Universitetet
i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study program/specialization: Information Technology - Automation and Signal Processing	Spring semester, 2017 Open / ~~Confidential~~
Author: Daniel Eidsvåg	Daniel Eidsvåg (signature author)

Instructor: Professor Trygve Eftestøl

Supervisor(s): Professor Trygve Eftestøl

Title of Master's Thesis:
Rhythm interpretation using deep learning neural networks
Norwegian title:
Rytmetolkning ved bruk av dype nevrale nettverk

ECTS: 30

Subject headings: Neural Networks, Feed forward, Convolution, Recurrent, LSTM, Cardiac arrest, cardiopulmonary resuscitation, cardiac rhythm classification.	Pages: 50 + attachments/other: 10 + embedded file Stavanger, 15th of June/2017 Date/year

Rhythm interpretation using deep learning neural networks

Daniel Eidsvåg

June 2017

MASTER'S THESIS

Faculty of Science and Technology

Department of Electrical Engineering and Computer Science

University of Stavanger

Supervisor: Professor Trygve Eftestøl

Abstract

Out-of-hospital cardiac arrest (OHCA) is a leading cause of death in the industrialized world, with an estimated annual incidence that varies by 52.5 (in Asia), 86.4 (in Europe), 98.1 (in North America), and 111.9 (in Australia) per 100,000 person-years. Lethal ventricular arrhythmias are the most frequent causes of OHCA. A defibrillation shock is an effective treatment, and early defibrillation is one of the key factors in survival from OHCA. However, chest compressions, ventilations, and drug play a key role in the treatment of cardiac arrest. Under resuscitation, automated external defibrillator (AED) require peri-shock pauses to analyze for a shockable rhythm. This peri-shock are associated with a decrease in survival to hospital discharge.

The objective of this thesis is to determine if different artificial neural network (ANN) structures can be used as a classifier, to determine the underlying heart rhythm under chest compressions to remove peri-shock pauses during cardiac arrest. The analysis is conducted from data obtained from 394 OHCA patients, where two datasets were used. Both containing 3-second segments with clinical rhythm annotations resulting in 2446 and 422415 segments.

Results in this thesis suggest that there is no clear best method in the different neural network methods for ECG data. However, FNN demonstrates the most promising results with an accuracy of 52.30%. This result emphasizes the problem regarding classification of ECG data with compression artifacts.

Preface

This thesis was written at the Department of Electrical Engineering and Computer Science at the University of Stavanger (UiS). I would like to thank my supervisor Prof. Trygve Eftestøl at UiS for his valuable advice and feedback. As well to Prof. Kjertsi Engan and Ali Bahrami Rad for valuable inputs. I would also like to thank the Department of Electrical Engineering and Computer Science at UiS. For investing in a new GPU-based computing facility which gives the opportunity to exhaustive explore deep learning and Theodor Iversdal for installation and operation on the server. Finally, I would like to thank Hanne Felt Eie for her support throughout the semester.

Contents

Chapter 1

Introduction

Out-of-hospital cardiac arrest (OHCA) is a leading cause of death in the industrialized world, with an estimated annual incidence that varies with 52.5 (in Asia) 86.4 (in Europe), 98.1 (in North America), and 111.9 (in Australia) per 100,000 person-years[18]. Lethal ventricular arrhythmias are the most frequent causes of OHCA. A defibrillation shock is an effective treatment, and early defibrillation is one of the key factors in survival from OHCA[17]. However, chest compressions, ventilations, and drug play a key role in the treatment of cardiac arrest[35, 40].

It has been shown that interruptions of compressions, such as ventilations during *Cardiopulmonary resuscitation* (CPR) or rhythm analysis when using an *automated external defibrillator* (AED), have a negative effect on survival rate [4]. Coronary perfusion pressure[2] requires a period of "rebuilding" to obtain the same pressure achieved before the interruption and drops rapidly when CPR are stopped [30]. The blood flow is inadequate during these interruptions, and perfusion pressure must be rebuilt after each pause in CPR [24]. When this occurs prior to defibrillation, the heart of no longer in the best possible state to receive defibrillation[11]. In 2011 a study was conducted to see the effect on peri-shock pauses[3] from OHCA events. The article states: "In patients with cardiac arrest presenting in a shockable rhythm, longer peri-shock and pre-shock pauses were independently associated with a decrease in survival to hospital discharge. The impact of pre-shock pause on survival suggests that refinement of automatic defibrillator software and paramedic education to minimize pre-shock pause delays may have a significant impact on survival"[6]. Peri-shock pauses are the motivation for this thesis, to be able to classify heart beat rhythms with compressions artifact and remove the analytic window using an AED to improve survival rate in OHCA.

[2]The pressure gradient that drives coronary blood pressure, it's a part of normal blood pressure that is specifically responsible for coronary blood flow[44].

[3]Pauses in chest compressions before and after defibrillator shock.

1.1 Prehospital scenario

Cardiac arrest may be witnessed by a bystander, and a crucial factor for survival is if the bystander knows the procedures of basic life support. If no life signs are verified, then CPR are a crucial factor until paramedics arrive at the scene. They assess the situation and provides necessary life support by clearing airways, continuing CPR, and give drug therapy and defibrillation. Finally, the patient is transported to a nearby hospital for further intensive care. If the patient is successfully resuscitated then the patient is admitted to the hospital for further observation and is finally conscripted from the hospital with a normal life function.

1.2 Thesis outline

Chapter 2 - Background:
This chapter is divided into a physiology and technological part. The physiology part gives the background for medical aspects of ECG, heart's function, and rhythms. The technological part presents the concepts of different neural network methods and how networks are trained in detail. In addition to a brief presentation how synthetic data can be generated.

Chapter 3 - Materials and methods:
Discusses the data material and presents the algorithm used to create the datasets. A description of the implementation of methods and the outline of preliminary testing and hyperparameters. In addition to the evaluation of the neural networks are presented.

Chapter 4 - Results:
Presents the best results generated from the different neural network models in detail.

Chapter 5 - Discussion:
This chapter contains a discussion of the material, performance of the classifiers and sources of misclassification.

Chapter 6 - Conclusion and future work:
This chapter contains a conclusion of the analyzed methods and directions for future research.

Appendix A - Performance metric and contingency tables:
A detailed presentation of the performance metric and contingency tables for the best results for each method.

Appendix B - Program files:

Matlab code:

A list of devised scripts and functions and their behavior are presented. All code described can be found in the embedded file *matlab.zip*

Python code:

A list of devised scripts and methods with their behavior are presented. All code described can be found in the embedded file *python.zip*. In addition to a list of prerequisites to be able to run the implemented code and information about the machine learning library used.

Chapter 2

Background

This chapter is divided into a physiology and a technological part, where first it describes ECG signals and the heart function and the rhythms that are evaluated in this thesis. Then, into the depth of artificial neural networks and the different methods available. There will also be explained how training a network is conducted with back-propagation. In addition to how synthetic data can be generated to deal with imbalanced dataset problem.

2.1 Physiology

2.1.1 ECG signals

An electrocardiogram (ECG) is a recording of the electrical activity in the heart. ECG signals can provide valuable information about abnormalities in the heart function as muscle damage or dangerous rhythms. ECG is therefore used for surveillance of hospital patients, health checks and in the stage before the patients arrive at the hospital.

2.1.1.1 The heart's function

The heart is a muscular organ the size of a large fist whose primary function is to pump oxygen-rich blood throughout the body. Its anatomy is divided into four chambers, two upper chambers called the atria, where the blood enters. Two lower chambers called ventricles, where the blood is forced into further circulation, see figure 2.1(a). A healthy heart contraction occurs in five stages: The sinoatrial node (SA) located in the wall of the right atria as seen in figure 2.1(b), emits an electrical impulse that stimulates the closest muscle cells in the atria to undergo depolarization such that they contract. This propagates through the conduction system, it is then delayed at the atrioventricular node (AV) which lets the blood in the atriums to be emptied into the chambers. The impulse is then passed on to the ventricles resulting in a contraction and the blood is transmitted to the circulatory system of the body and lung[41].

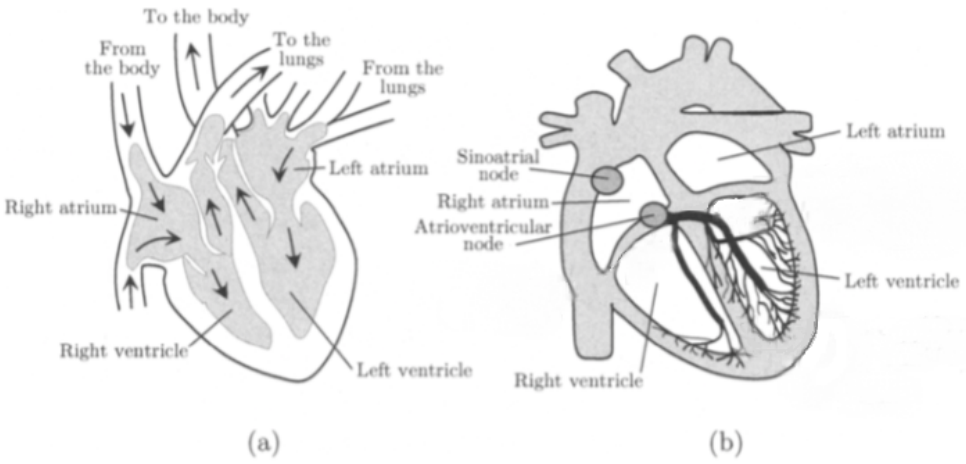

Figure 2.1: Cross section of the heart, where a) indicates the direction of the blood flow in and out of the heart and b) its electrical conduction system. (Source: Sørnmo and Laguna, 2005 [41])

2.1.1.2 Normal heart rhythm

In section 2.1.1.1 it was described how the heart function. Further, in this section, it will be explained how the heart function can be identified in an ECG as illustrated in figure 2.2. The P wave represents the atrial depolarization which is the sequential contraction of the right and left atria. The PQ interval corresponds to the time where the impulse is emitted from the sinoatrial node to the atrioventricular node. The QRS complex represents the simultaneous contraction of the right and left ventricles. The ST segment consists of the connection of QRS complex and the T-wave, during this time the ventricles are depolarized. The T wave represents the repolarization of the ventricles and the QT interval resulting in the time from the QRS complex, to the end of the T wave[41].

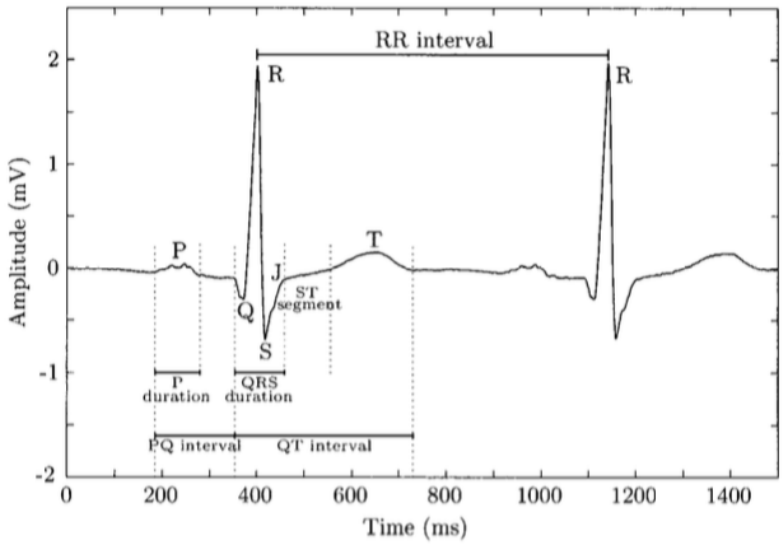

Figure 2.2: Wave definitions of a normal sinus rhythm with the important wave durations and intervals. (Source: Sørnmo and Laguna, 2005 [41])

It is important to remember that there are a wide range sinus rhythms depending on how the leads are placed on the body. In a standard ECG measurement, 10 leads[1] are placed in a standardized position on the body surface and creates 12 measurements that are used to visualize the electric activity in the heart from different angles. However, this thesis only evaluates data originating from a two lead measurement, measured from an AED.

2.1.1.3 Abnormal heart rhythms

A normal sinus rhythm originates from the SA node and is typical between 50 and 100 beats per minute (bpm) at rest. Any deviation of a normal sinus rhythm is called arrhythmia. There are many different arrhythmias, such as premature beats, atrial and ventricular arrhythmias[2]. Any rhythm below 50 bpm is referred as bradycardia and rhythms above 100 bpm is classified as tachycardia.

Ventricular arrhythmias are the most common arrhythmias and are a result of the reentry[3] mechanism. For this type of arrhythmia, we can have ventricular tachycardia and ventricular fibrillation. Ventricular tachycardia occurs at a rate over 120 bpm and consist of beats with an increased QRS width and large amplitude. VT is often the initiating rhythm of cardiac arrest, and the chance of successful resuscitation is very good. Ventricular fibrillation is a totally disorganized rhythm which the ventricles cease to depolarize in an orderly fashion. There is no mechanical activity in the heart, the heart is in a fibrillating state. As a result, the heart undergoing ventricular fibrillation cannot deliver oxygenated blood to the brain.

In this study, five different heart rhythms are evaluated; *Ventricular fibrillation* (VF), *Ventricular tachycardia* (VT), *Asystole* (AS), *Pulseless electrical activity* (PEA/PE), and *Pulse generating rhythm* (PR). With compression artifacts, the heart rhythms may be abbreviated as CVF, CVT, CAS, CPE, and CPR respectively. In figure 2.4 and 2.3, there is an example of 3-second segments with the corresponding classes with and without compression artifacts respectively. Note that the segments in figures 2.3 and 2.4 are taken from the same position at the corresponding episode, i.e. the segments without artifacts are extracted prior to the compression. One should notice that these segments vary on different episodes, with different amplitudes, bradycardia- tachycardia rhythms, and artifact noise, this will be explained in further detail in section 3.1.

[1] "The difference between a pair of electrodes is referred to as a lead."[41]
[2] For the interested reader see chapter 6 in [41]
[3] The electrical impulse conduction is not completing the normal circuit, but loops back upon itself

