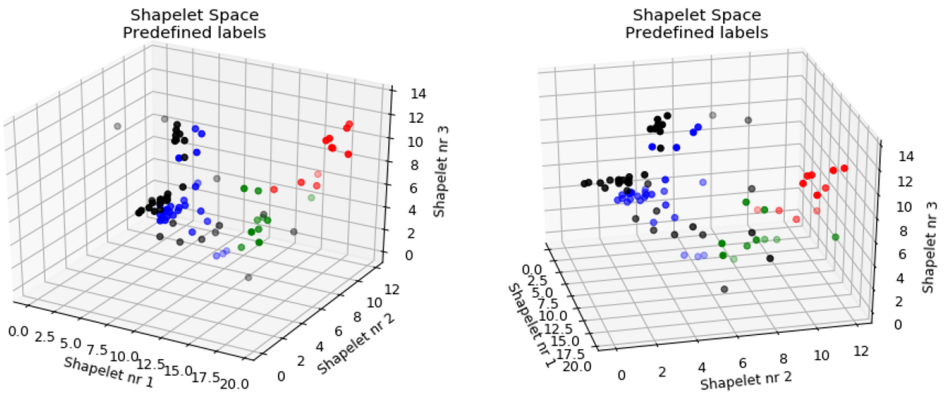


Figure 4.8: This figure shows two views of the shapelet space created by the shapelets. The red, green and blue points denote the time series with predefined labels and the black points denote time series with unknown labels.

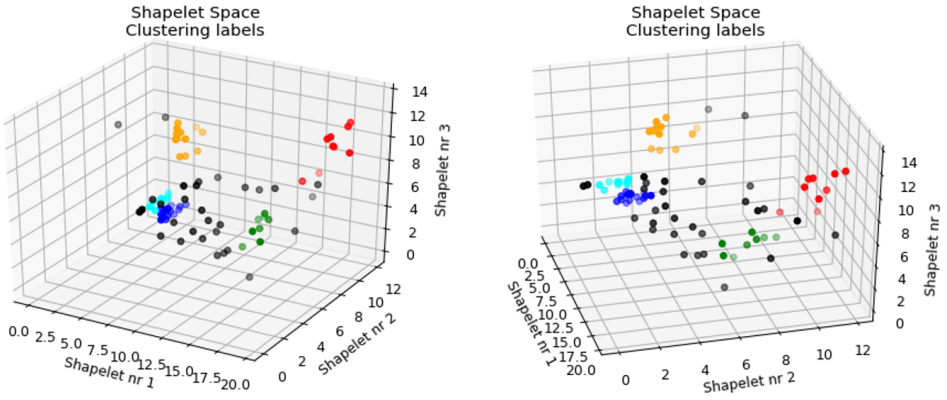


Figure 4.9: This figure shows the same two views of the same shapelet space as figure 4.8, but here the different colors denote the clustering labels assigned by OPTICS and black denotes points OPTICS regards as noise. The red and green colors labels seem to match the ground truth well, while the blue ground truth class has been split into three new clusters: blue, cyan and orange.

4.4 Deep Neural Networks

In this section the results from training the CNN and ResNet are presented. There will be 4 performance metrics in the figures below. The first one is loss, which should be as low as possible to indicate a good performance. The second one is Macro F1 score, which is the harmonic mean of the third and fourth metrics, precision and recall. For these, a high number indicates a good performance.

When training the networks the validation scores were quite unstable, an example of this is shown in Figure 4.10. We have not analyzed why this is happening since the analysis of the networks is out of scope for this master thesis.

