Faculty of Engineering
Lund University
Mathematical Statistics

AI-based Classification of Radar Signals

Ludwig Hollmann Max Fors Joki

Supervisors: Andreas Jakobsson, Gunnar Hillerström

Abstract

Military vehicles typically send out radar signals in order to detect their surroundings. Using an Electromagnetic Support Measures receiver, these can be detected and it is of interest to be able to classify them in order to identify the vehicle and type of radar. Although there already exists multiple methods to do this, it is of interest to automatize and speed up the process of classification as much as possible. Artificial Neural Networks is a form of machine learning that has proven to be successful in classifying sequential data from a large variation of sources. The purpose of this thesis is therefore to investigate how well Artificial Neural Networks can classify different types of radar signals, when the data is given on a sequential form of arrival times from radar pulses. Both feed-forward and recurrent neural networks of different types are considered and in addition to this, methods to apply them on the specific radar data are developed. The results show that artificial neural networks are capable of classifying radar signals of this form with a precision of up to 98 percent. In addition to this it could also be done with a couple of seconds worth of data using relatively simple models.

Keywords: Neural networks, TensorFlow, Keras, Radar signals

Acknowledgements

First of all, we would like to thank our supervisor at FOI, Gunnar Hillerström, for his guidance throughout the project.

We would also like to send our thanks to the members of the unit *Radar Electronic Warfare Systems* at FOI, for their helpful comments along the way.

We are also grateful that FOI provided us with all the needed resources to carry out the project.

Finally we would also like to thank our supervisor from Lund University, Andreas Jakobsson, for assisting us during this project.

Acronyms

ANN Artificial Neural Network

ESM Electronic Support Measures

LSTM Long Short-Term Memory

MLP Multilayer Perceptron

RNN Recurrent Neural Network

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Problem formulation . . . 1
- 1.3 Data set . . . 2
- 1.4 Plan and structure . . . 3

2 Theory **5**
- 2.1 Radar signals . . . 5
  - 2.1.1 Visualization and representation . . . 5
- 2.2 Artificial Neural Networks . . . 6
  - 2.2.1 Feed-Forward Neural Networks . . . 7
  - 2.2.2 Recurrent Neural Networks . . . 8
  - 2.2.3 One-hot encoding . . . 9
  - 2.2.4 Overfitting . . . 9
  - 2.2.5 Keras . . . 10
- 2.3 The k-nearest neighbors classifier . . . 10
- 2.4 FLD ratio . . . 11

3 Preprocessing and re-classification of data **13**
- 3.1 Examination of the initial division . . . 13
- 3.2 Method for Reclassification . . . 17
  - 3.2.1 Comparison of two observations . . . 17
  - 3.2.2 Merging the observations . . . 18
  - 3.2.3 Removing outliers . . . 19
  - 3.2.4 Manually sorting . . . 19
- 3.3 Performance metrics . . . 20
- 3.4 Results . . . 20
  - 3.4.1 Trash data . . . 22
- 3.5 Discussion . . . 23

4 Neural Networks **25**
- 4.1 Pre-processing and implementation . . . 25
  - 4.1.1 Test, train, validation split . . . 25
  - 4.1.2 Slicing and the windowing method . . . 25
  - 4.1.3 Constructing the data sets . . . 27
- 4.2 Comparing the networks: feed-forward and recurrent models . . . 27
  - 4.2.1 Results . . . 29
  - 4.2.2 Discussion . . . 29
- 4.3 Evaluation on sub-sequences of varying lengths . . . 30
  - 4.3.1 Results . . . 31
  - 4.3.2 Discussion . . . 32

5 Applications on whole observations **35**
5.1 Variation of sub-sequence sizes between observations . . . 36
5.2 Method . . . 36
5.2.1 Motivation and explanation . . . 36
5.2.2 Simple Method . . . 37
5.2.3 Majority layer . . . 39
5.2.4 Expanding to varying subsequence size . . . 41
5.3 Results . . . 41
5.3.1 Performance of the different models . . . 41
5.3.2 Discussion . . . 44
5.4 Handling the trash observations . . . 46

6 Discussion and Conclusion **49**
6.1 Future Work . . . 50
Bibliography . . . 50

Appendices **53**

Appendix A Loss and accuracy for MLP model **53**

Appendix B Loss and accuracy for LSTM model **55**

Chapter 1

Introduction

1.1 Background

Radar is a type of detection system that uses pulses of radio waves in order to determine location, angle and velocity of objects relative the source of the radar. It is commonly used in airplanes, ships, meteorology, geology and many other fields. Typically, a radar consist of a transmitter that sends out radar signals and a receiver that registers them as they return. Because of different applications, radar signals vary in their characteristics such as frequency, amplitude and time between pulses. This means that it is possible to detect and differentiate between different types of radar signals if one can measure these characteristics, allowing classification of the transmitter of radar signals in turn. This is of particular use in the military, where it is of interest to determine what type of planes, ships etc that one is dealing with. [20] [14]

A common way to classify types of radar is to only look at the times between pulses. Radar transmitters usually send out pulses with varying time intervals according to some pre-decided sequence. By having an Electronic Support Measures(ESM) receiver, measuring when the radar pulses arrive, this sequence can be decoded and the type of radar classified. However, this is usually problematized by the transmitter rotating in order to send out radar signals in every direction. This means that the sequence is regularly interrupted and the ESM receiver rarely gets the full sequence in one transmission. Instead, it usually receives parts, starting and ending at unknown positions, of the whole sequence. The task of classification therefore consists of using a combination of these parts to match them with sequences of known radar types. [20] [14]

Artificial neural networks is a type of machine learning that has yielded impressive results during the last years, in a large variety of different fields. In particular, it has performed very well on classification of sequences such as speech recognition, translation and decision making for automatic vehicles. [22]

1.2 Problem formulation

The purpose of this thesis is to determine if it is possible to classify different types of radar signals using Artificial Neural Networks. More precisely, given data on the form of a sequence of arrival times of radar pulses, can a neural network be constructed that can take this data as input and make a classification of the type of radar emitting said pulses? Furthermore, if this can be done, with what level of precision and speed? Also, if a model can be created that can perform this task, it should be investigated how it behaves for sequences of damaged and problematic data.

1.3 Data set

Given data consisted of measured arrival times of radar pulses at an ESM receiver from a number of different observations of ten minutes each. More precisely, in each measurement an ESM receiver had been switched on for ten minutes and listened for signals and each time a pulse was received, the arrival time was registered. Data consisted only of these arrival times and no information was available about amplitude, frequency etc. The reason for this was that the recorded pulses were assumed to come from transmitters of such a varying nature that a division into more roughly different radar types was sufficient for the purpose of the task, and for this division the time intervals between radar pulses were sufficient.

In addition to this, a first sorting algorithm had roughly sorted data on 17 classes that had been limited to 1,000 observations each. There were thus a total of 17,000 observations of the form described in the previous paragraph. This division was only a rough first division and a first step. More precisely, the data was on the form of 17 MATLAB data files with 1,000 cells in each where each cell consisted of an observation representing a ten minute measurement. These observations were of varying size but averaged around 2,000-3,000 arrival times per observation, regardless of class. This is shown in Figure 1.1.

Finally, it was assumed that the majority of observations consisted only of receptions from one type of radar, i.e. there were no regular disturbances or several radar transmitters that were measured simultaneously, but as a rule the ESM receiver only listens to one transmitter during a ten minute interception. The observations that still contained this would be considered rubbish and outliers. This means that the entire ten minute measurement can be classified as one class and does not need to be divided into several different classes.

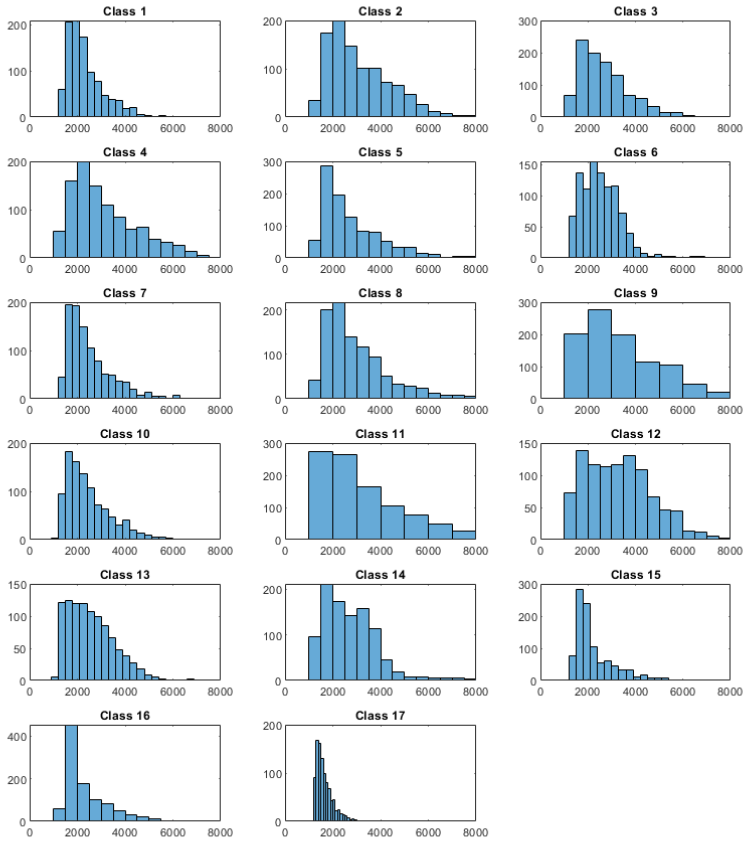


Figure 1.1: Histograms of sizes of the observations for each of the given 17 classes

1.4 Plan and structure

This problem differs from a typical sequence-classification problem in the sense that one deals with sequences having frequent and both regular and irregular interruptions. A large part of the problem is therefore to inspect and get around this problem. Furthermore, as was mentioned in the section before, the data set is assumed to be in need of a lot of preprocessing and some reclassification, further division and resorting of the given classes. In order to deal with these two issues, the report is divided into three broad parts. The first is concerned around preprocessing, where the given data is analyzed and treated. The goal here is to create a proper and clean division of the data into distinct classes in order to train as good of a neural network as possible. This part includes visualizing the given data, inspecting the issues with it, cleaning out outliers and developing methods and algorithms for obtaining a new partition of data. Optimally one wants to eliminate the human factor in this part and therefore a large focus is put on objective measures for dividing radar data into different classes.

In Chapters 4 and 5, methods are developed to deal with the issue of the interruptions in the sequences. In short, the broad problem of constructing a network that can classify an entire sequence of radar data is split up into smaller parts. Neural Networks will then be constructed, trained, tested and evaluated for these smaller problems. Finally, the chosen neural network models will be put back into the larger issue of classifying an entire observation. In addition to this, the ability to classify data marked as *trash* along the way will also be studied.

Chapter 2

Theory

2.1 Radar signals

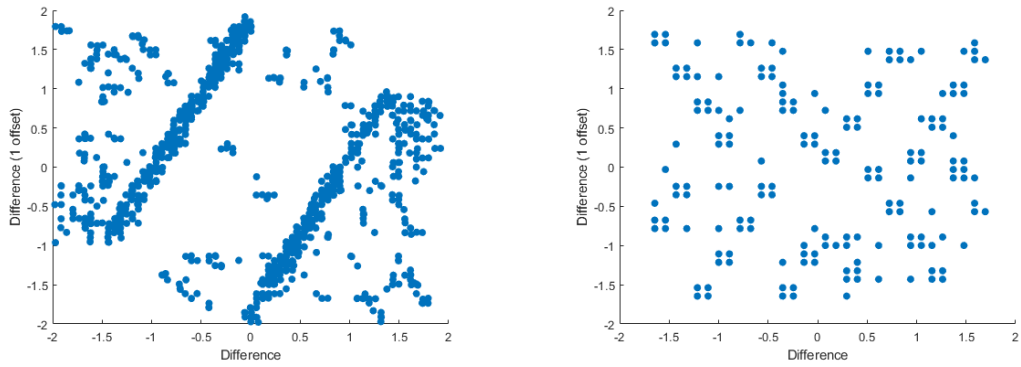
The purpose of a radar is to identify and detect objects in the surroundings. It emits electromagnetic pulses that collide with surrounding objects and is reflected back to a receiver collocated with the transmitter, which can then determine the object's position by measuring the time between the pulse being sent out and it returning.

Different types of radar differ depending on the time intervals between these pulses. A radar sends out pulses at intervals according to a sequence that defines just that type of radar. This enables an ESM receiver to determine the type of transmitter it is dealing with by identifying this sequence. In addition to this the sequence will also depend on which mode the transmitter is. The radar switches modes based on the distance it wants to detect objects. For longer distances the transmitter must take longer time between the pulses in order to avoid interference. However for the type of radar studied in this thesis, the underlying pattern remains the same for each mode, just scaled by a factor depending on the mode the radar is in. [20] [14]

2.1.1 Visualization and representation

As mentioned, a radar transmitter is defined by a sequence of time intervals between its pulses. As the transmitters rotate and are turned away from the ESM receiver, not all pulses will be registered. This results in only parts of the sequences being recorded and that they are interrupted with both regular and irregular intervals, and are resumed at another location in the sequence. Where these interruptions occur also varies depending on the distance between transmitter and receiver, which means that even sequences belonging to the same type of radar will not look the same.

All this means that it is not possible to represent this type of data as traditional time series where the same index in two different observations corresponds to each other. Instead, this type of radar is usually represented in a different way. First, the arrival times are converted into differences, because it is the intervals between the pulses being sent out that define the type of transmitter. Then the values are normalized in order to ensure that observations from the same radar type but in different modes look the same. Then the values that are abnormally large, corresponding to the time when the transmitter is turned away from the ESM receiver, and values that are 1.25 or 0.75 of the median, corresponding to missed or double registered pulses, are removed. Finally, this new difference vector is plotted against itself shifted one step in time. Two examples of this are demonstrated in Figure 2.1 where two whole ten minute sequences from two different classes are visualized according to the described method [13]



(a) The first observation from class 2

(b) The first observation from class 13

Figure 2.1: Visualization of two radar signals from different classes. The entire ten minute sequence is presented.

In Figure 2.1, it can be seen that the two different types of transmitters result in two distinct patterns. The reason why this type of visualization works well is that it does not place the same importance on order as a regular representation. As long as the same time interval occurs one after the other in some place in two different measurements, the same point will be plotted. This means that two sequences that are the same will form approximately the same images, regardless of when the interruptions occur, while measurements that are different will also usually form different images. There may be measurements that are different but still give rise to similar images as the same interval occurs at different places in the sequences but this is assumed to be unusual for a division between widely different classes which is the purpose of this work. All this means that a method for determining whether two observations should belong to the same class is to visualize them and pair them if they give rise to similar patterns. This is a traditional method used in radar classification. [13]

An alternative way of handling this type of radar data is to keep the original sequence of normalized differences but instead of creating the time-shifted plot, keeping the sequence as it is. This allows for more than two dimensions to be considered but also makes it complicated to visualize and handle for a human being or simpler algorithms. However, such sequential data could potentially be handled by more complex algorithms.

2.2 Artificial Neural Networks

Artificial neural networks (ANN) are a collection of machine learning algorithms that are vaguely inspired by how biological neurons in the animal brain work [3]. They have seen a big boost in popularity during the last decade as computations have become more efficient and have managed to solve a large scope of problems. In very broad terms, ANNs utilizes a large amount of nodes (called artificial neurons) that are connected to each other in a network of connections. How these neurons look vary from network to network but what generally holds is that the network has both adjustable weights and some sort of activation function. The weights represent linearity and the activation function non-linearity. This allows the network to combine both linearity and non-linearity in order to fit complex pattern. [29]

The network takes an input of some form and it is allowed to propagate through the nodes and connections and an output is produced. The purpose is then to adjust the weights in order to produce as good output as possible based on the input. This is usually done in the manner of supervised learning, that is, by feeding the network a large amount of training data, consisting of different types of input and a matching output. Using different types of algorithms and methods, the weights are then adjusted and tuned as it learns to recognize different patterns in the input in order to match the correct output. [28]

There are many different types of networks used for different types of tasks. In this thesis, we will focus on feed-forward networks as well as recurrent neural networks, both commonly used for classification. Feed-forward networks is the most common type of networks and traditionally used for classification [24] while Recurrent Neural Networks are networks designed to handle sequential type of data.[25] Below follows some brief summaries of Feed-forward and Recurrent neural networks, for a further explanation consult Deep Learning with Python [4].

2.2.1 Feed-Forward Neural Networks

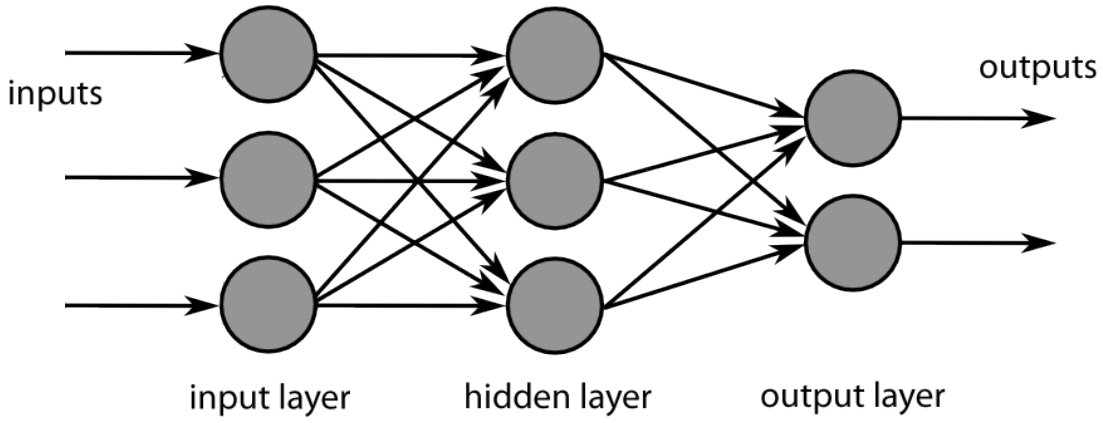


Figure 2.2: A visualization of a possible feed forward neural network architecture

Source: Wikimedia

A feed-forward neural network is a neural network where no connection forms a cycle. [29] In the most simple form it consists of a so called single layer perceptron. This perceptron takes the input nodes, multiplies each with a weight, sums it all together and sends it through an activation function that outputs a value based on the summed input. There are different types of activation functions that can be used depending on the traits of the network the designer wants. However what these activation functions have in common is that they introduce non-linearity to the model. [16] The weights can then be updated through training with labeled data, using gradient descent and a loss function[9] [21].