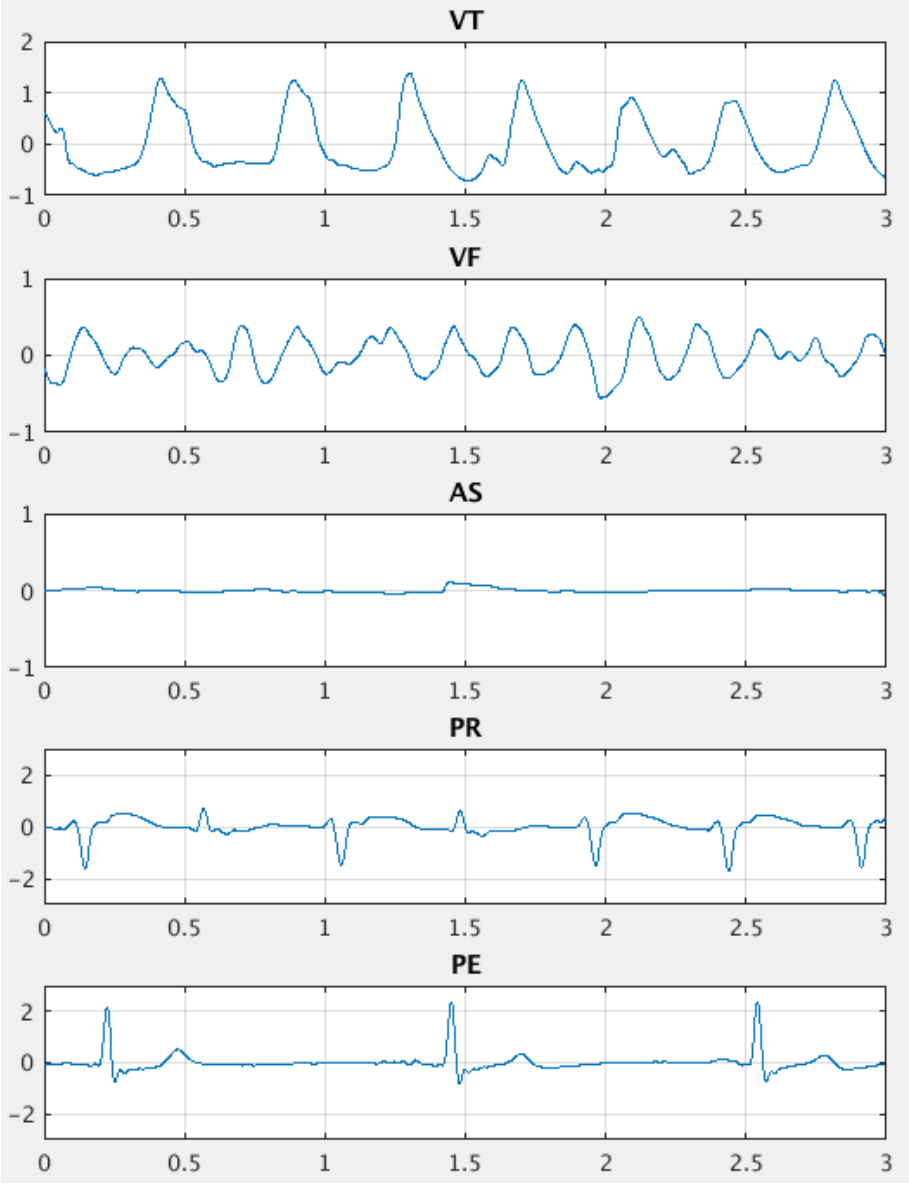

Figure 2.3: The five classes, VF, VT, AS, PR, PE without compression artifacts

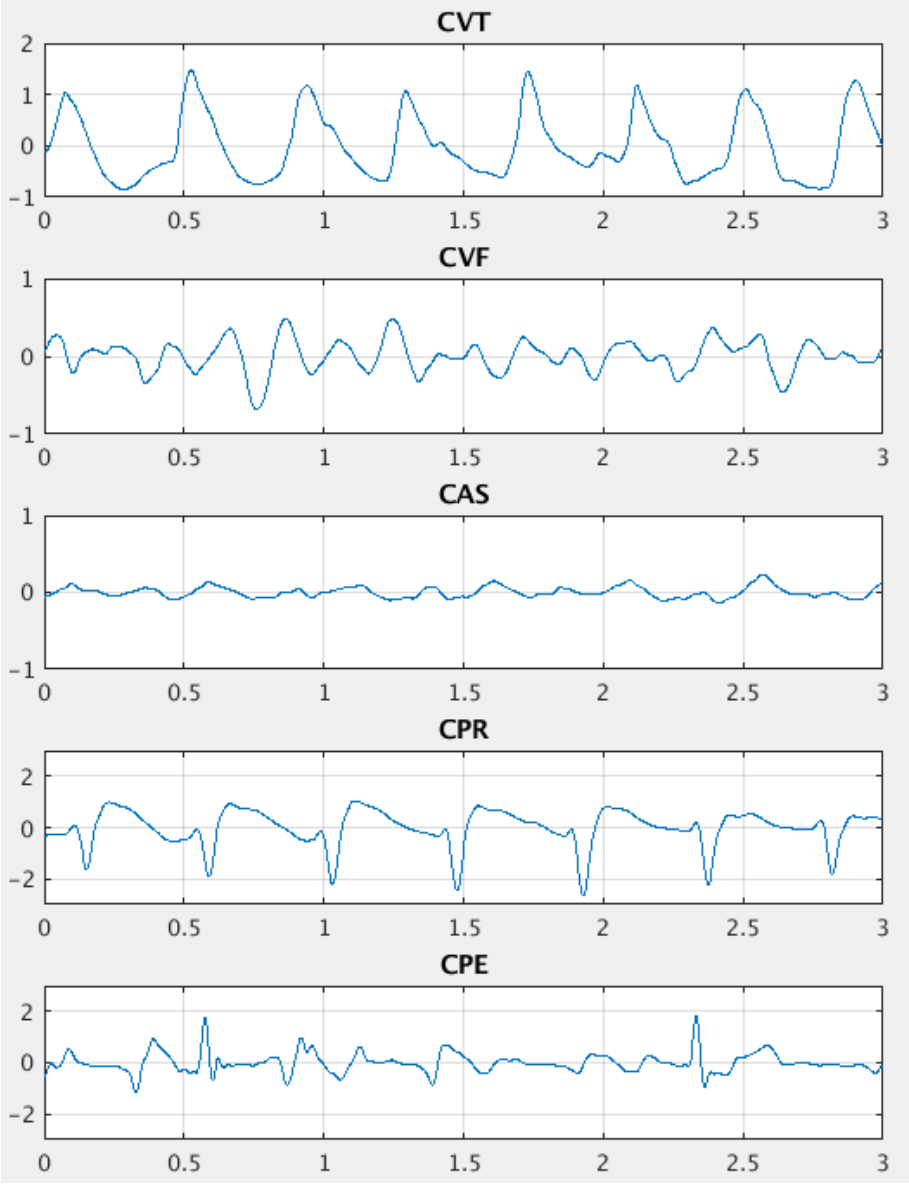

Figure 2.4: Corresponding segments, CVF, CVT, CAS, CPR and CPE with compression verified using data from compression depth and/or impedance. See Appendix B.1 for details.

2.2 Technology

2.2.1 Artificial neural networks

In machine learning, artificial neural network or simply neural network (NN) is a computational model for approximating mathematical functions. Which is based on a large connection of *units* referred as *neurons*, which is loosely inspired by the biology of the brain. Typically neurons are connected in layers and emit an activation signal to the following connected neurons. If a neuron receives enough input, the neuron will become activated and the signal travels to other interconnected neurons. These networks can be trained to learn arbitrary complex problems,

such as determine specific voices, objects in an image and so on.

2.2.1.1 Feed forward neural network

Figure 2.5 shows a simple three-layer feed forward neural network (FNN), it consists of an input-, hidden[1]- and an output layer. The network is made up of neurons that are interconnected by modifiable weights **w** and is represented in figure 2.5 by the links between layers. In a feed forward neural network, the data moves in only one direction from input to output without any form of cycles, hence the name feed forward.

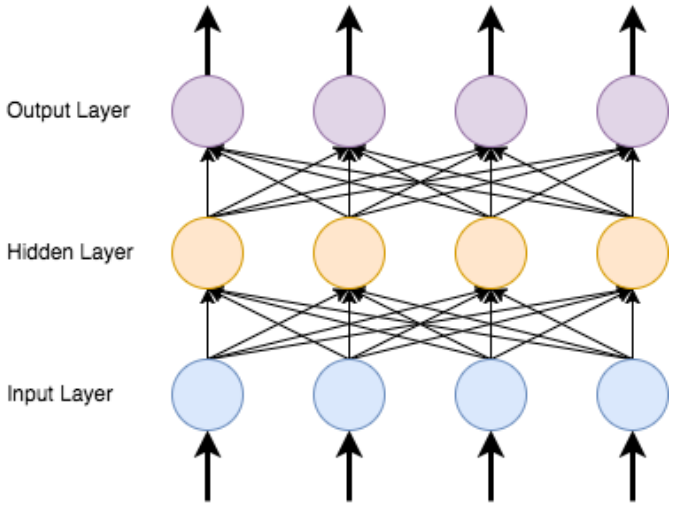

Figure 2.5: A simple example of a FNN structure.

Looking further into the details of feed forward neural network the input vector is presented to the input layer, and the output of each input unit equals the corresponding component in the vector. Each hidden unit computes the weighted sum of its inputs and uses a *activation function* which is denoted as $f(\cdot)$. There are many different activation functions such as sigmoid, which is the traditional function and are defined as:

$$f(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.1}$$

This activation function limits the output of a neuron between 0 and 1. To be able to better fit the data, a bias unit is connected to each unit, as displayed in a more detailed version of FNN in figure 2.6. In equation (2.2) we define *net* which is prior to the activation[9], and by denoting the feature value $x_0 = 1$, and append the bias as w_{j0}[2], for our hidden layer we can then write:

$$net_j = \sum_{i=1}^{d} w_{ji}x_i + w_{j0} = \sum_{i=0}^{d} w_{ji}x_i \equiv \mathbf{w}_j^t\mathbf{x} \tag{2.2}$$

[1]Everything in between the input layer and output layer is referred as a hidden layer.
[2]For equation 2.3 this would correspond to w_{k0}

where the subscript i indexes units in the input layer with d as of dimension of the data, j are the hidden unit and w_{ji} denotes the input-to-hidden layer weights at the hidden unit j. The output of a hidden unit is defined by equation (2.5), and for our output layer we can write:

$$net_k = \sum_{j=1}^{n_H} w_{kj} y_j + w_{k0} = \sum_{j=0}^{n_H} w_{kj} y_j \equiv \mathbf{w}_k^t \mathbf{y} \tag{2.3}$$

where the subscript k indexes unit in the output layer and n_H denotes the number of hidden units. k specifies which output unit and w_{kj} denotes the hidden-to-output layer weights at the output unit k. This can be written compactly as:

$$z_k = f\left(\sum_{j=1}^{n_H} w_{kj}\ f\left(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\right) + w_{k0}\right) \tag{2.4}$$

$$y_j = f\left(net_j\right) \tag{2.5}$$

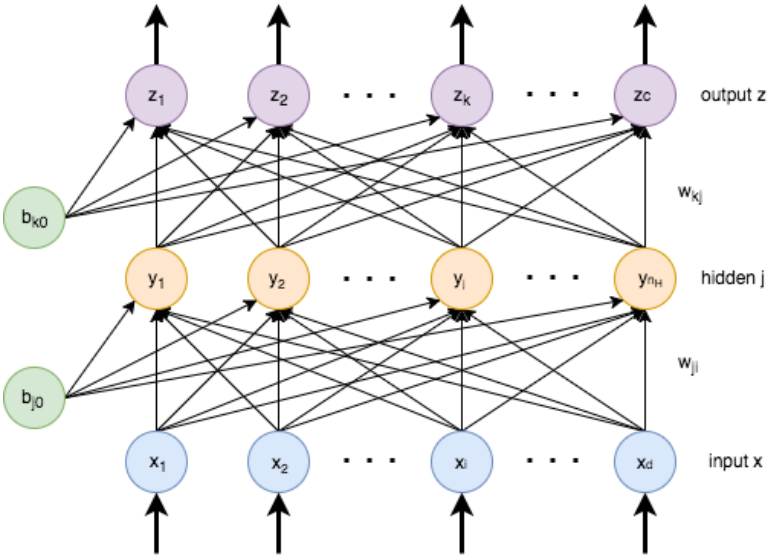

Figure 2.6: A detailed example of a FNN structure

The output of an output unit is the probability of the specific predicted class, the activation function used at the output is the softmax function and is defined as:

$$y_k = f(net_k) = \frac{e^{z_k}}{\sum_{m=1}^{c} e^{z_m}} \quad \text{for k} = 1, ..., c \tag{2.6}$$

The computation of the input vector through the hidden layer(s) to the output is called *forward propagation*. Generally, many of today's neural network consist of multiple layers which have led to the term *deep neural networks*, which opens up for highly complex nonlinear functions

as input. This leads to many of today's neural networks solving complex tasks such as speech recognition systems and self-driving.

2.2.1.2 Convolutional neural network

Convolutional neural networks (CNN) are very similar to ordinary neural networks as shown in 2.2.1.1. CNN consist of neurons that have trainable weights and biases, where each neuron receives an input and performs a dot product and optionally follows it with a non-linearity, e.g. an activation function. Instead of looking at the raw data as in FNN, CNN tries to find features in the data, e.g. if your input is an image of a car, the CNN would try to find out if it is a car and what type it is. "Convolutional networks combine three architectural ideas to ensure some degree of shift, scale, and distortion invariance: 1) local receptive fields; 2) shared weights; 3) spatial or temporal subsampling."[47] In figure 2.7 there is an example of a convolutional network with 8 filters[1]. "Each unit in a layer receives input from a set of units located in a small neighborhood in the previous layer"[47]. The small neighborhood in the previous layer is referred as the *receptive field* of the neuron, this size is specified by the kernel size. The receptive field neurons are able to extract features in signals or other visual features like edges in an image. Generally the input to a CNN can be defined as a volume with dimensions $N_{i_r} \times N_{i_c} \times$ D, where r and c are rows and columns respectively.

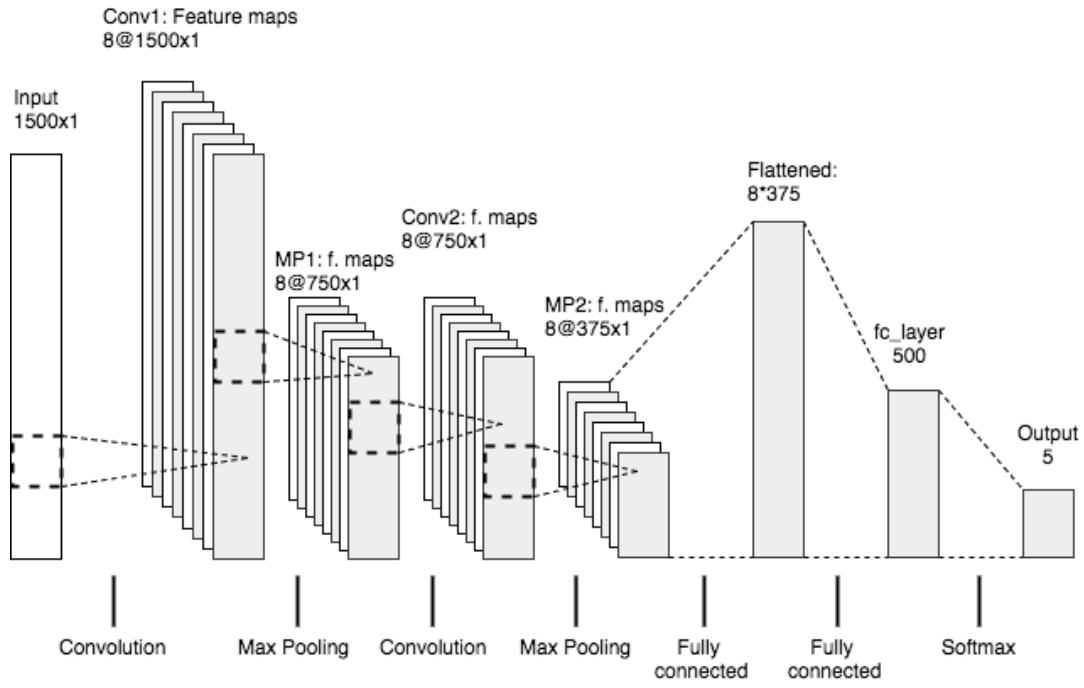

Figure 2.7: An example of a convolutional network consisting of; convolutional layer, pooling layer and fully connected layer

[1]The dimensionality of the output space, e.g. how many feature maps generated

Convolutional layers

A convolutional layer consists of a set of learnable filters. Every filter is small spatially (width and height), but extends to the full depth of the input volume. A typical filter for one-dimensional data could have size 1×2, or for two-dimensional data, the filter could be a 3×3×3 filter volume (width × height × channels/depth) for an RGB image. During forwarding computation, each filter is convolved across the width and height of the input volume and compute the dot product between the entries of the filter and the input at every position in the input. This generates an output called *feature map/activation map* that provides the responses of that specific filter at every spatial position. The units in a feature map are all forced to perform the same operation on different parts of the input. Having multiple feature maps ensures more features to be extracted at each location. This is done by having the same weight plus the bias on each specific feature map such that they detect the same feature at all possible locations in the input. These features can then be combined in deeper subsequent layers in order to detect higher order features.

As described in section 2.2.1.2 CNN consist of a local *receptive field*, in addition to *shared weights* and *spatial subsampling*. In 2.2.1.1 FNN have one neuron per input sample, for an input of 1500×1 or 128×128, then the network would have had 1500 or 16384 weights plus 1 bias for the input layer respectively. With multiple layers there are many neurons in the network structure, this does not scale well to large inputs and deeper nets[1]. CNN reduce the number of parameters needed by having neurons in a layer share weights[47].

The output of the convolution can be controlled with the hyperparameters *number of filters* (N_f), *strides*, *kernel size*, and *padding*.

We can define the spatial size (SS) of the output volume from a convolutional layer to be[43]:

$$SS = \left[\frac{\text{Input} - \text{Kernel size} + 2 \times \text{Padding}}{\text{Strides}} + 1 \times N_f\right] = \left[\frac{I - K + 2P}{S} + 1 \times N_f\right] \tag{2.7}$$

Where P are defined as[2]:

$$P = \frac{K - 1}{2} \tag{2.8}$$

The strides specify the length of the convolution, i.e. how much the filter is moved for each computation. Padding is either how many zeros are added around the border in an image or at the edge of a vector, or how many values are removed at the end of the input vector referred to as *same*- and *valid* padding respectively.

In figure 2.7 there are two convolutional layers, *Conv1* and *Conv2*. The convolutional layers compute the output of neurons that are connected to local regions in the input. Each computing a dot product between their weights and a small region they are connected to the input volume. Resulting in a volume with [n × filters] as shown in equation (2.7), where n is the length of the

[1] The real computational advantage of CNN is when we add higher dimensionality data

[2] To ensure the input and output volume will have the same size spatially.[43]

input[1] and filters determine the dimensionality of the output space (i.e. the number of filters in the convolution).