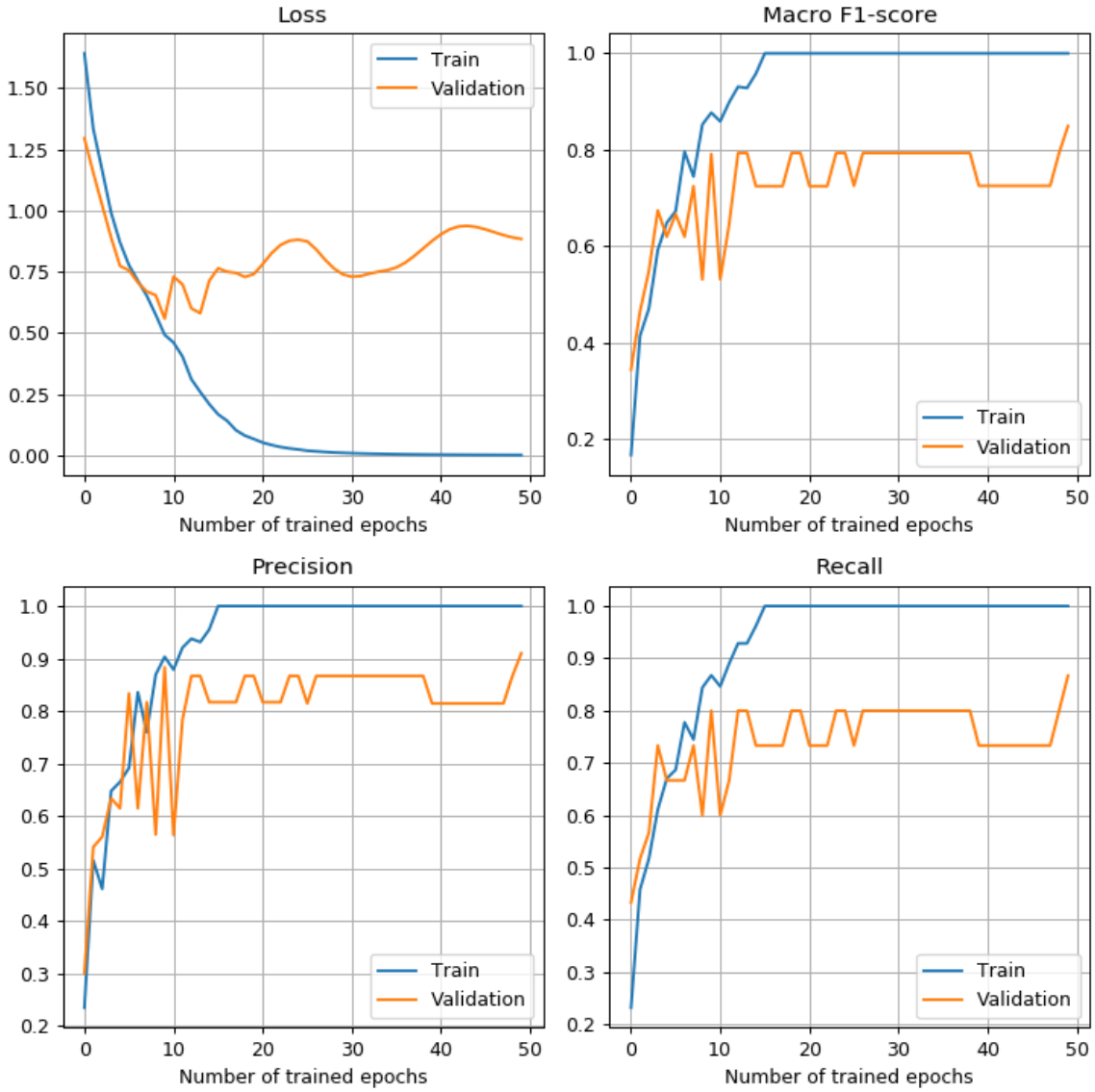


Figure 4.10: Example of the noisy nature of the training data. This plot is an extreme case and most of the time the training is more stable.

The noisy training data makes it difficult to know what performance can be expected from the networks. To get a better picture of what performance we can expect from the networks, the training scores are averaged over 50 training runs. The peak Macro F1 Score along with the peak precision and recall are shown in Table 4.3. The average training scores for the CNN and ResNet are shown in Figure 4.11 and Figure 4.12 respectively.

Table 4.3: Peak score of all three performance metrics for both the CNN and ResNet

Network	Peak Macro F1	Peak Precision	Peak Recall
CNN	0.858	0.913	0.810
ResNet	0.827	0.880	0.781

