Universitetet i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: Department of Electrical Engineering and Computer Science: Robot technology and Signal Processing	Spring / Autumn semester, 20.19. Open/~~Confidential~~
Author: Ruth Hognestad	........Ruth Hognestad........ (signature of author)

Programme coordinator:

Supervisor(s):

Kjersti Engan, Tonje Soraas Birkenes and Helge Myklebust.

Title of master's thesis:
Telephone CPR Instructions in Cardiac Arrest.

Norwegian Title:
Telefon HLR instruksjoner ved hjertestans.

Credits: 30

Keywords: Speech recognition, Mel Frequency Cepstral Coefficient , Convolutional Neural Network, Simulations with Akuttmedisinsk kommunikasjonssentral.	Number of pages: 58.................. + supplemental material/other: 17........ Stavanger, 15.06.2019........... date/year

Title page for Master's Thesis
Faculty of Science and Technology

Universitetet
i Stavanger

Telephone CPR Instructions in Cardiac Arrest

Master Thesis

Ruth Hognestad

Spring 2019

Under the supervision of and Professor Kjersti Engan(UiS),

Tonje Soraas Birkenes(Leardal medical) and Helge Myklebust(Leardal medical).

Department of Electrical Engineering and Computer Science

Robot technology and Signal Processing

University of Stavanger

Abstract

This thesis focus on developing a dataset of recordings between a caller and a dispatcher from Emergency Communication Centre during situations involving cardiac arrest. It also focused on developing and implementing a speech recognition system that would analyze 10 keywords in the dataset.

One of the methods tested was to design a convolutional neural network to identify the small 1 second keywords. A separate dataset with these keywords were developed and model was trained and validated.

A speech segmentation algorithm was developed to identify the boundaries of the words in the dataset. The algorithm analyzed the different energy levels in the signal to separate words from silence and to find the word boundaries. These words where then classified with the CNN model. If the likelihood of the word belonging to one of the classes was less then 99.99 %, then the word was not classified. However, if the likelihood was more then 99.99 % the word was classified and the output was written as text.

The other method was to find the keywords using an existing speech recognition system. Google API speech recognition system was used to transcribe the recordings to text.

Both the transcriptions from both of the methods were compared with the real transcription. The results gave a word error rate of 76.4% and false alarm rate of 282% for the CNN model and 32.48 % and 1.91% for the Google model.

In conclusion the Google speech recognition model was better at transcribing the dataset then the developed CNN model.

Preface

This is a master's thesis in Robottechnology and signal processing at the University of Stavanger. The topic for this Telephone CPR Instructions in Cardiac Arrests.

I would like to thank my supervisors, Kjersti Engan, for valuable discussions and support to successfully accomplish my MSc Thesis. I would also like thank my supervisors from Leardal Medical AS, Tonje Soraas Birkenes and Helge Myklebust, for suggesting this master thesis and setting up the simulations with Emergency Communication Centre.

Finally I would like to thank all the volunteers at University of Stavanger, Leardal Medical and Emergency Communication Centre for participating in the simulations and data collections. And final I would personally give thanks to Helene Lund from AMK.

Stavanger 15 June 2019

Ruth Hognestad

Contents

List of Figures

Contents

Abbreviations

CNN	*Convolutional neural network*
CPR	*Cardiopulmonary resuscitation*
ECC	*Emergency Communication Centre*
MFCC	*Mel-frequency cepstral coefficients*
NARKOS	*Nasjonal kompetansetjeneste for prehospital akuttmedisin*
NN	*Neural network*
RSM	*Short-term root mean square*
UiS	*University of Stavanger*
WER	*Word error rate*

Chapter 1

Introduction

The *Telephone CPR Instructions in Cardiac Arrest*, was in the spring of 2019 suggested as a master thesis by Laerdal Medical for students at the university of Stavanger.

1.1 Motivation

Emergency Communication Centre(ECC) is responsible for operating the emergency number 113 and the ambulance services in Norway [1]. There are the ones how receives a telephone inquiry about a medical emergencies.

When receiving a telephone inquiry they would use questions from the manuscript *Norwegian index for medical emergencies* to get important information from the caller. The questions they ask are: *Where is the patient?*, *Where is the accident?*, *What number are you calling from?*, *Does the patient reactis the patient breathing?* [2].

If the caller notice that the patient is not responding and breathing it is possible that the patient is suffering from cardiac arrest. It is important in this situation for the caller to start first aid Cardiopulmonary resuscitation (CPR) on the patient immediacy as it can increase survival rate[3].

If there indeed is a case of cardiac arrest, the dispatcher will ask the caller to start CPR on the patient. The dispatcher will ask them start 30 compressions and 2 breaths or a set of continuous compressions depending on weather the caller knows CPR or not. During the conversion the dispatcher will also give the caller feedback, asking them to *press fast*, *press slower* and *count with me.* This is done to make sure that the compression rate is within 100 a minute.

When doing compressions on a patient it is important that the compression rate is within 100 a minute. Dispatcher will get some feedback from the rate of counting, but that is not always good enough. Noise, bad reception and or stressed callers can sometimes give inaccurate feedback. For this reason Laerdal Medical have together with students at university of Stavanger developed an app **TCPR-link** to visualize the compression rate.

TCPR link was first launched in 2017 at Emergency Cardiovascular Care Update.[4] The purpose of the app is measure and visualize the compression rate during chest compressions and determine whether compression rate is in-between 100 and 120 compressions per minute. The app uses a smart-phone camera to analyze the change in head position and uses this information to estimate the compression rate. [5]

The compression rate is visualized by a speedometer on the application. The arrow can during chest compressions move up or down, visualizing the compression rate. If the arrow is within the green line, the compression is between 100 and 120 compressions per minute. If the arrow is below or above the green region, the compression rate is to slow or too fast. Figure 1.1 is an image of the app in use.

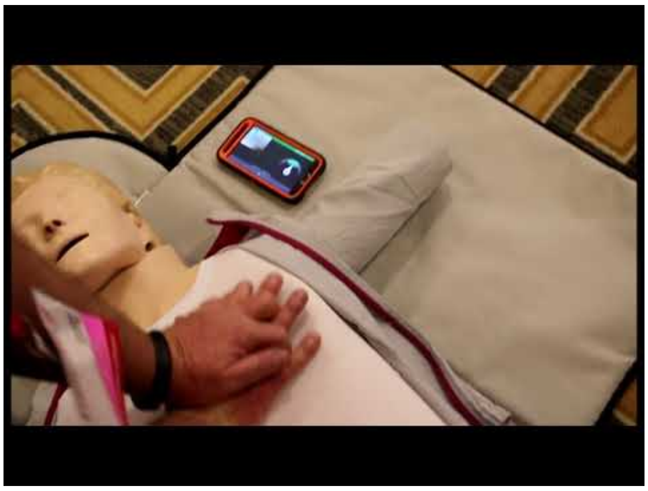
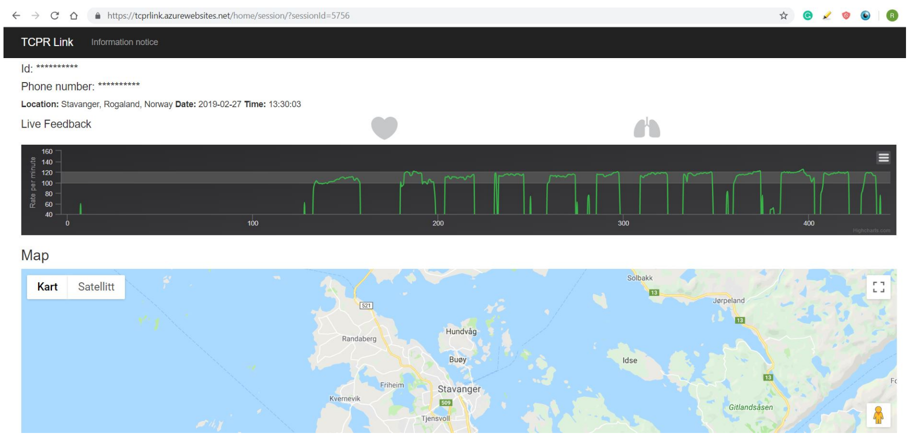


Figure 1.1: Testing of TCPR link.
Source: **https://www.youtube.com/watch?v=jD9y7gOEIpY**, 2019.

The application also sends the real-time compression rate, downtime and the phones GPS coordinates to the web-server, **tcprlink**. The web-server visualize the compression rate with a graph and the GPS coordinates with a map pointing to the location of the call. See image 1.2.


https://tcprlink.azurewebsites.net/home/session/?sessionId=5756

TCPR Link Information notice

Id: **********

Phone number: **********

Location: Stavanger, Rogaland, Norway **Date:** 2019-02-27 **Time:** 13:30:03

Live Feedback

Map

Figure 1.2: Graph showing compression rate and location of app user.
Source: **https://tcprlink.azurewebsites.net/home/session/?sessionId=5756**, 12.05.2019.

The graph on the web server shows the compression rate on the y-axis in rates per minute and the time in seconds. If the green line lies between the white row in the figure, the compression rate is in between 100 and 120 compressions per minute. However, if the compression rate is outside the line the compression rate is incorrect, and the compression rate must increase or decrease to follow the correct procedure.

This information can be accessible by both the caller and the dispatcher during training or call to 113 if the app is activated during the conversation. If the dispatcher asks the caller to do CPR, he or she can get access to their performance by going to the website and looking at the compression rate. Thereby the dispatcher gets some extra feedback to how the caller is doing and might give help the caller preforming CPR.

1.2 Statement of problem

Per today the dispatchers have two tools to tutor caller in out-off hospital CPR, sound from the microphone and the visualization of the compression rate from the TCPR-app. But there is still not a method for getting feedback for what have being said and by how during the of the conversation.

Leardal wish to *investigate the possibility of recognizing keywords used to modify cardiac arrest (e.g. press deep, press fast, count with me) and associations between instructions and CPR performance.* In order words they wish to record a conversation between the dispatcher and caller and use speaker recognition and speaker verification to identify what is spoken.

By logging and displaying the frequency of specific words uttered by the dispatcher during the phone conversation, the dispatcher can determine whether the caller understands the words used in the situation or if he/she should use different words to guide the caller preforming CPR. It is the hope that this feature can improve the response-time and effectiveness of the CPR and time for the caller to understand the situation and start preforming CPR. Good communication between the dispatcher and the caller is therefore important for making sure the CPR is started as fast as possible.

1.3 Current available technology

There a lot existing solutions in the field of speech recognition. Including Google's speech recognition system and Amberscript.

Google have launched a Cloud Speech-to-Text enables developers to convert audio to text by *applying powerful neural network models in an easy-to-use API* [6]. For example can the Speech-to-Text **Default** model transcribe a number of language from a single speaker. This includes Norwegian bokmål (NO) and English(EN). Google API model also have the opportunity to transcribe audio from a microphone giving output in real-time [7].

Amberscript is transcriber service that can transcribe audio files from 27 different languages including Norwegian and English. Amberscipt used AI technology to automatically transcribe speech to text [8]. To use Amberscript the user can upload a audio file of any length to the web server, choose the language and the number of speakers and get a result after a some minutes depending how long the audio file is. The output file will give information about who spoke, what they spoke and what time they spoke.

1.4 Proposed system overview

Figure 1.3 shows the system overview. The callers telephone records the conversation between the dispatcher and caller during the an emergency situation involving heart failure. While the telephone records the audio the TCPR-link is activated, sending data to the webside. After or during the conversation the mobile will transfer the audio to a web server. The web server will analyse who is speaking and what they are speaking in the audio file. This information is accessible from the dispatchers monitor where he or she will get a overview over words spoken.

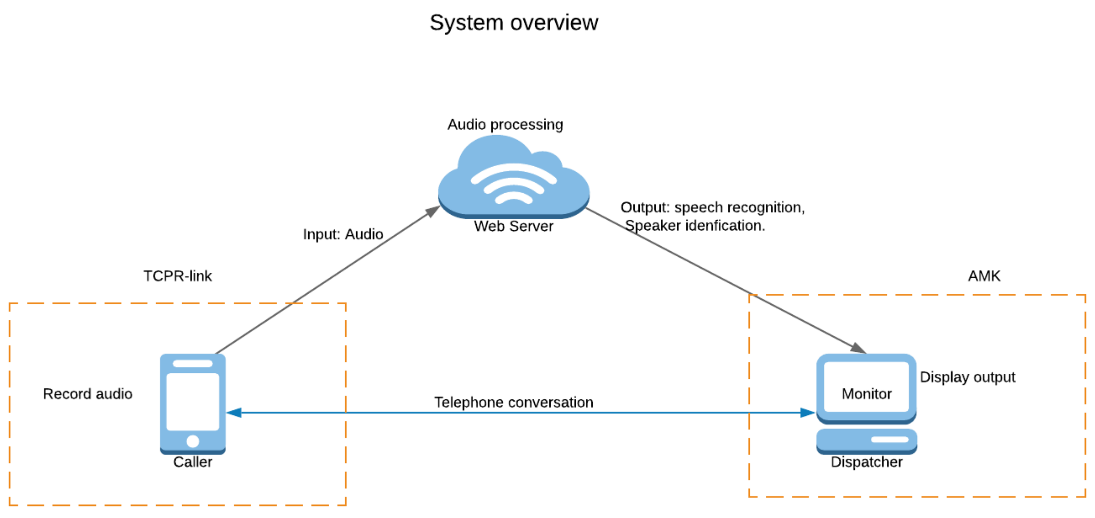


Figure 1.3: System overview

1.5 Objectives

The objectives chosen to focus on in this thesis:

- Record a telephone conversation between a caller and a dispatcher.
- Transcribe the conversation.
- Find keywords in the recording using a speech recognition method.
- Train a neutral network to find keywords in the recordings.
- Develop a dataset of keywords that can be used for training the neutral network.
- Compare the created model with Google's speech recognition model.

1.6 Thesis outline

- **Chapter 1 - Introduction:**
  The introduction chapter goes over the motivation and the objectives for the master thesis.
- **Chapter 2 - Technical Background:**
  The chapter background describes relevant theory used to develop a method of implementation.
- **Chapter 3 - Data development:**
  This chapter describes the process of recording a call between a caller and dispatcher during simulated scenarios. Also describes the development of data of keyword utterance.
- **Chapter 4 - Method:**
  This chapter describes the approach for training a neutral network on keywords. It also describe the process of identifying the keywords in the simulated scenario using the neutral network and a Google's speech recognition model.
- **Chapter 6 - Experiments and Results:**
  The chapter results describes the results and experiments done to finish the goals describe in the method.
- **Chapter 8 - Discussion:**
  This chapter summarize and discuss the results develop in the previous chapter.
- **Chapter 8 - Conclusion:**
  This chapter summarize everything done in the Master thesis.

Chapter 2

Technical Background

The chapter Technical background give a description of a methods used for speech recognition.

2.1 Speech Recognition

Speech recognition is a machine learning method for a machine to identifying and classify human speech. The purpose of speech recognition is to translate spoken language into a language a machine can understand and respond to. A response could be for the machine to perform a specific task like transcribing spoken language, answer a question, open a file and more. Applications that uses this technology uses speech as an input to their system in the same way that a keyboard uses mechanical switches as an inputs to an computer.

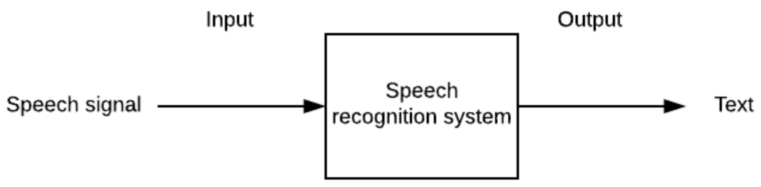


Figure 2.1: Speech Recognition system

Figure 2.1 describes a simple speech recognition system. Given a audio input X, the speech recognition system S makes a prediction Y to which class in the reference model the input audio X belongs to. The output from the prediction is given in form of text.

The following section gives a description of a simple speech recognition system model. The range of information will range for preparing the data for classification, creating a classification model and testing a classification model on continuous audio. An overview of the methods can be seen in figure 2.2.

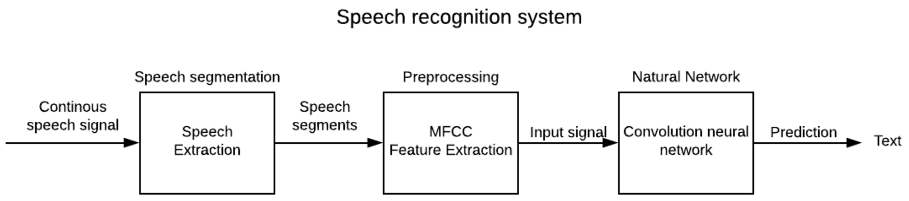


Figure 2.2: A simple description of a speech recognition system. Spoken language is used as an input to a application, the application makes a prediction of given output, and gives an output.