In practice a single-layer perceptron isn't capable of handling complex problem. Therefore the multilayer perceptron (MLP) is introduced. Henceforth the feed-forward networks used will be referred to as MLP networks. This functions in a similar manner to the single-layer perceptron but there are multiple layers put after each other. This means that the activation function sends its output as input to the next layer and the patterns is repeated until the final layer. Usually there are also more than one node in each layer, resulting in a full grid of nodes. How this can look in practice is visualized in Figure 2.2. The network is called feed-forward because each node is only connected to nodes in the layer right after it, meaning there are no connections backwards or between nodes in the same layer, nor are there any skipping a layer. Usually a feed-forward neural network is also fully connected which means each node is connected to every node in the layer in front of it. [15]

In order to deal with the large amount of weights in a fully-connected MLP network, a method called backpropagation is used to calculate the gradient with respect to the loss function. In very broad terms backpropagation utilizes dynamic programming to calculate the partial derivatives and then multiply them together according to the chain rule. [12] In practice pure backpropagation is rarely used but instead regularization algorithms such as ADAM [19] that build on backpropaga-

tion are used. They have the added benefit of making the weight calculation a bit more stable and reliable.

When used in practice, the network functions so that each node in the input layer represents a feature of the data that is to be handled by the layer and each output node represents a dimension in the output space. Typically feed-forward neural networks handle either classification problems where the output is a class or a regression problem where the output is a value. Some common hyperparameters that have to be adjusted during training are a regularization algorithm, loss and activation functions, the number of hidden layers and the number of nodes within the hidden layers. In addition to this the amount of epochs the network is trained for has to be considered both in order to deal with overfitting [26] as well as precision. [30]

2.2.2 Recurrent Neural Networks

A recurrent neural network (RNN) is a special type of ANN tailored towards handling sequential data. The main difference between a recurrent and a MLP network is that connections within a recurrent neural networks forms a directed graph along a time sequence. This allows the network to handle dynamic behaviour in a way a MLP network cannot do. In short terms this means that a sequence can be given as input where the output from each node is used as new input together with the next part of the sequence. A way to visualize it is as a MLP network with very sparse connections where each timestep is a layer. [23] This network becomes many layers deep quite quickly which introduces some issues with a vanishing gradient, that is that the multiples during gradient descent goes towards zero very quickly and it is hard to update the weights. In order to deal with this the concept of a Long short-term Memory (LSTM) RNN is introduced. A visualization of a LSTM network is shown in Figure 2.3.

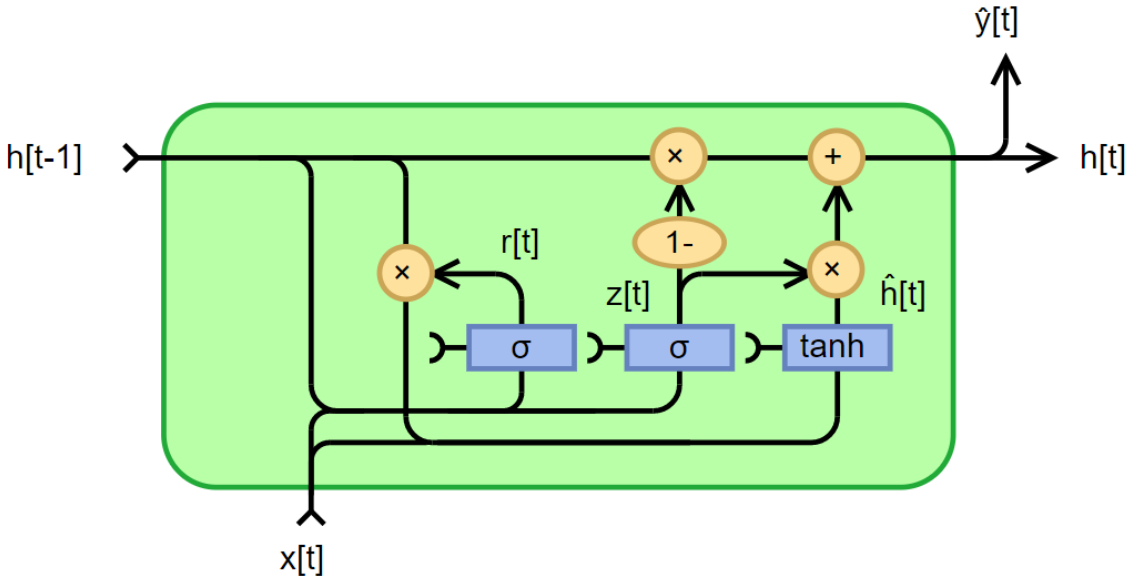


Figure 2.3: A visualization of a LSTM recurrent neural network

Source: Wikimedia

The specific details of the network shown in Figure 2.3 are out of the scope of this report. It is sufficient to point out that the architecture has different gates allowing it to remember and forget features as one goes deeper into the sequence. This allows for easier handling of much longer sequences of data. The number of layers is a hyperparameter that allows for detection of more complicated structures, however it should be noted that using more layers increases the complexity of the model by a large amount. The number of nodes represents the number of patterns the network can keep stored. For a LSTM network the activation function can not be chosen. [10]

2.2.3 One-hot encoding

Typically when dealing with classification problems that have more than one target type, a technique called one-hot-encoding is used. It works by having the target and output being a vector of size N where N is the amount of potential types the input can be classified as. The target vector has a 1 at the position of the correct class and 0 at the other positions. The output vector is also of size N and consists of estimated probabilities for each possible class. In order to have the sum of the probabilities being between 0 and 1, a softmax activation function is used in the last layer [11]. In order to do this when classifying sequential data using LSTM, a dense layer must therefore be added at the end. [2]

2.2.4 Overfitting

A very common problem in neural networks is overfitting, meaning that the model becomes very good at fitting the data from the training set but do worse on unseen data from the same distribution. The cause of this is that the network learns to perfectly fit the training data but in the process also includes natural noise or randomness and therefore misses the underlying real trend. [6] An example of how overfitting can look is visualized in Figure 2.4. Here the model has been trained to perfectly separate the two classes in the data set, shown in green. However it is very unlikely that this is a natural separation which rather should look more similar to the black curve.

Usually one has to introduce measures to deal with the problem of overfitting in order to get as good of a performance on unseen data as possible. A typical way to catch overfitting is to separate the data set into training and validation data. The model is then trained on the training data and with regular intervals tested on the validation data. Since no training has been done with the validation data, the model can not be perfectly fit to it and generalization is kept. According to the Universal approximation theorem [5], the accuracy of the model on the training data should go towards 100% as the model is trained longer and learns to perfectly fit the data. For the validation data the accuracy starts going down when generalization is lost. Therefore it is possible to plot the accuracy for the training and validation data next to each other and once the validation accuracy starts getting lower, once can terminate the training. This method is called early stopping and is a powerful way to handle overfitting [26]. Another possible method is to force generality into the model by introducing some regularization method that punishes large weights, as large weights contributes to fitting strange patterns [26]. One can also introduce a dropout layer [17], which removes a random subset of the nodes in each round of training. This forces the model to not specialize into overly strange patterns and keep generality.

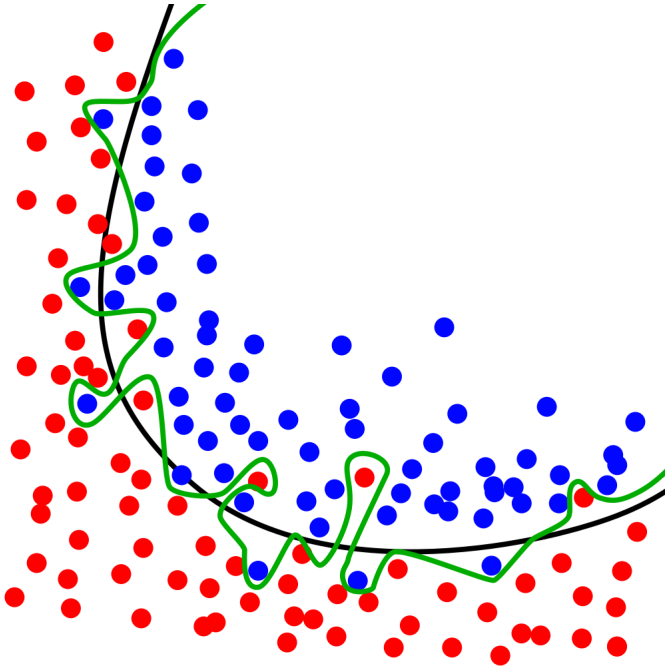


Figure 2.4: A visualization of overfitting during a classification problem in a 2D-space with 2 classes

Source: Wikimedia

2.2.5 Keras

Keras is a open source software library that works as a Python interface for artificial intelligence. It is an interface for the TensorFlow library that allows the user to implement deep neural networks. [18]

2.3 The k-nearest neighbors classifier

When considering the observations using the 2D-perspective, it was of interest to be able to classify an unlabeled sample. One of the simplest approaches to do this is to use the k-NN classifier. The classifier works by taking an unlabeled vector in an n-dimensional space - in our case $n = 2$ - and then considering the k nearest points, to classify the sample. The variable k is set manually and usually depends on the problem. Figure 2.5 provides a great illustration of how the algorithm works. In the figure, we have two known labels; blue and red 2D-points. In the training phase, the points are simply stored in the model, to be used later during the classification. When an unlabeled point is to be classified, the distances to all points in the training set are computed and sorted. The k smallest distances are then considered. As can be seen in Figure 2.5, when $k = 3$, the majority of the three closest points are red, leading to the green unlabeled point to be classified as red. Letting $k = 5$ however, leads to the point being classified as blue instead. [1] [8]

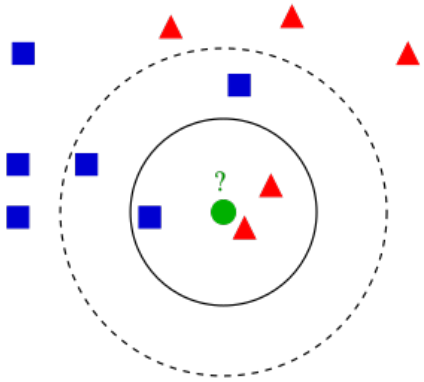


Figure 2.5: Example of k-NN classification. The green point is to be classified.

Source: Wikimedia

2.4 FLD ratio

To determine how well a classification has been done when data labels are missing, one method may be to consider the variance between classes and the variance within classes. The goal for a good division will then be to maximize the former and minimize the latter. If one can in any way quantify each element, for example by projecting it on a line, then it is possible to use the Fisher Discriminant Ratio, described below for two classes.

$$FDR = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_1^2} \tag{2.1}$$

where μ_i and σ_i are the mean and the standard deviation of the elements in class i, respectively. The value increases the greater the difference in the mean values and the smaller the variances are, which was the goal described above. This ratio can also be generalized to more than two classes if the classes are assumed to be independent. It is then written on the form

$$FDR = \frac{1}{N} \sum_{i=1}^{N} w_i \sum_{j=1}^{N} \frac{(\mu_i - \mu_j)^2}{\sigma_i^2 + \sigma_j^2} \tag{2.2}$$

$$\sum_i w_i = 1$$

where N is the number of classes and each class is weighted according to the proportion of all samples belonging to the class. [7] [27]

Chapter 3

Preprocessing and re-classification of data

The original 17 classes were the result of a quick K-means algorithm applied on unsorted data. It was suspected to contain a lot of misclassified observations, mixed up classes and classes that should be merged. The first thing that was done was therefore some visualization and examination of the given classes in order to determine precisely which issues that existed. After this was done, a method was developed for reclassification of the data.

In the developed method, each observation is seen as unlabeled to begin with and the original classification disregarded. However, the original 17 classes were still seen as a reference point for how precise the classification should be. Largely similar differences as were shown there is what we wanted in our final division of the data, thus it was assumed that there were a total of around 15-25 classes and not many more or less based on inspecting the data set manually. The purpose was then to develop a method that was consistent in how it divided classes. Just looking at each observation and doing it by hand was to be avoided as that introduced an unwanted bias. In short, a class was to be split or two classes were to be merged so that the overall difference between classes were roughly similar and this similarity was of the degree of precision introduced in the original data set.

3.1 Examination of the initial division

In order to confirm the suspicion, the initial 17 classes were visualized and examined. The entire first observation from each class was plotted according to the method in subsection 2.1.1, as shown in Figure 3.1. Some things that were immediately noticed were that there seem to be two different types of classes. Partly radar that behaves like a collection of points where each point gathers around a couple of places, and partly classes that behave more like clusters where they are more widely spread. In the classes where points gather around specific location, however, there is a hidden complexity with internal structures that become clearer when you zoom in. An example of this can be seen in Figure 3.2 where the internal structure of the first observation from class 1 has been visualized. Whether these structures lead to further division or merging was something that was left to be investigated.

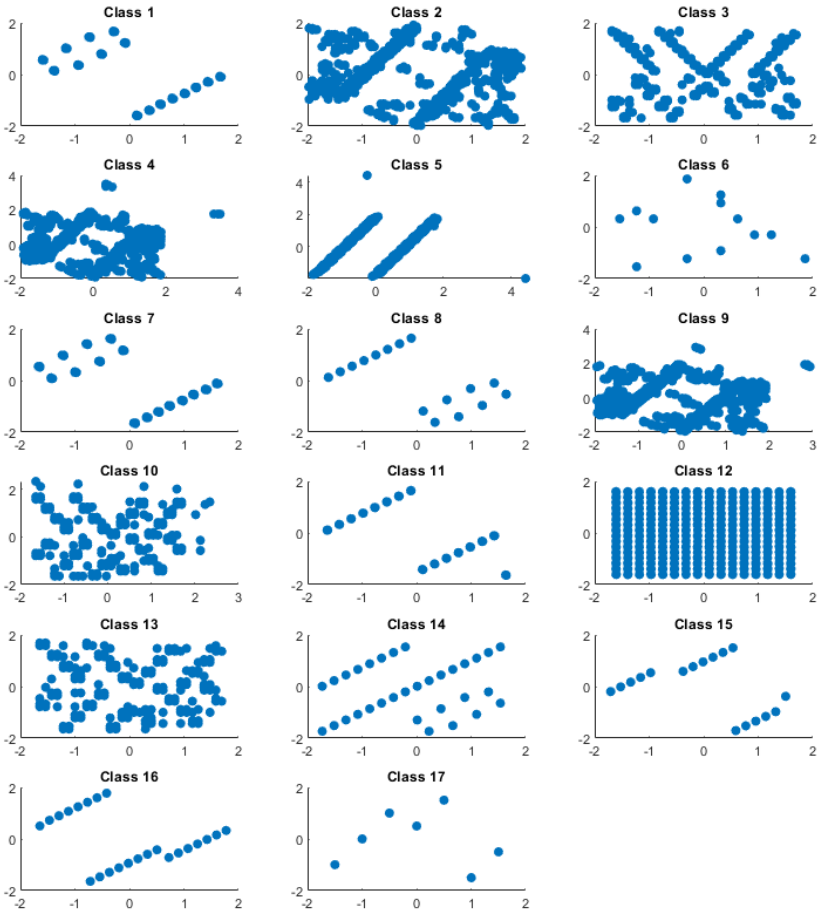


Figure 3.1: First observations from the initially given 17 classes

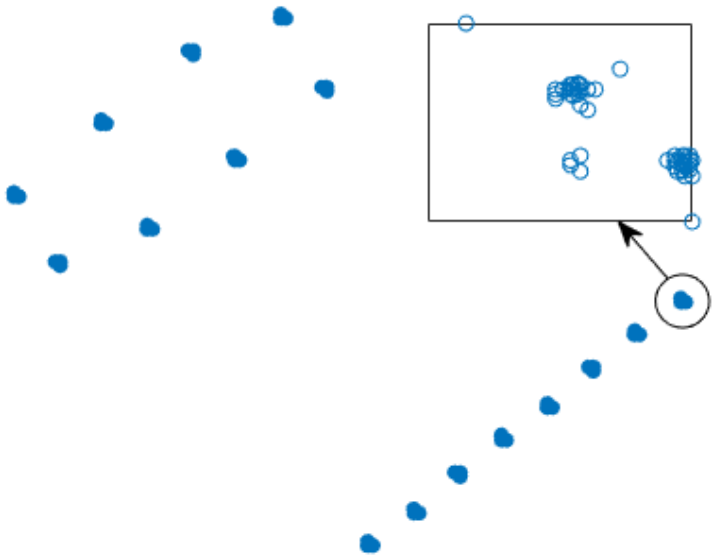


Figure 3.2: Zooming into an observation from Class 1.