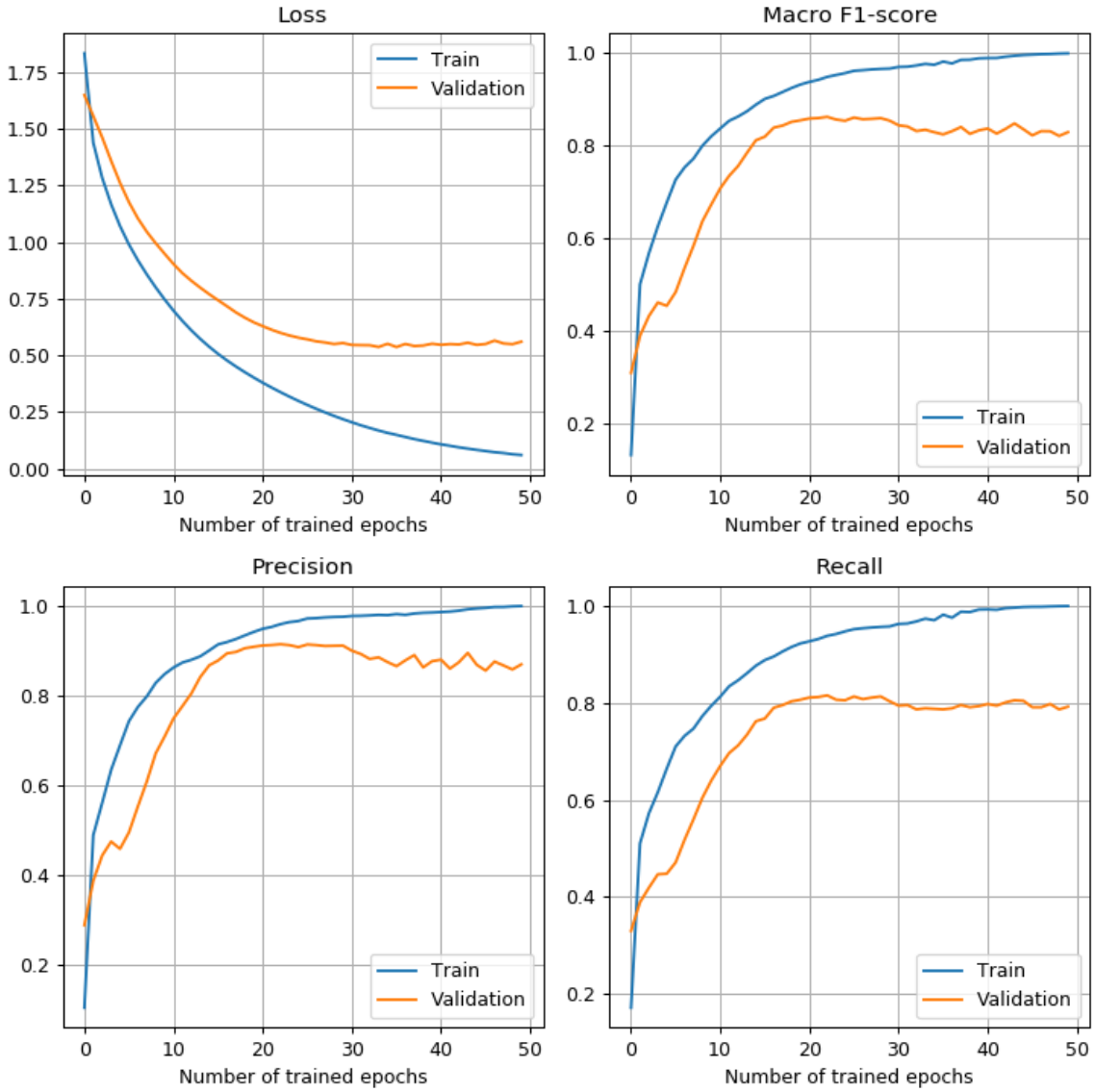


Figure 4.11: This figure shows that average training data over 50 runs when training the CNN with the Volvo dataset