2.2 Preprocessing

Preprocessing involves preparing the input data for classifications. Recorded samples never produce identical waveforms; the length, amplitude, background noise may vary. It is therefore important to perform signal preprocessing to extract only the speech related information. This means that using the right features are crucial for successful classification
citepgevaert2010neural.

A normal speech signal have a high variance due to different speakers, speaking rates, contents and acoustic conditions. They are not sufficient enough to be used for classification. Instead segments or features are extracted from the original signal. This is process is called speech extractions and involves extracting information form the images that can be used for classification. Power, pitch, and vocal tract configuration are examples of the information that can be extracted from the signal and used for classification. [9].

The are many methods for feature extraction including a power spectrum analysis (FFT), linear predictive analysis (LPC), Linear predictive cepstral coefficients (LPCC) and Mel-frequency cepstral coefficients (MFCC). However, the standard method for feature extraction is Mel-frequency cepstral coefficients since it has the best performance [9].

2.2.1 Mel-frequency Cepstral Coefficient (MFCC)

Mel-frequency Cepstral Coefficients was designed replicates human perception of speech. The human ear can hear frequencies in range 20 Hz to 20 kHz and tends resolve frequencies non-linearly. [10]. Unlike a normal FFT spectrogram which represents frequencies linearly, a MFCC spectrogram represents the frequencies in signal non-linearly making it a better representation of how humans understand speech.

The processes of extracting MFCC from a speech signal can be divided in 6 parts: *Pre-Emphasis*, *Frame blocking*, *Windowing*, *FFT*, *Filter Banks* and *Discrete cosine transform(DCT)* [11]. A overview of this process can been seen in figure 2.3.

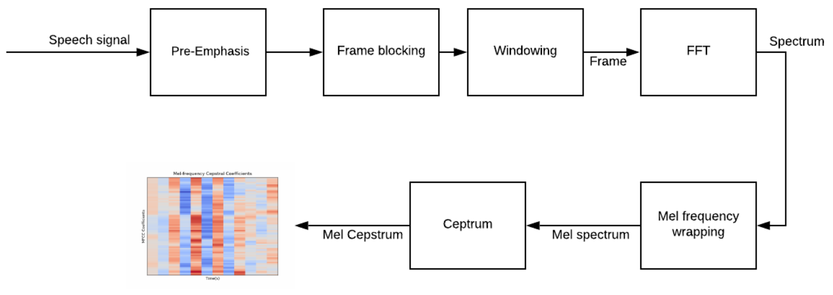


Figure 2.3: Creating MFCC

The first part of the process is to emphasis the frequencies in the speech signal using the equation 2.1 to amplify the signal. This balances the frequency spectrum, avoid numerical problem and possible improves the Signal-to-Noise ratio.[11]

$$H(z) = -1 + (-0.95z^{-1}) \tag{2.1}$$

Figure 2.4 show how the input signal is emphasized. The y-axis shows the amplitude and x-axis shows the time.

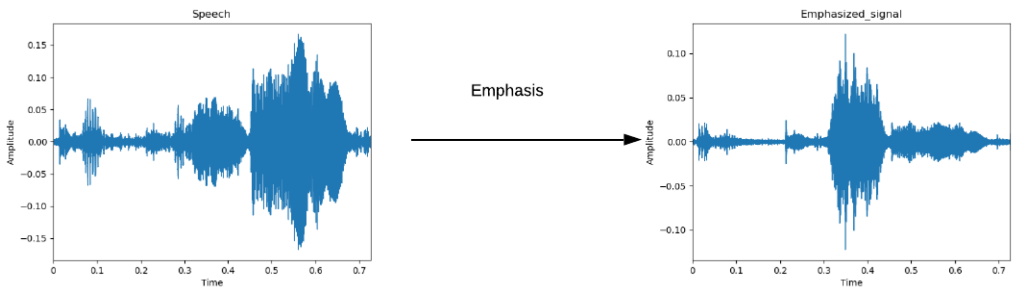


Figure 2.4: Emphasize input signal

The emphasized signal is split into short time frames. This gives a better representation of frequencies since they change over time. A typical frame size in speech processing ranges from 20 ms to 40 ms with a 50 % (+/-10%) overlap between consecutive frames. [12] In this example the frame size is set to 25 ms and frame step to 10 ms. This means that signal sampled with 16000 HZ have the frame size of 0.025*16000=400 samples and the frame step is set to 160 samples.

When the signal have been framed into smaller windows the discrete Fourier transform is calculated for each frame using equation 2.2. The number of samples N is typical between 256-512 and the filter h is usually defined as a hamming window as seen in 2.3.[12] Small s is the the signal frames.

$$S_i = \sum_{n=1}^{N} s_i(n)h(n)e^{\frac{-j2\pi kn}{N}} 1 \leq k \leq K \tag{2.2}$$

$$h[n] = 0.54 - 0.46cos(\frac{2\pi}{N-1}) \tag{2.3}$$

After the Fourier transform is calculated on each frame, the power of each of the frames is calculated.

$$P_i(k) = \frac{1}{N}|S_i(k)|^2 \tag{2.4}$$

The next step is to convert the frequencies in the power spectrum to Mel scale. The Mel-scale aims to mimic the non-linear human ear perception of sound, by being more discriminative at lower frequencies and less discriminative at higher frequencies[11]. This is done by using a set of 40 triangular filter banks to extract the frequencies bands. Equation 2.5 and 2.6 shows the conversion between f(Hz) and Mel(m) spectrogram.

$$m = 2595log_{10}(1 + \frac{f}{700}) \tag{2.5}$$

$$f = 700(10^{\frac{m}{2595}} - 1) \tag{2.6}$$

The filter banks are created with the equation **??**. The first filter-bank will start at zero at the first point, reach its peak at its second point and return to zero at its third point. This continuous until the filter bank have covered all the samples in spectrum. Se figure 2.5.

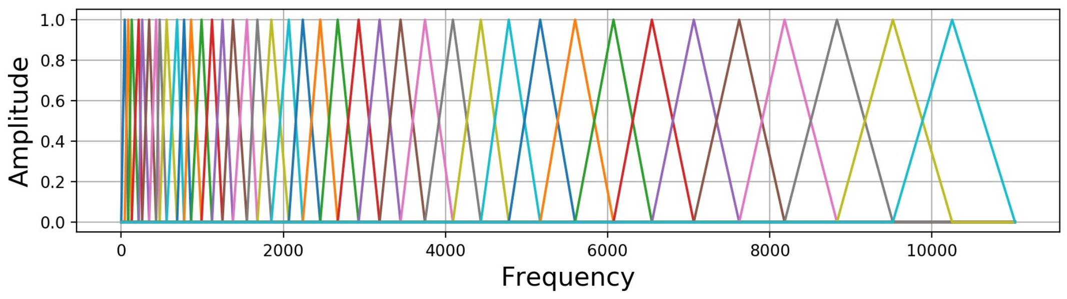


Figure 2.5: A mel scale filter bank

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \frac{k-f(m-1)}{f(m)-f(m-1)} & f(m-1) \leq k < f(m) \\ 1 & k = f(m) \\ \frac{f(m+1)-k}{f(m+1)-f(m)} & f(m) < k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{2.7}$$

The final step is to compress and decorrelate the new spectrogram created with the filter-banks. This is important to remove the correlations between the filter bank coefficients that can be problematic in some automatic speech recognition algorithms. A discrete cosine transform (DCT) is applied to the MFCC coefficients.

The final spectrum can be seen in figure 2.6. The spectrum is a representation of how human hears.

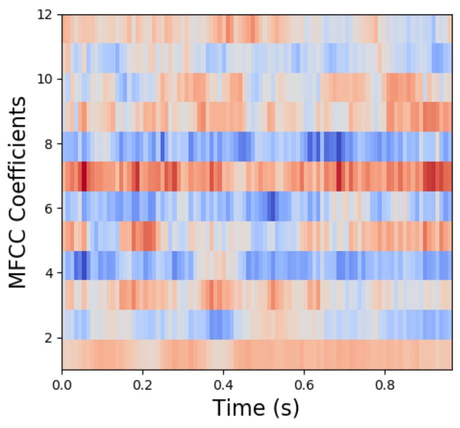


Figure 2.6: Mel-frequency Cepstral Coefficients

The MFCC spectrogram is xyz-graph that describes signals the change of the MFCC coefficients with the change of time. The Coefficients in the signal describes the rate of change in the spectral band and are represented with 12 Coefficients[13]. The z-axis describes the frequencies in the signal at time t.

2.3 Neural Networks

A neural network is a set of algorithms that are designed to recognize patterns. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated. In other words a neutral network the learn how to classify data of the same class based on similarities along the input data [14].

Figure 2.7 shows a simple three-layer neutral network. The architecture of normal network contains an input layer, a hidden layer and a output layer.

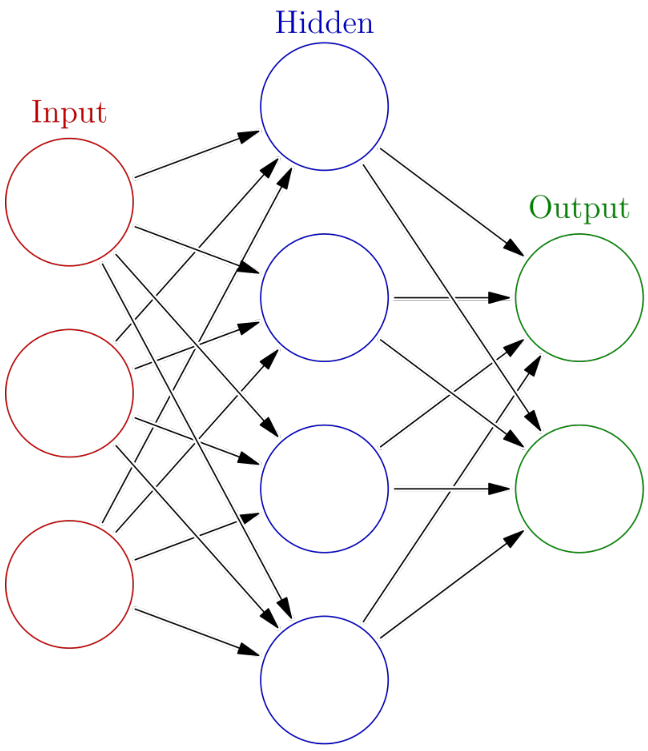


Figure 2.7: Neutral network. Available from: wikipedia.org/wiki/Artificial_neural_network[link] [accessed 14.05.2019]

The input layer is raw data that should be classified. In the speech recognition system, the input layer is the raw speech data presented as a vector. That means that the MFCC spectrograms must be converted in Nx1 vector before the neutral network can use it for classification.

The next layer is the hidden layer where the input layer x is multiplied with a weight function **w** and gives the output net. This layer extracts the implicit information of the features from the input layer. This layer can be computed with:

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} \tag{2.8}$$

The final layer is the output layer **y** or the prediction layer. This makes a prediction to which class the input layer belongs to and is computed with a activation function **f** that defines the output behavior of each node. The activation function allows the neural network to learn complicated, non-linear mappings between inputs and response variables [15].

$$y_j = f(net_j) \tag{2.9}$$

2.3.1 Activation functions

In a neutral network the activation function is defined an artificial neuron that delivers an output based on inputs. Some of the activation functions is **Relu** and **Softmax** [15].

The Rectified Linear Unit layer: The Rectified Linear Unit (ReLU) layer is a activation function. The activation function Rectified Linear Unit (ReLU) is computed with the function:

$$f(x) = max(0, x) \tag{2.10}$$

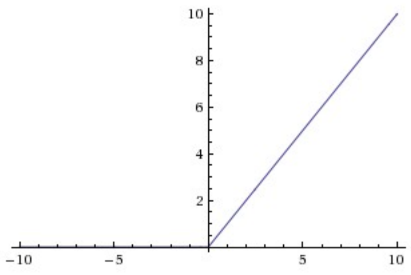


Figure 2.8: Rectified Linear Unit (ReLU) activation function. Available from: http://cs231n.github.io/neural-networks-1/ [accessed 13.05.2019]

Softmax: The softmax function is a activation function that takes a input of N real numbers and normalize it to a probability distribution of N variables. The output for this function is a number in the interval 0 and 1. The softmax function is useful as activation function for the output layer giving output as a likelihood for the layer belonging to that class. The softmax function is computed with the function:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}} i = 0, 1, 2..k \tag{2.11}$$

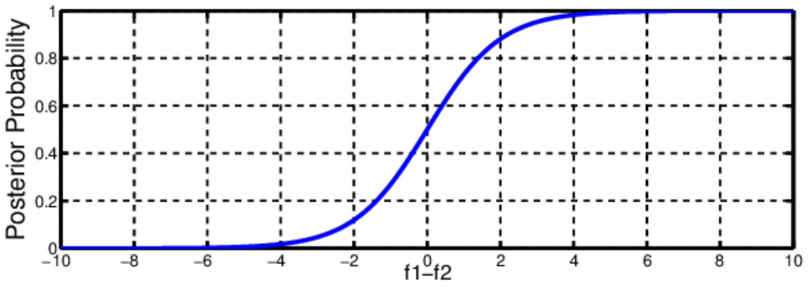


Figure 2.9: Noisy Softmax: Improving the Generalization Ability of DCNN via Postponing the Early Softmax Saturation - Scientific Figure on ResearchGate. Available from:https://www.researchgate.net/figure/Softmax-activation-function_fig2$_1$9121953 [accessed 13 May, 2019]

2.3.2 Training, validation and testing.

Now that the network is defined, the weights have to be set up to predict the desired output. This process called back-propagation will train the weights in the hidden layers to adapt the data that are to be predicted on [16]. For the speech recognition system this means that the algorithm learns the patterns in the MFCC data that belongs to each class. The training continuous until the error rate between the input data and the classification is minimized or until the training iterations ends.

The training error **J** is the difference between the desired output **t** and the predicted output **z**.

$$J(w) = \frac{1}{2} \sum_{k=1} c(t_k - z_k)^2 \tag{2.12}$$

The weights in the network is updated with the function:

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

η is the learning rate that indicates how much the weights are updated.

The new weights are defined with the function:

$$w(m+1) = w(m) + \Delta w(m) \tag{2.14}$$

The weights are updated for each iteration, learning more and more of the features in the MFCC spectrograms.

An alternative method then training the weights from scratch is to use **transfer learning**. Transfer learning is the process using the weight of a trained model and train it on a new set of data. This method is used in image classification and speech recognition system where there is there is not enough time or data to train a new model from scratch [17]. Since a natural network often requires months to train and a large amount of data, transfer learning can be useful. Using a pre-trained model for trained on similar dataset can be time and cost effective and can also improve accuracy of a previous trained model.

2.4 Convolutional neural network(CNN)

Convolutional neural network (CNN) is a variant of a standard neutral network. But instead of designing the hidden layers in the network structure as mentioned in the previous section, the convolutional neural network uses convolution and pooling layers [18].

An advantage of using a convolution neural network instead of a standard neutral network, it's the convolution layers ability to extract important patterns in the 2-dimensional MFCC spectrogram that can be used for classification.

2.4.1 Convolutional architecture:

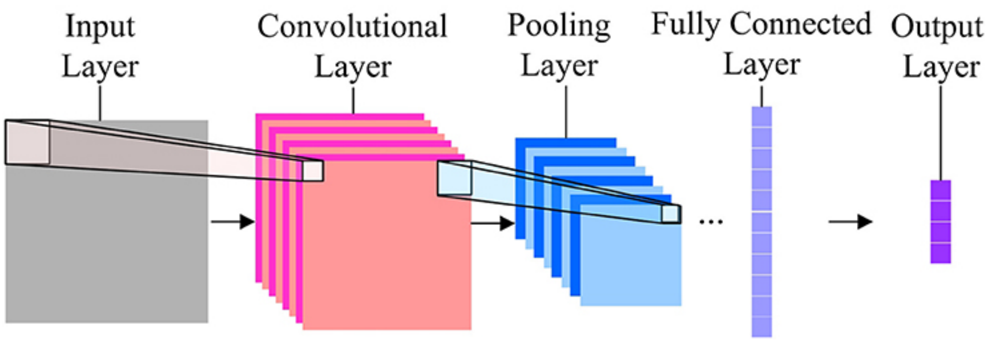


Figure 2.10: Convolutional neural network. Available from: https://www.frontiersin.org/articles/10.3389/fpsyg.2017.01745/full [accessed: 13.05.2019]

Input layer The first layer in a convolutional neural network is the input layer. The input layer is the data the convolutional neural network will use for classification. In speech recognition system this is the 2-dimensional MFCC spectrograms creates in the preprocessing stage.