Max Pooling layers

To be capable of constructing a more robust network, reducing the sensitivity of the output to shifts and distortions, spatial subsampling is used. A common way to implement spatial subsampling is to use *max pooling*. Max pooling is a non-linear downsampling which reduces the size of the feature maps in half. It divides the feature maps into non-overlapping regions and discards all but the highest value in each region as shown in figure 2.8. Max pooling works over the spatial dimension but keeps the depth of the volume intact. There are other pooling and subsampling techniques available, but max pooling has demonstrated to yield better results [37, 31].

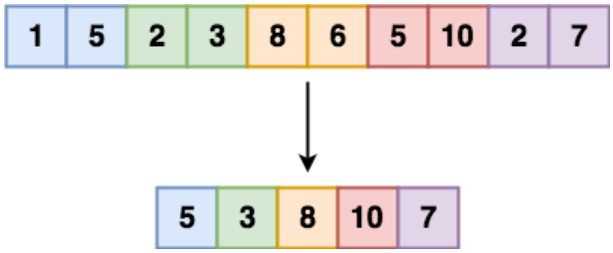

Figure 2.8: An example of 1 × 10 max pooling

Figure 2.7 also show fully connected layers, which does classification from the features extracted from convolution and max pooling.

2.2.2 Recurrent neural network

Reading a book, you can understand words based on an understanding of previous words, i.e. thoughts are persistent. Traditional FNN and CNN doesn't have this ability, if you are playing a ping pong game, knowing if the ball is coming toward or away from you would not be a trivial task to solve. RNN address this issue by recurring the data, e.g. loop the output back to the input. RNN can be seen as a sequential neural network, an example of RNN can be seen in figure 2.9.

An RNN can be seen as multiple copies of itself with the ability to passing its output to the next neuron as defined as a *state*. In figure 2.10 this is tried to be illustrated by unrolling figure 2.9.

Traditional neural networks such as FNN and CNN can be illustrated in figure 2.11 as one to one, where there is a fixed size input to a fixed-size output. RNN, however, is much more flexible, by looking at figure 2.11, RNN is able to classify problems one to many. I.e. the input could be an

[1]If the input is an image, it would result in a volume [n × m × filters], where n and m would be rows and columns respectively in the image.

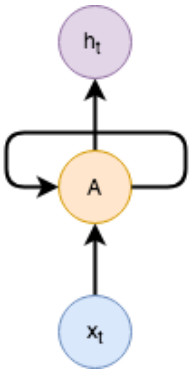

Figure 2.9: RNN with input x_t to a network A and outputs a value h_t. The loop allows information to be passed from one step of the network to the next. (Source: Christopher Olah, 2015 [33])

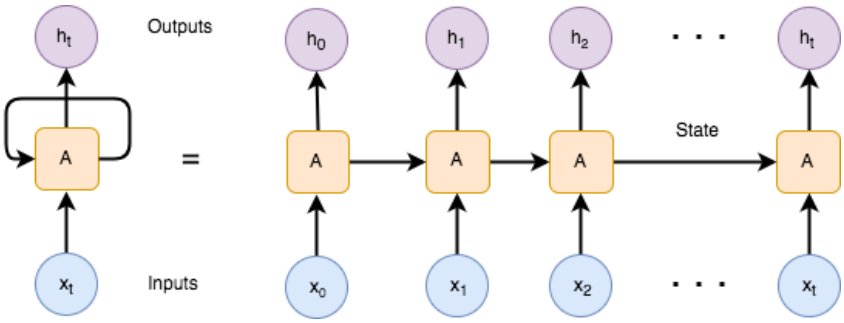

Figure 2.10: A unrolled visualization of RNN. (Source: Christopher Olah, 2015 [33])

image and its outputs a sequence of objects in the image. Many to one take a sequential input such as a speech and the output could be whether it is a male or a female. Many to many is a sequential input and output, where bi-directional many to many could be machine translation, e.g. translation of a sentence in Norwegian to English. Many to many furthest to the right could be a synced sequence of input and output, e.g. video classification where the task is to count every people in each frame.

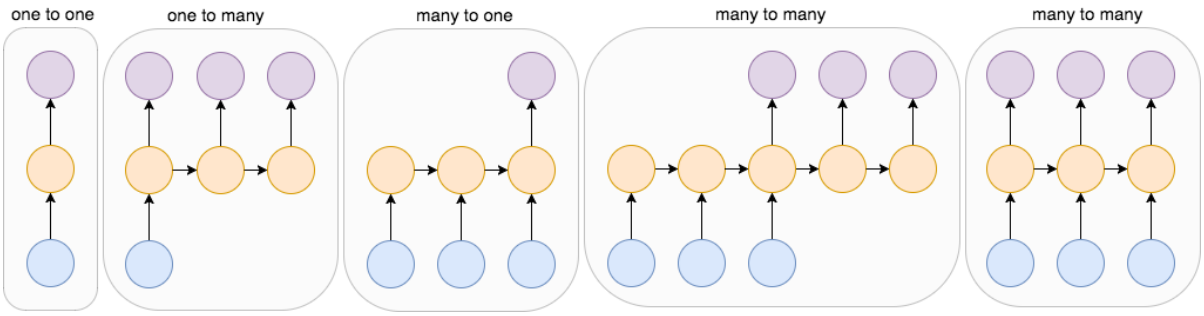

Figure 2.11: A graphical representation of different areas of use for a RNN. (Source: Andrej Karpathy, 2015 [22])

The main problem with RNN is called *long-term dependencies*, e.g. predicting the last word in a text "I grew up in Norway... I speak fluent *Norwegian*. By looking at the previous information in the text the information suggests that the next word would be the name of the language.

But to be able to narrow down which language, you would need the context of Norway from before. Since the depth of RNNs can be arbitrarily long if at any point the gradient hits a low number. E.g. close to zero, the neurons becomes saturated and draw all the earlier layers to zero explained in detail in section 2.2.3.

2.2.2.1 Long Short Term Memory - LSTM

LSTM are a special type of RNN with the capabilities of learning long-term dependencies, it was originally introduced by Hochreiter and Schmidhuber [19]. LSTM's are explicitly designed to avoid the long-term dependency problem, being able to remember information for a long period of time. The typical RNN structure can be seen in figure 2.12 with a single hyperbolic tangent (*tanh*) layer.

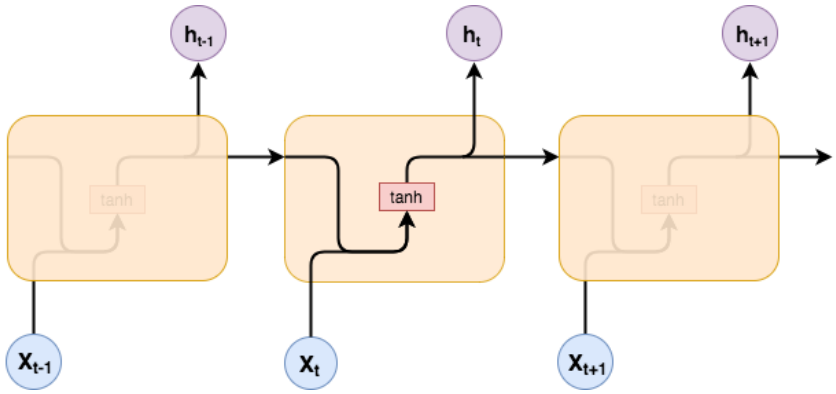

Figure 2.12: Repeating module in a standard RNN containing a single layer. (Source: Christopher Olah, 2015 [33])

An example of a repeating LSTM module can be seen in figure 2.13. In red, there are different neural network layers such as sigmoid and *tanh*. The gray circles represents pointwise operations, such as vector multiplication and additions. Vector transfer represents connections, i.e. data being passed from layer to layer. Lines that are merging denotes concatenation and copy represents the output of each cell being copied to the next cell and output.

By looking closer in the LSTM cell in figure 2.14, the cell consists of 6 equations as displayed in (2.9)[33]. The first equation f_t is called the *forget gate*, here the LSTM decides what information to throw away from the cell state. It takes the previous output h_{t-1} and the current input x_t and outputs a number between 0 and 1 for each value in the cell state C_{t-1}, where 1 represents keep and 0 forget. E.g. reading a book and there is a girl Sarah that is of no importance, she can then be forgotten.

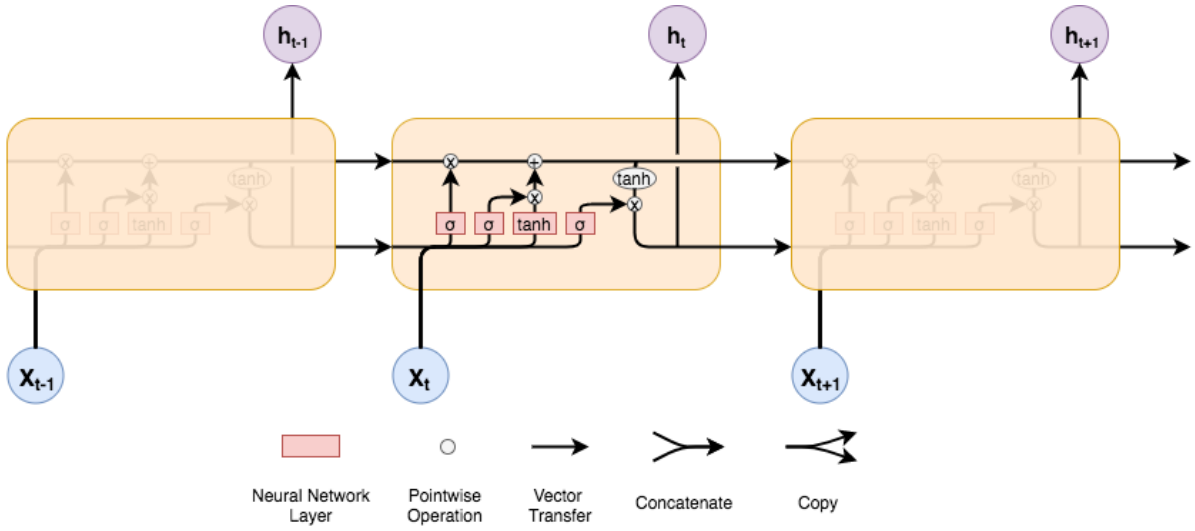

Figure 2.13: Repeating module in an LSTM containing four interacting layers. (Source: Christopher Olah, 2015 [33])

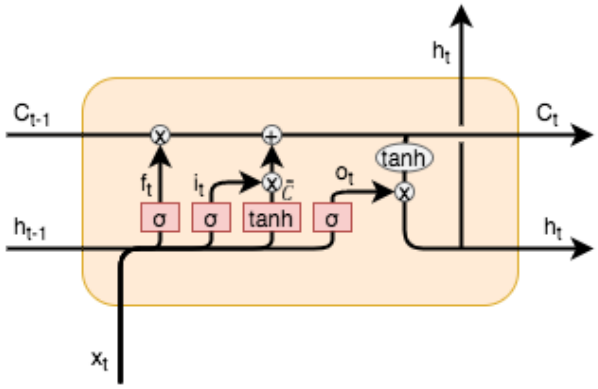

Figure 2.14: A detailed view of a LSTM cell. (Source: Christopher Olah, 2015 [33])

$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\
\tilde{C} &= tanh\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C} \\
o_t &= \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \\
h_t &= o_t \odot tanh(C_t)
\end{aligned}
\tag{2.9}
$$

Next, the LSTM decides what new information it wants to store in the cell state. This consist of two parts. One sigmoid layer called the *input gate*, i_t which decide which values to update, and one hyperbolic tangent ($tanh$) layer that creates a vector of new candidate values, $\tilde{C}$. Both these are combined via a multiplication to be added to the new cell state C_t[1]. E.g. reading

[1]The cell state C_t can be explained as a memory vector, where "at each time step the LSTM can choose to

the same book and come across a new girl Jennifer, she is then added to the new cell state to replace Sarah.

The new cell state C_t is a multiplication of the previous cell state C_{t-1} and f_t, which forgets the data decided earlier. C_t is then added the new candidate values $i_t \times \tilde{C}$, i_t is gating the input and the previous state to the current state $\tilde{C}$. Meaning that it's scaled by how much to update each new state value. This is where the information about Sarah is forgotten and Jennifer is added.

The output h_t is then defined by which parts the previous cell state to output, o_t. Further, o_t is multiplied by the current cell state C_t through a tanh to force the cell state values between -1 and 1. In the book, this corresponds to outputting what's coming next, e.g. whether the girl has blond or red hair. Note that W_f, W_i, and W_o in equation (2.9) is the trainable weights and b_f, b_i, and b_o is the biases.

In [13], Gers and Schmidhuber added *peephole connections*, which means that f_t and i_t look at the previous cell state and o_t look at the current cell state, as shown in equation (2.10). There are different variants of the LSTM architecture for RNN, and in [15] they did a large scale analysis of eight different LSTM's in speech recognition, handwriting recognition, and polyphonic music modeling, finding there was very little difference from the original LSTM [19]. They also found out that the most critical components in LSTM are the forget gate and the output activation function. However, in [21] they did another large-scale test on different RNN structures, finding some worked better than LSTM at certain tasks. They also concluded which gates are the most important. Which can be listed in following chronological order: forget gate, input gate and then the output gate.

$$\begin{aligned} f_t &= \sigma\left(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f\right) \\ i_t &= \sigma\left(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i\right) \\ o_t &= \sigma\left(W_o \cdot [C_t, h_{t-1}, x_t] + b_o\right) \end{aligned} \tag{2.10}$$

2.2.3 Back-propagation

Neural networks are always initialised with random weights on the connections. To be able to learn the network to classify correctly it needs to be trained to do so. This is done by minimising a cost function with respect to the weights in the network. By looking at the *mean squared error*:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|\mathbf{t} - \mathbf{z}\|^2 \tag{2.11}$$

where $\mathbf{t}$ is the targets and $\mathbf{z}$ is the predicted value vectors from the network with length c and $\mathbf{w}$ represents all the weights in the network. The cost function quantifies the error in the network

read from, write to, or reset the cell using explicit gating mechanisms[1]

by comparing the output to the target vector. To minimize the quantified error the weights $\mathbf{w}$ are changed in the direction that will reduce the error. This can be implemented by using the optimization algorithm called *gradient descent*. The weights are updated by taking a step of length η[1], in the direction of the steepest descent as defined by the gradient. As displayed by the update formula in (2.12) where the gradient is defined in (2.13)

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta\mathbf{w}(m) \tag{2.12}$$

$$\Delta\mathbf{w} = -\eta\frac{\partial J}{\partial \mathbf{w}} \tag{2.13}$$

Looking at the example in figure 2.6 where there is a three-layer network. By back-propagating[7] from the top layer, backward to the input. This leads to the use of the chain rule as seen in equation (2.14), first from the output to the hidden layer. From the chain rule, we get $\frac{\partial J}{\partial net_k}$ which can be rewritten as (t_k - z_k)f'(net_k), resulting in the derivative of the activation function[2].

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} \tag{2.14}$$

Neural networks can become very large, and one has to be mindful of the effect of back-propagating with the resulting derivative of each layer. By looking at the functions in figure 2.15, we can consider using the sigmoid as our activation function. If the input to the sigmoid function is between -2 and 2 the gradient of the function is already 0.1. Looking at an example; given a network with 3 layers, with the same values as described above in all layers. The further back to the input of the network the less the weights will change due to the derivative. $0.1\times0.1\times0.1 = 0.001$, resulting in no updates in the weights. This is called a *saturated* neuron, if a neuron is saturated it will block the gradient from flowing further down the network. In general, this problem is called the problem of *vanishing gradient*. A solution to vanishing gradient is to choose the rectified linear unit (ReLU) activation function displayed in figure 2.15, where the derivative is always one. If the ReLU function receives an input that is less than zero, the derivative is zero and no gradient will flow. This means that a neuron outputs a value of zero, and will never be updated. These dead ReLU's may occur if the step size is too high during training and the weights accidentally get updated in a way that the input to the neuron never will be positive again. In addition to ReLU, there is a modification called ReLU6 where every input value x is restricted from 0 to 6 and is zero otherwise, this function has been evaluated as a hyperparameter.

The most common cost function for training neural networks is the categorical cross entropy function[8]. This is also the loss function that is being used in this report.