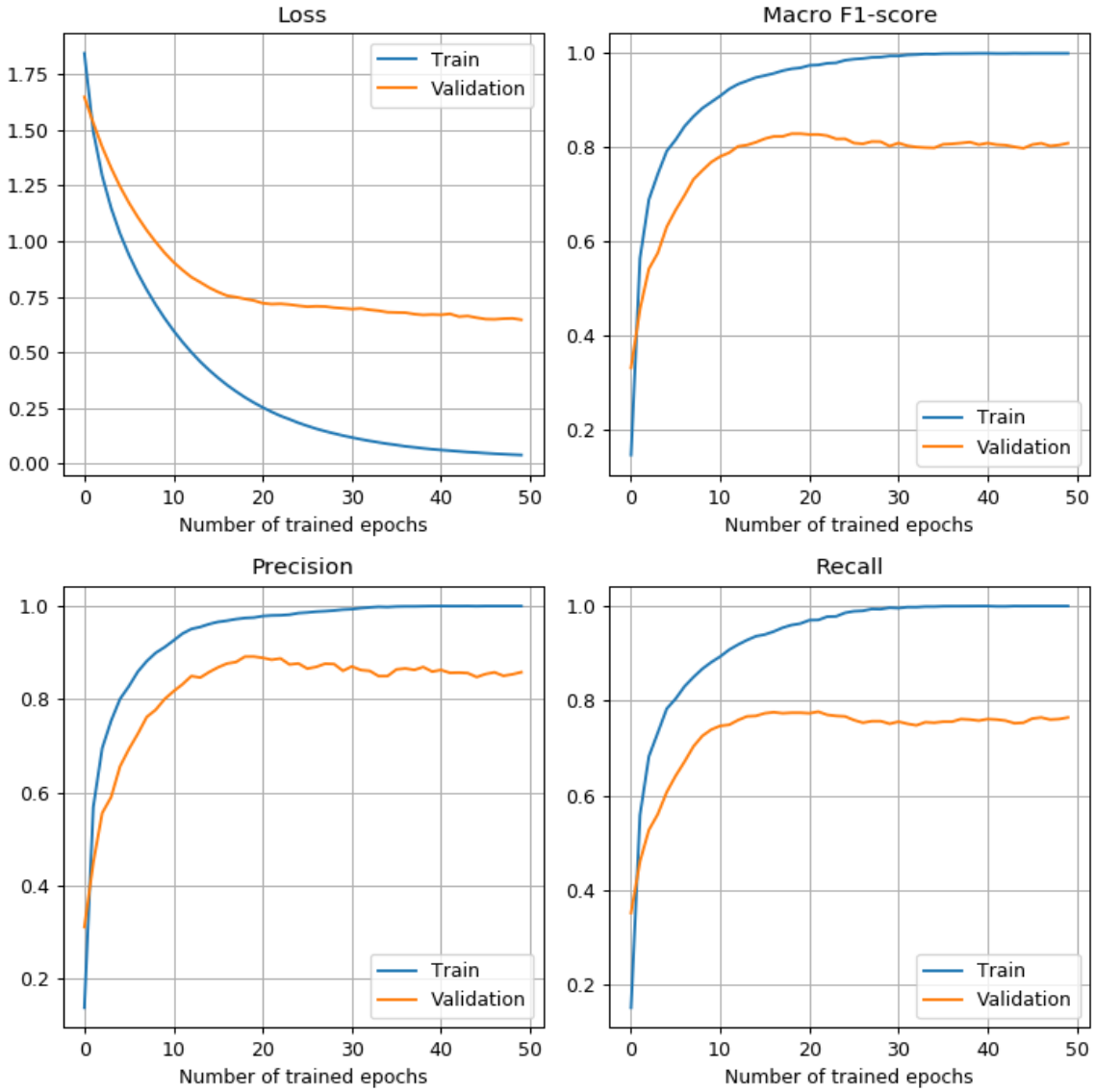


Figure 4.12: This figure shows that average training data over 50 runs when training the ResNet with the Volvo dataset

4.4.1 Final Network

In this section, the best CNN and ResNet are presented to give an indication of well the networks can perform at their best. The peak Macro F1 Score along with the corresponding precision and recall are shown in Table 4.4.

Table 4.4: Peak Macro F1 scores for the CNN and ResNet along with the corresponding precision and recall.

Network	Peak Macro F1	Precision	Recall
CNN	0.945	0.956	0.944
ResNet	0.895	0.930	0.885

The training performance for both the CNN and ResNet are shown in figures 4.13 and 4.14 respectively. The performance of the networks presented here in section 4.4 answers our second research question, DNNs achieve a useful level of precision and recall for classifying time series. The goal performance set in the introduction was 0.66 precision and 0.4 recall and this goal was achieved for both networks, with margin.