Convolutional layer: In the convolutional layer each neuron takes an input from small window from the previous layer and preforms a dot product with a filter. The window is a small kernel filter of 2x2, 3x3, 4x4 that is convoluted across the entire input space to detect a specific kind of pattern in the input layer. The output of the convolution layer producing a 2-dimensional activation map of that filter. [19]

The speech recognition system the input layer is the MFCC spectrogram created in the pre-processing stage. The convolutional layer does a computation across the time-and frequency axis of the MFCC spectrogram. Given a MFCC spectrogram of size 20x32 and 2x2 spectrogram, the output from that layer is spectrogram of size 19x31. The layer consists of a set of learned filter or kernels. During a forward computation each kernel is convoluted across frequency axis of the MFCC spectrogram.

Pooling: After a convolution layer there is a pooling layer. The pooling layer takes the output from the previous convolution layer and down-samples it to produce a single output for that region. This feature reduces the computational complexities from the previous conversational layer and allows assumptions to be made for features in the local regions. [19]

The most common method for down sampling in a Convolutional neural network is maxpooling. Max-pooling means that a max filter is applied to non-overlapping subregions of the convolution filter and the output of each region is the max value for that region [20]. If for example a filter of size 2x2 is applied to 4x4 MFCC spectrogram , the output of the maxpooling is a 2x2 matrix with the highest value from the original spectrogram.

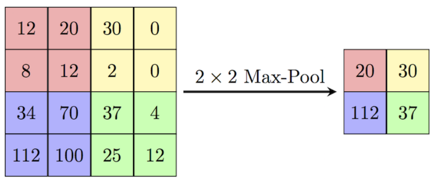


Figure 2.11: Max-pooling. Available from: https://computersciencewiki.org/index.php/Max-pooling_-_Pooling[accessed 13.05.2019]

Fully connected layer:

The fully connected layer or the flatten layer creates a Nx1 dimensional vector from the output from the previous layer. This is used for the final prediction of the classes where N is the number of classes. Each number in the vector present a probability that the corresponding input layer belongs to one of the classes. The class with the highest probability has the highest likelihood being the of the same class as the input speech signals.

In the speech recognition system this means that if the input signals is the sound of the word *Kompresjoner*, the output layer will predict the word *Kompresjoner* as text if the training and the testing of the convolution neural network is done properly.

2.5 Speech segmentation

Speech segmentation is the process of identifying boundaries between words, syllables and phonemes spoken i natural language [21].One method for involves separating speech from non-speech based on specific features in the original speech signal. This can be useful for separating keywords from continuous speech as there are usually non-speech or silence between when each word are spoken. Some of the features used for speech segmentation is extraction based on short-term root mean square(RSM) energy, zero-crossing or even Frequency-Domain Signal Features.

One method, short-term root mean square (RSM) energy, uses the measurement of energy to determine whether the signal have speech or non-speech. Energy is the measurement of how much signal there is at any time. Signal with high frequency have higher energy then signals with lower frequency[21]. Since silence or non-speech have lower energy then the speech signal, the measurement of energy can be used as a threshold to separate speech from non-speech.

Formula for the RSM:

$$E_n = \frac{1}{N} \sum_{m=-\infty}^{\infty} [x(n-m)w(m)]^2 \tag{2.15}$$

Amplitude to Decibel:

$$Gdb = 20log_{10}(A_2/A_1) \tag{2.16}$$

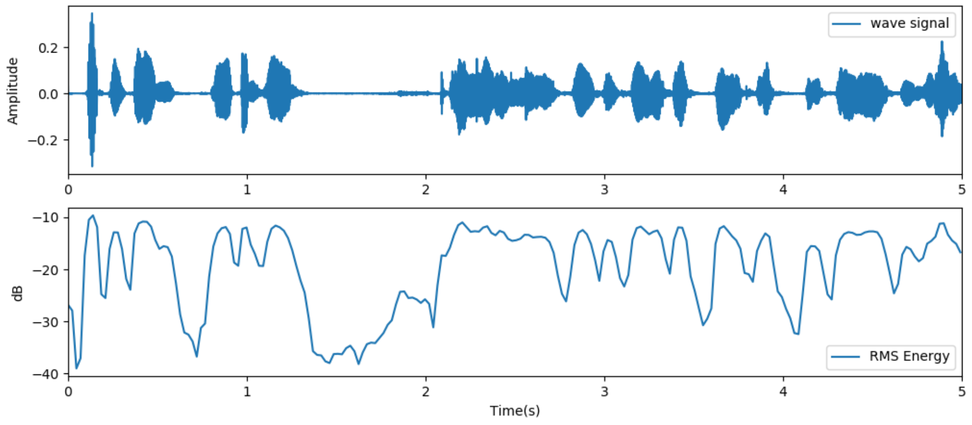


Figure 2.12: Wave and Short-Time Signal Energy plot of continuous speech signal

Figure 4.13 shows the wave-plot of the continuous signal and the RSM energy of the continuous signal. The first figure shows a continuous speech signal with both the speech and silence segments. The red lines indicate the start and the stop of each of the words and the corresponding transcription.

The second figure shows the RSM energy of the signal, plotted in dB. The RMS energy of the signal is at are highest when the amplitude of the original signal is at are highest. Since the amplitude of the unvoiced segments are lower than the speech segments, the corresponding energy is at is lower in these areas. A threshold that separates speech form non-speech can be set.

2.6 Speech recognition evaluation

When evaluating the performance of a speech recognition system it important to define specific terms: When the speech recognition system is classified correctly the speech recognition system have a **true positive**. However if the speech recognition system is classified incorrectly or not classified at all the system have **false positive** or an **false negative**. An overview over the terms:

- TP = True positive - the word is classified correctly.
- FP = False positive, when the word classified does not correspond to the reference word.
- FN = False negative, when a word is not detected. FN=D+I+S
- N- The is the number of words in the reference.

Word error rate(WER) is the standard measurement to assess the performance of an automatic speech recognition.[22] The word error rate measures the distance between the word predicted and the actual word and can be computed as:

$$WER = \frac{FN}{N} \tag{2.17}$$

False alarm rate is defined as the how often the speech recognition algorithm gave a false positive output compared with the how often that word was being said. The computation of this:

$$FalseAlarmrate = \frac{FP}{N} \tag{2.18}$$

Both WER rate and the false alarm rate has to be low in order to improve the accuracy of the speech recognition system.

2.7 Python

Python is a high level programming language with a thousands of libraries[23]. Some of the libraries that can be used for audio processing and speech recognition are listed below:

1. SpeechRecognition
2. librosa
3. keras
4. sklearn

5. tensorflow

SpeechRecognition is a python library for speech recognition. SpeechRecognition is a API that supports *CMU Sphinx*, *Google Cloud Speech API* and *Microsoft Bing Voice Recognition.*Sends a recuset for transcription.

librosa is a python package for music and audio analysis. [24]. Functions like *librosa.load*, *librosa.write* and *librosa.effect.split* can be used for respectively reading a audio-file, writing to an audio-file and splitting audio into segments.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation [25]. Keras can bes used to create a neutral network models and train them. Functions from *keras.layers* can be used to create models in LSTM, CNN and others.

sklearn is simple and efficient tools for data mining and data analysis. *sklearn.model_selection.train_test_split* can be used to split the dataset in train and split data.

tensorflow is an end-to-end open source platform for machine learning. *Tensorboard.*

Chapter 3

Data development

Two datasets ware developed during the course of this project with the purpose of training, validation and testing a speech recognition system.

3.1 Simulations with Emergency Communication Centre

The simulations with Emergency Communication Centre(ECC) was a project between the university, Laerdal Medical and ECC dispatchers. The project evolved over a couple of weeks and involved collecting audio recordings of conversations between a dispatcher and a caller during simulated emergency situations with the use of CPR.

3.1.1 Recording format

The communication between the caller and the dispatcher was set up with the TCPR-link app. This app would automatically call the number of a training partner defined in the app setting. During the simulations this number was set to a telephone number of an ECC dispatcher.

The audio was recorded on the caller side of conversation with the help of the telephone recorder app **BoldBeast recorder** installed on Sony Xperia Z5 [26]. **BoldBeast recorder** can record a live telephone conversation with clear sound from both side of the conversation with different sample rates and recording formats. The sample rate and the recording format was set to 16000 Hz (mono) channel and WAV format during the simulation.

3.1.2 Simulations setting

The recordings were set in closed, isolated rooms at both Laerdal Medical and University of Stavanger. On the floor of the recoding room a child sized CPR doll or an adult sized CPR doll was placed next to the telephone recording device. This telephone was covered with a black sheet that hid an activated TCPR-link app during the simulations. This was done to avid the caller looking at the TCPR-link app during the simulation. A overview of the simulation setting are described in figure 3.1.

On arrival the callers were asked to go into the simulation room. The call was started on behalf of the caller with TCPR-link communication. Simultaneously the recording of the audio was set up with **BoldBeast recorder**. When connection with the dispatcher was made, the caller was left in the room with the dispatcher on speaker. After 7 minutes the recording was stopped.

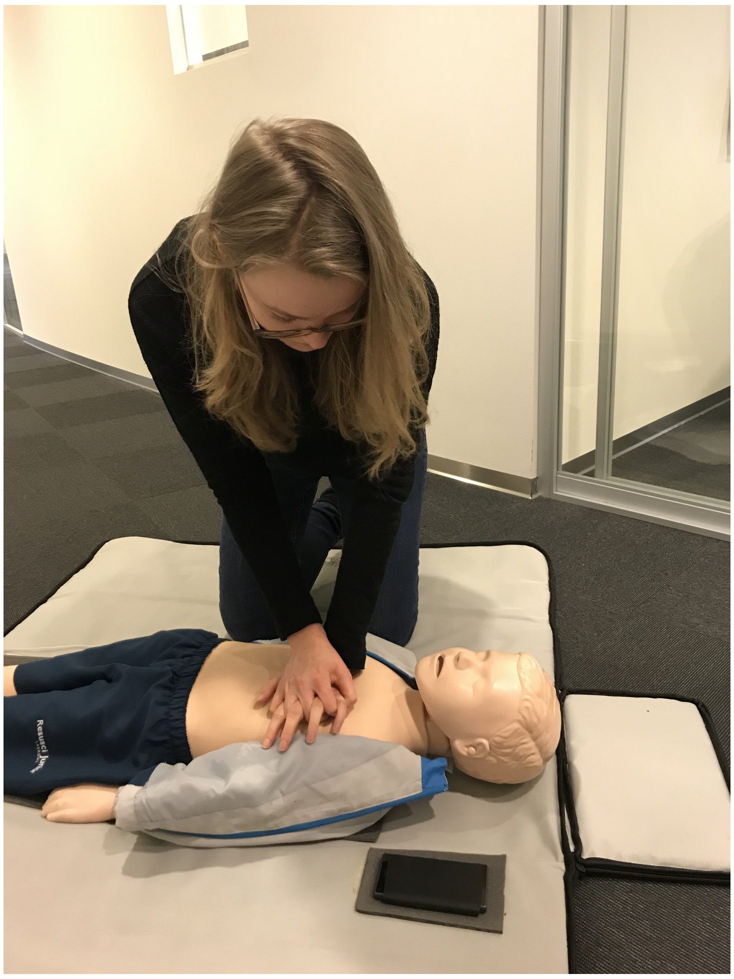


Figure 3.1: Simulation setting

3.1.3 Scenarios

Before each recording the volunteer callers was given a scenario:*heart-failure on adult with abnormal breath*, *Drowning of a 10 year old boy*, *Caller is health personnel*, *stressed caller*, *heart-failure on adult without breath*. An overview of these scenarios are described in table 3.1. These scenarios were based on real life emergency calls between a dispatcher and a caller.

During the simulations the dispatcher would ask questions about the location and the situation. The caller would respond with the location of the call, the situation and whether a patient was breathing. The dispatcher was not aware of the scenarios prior to the call and would work as they normally would in the situation.

Scenarios	First sentence	Location	Question about pulse
Heart-failure on adult with abnormal breath	*I need help!* NAME *is lying on the floor and his face is pale. I am unable to get contact with him. He has gotten a heart failure. His arms are moving. You have to send a ambulance*	At home	He breaths and makes snore sounds. Breathing stops after a while.
Drowning of a 10 year old boy	*My nephew has fallen in the lake and is drowning. He does not breath, he is dead. He has been in the lake for a while, I found him 50 meters from our dock*	Likaiveien 13, Hommersåk	He does not breath, he is lifeless.
Caller is health personnel(doctor, nurse)	*I have found a young man on the ground at the bus stop at Madlaveien*	Bus stop at Madlaveien	He does not breath.
Stressed caller	*I need Help! My father have gotten a heart attack! He fell off the kitchen-chair and is lying on the floor. Help! He is not responding, his face is pale. Help! You have to send an ambulance right away.*	At home	Hard to tell if he is breathing. He makes heavy breathing noises now and then. He makes sounds.
Heart-failure on adult without breath	*I need some help, there is someone that have fallen over. Help! There is someone that have fallen over with Stokkavannet, at Trafoststasjonen, with the trail*	At the trail with Stokkavannet, with Trafostasjonen	He struggles to breathe. You can hear coughing.

Table 3.1: Scenarios in simulations between the caller and dispatcher.

3.1.4 Participants

The simulations required volunteers that could pretend to be caller and someone be the dispatcher.

The dispatchers in the simulation was selected from real life ECC employees. These employers would mostly follow manuscript from *Norsk Index*, but they would also use different words and dialects during the simulations. An overview of the dispatcher in the simulation are described in table 3.2. There were 9 dispatchers from ECC that volunteered to participate in the simulations. 6 Male and 3 females. Each of the dispatchers had multiple calls.

Dispatcher	Gender	Simulations	Accent
1	Male	1,15,5,24	Rogaland
2	Male	2,7,17,27,28	Rogaland
3	Male	3,9,14,22,25	Hordaland
4	Male	4,6,11,23	Rogaland
5	Male	5,16,18,30	Østlandet
6	Female	8,13,19,33	Rogaland
7	Male	12,21,26,31	Rogaland
8	Female	29	Hordaland
9	Female	32	Rogaland

Table 3.2: The gender and the accent of the volunteer dispatchers and witch simulations the selected dispatcher was a part of.

The volunteer callers were recruited from both Laerdal Medical and UiS. Both student and employees were asked to participate. The age-group and the gender varied. About 33 callers was recruited. Of those there were 18 male and 15 females volunteered.

Callers had various degree of experiment in CPR. Employees of Laerdal Medical had more experiments, but student and UiS employees had less or little experience. There were no requirements knowing CPR, as the ECC employees would always guide caller in CPR on a patient suffering from heart-failure. Callers with little experiment was a advantage for the simulation as they would represent most people calling ECC. An overview of the callers are described in table 3.3.

When the recording was ended, the beginning and the end of each audio recording was removed. This was done to remove the calling sound and the sound of the stopping of the recordings. The final audio was saved as the with the label of the simulation, the scenario and the time and the date of the recording. The caller and the dispatcher renamed anonymous. An overview over the recoding file, the recoding setting, data, gender of caller and dispatcher, the duration and weather the caller knew CPR. in 3.3.