A problem with the graphs in Figure 3.1 was that they only showed the first observation from each class. No information was available on the variation between the observations. Therefore, several observations were visualized simultaneously, from both the same and different classes, both above each other and side by side, in order to see similarities and differences between them. In addition to this, parts of the observations were also plotted to see how fast they converge towards the image that is visible to the entire observations. An example of such a visualization can be seen in Figure 3.3 where five randomly selected observations from each class are placed on top of each other. This type of visualization was used throughout the work in order to study the behaviour of the classifications.

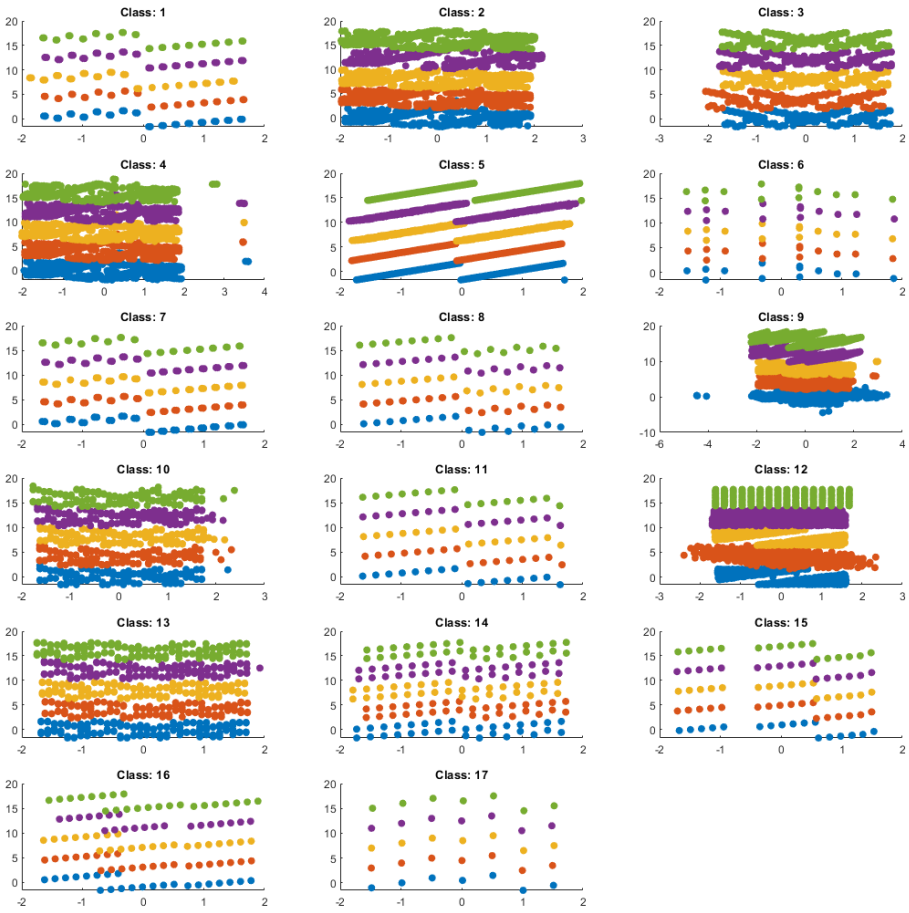


Figure 3.3: Visualization of five observations from each class, chosen at random and plotted above each other in the same graph

In this visualization, some conclusions could immediately be drawn just from looking at the plots. Class 1 and class 7 had great similarities and so did class 10 and class 13. Class 2 and class 4 seemed to be the same but there were some differences. Other classes, most notably class 12 and class 9, obviously consisted of several different types and needed to be further subdivided. In addition to this, a k-NN algorithm was tested to see how well a simple algorithm can classify data based on this first division. This was assumed to partly provide a kind of performance measure of how well divided the data is and partly provide information about which classes are difficult to classify.

The results from having examined the visualization of data and the results of the k-NN algorithm supported the thesis that further division and re-sorting needed to be done. It was assumed that some classes would be merged while others would be further divided. Many observations were also considered to be outliers and needed to be cleared out. To do this, the original division was used to set an accuracy, but the data were chosen to be considered unsorted. Then an algorithm was created to objectively classify, merge and divide observations based on this accuracy. The purpose was not only to make an assessment with the eyes but to create an algorithm that can also take the internal structures into account.

3.2 Method for Reclassification

The task was now to create a method for reclassification of the original data. The data was marked as unlabeled but the precision of the original division were still used as a reference point, as explained in the introduction of the section. First, a method to quantify a comparison between observations was introduced. Then, an algorithm consisting of three steps was constructed in other to create a new division of the data. In the first step the observations were split into multiple classes based on a high similarity. In the second step, those classes were merged in order to get the correct level of precision. In the third step, some remaining outliers and smaller classes that were unable to be merged were thrown out. Finally, a last manual sorting was made using a professional judgement to fine tune the division.

To compare the quality of the reclassification, two performance measures were used. First, the k-NN algorithm were used on the different classifications. The reasoning was that the more clear a classification is, the better result k-NN would yield. The second performance measure was the Fisher Linear Discriminant which punishes large internal variation within classes and small variation between classes. These performance measures were applied on the initial division of the data as well as the new divisions in order to determine how well the reclassification had worked.

3.2.1 Comparison of two observations

Since all data were now considered unclassified, it was necessary to form a measure of how equal two observations are to each other. This measure is to be later used in the algorithm when computing a new partition. However, it was not entirely clear how this measure would be designed as there are several difficulties with data. As previously mentioned in section 3.1, there are several irregularities between observations belonging to the same class. For example, two observations from the same class can be of varying length, and also contain very different arrival times. These irregularities can be explained by them not having started the collection of data at "the same pulse". It was therefore natural to consider only the differences obtained from the original vector of arrival times. As described in section 2.1.1, after a standardization, a 2D representation of these differences can be formed. This representation will be used to obtain a new partition of the data, beginning with the algorithm described below.

The method for determining how close two observations are is based on this representation. Assume we have two observations A and B. If these observations are very similar, they will also most certainly overlap. Then each point in one observation should have an corresponding point in the other observation. For each point in A, compute the distance to the closest point in B. The average of all these distances will then be very small if the two observations are similar and hence belong to the same class. More formally explained, the method works as follows: Start from two observations A and B, where the first has m points and the second has n, in the xy plane. For each point in the first observation, all distances to the n points in the second observation are determined. These distances are entered into a matrix of size $m \times n$. For performance reasons, the squares of the distances are calculated. To determine how close A is to B, the average of the smallest value for each row is taken. Similarly, the distance from B to A can be determined by considering the columns instead of the rows.

As previously mentioned, all data were treated as unsorted. The algorithm begins by taking a list of observations that have not yet been sorted. Initially, this list contains all observations, where they are read one at a time. The very first observation forms a new class, the *new* class 1. The first observation in each new class is here called the *representative* of the class. The algorithm continues by taking the rest of the observations, one at a time, and which are compared with the representatives of the new classes. This comparison is made according to the method outlined in the previous paragraph. However in this case, the comparison is done in both directions and the largest value of these two distances is used. If the observation is sufficiently similar to one of the representatives, it ends up in the corresponding class. If all comparisons with the existing classes are greater than a given *tolerance*, then the observation forms a new class. This allows the unsorted

data to form several new classes of varying sizes. The process is done a total of three times, where at the end of each iteration the observations belonging to classes with less than five observations are collected. These are then reinserted into the algorithm, but now with increased tolerance. The reason for this is that in the initial step, a low tolerance is needed to be able to say with certainty that observations within the same class belong together. However, some observations are only correctly classified when the tolerance increases by a factor of 2 or 4, which is why it is gradually increased. The observations that truly do not belong in any class usually give a distance greater than $10x$ the initial tolerance. The observations that after all three iterations belong to classes with less than 5 elements are assigned to a new class, labeled *trash*. The core of the algorithm is outlined in pseudo-code below.

Algorithm 1: Sorting data into a new partition

```
Result: classified
unclassified = [1, 1; 1, 2; ...17, 999; 17, 1000];
classified ← {};
found_classes ← 0;
tol ← 0.2;
for obs in unclassified do
    min_value ← intmax;
    for current_class ← 1 : classified do
        rep ← classified(current_class, 1);
        similarity ← max(dist(rep, obs), dist(obs, rep));
        if similarity < min_value then
            min_value ← similarity;
            min_class ← current_class;
        end
    end
    if min_value ≤ tol then
        append obs to classified{min_class};
    else
        found_classes ← found_classes + 1;
        classified{found_classes} ← obs;
    end
end
```

A total of 49 new classes were obtained, while 57 observations were labeled as trash after this first step.

3.2.2 Merging the observations

The second step of the algorithm was to merge the observations divided into 46 classes in the first step. The quantification of a similarity between two observations described in section 3.2.1 was once again utilized, along with the idea of allowing the first observation in a class represent the whole class. In addition to this, an assumption about the total amount of classes had to be made. Glancing through the observations and looking at the distinctions made in the initial partition, it was reasoned that no more or less than 15-30 classes were reasonable.

The first thing that was noticed was that certain classes were way more sensitive to internal variation. Looking at 3.3 one can see that some classes follow a more clear pattern of points (referred to as point classes) while other seem to be more of a cluster (referred to as cluster classes). It made sense to handle these two separately as there was unlikely to be any mix up between them anyway. Some further studying suggested that there are were a maximum of ten point classes and at most 20 cluster classes. The algorithm then followed as below, with the two types handled individually.

Algorithm 2: Calculating the similarity of two classes

```
Result: P
N = number of classes;
Let first observation in each class be representative;
Create three NxN matrices, M,V and P;
M is for the means, V for the variance and P the probabilities;
for each class c1 do
    for each class c2 do
        for each observation o do
            Calculate distance between o and representative obs of c1;
            add it to vector of distances;
        end
        Calculate the mean and variance;
        Insert in position (c1,c2) in M and V;
    end
    for each class c1 do
        for each class c2 do
            P(c1,c2) = (M(c1,c2) - M(c1,c1))/(V(c1,c2) + V(c1,c1));
        end
    end
end
```

This resulted in a 46×46 matrix of probabilities that two classes were to be merged. The probability is larger for classes that are more similar and smaller for classes with a small internal variation. As the goal was to be consistent in merging classes, one now wants to merge classes based on similar probabilities. That is, one could not just go through the matrix and merge the largest probabilities because it would be unclear where to stop. For example, one could theoretically merge all of them into one class and achieve perfect results, but obviously this was not the goal. What instead was done was that a limit was chosen, initially high, and all class with probabilities higher than that were merged. This limit was then lowered by an arbitrary scaling factor and the process repeated. This was done until fewer than 10 point classes and 20 cluster classes existed. The goal here was therefore to choose a scaling factor that wasn't so big that all classes were merged at once but also not so small that only one at a time were chosen. Instead it should be chosen so that approximately equal probabilities were merged at the same time. This was done a few times as the scaling factor was tuned and finally a result of 28 classes were achieved.

3.2.3 Removing outliers

In the final step of the algorithm, outliers were removed. An observation was classified as an outlier if the distance from it to the representative observation was more than 3 standard deviations from the mean distance within the class, once again using the distance measurement from section 3.2.1. In addition to this, classes smaller than 100 were also rejected. The reason for this was that at least 100 observations were needed to properly train and test the neural network. The observations from these classes were then handled individually and inserted into one of the remaining classes if they were more similar than the limit for what constitutes an outlier, defined above. The remaining observations and outliers were then put into a 19th class, called the trash class, along with the rejected observations from section 3.2.2.

3.2.4 Manually sorting

After the data had passed through the previous steps, a total of 16 new classes had been formed. The goal in the previous steps had always been to achieve a partition where it was clear to see the difference between the classes. However, there was an internal spread for each class, which differed between the classes. The smallest spread was found in the classes with few clusters, which have been previously referred to as *point classes*. Four of the so-called *cluster classes* contained such a large variation that it was considered appropriate to study these observations manually. Therefore,

assistance was taken from Tommy Hult at FOI in Linköping who is experienced at studying radar signals, especially the data-set in question. Together with Hult, all observations from the four classes were studied and assigned into new classes. After this process, the observations in the four previous classes had been divided into 18 new classes.

3.3 Performance metrics

Once a new partition had been obtained, it was important to use some performance metrics to actually verify that a better classification had been achieved. As described in 2.3, the k-NN classifier is a simple and straight-forward method for classifying 2D-points. In our case, we were dealing with observations consisting of a large number of 2D-points, usually well over 1,000. Our reasoning in this step went as follows: if two observations belong to the same class, then the points from each observation would certainly overlap in 2D-space. When considering a single point from an observation, it would then most likely be classified to the correct class. This stems from the reasoning that there are plenty of points in the neighbourhood of that point, that also belong to the same class. We considered an entire observation as a collection of 2D-points. This collection would then be classified, one point at a time, using the k-NN classifier. The class that most points were classified as, would be the label of the entire observation. A total of 100 observations from each class were used, with a split of 70-15-15 between the three data sets: *train*, *val*, *test*. For the validation data k varied for $k = 1, 5, ..., 195, 200$. The value of k that gave the highest accuracy, was then used on the test data.

The Fisher Discriminant Ratio was computed in MATLAB according to equations 2.1 and 2.2.

3.4 Results

The performance metrics used on the partitions were the Fisher Discriminant Ratio and the accuracy from the k-NN classifer. To begin with the Fisher Discriminant Ratio, described in section 2.4, was used to acquire one value for each partition. This was done for the initial given partition as well as for the final partition. The respective FDR values were 13.97 and 19.00. The accuracy for the k-NN methods were 72 percent and 82 percent respectively. The final 18 classes are shown in Figure 3.4 and the results for the k-NN algorithm are visualized in Figure 3.5.

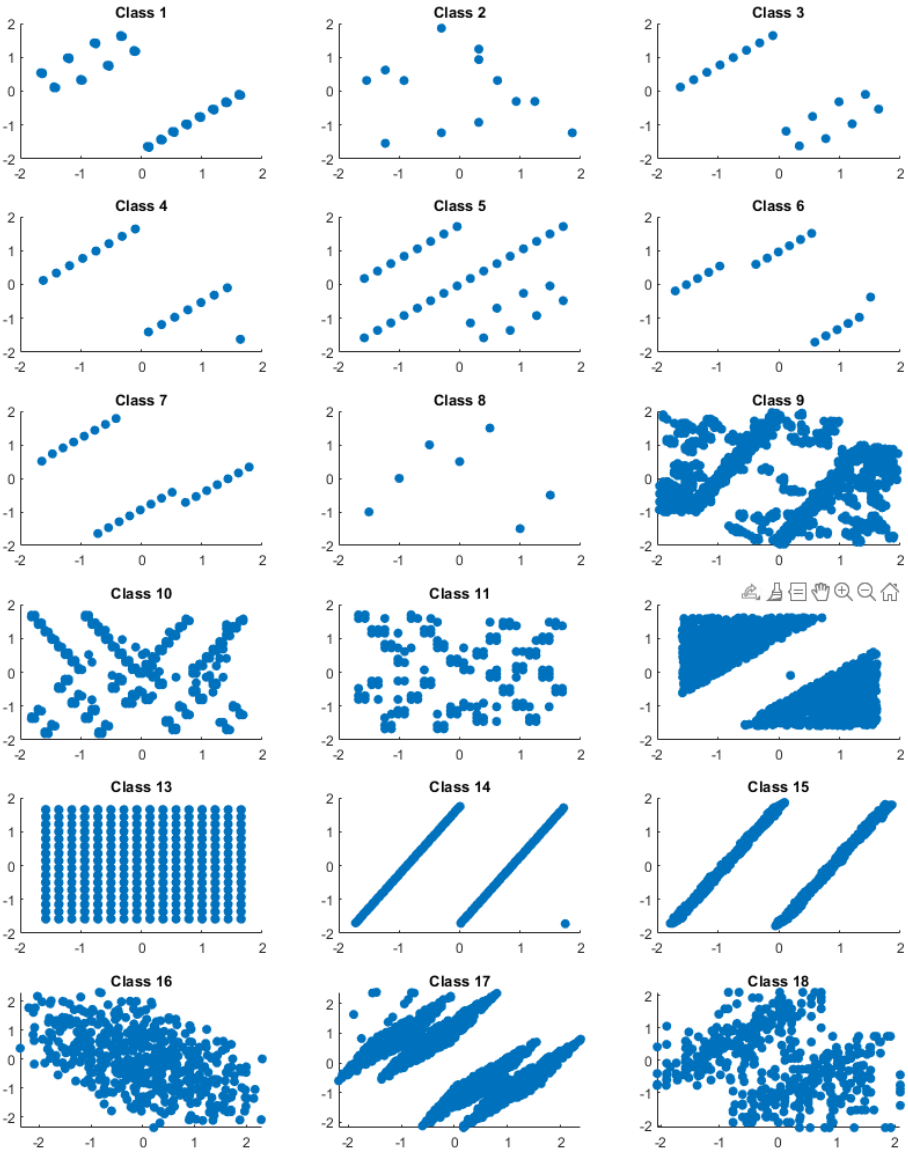
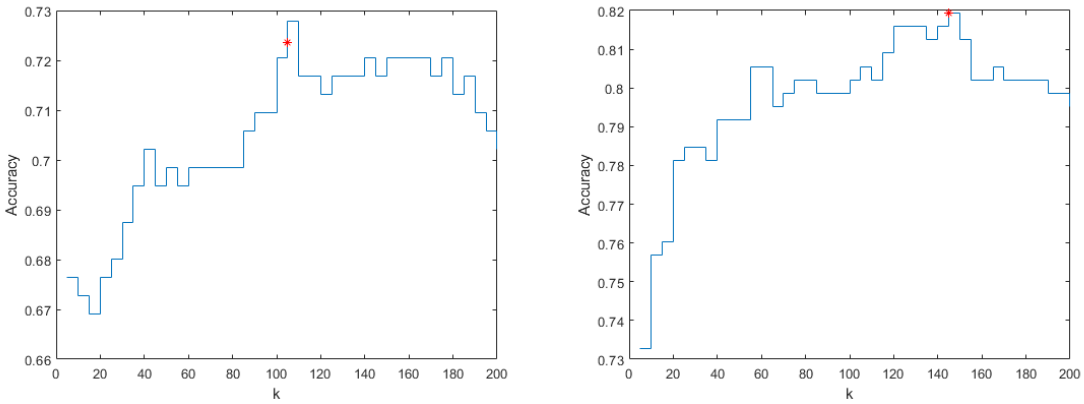


Figure 3.4: Final 18 classes, visualized by one observation in each plot.