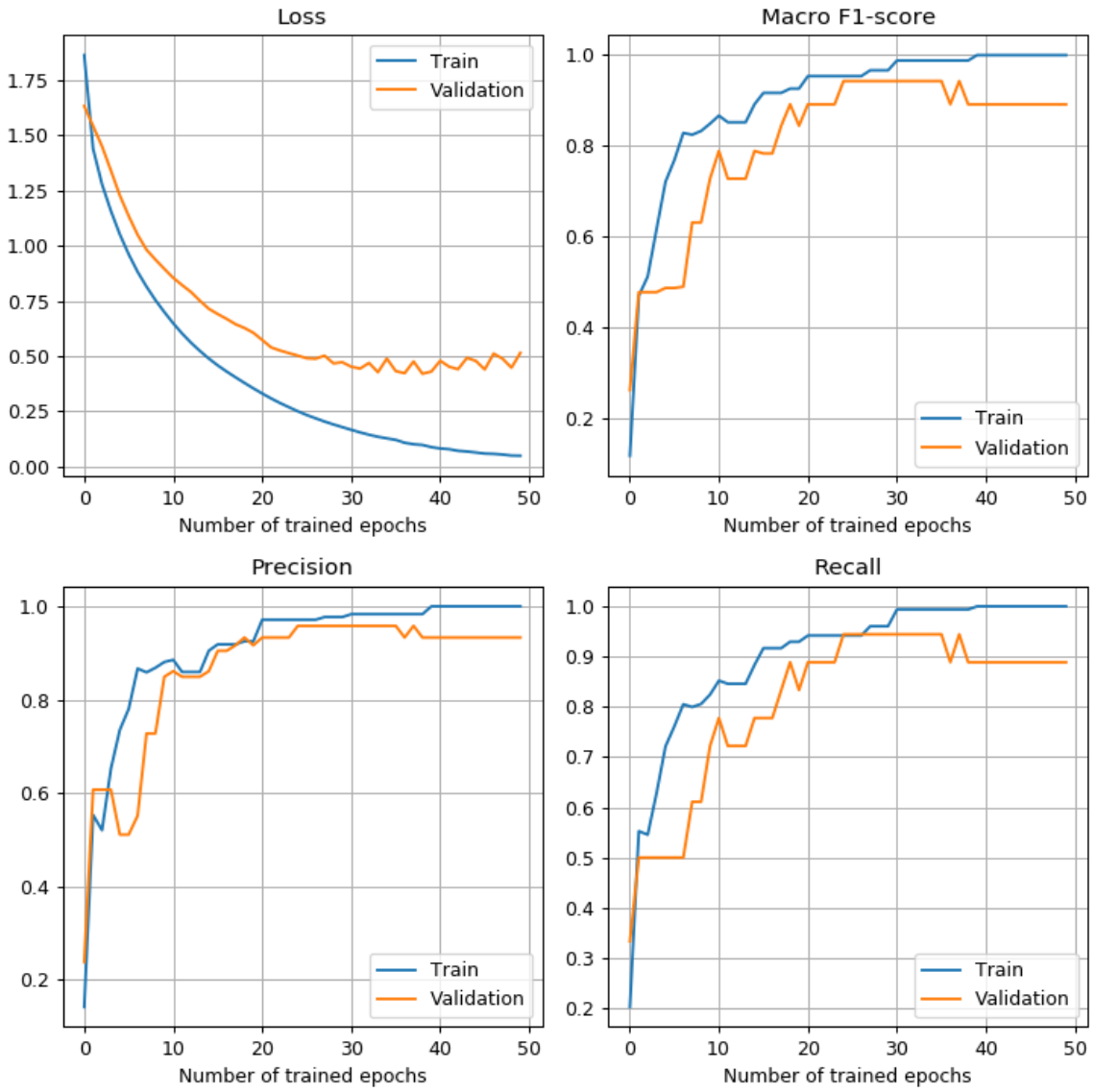


Figure 4.13: The best CNN network achieved when training.

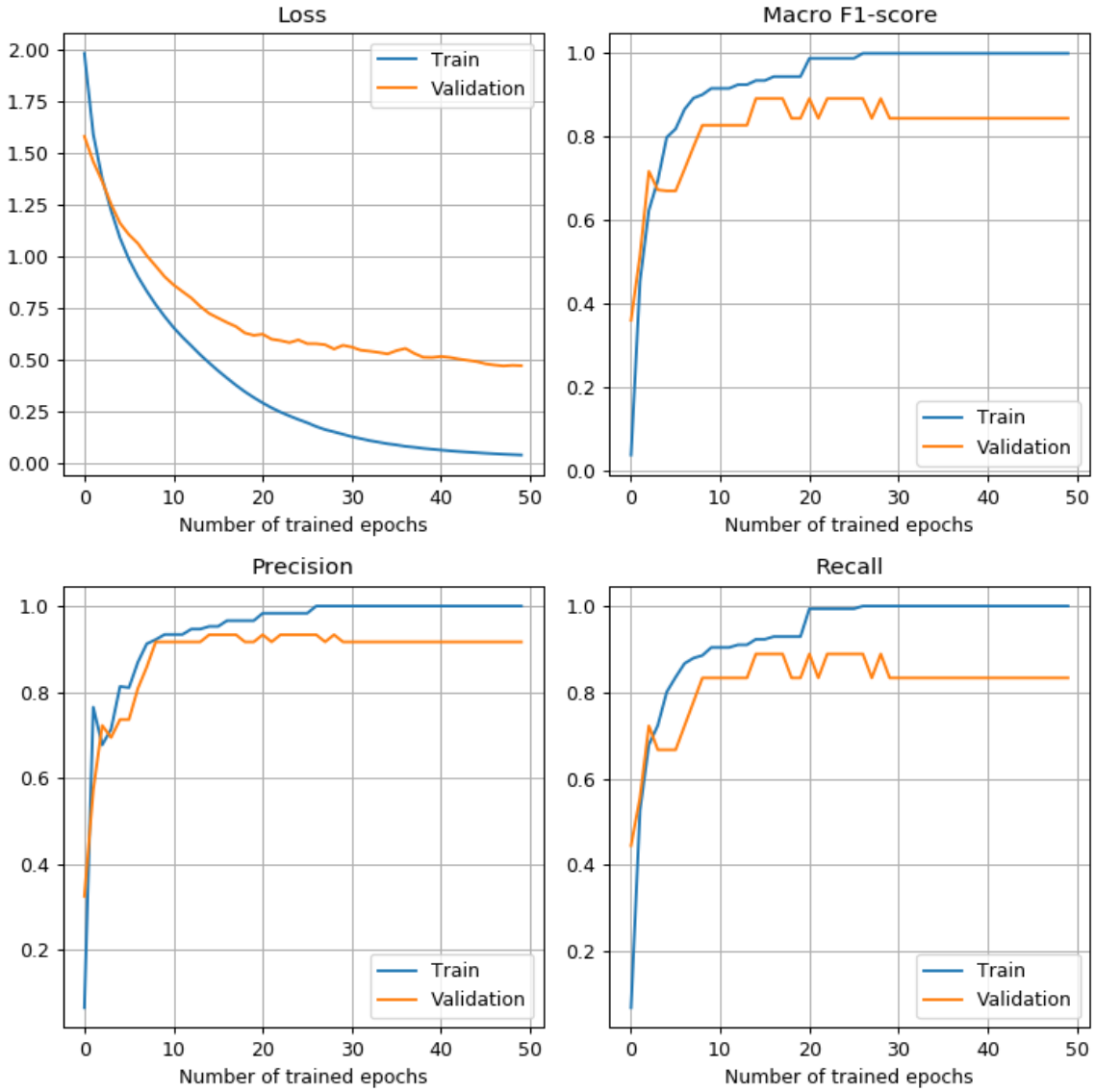


Figure 4.14: The best ResNet network achieved when training.

4.5 Verification

To demonstrate that the produced algorithm can be generalized to other datasets than the one provided by Volvo, the evaluation of other datasets is done as well. The first dataset is a constructed toy dataset. This dataset is built to demonstrate the properties of the algorithms and allows easy visualization of the intuition regarding the algorithm.