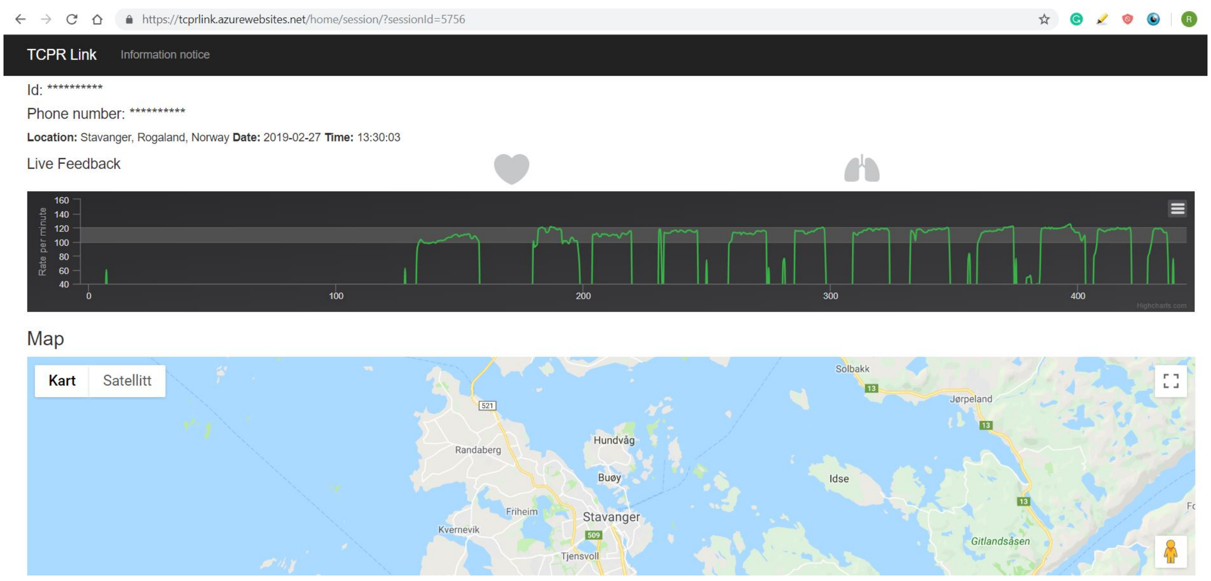
Simulation	Scenario:	Date:	Time:	Dispatcher:	Caller:	Location	CPR?:	Duration:
1	4	25.02.2019	13:30	Male	Female	Laerdal	Yes	07:12
2	2	25.02.2019	13:50	Male	Male	Laerdal	Yes	06:37
3	2	25.02.2019	14:10	Male	Female	Laerdal	Yes	07:21
4	5	25.02.2019	14:30	Male	Female	Laerdal	Yes	07:20
5	5	25.02.2019	14:50	Male	Female	Laerdal	partial	07:15
6	2	27.02.2019	13:30	Male	Female	Laerdal	partial	07:00
7	1	27.02.2019	13:50	Male	Female	Laerdal	Yes	07:10
8	5	27.02.2019	14:10	Female	Male	Laerdal	Yes	07:01
9	1	27.02.2019	14:36	Male	Female	Laerdal	Yes	07:12
10	5	27.02.2019	14:52	Male	Female	Leardal	Yes	07:06
11	1	04.03.2019	13:31	Male	Female	UiS	No	07:36
12	3	04.03.2019	13:50	Male	Female	UiS	No	06:56
13	2	04.03.2019	14:10	Female	Male	UiS	partial	07:04
14	1	04.03.2019	14:30	Male	Male	UiS	partial	07:00
15	3	04.03.2019	14:50	Male	Female	UiS	No	07:05
16	1	04.03.2019	15:10	Male	Male	UiS	Yes	07:08
17	5	05.03.2019	13:30	Male	Male	UiS	Yes	07:04
18	2	05.03.2019	13:50	Male	Male	UiS	partial	06:59
19	3	05.03.2019	14:10	Female	Male	UiS	Yes	07:08
20	5	05.03.2019	14:30	Male	Female	UiS	partial	07:02
21	2	05.03.2019	14:50	Male	Male	UiS	yes	07:05
22	5	05.03.2019	15:10	Male	Female	UiS	partial	07:00
23	4	06.03.2019	13:30	Male	Male	UiS	yes	07:00
24	1	06.03.2019	13:50	Male	Male	UiS	yes	07:01
25	4	06.03.2019	14:10	Male	Male	UiS	no	07:04
26	1	06.03.2019	14:30	Male	Male	UiS	no	07:03
27	3	06.03.2019	14:50	Male	Male	UiS	partial	06:58
28	3	15.03.2019	13:30	Male	Female	Laerdal	yes	07:02
29	4	15.03.2019	13:50	Female	Male	Laerdal	yes	06:54
30	3	15.03.2019	14:10	Male	Male	Laerdal	yes	07:04
31	4	15.03.2019	14:30	Male	Female	Laerdal	yes	07:06
32	3	15.03.2019	14:50	Female	Female	Laerdal	yes	07:06
33	4	15.03.2019	15:10	Female	Male	Laerdal	yes	07:02

Table 3.3: The name of the recoding file, the recoding setting, data, gender of caller and dispatcher, the duration and weather the caller knew CPR.

3.1.5 TCPR-link data

In addition to the recordings the results from TCPR-link was also saved. Since the caller was communicating with the dispatcher trough TCPR-link, the compression rate could also be viewed from TCPR-link during and after the conversation.

The result from the use of TCPR-link in simulation 1 are shown in figure 3.2. The green line in the graph were the compression rate during the simulation. When the compression rate was within the white the line, the compression rate was correctly. The downtime was when the caller did two rescue breaths.


https://tcprlink.azurewebsites.net/home/session/?sessionId=5756

TCPR Link Information notice

Id: **********

Phone number: **********

Location: Stavanger, Rogaland, Norway **Date:** 2019-02-27 **Time:** 13:30:03

Live Feedback

Map

Figure 3.2: TCPR link data from simulation 1. The figures show both the callers compression rate and the location of the caller.

3.2 Single utterances

Along with these simulations a different set of training data was collected from single speakers. The speakers were asked to record themselves saying single Norwegian words or utterances from a piece of paper. All the recordings was done with the app *Voice recorder* [27] and each recording was recorded in wav format with a 16000 Hz sampling-rate.

The recordings were done by 13 females and 30 males, some of the participants were asked to read the words 1 to 3 times. The result from these recording was a dataset of 1054 recordings with 10 different words: *kompresjoner*, *innblåsninger*, *munn*, *falt(datt)*,*tell*, *dypt*, *hardt*,*gjenoppliving*, *trykk*. The result from these recordings can be seen in table 3.4.

word	recordings
falt	91
hardt	93
dypt	91
tell	121
kompresjoner	110
innblåsninger	99
munn	80
puste	157
trykk	116
gjenoppliving	96

Table 3.4: small 0.2-1 seconds 16000 Hz audio files with 10 keywords. The data can be downloaded from **link**.

Chapter 4

Method

4.1 Proposed method

The proposed method was to find 10 keywords: *dypt,falt*, *Kompresjoner*, *gjenoppliving*, *hardt*, *innblåsinger*, *puste*, *tell* and *Trykk*, in a audio file. The methods for finding these keyword in both small 1 second signals and continuous speech are describe here. The sections and experiments are divided in:

- **Transcription**: Transcription of the simulation data.
- **Experiment 1**: CNN model for word classification
- **Experiment 2**: Speech recognition - identification of keywords using CNN model.
- **Experiment 3**: Speech recognition - identification of keywords using Google's speech recognition model.

Al the experiments was done in **python 3.6** using the libraries **keras 2.2.4**, **tensorflow 1.13.1**,**librosa 0.6.3** and *sklearn 0.21.2*. The python codes used in the experiments are linked in the zip file **main_project**.

4.2 Transcriptions

All the simulation data with ECC was transcribed with the help of **oTranscribo** [28] and **Amberscript** [8]. The audio files were transcribed in sentence and showed both the how spoke during the conversation, when (s)he spoken and what s(he) spoke. The results from the transcription were to be compared with the output from experiment 2 and 3.

4.3 CNN model for word classification

In experiment 1 a simple convolutional neural network model was created and trained and tested using the data from simulations and the speech utterance. An overview of the implementation method can be seen in figure 4.1.

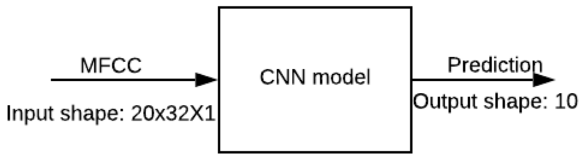


Figure 4.1: Overview of Experiment 1

4.3.1 Preparing the Norwegian data

The training and validation data for the model had to be prepared. This was done by manually cutting the keywords out of the audio files in table 4.1 with the audio editor and recorder Audacity [29]. These keywords was cut out by listening to where in audio file the keyword started and ended, and then save this segment as a new audio file. The final extracted words was between 0.2-1 second long and was cut from the start of the word to the end, as described in figure 4.2.

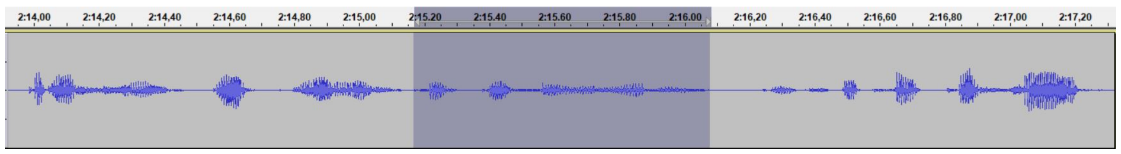


Figure 4.2: Cutting the audio into words

These keywords were then placed in directory marked with the name of the class they belong to. For example, the audio file with the keyword *kompresjoner* was placed in the directory *kompresjoner*. An overview of the datasets used for the training and the validation of the simulations are described in 4.1.

Training and validation data				
Speech utterance	Simulation 29	Simulation 22	Simulation 19	Simulation 2
Simulation 2	Simulation 1	NST acoustic speech database for Norwegian [1]		

Table 4.1: Training and validation data

[1]Speech utterances *falt* and *tell* collected from NST acoustic speech database for Norwegian. This dataset is a part of *Nasjonalbiblotekets* resource directory

4.3.2 Labelling the data

.

The dataset was first labelled with a number between 0 and 9. The result of this are shown in table 4.2, where each of the data in on class was given a label in alphabetical order.

word	"dypt"	"falt"	"gjenoppliving"	"hardt"	"innblåsning"	"kompresjoner"	"munn"	"puste"	"tell"	"trykk"
Number of samples	120	119	101	135	104	110	80	158	121	116
Label encoding	0	1	2	3	4	5	6	7	8	9

Table 4.2: The label of the keywords

After this was done labels was given a one-hot encoding label. The transformation from a label to a one-hot was done with the python library tensorflow.keras.utils.to_categorical, which automatically converted the matrix of labels encodings into a matrix of one-hot encodings. The result for this transformation are described in 4.3. Each of the of keyword are described with 10 bits with one high bit and all others low. This meant that label 0. *dypt* was described with " 1 0 0 0 0 0 0 0 0 0".

label	One-hot encoding									
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1

Table 4.3: One-hot encoding for each of the keywords

The features from the audio file was then extracted using a MFCC. This computed in python using the librosa library. The *librosa.feature.mfcc* would automatically calculate the Mel-frequency cepstral coefficients(MFCC) of the input signal and returned the MFCCs time series of shape=(n_mfcc, t). Were n_mfcc was set to 20 by default and t varied with the time length of the input signal as shown in figure 4.3

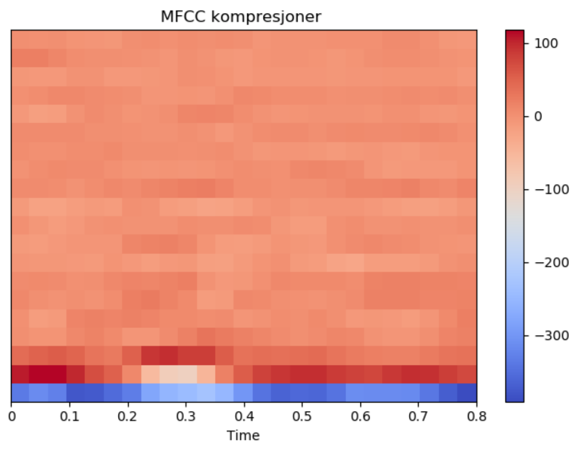


Figure 4.3: MFCC spectrogram of speech utterance.

The input data for the CNN model had the same shape, meaning that the duration of audio file had to be of the same length. A max length was set to increase or reduce the size of the input data to fit the same length. This length was set to 1 second, which was the same t=32 for a MFCC spectrogram sampled with 16000 Hz. The data that was shorten then t=32, was zero-padded to this length while the data longer then t=1 was cut into 1 second. An example of a zero-padding can be seen in figure 4.4, where the size of MFCC spectrogram in figure 4.3 was increased to (20,32).

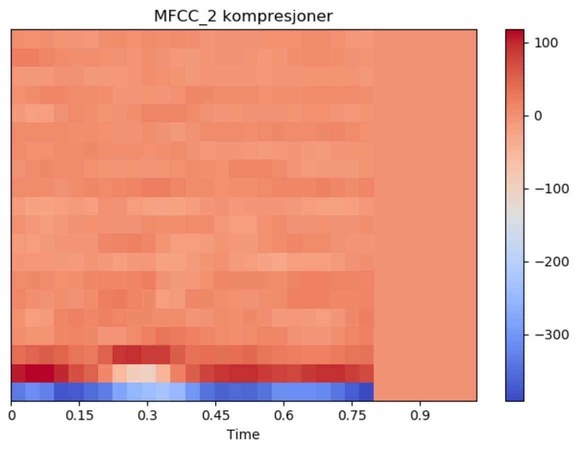


Figure 4.4: MFCC spectrogram of $t < 32$ was zero-padded to t=32

The implementation of the set size MFCC, was created with the function *wav2mfcc* 4.5 in python. This function converts the input signal from the file_path to MFCC of shape=[20,32]. The results were saved and saves as an array.

```
def wav2mfcc(file_path, max_len=32):
    wave, sr = librosa.load(file_path, mono=True, sr=None)
    mfccs = librosa.feature.mfcc(wave, sr=sr)
    mfcc=zero_padding(mfccs, max_len)
    return mfcc
```

Figure 4.5: Python code used to create MFCC spectrogram of shape (20,32)

4.3.3 CNN model

The CNN model was designed using the Tensorflow implementation of the keras specification, *tensorflow.keras*. An overview of the figure and the layers in the model are described in figure 4.6. The input layer was first convoluted with 3 convolutional layers of kernel size 2x2 and filter size 32,48 and 120 respectively. The output from the convolutional layer was sent through a Maxpooling layer of pooling size 2x2, before the output was flatten by a flatten layer. The output from this layer went through 3 dense layers, were Dense_1 classified the output.

Layer	Name	Type	Input	output shape	Filter	kernel size	pool size	Dropout rate	activation function
1	Conv2D_1	Convolution2D	(None,Input_shape)	(None,19, 31, 32)	32	2x2			relu
2	Conv2D_2	Convolution2D	(None,19, 31, 32)	(None, 18, 30, 48)	48	2x2			relu
3	Conv2D_3	Convolution2D	(None, 18, 30, 48)	(None, 17, 29, 120)	120	2x2			relu
4	MaxPol_1	MaxPooling2D	(None, 17, 29, 120)	(None, 8, 14, 120)			2x2		
5	Dropout_1	Dropout	(None, 8, 14, 120)	(None, 8, 14, 120)				0.25	
6	Flatten_1	Flatten	(None, 8, 14, 120)	(None, 13440)					
7	Dense_1	Dense	(None, 13440)	(None, 128)	128				relu
8	Dropout_2	Dropout	(None, 128)	(None, 128)				0.25	
9	Dense_1	Dense	(None, 128)	(None, 64)	64				relu
10	Dropout_3	Dropout	(None, 64)	(None, 64)				0.25	
11	Dense_1	Dense	(None, 64)	(None, output_dim)					softmax

Table 4.4: Layers in Model

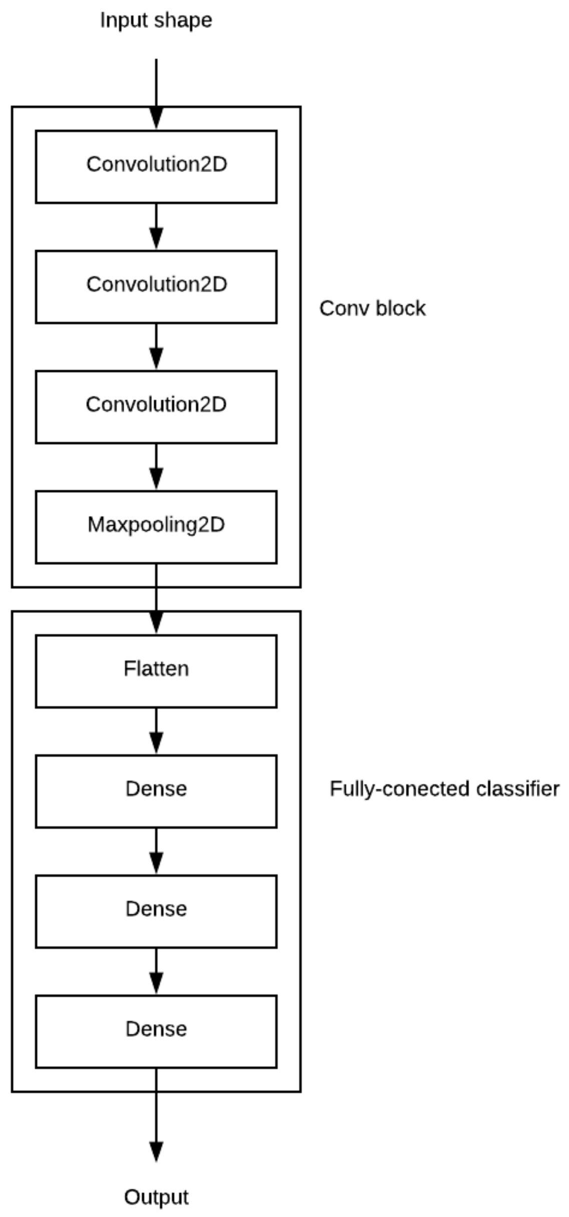


Figure 4.6: Layers in the model

The implementation of the model in python are described in figure 4.9.

```
def model(input_dim,output_dim):
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(2, 2), activation='relu', data_format='
    channels_last', input_shape=(input_dim[0], input_dim[1], input_dim[2])))
    model.add(Conv2D(48, kernel_size=(2, 2), activation='relu'))
    model.add(Conv2D(120, kernel_size=(2, 2), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(output_dim, activation='softmax'))
    model.compile(loss='categorical_crossentropy',
                  optimizer='Adadelta',
                  metrics=['accuracy'])
    return model
```

Figure 4.8: CNN model created in python using Keras

4.3.4 Preparing the Training and Validation data

Before the training of the CNN model could start, the validation and training data had to be prepared. The model was trained in two epochs: first with a collection of 15 different 1 second English words collected from *Speech commands: A dataset for limited-vocabulary speech recognition* [30].Then the model was retrained with 10 Norwegian keywords. Validation and training data for both datasets had to be prepared.