[1] Also known as the *learning rate*

[2] For the interested reader; the full back-propagation is derived in page 290-292 in [9]

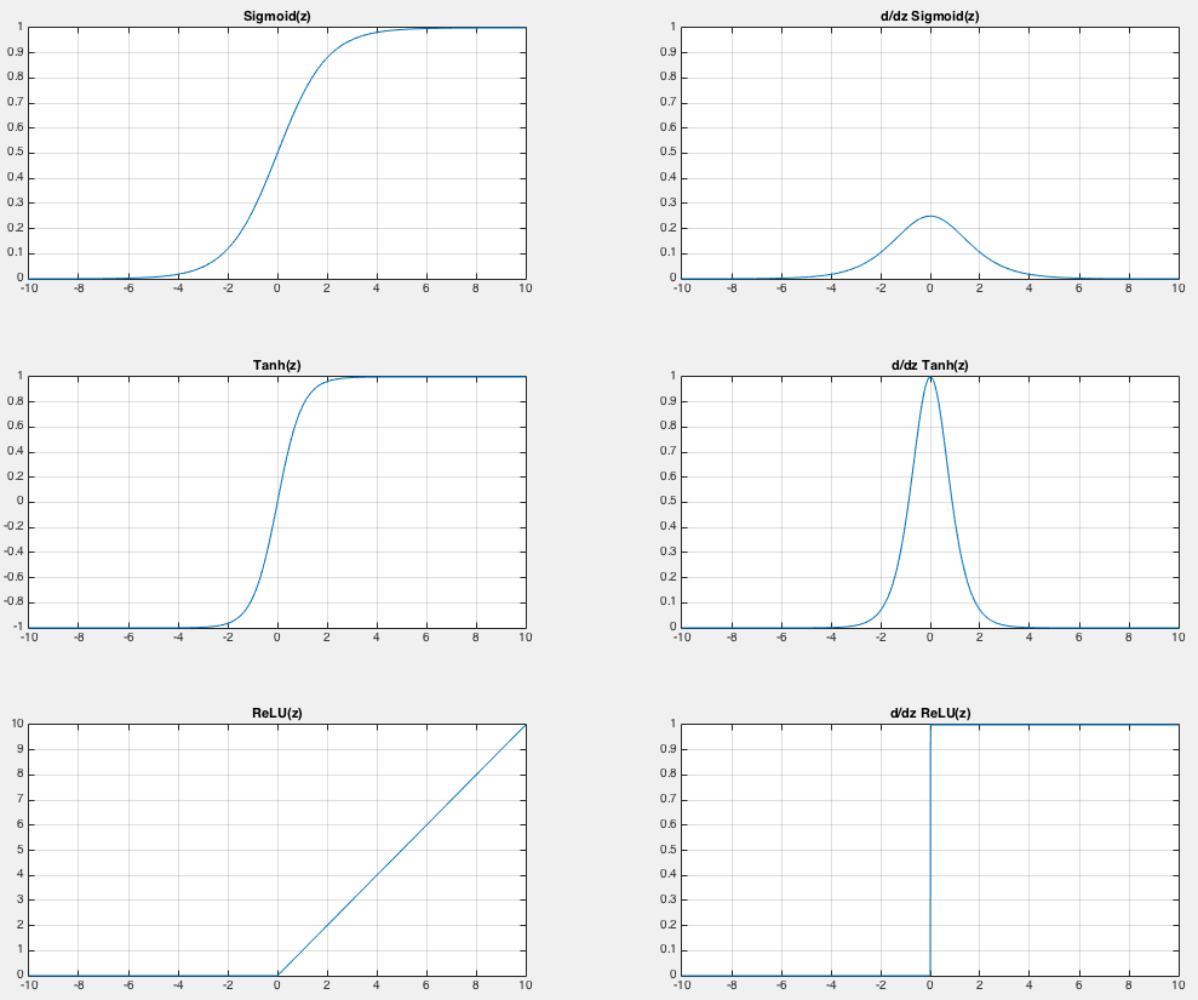

Figure 2.15: Activation functions; sigmoid, *tanh* and *ReLU* (left) and its derivative (right)

$$J_{CE} = -\frac{1}{n}\sum_{x}(yln(a) + (1-y)ln(1-a)) \tag{2.15}$$

2.2.4 Training neural networks

There are many ways to train a neural network, the most popular ones are either; split the dataset into two or three sets: a training and test set, alternatively a cross-validation (CV) set. Each set consists of n example input vector with a given length m. The sets are not necessarily the same size, a popular way of dividing data usually consist in dividing the data into 70% and 30% for training and test set respectively. The cross-validation, however, is extracted from the training set, and a common way to create a CV set is by using Pareto's principle. Pareto's principle states that for many events, roughly 80% of the outcome, comes from 20% of the causes.

The training set is used to update the weights while training the network, while CV set is used during training to see how well the network perform on data not seen during training. CV set

is forward propagated through the network and cost and accuracy calculated without updating weights. CV is a good indicator to determine if the network is overtrained[1] by evaluating if the cost in the CV set is much higher than the cost of the training set. The validation set can be used to only store improvements from validation and stop training if no improvements are made in i epochs[2]. This can be visualized in figure 2.16.

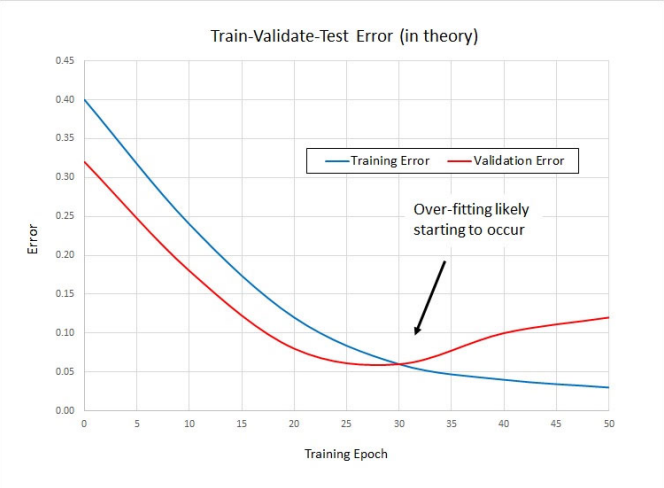

Figure 2.16: Early stopping by looking at the cost function. (Source: James McCaffrey, 2015 [29])

There are different gradient decent methods available, *batch gradient descent* interprets the gradients from the entire training data. And are summed up before one step/iteration with the given η in the direction of steepest decent. This is very unpractical and computationally inefficient with large datasets. In practice, the data is split up into *mini-batches* randomly selected from the dataset and the weights are updated with the gradients found from these few examples. This is called *mini-batch stochastic gradient descent* (SGD), typically the batch sizes vary from 32-256 depending on the capacity of the GPU/RAM of the computer. However, SGD is a slow algorithm and needs dozens of iterations to converge. An example of SGD can be seen in figure 2.17.

[1] Overtraining or *overfitting* means that the network is specialised to predict the data in the training set but predicts new data poorly.

[2] One epoch corresponds to training trough the whole training data set

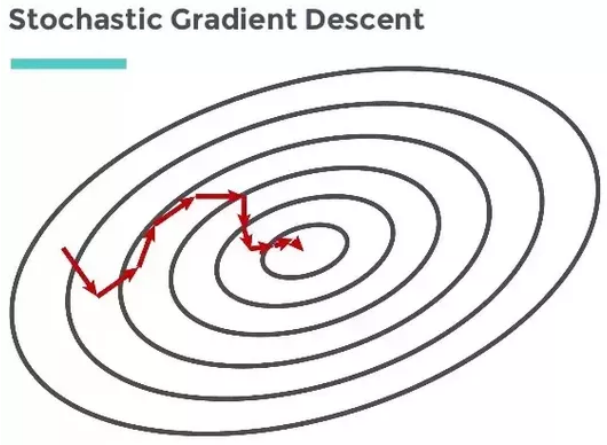

Figure 2.17: SGD, red arrow illustrates each step/iteration. (Source: Chris Fregly, 2016 [14])

Therefore, *stochastic gradient descent* (SGD) is not the optimizer that's being used, but a modification called *Adaptive Moment Estimation* (ADAM). "The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients"[25]. In figure 2.18 there is shown a comparison of ADAM and SGD optimizer.

Figure 2.18: Training network with SGD (in blue) vs ADAM (in orange). This is a rather extreme scenario, the SGD struggles to find improvement, given enough time SGD should reach ADAM's prediction.

As mentioned before, overtraining the training data are an important part of training neural networks. This is characterized by a low error in the training data but a high error in the test data. A way of handling overtraining is by implementing dropout[42]. Dropout is only keeping a neuron active with a probability p (a hyperparameter) and its connection to units in the next layer with weight $\mathbf{w}$. During testing of the network, all neurons are active, but each weight $\mathbf{w}$ are multiplied with p[42]. Intuitively, this forces the network to be accurate even in the absence of certain information. It prevents the network from becoming too dependent on any of the

neurons, thus learning more useful features on its own. This can be viewed in figure 2.19

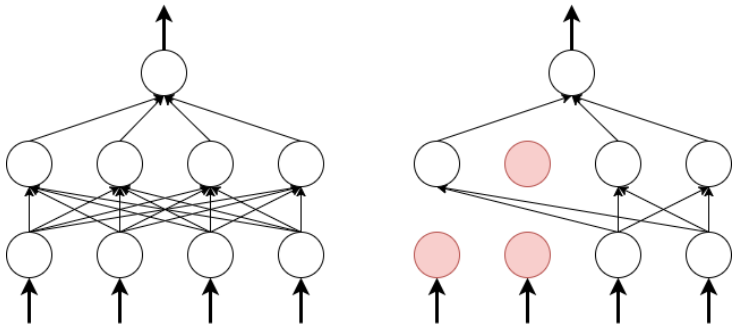

Figure 2.19: An ordinary neural network (left) and a network with randomly selected neurons dropped during training (right)

2.2.5 Synthetic Minority Over-sampling Technique (SMOTE)

To meet the data needs of deep learning it's common to create more training examples by modifying the examples in the training set. For images, this could be to rotate, flip or crop, for signals however this is not the correct way to proceed. "The performance of machine learning algorithms is typically evaluated using predictive accuracy. However, this is not appropriate when the data is imbalanced and/or the costs of different errors vary markedly"[5]. A way of dealing with an imbalanced dataset or generating more data is to synthesize new data to the minority class or to the dataset respectively. Using Synthetic Minority Over-sampling Technique (SMOTE)[5] is a popular way to synthesize new data. The whole idea is to create data by interpolating between existing data, an example can be seen in figure 2.20

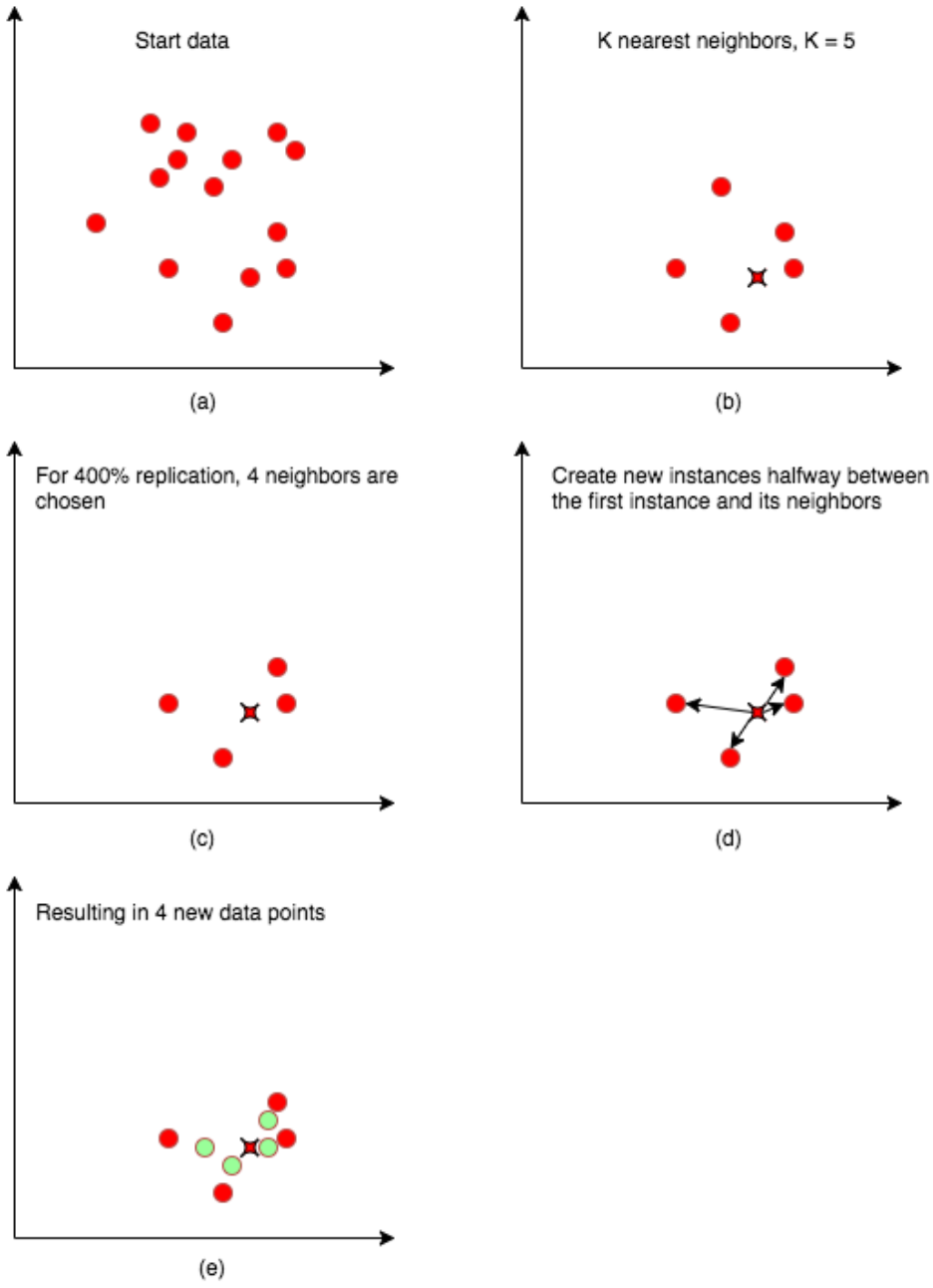

Figure 2.20: Example of an implementation of SMOTE, (a) is the minority class, (b) use KNN on every minority instance, (c) For every minority instance, choose its k nearest neighbors, (d) Creating new instances between the first instance and its neighbors, and (e) resulting in new synthesised data

Chapter 3

Materials and methods

This chapter explains the data and how it is extracted, in addition to how the datasets are created and preprocessed. It will as well explain how the neural network models are implemented and how the preliminary testing are conducted to find suitable models for further testing are done. As well how the training and evaluation are carried out.

3.1 Data Collection

The out-of-hospital cardiac arrest (OHCA) database consist of 394 patient records. It was obtained from a multicenter cardiac arrest study conducted to evaluate cardiopulmonary resuscitation quality[46]. The study was conducted in three geographic locations Akershus (Norway), Stockholm (Sweden), and London (UK) between March 2002 and September 2004. The surface ECG was acquired using a modified Laerdal Heartstart 4000 defibrillator, with a sampling rate of 500 Hz and 16 bits for a resolution of 1.031 µV per least significant bit. Rhythm annotations on the data were done by clinical experts using five classes: ventricular fibrillation (VF), ventricular tachycardia (VT), pulseless electrical activity (PEA/PE), pulse generating rhythms (PR) and asystole (AS). Chest compressions intervals were annotated using the compression depth available from a CPR assist-pad.

AS was defined as peak-to-peak amplitude below 100 µV, and/or rates under 12 bpm. Rhythms with supraventricular activity (QRS complexes) and rates above 12 bpm were labeled as either PR or PE. PR were based on clinical annotations of return of spontaneous circulation (ROSC) made in patient charts during CPR and on the observation of fluctuations in the transthoracic impedance (TTI) signal aligned with QRS complexes. Irregular ventricular rhythms were annotated as VF (coarse VF was defined for peak-to-peak amplitudes above 200 µV). Fast and regular ventricular rhythms without a pulse and rates above 120 bpm were annotated as VT[3].

ECG segments were automatically extracted based on these annotations with the following criteria: 3-second chest compression artifacts, a single rhythm. And the same signal annotation

before and after chest compression artifacts for the extracted signal with a minimum 1000 samples of the same annotation after chest compression artifacts.