4.5.1 Toy example

This toy dataset example is trivial for the shapelet algorithms to work with and therefore produces close to perfect clustering results. The clustering scores are presented in Table 4.5. In Figure 4.15 the dataset classes are shown as reference for

the reader to look at when the shapelets are shown below.

Table 4.5: Clustering scores when running the shapelet algorithms and OPTICS with the toy dataset

Cluster Entropy	Class Entropy	Silhouette Score
0.0	0.038	0.611

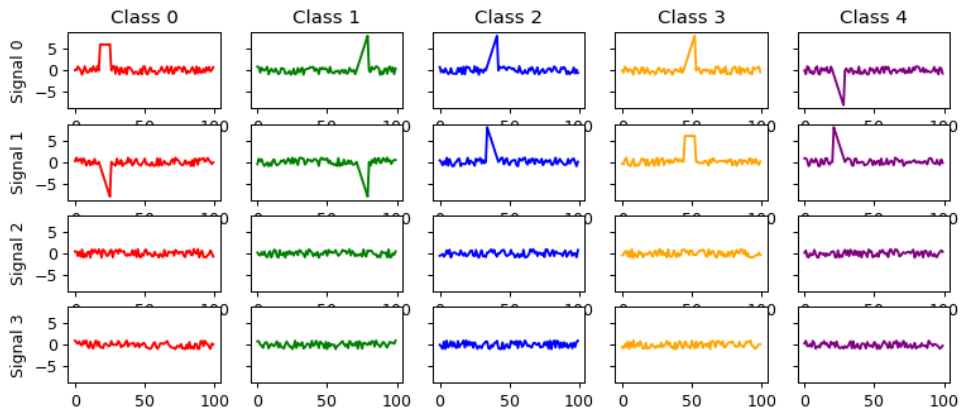


Figure 4.15: This figure shows an example time series from each class in the toy dataset.

While the toy dataset does not yield any comparative results regarding the performance of the algorithm, it does give the opportunity to verify that it behaves as designed. Starting with the first shapelet extracted in Figure 4.16, it selects the upward slope feature of class 2 (in blue) along with a trailing piece of noise. This enables it to separate out other classes. Class 1 (in green) and 3 (in orange) share the same slope. However, the trailing noise differentiates the classes from class 1. This can be seen as these classes lie closely on the orderline, not too far away from class 2 (in blue). Class 4 (in purple) has a downward slope and is thus less similar to the upward slope and is, as expected, placed far to the right.

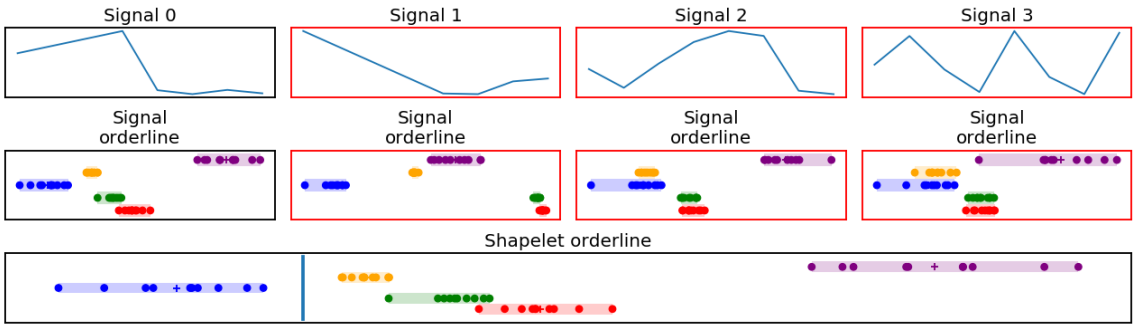


Figure 4.16: The first shapelet extracted when using the toy dataset

The second shapelet, shown in Figure 4.17 is the shapelet extracted from the data right of the split point in 4.16. It selects a sequence from class 1 (in green) where

signal 0 has an upward slope and signal 1 has a downward slope, allowing it to separate out the other classes.

Two additional shapelets are produced, but they exhibit the same behaviour as the two presented, and thus do not add any more information. Hence, they are omitted from the report.

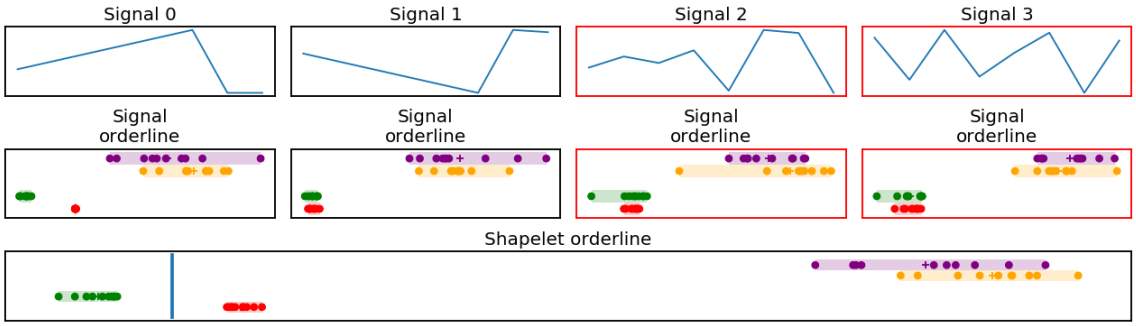


Figure 4.17: This shapelet is the second one extracted for the toy dataset and is the right branch of the shapelet tree.