This was done with the python function *sklearn.model_selection.train_test_split* witch automatically split prepared the MFCC into training and validation data. Where 60 % of the data was split into training data and 40 % was split into validation data with a random rate of 42. The training and the validation for both the English data and the Norwegian data are described in figure 4.5.

4.3.5 Training and Validation

Finally the model was trained first with the English model with the parameters described in 4.6. The epoch and the batch size were set to 50. The loss, The optimizer and the Metrics were set to

Language	Training data	Validation data
English	21658 samples	14440 samples
Norwegian	762 samples	validate on 508 samples

Table 4.5: Training an validation data for the English data

categorical crossentropy. *Adaelta* and *accuracy* respectively. Finally the input shape was in set to [20,32,channel], while output shape was set to 15 for the English model and 10 for The Norwegian Model.

Batch size	50
Epoch size	50
Loss	'categorical_crossentropy'
Optimizer	'Adadelta'
Metrics	'accuracy'
Input_shape	[20,32,channel], channel=1
Output_shape	15(EN), 10(NO) [1]

Table 4.7: Parameters to train the model

Once the model was trained on the English data. The model was retrained on the Norwegian data. This was done by freezing the first 6 layers of the model and removing the 7-11 last layers from the previous model. This meant that weights from the first 6 layers could not be retrained and the weights in the last layers in model, used from classifying the English dataset, was removed. The model could then be trained and classified on Norwegian data.

This was done by adding 6 new layers to the frozen English model. These layers were the same as the layers from original model in 4.4. However, these layers were not trained on English data and the output shape of the last dense model was set to 10. The implementation of this model in python and simple figure of the retrained model can be seen in figures 4.9 and 4.11.

[1] EN-English, NO-Norwegian

Language	Total parameters	Trainable parameters	Non-trainable parameters
Norwegian	1,650,891	1,621,379	29,512

Table 4.8: Trainable parameters in the Norwegian model

```
# Freeze layers in English model.
english_model.summary()
for layer in english_model.layers:
  layer.trainable=False

# Create a new model where the last 5 layers are removed
new2_model=Model(english_model.inputs, english_model.layers[-7].output)
new2_model.summary()

# Add new layers to the new model
def get_model():
 model=Sequential()
 model.add(new2_model)
 model.add(Dense(120, activation='relu', name='Dense_1'))
 model.add(Dropout(0.25, name='Dropout_2'))
 model.add(Dense(64, activation='relu',name='Dense_2'))
 model.add(Dropout(0.4,name='Dropout_3'))
 model.add(Dense(num_classes, activation='softmax',name='Dense_3'))
 model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
 return model
```

Figure 4.10: Model created in python using Keras

The trainable and non-trainable parameters in the model are described on 4.8. The model had 1,650,891 parameters were 1,621,379 of them were trainable.

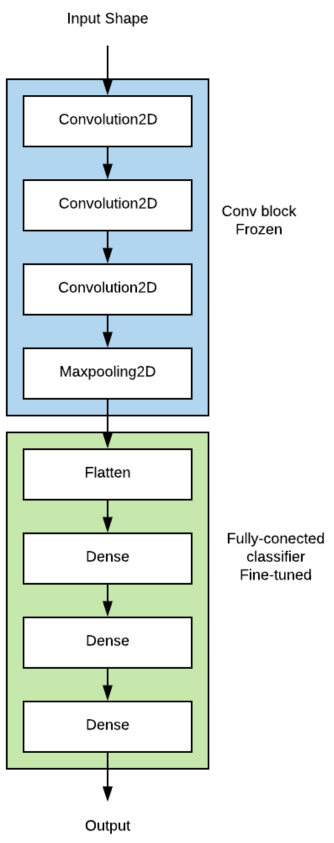


Figure 4.11: Overview of frozen and trainable layers in the model retrained on Norwegian data.

4.3.6 Prediction

Once the model had been trained on English data and retrained on Norwegian data, the model could be tested on speech utterances. This was done by sending an input data through the CNN model and getting an output as the 9-bit probability of the input data belong to this class. The output is then compared with the one-hot encoding labels.

In figure 4.9 the output from CNN model is compared with one-hot encoding labels. In the figure column number 6 have the highest probability. This means that the output has the highest likelihood of being of compatible with label 5 which is the keyword *kompresjoner*.

Output	0.033	0.055	0.31	0.21	0.023	0.9999	0.1	0.03	0.2	0.33

$\updownarrow$

label	One-hot encoding									
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1

Table 4.9: The output from the prediction is compared with the encoding labels. In this example the output is classified as label 5 which is compatible with the word ”kompresjoner”.

4.4 Speech recognition - identification of keywords using CNN model

In experiment 2 the model created in experiment 1 was used to find the keywords in the data from the simulations between the caller and the dispatcher. The simulation data were used as test data for the model.

But before the audio could be classified the keywords from the simulation file have to be extracted. This meant that a audio file of the same length as the input data for CNN model had to be created.

4.12 described the method of implementation. A speech segmentation algorithm would separate the audio file into words. This speech signal was then converted to MFCC spectrogram and reshaped to fit the input data of the model. And finally, the input data was classified using the model and methods from experiment 1.

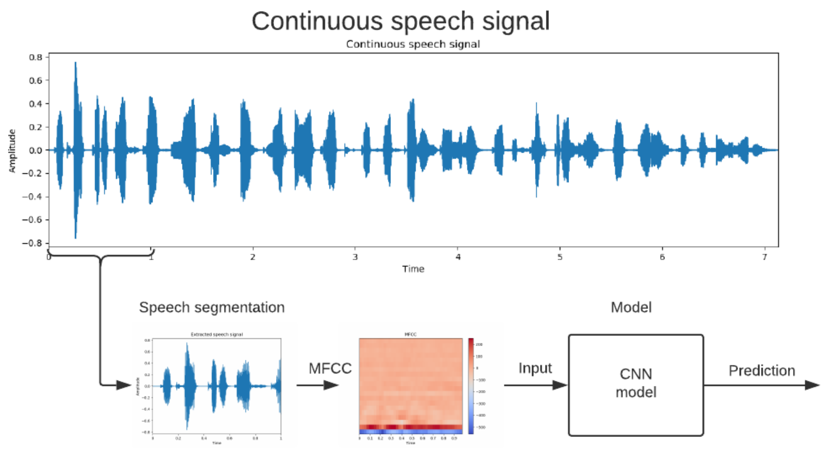


Figure 4.12: Speech recognition system

4.4.1 Speech segmentation

A speech segmentation algorithm would first calculate the short-term root mean square Energy of a window with frame length=500 and hop length=200. The result from this calculation is seen in figure as seen in figure 4.13. Then a threshold would determine whether the selected window were speech or silence. Since there were silence or non-speech between the end of a word to the start of a new word, the best method for extraction seemed to separate the audio in speech and non-speech segments. The threshold, set to 50 dB based on prior experiments, would determine a silence window as be a window with energy less then 50 dB. Samples above this threshold was considered as speech. The windows that was considered speech was saved in a separate folder.

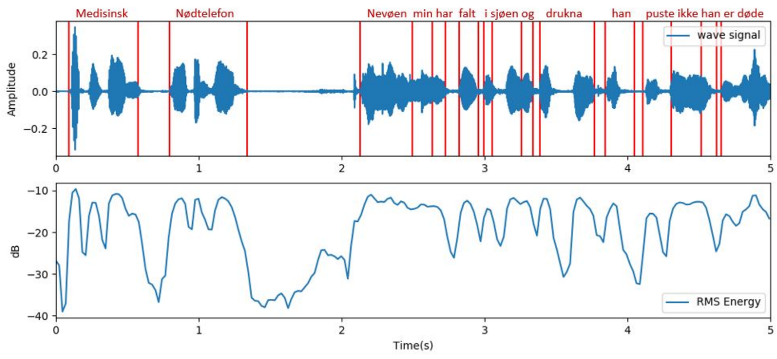


Figure 4.13: RMS energy of continuous speech signal and the start stop points of a speech segment.

In python the function *librosa.effects.split* was used to detect the speech segments in the simulations.

The function gave the speech segments output in intervals, intervals[i] == (start_i, end_i). These points were used to split the audio and save them in a directory. A fragment of this code is described in figure 4.14.

```
# Split audio in 1 second time sections.
    y, sr = librosa.load(filepath)
    intervals = librosa.effects.split(y, top_db=50, frame_length=500, hop_length=200)

    for i, (start_i, stop_i) in enumerate(intervals):
        audio = y[start_i:stop_i]
        filename = 'Chuncks/chunck-{}.wav'.format(i)
        librosa.output.write_wav(filename, audio, sr)
```

Figure 4.14: Cut the audio-file into smaller segments using python.

These speech segments was then converted to MFCC with the same method described in 4.3Experiment 1 and used as input to the same CNN model created in the same experiment.

4.4.2 Prediction

The output from the model gave a prediction to weather that class belong to that label. However to avoid the speech segments containing utterances from a non-class to be predicted on a threshold was set to 0.9999 to makes sure a speech utterance could not be classified without having a high likelihood of belong to one of the classes. If one of the columns had a higher likelihood then 0.9999 of belonging to a class, the speech segment was classified.

The result from predicted output was printed out as a text file and with time and spoken word. This was compared with the real transcription to find the word error rate and the false alarm rate.

4.5 Speech recognition - identification of keywords using Google's speech recognition model

In this experiment python was connected with the Google Web speech API with the help of the python library *SpeechRecognition.* The language was set up to Norwegian. A simple representation of the set-up is described in figure 4.15.

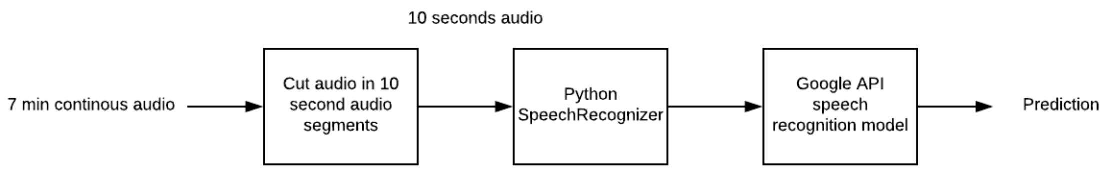

10 seconds audio
7 min continous audio
Cut audio in 10 second audio segments
Python SpeechRecognizer
Google API speech recognition model
Prediction

Figure 4.15: Connection with Google Web Speech API

The test data was first cut in 10 second segments as the input data could not be longer than 30 second and less than 10 seconds. This audio files were saved in a separate folder and removed once the prediction was over.

Once this is done all the 10 second test data is recorded in to *Audiodata*, the files were transcribed with *recognize_google*. After a few second the transcribing was done, and the output was printed out as text. This was compared with the real transcription to find the word error rate and the false alarm rate.

```
import speech_recognition as sr


    all_text = []
    r = sr.Recognizer()
    for audio_file in sorted(glob.glob("Chuncks/*.wav"),key=numericalSort):
          harvard = sr.AudioFile(audio_file)
          with harvard as source:
               audio = r.record(source)
          # Determine which type of API to use to determine what was spoken.
          text = r.recognize_google(audio, language='no-NO')
          all_text.append(text)

    transcript = ""
    print(transcript)

```

Figure 4.16: SpeechRecognizer

Chapter 5

Experiments and results

5.1 Transcriptions

All of 33 simulation with ECC was transcribed. These transcriptions gave an accurate representation of what was spoken in the simulation. The transcriptions from simulation 2, simulation 5 and simulation 11 are linked in the appendix **A.1**. The rest of the simulations remained anonymous due to privacy policy.

An extract of simulation are shown in figure 5.1 and figure 5.2.There were 8 keywords detected in the extract. These keywords was marked with red boxes. In the first extract the keyword *puste* was spoken 4 time by both the dispatcher and caller. While in the second extract the keywords *trykk*, *telle* and *trykker* and *hardt* was spoken by the dispatcher.

00:29 Dispatcher: Hvis du rister godt i han. Svarer han da?
00:33 Caller: Hallo. Nei, Nei. Det er ikke noe tegn til liv.
00:37 Dispatcher: Ikke noe tegn til liv. Ser du om han $\boxed{\text{puster}}$ normalt?
00:39 Caller: Ehh. Nei, det virker ikke som han. Nei, han strever med å $\boxed{\text{puste}}$. Du hører han hoste og harke liksom.
00:47 Dispatcher: Åja, okei. Ligger han på ryggen eller?
00:49 Caller: Ja, han ligger på ryggen.
00:51 Dispatcher: ja, okei. Hvis du bøyer hodet hans bakover, $\boxed{\text{puster}}$ han bedre da syns du?
00:59 Caller: ehh. Det er litt mindre hosting, men det er litt sånn rar $\boxed{\text{pusting}}$. Jeg vet ikke.

Figure 5.1: Simulation 5 test 1

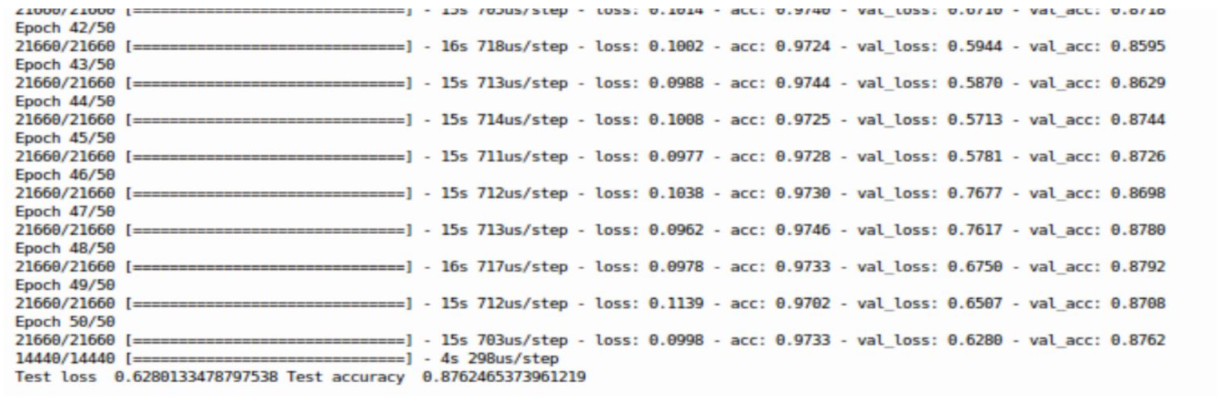
01:39 Dispatcher: og bruk strake armer og så $\boxed{\text{trykk}}$ brystkassen ned 5cm i takt på 100 i minuttet. Jeg skal hjelpe deg å $\boxed{\text{telle}}$.
01:47 Caller: Ja.
01:48 Dispatcher: Så $\boxed{\text{trykker}}$ vi $\boxed{\text{hardt}}$ med strake armer i denne takten.