(a) Original partition

(b) New partition

Figure 3.5: Graphs showing the original and the new partition of the data being evaluated using the k-NN classifier for varying values of k. The blue line shows the accuracy of validation data while the red star indicates the accuracy for the test data.

3.4.1 Trash data

In total were 393 i.e. roughly 2.3 percent of all observations considered trash and were put in the the trash class. These were either the result of disturbing outlier points, faulty receivers or transmitters or multiple signals being received at the same time. They were saved out of interest to later examine how the future networks would classify them. A few examples of observations classified as trash are shown in Figure 3.6.

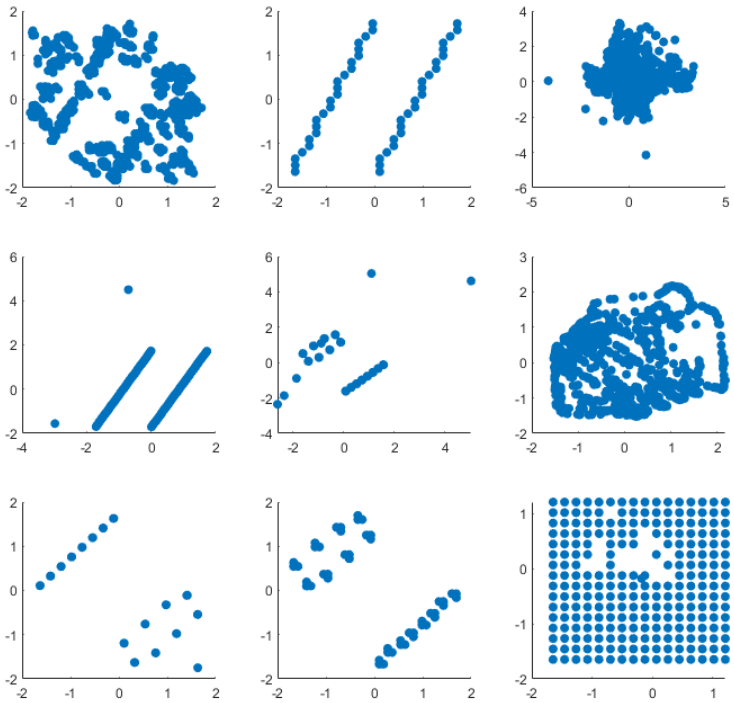


Figure 3.6: Nine observations that are labeled as trash

3.5 Discussion

Looking back, we are surprised at how well the algorithm for sorting the data worked. The measure for how two observations were compared was designed from scratch, as were the algorithms and methods used to form the final partition. Although there were some problems along the way, as having to separate the classes into *point classes* and *cluster classes*, we were able to achieve a working method for sorting the data. An important note however, is that this method depends on a few parameters and for this reason, our algorithm might not perform as well on other data sets. The parameters were also adjusted based on the output, introducing bias. However, some assumptions about data always has to be made, especially when no specification about the data is given beforehand.

The measurement used to compare two observations has a few flaws. First, it does not take into account how many occurrences there are at each point. Two observations that look similar when visualized in 2D might not be the same if the distributions of the points differs. The second disadvantage of the measure comes when taking the mean of the distances. If a few points in an observation deviate heavily, then this is "suppressed" when taking the mean, and hence the observation containing outliers might not be correctly sorted.

In the second and third parts of the algorithm, the sorting is made using statistical models. However, this is based on the distances between observations, rather than a property of each observation. Having a bad *representative* for a class, or simply adding an observation to a class that doesn't belong, could heavily impact the results. An alternative approach would have been to examine the classes further to distinguish features that were specific to each class. The lengths of the observations along with how the 2D-points were distributed are only two examples of features that could be considered.

Chapter 4

Neural Networks

This chapter is split into three main parts. The first part concerns the methods used to pre-process the data, preparing the appropriate data sets needed for training and evaluating neural networks. In the second part, different types of networks, with varying hyper-parameters, will be tested and compared against each other. The model that provides the highest accuracy, along with a reasonable model complexity, will then be chosen to be used in the final part of this chapter. In this third part, it will be examined how the chosen network model performs for sub-sequences of different lengths.

4.1 Pre-processing and implementation

In this section, the methods used for pre-processing will be outlined. We will start off by describing the steps carried out to achieve the data sets *train*, *validation* and *test*. Pre-processing of data is a crucial step when working with neural networks, since if the data has not been processed in a adequate way, the resulting accuracy will be poorly affected. In this sense, the entire previous step of creating a new partition of data can be seen as the first step in the pre-processing.

Once the data sets required for training a neural network have been obtained, the properties of the neural networks, that will later be evaluated, will be described. Since most of the theory behind neural networks has been covered in section 2.2, it is assumed that the reader is familiar with the fundamentals of how a neural network operates, and the common terms used in this field. All of the methods described were applied on the *time difference vector* of an observation, as outlined in section 2.1.1, and not on the actual time of arrival data.

4.1.1 Test, train, validation split

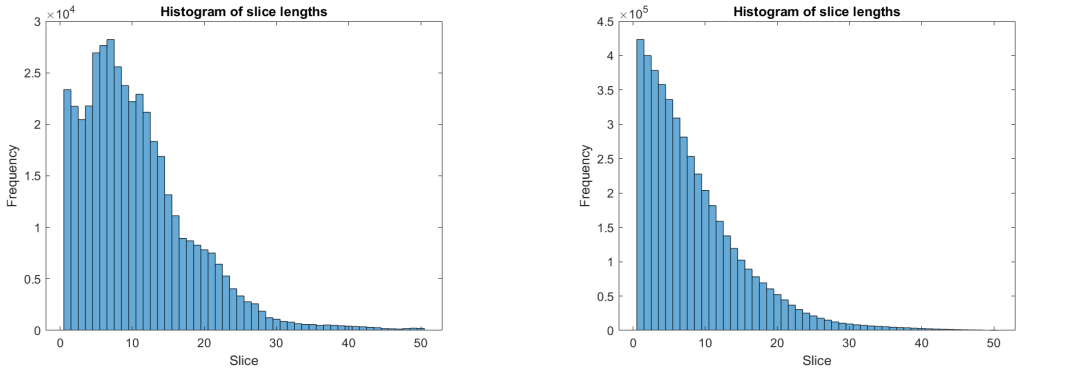
In Chapter 3, a new partition of the data was obtained, with distinct classes of varying sizes. Class 18 contained the smallest amount of samples, only 113, while class 9 contained the most, totaling 2522. To avoid any bias in the training of the networks, a total of 100 observations were chosen at random from each class. These 100 observations were fixed throughout the remainder of the project. For each class, the 100 chosen observations were split with 70 observations used for training and 15 observations used each for testing and validation.

4.1.2 Slicing and the windowing method

When we were computing the new partition of the data, we were able to consider an observation in its entire entity. For instance, two observations could easily be compared against each other to determine if they belonged to the same class or not. With neural networks however, it is not feasible to use an entire observation as input when training and classifying. This is for multiple reasons. For starters, each observation consists of at least 1,000 data points, with the largest observation consisting of 36,225 data points. Having a neural network with an input size this large will require an excessive number of weights, all of which need to be calculated during training. When

combining this with larger and more complex networks, the network will be too complex to train efficiently. More important however, is the fact that the data consists of continuous arrival times, along with both regular and irregular interruptions. Since a neural network learns to recognize patterns in different nodes, or sets of nodes, training it on data that is not aligned will result in poor accuracy.

Instead of taking an entire observation as input, a better way is to slice the data at the break points. Hence, one observation results in many smaller slices of varying lengths. By doing this, one can be certain that all data within a slice is in the correct order. It was important to know how common slices of different lengths were. Therefore, a histogram of the lengths of the slices, obtained from the 1,800 observations, was computed and is shown in Figure 4.1a.



(a) Number of slices of lengths n

(b) Number of slices of lengths n or greater

Figure 4.1: Histograms for the sub-sequences.

As can be seen, almost all of the subsequences are shorter than 30 time steps, as shown in Figure 4.1a. More interesting however, is to consider the total number of subsequences with lengths greater or equal to a certain threshold n. This is illustrated in Figure 4.1b. As can be seen, this is a rapidly declining curve. Longer slices contain more information and should result in higher accuracy when training networks. However, this comes at a cost as the amount of remaining data decreases as the slice size is increased. For instance, when $n = 12$ about half of the data remain, whilst when $n = 20$, only about 10 percent of the is available. To avoid discarding too much of the available data, only models with input length 12 or shorter were considered.

Whether or not one wants to normalize the data before slicing was an early question. When it was tried to use un-normalized data for the networks. About $5.55\% = \frac{1}{18}$ were classified correctly, which is precisely as good as a *random number generator* would perform. Therefore, all data used in the rest of the chapter was normalized.

Since the number of available slices were rapidly declining as n increased, we wanted to extract as much data as possible from each slice. If a network requires an input of size n then there are two ways to prepare slices of sizes m where $m > n$. First, one could simply extract either the n first or last values from each slice. However, this method would discard most of the data. A more important reason however, is that the inner sub-sequences would be lost. Since the data is full of irregular interruptions, it is important to consider every sequence possible, to obtain a well-trained network. A solution would be to allow a "sliding window" to run over the slices. In this way, one slice of length m would result in $m - n + 1$ new slices, all of length n. In this way, as much data as possible could be extracted from a sequence. This method is what is meant when the term *windowing* is used later on.

4.1.3 Constructing the data sets

As touched upon earlier, a split of 70-15-15 was used when construction the training, test and validation sets. Construction the final data sets used for training and evaluating was done in the following way. First an observation would be split into slices. If the model used input of length n then only slices of length n or greater would be considered. The larger sequences would be split up into slices of appropriate length using the previously described windowing method. However, if windowing was not used, only the first n elements from each sequence would be considered. The extracted slices would then be added to the correct data set (either training, testing or validation) and labeled. The labeling was done on the format of one-hot encoding, described in section 2.2.3. Depending on the type of class considered, each observation would generate a different number of slices. To avoid bias from the classes containing more slices than others during training, we made sure to cut the number of data samples from each class to the same size. In this way the slices from each class in `x_train`, the set used to train the network, would all be the same amount. It was also important to have the extracted data from all classes be of equal size in the testing and validation sets. This is because some classes might be easier for a network to classify than others. Bias in the number of samples from each class could hence affect the final accuracy.

Table 4.1: Sizes of the three data sets when $n = 10$

(a) Without windowing

Data set	Frequency	Percentage
`x_train`	93708	74.5 %
`x_val`	16344	13.0 %
`x_test`	15714	12.5 %

(b) Using windowing

Data set	Frequency	Percentage
`x_train`	308628	74.1 %
`x_val`	40626	9.8 %
`x_test`	67086	16.1 %

The table above shows the number of sub-sequences in each data set for $n = 10$. Notice how the partition split of 70-15-15 will not be maintained the number of sub-sequences extracted will vary for each observation. Also notice how windowing greatly increases the sizes of the sets.

4.2 Comparing the networks: feed-forward and recurrent models

Once the data sets had been constructed, it was time to build and evaluate neural networks of different types along with varying hyper-parameters. The first type of networks considered were feed-forward neural networks. As described in section 2.2.1, a feed-forward network (referred to from here on as *MLP*) is among the simpler kinds of neural networks. These networks consist of an input layer followed by one or more hidden layers, as well as a final output layer. Although there were plenty of parameters that could vary, the shapes of the input and output layers were always the same, regardless of the type of network chosen. Since the input was sub-sequences of length n, the input layer consisted of n nodes. Likewise, since the labels were constructed using one-hot encoding, the output layer consisted of 18 nodes, with `softmax` as activation function. The loss function was set in Keras as `categorical_crossentropy` due to the output from the network consisting of multiple distinct categories. As for the optimizer, ADAM was chosen due to its efficiency and it being well-suited for problems with large data sets [19].

For the MLP models, there were mainly three properties that were interesting to investigate. The first was to determine if the use of windowing when creating the data sets would improve training and hence result in a higher accuracy. The two remaining properties relate to the structure of the network. Namely, the complexity and number of layers in the network. One would assume that as the number of nodes in each layer and number of layers increase, that the accuracy would increase. Due to training and evaluating neural networks being a time consuming process, it was not possible to try as many combinations as desired. However, by carefully considering a few combinations to evaluate on, one was able to get a good understanding of how different properties

of the network affect the accuracy. The number of epochs (30) along with the batch size (128) used during the training remained constant throughout the run. Although the values set for the number of epochs and batch size may not result in the optimal model, this was of lesser concern. At this stage, gaining an understanding for how different feed-forward networks compared was more crucial. In total, 24 different combinations were run and evaluated. The inner layers varied from one to three, with the number of nodes in each layer ranging from 128 to 1,024. This was all done for the data sets with and without windowing. Table 4.2 displays the accuracy on the validation data for different network models.

As touched upon earlier, a feed-forward neural network is well suited for cases where there is a pattern in the features of the input data. In our case, the data being used consists of continuous time steps, where one would assume a strong time dependency between the values of the input nodes. In this regard, the relative difference between the values of the input data is of greater importance than the actual values at the exact positions. As outlined in section 2.2.2, a recurrent neural network, such as LSTM, is very well suited for these types of problems. In Keras, the overhead when changing from a feed-forward network to a LSTM-based is minimal, with the main difference being in how the input data is passed. In the case of the MLP models, each sample in the input data was being passed as vector of length of length n. When using a recurrent network, the input data is instead passed as a matrix of size $n \times 1$. This is interpreted as data from n time-steps with one feature at each step, just as desired. Hence, the shape of the input layer changed, but there was no need to re-compute the input data.

The MLP models considered either had one, two or three hidden layers. These were now all replaced by a single LSTM layer, with the number of nodes being varied. Besides also the output layer, which remained the same as before, there were no additional layers. There were two reasons for this. First, the number of trainable parameters greatly increase as the number of nodes is increased. This is due to the recurrent nature of the LSTM layer. Therefore, adding more layers might result in a network with too many parameters, making it difficult to train efficiently. More important though, is that the input data was so simple that it was assumed that a single layer of LSTM would be able to yield satisfying results. As long as the number of nodes in the LSTM layer was high enough, it was likely that the majority of the patterns in the input data would be learnt by the network.

As in the case with the feed-forward network models, a number of variations were tried and evaluated. The number of nodes in the LSTM layer ranged between 25 and 500 nodes. As in the previous case, this was done with and without windowing. The results are shown in Table 4.3.

4.2.1 Results

Table 4.2: Feed-forward neural network models for $n = 10$. Accuracy for validation data when at the end of training

Layers	Nodes per Layer	Trainable Parameters	Accuracy	Accuracy (windowing)
1	128	3,730	85.1 %	92.6 %
	256	7,442	90.1 %	96.0 %
	512	14,866	89.6 %	96.5 %
	1,024	29,714	87.7 %	96.0 %
2	128	20,242	94.6 %	97.4 %
	256	73,234	96.2 %	98.3 %
	512	277,522	96.6 %	98.2 %
	1,024	1,079,314	96.7 %	97.9 %
3	128	36,754	95.2 %	97.6 %
	256	139,026	96.5 %	98.3 %
	512	540,178	95.9 %	97.8 %
	1,024	2,128,914	4.6* %	97.5 %

* Deviating value. Further examination in the discussion.

Table 4.3: LSTM-based network models for $n = 10$. Accuracy for validation data at the end of training

Nodes in LSTM Layer	Trainable Parameters	Accuracy	Accuracy (windowing)
25	3,168	88.7 %	95.4 %
50	11,318	94.8 %	97.3 %
100	42,618	96.6 %	99.0 %
200	165,218	97.1 %	98.9 %
300	367,818	97.4 %	99.2 %
500	1,013,018	97.2 %	98.4 %

4.2.2 Discussion

In the first part of this section, when MLP networks were compared against LSTM-based recurrent networks, there were plenty of interesting results. In Table 4.2 one immediately notices how windowing greatly improves the accuracy. There appears to be an increase of at least five percentage points for the networks with one hidden layer. For the networks with two and three hidden layers, the increase is not as large, but still noticeable and consistent for every model. One also notices how the accuracy increases as the number of nodes per layer is increased. The accuracy usually reaches its maximum between 256 to 512 nodes, to then decline for more complex models.

In terms of being the best model, there were two clear candidates. The first was the model which consisted of two inner layers with 512 nodes each. While the second candidate was the case where we had three inner layers with 256 nodes in each layer. With similar accuracies, we have to consider the complexities of the models. With the second model having a total number of trainable parameters roughly equal to half of the first, this one is to prefer.

With most of the values being as expected, there was however one value in the table that stood out. When training the model for three layers with 1,024 nodes in each, the resulting accuracy (without windowing) was only 4.6 %. This is marked in the table by an asterisk (*). It is not fully clear why this happened. Since not using windowing results in much smaller data sets, a hypothesis could be that the ratio between the number of trainable parameters and the number of training

samples is too great. It might just be that the optimizer can't efficiently compute all the weights required to increase the accuracy. This was confirmed when closer examining the *training history*. As seen in appendix A.1, it is clear that the network struggles to optimize the parameters. The same problem does not seem to appear when a larger training set, as when windowing is turned on.