In the OHCA dataset, a formal description of the context in the episodes is used to be able to design algorithms that can assess the information needed in terms of the course of time and type of event. For each event type e_i, there is a time point t_i describing the start time. Events mark a change of state, the state E_i is determined by the type of event at time t_i. The state is unchanged until the next event e_{t+1} which marks the transition into the next state E_{t+1}. For each state, there is defined a corresponding time interval, $T_i = [t_i, t_{i+1}]$. Hence, the course of events during a resuscitation episode are defined as a continuous sequence of states S = $\{(T_1,E_1), (T_2,E_2), ..., (T_N,E_N)\}$ where the time intervals are ordered according to the start time of the episode, t_s. The therapeutic states are either the rhythmic states S_r = {VT, VF, AS, PR, PE} which represents ongoing rhythms defined by the start and end by the corresponding transition events. Compression sequences are marked with a C, resulting in rhythmic states with compression S_{r_c} = {CVT, CVF, CAS, CPR, CPE} and the defibrillations as *dfb*. This can be combined into a sequence S_c = {([50.2, 113.8], AS), ([113.8, 180.2], CAS), ([180.2, 300], VF)}[10]. All classes used in this thesis are defined by S_{r_c} = {CVT, CVF, CAS, CPR, CPE}.

Further, the dataset was divided into two different sets.

3.1.1 Small dataset

The first dataset is based on the same extraction as in the data set of [3], where there would be no overlap in the segments. Due to the low appearance in the classes CVT and CPE, the signals were extracted with non-overlapping successive windows to get more data. This was also done in [3] for the class VT. Afterward, the signals were analyzed using a GUI in MATLAB[1] with TTI and compression depth data to determine whether to keep or remove the segment. The following exclusion criteria's were used; compression depth or TTI were insufficient, severe signal noise or complexity generated by the heart, e.g. in asystole sudden pulse rhythms could occur. Before analyzing, the dataset contained a total of 7171 segments but due to this task is very time consuming, and the classes not evenly distributed. The final dataset contained a total of 2446 segments. For the class CVT, SMOTE was used to increase the data by a factor of 4. An example of a synthetic generated data from SMOTE can be seen in figure 3.1. 1674 segments were used for training with the following class setup; 280 CVT (n^2 = 7) , 384 CVF (n = 61), 377 CAS (n = 51), 384 CPR (n = 19) and 375 CPE (n = 55). The remaining 772 were used as test data with the following class setup; 116 CVT (n = 2), 169 CVF (n = 16), 168 CAS (n = 18), 194 CPR (n = 11) and 183 CPE (n = 15). The patients in training and test set were kept separated to avoid data leakage. Data leakage is among the top 10 machine learning and data mining mistakes[32]. Leading to an overestimation of performance in machine learning classifiers[23].

[1]See Appendix B.1 for details.
[2]Number of patients segments are extracted from.

3.1.2 Large dataset

To be able to get as much data as possible, some of the previous criteria were ignored. For this dataset, the ECG segments were extracted by the same criteria as referenced in section 3.1, but with a moving window of 30 samples per cut. From each start annotation, the first 200 samples were ignored due to transient in the signals. This increased the dataset to 1451111 segments before processing of the dataset. For this dataset the classes were as follows; 587330 CAS (n = 185), 500781 CPE (n = 173), 56972 CPR (n = 45), 300918 CVF (n = 130) and 5109 CVT (n = 9). To avoid an imbalanced dataset, CAS, CPE, and CVF classes were reduced to 100.000 cuts. For CPR and CVT, SMOTE were added to increase the data to 100000 and 20436 respectively. Further the dataset were divided into 70% for training and 30% for test resulting in; Training = 296105 with the classes representing; 14392 CVT (n = 7), 69963 CVF (n = 22), 70376 CAS (n = 26), 71524 CPR (n = 12) and 69850 CPE (n = 25). Test = 126310 with the classes representing; 6044 CVT (n = 3), 30074 CVF (n = 22), 30018 CAS (n = 12), 29832 CPR (n = 7) and 30342 CPE (n = 8). The patients in the training- and test dataset were kept separated to avoid data leakage.

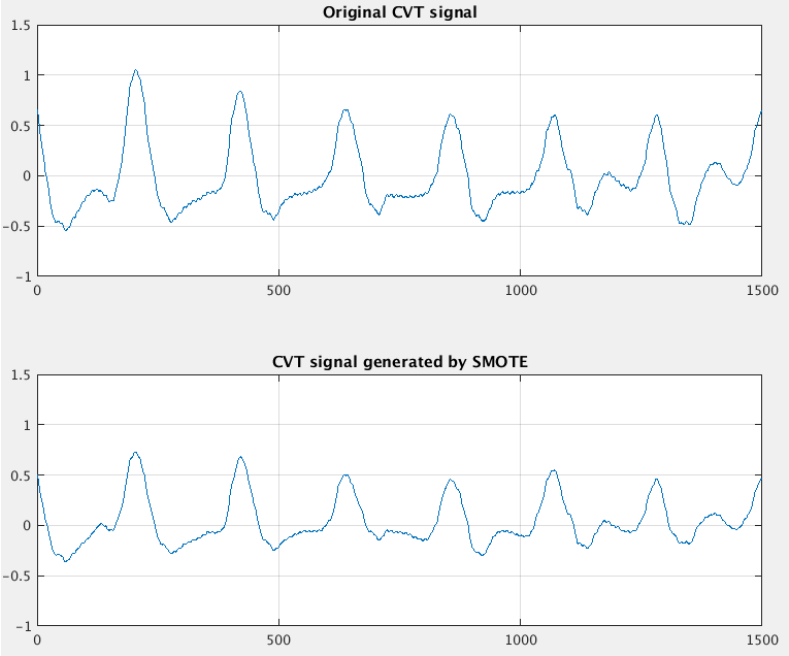

Figure 3.1: Example of VT with compressions (CVT) generated by SMOTE

3.1.3 Pre-processing of the dataset

The ECG was band limited to 0.5 - 30 Hz (10th order IIR Butterworth filter), a typical ECG monitor bandwidth used in AEDs [26], which removes baseline wander and high-frequency noise. This was implemented by using anti-causal, zero-phase filtering, also known as forward-backward filtering on each segment. E.g. first apply one causal filter to the signal in forwarding direction and a second anti-causal filter backward direction on the filtered signal[34, 16][1]. The filter function used reduces filter startup transients, but to be sure no transients to occur in the filtered signal. There was added 1-second transient interval at the start and end of the segment before filtering and then removing it afterward. The result can be shown in figure 3.2 where there is a clear transient in the start and end of the signal.

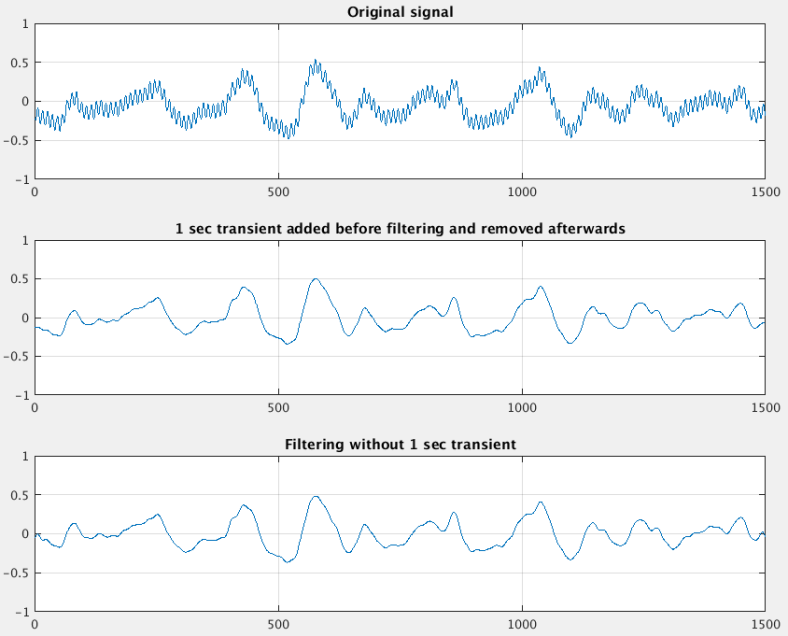

Figure 3.2: Example of VF with compression artefacts (CVF) band limited to 0.5 - 30 Hz

[1]Matlab's function for this is called *filtfilt* which can be found in Matlab's Signal Processing Toolbox [45]

3.1.4 Algorithm for extracting data

The algorithm for extracting data from the OHCA dataset is written in MATLAB, and a flowchart of the code is displayed in figure 3.3. *Signal annotations* and *Sample annotations* consist of the complete list of annotated episodes and the corresponding sample values for each of the 394 episodes in the OCHA dataset. There is also some parameters which define the minimum sample length for the annotations after compressions, and the maximum compression length. Both these parameters were tested with different values to extract as much data as possible for the class CVT.

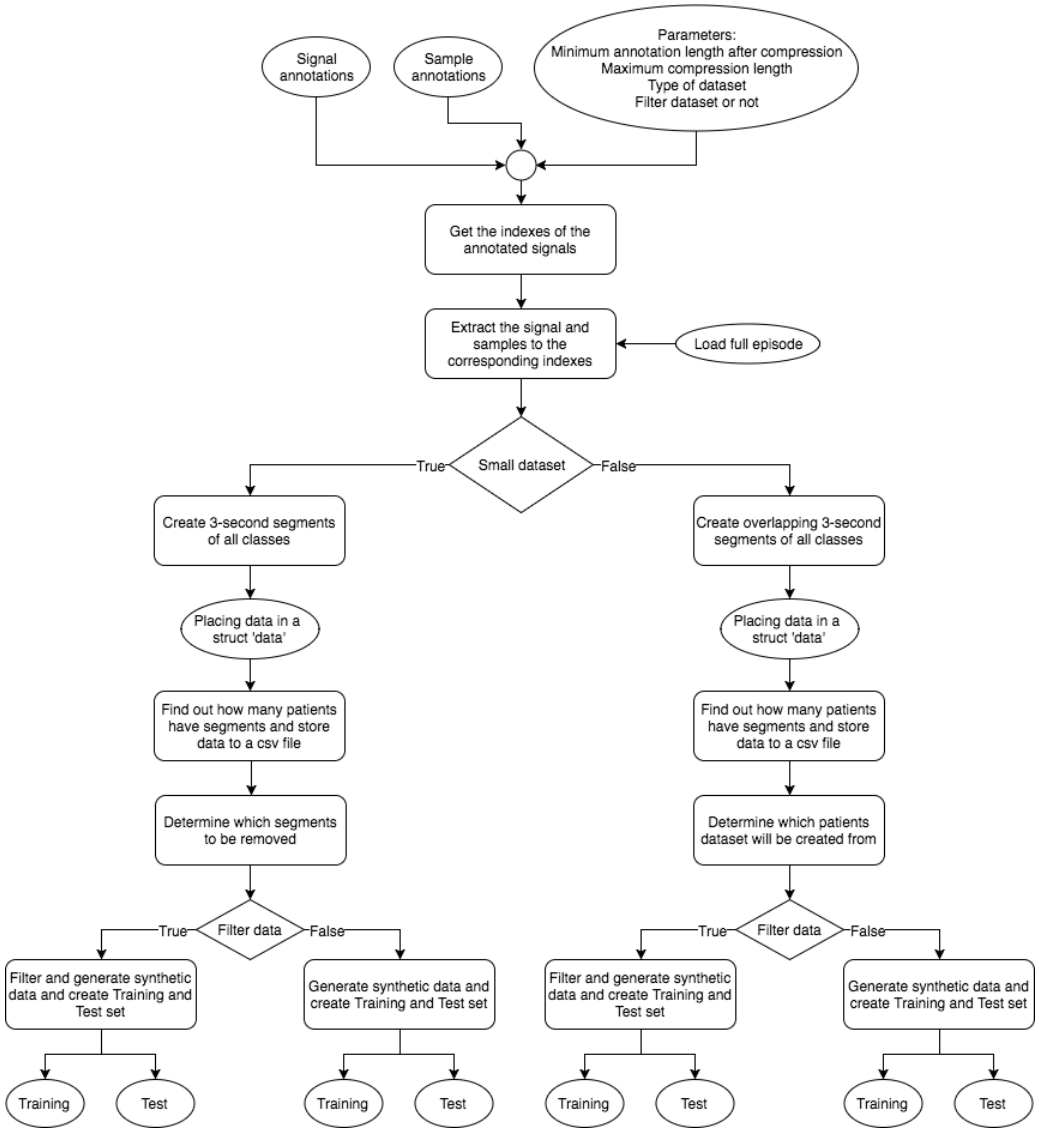

Figure 3.3: Algorithm for data extraction, squares equals functions, ellipse equals variables/data storage and tilted squares are if statements

Get the indexes of the annotated signals receives the signal and sample annotations, and the parameters as input, and outputs signal indexes for all five classes in separate variables. The function examines for the same annotations prior to and after for compression artifacts. E.g. for CVT: the function searches for VT as the annotation before and after CVT, this can be seen as a sequence vector[1] with the following annotations [VT, CVT, VT][2]. The function then uses the minimum annotation length after compression to ensure that VT is with a requirement length after CVT. This is because of rhythm changes during compression was not annotated in the dataset, and could lead to the annotation after compression to deviate from the time the compression stopped, resulting in false labeling. Maximum compression length ensures that the compression is not too long to avoid potential rhythm changes during compression.

Extract the signal and samples to the corresponding indexes receives the output from the previous function and the full episode as input and returns the corresponding full signal and the samples from the indexes for each class.

Create 3-second segments of all classes and *Create overlapping 3-second segments of all classes* are two functions depending on the size of the dataset as described in section 3.1.1 and 3.1.2. Both functions receive the full signal and sample for each class respectively as input and return the signal segments for all five classes and the corresponding start and end sample values[3]. In the small dataset 3.1.1 the three classes CVF, CAS, and CPE are all extracted from the center position of the signal. Meanwhile, due to low occurrences in CVT and CPR all signals have non-overlapping segments of the signals. For the large dataset 3.1.2 all signals have overlap with 30 sample displacement. At the start of every segment, the first 200 samples were ignored due to transitions.

All the data is then placed in a struct which consists of; 3-second segments, sample frequency, rhythm type, patient identity, the original registry for the specific geographic locations, start and end sample value for the segments and the name of the specific segment. The structure is based on how [3] used the MATLAB GUI in Appendix B.1, where it is possible to load each episode and watch every signal cut.

Find out how many patients have segments and store data to a CSV file receives the data struct as an input parameter, and uses its content to determine how many patients have segments, and create a CSV file with the data.

Filter/Generate synthetic data and create Training and Test sets receive the data struct, which patients desired in 3.1.2. Or all the segments that should be removed for all five classes in 3.1.1, and whether the dataset is to be band limited or not as the input parameter. The function returns Training and Test set with the corresponding labels for all classes[4].

[1] As described in section 3.1

[2] An annotation with a sequence vector [VT, CVT, AS] or [AS, CVT, VT] is not included further for data extraction.

[3] Start and end sample values are needed to inspect the segments in the MATLAB GUI.

[4] How many percent for training and test is defined prior to this function

3.2 Neural network

Training the neural network was conducted in Python using a library package called TensorFlow[1] developed by Google. A flowchart of the implementation can be seen in 3.4. The input to the different networks are the same 3-second segments extracted directly from section 3.1.4, which also can be visualized in figure 2.4.

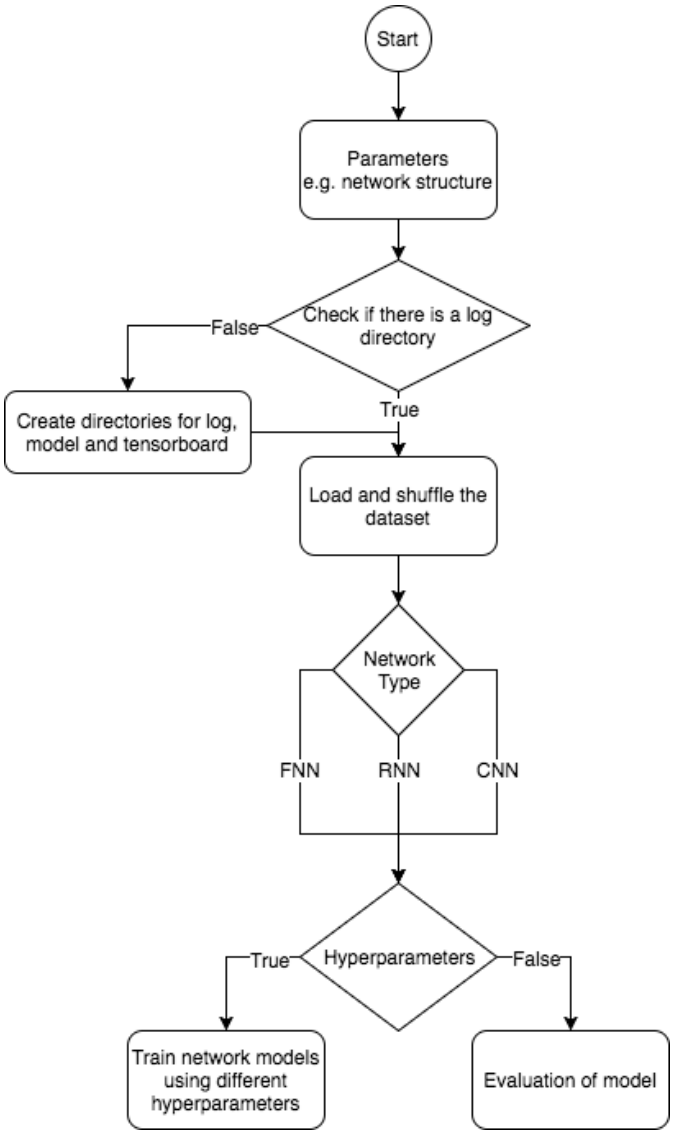

Figure 3.4: Flowchart of Neural Network training

In *Parameters*, every parameter[2] and hyperparameter[3] are defined. Since there are an arbitrary number of different network models to be evaluated, each model is stored in specified directories. Such it's easy to reload or evaluate the specific model. There is also created a logfile which stores all the necessary information about the specific runs.