Figure 5.2: Simulation 5 test 1

5.2 Experiment 1

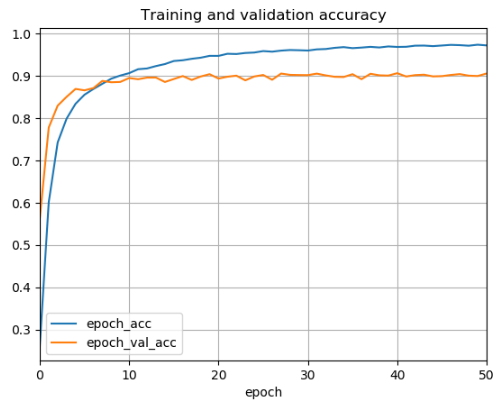
5.2.1 Training on English data

The model was first trained on the single utterance from the English dataset. Figure 5.3, 5.4a and 5.4b were the result from the training and validation of the mode.. The accuracy of both training and the validation data increased rapidly stabilizing at about 15 epochs. The training loss and validation loss decreased until they stabilized after 40 epochs. After 50 epochs the training loss was 0.0998, the training accuracy was 0.9733, the validation loss was 0.6280 and the validation accuracy was 0.8762.

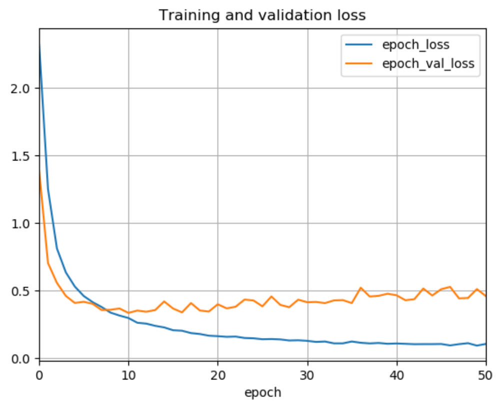


```
Epoch 42/50
21660/21660 [==============================] - 16s 718us/step - loss: 0.1002 - acc: 0.9724 - val_loss: 0.5944 - val_acc: 0.8595
Epoch 43/50
21660/21660 [==============================] - 15s 713us/step - loss: 0.0988 - acc: 0.9744 - val_loss: 0.5870 - val_acc: 0.8629
Epoch 44/50
21660/21660 [==============================] - 15s 714us/step - loss: 0.1008 - acc: 0.9725 - val_loss: 0.5713 - val_acc: 0.8744
Epoch 45/50
21660/21660 [==============================] - 15s 711us/step - loss: 0.0977 - acc: 0.9728 - val_loss: 0.5781 - val_acc: 0.8726
Epoch 46/50
21660/21660 [==============================] - 15s 712us/step - loss: 0.1038 - acc: 0.9730 - val_loss: 0.7677 - val_acc: 0.8698
Epoch 47/50
21660/21660 [==============================] - 15s 713us/step - loss: 0.0962 - acc: 0.9746 - val_loss: 0.7617 - val_acc: 0.8780
Epoch 48/50
21660/21660 [==============================] - 16s 717us/step - loss: 0.0978 - acc: 0.9733 - val_loss: 0.6750 - val_acc: 0.8792
Epoch 49/50
21660/21660 [==============================] - 15s 712us/step - loss: 0.1139 - acc: 0.9702 - val_loss: 0.6507 - val_acc: 0.8708
Epoch 50/50
21660/21660 [==============================] - 15s 703us/step - loss: 0.0998 - acc: 0.9733 - val_loss: 0.6280 - val_acc: 0.8762
14440/14440 [==============================] - 4s 298us/step
Test loss  0.6280133478797538 Test accuracy  0.8762465373961219
```

Figure 5.3: Training English model



(a) Training and Validation accuracy

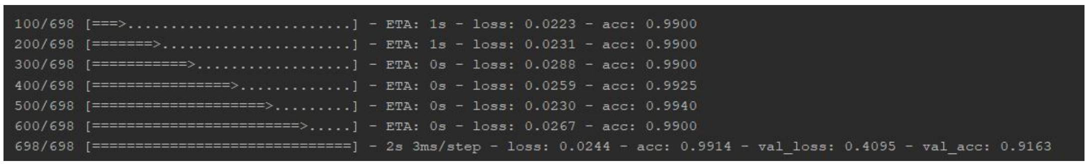


(b) Training and Validation loss

Figure 5.4: Result from training and validating the CNN model on the English data

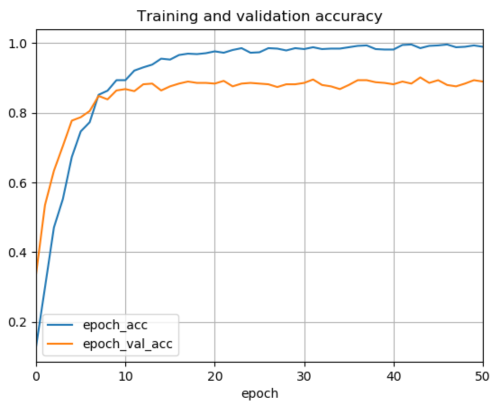
5.2.2 Training on Norwegian

The model was then retrain on the speech utterance from dataset. Figure 5.5, 5.6a and 5.6b were the result from the retraining and revalidation of the model. The training and validation accuracy increase continuously until about 10 epochs. The training and validation loss decreased until about 20 epochs. After 50 epochs the training loss was to 0.0244, the training accuracy was 0.9914, the validation loss was 0.4095 and the validation accuracy was 0.9163.

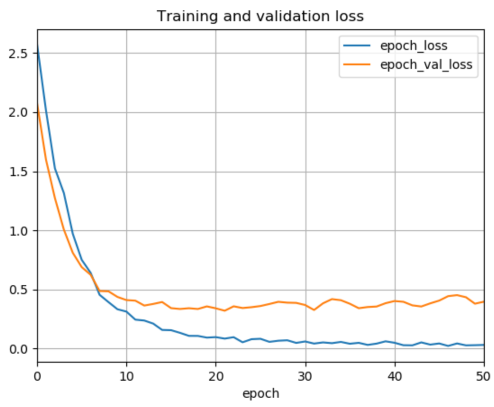


```
100/698 [===>..........................] - ETA: 1s - loss: 0.0223 - acc: 0.9900
200/698 [=======>......................] - ETA: 1s - loss: 0.0231 - acc: 0.9900
300/698 [===========>..................] - ETA: 0s - loss: 0.0288 - acc: 0.9900
400/698 [================>.............] - ETA: 0s - loss: 0.0259 - acc: 0.9925
500/698 [====================>.........] - ETA: 0s - loss: 0.0230 - acc: 0.9940
600/698 [========================>.....] - ETA: 0s - loss: 0.0267 - acc: 0.9900
698/698 [==============================] - 2s 3ms/step - loss: 0.0244 - acc: 0.9914 - val_loss: 0.4095 - val_acc: 0.9163
```

Figure 5.5: Training Norwegian model



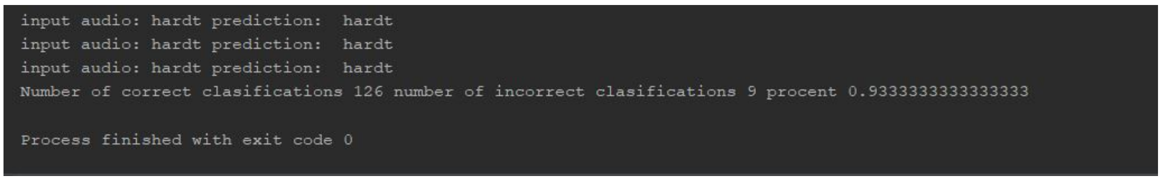
(a) Training and test accuracy



(b) Training and validation loss

5.2.3 Testing the model on speech utterance

The model was then tested on data from the speech utterance. The result can be seen in 5.7. The model was able to classify the word *hardt* with 0.933 percent accuracy. The Word error rate was 0.0667 and the false alarm rate was 0.



```
input audio: hardt prediction:  hardt
input audio: hardt prediction:  hardt
input audio: hardt prediction:  hardt
Number of correct clasifications 126 number of incorrect clasifications 9 procent 0.9333333333333333

Process finished with exit code 0
```

Figure 5.7: Testing model on 1 second utterances.

5.3 Experiment 2

Figure 5.8 and 5.9 were the results from testing the CNN model on test data from simulation 5 with ECC. The output from the prediction gave both the prediction of the word and when the word was spoken. The red box around the prediction indicated that the prediction was classified correctly.

```
00:37 prediction tell
00:47 prediction tell
00:50 prediction gjenoppliving
00:51 prediction tell
00:54 prediction tell
00:58 prediction falt
01:02 prediction puste
```

Figure 5.8: Results from speech recognition using trained CNN model. The output from the prediction showed the time and the prediction.

```
01:32 prediction falt
01:32 prediction tell
01:33 prediction kompresjoner
01:36 prediction puste
01:36 prediction tell
01:38 prediction tell
01:41 prediction falt
01:46 prediction tell
01:46 prediction tell
01:51 prediction tell
01:52 prediction gjenoppliving
01:54 prediction tell
```

Figure 5.9: Simulation 5 with experiment 2

As seen in the figures, the model was only able to predict the keyword *puste* at time 01:02 and *tell* at time 01:46 in the simulation. This mean that the only 2 of the 8 classes in the original transcription 5.1 and 5.2.

All the classification errors for the keywords between the real simulation 5 transcription and the entire output from the model are described in table 5.1. The column *Number of words* in the table 5.1 was the number of times the word of a specific class was spoken during the real transcription. This was compared with the number of *True positives*, *False positives* and *False negatives* in output from the prediction. The **WER** was the rate of wrong classifications in the prediction.

Simulation 5	CNN model				
Utterance	Number of words	True positive	False positive	False negative	WER
puste	4	0	3	4	1
tell	5	4	39	1	0,2
dypt	0	0	0	0	0
hardt	1	0	1	1	1
falt	3	0	9	3	1
trykk	7	0	1	7	1
munn	2	0	0	0	0
kompresjoner	1	0	2	1	1
innblåsninger	4	0	0	0	0
gjennoppliving	1	1	23	0	0
Sum	28	5	78	17	0,607142857

Table 5.1: The number of spoken words, True positives, False positives, False negatives and WER for the keywords in simulation 5

By comparing the entire transcription from simulation 5 with the output from the model it was determined that only 5 of the 28 keywords in the simulation was classified correctly. This meant the sum of the false negative rate in the simulation was 60.7 %. The number of false positives was 78 meaning the false alarm rate in the simulation was 278 %.

Admittedly this was not as good of result as the first experiment. For this reason the classification errors in simulation 5 was analyzed, as seen in figure 5.2, and could be summarized in threes flaws with this speech recognition method: *False negative due to the prediction threshold*, *False negative due to segmentations*, *False negative due to model.*

False negative due to segmentation was the most common reason for a false negative in simulation 5 and was the result of speech segmentation algorithm would either removed or cut the audio files containing a keyword incorrectly. This could for example be that as a result of the algorithm classifying a speech segment as silence and removing it from the prediction or the algorithm cut the audio file containing a keyword in two septate segments. As a result of these 10 keywords were false negatives in the prediction.

False negative due to Model was a result of a false negative in the prediction. As a result, the model was unable to predict a keyword from the segmentation algorithm correctly. This meant that the segmentation algorithm was able to cut the keyword out correctly, but the model was not able to classify the correct result. Because of this there were 5 false negatives in simulation 5 due to this model.

False negative due to the prediction threshold was the misclassification due to the prediction threshold in the audio file. The prediction threshold was set to 0.9999 to avid other non-trainable classes from being classified. Speech segments with classification output lower than this threshold was classified as a non-class and removed from the output. This meant that the speech segment belonging to a specific class with a low percentage prediction, would not be a output from the prediction. As a result of this there were 7 false negatives in simulation 5 due to this threshold.

	False negative due to		
Utterance	The prediction threshold	segmentation	Speech recognition model
Puste	1 [2]	1 [3]	1 [4]
Tell	1 [4]		
Dypt			
Hardt		1 [3]	
Falt	1 [1]	1 [4] ,1 [2]	
Trykk	1 [1],1 [1],1 [1]	1 [3],1 [3]	1 [1],1 [1],1 [1],1 [1]
munn		1 [2] ,1 [2]	
kompresjoner	1 [4]		
innblåsinger		1 [3],1 [2] , 1 [2] , 1 [2]	
gjennoppliving		1 [4]	
Sum	7	11	5

[1] The speech segment is classified incorrectly due to low

[2] The speech segment is cut before the word ends.

[3] The speech segment is considered non-speech and is removed from the classification.

[4] The speech segment is longer than 1 second, and is automatically cut.

[5] There is noise in the speech segment.

Table 5.2: Reasons for false negatives in simulation 5.

5.4 Experiment 3

Figures 5.10 and 5.11 were the results from the transcriptions from the google speech recognition model. The red boxes indicated were a keyword were spoken.

00:00:30 viktigste gå til han svarer han da Hallo Nei Nei det er ikke noe liv ikke noe selvtillit sier du om han $\boxed{\text{puster}}$ normalt
00:00:40 Nei det virker ikke som han Nei han strever med å $\boxed{\text{puste}}$ du hører om hoster og harker i så ligger han på ryggen eller Ja
00:00:50 hvis du gå inn på hodet hans godt bakover $\boxed{\text{puste}}$ ha noen bedre da synes du
00:01:00 hosting men men det er liksom rar $\boxed{\text{pusting}}$ Jeg vet ikke

Figure 5.10: The transcription output for simulation 5 using Google's speech recognition model.

00:01:30 levetiden av brystet til til denne Pasienten få se du hendene dine midt på pasientens brystkasse sitt helt inn til gjennombrudd
00:01:40 Og så $\boxed{\text{trykker}}$ du brystkassen ned 5 cm i takk på 100 minutter jeg skal hjelpe deg å $\boxed{\text{telle}}$ vi $\boxed{\text{hardt}}$ og dypt Vestre Aker

Figure 5.11: The transcription output for simulation 5 using Google's speech recognition model.

Of the 10 keywords in the original transcription only 8 keywords was classified correctly with the Google's speech recognition model. This gave a false negative rate of 0.2 and a false alarm rate of 0.

The results from the analysis of the entire output from the Google's speech recognition model with the transcription are described in table 5.3. Of the entire dataset there was 17 false negatives, 1 false positive and 8 false negative. This resulted in a false negative rate of 0.28 and a false alarm rate of 0.03.

Simulation 5	Google API				
Keywords	Number of words	True positive	False positive	False negative	WER
puste	4	4	0	0	0
tell	5	3	0	2	0,4
dypt	0	0	1	0	0
hardt	1	1	0	0	0
falt	3	2	0	0	0
trykk	7	4	0	3	0,428571429
munn	2	2	0	0	0
kompresjoner	1	0	0	0	0
innblåsninger	4	1	0	3	0,75
gjennoppliving	1	0	0	0	0
Sum	28	17	1	8	0,285714286

Table 5.3: True positive, False positive, False negative and word error rate in Simulation 5, Google API

5.5 Summary

The mean word error rate and false alarm rate for all the simulation was calculated by analyzing the 6 of the 33 simulation. The number of keywords , True positives, false positive and false negatives was found and each of the simulation both for experiment 2 and 3. The mean word error rate was calculated by finding the mean of the sum word error rate of all 6 simulations. The false alarm rate was calculated the same way.

	WER	False Alarm rate
Experiment 2	76.4%	282%
Experiment 3	32.48%	1.91%

Table 5.4: Summary of false alarm rate and word error rate in audio files

The result can be seen in figure 5.4. The word error rate and the false alarm rate was 32.48 % and 1.91% for the Google speech recognition model. While the word error rate and the false alarm rate was 76.4% 282% for the CNN model.

Chapter 6

Discussion

In this chapter the result and experiments from chapter 5 are summarized and further discussed.

6.1 Experiment 1

The first experiment showed how the model was trained on keywords collected from single utterance. Both the accuracy of the training data and the validation data was good, and the model was able to predict the correct keyword from the single utterance.

With this in mind the model was trained with only 100 datasets from 43 different people saying the same word between 1-3 times in a silence environment and some dataset collected from the simulation data. This meant that the model did not have a lot of variations in the datasets it was being trained on. This could result in the model being very good at finding the key features in the data the model was being trained but not good finding these features in non-trainable data. Also since most of the participants would say the same keyword multiple times there was a possibility of model very good at predicting word from the same speaker and could be part of the reasons for the validation and test accuracy had such a good result.

A solution for this problem would be to train the model on more data with more variation between speaker and different noise levels. This was something that was difficult to prepare for in the thesis as there were not a lot of Norwegian data freely available and also because manually cutting the audio data into keywords took time to prepare. Still there a possibility of making this happen if there were more data collected.