When looking at the results from the recurrent networks, we again notice a drastic improvement when using windowing. This increase is, once again, more profound for simpler models with fewer parameters. One also notices how LSTM-based networks tend to perform better than traditional feed-forward networks when the number of trainable parameters are roughly the same. We also notice how LSTM networks provide much higher accuracies, despite far fewer trainable parameters. For instance, the LSTM(100) network with 42,618 parameters outperforms even the most complex MLP models. Just as for the networks in Table 4.2, the accuracy tends to decrease as the complexity of the network increases above a certain point. Among the networks in Table 4.3, we find that the network with a LSTM(300) layer gives the highest accuracy. However, this increase in accuracy comes at a price. When comparing LSTM(100) against LSTM(300) the increase in accuracy is about .2 percentage points, but with 8.6 times the number of trainable parameters. We therefore decided to use both network models in the next section to see how they compare for sub-sequences of different lengths. Since windowing always seem to yield higher results, it will from now on always be used.

4.3 Evaluation on sub-sequences of varying lengths

Once it was decided on which models to use, it was time to test and evaluate how they performed. It was decided to try all sub-sequences with lengths $3 \leq n \leq 12$. The reason not considering longer sub-sequences than twelve was partly due to the rapidly declining number of available sub-sequences as the length increases. More importantly though is that, since the networks with input length $n = 10$ performed so well, we were keen to examine how it would perform on shorter sub-sequences. Due to the nature of collecting radar signals, usually having scarce data, it was more interesting to see how well the network performed for smaller inputs, along with at what point the biggest drop in accuracy would occur. Sub-sequences of length two would most likely not hold enough data to yield any good results and were never considered in this test.

The process of implementation was quite straight-forward. Since the different length sub-sequences being evaluated all need a different number of nodes in the input layer, each test for sub-sequences of length n, require a new model to be trained. And since the three data sets *test, validation* and *train* depend on the chosen slice size, the test-data were different for each test. Imagine we have a sub-sequences of length ten. When we compute the data sets for sub-sequences of length three, this single original sub-sequence will create a total of eight new sub-sequences. The data sets not only grow in size, but also in the number of different sub-sequences, as n is decreased. As the lengths of the sub-sequences become shorter, the sequences from different classes become harder to discern, since they contain less information. This makes it harder for the network to find patterns, which increase the number of epochs required to provide a good result. For these reasons each model trained for a total of 100 epochs. When examining the accuracy plots for the validation data, with the intent of finding after how many epochs the accuracy leveled out, it became clear that 100 epochs was enough for all the models. A concern that the reader might have is the risk for overfitting when increasing the number of epochs during training. This will be addressed in the following discussion.

4.3.1 Results

Table 4.4: LSTM(100), accuracy for test data after training

n	3	4	5	6	7	8	9	10	11	12
Accuracy	85.3 %	91.4 %	93.4 %	95.1 %	95.5 %	96.6 %	97.1 %	97.3 %	97.4 %	97.8 %

Table 4.5: LSTM(300), accuracy for test data after training

n	3	4	5	6	7	8	9	10	11	12
Accuracy	85.9 %	92.0 %	94.0 %	95.5 %	96.3 %	97.3 %	97.6 %	98.1 %	97.8 %	97.8 %

Table 4.6: LSTM(100), confusion matrix for $n = 3$

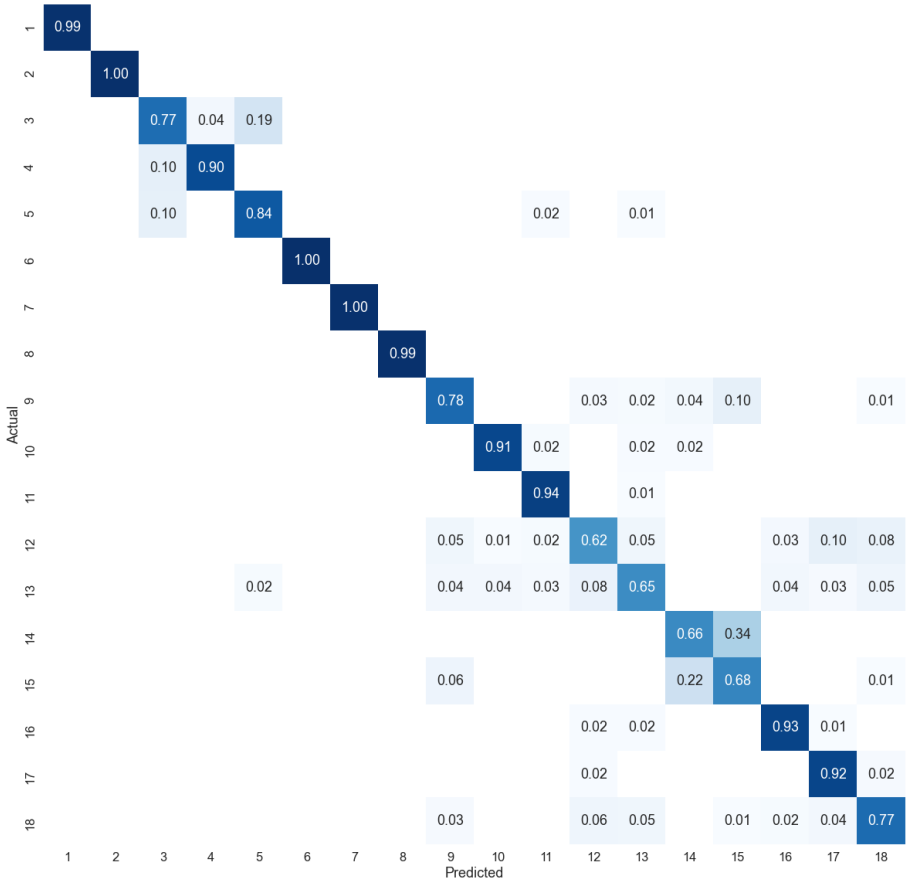
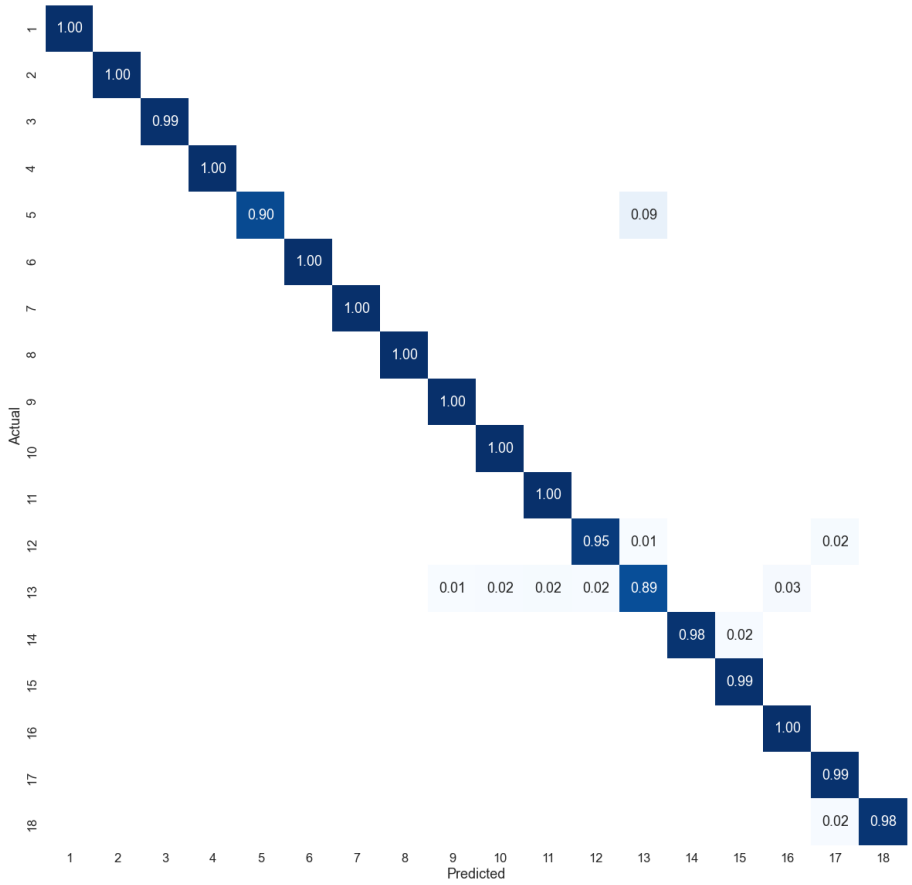


Table 4.7: LSTM(100), confusion matrix for $n = 12$



4.3.2 Discussion

As can be seen in Table 4.4 and Table 4.5, the accuracy seems to get worse as n is decreased. This was just as expected. What was not expected however, was how small the decline would be. Even for sub-sequences of length four, the accuracy was still above 90 percent. The largest drop in accuracy was noted between $n = 3$ and $n = 4$. This indicates that it becomes harder for the network to find patterns at this stage. The lowest accuracy, which was 85.3 percent for $n = 3$ in the LSTM(100) model, is still better than the results from using the k-NN classifier, which gave 82 percent. It is also interesting to note how the accuracy seems to stagnate for $n \geq 10$. This indicates that the optimum for how well this type of network can perform on the original data set has been reached.

Overall, the LSTM(100) model performed worse or equal than LSTM(300) for every n. However, the loss in performance was usually only about half a percentage point. The small increase in performance comes at the cost of a much more complex model. The LSTM(300) model consists of more than 8 times as many parameters as the LSTM(100) model. Due to this, the more complex model took nearly 6 times as long to train, and during evaluation roughly 3.5 times as long compared to the simpler model. We therefore decided to use the model based on LSTM(100).

In Table 4.6 and Table 4.7 the confusion matrices are shown for $n = 3$ and $n = 12$, respectively, when using the LSTM(100) model. Since $n = 3$ resulted in the lowest accuracy, and $n = 12$ performed the best, the two corresponding confusion matrices were included in the report. By considering Table 4.6 we immediately notice how the accuracy differs for the so called *point classes* vs *cluster classes*. Since the point classes consist of simpler, more predictable patterns, this is very reasonable. The only point classes that the network seems to struggle with, are classes 3 and 5, which sometimes are confused. This is very likely due to them having a similar appearance, as

visualized in 2D in Figure 3.4.

Table 4.7 shows a clear improvement. The problem with the cluster classes being harder to classify appears to be gone. Just as before, it appears that class 3 is often classified as class 5, although to a lesser extent than for $n = 3$. It appears however that class 5 is sometimes (in 9 percent of the cases) classified as class 13. Unfortunately, we do not have a good explanation for this. It could, for instance, be a result of the network trying to increase the total accuracy by improving the accuracy for other classes. The cost of gaining an overall higher accuracy might be that the accuracy for class 5 decreases.

The issue of overfitting was also considered. For our LSTM(300) models the accuracy plots for the training and validation data were plotted next to each other in order to study this, an example is shown in appendix B.1. It is seen that the training accuracy seems to continue to grow at a very slow pace while the validation accuracy plateaus. Therefore no overfitting seems to be taking place, at least not within the amount of epochs the model is trained on. This was attempted for a very long training time of 1000 epochs as well and the same results were obtained.

The reason for this is probably the very large amount of training data in combination with the relatively small sequences used in the models. In order for overfitting to take place, the weights would have to be adjusted to fit the output to always match, for example, class 3 to a specific sequence belonging to class 3. However since there are so many different sequences of class 3 being considered in each epoch, this adjustment keeps getting tuned to fit new sequences and thus never becomes a perfect fit for one. Usually in overfitting the model manages to still fit all the training sequences in a way that makes it lose generality but naturally this goes down as the diversity of sequences goes up. In addition to this a smaller input sequence means there are less patterns to fit perfectly, making the input of a large data set stronger. Therefore, it was concluded that for this specific implementation, overfitting was not a concern. However there was still the possibility that some generalization could be lost and dropout was also attempted. However this yielded worse results and the layers were therefore removed.

Chapter 5

Applications on whole observations

In Chapter 4, neural networks dealing with subsequences were established, trained and tested. In this chapter methods were developed to apply these networks on the whole full length observations instead of isolated subsequences. The reasoning was that a longer observation can be seen as a chain of subsequences and therefore it should be possible to apply networks able to handle subsequences in tandem to classify a longer sequence. In order to do this, the LSTM(100) networks from Chapter 4 were considered final and the focus of this part is to vary the n parameter and adjust it to different methods in order to see how it performs for longer sequences. These different neural network models will often be referred to as "Model n" for a specific value of n in order to simplify the text. Henceforth, "Model 5" will therefore refer to a LSTM(100) network with an input size of five.

To determine how well a method and network performs, four performance measures were considered. They were the simplicity of the model, the accuracy, the speed and if an observation is classifiable or not. The simplicity of the model referred to how few parameters are required to be saved and how few calculations are needed to make a decision. The accuracy of the model meant how often an observation is classified correctly. With the speed of the model, how much time a model requires to classify an observation was considered. How often a model could classify an observation referred to how often subsequences of the required size can be found for the network models to be able to make a classification. These performance measures were gathered by applying the models on the test data, that is the final 15 observations from each class that has remained untouched during the training and validation process. This was to avoid most of the bias and give as general of a result as possible. These results were first acquired for the developed methods using the fixed size networks introduced in Chapter 4 and then for more complicated models being able to handle varying input sizes.

In addition to this, the same networks and methods were also applied to trash data in order to study their behaviour for observations that did not belong to a class, or were parts of a class but heavily disturbed. Optimally, the model should be able to determine not only which class a sequence is from, but also if it is not from a class in the distribution at all, or if it belongs to a class that has been disturbed. One possible example could be a sequence that largely belong to one class but certain subsequences are erroneous, causing it to be marked as trash in the steps described in Chapter 3. Another could be a sequence detecting two different radar types, appearing in tandem. This is something the model should optimally be able to detect, allowing the supervisor to draw as much information as possible from the sequence.

5.1 Variation of sub-sequence sizes between observations

While it was made sure that there were an equal amount of subsequences from each class in Chapter 4, no importance was placed on from which observations they came. This means that it is possible that a network can perform very well on all the subsequences it is tested on but if all of these sequences only come from a few observations, that is you can not find sequences of that size in most observations, the model will still be very bad in practice. This is because it will not be able to make a classification at all for most observations. In addition to this it could also be that subsequences of certain sizes are very rare, meaning it could take a long time for the detector to find such subsequences, making the model very slow. This was investigated in order to confirm the suspicions. The results are visualized in Figure 5.1 where the sizes of subsequences for each observation in each class have been studied. The table shows the maximum slice size one can have for the model so that X percent of observations from class C will be able to be classified.

Looking at Table 5.1 one can see that in order to be certain that an observation can always be classified one can't have a size larger than six. For 90 percent that number is instead eight and for 75 percent nine. However for the majority of classes, larger models can still classify all observations and there is a large variation between classes. Recalling the results from Chapter 4, this adds another dimension to choosing the input size for the model. While larger sizes performed better, they will ultimately more rarely be able to make that classification for certain classes, yielding a worse result in practice. The large variation also offers some flexibility depending on the purpose of the task though. If the supervisor is only interested in classifying a certain amount of classes, he can allow himself to use a larger input size, and thus also increase the accuracy.

Table 5.1: The maximum sub-sequence sizes one can use for each class so that a certain percent of observations can be classified

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
75 %	12	13	15	18	15	9	9	13	19	13	15	20	17	15	13	20	16	12
90 %	9	11	11	14	15	8	8	12	14	9	11	18	16	10	10	20	13	9
100 %	7	7	9	9	11	7	6	10	9	6	8	11	14	9	8	11	8	7

5.2 Method

5.2.1 Motivation and explanation

In Chapter 4, neural networks for inputs consisting of uninterrupted sequences of a pre-fixed size were developed. In this part these models should be extended to longer sequences with regular and irregular interruptions, dividing them into subsequences of varying sizes. Once again these longer sequences are normalized, similar to how the subsequences were normalized before. The method should also be able to view the observations in real time. That is, the data points should be fed to the model in order and it should make a classification as quickly as possible. This introduces a couple of problems when it comes to normalization as one in practice can only normalize using the existing points. For simplicity this was ignored during this Chapter but the issues are addressed in Chapter 6. This classification should then be able to be updated as time progresses and more data becomes available. What this means is that rarely will the whole ten minute sequence be needed and optimally a decision and all the relevant information should be able to be found much quicker. Therefore the full ten minute sequence will often not be plotted as one can get a full perspective from a much shorter period of time. In addition to this it also saves time as running neural networks for long sequences can take a lot more time. However, all the results presented holds equal for the full sequence as well.

Two different methods were considered, where the second is a build-on from the first. This is because there are different scopes of interest. In some cases, one is interested in examining how classification varies over time, for example in order to detect types of disturbances/interference

between different radars. In other cases, one will be more interested in getting a single classification of the observation as quickly as possible in order to take a decision. The method dealing with the first case will be presented first. These methods are first explained for the more simple fixed-size-input neural network models but with a slight modification they can also be utilized for the more complex varied-input-size models that were attempted later on.