4.5.2 External Datasets

The external datasets chosen were the ESR and GunPoint datasets. We repeat the properties of the datasets as they are very relevant for this analysis. The ESR dataset contains only signals which carry useful information for the separation of every class in the dataset and the GunPoint dataset contains only a single signal.

The ESR should intuitively be hampered by the signal filtering since all signals have relevant information, thus we lose information by filtering them. As the GunPoint dataset only contains a single signal, the signal filter is expected to not filter anything.

The analysis of ESR is in the following subsection, followed by the analysis of the GunPoint dataset in the next subsection.

4.5.2.1 Epileptic Seizure Recognition

The ESR dataset is a multiclass multivariate dataset, making it somewhat similar to the Volvo dataset which was the focus of the thesis. The main difference between the datasets is the correlation of signals to classes. The Volvo dataset consists of many signals where few correlate and these correlations differ by the class. The ESR dataset, on the other hand, consists only of signals correlating with all labels. This should make the filtering useless and instead the result reflects the performance gained by the multivariate distance measures and the adapted scoring.

The clustering results are presented in Table 4.7. These results are worse than we had expected. We expected the score to be much closer to the Volvo dataset scores, with at least low cluster entropy. The higher cluster entropy indicates that the

ground truth classes are mixed in the new clusters. This indicates that something with our methods does not work well with this dataset and there needs to be more research to find out why the algorithm performed worse than for the Volvo dataset. Unfortunately, due to time constraints, we did not analyze these results as much as we had intended.

Table 4.6: Clustering scores when running the shapelet algorithms and OPTICS with the ESR dataset with $minPts = 12$

Cluster Entropy	Class Entropy	Silhouette Score
0.1511	0.3866	0.1982

4.5.2.2 GunPoint

As previously stated, the GunPoint dataset is included as a comparative result towards previous implementations. The GunPoint dataset is not only univariate, it is also a binary classification problem containing only two labels.

Due to being a univariate dataset, the signal filter becomes an identity function since it has no signals to filter out. When applying the multivariate dimension computations on a univariate dataset, the operation becomes equivalent to the univariate measure from the original paper [9].

The changes remaining are the adapted scoring and the shapelet tree derivation. As the dataset consists of two classes, it should be separable by a single shapelet, possibly making the shapelet tree extraction algorithm equivalent to the unsupervised extraction algorithm.

However, when doing the clustering we use 6 shapelets. The clustering also mixes the two ground truth classes. These results were unexpected and show that our algorithms do not apply well to univariate datasets, such as the GunPoint dataset. The results from these external datasets show that more work is needed to generalize the algorithms.

Table 4.7: Clustering scores when running the shapelet algorithms and OPTICS with the Gun Point dataset with $minPts = 19$

Cluster Entropy	Class Entropy	Silhouette Score
0.2304	0.6864	0.2087

5

Conclusion

The purpose of this thesis was to investigate the possibility of creating a tool that could aid engineers in fault analysis of recordings from the system bus of trucks. The literature study concluded that classifying time series data such as the data on the bus is viably done using DNN trained with supervised methods. Two potential networks were extracted from existing research to act as test subjects. The best DNN was able to achieve a peak F1-score of 94.5%, showing that the algorithms developed in this thesis can be used to label a multivariate time series dataset in order to train DNNs. With this performance, we can also conclude that a DNN can be a useful tool to deliver a preliminary indication of what faults a report may contain and hence reduce the time engineers spend troubleshooting.

The main problems faced in this thesis was the unprocessed and annotation-less dataset provided. This resulted in the decision to create a semi-supervised annotation solution for time series data, mainly working with data from fault reports generated by truck drivers using beta software. The motivation for a semi-supervised approach is the required ability to control the annotations as to create relevant annotations for the engineers.

It was found that this is indeed possible. However, the development revealed issues with the predefined class labels not representing similarities found across all related time series. The concept developed during the thesis is based on using *shapelets* as a similarity measurement on the time series, extended with the potential to consider relevant combinations of time series in each recording as well as a semi-supervised approach to extracting shapelets. Since the concept is based on similarity, this requires recordings belonging to a specific label to share similarities distinct for the specific label. This was found not to be the case with the labels provided by the domain experts for annotation. This could potentially be solved by replacing the existing labels with new labels matching similarities present in all time series for each class. Alternatively, develop an algorithm capable of using several similarities for a single label without requiring each time series in each class to contain all similarities. As the domain experts were unavailable for the last half of the thesis, due to the Corona pandemic, changing labels was not possible. Investigating and developing a solution capable of using several similarities was not feasible within the time limits of the thesis. As such, the implementation and verification of the potential solutions are left as topics for future research.