[1] See Appendix B.2.2 for details
[2] Type of dataset, how many epochs, requirements etc..
[3] Different NN structures such as layers, dropout, activation function, and so on.

There are two options in the flowchart, *Train network models using different hyperparameters* and *Evaluation of model*. Each of these two are explained in detail in section 3.2.1 and 3.2.3 respectively.

3.2.1 Training the neural network

In section 3.2, figure 3.4 visualized an overall flowchart over the neural network code. Further, figure 3.5 highlights *Train network models using different hyperparameters*, this box can be seen in detail in figure 3.6.

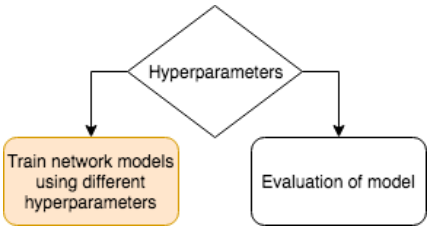

Figure 3.5: Highlighted box for further explanation

From the previous parameters defined in figure 3.4, the specified network structure for either FNN, CNN, or RNN is created. All weights and biases are initialized with a truncated normal distribution. *Create and shuffle training and validation batches* are defined such the dataset is divided into n smaller batch sizes. For each iteration i, the network uses the next batch for training. When the network has completed training on the last batch the dataset, it is then reshuffled and new batches are used for training. This is to ensure different training data at each time-step. To be able to reload the best weight and bias values, a new model is stored for each new improvement in validation accuracy and cost. If the program has reached max iterations or if no improvements are found in i iterations, the program stops and restores the best weights and displays results.

3.2.2 Preliminary testing

Since there are an arbitrary different number of network structures, hyperparameter training where conducted to test a vast amount of models. For the FNN structures, the following hyperparameters where cross evaluated with the corresponding values:

- Learning rate = [0.01, 0.001, 0.0001]
- Batch size = [32, 64,128]
- Number of hidden nodes in every layer = 100 to 1000 with a step of 100
- Number of layers = [2, 3, 4, 5, 6]
- Activation function = [ReLU, ReLU6, *tanh*][1].

[1]There was a pre evaluation of activation functions conducted to evaluate all activation functions on TensorFlow such as: exponential linear (elu), softplus, softsign, and sigmoid. But these gave poor results and were excluded.

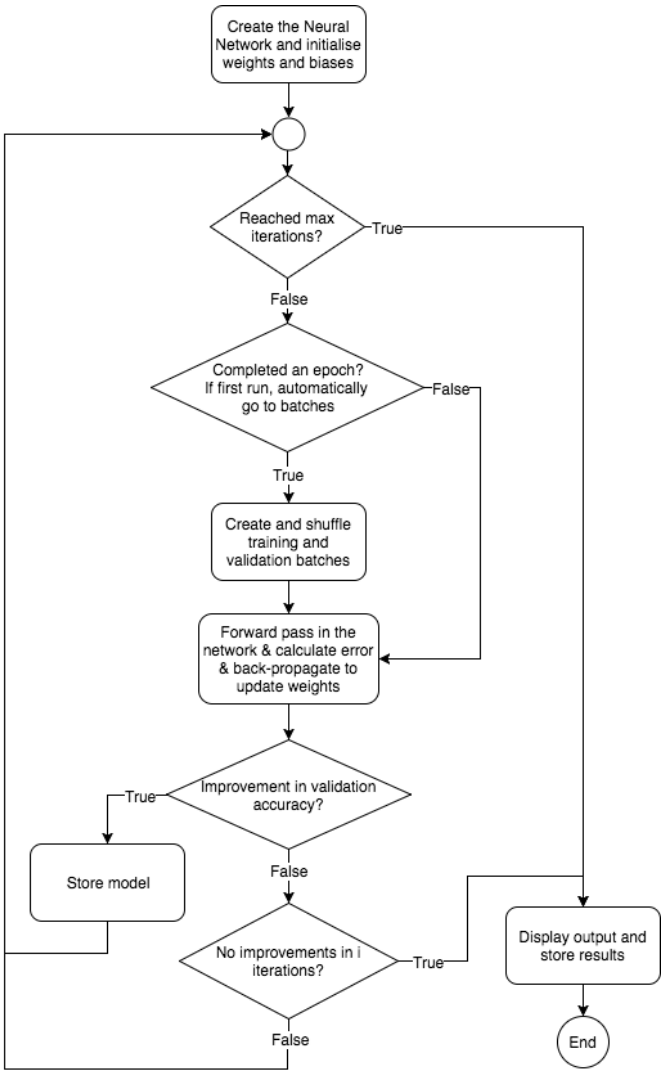

Figure 3.6: Detailed visualization of the training of network models

- Dropout = [0.1, 0.2]

Resulting in a total of approximately 1000 different model structures[1] for FNN networks.

In the CNN structure, the following parameters were cross evaluated with the corresponding values:

- Learning rate = [0.01, 0.001, 0.0001]
- Batch size = [32, 64,128]
- Filters = [8, 16, 32, 64]
- Number of convolutional layers = [2, 3, 4]
- Number of fully connected layers[2] = [1, 2, 3]

[1]Learning rate where reduced to 0.0001 as a parameter and the activation function where reduced to ReLU and ReLU6 due to poor results from *tanh*, resulting in fewer models tested.

[2]After of the convolutional layers

- Number of hidden nodes in every fully connected layer = 100 to 1000 with a step of 100
- Padding = [Same, Valid]
- Activation function = [ReLU, ReLU6, *tanh*]
- Dropout = [0.1, 0.2]

Resulting in a total of approximately 2500 different model structures[1] for CNN networks.

In the RNN structure, the following parameters were cross evaluated with the corresponding values:

- Learning rate = [0.01, 0.001, 0.0001]
- Batch size = [32, 64,128]
- Number of layers = [2, 3, 4, 5, 6]
- Forget bias = [0.1, 0.2, 0.3, 0.4]
- Dropout = [0.1, 0.2]

Resulting in a total of 120 different model structures for RNN networks.

All tests where done in following order:

- All hyperparameter runs were tested on the small dataset as described in section 3.1.1, for both filtered dataset and unfiltered for FNN, CNN, and RNN. This was due to the task being too time-consuming using the large dataset as described in section 3.1.2.
- Only models with the best validation accuracy[2] were included for further testing.
- Models might enter a local optimum, resulting in a good result for that specific run. Therefore the 10 best hyperparameters run for FNN, CNN, RNN were retested 3 times to ensure models from entering a local optimum.
- Further, the 5 best network structures from the small dataset 3.1.1, were tested three times on the large dataset 3.1.2 on both filtered and unfiltered data.

The final result included the 3 best result from FNN, CNN, and RNN to be trained 10 times to ensure that the performance is not biased on the selected data. And calculate the average and standard deviation of the sensitivity, positive predictive value, total accuracy and the unweighted mean of sensitivity for all five classes from a confusion matrix.

3.2.3 Evaluation of the network

In section 3.2, figure 3.4 visualized an overall flowchart over the neural network code. Further figure 3.7 highlights *Evaluation of model*, this box can be seen in detail in figure 3.8

[1] In total if all these parameters where evaluated this number would reach 38880 models. So actions were taken such as all structures were trained using 500 neurons in the first tests. Then, chose the best hyperparameters for further testing. The learning rate and activation functions where reduced to one and two hyperparameters respectively, as in FNN.

[2] Typically over 60%

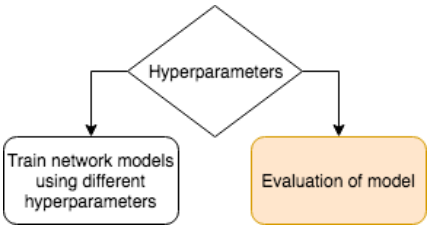

Figure 3.7: Highlighted box for further explanation

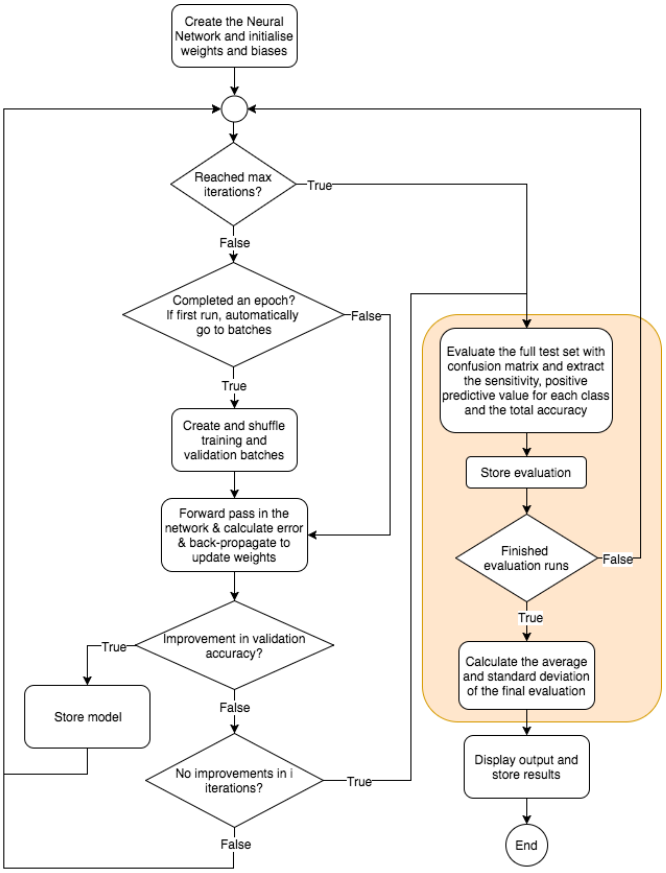

Figure 3.8: Detailed visualization of the evaluation of the trained network models

The evaluation is done by using confusion matrix which provides a complete description of any classification result. For performance metrics, the *positive predictive value* (PPV), *sensitivity* (Sen), *multiway accuracy* (MulAcc), and the *unweighted mean of sensitivities* (UMS) was computed (see Appendix A.1 for a detailed technical description). For each class i, Sen and PPV can be expressed as:

$$Sen_i = \frac{N_{i,i}}{T_i} \quad \text{and} \quad PPV_i = \frac{N_{i,i}}{C_i} \tag{3.1}$$

Since *MulAcc* measures the total accuracy of the network, it's sensitive to class imbalance. But the datasets in section 3.1.1 and 3.1.2 is taken into account for unbalanced data, resulting

in *MulAcc* treated as the main performance criterion. *UMS* weights the classes equally, thus resulting to be unaffected by class imbalance. *Finished evaluation runs* in figure 3.8 ensures that there can be computed an average and standard deviations of the performance metrics to be used as a final selection of the best performing model. *MulAcc* and *UMS* can be expressed as:

$$MulAcc = \frac{1}{N_{tot}} \sum_{i=1}^{5} N_{i,i} \quad \text{and} \quad UMS = \frac{1}{5} \sum_{i=1}^{5} Sen_i \tag{3.2}$$

Chapter 4

Results

The experimental setup described in section 3.2 concluded that the best hyperparameters for each network structure, FNN, CNN, and RNN to be as follows:

- FNN: ReLU6 as the activation function with a high number of hidden neurons (typically 700-900 in each layer), and a dropout of 20 percent. The depth of the network varied with values from 3 to 6 layers leading to a small decline in performance to deep networks.
- CNN: ReLU6 was the activation function which gave the best results, and the number of hidden neurons varied between 300-600. The dropout did not have any determined value, resulting in a variation from 10 to 20 percent.

  The depth of the network had the best result in a deep four layer convolution with two max pooling layers, followed by three layers consisting of fully connected layers[2].

  There was a high consistency in using three fully connected layers compared to one and two layers. The number of filters which gave the best performance was typically low number of 8 with some exceptions using 16 filters for some models. There was no large differentiation by using same or valid padding.
- RNN: The best results for RNN came from either 3 or 4 layers with a dropout varied from 10 to 20 percent, and a high forget bias of 40 percent

This chapter presents results achieved by the 3 different network structures described in section 3.2. All evaluation and results are referred to the test set on the large dataset. Based on preliminary testing described in section 3.2.2, the performance of the classifiers has been ranked in terms of *MulAcc* and *UMS*. A detailed display of the best results for each method using the mean of *PPV* and *Sen* for each class can be seen in section 4.4. The hyperparameters *learning rate* and *batch size* gave the best results from the preliminary testing at 0.0001 and 32 respectively, and are therefore not displayed in the tables below.

[2]In Appendix B.2.2, figure B.3 visualize the network structure.

4.1 Results from FNN

The best result from FNN models consists of the three best mean test result for the best hyperparameters with and without band limitation. The best network obtained from the best models was further tested 10 times with the mean value of *MulAcc* and *UMS* for the global measures of performance. Results for FNN models can be seen in table 4.1.

Table 4.1:
The 3 best results for FNN with different hyperparameters using the large dataset, the numbers are an average of 10 runs with their standard deviations in parentheses.

FD	FL	HU	DO	AF	*MulAcc*	*UMS*
y	3	900	0.2	ReLU6	**52.30(0.27)**	**44.08(0.23)**
y	5	700	0.2	ReLU6	51.69(0.81)	43.57(0.58)
y	6	800	0.2	ReLU6	50.58(0.08)	42.60(0.06)

Abbreviations:
FD = Filter data (y = yes / n = no), FL = Fully connected layers, HU = Hidden units, DO = dropout, AF = Activation function.

4.2 Results from CNN

The best result from CNN models consist of the three best mean test result for the best hyperparameters with and without band limitation. The best network obtained from the best models where evaluated with the mean value of *MulAcc* and *UMS* for the global measures of performance as in section 4.1 for FNN. Results for CNN models can be seen in table 4.2.

Table 4.2:
The 3 best results for CNN with different hyperparameters using the large dataset, the numbers are an average of 10 runs with their standard deviations in parentheses.

FD	CL	FL	NF	HU	DO	AF	Pad	*MulAcc*	*UMS*
y	2	3	8	600	0.1	ReLU6	Same	51.25(1.6)	49.84(9.13)
y	4	3	8	300	0.2	ReLU6	Valid	**52.39(6.3)**	**46.47(2.19)**
y	2	3	16	300	0.2	ReLU	Same	48.97(21.85)	48.52(1.0)

Abbreviations:
FD = Filter data (y = yes / n = no), CL = Conv layers, FL = Fully connected layers, NF = Number of filters, HU = Hidden units, DO = dropout, AF = Activation function, Pad = Padding.

4.3 Results from RNN

The best result from RNN models consist of the three best mean test result for the best hyperparameters with and without band limitation. The best network obtained from the best models where evaluated with the mean value of *MulAcc* and *UMS* for the global measures of performance as in section 4.1 for FNN. Results for RNN models can be seen in table 4.3.

Table 4.3:
The 5 best results for RNN with different hyperparameters using the large dataset, the numbers are an average of 10 runs with their standard deviations in parentheses.

FD	HL	FB	DO	*MulAcc*	*UMS*
y	3	0.3	0.1	51.07(0.49)	44.58(0.11)
y	3	0.2	0.2	51.22(0.08)	44.85(0.47)
y	4	0.4	0.2	**51.43(0.04)**	**45.23(0.11)**

Abbreviations:
FD = Filter data (y = yes / n = no), HL = Hidden layers, FB = Forget bias, DO = dropout.

4.4 Best result from FNN, CNN, and RNN

There is a small difference in the results from FNN, CNN, and RNN. However, the best network that was obtained was a band limited FNN with 3 layers and 900 hidden units. In addition to a dropout of 20 percent and ReLU6 as activation function. This model yielded a *MulAcc* of 52.30% and a UMS of 44.08% with a standard deviation of 0.27 and 0.23 respectively. Table 4.4 show a detailed analysis with PPV and *Sen* for each specific class, where it's clear that the model has a large problem classifying CVT with a PPV of 0.50%. However, CPR shows more promising results with a PPV of 77.0%.