5.2.2 Simple Method

First a sub-sequence size N is fixed and the corresponding trained neural network model from Chapter 4 is imported. The plan is to use this model on batches of data of size N in order to match the input size of the model. For each such batch the model makes a classification that is added to a larger output vector at the matching time point of the last element in the input batch. The result of this becomes a large output vector where each element is a classification of the current batch of the observation. A batch has to consist of N following elements without any interruptions between them in order to match how the neural network model was trained on uninterrupted subsequences. A new batch is created by iterating over the data, taking steps of a pre-defined size to include either entirely new data or a mix of old and new data. If a batch includes an interruption the output should just be that no prediction could be made. As before, an interruption is represented by a NaN-value and the same holds for the output when a classification can't be made. The way the method works is visualized in Figure 5.1

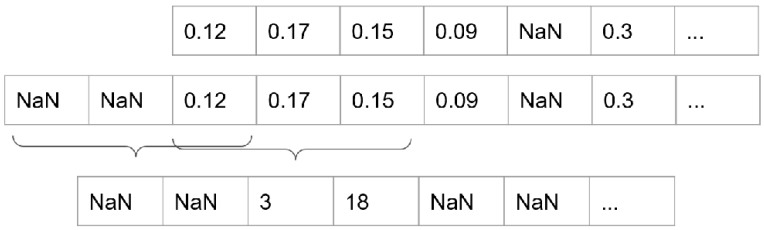


Figure 5.1: Visualization of how the simple layer works

A more precise description of how the algorithm works; The input is once again in the shape of a time-of-arrival vector. This is transformed into a difference vector and differences larger or smaller than a certain threshold is transformed into NaN-values in order to represent interruptions. A sub-sequence size N and a step size K are then defined and the corresponding neural network model for size N is imported. The difference vector is padded by $N-1$ NaN-values in the beginning so that the batch can begin with the first value as its last element. A window of size N taking batches is then allowed to iterate over the data, either over all of it if the data already exists, or over it as more comes if the model is being applied in real time. This window takes a batch, checks if it includes any NaN-values (if so it outputs NaN) and if it does not, it sends it to the neural network model as input and take the single output of the neural network as output. The output is placed in the larger output vector that is growing iteratively as more data is being processed. If a step size of one is used, the padding in the beginning will make it so that the index i of the output vector corresponds to index $i+1$ in the original time-of-arrival vector. Thus, one can map the classification to a certain time point, which is especially useful in real time. One can see that the first $N-1$ elements in the output vector will always be NaN for example, which is because no classification can be made with the model until at least N data points exist. If a larger step-size is used one simply has to make K copies of the most recent output in the output vector in order to have the same correspondence to the time-of-arrival vector

In Figure 5.1 an example of this is visualized. Here the first layer is a difference vector with arbitrary values and a single interruption shown. N is set to three and K is set to one. The difference vector is therefore padded with two NaN-values in the beginning, shown in the second layer. The window slides over the vector one step at a time and if no NaN-values are present in

the window, takes it as input to the imported neural network model, and outputs a classification. This classification or NaN-value is added to the output vector at the corresponding index shown in the third layer. Here, there are two NaN-values in the beginning as the two first windows includes a NaN-value. Then two classifications are made as there are four cells with non-NaN values and then a NaN-value is included again, resulting in yet another NaN-output.

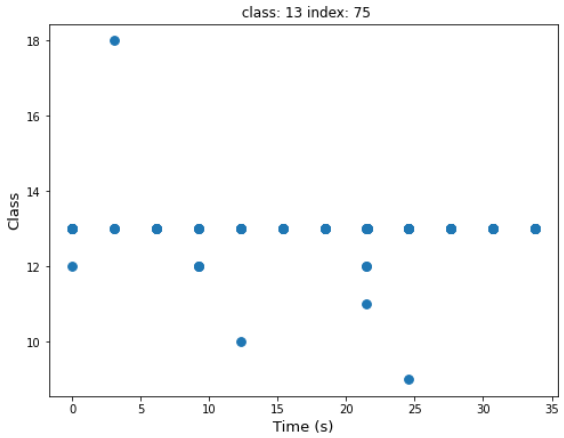
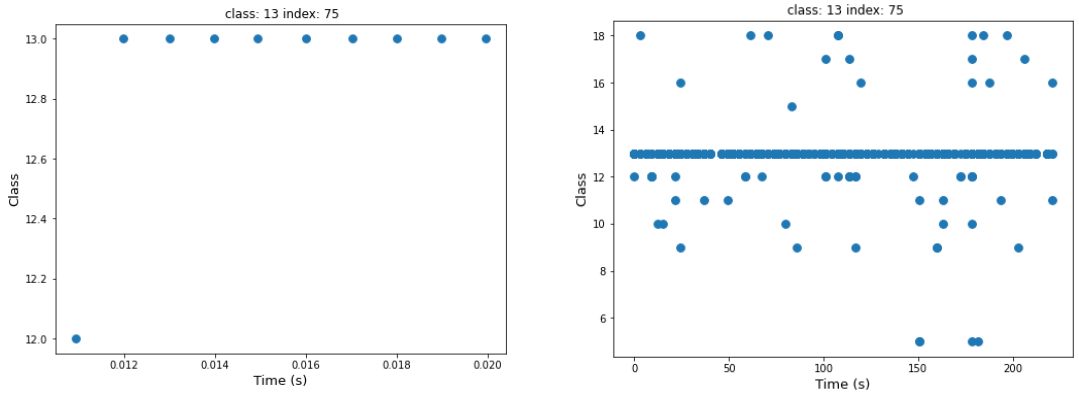


Figure 5.2: The method applied on the first 250 data points of observation 75 from class 13, using the neural network model with fixed input size seven

An example of how this can look in practice is shown in Figure 5.2 where the method has been applied on an observation from the data set. The neural network used here is with input size seven. The time is on the x-axis and the classifications on the y-axis and the output has been plotted for the first 250 data points, roughly corresponding to the first 35 seconds of the observation. When no classification is made it is represented by an empty value, resulting in large gaps when the transmitter is turned away from the receiver, which also can be seen in the plot. The same observation is plotted for 20 and 1500 data points as well in Figure 5.3, corresponding to about 20 milliseconds and four minutes respectively. Here it can be seen that there are multiple points hidden in each marking in Figure 5.2, meaning the method rather quickly makes a lot of predictions before the longer interruptions. One can also see that the behaviour does not really change over longer time periods. The method mostly classify the observation correctly as Class 13 but there are faulty classifications occasionally. In these examples a subsequence size of seven has been used but both smaller and larger models were also tested and the results studied. In general smaller sizes resulted in more frequent classifications but also more faulty classifications, as was expected.



(a) (b)

Figure 5.3: The same model and observation as before, but visualized for the first 20 data points (a) and the first 1500 data points (b)

The method gives rise to some interesting patterns and an alternative way to visualize the radar signals to the previously used 2D-pattern. Poor subsequences being wrongly classified that in the 2D-pattern might have given rise to weird shapes distorting the classification process can now be isolated and studied to see what went wrong. One could also remove the data points corresponding to those subsequences and plot the 2D-pattern again to see if it looks different. However, it is also a possibility that the reasons certain subsequences are classified wrong just comes down to the fault of the network model, especially for those with a smaller slice size. This type of analyzing will be further explored in a coming section.

5.2.3 Majority layer

An issue with the method described in the previous section is that while it usually makes the right classification, it also makes faulty ones occasionally. What is worse is that these misclassifications are not isolated to the early parts of the observation but can happen at any time. This is reasonable since no further memory aspect than N elements back have been introduced. The purpose of the improved method is therefore to introduce a form of memory, allowing the model to be more consistent in how it classifies a sequence, especially over time. This is done by adding yet another layer to the previous method, called a majority layer. How this is done is visualized in Figure 5.4 and explained in the next paragraph.

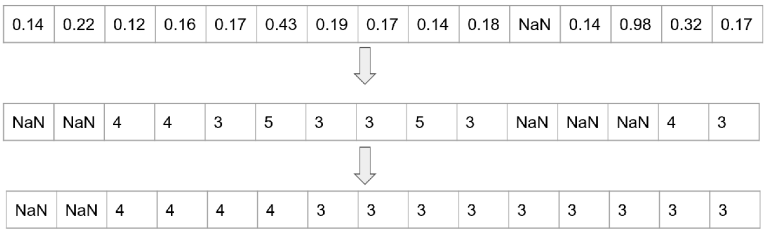


Figure 5.4: Visualization of how the majority layer works

The first two layers work exactly the same as in the previous task. (Note that there should actually be four layers here, but the padding part has been excluded in the visualization for visibility) But in addition to this the third layer takes the output of the previous layers and changes the value at index i to be the majority of all values on positions smaller than i in the previous layer. What this means is that the output is said to always be the majority sum of what

the network model has previously predicted the observation to be, unrelated to what the current subsequence is classified as. The motivation is that even if the model sometimes classify wrongly, it is assumed to be correct most of the time. Using probability this means the majority layer will converge to the correct classification as more data becomes available and the correct classifications are likely to outweigh the incorrect ones.

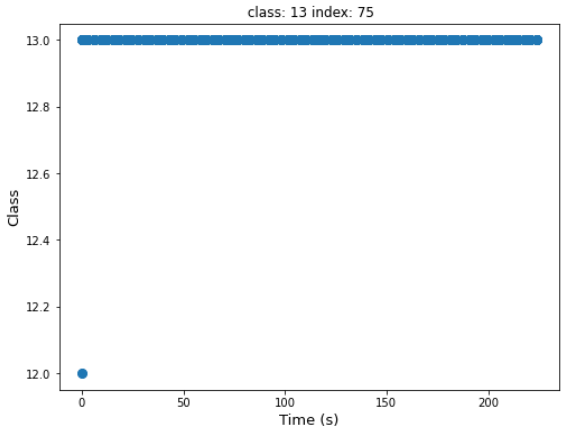
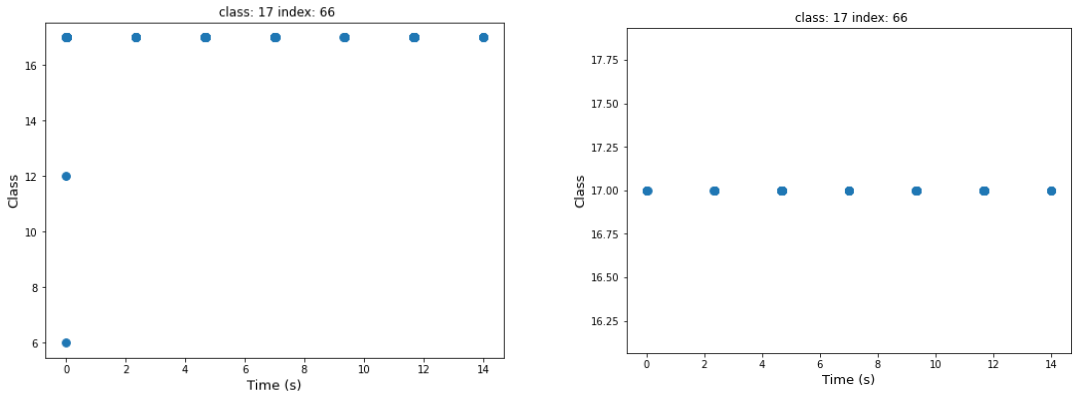


Figure 5.5: Majority layer applied on the same observation as before

In Figure 5.5 the majority layer method is visualized for 1500 data points, once again using the same observation and neural network as in the visualization of the simpler model. It can be seen that it converges to the right class quite rapidly. Comparing it to Figure 5.3 b) one sees that there are no occasional errors after the initial misclassification and that the method converges to the correct value very quickly. There are also no large gaps in the classifications. Another example is shown in Figure 5.6 where the majority layer has been used for two different neural network models on the same observation. The first one is with input size three and the other with input size nine. Here, one can see how the smaller neural network is making a couple of really quick faulty classifications before it converges to the correct class. The larger network on the other hand takes a longer time to decide but immediately converges to the correct decision.



(a) (b)

Figure 5.6: Majority method used on observation 66 from class 17 with two LSTM(100) neural networks with different input sizes. In a) the input size three was used and in b) the input size nine

Due to the nature of the majority layer, it almost always converges to some classification, whether it is right or wrong. It therefore opens up a new scope of interest besides accuracy, namely how quickly the model converges. As is seen in Figure 5.6, there is a very different behaviour for two

different neural networks even though both classify the observation correctly in the end. It is reasonable to assume that the speed with which the classification converge largely vary based on which model one is using. If two models have an equal level of accuracy, clearly the quicker one is to prefer if one works in real time. The majority layer will be used to compare the different types of models where along with accuracy, the convergence speed will also be used as a performance measure.

5.2.4 Expanding to varying subsequence size

An issue with the methods discussed above is that one either has to decide to use a small model and get possibly worse results, or use a larger model but discard a lot of data. It seems like a better solution is to allow for varying subsequence sizes and simply take all points between two interruptions as input. However it is also possible that this would result in some setbacks, as it is not certain that a larger subsequence size always performed better cumulatively. For instance, a larger size could lead to a slower convergence and also the inclusion of erroneous data points in the network estimations, corrupting the result. It was clear this had to be investigated experimentally.

There were two possible ways of doing this. First one could use a so called padded model. That is, a new network model that is trained on different subsequence sizes that it pads to a pre-decided size, allowing it to use input of the same size every time. Such a model was created and trained with a size of ten in order to keep the consistency from Chapter 4, allowing it to be trained on the same data for the same amount of time with the same type of network as for the fixed-size models.

Another idea was to use all the trained models for sizes up to ten and use the largest model possible for each subsequence. This meant that for each subsequence, a classification could be made while the largest model was always chosen. A setback of this this was that it would require about eight times as many parameters to be stored though, but if the results were significantly better, it could potentially be worth the trade off. The reason a maximum model size of ten was chosen was that the larger network models simply did not perform better so including any larger models would be a waste of memory space. The padded model also performed its best on subsequences for a size of ten, meaning that ten was chosen for that model as well.

For both models, a slight adjustment had to be made. Instead of using a fixed subsequence size to determine N, it was simply always set to three, allowing sub-sequences of at least size three to always be classified. Lower subsequence sizes were not considered since that was deemed to be too simple to include much interesting information. In addition to this the window was not of a fixed size taking fixed steps but instead was allowed to grow until it reached size ten or included a NaN, after which it behaved as before.

5.3 Results

With the methods to apply the different models in practice established, the models were ready to be tested. The eight fixed-size models along with the two varied-size models were tried out for the entire test data set, consisting of 18x15 observations, and the results were summarized. There were four performance measures to be considered. The accuracy of a model, the speed with which it converged, how often it could classify at all and how simple the model was.

5.3.1 Performance of the different models

The first performance measure considered was accuracy. Simply, using the majority layer, how often did the model settle for the correct class. Recalling that each observation outside the trash data was assumed to only belong to one class of radar, there were always assumed to be a correct decision. The model was considered decided for a class after 100 outputs in a row had been the same class. If that never happened, the specific observation was to be considered unclassified similar to if it only contained NaN-values. The second performance measure was how often a class could be classified at all. That is, what share of observations belonging to the class in the test

data was marked as unclassified by the model, meaning the output only consisted of NaN-values. The reasoning behind separating unclassified from wrongly classified observations was that it was considered more dire to make a faulty classification than to simply be unable to classify a observation.

The third performance measure was how quickly the model converged to a decision, irrelevant if that was correct or not. As was stated in the previous paragraph, the model was considered converged if 100 following values in the output vector was the same. A model was therefore considered converged at the first time point of such a sequence of 100. In addition to this, the simplicity of the model was also a factor. However all the models were of the sort LSTM(100) and thus had as many parameters to store. The one exception was the model using a combination of all the fixed-size models that therefore used eight times as many parameters.

To begin with the test data set was extracted, consisting of 15 observations for each class, or 270 observations in total. The accuracy and convergence speed for the fixed-size models were then acquired and the results are summarized in Table 5.2 and Table 5.3, respectively. In Table 5.2 each column represents one of the neural network models and the rows a specific class. The cells contain the amount of observations from that specific class that were correctly classified for that specific model. The final two rows show the total percentage of observations that were classified correctly and were unable to be classified, respectively, for each neural network model.

For the convergence speed table the columns once again represent the different models and the rows the different classes. However here the cells show the average convergence speed for that specific class and model. The final row shows the exact average convergence time for all classes for that specific model. In addition to this, all the models have been run for the first 1000 data points in each observation. Due to the large variation in length between observations, shown in Figure 1.1, this was chosen to guarantee that no observation would be unclassifiable because of a lack of data points. 1000 data points on average represent about a third of the total ten minute time span, or around three to four minutes of listening. It was also confirmed that different results were generally not achieved when including more than a couple of minutes, thus the whole 10 minute sequence was seen as redundant. This is further confirmed by the convergence times shown in Table 5.3 which never are of the order of hundreds of seconds.

The same performance measures were then acquired for the two models that utilizes a varying input size, described in the previous section. The results for these two are shown in Table 5.4 and Table 5.5. Note that the amount of unclassifiable observations are emitted here as it is zero for both models.