The annotation solution was verified in part by training DNNs which has been shown by other researchers perform well on time series classification tasks. The solution was also verified by means of a biased dataset and external datasets. The implementation is compared to naiv̈e extensions of existing unsupervised solutions. The trained DNNs is considered proof that the tool can be implemented. Given the limitations of the missing similarities, the solution managed to achieve considerable performance. With that said, the step to move from research to a production-ready tool is to combine the pre-processing and conversion of recordings along with the trained DNN into a package usable by engineers.

Bibliography

[1] J. G. Rivard, "The electronic control unit for production electronic fuel injection systems," in *1976 Automotive Engineering Congress and Exposition*, SAE International, Feb, 1976.

[2] D. Huo, T. Ding, C. McMillan, and M. Gethers, "An empirical study of the effects of expert knowledge on bug reports," in *2014 IEEE International Conference on Software Maintenance and Evolution*, pp. 1–10, Sep. 2014.

[3] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, pp. 917–963, Jul 2019.

[4] N. D. Lane and P. Georgiev, "Can deep learning revolutionize mobile sensing?," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, (New York, NY, USA), p. 117–122, Association for Computing Machinery, 2015.

[5] E. Park, D. Kim, S. Kim, Y. Kim, G. Kim, S. Yoon, and S. Yoo, "Big/little deep neural network for ultra low power inference," in *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 124–132, 2015.

[6] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *SIGARCH Comput. Archit. News*, vol. 44, p. 243–254, June 2016.

[7] M. Längkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognition Letters*, vol. 42, pp. 11 – 24, 2014.

[8] A. J. Bagnall, A. Bostrom, J. Large, and J. Lines, "The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version," *CoRR*, vol. abs/1602.01711, 2016.

[9] L. Ye and E. Keogh, "Time series shapelets: a new primitive for data mining," pp. 947–956, 06 2009.

[10] J. Zakaria, A. Mueen, and E. Keogh, "Clustering time series using unsupervised-shapelets," in *2012 IEEE 12th International Conference on Data Mining*, pp. 785–794, Dec 2012.

[11] A. Bostrom and A. J. Bagnall, "A shapelet transform for multivariate time series classification," *CoRR*, vol. abs/1712.06428, 2017.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[13] "Reducing CO2 emissions from heavy-duty vehicles." [Online] Available: `https://ec.europa.eu/clima/policies/transport/vehicles/heavy_en`, [Accessed: 2020-05-21].

[14] "Volvo group annual and sustainability report 2019." [Online] Available: `https://www.volvogroup.com/content/dam/volvo/volvo-group/markets/global/en-en/investors/reports-and-presentations/annual-reports/annual-and-sustainability-report-2019.pdf`, 2020.

[15] "Hydrogen fuel cells." [Online] Available: `https://www.volvogroup.com/en-en/innovation/electromobility/fuel-cells.html`, [Accessed: 2020-05-21].

[16] OECD Expert Group on AI (AIGO), "OECD Principles on AI." [Online] Available: `https://www.oecd.org/going-digital/ai/principles/`, [Accessed: 2020-05-24].

[17] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[18] J. Zakaria, A. Mueen, E. Keogh, and N. Young, "Accelerating the discovery of unsupervised-shapelets," *Data Mining and Knowledge Discovery*, vol. 30, p. 243–281, Jan. 2016.

[19] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, p. 264–323, Sept. 1999.

[20] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc.Division of Simon and Schuster One Lake Street Upper Saddle River, NJUnited States, 1988.

[21] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters," *KDD-96*, pp. 226–231, Aug. 1996.

[22] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," *SIGMOD Rec.*, vol. 28, p. 49–60, June 1999.

[23] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering – a decade review," *Information Systems*, vol. 53, pp. 16 – 38, 2015.

[24] J. He, A.-H. Tan, C.-L. Tan, and S.-Y. Sung, "On quantitative evaluclustering and information retrievalinformation systems," vol. 11, pp. 105–133, 2004.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "2.3. clustering." [Online] Available: `https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation`, [Accessed: 2020-04-30].

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

[27] X. Glorot, A. Bordes, and Y. Benigo, "Deep sparse rectifier neural networks," vol. 14, pp. 315–323, 2011.

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

[29] "Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling," standard, International Organization for Standardization, Geneva, CH, Dec. 2015.

[30] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Comparing different clustering algorithms on toy datasets." [Online] Available: `https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html`, [Accessed: 2020-04-16].

[32] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 1578–1585, 2017.

[33] J. Villar, P. Vergara, M. Menéndez González, E. Marín, V. González, and J. Sedano, "Generalized models for the classification of abnormal movements in daily life and its applicability to epilepsy convulsion recognition," *International Journal of Neural Systems*, vol. 26, 04 2016.

[34] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML, "The UCR Time Series Classification Archive," October 2018. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.