In Appendix A.2 there are displayed contingency tables taken from a selection of the 10 runs displayed in table 4.4 for FNN, CNN, and RNN.

Table 4.4:
The final results for the best networks, the numbers are an average of 10 runs with their standard deviations in parentheses.

			CVT		CVF	
Network	*MulAcc*	*UMS*	*PPV*	*Sen*	*PPV*	*Sen*
FNN	**52.30(0.27)**	**44.08(0.23)**	0.50(0.36)	0.68(0.7)	49.30(0.85)	63.56(5.56)
CNN	52.39(6.3)	46.47(2.19)	9.72(15.81)	15.18(58.69)	47.78(4.49)	64.96(22.88)
RNN	51.43(0.04)	45.23(0.11)	8.12(1.85)	12.44(5.57)	48.46(0.23)	52.33(0.41)

	CAS		CPR		CPE	
	PPV	*Sen*	*PPV*	*Sen*	*PPV*	*Sen*
...	41.16(0.73)	41.54(7.52)	77.0(0.04)	68.07(0.35)	62.35(2.8)	46.56(8.21)
	44.67(5.36)	41.90(26.47)	74.02(19.52)	62.97(59.70)	62.91(31.58)	47.32(8.34)
	39.17(0.06)	44.99(0.94)	78.75(0.17)	67.78(0.13)	60.37(0.28)	48.63(0.70)

Chapter 5

Discussion

In earlier work [3, 36], it has been presented promising results using neural networks as ECG rhythm classifiers. In this thesis, it has been evaluated if different neural network structures with ECG as an input can be used for ECG-based resuscitation rhythms with compression artifacts.

5.1 Data set

The dataset used in this thesis is a result of its use in previous research. A much larger dataset is desired, but the 394 patients in this data set is a good starting point for this analysis. However, since the class CVT is so underrepresented to the remaining classes, this leads to generating synthetic data. Generating synthetic data might create data that might be more similar to other classes than original data. To be able to avoid generating synthetic data, more data is desirable. In the preliminary testing described in section 3.2.2, the validation accuracy from the small- to the large-dataset had an increase of 20-30% in accuracy, this just amplifies the importance of larger datasets.

The use of longer segments could affect the performance, as the chosen 3-second segments might be too small due to unstable signals described in section 5.3. These larger segments could aid in the prediction as longer segments contain more vital information to the analysis. However, this reduces the dataset, which can lead to inadequate results.

5.2 Performance of the Classifiers

Training network on the small dataset as described in section 3.1.1, complex models tend for quicker overfitting. This is due to too few training data, resulting in too little information and the neurons are trained to detect noise or unimportant features. Running the network structure on the large dataset 3.1.2 this does not occur. Due to this, it would be better to generate a larger dataset to be used for preliminary testing.

The testing on the models was done by retraining the network 10 times to ensure that the performance is not biased on the selected data, and display the mean result as described in section 3.2.3. However, the testing might be unbiased due to the episodes vary in quality. Which can result in better or worse performance in the models by dividing data into training and test set and not using all the episodes in the dataset. Using a repeated 10-fold cross-validation might be better in some cases. Nonetheless, in this specific scenario, it can be hard to create evenly distributed 10 folds, due to data leakage leading to a false performance in the models.

5.3 Sources of Misclassification

In Appendix A.2 there are displayed contingency tables taken from a selection of the 10 runs displayed in section 4.4, table 4.4 for FNN, CNN, and RNN.

An inspection of the contingency table A.2 for FNN, show that there is a clear problem with CVT predicted as CPR. This might be due to both classes have synthetic generated data, resulting in similar data created. This is as well a problem with results from CNN and RNN shown in contingency table A.3 and A.4 respectively. However, both CNN and RNN show better results for CVT predicted as CPR than FNN.

A total inspection of all the contingency tables A.2, A.3, and A.4, reveals the main sources of misclassification. Errors due to borderline rhythms such as CVF with low amplitude and/or dominant frequency, where CAS are classified as CVF or vice versa. CPE or CPR with bradycardic rhythms or low amplitudes, where the compression dominates is predicted as CAS. Errors due to large compression artifact noise, resulting in compressions dominating over heart rhythms. Errors due to change of heart state under compression or incorrect labeling, where there are small stops in compression labeled as compression. The last two emphasize the importance of quality assurance in the review process of resuscitation episodes.

Rhythms during resuscitation are unstable and will change either spontaneously or by intervention [38]. This is a challenging problem for a short 3-second analysis window and can be addressed by using sequential classification algorithms based on the analysis of consecutive windows [20] or by using RNN for sequential classification.

Chapter 6

Conclusion and future work

Assessing ECG with artifacts is an important task in removing peri-shock pauses in cardiac arrest scenarios to increase survival rate in OHCA. For this reason, a system that can simplify the process of training different neural network models to determine heart rhythms may be useful for further research on this subject.

In this thesis, it has been developed a tool for training different neural network classifiers which can be used for ECG-based resuscitation rhythms with or without compression artifacts. An algorithm for extracting a large amount of data has been developed. This makes it possible to train deep neural networks with task specific labeled data. 3-second ECG segments with compression artifacts were extracted from 394 OHCA patients. The segments were band limited from 0.5-30 Hz and synthetic data where generated to balance the dataset.

Results in this thesis suggest that there is no clear best method using a neural network for ECG data classification. However, FNN demonstrates the most promising results with an accuracy of 52.30%. This result emphasizes the problem regarding classification of ECG data with compression artifacts, leading to future research.

6.0.1 Future work

For future work, implement transfer learning by pretraining the models with artifact-free data, or the use of restricted Boltzmann machines[39] to increase the performance[36]. Alternatively, utilize unsupervised learning such as autoencoders. By comparing results from this thesis and other ECG-rhythm classification tasks using neural networks such as [3, 36]. It's clear that more pre-processing of the data is needed, such as artifact removal by adaptive filtering[12, 2]. More data should be collected for further research, these should be collected from several ambulance systems.

Bibliography

[1] L. Fei-Fei A. Karpathy, J. Johnson. Visualizing and understanding recurrent networks. In *ICLR 2016*, arXiv:1506.02078, 2016.

[2] U. Ayala, J. Ruiz U. Irusta, J. Kramer-Johansen T. Eftestøl, E. Alonso F. Alonso-Atienza, and D.G. González-Otero. A reliable method for rhythm analysis during cardiopulmonary resuscitation. *BioMed Research International*, 2014:11, 2014.

[3] Ali Bahrami Rad, Trygve Eftestol, Kjersti Engan, Unai Irusta, Jan Terje Kvaloy, Jo Kramer-Johansen, Lars Wik, and Aggelos K Katsaggelos. Ecg-based classification of resuscitation cardiac rhythms for retrospective data analysis. *IEEE Trans Biomed Eng*, Mar 2017.

[4] J. Berdowski. Delaying a shock after takeover from the automated external defibrillator by paramedics in associated with decreased survival. *Resuscitation*, 81(3):287–292, 2010.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.*, 16:321–357, 2002.

[6] Sheldon Cheskes, Robert H Schmicker, Jim Christenson, David D Salcido, Tom Rea, Judy Powell, Dana P Edelson, Rebecca Sell, Susanne May, James J Menegazzi, Lois Van Ottingham, Michele Olsufka, Sarah Pennington, Jacob Simonini, Robert A Berg, Ian Stiell, Ahamed Idris, Blair Bigham, Laurie Morrison, and Resuscitation Outcomes Consortium (ROC) Investigators. Perishock pause: an independent predictor of survival from out-of-hospital shockable cardiac arrest. *Circulation*, 124(1):58–66, Jul 2011.

[7] Geoffrey E. Hinton David E. Rumelhart and Ronal J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533 – 536, 1986.

[8] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005.

[9] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. Wiley, New York, 2nd ed edition, 2001.

[10] Trygve Eftestøl and Lawrence D Sherman. Towards the automated analysis and database development of defibrillator data from cardiac arrest. *Biomed Res Int*, 2014:276965, 2014.

[11] Trygve Eftestøl, Kjetil Sunde, and Petter Andreas Steen. Effects of interrupting precordial compressions on the calculated probability of defibrillation success during out-of-hospital cardiac arrest. *Circulation*, 105(19):2270–3, May 2002.

[12] J. Eilevstjønn, S.O. Aase T. Eftestøl, J.H. Husøy H. Myklebust, and P.A. Steen. Feasibility of shock advice analysis during cpr through removal of cpr artefacts from the human ecg. *Resuscitation*, 61(2):131–141, 2004.

[13] J. Schmidhuber Felix A. Gers. Recurrent nets that time and count. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, 2000.

[14] Chris Fregly. Gradient descent, back propagation, and auto differentiation - advanced spark and tensorflow meetup. "`https://www.slideshare.net/cfregly/gradient-descent-back-propagation-and-auto-differentiation-advanced-spark-and-tensorflo` Aug 2016.

[15] Klaus Greff, Rupesh K Srivastava, Jan Koutnik, Bas R Steunebrink, and Jurgen Schmidhuber. Lstm: A search space odyssey. *IEEE Trans Neural Netw Learn Syst*, Jul 2016.

[16] F. Gustafsson. Determining the initial states in forward-backward filtering. *IEEE Transactions on Signal Processing*, 44(4):5, 1996.

[17] A P Hallstrom, J P Ornato, M Weisfeldt, A Travers, J Christenson, M A McBurnie, R Zalenski, L B Becker, E B Schron, M Proschan, and Public Access Defibrillation Trial Investigators. Public-access defibrillation and survival after out-of-hospital cardiac arrest. *N Engl J Med*, 351(7):637–46, Aug 2004.

[18] Meiso Hayashi, Wataru Shimizu, and Christine M Albert. The spectrum of epidemiology underlying sudden cardiac death. *Circ Res*, 116(12):1887–906, Jun 2015.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9, 1997.

[20] Unai Irusta, Jesús Ruiz, Elisabete Aramendi, Sofía Ruiz de Gauna, Unai Ayala, and Erik Alonso. A high-temporal resolution algorithm to discriminate shockable from nonshockable rhythms in adults and children. *Resuscitation*, 83(9):1090–7, Sep 2012.

[21] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. *Journal of Machine Learning Research*, 2015.

[22] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. "`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`", May 2015.

[23] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. Leakage in data mining: Formulation, detection, and avoidance. *ACM Trans. Knowl. Discov. Data*, 6(4), December 2012.

[24] K.B. Kern, R.A. Berg R.W. Hilwig, and G.A. Ewy A.B. Sanders. Importance of continuous chest compressions during cardiopulmonary resuscitation. *Circulation*, 105(5):645–649, 2002.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR 2015*, 12 2014.

[26] Vessela Krasteva, Irena Jekova, Ivan Dotsinsky, and Jean-Philippe Didon. Shock advisory system for heart rhythm analysis during cardiopulmonary resuscitation using a single ecg input of automated external defibrillators. *Ann Biomed Eng*, 38(4):1326–36, Apr 2010.

[27] ... M. Abadi. Tensorflow: Large-scale machine learning on heterogeneous distributed system. Preliminary White Paper, November 2015.

[28] Tanis Mar, Sebastian Zaunseder, Juan Pablo Martínez, Mariano Llamedo, and Rüdiger Poll. Optimization of ecg classification by means of feature selection. *IEEE Trans Biomed Eng*, 58(8), Aug 2011.

[29] Dr. James McCaffrey. Neural network train-validate-test stopping. "`https://visualstudiomagazine.com/Articles/2015/05/01/Train-Validate-Test-Stopping.aspx?Page=1`", May 2015.

[30] E.P. Rivers N.A. Paradis, G.B. Martin. Coronary perfusion pressure and the return of spontaneous circulation in human cardiopulmonary resuscitation. *JAMA*, 263(8):1106–1113, 1990.

[31] J. Nagi, G. A. Di Caro F. Ducatelle, U. Meier D. Ciresan, F. Nagi A. Giusti, J. Schmidhuber, and L. Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *IEEE International Conference on Signal and Image Processing Applications (ICSIPA2011)*, 2011.

[32] Robert Nisbet, John F Elder, and Gary Miner. *Handbook of statistical analysis and data mining applications*. Academic Press/Elsevier, Amsterdam, 2009.

[33] Christopher Olah. Understanding lstm networks. "`http://colah.github.io/posts/2015-08-Understanding-LSTMs/`", August 2015.

[34] Alan V. Oppenheim, Ronald W. Schafer, and John R Buck. *Discrete-time signal processing*. Prentice Hall, Upper Saddle River, N.J., 2nd ed edition, 1999.

[35] Gavin D. Perkins. European resuscitation council guidelines for resuscitation 2015: Section 2. adult basic life support and automated external defibrillation. *Resuscitation*, 95:81–99, 2015.

[36] V.J.R Ripoll, E. Romero A. Wojdel, and J. Brugada P. Ramos. Ecg assessment based on neural networks with pretraining. *Applied Soft Computing*, 49:399–406, 2016.

[37] Dominik Scherer, Andreas Müller, and Sven Behnke. *Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition*, pages 92–101. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[38] Eirik Skogvoll, Trygve Eftestøl, Kenneth Gundersen, Jan Terje Kvaløy, Jo Kramer-Johansen, Theresa Mariero Olasveengen, and Petter Andreas Steen. Dynamics and state transitions during resuscitation in out-of-hospital cardiac arrest. *Resuscitation*, 78(1):30–7, Jul 2008.

[39] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. *Parallel Distributed Processing*, 1:194–281, 1987.

[40] J. Soar. European resuscitation council guidelines for resuscitation 2015: Section 3. adult advanced life support. *Resuscitation*, 95:100–147, 2015.

[41] Leif Sörnmo and Pablo Laguna. *Bioelectrical signal processing in cardiac and neurological applications*. Elsevier Academic Press, Amsterdam, 2005.

[42] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, June 2014.

[43] CS231n Standford University. Cs231n: Convolutional neural networks for visual recognition. "`http://cs231n.github.io/convolutional-networks/`", 2017.

[44] R.M. Sutton, M.R. Maltese S.H. Friess, G. Bratinov M.Y. Naim, and T.R. Weiland. Hemodynamic–directed cardiopulmonary resuscitation during in–hospital cardiac arrest. *Resuscitation*, 85(8):983–986, 2014.

[45] Inc The MathWorks. *Signal Processing Toolbox User Guide*. The MathWorks, Inc, 3 Apple Hill Drive, 2017.

[46] Lars Wik, Jo Kramer-Johansen, Helge Myklebust, Hallstein Sørebø, Leif Svensson, Bob Fellows, and Petter Andreas Steen. Quality of cardiopulmonary resuscitation during out-of-hospital cardiac arrest. *JAMA*, 293(3):299–304, Jan 2005.