Table 5.2: Accuracy for the fixed-size neural network models

Class \ n	3	4	5	6	7	8	9	10
1	15	15	15	15	15	14	13	12
2	15	15	15	15	15	15	14	14
3	12	13	13	13	13	12	12	11
4	15	15	15	15	15	15	15	15
5	15	15	15	14	13	11	11	11
6	15	15	15	13	12	10	9	8
7	15	15	15	13	9	7	6	6
8	15	15	15	15	15	15	15	15
9	15	15	15	15	15	15	15	14
10	15	15	15	14	14	14	13	12
11	15	15	15	15	15	15	14	14
12	15	15	15	15	15	15	15	15
13	15	15	15	15	15	15	15	15
14	7	9	10	12	14	15	15	14
15	14	14	14	15	15	14	13	10
16	15	14	14	14	14	14	14	14
17	14	14	14	15	15	15	15	15
18	15	15	15	15	14	13	12	12
Accuracy	95.2%	95.9%	96.3%	95.6%	93.7%	90.4%	87.4%	84.1%
Unclassifiable	0%	0%	0%	1.1%	4.1%	6.7%	10.0%	13.0%

Table 5.3: Convergence speed for the fixed-size neural network models

Class \ n	3	4	5	6	7	8	9	10
1	0.003	0.472	0.962	5.373	8.258	35.482	43.986	47.763
2	0.155	0.156	0.691	0.998	6.924	15.791	18.243	29.165
3	3.638	7.421	7.422	8.743	15.948	45.921	50.748	58.978
4	0.003	2.615	2.616	2.618	2.619	2.620	5.233	5.234
5	1.008	0.174	2.478	0.009	0.011	0.012	0.013	0.243
6	1.843	2.845	5.534	24.952	38.135	43.001	37.695	27.971
7	0.787	5.828	31.443	34.029	28.089	0.419	4.022	13.946
8	0.347	0.348	5.418	5.600	6.144	6.326	27.853	35.873
9	0.703	2.877	8.425	8.425	8.426	8.711	18.131	20.531
10	6.689	9.407	9.679	12.307	17.613	19.672	25.621	8.508
11	0.343	0.189	2.611	2.612	7.362	32.014	9.657	18.911
12	16.584	17.982	18.507	20.657	20.658	21.514	26.483	35.504
13	1.381	3.723	4.129	4.098	4.918	15.964	16.766	35.619
14	2.849	1.508	1.133	6.557	0.793	4.815	4.630	2.497
15	6.400	6.913	9.656	9.149	22.537	8.988	20.263	8.467
16	2.672	2.125	2.125	2.707	2.708	3.289	3.471	3.472
17	1.426	2.804	3.454	5.236	5.389	8.973	8.507	9.751
18	6.066	5.199	6.582	13.196	7.846	14.334	15.385	28.735
Average (s)	3.629	4.033	6.826	9.302	11.354	15.989	18.704	21.709

Class	Accuracy / 15	Speed (s)
1	15	0.003
2	15	0.155
3	13	3.358
4	15	2.617
5	13	11.356
6	15	1.843
7	15	0.787
8	15	0.347
9	15	6.251
10	15	9.388
11	15	16.099
12	15	23.559
13	15	37.935
14	9	12.585
15	15	5.005
16	14	5.007
17	14	31.647
18	15	5.199
Mean	90.0%	9.619 s

Table 5.4: Padded Model

Class	Accuracy / 15	Speed (s)
1	15	0.003
2	15	0.155
3	13	3.359
4	15	0.003
5	14	7.830
6	15	1.843
7	15	0.787
8	15	0.347
9	15	0.702
10	15	6.689
11	15	0.180
12	15	15.029
13	15	3.727
14	15	19.397
15	15	5.647
16	14	2.124
17	15	2.251
18	15	5.199
Mean	98.5%	4.182 s

Table 5.5: Combined Model

5.3.2 Discussion

Beginning with the fixed-size models, the highest accuracy achieved was 96.3 percent for Model 5. In addition to this Model 3 was also the quickest at converging with a 3.629 second convergence time on average. For larger models the speed generally dropped as expected but the accuracy also started dropping very quickly which was surprising. The lack of subsequences of matching size, described in section 5.1, seems to be the cause as one can see that the amount of observations that could not be classified at all increases quite rapidly for the larger models. Meanwhile the accuracy remained fairly high for n as small as three. Comparing Model 3 and Model 10, one sees a difference in accuracy of over ten percentage points in favor of Model 3. This stood in contrast to the results acquired in Chapter 4 where one had a 85.3 percent accuracy for Model 3 and 97.3 percent for Model 10. This confirmed the suspicions described in section 5.1 and showed the importance of investigating the full model rather than just smaller subsequences. Weighting accuracy, speed and simplicity, the model of size three seems to perform the best but it should be noted that for Class 14 it performs significantly poorer than the larger models, making Model 5 the best contender for the best model. The larger models can be rejected based on their lower accuracy, slower convergence time and more complicated structure. Therefore it was concluded that a LSTM(100) network with input size 5 was the best network for a fixed input size.

Looking at the varied-size models instead. One sees a worse accuracy for the padded model and a slightly better accuracy for the combined model of 90.0 percent and 98.5 percent respectively. The average convergence speed for the two models were 9.62 seconds and 4.18 second respectively. The first thing that stands out here is that the padded model performs very poorly, as it is less accurate, takes longer to converge and is more complex. It can therefore safely be rejected. If one instead compare model 5 to the combined model, it has approximately the same convergence speed but the combined model has a two percentage point accuracy advantage. Therefore one has to weight the benefit of that extra precision against the cost of saving over eight times as many parameters in the memory and the added complexity of constantly switching models. It should be noted, though, that both models have a 100 percent accuracy for many classes but for the classes where there is a difference in performance, it is larger. Looking at class 14, Model 3 is only correct 46 percent of the time and Model 5 66 percent of the time. Meanwhile the combined model is always correct. Thus, if the emphasis is on obtaining a more reliable classifier, the combined model

is to prefer.

Moving on to examine how the models performed for specific classes, the results were mostly in line with what was discovered from the confusion matrices in Chapter 4. It was of interest to note that the general performance is not much better than for the subsequences isolated, once one disregards the unclassifiable observations. This suggests that problematic subsequences tend to group together rather than being evenly spread out. This holds true because if they had been evenly spread out, the nature of the majority layer would've handled them, causing a higher total accuracy. It was noted that the performance for certain classes, in particular Class 6 and Class 7, dropped heavily for the larger models while others such as Class 13 remained stable. This is in line with the issues discussed in section 5.1 about certain classes being unclassifiable more often for larger models.

The two classes causing the most problems for the smaller models were Class 3 and Class 14 which also were the classes hardest to classify for the networks. As discovered in Chapter 4, this is once again because observations from Class 3 is being classified as Class 5 and observations from Class 14 as Class 15. Looking at Figure 3.4 this makes sense as those pairs resemble each other. Class 3 is clearly a subsequence of Class 5 and Class 14 and Class 15 both resemble two diagonal lines with the difference that Class 15 is a bit more jagged and Class 14 has a regular point appearing outside the diagonals. In addition to this, a single observation from Class 16 could never be classified except for the smallest fixed-size model. This was odd but was rejected as a strange outlier.

It was seen that the performance on observations from Class 14 increases with larger models, suggesting that a larger input allows it to be classified better. This is probably because of the point lying a bit on the outside of the shape in observations from Class 14 being included more frequently with a larger input. If it is excluded, it is very likely the pattern in an observation from Class 14 will be identical to the pattern in class 15. However, this is not repeated for Class 3 that performs poorly for the larger models as well. Looking at the pattern of Class 3 in Figure 3.4, one sees that Class 3 consists of a 16 data points long pattern, all included in the larger 35 point pattern of Class 5. This means that one would need at least 17 consecutive data points in order to certainly determine if a subsequence belongs to Class 5 instead of Class 3. Subsequences are rarely of this length and never in our models, thus the classes are mixed up. It is noted that, with the exception of the varied input size models, Class 5 is classified correctly. This is probably just due to it having the longer pattern, meaning there is a smaller risk that a subsequence of it is similar to the subsequences typical for Class 3. The reason the varied input size model behaves oddly in regards to Class 5 and Class 14 is probably due to the nature of training with a padded model causing some issues.

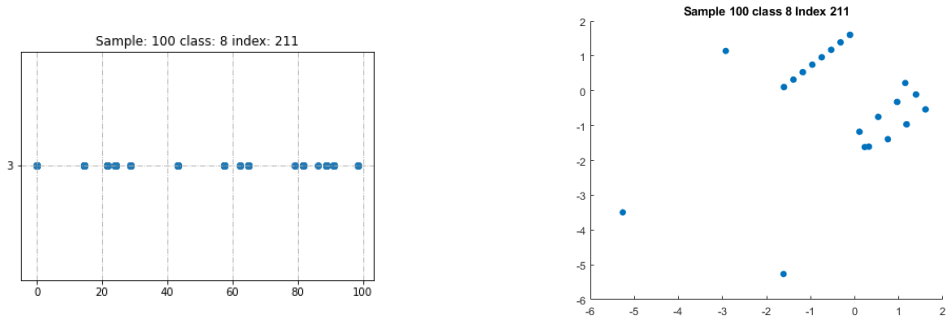
Finally a note on the accuracy. Since only 15 test observations exist for each class, the results aren't entirely reliable and using a a larger test set would've likely lead to more precise accuracy estimations. For example, it is unlikely that any class can be classified correctly 100 percent of the time in practice but with only 15 test observations any real probability over 96.7 percent will on average be rounded up to 100. However the main purpose here is to compare models to find out the best one and since no bias has been introduced, as the test data has not been used in choosing/training the models, it is likely the comparison between them still hold. Also while the results might be a bit off, it is unlikely that one would see so much worse results that the main conclusion would not hold. A top accuracy of 95-99 percent can probably be assumed which means that classifying radar signals using neural networks definitely is possible. More test data would have been welcomed for the smaller classes but from the issues described in Chapters 3, those types were relatively rare and quite hard to acquire. However, it could be a project for future studies.

5.4 Handling the trash observations

In this part, the purpose is to study how the models behave for the observations in the trash class. Since one single classification is not really of interest but rather how the observations are classified over time, the more simple two-layer method is used.

Many of the observations in the the trash class were presumed to be observations from one of the 18 classes but disturbed by a single or a few outlier points. These outliers caused the whole difference-vector to be slightly shifted post-normalization. The hope was therefore that the model would be able to handle this and still classify them correctly for the subsequences outside the outlier points. In addition to this, it was suspected that a lot of the trash observations were the result of two or more types of radar being detected simultaneously, resulting in an odd plot in the 2D-perspective. If this was the case it was unlikely that the two radar signals would interfere during the exact same timeframe, and thus the subsequences should still belong to just one class. This should in theory be detectable in the realtime plots as classifications of two or more classes in intervals matched by when the corresponding signal is detectable by the receiver.

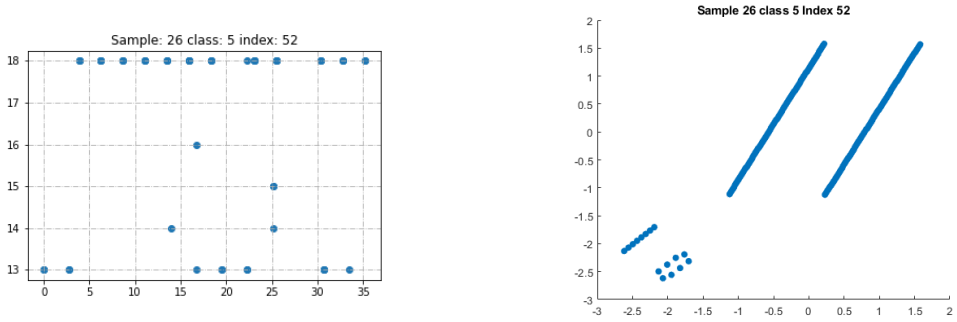
In order to visualize this, a combination of the original 2D-perspective from Chapter 3 and the method developed earlier in the current section were used. First the two layer method using the Model 5 from the previous part was applied for each observation from the the trash class. The resulting time frame plot was then plotted next to the matching 2D-plot. Thus one could easily look at the classified classes in the timeseries plot and compare their corresponding 2D-plots, seen in Figure 3.4, to the observations 2D-plot to see what conclusions could be drawn. Note that only the first 250 points were plotted in order to save time. This means that certain outlier subsequences might not be visible as it is impossible to determine from what part of the sequence they come from just by looking at the 2D-plot. However the relevant information can still be gathered from a shorter part of the whole sequence.



(a) (b)

Figure 5.7: An observation that seems to be from Class 3 with four to five outlier points

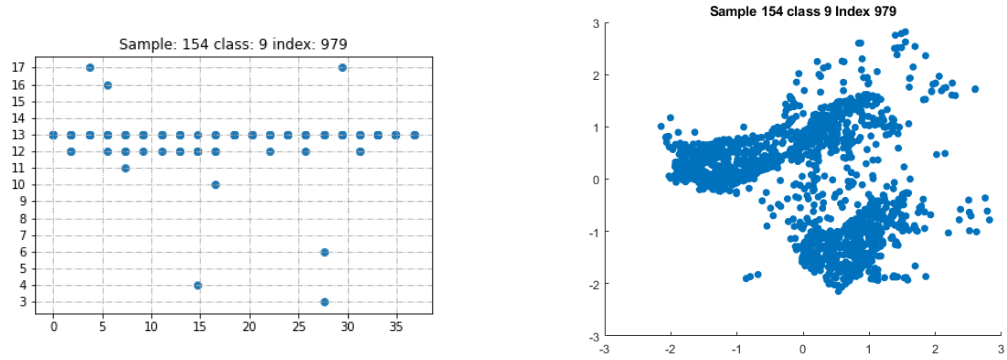
The first observation that this was done for was an observation that seems to be from Class 3 but slightly disturbed by four to five outliers. The results are shown in figure 5.7 where one can see that it indeed is classified as Class 3. This makes sense because even though outliers exist, most subsequences will still be clear examples from Class 3. This indicates that the slight skew from the normalization did not impact the results. This was further confirmed by testing out other observations looking like they belong to one of the existing classes but with a couple of outliers. These results meant that even if an observation has a couple of outliers, it can still be classified correctly. This was of interest because the test set in the previous section only contains cleaned observations and no information on how well they handled observations with outliers had been obtained. Since outliers can appear quite frequently, this means that one does not necessarily have to involve any sort of outlier-removing preprocessing before sending the observation through the model in order to achieve results of the quality seen in the previous section.
The second situation that was to be tested was an observation that seemed to be the result of two



(a) (b)

Figure 5.8: An observation that seems to be a mix of two different observations from two different classes

signals interfering with each other. In Figure 5.8, one can see an observation that seems to be a combination of Class 14 or Class 15 and Class 3. However, it is for the most part classified as Class 13 and Class 18. This seemed odd at first since the individual subsequences were presumed to still be part of their respective classes. It can also be noticed that, due to the normalization, their positions are quite skewed in regards to how individual instances of Class 3 and Class 14/Class 15 look like. Since the differences appear spread out and not really resembling anything, the network classifies them as Class 13 and Class 18 which are the most spread out classes available. Looking at other observations with similar behaviour, one could see the same pattern of classifications as Class 13 and Class 18, supporting this hypothesis. While this is something that is quite hard to get around, it could potentially be solved by normalizing each subsequence individually instead of the whole vector at once, But that also has the drawback of making non-disturbed observations more prone to noise and being more spread out. Due to time constraints this was not explored further but it is perhaps something that can be attempted in the future.



(a) (b)

Figure 5.9: Visualization of an observation that seems to not resemble any other observation

Finally observations that were of a more chaotic pattern, not really resembling any established class, were observed. These were assumed to either belong to one class and be heavily disturbed by noise, or belonging to two or more classes interfering with each others sub-sequences, resulting in strange patterns. An example of this is shown in Figure 5.9 which is classified as mostly Class 13 and Class 12. It is known from looking at other plots that both chaotic and mixed up plots are classified as class 13 due to the nature of class 13 being very spread out. So if one decides to ignore the classifications as Class 13, one could potentially see this observation as a heavily disturbed observation from Class 12. However, due to the issues pointed out in the previous paragraph, this is not reliable. It is clear that further work is needed in order to make any clearer predictions of

observations such as these.

Chapter 6

Discussion and Conclusion

Looking back at the problem formulation, we believe that we can conclude that artificial neural networks can classify different types of radar signals. In addition to this it can also be done with a very high accuracy after just a couple of seconds. The exact preciseness of the speed and accuracy is not certain due to the limited test data but it is unlikely to not be higher than 95 percent within a couple of seconds.

An interesting comparison with the traditional 2D-perspective used when classifying radar signals and the neural networks used in this part is the ability to classify small subsequences. It is unlikely that just plotting four to five data points would allow a person to make a classification with over 95 percent certainty by just looking at the plot. However the neural network models managed to do this. This is not only an indication of the power of neural networks but also reveals that there is potentially a lot more information stored in a subsequence than what is seen from a 2D-plot of the time shifted differences, meaning more than two dimensions contain useful information.

A few comments on issues and what could have been handled differently. The most clear drawback is that a LSTM(100) network has been used for every input size, both fixed and varied. It is very unlikely that the same model performs the best for different input sizes and therefore it is likely that potentially even better results could have been achieved if the optimal model was found for each individual input size instead. In addition to this, it is possible that a less accurate but much simpler model for isolated subsequences could still classify with comparable accuracy once applied on the full sequences. The reason this was not fully investigated was mostly time constraints. As training, testing and evaluating different neural networks take a lot of time and it would have required eight times as much testing to find the optimal model for every input size, it was deemed unfeasible. Furthermore the results were already sufficiently high and it was unlikely that a much higher accuracy could be reached. The biggest room for improvement was therefore concluded to be in the scope of model simplicity. Training for longer time periods, tuning parameters and testing it out on full observations could potentially lead to a less complex model performing on similar levels.

Another issue was how the normalization of data was handled. The networks found in Chapter 4 were trained and tested on normalized data which did not present any problems. However when it came to classifying observations in real time as was explored in Chapter 5, this became a problem. As data appears in real time it is not normalized and therefore it must first be normalized before it can be handled by the network models. This can be handled by constantly normalize incoming data using the existing data points but this results in a normalization likely to differ from the one used for training early on as only a few data points are available. This will in turn increase the risk for poor classifications during the early stages of the signal.