6.2 Experiment 2

The second experiment involved testing the simulations data with ECC with the CNN model, to decide whether or not the model could be used to predict continuous audio files. The result from this experiment was a output with both high word error rate and false alarms rate. This was not a good result as a speech recognition algorithm should always predict the correct result in a conversation and not give any false results. A wrong prediction gave a wrong indication of how many times the words was being said in the audio file. The words simulations were not spoken as many times as the model suggested nor where the words that actual was there classified correctly.

The reasons for misclassification was summarized in*False negative due to segmentations*, *False nega-*

tive due to prediction threshold and *False negative due to segmentations*, *False negative due to model* and was the result for method of segmentation, the set threshold and the CNN model as discussed in experiment 1.

One of the biggest problems with the speech segmentation algorithm was the algorithm lack of ability of finding the start and the stop points the keywords. Usually there were silence between the end of a word and the start of a new word. However in continuous speech, when people spoke faster, the silence segments was not always detectable. As seen in figure 4.13 there were often a short window between the end of word to the start of the word. Often there was not a distinctive transition that made the algorithm able to separate the start and stop points of the keywords. This resulted in some cases that two or more words was separated into one file that should only contain only one word.

Consequently if the resulting file was longer than 1 second long, the preprocessing algorithm created to prepare input data for classification would automatically remove the rest of the signal and only keep the first 1 second of the file. This resulted in some cases to that keyword with start point after the 1 second mark, would be removed from the classification and would not give the model a positive output.

Another problem with the speech segmentation algorithm was that the speakers spoke with different energy levels. In some cases, the dispatcher spoken louder than the caller, the caller spoke louder than the dispatcher or the both the caller and dispatcher spoke quietly. Since the threshold from separating speech from non-speech was set to 50 dB, segments of audio that was below this point would automatically be classified as silence. This means that speakers that naturally spoke with a quiet speaking voice could be removed from prediction.

On solution for solving this problem would be analyze the RSM energy in each simulation and to not set a stationary threshold for all the simulations. Since the speakers spoke with different energy levels it would be more useful to analyze a smaller window of the speech signal. Then analyze the energy level in this window and make a prediction to find the start and endpoints of the model. Alternatively, a neutral network could be trained to find the start and endpoints of the keywords [31].

The prediction threshold set to prevent prediction of a non-trained class in the test data. However, this required that one of the inputs had a high percentage prediction output and was compatible with the training data. This was not always the case as sometimes the keywords in the continuous audio file had a low percentage accuracy and the outcome from the prediction would be a false negative.

A simple solution for this would be to reduce the size of the threshold. However, this would result in a lot of more false positive outputs from the CNN model. The false alarm rate with the threshold was the high number 282 % and the rate only increased when the threshold was removed. This was still not a good result and only meant that the model had problems separating the data model was trained on with the data the model was not trained on.

Subsequently a larger dataset with a lot more words collected from continuous speech from more speaker would resolve some of the problems with separating the trained and non-trained data. A s mentioned before, training the model on a larger dataset with more variations would allow the model to learn more significant features that could be used for classification. These features would then not be constantly be confused with features in non-trained data. Still a threshold had to be set to prevent non-trained data from being classified.

6.3 Experiment 3

One of the advantages of using the Google speech recognition model was that the model did not required some preparing of data or training of a model. The Google speech recognizer was essentially just a box where the predicted words were sent inn to the model and a transcription was given in return. This method saved a lot of time of data preparation but had the disadvantage of not being able to customize the model and train the model on new data. If there was a false classification in the output, the error could not be changed or improved.

As a result, the output from this model gave a false alarm rate of 1.91 %. The result was better than the result from the speech recognition on the CNN model. This was partly due to the Google model was trained on a larger set of data but had a larger set of classes to choose from. This meant that model was very unlikely to give a false positive unlike the CNN model.

The word error rate was 32.38 % on the keywords. This was still not the best, but it was still better than the result from speech recognition on the CNN model. A part of the problem was that the Google speech recognition algorithm was trained on Bokmål and was not necessary that good at recognizing accents. Since must of the participants in the simulation spoke with a Stavanger accent, this could be a reason for the model having problems classifying the keywords correctly.

An other problem was that the simulation data had be cut into 10 second segments in order for the model to analyze the input data and give a prediction. This was due the model could not handle audio files longer than 30 second or less than 10 second at time. This meant that the audio file had be cut inn smaller segments before being analysed. The input duration of 10 seconds was chosen as the was the smallest size the model could handle. This resulted in all the audio files was cut into 10 second segments without worrying about when a speaker finished speaking. This resulted in some cases to that word from the audio was cut in to two and could be used for analysis.

6.4 Further work

There is still a lot to be done in order for the original system mentioned in the introduction to work.

On the speech recognition part, the Google model was better at recognizing the keywords then the CNN model. The model had better WER rate and false alarm rate then the CNN model, but was impossible to train the model further. Still this is something that is constantly improved by Google [7]. On the other hand, using a neutral network model trained on the keywords would give more creative freedom, but would take to time to train and improve. The simplest solution would be to use a Google speech recognition system.

Another aspect of the further work is to identify how spoke during the conversation. This was not the focus in this thesis as separating the recorded audio file into speaker turned out to be a more difficult process then previously participated.

It was original suggested that the problem could be solved by recording the conversation separately. This meant that the sound from microphone and the output of the speaker in the telephone recording on the caller's phone would be recorded separately, giving one audio file for the dispatcher and one for the caller. This would have solved the problem of separating the speakers from the beginning and the keywords in the audio file could be analysed separately. However, it turned out there were no application on the phone who could do this easily.

Still there are solutions for solving this problem after the recoding was done. One solution would be to train a Neutral network to recognize who spoken during the course of the conversation. This method is refereed speaker diarization, and would involve analyzing the changing points in the audio and grouping the speech segments into speakers based the speaker characteristic [32]. This would allow transcript output to have both what was spoken and who spoke during the conversation.

Chapter 7

Conclusion

The purpose of this thesis was to record a telephone conversation between a caller and Emergency Communication Centre(ECC) during a situation involving cardiac arrest, and to use an automatic speech recognition system to identify 10 keywords during the course of the conversation.

Two methods were explored to find the keywords in the simulation. The first method involved training a convolutional neural network to identify these keywords. This involved creating a separate dataset with 1 second keywords, convert them to MFCC spectrograms and used them as input to the CNN model.

A speech segmentation algorithm was developed to find the boundaries between the words in the simulation data. These words were then saved in a separate folder and the CNN algorithm would analyze whether this word was same as the words in the model and give a prediction. If the likelihood of the word was one of the keywords where less than 0.9999, then the word was not part of one of the classes. However, if the likelihood was more than 0.9999, the word was classified as one of the keywords.

The other method involved finding the keywords with the help of Googles speech recognition model.

Both of the methods gave the output from the classification as text. This text file was compared with the real transcription from the simulation and the word error rate and false alarm rate was calculated.

The result from this calculation gave a word error rate and false alarm of 32.48 % and 1.91% for the Google speech recognition model. Reasons for the false classifications were due the model was not trained on a Stavanger accent and that audio had to be cut into 10 second speech segments.

Similarly, the word error rate and false alarm rate was 76.4% 282% for the CNN model. This was a worse result than the Google model, however this model was trained on less data with little variation. There was also a problem with finding the boundaries of the words in simulations and as a result some data was removed from the classification. This was partly due to the speech recognition algorithm was to aggressive and would remove wanted data. A better speech segmentation algorithm that would analyze the energy levels of each of the simulations and give a better prediction on the speech boundaries in the simulation data, would solve this problem.

Of the two methods used to identify how spoke in the simulation. The Google speech recognition algorithm was clearly the best, giving the best word error rate and false alarm rate. This model had also the advantage not taking time to prepare.

Bibliography

[1] S. N. Leksikon. (2019) Amk-sentral. [Online]. Available: https://sml.snl.no/AMK-sentral

[2] B. Guldvog, *Norsk index for medisinsk nødhjelp, 4. utgave 2018 opplæringhefte*, 4th ed., Nasjonal kompetansetjeneste for prehospital akuttmedisin(NAKOS), The address of the publisher, 2018, an optional note.

[3] R. A. B. A. B. S. K. B. Kern, R. W. Hilwig and G. A. Ewy, "Importance of continuous chest compressions during cardiopulmonary resuscitation," 2002.

[4] M. Løvås, "Lanserte livreddende app i USA," December 2017. [Online]. Available: https://www.uis.no/fakulteter-institutter-og-sentre/det-teknisk-naturvitenskapelige-fakultet/lanserte-livreddende-app-i-usa-article122350-8106.html

[5] Ø. Meinich-Bache, K. Engan, T. S. Birkenes, and H. Myklebust, "Real-time chest compression quality measurements by smartphone camera," *Journal of healthcare engineering*, vol. 2018, 2018.

[6] google. (2019) google. [Online]. Available: https://cloud.google.com/speech-to-text/docs/languages

[7] Google. (2019) Cloud speech-to-text basics. [Online]. Available: https://cloud.google.com/speech-to-text/docs/basics

[8] Amberscript. (2019) Amberscript. [Online]. Available: https://www.amberscript.com/en

[9] U. Shrawankar and V. M. Thakare, "Techniques for feature extraction in speech recognition system: A comparative study," *arXiv preprint arXiv:1305.1145*, 2013.

[10] C. K. On, P. M. Pandiyan, S. Yaacob, and A. Saudi, "Mel-frequency cepstral coefficient analysis in speech recognition," in *2006 International Conference on Computing & Informatics*. IEEE, 2006, pp. 1–5.

[11] H. Fayek. (2016) Speech processing for machine learning: Filter banks, mel-frequency cepstral coefficients (mfccs) and what's in-between. [Online]. Available: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-

[12] practical cryptology. (2019) Mel frequency cepstral coefficient (mfcc) tutorial. [Online]. Available: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/#computing-the-mel-filterbank

[13] Wikipedia. (2019) Cepstrum. [Online]. Available: https://en.wikipedia.org/wiki/Cepstrum

[14] skymind. (2019) A beginner's guide to neural networks and deep learning. [Online]. Available: https://skymind.ai/wiki/neural-network

[15] DeepAI. (2019) Activation function. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/activation-function

[16] *Patteren classification.* John Wiley Sons,Inc, 2001.

[17] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques.* IGI Global, 2010, pp. 242–264.

[18] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[19] J.-T. Huang, J. Li, and Y. Gong, "An analysis of convolutional neural networks for speech recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2015, pp. 4989–4993.

[20] computersciencewiki.org. (2018) Max-pooling / pooling. [Online]. Available: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

[21] A. E. Sakran, S. M. Abdou, S. E. Hamid, and M. Rashwan, "A review: Automatic speech segmentation," *International Jornal of Computer Science and Mobile Computing, IJCSMC*, vol. 6, no. 4, pp. 308–315, 2017.

[22] I. A. McCowan, D. Moore, J. Dines, D. Gatica-Perez, M. Flynn, P. Wellner, and H. Bourlard, "On the use of information retrieval measures for speech recognition evaluation," IDIAP, Tech. Rep., 2004.

[23] Python. (2019) Python. [Online]. Available: https://www.python.org/

[24] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, 2015, pp. 18–25.

[25] Keras. (2019) Keras: The python deep learning library. [Online]. Available: https://keras.io/

[26] Boldbeast Software Inc., "Call recorder." [Online]. Available: https://play.google.com/store/apps/details?id=com.boldbeast.recorder

[27] smart-pro android apps. Voicerecorder. [Online]. Available: https://play.google.com/store/apps/details?id=com.motion.voice.recorder

[28] otranscribe. (2019) otranscribe. [Online]. Available: https://otranscribe.com

[29] Audacity, "Audacity." [Online]. Available: https://www.audacityteam.org/

[30] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[31] M. R. Brent, "Speech segmentation and word discovery: A computational perspective," *Trends in Cognitive Sciences*, vol. 3, no. 8, pp. 294–301, 1999.

[32] Wikipedia, "Speaker diarisation — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Speaker%20diarisation&oldid=893898740, 2019, [Online; accessed 14-June-2019].

Appendix A

Appendix

A.1 Transcriptions

A.1.1 Simulation 5 Scenario 5 25.02.2019

Dispatcher: Male, Østlandet Caller Female: Østlandet

00:01 Dispatcher: Medisinsk nødtelefon

00:02 Caller: Hei. Du. Jeg trenger hjelp. Det er en som har falt om. Det er en som har falt om ved stokkavannet ved trafostasjonen inne på turstien.

00:10 Dispatcher: Ved trafostasjonen ved Stokkavannet, ja. Og du ringer fra 95440122, stemmer det?

00:15 Caller: Ja, det er korrekt.

00:18 Dispatcher: Ja den personen som er falt om. Er du i nærheten av den?

00:20 Caller: Ja, jeg er rette ved han.

00:22 Dispatcher: Rett ved han. Er han våken og svarer på spørsmål?

00:26 Caller: Nei, Nei. Han svarer ikke i det hele tatt. Jeg har prøvd alt.

00:29 Dispatcher: Hvis du rister godt i han. Svarer han da?

00:33 Caller: Hallo. Nei, Nei. Det er ikke noe tegn til liv.

00:37 Dispatcher: Ikke noe tegn til liv. Ser du om han puster normalt?

00:39 Caller: Ehh. Nei, det virker ikke som han. Nei, han strever med å puste. Du hører han hoste og harke liksom.

00:47 Dispatcher: Åja, okei. Ligger han på ryggen eller?

00:49 Caller: Ja, han ligger på ryggen.

00:51 Dispatcher: ja, okei. Hvis du bøyer hodet hans bakover, puster han bedre da syns du?

00:59 Caller: ehh. Det er litt mindre hosting, men det er litt sånn rar pusting. Jeg vet ikke.

01:04 Dispatcher: Ja, jeg skjønner. Hvis jeg sender hjelpen til deg. Bare hold linja med meg, ikke legg på.

01:08 Caller: uhmum

01:10 Dispatcher: Nå må vi starte gjenoppliving. Greier du å slå telefonen din på høyttaler?

01:15 Caller: Ja, han er på høyttaler

01:16 Dispatcher: Ja, det er veldig bra. Er du alene der?

01:19 Caller: umhum

01:20 Dispatcher: Du er alene, ja. Det er greit. Kan du hjerte-og-lunge redning.

01:24 Caller: Ja, sånn noenlunde. Jeg har gjort det før ihvertfall.

01:28 Dispatcher: Jeg skal hjelpe deg, vett du.

01:29 Caller: Ja.

01:30 Dispatcher: Sett deg på kne ved siden av brystet til denne pasienten. Så plasser du hendene dine midt på pasients brystkasse. Sett helt inntil pasienten

01:38 Caller: Ja.

01:39 Dispatcher: og bruk strake armer og så trykk brystkassen ned 5cm i takt på 100 i minuttet. Jeg skal hjelpe deg å telle.

01:47 Caller: Ja.

01:48 Dispatcher: Så trykker vi hardt med strake armer i denne takten.

01:51 Caller/Dispatcher: 1 2 3 4 5 6 7 8 9 10 11 12

01:59 Caller: kan du telle? Er det riktig takt?

02:02 Dispatcher: Ja, veldig bra.

02:06 Caller: 21 22 23 24. Skal jeg bare komprimere til noen kommer. Jeg vet ikke.

02:11 Dispatcher: Ja, bare forsett nå.

02:13 Caller: Ja.

02:16 Dispatcher: Hvor langt har du kommet nå?

02:17 Caller: Jeg har slutte å telle. Du må telle med meg.

02:20 Dispatcher: Du har sluttet å telle. Okei. Da kan du blåse munn-til-munn.

02:23 Caller: ja.

02:25 Dispatcher: Bøy hodet til pasienten godt bakover med den ene hånden på pannen og så løfter du haken med den andre. Så klemmer for nesen og gir 2 korte innblåsinger. Gjør det nå.

02:37 Caller: ja.

02:39 Dispatcher: Så du at brystkassen hever seg.

02:41 Caller: ja.

D02:43 dispatcher: Kjempefint.

02:44 Caller: delvis.

02:44 Dispatcher: Da forsetter du med 30 kompresjoner.

02:47 Caller/dispatcher: 1 2 3 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

02:58 Dispatcher: Du er kjempe flink.

Caller: 22.

03:00 Dispatcher: Bare forsett

Caller: 22...30

03:05 Dispatcher: Og så gir du to innblåsinger igjen.

03:09 Dispatcher: Kjempebra. Ambulansen er på vei.

03:12 Dispatcher: og så når du har blåst 2 ganger så forsetter du med 30.

Caller: 1..10

03:20 Dispatcher: Trykk med strake armer så blir det mye lettere for deg.

Caller:10.30

03:32 Dispatcher: og så to innblåsinger. Kjempebra. Og kom igjen.

03:39 Dispatcher: og 30.