[47] Y. Bengio Y. Lecun, L. Bottou and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

Appendix A

Performance metric and contingency tables

A.1 Performance metric

The performance metric can be derived by looking at the contingency table shown in table A.1, where $N_{i,j}$ represents the number of segments of class i (true label) classified as class j (predicted label). For this study $i,\ j \in \{CVT,\ CVF,\ CAS,\ CPR,\ CPE\} \equiv \{1,\ 2,\ 3,\ 4,\ 5\}$. Further the total number of true samples in class i and predicted class j which is denoted as T_i and P_j respectively[28, 3], can be expressed as:

$$T_i = \sum_{j=1}^{5} N_{i,j} \quad \text{and} \quad P_j = \sum_{i=1}^{5} N_{i,j} \tag{A.1}$$

And the total amount of segments can be calculated as:

$$N_{tot} = \sum_{i}^{5}\sum_{j}^{5} N_{i,j} = \sum_{ji}^{5} T_i = \sum_{j}^{5} P_j \tag{A.2}$$

Other relevant factors are the true positive value (TP), false positive (FN), false negative (FN), and true negative (TN) for each class i can be expressed as:

$$\begin{aligned} TP_i &= N_{i,i} \\ FP_i &= P_i - N_{i,i} \\ FN_i &= T_i - N_{i,i} \\ TN_i &= N_{tot} - TP_i - FP_i - FN_i \\ &= N_{tot} - T_i - P_i + N_{i,i} \end{aligned} \tag{A.3}$$

Table A.1:
Contingency table for classification of resuscitation cardiac rhythms with artefacts

		Prediction Label					
		CVT	CVF	CAS	CPR	CPE	$\sum$
True Label	CVT	$N_{1,1}$	$N_{1,2}$	$N_{1,3}$	$N_{1,4}$	$N_{1,5}$	T_1
	CVF	$N_{2,1}$	$N_{2,2}$	$N_{2,3}$	$N_{2,4}$	$N_{2,5}$	T_2
	CAS	$N_{3,1}$	$N_{3,2}$	$N_{3,3}$	$N_{3,4}$	$N_{3,5}$	T_3
	CPR	$N_{4,1}$	$N_{4,2}$	$N_{4,3}$	$N_{4,4}$	$N_{4,5}$	T_4
	CPE	$N_{5,1}$	$N_{5,2}$	$N_{5,3}$	$N_{5,4}$	$N_{5,5}$	T_5
$\sum$		P_1	P_2	P_3	P_4	P_5	N_{tot}

Numerical class equivalences are:
1 = CVT, 2 = CVF, 3 = CAS, 4 = CPR, 5 = CPE

Further by using TP_i, Sen_i and PPV_i can be expressed as:

$$Sen_i = \frac{TP_i}{T_i} = \frac{N_{i,i}}{T_i} \quad \text{and} \quad PPV_i = \frac{TP_i}{P_i} = \frac{N_{i,i}}{C_i} \tag{A.4}$$

MulAcc and *UMS* are used as a summarize of the full confusion matrix into a single parameter and facilitates the selection of the best performing model[28], they can be expressed as:

$$MulAcc = \frac{1}{N_{tot}} \sum_{i=1}^{5} N_{i,i} \quad \text{and} \quad UMS = \frac{1}{5} \sum_{i=1}^{5} Sen_i \tag{A.5}$$

A.2 Contingency tables for the best network models

A.2.1 Contingency table for FNN

		Prediction Label					
		CVT	CVF	CAS	CPR	CPE	$\sum$
True Label	CVT	24	1	0	6019	0	6044
	CVF	37	19115	8742	11	2168	30073
	CAS	14	11025	12817	17	6144	30017
	CPR	8064	1104	81	20051	531	29832
	CPE	28	6819	9515	49	13927	30338
$\sum$		8168	38064	31155	26147	22770	126304

Table A.2: Contingency table for classification of resuscitation cardiac rhythms with artifacts for the best FNN model

A.2.2 Contingency table for CNN

		Prediction Label					
		CVT	CVF	CAS	CPR	CPE	$\sum$
True Label	CVT	1352	13	61	2862	1756	6044
	CVF	153	18839	8279	69	2734	30074
	CAS	32	13303	11562	41	5077	30015
	CPR	8277	1461	219	17997	1877	29831
	CPE	128	5670	7095	2232	15215	30340
$\sum$		9942	39286	27216	23201	26659	126304

Table A.3: Contingency table for classification of resuscitation cardiac rhythms with artifacts for the best CNN model

A.2.3 Contingency table for RNN

		Prediction Label					
		CVT	CVF	CAS	CPR	CPE	$\sum$
True Label	CVT	1027	26	2	4904	85	6044
	CVF	43	15718	12046	45	2221	30073
	CAS	45	9817	13658	33	6465	30018
	CPR	8427	775	27	20042	560	29831
	CPE	25	6454	9379	200	14280	30338
$\sum$		9567	32790	35112	25224	23611	126304

Table A.4: Contingency table for classification of resuscitation cardiac rhythms with artifacts for the best RNN model

Appendix B

Program files

B.1 Matlab code

The following MATLAB files are embedded as matlab.zip 

getOHCAdata.m
The main script, specify parameters, creates data struct and runs the entire code

getAnnoatedIndexes.m
Inputs: Signal and sample annotations and specification parameters. Outputs all indexes for all five classes.

getAnnotatedSignals_Samples.m
Inputs: Indexes for all five classes and the patient episodes. The function outputs all annotated signals and samples for each specific classes.

getOrigSeg_Ecg.m
Inputs: All annotated signals and samples for each specific class. Creates 3-second segments and start and end sample values for the segments for each class. This function creates the small dataset described in 3.1.1.

getOrigSeg_Ecg_Big.m
Inputs: All annotated signals and samples for each specific class. Creates non-overlapping 3-second segments and start and end sample values for the segments for each class. This function creates the large dataset described in 3.1.2.

data2csv.m
Inputs: data struct, the function finds out how many patients have segments and store data in an array.

cell2csv.m
Inputs: store data vector from *data2csv*, the function create a CSV file from the content in the data array. source: MathWorks.

SMOTE.m

Creates synthetic data, source: MathWorks.

nearestneighbour.m

Computes the k-nearest neighbour, source: MathWorks.

filterData.m

Inputs: data struct, filters the data and returns the result.

createDataset_BIG.m

Inputs: data struct and parameters for filtering and generating synthetic data, and the patient id for training and test data. The function creates training and test set containing all five classes with the corresponding labels. As explained in detail in section 3.1.2

createDataset.m

Inputs: Parameters for filtering and the segments that are to be removed. The function creates training and test set containing all five classes with the corresponding labels. As explained in details in section 3.1.1

viewSister.m

GUI for analyzing segments generated, source: [3]

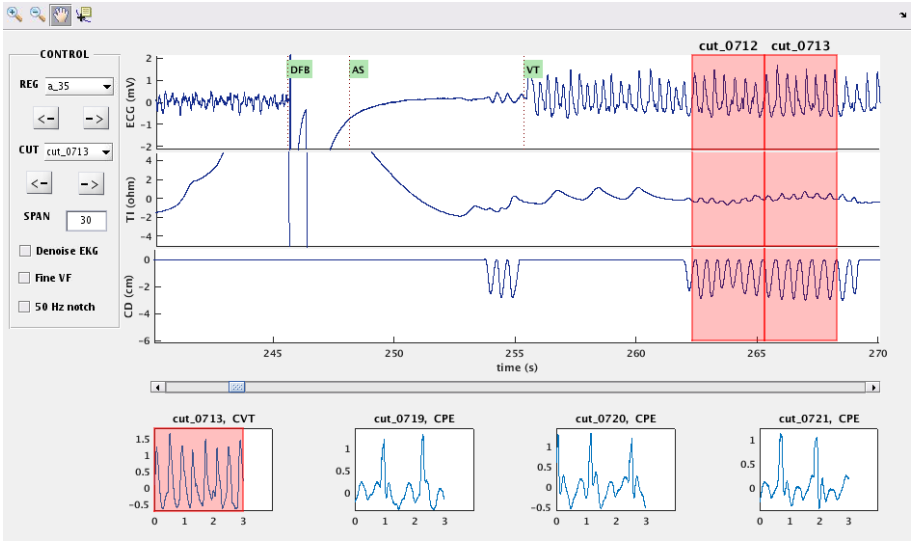

Figure B.1: GUI for analyzing segments in dataset, CD = compression depth, TI = Thoracic Impedance.

B.2 Python code

The following Python files are embedded as python.zip

Main.py

The main script, specify parameters such as network structure, log directory, and dataset. The script creates a log directory and stores tensorboard and models in subdirectories. The code then runs either with different hyperparameters or evaluate specific models as described in section 3.2. *Main.py* is visualized in figure 3.4

some_functions.py

A script that contains functions as:

- load_dataset
- load_dataset_Big
- shuffle_data_set
- create_mini_batches
- create_validation_set
- fc_layer, creates a fully connected layer
- conv_layer, creates a convolutional connected layer
- multi_rnn_layer, creates n-deep layer consisting of LSTM cells
- rnn_layer, creates 1 layer consisting of LSTM cells
- make_hparam_string, creates a hyperparameter string

FNN_ECGdata.py, CNN_ECGdata.py, RNN_ECGdata.py

Script to create and train FNN, CNN, RNN model(s), and shuffles the dataset and creates validation set. It then creates cost functions and training optimizers. In addition to creating file writers to store data to be able to visualize the network and its content in tensorboard[1]. The model is then trained for the specified amount of epochs and stores the best model and does an early stopping as explained in section 2.2.4. The model can either train on different hyperparameters or evaluates on the test data and calculates the performance of a confusion matrix as described in section 3.2.3.

eval.py

This script loads a previously saved model without creating the network structure and evaluates its performance. The code can also be rewritten to use the best model as a classifier. This code has not been used in the final process in results but is added for future work.

B.2.1 Prerequisites

To be able to run the code in Python, the following libraries needs to be installed.

- python v.3.5
- tensorflow
- os
- datetime, time, logging

[1]See Appendix B.2.2 for details

- numpy
- pandas_ml
- math
- h5py

B.2.2 TensorFlow

The fastest growing open source software library for machine learning is developed by Google and are called TensorFlow. "TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms"[27]. It was originally designed to meet Google's need for systems capable of building and training neural networks to meet the requirements of creating advanced machine learning tools. Such as speech recognition systems, computer vision, text analytics, and search. It's a software that is both used for research and production and is created such that it's easy to go from research to implementing it directly into products without having the need to rewrite code. The networks constructed in TensorFlow can be run on GPU for faster training.

Tensoboard

In addition, TensorFlow comes with a so-called *flashlight* to the black box hidden layers are in neural networks, called *tensorboard*. Tensorboard gives information about the network such that it's possible to visualize the code. This can be seen in figure B.3 and B.3. Tensorboard makes it possible to see weights, biases being trained and various other important elements.

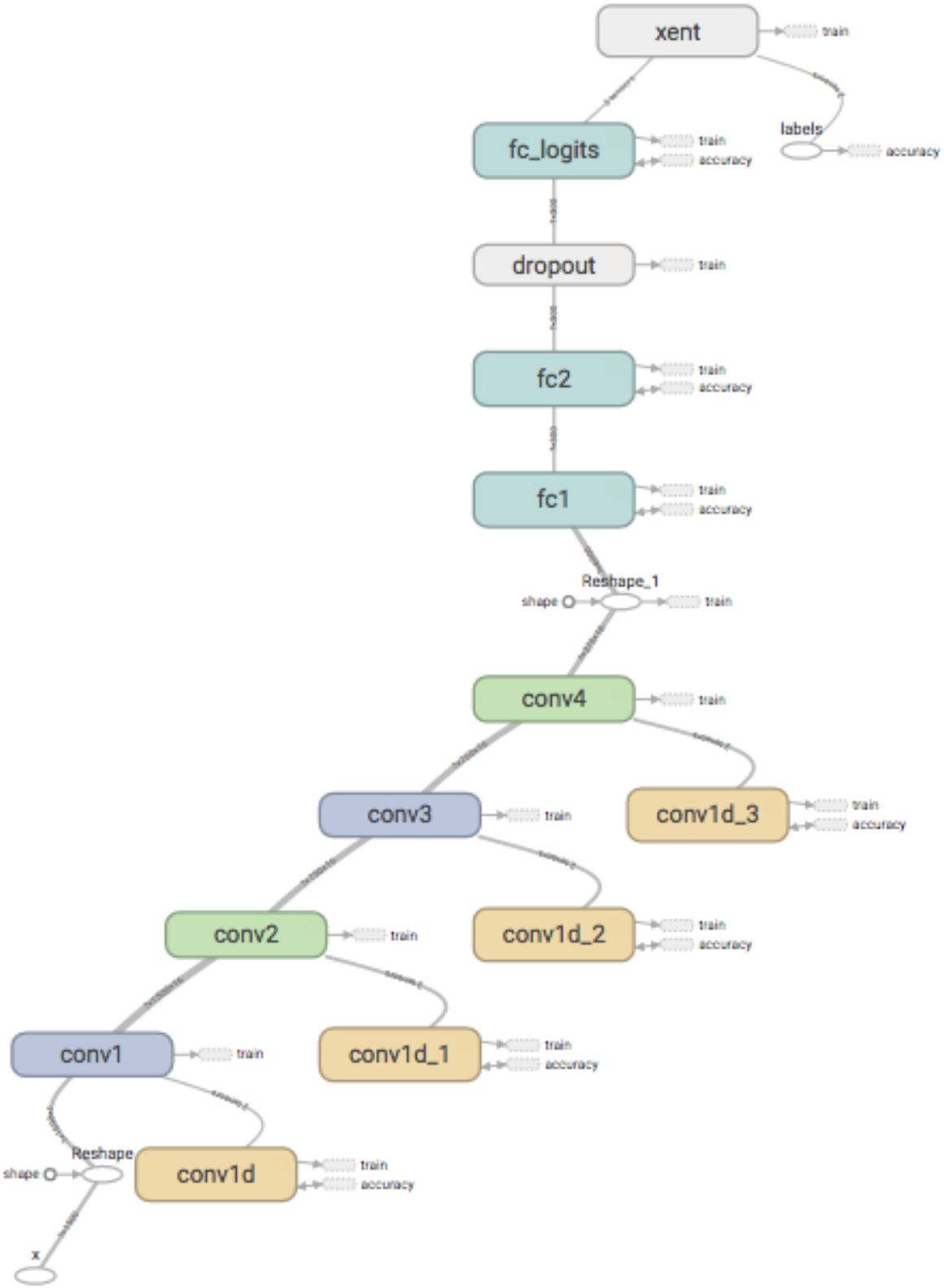

Figure B.2: Tensorboard's visualization of the best CNN network structure

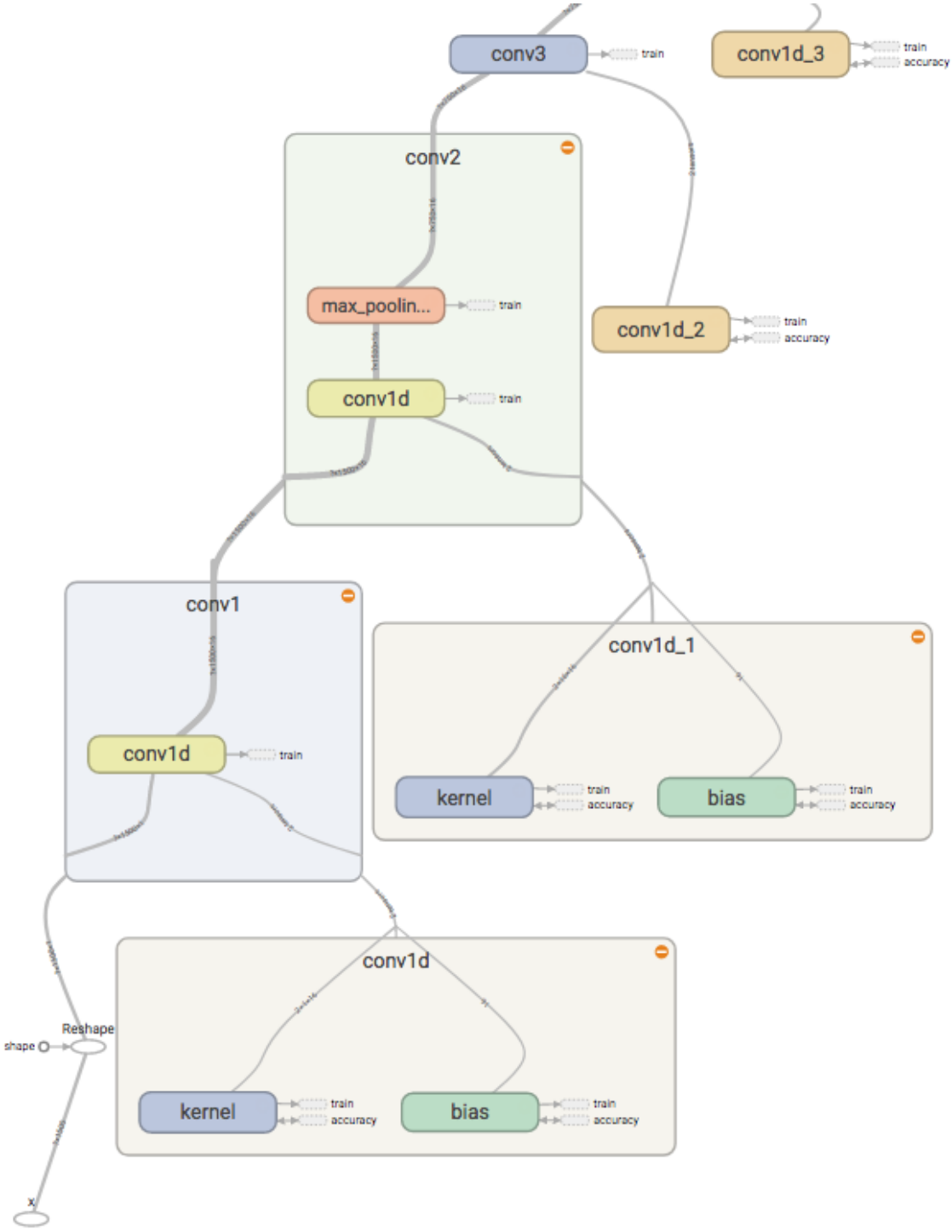

Figure B.3: Tensorboard's visualization of the best CNN network structure, where some boxes are opened for details