Due to reasons of simplification, this perspective has been largely ignored during the scope of this report and instead the observations have been used as pre-normalized. There is a risk that this could cause some problems in the accuracy during the early stages of the observations if one

uses the models in real time. However due to the fact that the observations come from the same source, the normalization will become increasingly similar as more data is acquired. This was investigated and it was found that after a couple of seconds of data this no longer presented a problem. Since the results acquired were said to converge after about four seconds, this was assumed not to affect the results and the conclusions of this thesis in a significant manner. Still it was attempted to redo the models on unnormalized data to get around the problem but this resulted in very poor results and the idea was abandoned. It should also be noted that most of the normalization deals with handling the mode the transmitter is in. If one has some prior knowledge about the position of the object one is trying to classify, it is possible that the mode can be determined beforehand and the normalization process can be done quicker even for a smaller amount of data points.

6.1 Future Work

Since the network managed to classify the broad division of radar signal classes given in this report, it would be interesting to study how well it could classify more closely related types of radar signals. Looking at Class 14 and Class 15 which have a finer difference than the other classes, the best model still managed to separate them. This indicates that artificial neural networks might be capable of classifying more closely-related types of radar signals than have been given here. Another potential interesting thing to do could be to use amplitude or frequency instead of the time of arrival data to see if as good of a classification could be made.

As was mentioned in the possible issues, it is likely that simpler models could be acquired with a more thorough investigation. In particular it is likely that training feed-forward networks for a longer time period could result in possibly equal results. In addition to this models that perform worse for specific subsequences could still potentially perform comparably well on longer observations. It could therefore be of interest to try to find the absolutely best model, reaching the best result on all the performance measures.

Finally another possible field of future work could be to obtain more data from (to the model) unknown radar types and see how it classifies them over real time. This would also be of interest to do for known types but heavily disturbed as was attempted in Section 5.4 but with the difference that one utilizes observations with a known cause of disturbance/error. This could potentially allow the observer to connect the models behaviour to a certain type of error or disturbance.

Bibliography

[1] N. S. Altman. An introduction to kernel and nearest neighbor nonparametric regression. https://ecommons.cornell.edu/bitstream/handle/1813/31637/BU-1065-MA.pdf, 1992.

[2] J. Brownlee. "why one-hot encode data in machine learning?". https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/, 2017.

[3] Y.-H. K. C.-C. C. M.-H. Y. I.-H. Chen, Yung-Yao; Lin. Design and implementation of cloud analytics-assisted smart power meters considering advanced artificial intelligence as edge analytics in demand-side management for smart homes. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6539684, 2019.

[4] F. Chollet. *Deep Learning with Python*. Manning, Nov. 2017.

[5] B. C. Csáji. *Approximation with Artificial Neural Networks; Faculty of Sciences*. Eötvös Loránd University, Hungary, 2001.

[6] H. Draper, Norman R.; Smith. *Applied Regression Analysis (3rd ed.)*. Wiley, 1998.

[7] R. A. Fisher. The use of multiple measurements in taxonomic problems. https://digital.library.adelaide.edu.au/dspace/bitstream/2440/15227/1/138.pdf, 1936.

[8] J. L. Fix, Evelyn; Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. https://apps.dtic.mil/dtic/tr/fulltext/u2/a800276.pdf, 1951.

[9] R. E. Freund, Y.; Schapire. "large margin classification using the perceptron algorithm". http://cseweb.ucsd.edu/~yfreund/papers/LargeMarginsUsingPerceptron.pdf, 1999.

[10] N. N. S. J. Gers, Felix A.; Schraudolph. "learning precise timing with lstm recurrent networks". https://www.jmlr.org/papers/volume3/gers02a/gers02a.pdf, 2002.

[11] Y. C. A. Goodfellow, Ian; Bengio. *6.2.2.3 Softmax Units for Multinoulli Output Distributions"*. MIT Press. pp. 180–184, 2016.

[12] Y. C. A. Goodfellow, Ian; Bengio. *"6.5 Back-Propagation and Other Differentiation Algorithms"*. MIT Press, 2016.

[13] R. Guillaume. "uncertainty in radar emitter classification and clustering". https://tel.archives-ouvertes.fr/tel-02275817/document, 2019.

[14] M. V. Guy. *"Airborne Pulsed Doppler Radar"*. Artech House, 1988.

[15] R. F. J. Hastie, Trevor. Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[16] K. Hinkelmann. "neural networks, p. 7" (pdf). university of applied sciences northwestern switzerland. http://didattica.cs.unicam.it/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay_1718:ke-11_neural_networks.pdf.

[17] N. K. A. S.-I. S. R. R. Hinton, Geoffrey E.; Srivastava. "improving neural networks by preventing co-adaptation of feature detectors". https://arxiv.org/abs/1207.0580, 2012.

[18] Keras-Team. Keras. https://github.com/keras-team/keras, 2021.

[19] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980, 2017.

[20] Q. S. Kingsley Simon. *"Understanding Radar Systems"*. McGraw-Hill Book Company, 1992.

[21] C. Lemaréchal. "cauchy and the gradient method". https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf, 2012.

[22] N. Malik. "artificial neural networks and their applications". https://www.researchgate.net/publication/1958135_Artificial_Neural_Networks_and_their_Applications, 2005.

[23] D. Samuel. *"A thorough review on the current advance of neural network structures"*. Annual Reviews in Control. 14, 2019.

[24] J. Schmidhuber. Deep learning in neural networks: An overview. https://arxiv.org/abs/1404.7828v4, 2014.

[25] A. Tealab. Time series forecasting using artificial neural networks methodologies: A systematic review. https://www.sciencedirect.com/science/article/pii/S2314728817300715, 2018.

[26] D. J. L. A. I. Tetko, I. V.; Livingstone. Neural network studies. 1. comparison of overfitting and overtraining. http://www.vcclab.org/articles/jcics-overtraining.pdf, 1995.

[27] B. D. Venables, W. N.; Ripley. *Modern Applied Statistics with S (4th ed.)*. Springer Verlag, 2017.

[28] B. Wilson. The machine learning dictionary. http://www.cse.unsw.edu.au/~billw/mldict.html, 1998-2012.

[29] A. Zell. *"chapter 5.2". Simulation neuronaler Netze [Simulation of Neural Networks] (in German) (1st ed.)*. Addison-Wesley, 2003.

[30] H. T. R. S. Zhen Wang, Ryusuke Egawa. Auto-tuning of hyperparameters of machine learning models. https://www.researchgate.net/publication/329269354_Auto-tuning_of_Hyperparameters_of_Machine_Learning_Models, 2018.

Appendix A

Loss and accuracy for MLP model

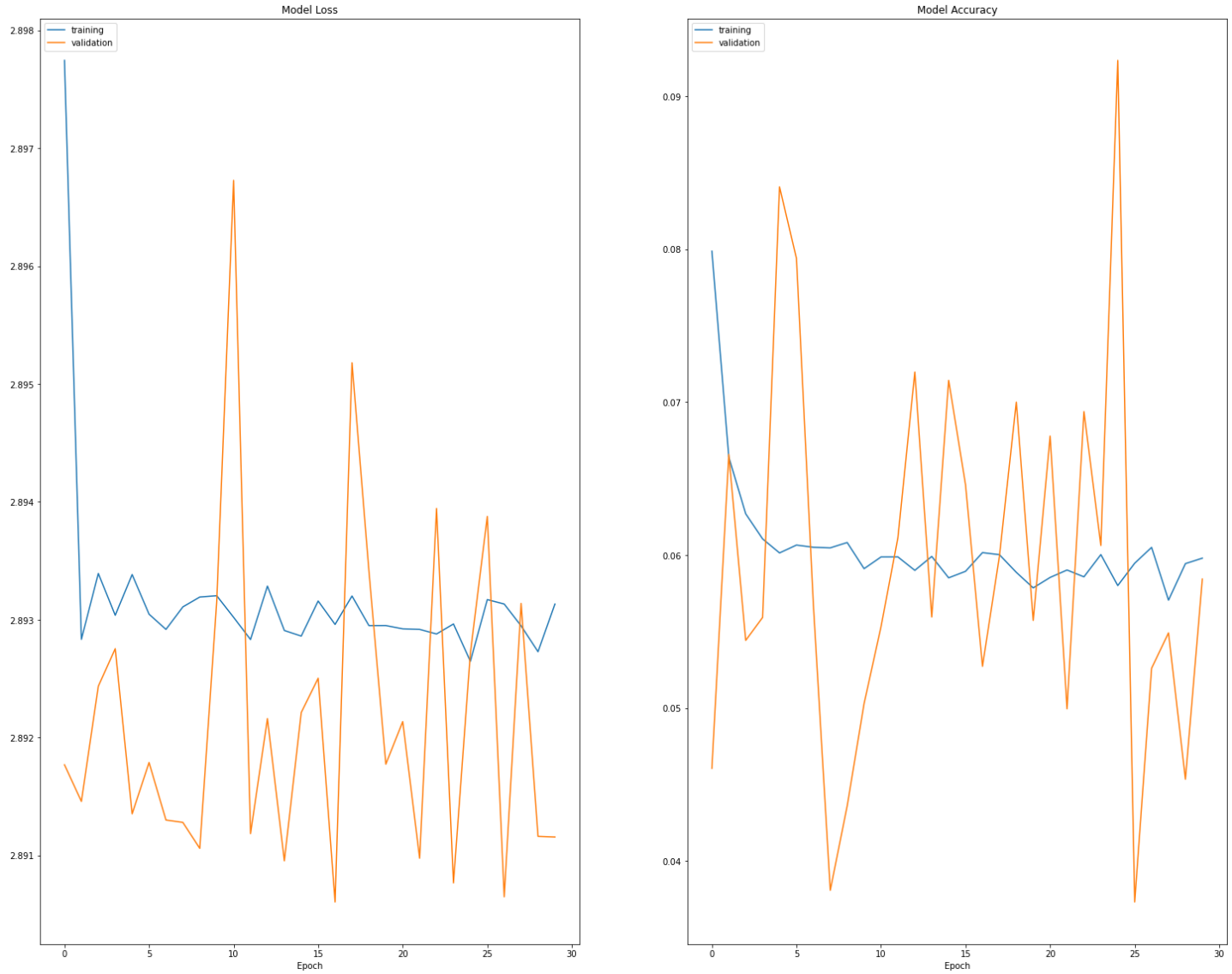


Figure A.1: Loss and accuracy plots for a network with 3 hidden layers, all with 1,024 nodes each.

Appendix B

Loss and accuracy for LSTM model

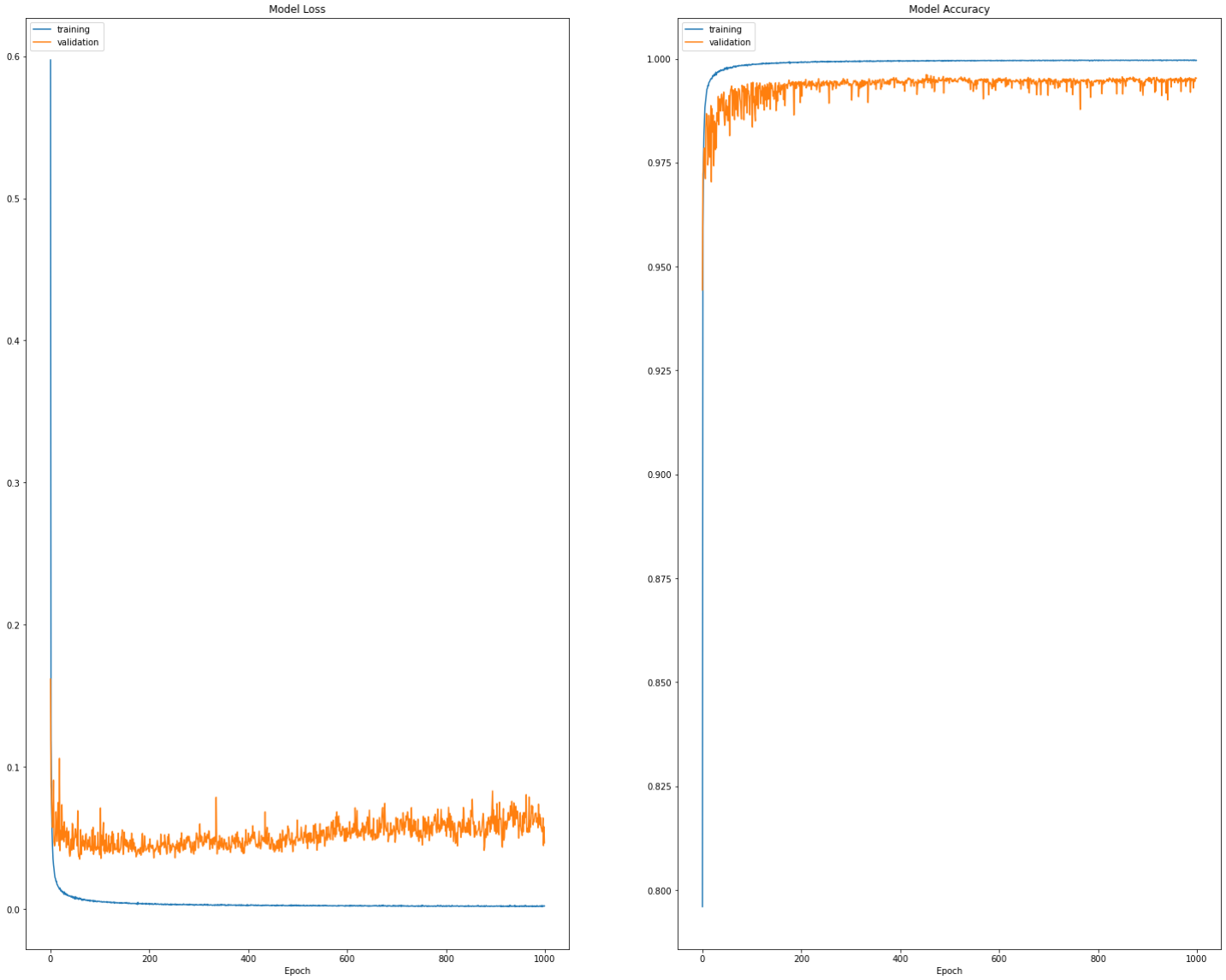


Figure B.1: Loss and accuracy plots for a network with one hidden LSTM layer with 300 nodes.

MATEMATIKCENTRUM | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2021-04-14

EXAMENSARBETE AI-based Classification of Radar Signals

AI-baserad klassificering av radarsignaler

STUDENTER Ludwig Hollmann, Max Fors Joki

HANDLEDARE Gunnar Hillerström (FOI), Andreas Jakobsson (LTH)

EXAMINATOR Niels Christian Overgaard (LTH)

AI-baserad klassificering av radarsignaler

POPULÄRVETENSKAPLIG SAMMANFATTNING **Ludwig Hollmann, Max Fors Joki**

I militära sammanhang är det av intresse att kunna identifiera militärfordon som försöker hålla sig dolda. Ett sätt att göra detta är att klassificera de sekvenser av radarsignaler fordonen skickar ut. I det här arbetet så har vi undersökt hur väl artificiell intelligens klarar av det.

Radar är teknik som används av bland annat flygplan och fartyg för att identifiera sin omgivning. Detta görs genom att pulser av radiovågor skickas ut i en viss riktning och reflekteras tillbaka till sändaren av objekt som de stöter på. För att radiovågorna ska ha tid att nå omkringliggande objekt och färdas tillbaka så måste det gå en viss tid efter att pulsen skickats ut till att nästa puls sänds. Tidsintervallen mellan pulserna varierar ofta enligt en förbestämd sekvens som är unik för just den typen av radar. Denna sekvens längd skiljer sig mellan radartyper och upprepas medan radarn är påslagen.

I olika militära scenarion då fartyg och flygplan ofta vill hålla sig dolda är det av intresse att kunna identifiera dem, så väl med så hög precision som möjligt som så snabbt som möjligt. Detta kan göras genom att snappa upp deras radarpulser och återskapa den underliggande sekvensen av utsändningstider. Denna sekvens kan sedan klassificeras som en känd typ av radar av en erfaren operatör, vilket i sin tur gör att typen av fordon kan identifieras. Rekonstruktionen av sekvensen försvåras av att sändaren roterar och därför mottas endast en liten andel av alla pulser åt gången. Detta innebär att mottagaren får in osammanhängande delsekvenser från varierande delar av den större sekvensen, istället för hela sekvensen i ordning. Det krävs alltså av operatören att han använder sig av olika slags algoritmer för att kunna koppla dessa delsekvenser till en känd sekvens.

I detta arbete så har vi använt artificiell intelligens för att lösa det här problemet. För att göra detta så har riktig uppmätt radardata på formen beskriven ovan använts. Denna data består av ett stort antal mätningar om tio minuter var från 18 olika typer av radar. Den modell som använts är ett artificiellt neuralt nätverk. Detta nätverk har lärt sig att känna igen strukturer i den stora mängd delsekvenser som finns direkt, för att sedan kunna klassificera dem som en känd typ av radar. Detta innebär att vår modell kan göra klassificeringar i realtid direkt ett fåtal pulser kommer in till mottagaren. Denna klassificering blir även mer exakt allt efter att fler pulser inkommer.

De resultat vi fann var att även ett relativt simpelt neuralt nätverk kan nå en hög precision snabbt. Vår bästa modell kunde klassificera 97% av alla mätningar korrekt inom ett par sekunder. Vissa typer kunde i samtliga fall klassificeras korrekt medan andra var svårare och blev felklassificerade i ca 15% av fallen. Dessa resultat antas vara bra då det är tveksamt att det går att nå en så hög precision med så lite data ens manuellt, där fördelen med automatisering inte heller finns.

Vår slutsats är alltså att artificiell intelligens klarar av att klassificera denna typ av radar och därför kan vara till stor hjälp vid identifiering av dolda militärfordon.