03:41 Caller: 1..5

03:43 Dispatcher: Du er kjempeflink, du er dyktig.

Caller. 5..24.

03:54 Dispatcher: Bra tempo.

Caller: 30.

03:59 Dispatcher: og to innblåsinger.

04:06 Dispatcher: og så trykker du.

Caller: 1..15

04:16 Dispatcher: Kjempefint.

Caller: 15..30

04:25 Dispatcher: og så blåser du to ganger.

04:32 Dispatcher: og så 30 stykk.

Caller: 1..11

04:39 Dispatcher: du gjør en veldig god jobb.

Caller: 11.30

04:50 Dispatcher: Og så to blås.

04:57 Dispatcher: og 30 stykk.

Caller: 1..5

05:00 Dispatcher: Når ambulansen kommer, forsetter du å trykke helt til de overtar. Ikke stopp når de kommer.

Caller: 14.15.18. Ok. 30.

05:15 Dispatcher: og to blås.

05:22 Dispatcher: Kjempebra. Kom igjen nå. og så trykker du.

Caller: 1..30

05:40 Dispatcher: og to blås igjen. Nå er de snart fremme hos deg.

05:46 Caller: Jeg føler at jeg ikke får til de derende blåsende.

05:49 Dispatcher: Nei. Greier du å bøye hodet godt bakover når du blåser da og ser at brystkassen hever seg?

Caller: 17..30.

06:04 Dispatcher: Bøy godt bakover og klem for neseborene og løft haken frem når du blåser.

06:11 Dispatcher: Ser du at det hever seg.

06:13 Caller: Nei, det så jeg ikke.

06:14 Dispatcher: Nei.

Caller: 1..30.

06:30 Dispatcher: To nye blås.

06:30 Caller: Når er det de kommer.

06:32 Dispatcher: Nei. Dem er der øyeblikk nå.

Caller: 1..30.

06:59 Dispatcher: Ser du noe til ambulansen?

07:01 Caller: Jeg synes jeg hører dem nå.

07:03 Dispatcher: så fint.

Caller: 1. 13.

07:10 Dispatcher: kjempebra, husk å trykk med strake armer så det ikke blir så tungt.

07:13 Caller: Ambulansen er her.

07:15 Dispatcher: Veldig bra.

A.1.2 Simulation 2 Scenario 2 25.02.2019 13.50

Dispatcher: Rogaland dialekt Caller: Rogaland dialekt.

00:00 Dispatcher: Medisinsk nødtelefon

00:02 Caller: Nevøen min har falt i sjøen og drukna. Han puster ikke. Han er død. Han har ligget i sjøen en stund så jeg fant han 50 meter fra brygga vår.

00:09 Dispatcher: Du, vi skal hjelpe deg. Hvor er pasienten? Hvor er ulykken?

00:14 Caller: Likaiveien 13, Hommersåk.

00:17 Dispatcher: Likaiveien 13, på Hommersåk.

00:18 Caller: Ja

00:20 Dispatcher: Ja. Og så sier du at det er din far som har druknet?

00:23 Caller: Nei. Nevøen min har falt i sjøen. Han puster ikke. Han har falt i sjøen. Han er helt livløse.

00:28 Dispatcher: Han er helt livløse ja.

00:29 Caller: Ja

00:30 Dispatcher: Vi skal hjelpe deg. Vi sender hjelp på vei til deg, imens meg og deg snakker.

00:35 Caller: Kjempeflott

00:37 Dispatcher: Ja. Så ambulansen er på vei. Du må starte gjenopplivingen nå. Jeg skal hjelpe deg. Har du pasienten ved din side nå?

00:45 Caller: Ja, det har jeg.

00:46 Dispatcher:Ja. Ja. Og ikke legg på telefonen det er viktig og slå på telefonens høyttaler hvis du kan det. Kan du det?

00:53 Caller: Ja, det har jeg gjort.

00:55 Dispatcher: Ja. Vet du om det er en hjertestarter i nærheten eller noen andre som kan hente?

01:00 Caller: Nei, det er ikke det.

01:02 Dispatcher: Det er ingen andre der. Okey.

01:05 Caller: (vague/utydelig) Nei, det er ingen andre her.

01:07 Dispatcher: Ja. Da er det noen. Kan du hjerte-lunge redning.

01:11 Caller: Ja.

01:12 Dispatcher: Du kan det. Kjempefint. Da legger du pasienten på ryggen.

01:16 Caller: Ja

01:18 Dispatcher: Du har gjort det?

01:19 Caller: Ja

01:20 Dispatcher: Og så sett deg på kne ved siden av brystet.

01:22 Caller: Ja, Jeg har gjort det.

01:25 Dispatcher: Du har gjort det. Og så plasser hendene dine midt på pasientens brystkasse.

01:29 Caller: Ja

01:29 Dispatcher: Så sett helt inntil pasienten. Så det viktig at du bruker helt strake armer nå.

01:34 Caller: hmh

01:36 Dispatcher: ja. Så skal du trykke brystkassen ned 5 cm i en takt på ca 100 per minutt. Så er det viktig at du slipper opp helt opp i mellom kompresjonene.

01:48 Caller: Ja

01:49 Dispatcher: Og så skal du trykke hardt og dypt med da strake armer i den takten. Så kan eg telle med deg. Hvis du begynner nå.

01:55 Caller: Ja

01:56 Dispatcher: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

02:21 Dispatcher: Du kan telle i sammen med meg neste gang.

02:22 Caller: Ja.

02:24 Dispatcher: Nå skal du blåse munn-til-munn. Du kan det?

02:27 Caller: Ja.

02:29 Dispatcher: Ja. Då bøyer du hodet bakover og så tar du en hånd på pannen.

02:34 Caller: Ja

02:36 Dispatcher: Og så løft haken med den andre hånden.

02:38 Caller: Ja

02:39 Dispatcher: Og så klem på nesen. Og så gir vi to innblåsinger.

02:42 Caller: Ja

02:48 Caller: (utydelig) Sånn. Nå var det gjort.

02:49 Dispatcher: Kjempefint. Det hørte jeg. Forsett nå med 30 kompresjoner og to innblåsninger helt til helsepersonell overtar

02:56 Caller: Ja.

02:56 Dispatcher: eller pasienten våkner.

02:57 Caller: Ja.

02:59 Dispatcher: 1

03:01 Caller: Da skal vi. Ja, bare tell høyt.

03:02 Caller/Dispacther: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

03:18 Dispatcher: Litt fortere.

03:20 Caller: 21 22 23

03:20 Dispatcher: Hvis det er enklere. Så kan du starte med å telle til 10.

03:26 Caller: Ja. Nå har jeg tatt 30.

03:29 Dispatcher: Du har tatt 30 ja. Ja. Da bare forsetter du med to innblåsinger.

03:37 Caller: Sånn. Også bare forsetter jeg med kompresjoner?

03:39 Dispatcher: Ja. Så kan du telle til 10 eller 30. Jeg skal høre på deg. Bare tell høyt.

03:42 Caller: 1 2 3 4 5 6 7 8 9 10

03:49 Caller 1 2 3 4

03:50 Dispatcher: Kjempefint. Trykk hardt og dypt. Bruk strake armer og bruk kroppstyngden din.

03:56 Caller: 1 2 3 4 5 6 7 8 9 10

04:03 Dispatcher: Om du ser at brystet hever seg når blåser?

04:06 Caller: Ja.

04:07 Dispatcher: kjempefint. Det er kjempefint. Jeg hører deg når blåser nå. Det er veldig bra. Da bare forsetter du med kompresjoner og innblåsinger helt til det kommer hjelp.

04:18 Caller/dispatcher: Takk, skal du ha.

04:19 Caller/dispatcher: 1 2 3 4 5 6 7 8 9 10

04:25 Caller/dispatcher: 1 2 3 4 5 6 7 8 9 10

04:32 Caller/dispatcher: 1 2 3 4 5 6 7 8 9 10

04:39 Dispatcher: Og så 2 innblåsinger.

04:42 Dispatcher: Er det noe tegn til liv? Ser du noe?

04:44 Caller: Nei.

04:45 Dispatcher: Nei. Da bare forsetter du. Du er kjempeflinke. Veldig bra. Bare forsett sånn.

04:50 Caller: 1 2 3 4 5 6 7 8 9 10

04:53 Caller: 1 2 3 4 5 6 7 8 9 10

05:00 Caller: 1 2 3 4 5 6 7 8 9 10

05:13 Dispatcher: Kjempebra. Brystet hevet seg ikke sant.

05:16 Caller: Ja

05:17 Dispatcher: Ja. Veldig bra. Bare forsett. Nå er hjelpen din på vei. Så de er der om ikke så alt for lenge.

05:25 Caller: 1 2 3 4 5 6 7 8 9 10

05:28 Caller: 1 2 3 4 5 6 7 8 9 10

05:43 Caller: 1 2 3 4 5 6 7 8 9 10

05:54 Dispatcher: Bruk hele tyngden din på kroppen. Fordi da blir det lettere for deg.

05:59 Caller: Jeg merker det. Hvis jeg holder armene stive så går det greit.

06:02 Dispatcher: Ja, Ja. Du merker nok det. Det er kjempebra jobbet. Og når du bruker hele kroppen, så blir det lettere for deg. Fordi når har du holdt på en stund. Da blir det tungt.

06:11 Caller: 1 2 3 4 5 6 7 8 9 10

06:13 Caller: 1 2 3 4 5 6 7 8 9 10

Caller: 1 2 3 4 5 6 7 8 9 10

06:18 Dispatcher: Du holder en veldig fin takt nå. Så bare forsett sånn.

06:21 Caller: 1 2 3 4 5 6 7 8 9 10

Caller: 1 2 3 4 5 6 7 8 9 10

Caller: 1 2 3 4 5 6 7 8 9 10

06:28 Dispatcher: Nå seg jeg at de nærmer seg her. Ambulansen. Så kommer brannvesenet og der. Så bare forsett.

A.1.3 Simulation 11, scenario 1, 04.03.2019

00:00:00 Dispatcher: Medisinsk nødtelefon

00:00:01 Caller: Hei. Jeg trenger hjelp. Eh

00:00:03 Dispatcher: Hva du?

00:00:04 Caller: Lars ligge på gulvet i han er blek ansiktet

00:00:09 Dispatcher: Ja. Hvilken adresse er du på?

00:00:10 Caller: Vi er i kistebergstien 32

00:00:15 Dispatcher: Sa du Tyttebærstien

00:00:17 Caller: kistebergstien 32

00:00:19 Dispatcher: Ja. Hvilken kommune er det i?

00:00:22 Caller: det er i Stavanger kommune

00:00:24 Dispatcher: og du ringer fra telefonnummer?

00:00:26 Caller: 90 60 ** **

00:00:30 Dispatcher: Hva er det som er sjedd for noe sier du?

00:00:32 Caller: Laren ligger på gulvet. han er blek i ansiktet Jeg får ikke kontakt. Han har fått et anfall. Armene beveger seg.

00:00:41 Dispatcher: Er han våken. Puster han normal?

00:00:44 Caller: Han puster. Han lager noen snorkelyder, men

00:00:48 Dispatcher: Puster han som meg og deg?

00:00:51 Caller: Nei det gjør han ikke

00:00:52 Dispatcher: Det du må gjør. Du må sette deg ned ved siden av pasienten og så gi pasienten fri luftvei. så må vi sjekke hvordan pasienten puster. Dette bruker du ca. 10 sekund på. Vet du hvordan du gjør frie luftveier?

00:01:04 Caller: Er det du ser inn i munnen om det er noe eller er det?

00:01:07 Dispatcher: Ja, hvis du setter deg ned ved siden av pasienten, legge en hånd i pannen og en hånd under haken og så bøye hode forsiktig tilbake sånn at du får hodet litt bakover. Så kan du jo se nedi munnen om det er noe i munnen. Hvis ikke så bøyer du øret ditt og skinnet ditt ned til nese og munn, og ser du om brystkassen eller øvre delen av magen beveger seg, mens du kjenner om det kommer pust ut.

00:01:35 Caller: Han puster ikke lenger

00:01:40 Dispatcher: Han puster ikke. Da må du starte hjelp hjerte-lunge-redning. Hjelp er på vei og jeg skal hjelpe deg Ikke legg på slå telefonen på høytaler hvis du kan det.

00:01:50 Caller: ja ok. Den er på høytaler nå.

00:02:00 Dispatcher: Kan du hjerte-og-lungeredning?

00:02:00 Caller: Nei, det kan jeg ikke.

00:02:00 Dispatcher: Nei. Da legger du pasienten på ryggen på gulvet.

00:02:03 Caller: Ja

00:02:04 Dispatcher: sett på kne ved siden av bryste. Plasser hendene dine midt på pasientens brystkasse. Sitte helt inn til pasienten og bruke strake arm.

00:02:14 Caller: Okei

00:02:14 Dispatcher: Trykk brystkassen ned 5 cm i en takt på 100 minutter og slipp helt opp mellom kompresjonene.

00:02:24 Caller: Okei

00:02:24 Dispatcher: Trykk hardt og dypt med strake armer i denne takten:

00:02:27 Dispatcher: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25 26 27 28 29 30

00:02:45 Dispatcher: Sa teller du høyt i sammen med meg og så fortsetter du med det

00:02:48 Caller: Ja, ok.

00:02:49 Caller: 1 2 3 4 5 6 7 8 9 10 11 12

00:02:59 Dispatcher: Prøv å komprimer litt fortere er du grei.

00:02:59 Caller: 12 13 14 15 16 17 18 19 20

00:03:06 Caller: Veldig bra. Når har du fine takt og du teller høy. Det er godt

00:03:08 Caller: 24 25 26 27 28 29 30.

00:03:13 Caller: Skal jeg bare forsette med dette?

00:03:14 Dispatcher: Bare forsett med å trykk du.

00:03:17 Caller: Ja ok

00:03:18 Caller: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 34 35.

00:03:44 Dispatcher: veldig bra. du telle veldig bra. Du holde en fine takt du gjør en god jobb.

og husk på å ha strak arm og bruk kroppsvekt din til å trykke.

00:03:53 Caller: Ok. 2 3 4 5 7 8 9 10

00:04:10 Dispatcher: Husk å telle høyt er du grei. Hvis du synes det er lettere å telle til 30 og bare gjenta det. Så bare gjør du det. Hvis ikke så teller du til 30.

00:04:23 Dispatcher: 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7

00:04:23 Dispatcher: 5 6 7 8 9 10

00:04:27 Caller: 1 2

00:04:27 Dispatcher: 1

00:04:29 Caller: 2

00:04:29 Dispatcher: 3 4 5 6 7 8

00:04:33 Caller: 9 10

00:04:34 Caller: 1 2 3 4 5 6 7 8 9 10

00:04:41 Caller: 1 2 3 4 5 6 7 8 9 10

00:04:48 Caller: 1 2 3 4 5 6 7 8 9 10

00:04:58 Dispatcher: Du kompremer i den takten som du teller ikke sant?

00:05:12 Caller: Ja

00:05:12 Dispatcher: Veldig bra. Bare forsette. Bruk strake armer og kroppstyngen din slik at du får mest mulig hjelp i tyngden i kroppen. Bare forsett.

00:05:12 Caller: to tre fire fem seks

00:05:16 Dispatcher: Og telle høyt

00:05:16 Caller: sju åtte ni ti 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 to tre fire fem seks syv åtte

00:05:59 Dispatcher: Du gjør veldig god jobb. Hjelp er på vei som sagt også bruk strake armer og brukt kroppstyngden din. Og tell høyt.

00:06:09 Caller: 1 2 3 4 5 6 7 8 9 10 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8

00:06:34 Dispatcher: Bare fortsett å komprimere. når hjelper komme frem Så er det viktig at du forsette å komponere helt til du får hjelp til og de overtar. Ikke stopp opp før de hjelper deg.

00:06:47 Caller: 0 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 2 3 4 5 6 7 8 9 10 1 2 3 5 6 7 8 9 10 3 4 5 6

00:07:19 Caller: 7 8 9 10

00:07:22 Dispatcher: veldig bra bare fortsette du. Tell høyt. Bruk strake armer.

00:07:27 Caller: 1 2 3 4 5 6 7 8 9 10 1 2 3

This speech-to-text was automatically created by www.amberscript.com and some corrections

A.2 Python code

: The full folder of codes used in the project can be collected from website . An short overview of all the python files in the folder:

- **go4.py-** set up gorina 4
- **model.py** - CNN model
- **preprocessing.py** - labelling the data, create MFCC, save MFCC and create training an validation data.
- **speech_recognizer.py** - speech recognition using Googles speech recognizer.
- **Training_validation.py** - training the model on English dataset.
- **Retraining.py** - retraining the model.