UPPSALA
UNIVERSITET

UPTEC IT 19021
Examensarbete 30 hp
November 2019

Snore Detection In Uncontrolled Environments Using Neural Networks for Mobile Devices

Johan Windahl

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Snore Detection In Uncontrolled Environments Using Neural Networks for Mobile Devices

Johan Windahl

Sound classification in audio data with the labels Snoring and Non-snoring has been created. A wide range of neural networks has been created. The networks have a diverse complexity consisting of three different architectures. The dataset used for the models was preprocessed using three separate audio window lengths. The dataset used was assembled from audio recordings in a sleep tracking mobile application and a subset of a large independent dataset. A total of 129 models were implemented, compared and analyzed. One model was elected winner based on desirable preferences, with an accuracy of 96,38%. The conclusion was drawn that the simpler models combined with a shorter window length was preferred over the larger complex models with longer window length, with the chosen preferences in mind.

Handledare: Petter Lönnstedt
Ämnesgranskare: Olle Gällmo
Examinator: Lars-Åke Nordén
UPTEC IT 19021
Tryckt av: Reprocentralen ITC

Sammanfattning

Ljudklassificering av ljuddata med klasserna snarkning och icke-snarkning har skapats. Ett brett spektrum av neurala nätverk skapades. Nätverken hade varierad komplexitet och var utav tre olika arkitekturer. Datamängden som användes till modellerna förbehandlades med hjälp av tre separata ljudfönsterlängder. Datamängden sammansattes med inspelad sömndata från en mobilapp och en delmängd av en stor oberoende datamängd. Totalt blev 129 modeller implementerade, jämförda och analyserade. En modell korades som vinnare baserat på önskvärda preferenser med en precision på 96,38%. Slutsatsen drogs att små enklare modeller kombinerat med en kortare fönsterlängd var att föredra gentemot stora komplexa modeller med lång fönsterlängd, utifrån de valda preferenserna.

Acknowledgements

My reviewer: Olle Gällmo

The team at Urbandroid, Jan Marek and Petr Nálevka

The team at Consid, Mikael Axelsson and Petter Lönnstedt

Contents

1 Introduction

Every day, all around the world we humans share a common activity, sleep. This activity is a vital part of our life used for restoration and recovering. Even though sleep is a widely-spread phenomenon, it's still in a sense a mystery. It has an unconscious and unknown element that leaves many questions unanswered.

When a person enters sleep this unconscious element takes over, an automatic deeper state grasps the motor control and mind. A side effect while sleeping is snoring which is common among the population and is one part of this mystery. The fundamental cause and whether it is dangerous is not confirmed. However, are there substantiated health risks connected to heavy snoring. [27] [36].

Advanced research in these domains of sleep and snoring was first only available to people with controlled environments, advanced equipment, sensors, and time. There is no perfect sleep assessment method, but one of the current (2018) most accurate methods is Polysomnogram (PSG) ([29]). PSG is an extensive recording of the physiological changes that occur during sleep, the main drawback is its expensiveness and exclusiveness.

A part of the sleep assessment field has now opened up as the technology has developed and with the introduction of the smartphone. Today a common man can use his mobile device to measure different aspects of his sleep without any extra equipment[63] [47]. The main idea behind this task is using the accelerometer of the device to measure the body's physical movement during a sleep session. Then use the data recorded to mimic advanced sleep equipment differentiating the sleep-stages throughout the session. A side-effect during this process is the possibility to use the microphone for audio recording. The audio can later be used to detect whether snoring or other sound-events are occurring during the session.

In this project, automatic snore-detection within audio files has been enabled which has many practical advantages.

- To find potential health risks included in various forms of sleep apneas using a common mobile device. Recent evidence shows that these obstructive sleep apneas is greatly under-diagnosed and has a high prevalence in the population. [76]
- The opportunity to automatically identify snoring. This enables the potential for only saving the relevant segments of data during a recording session when the actual event occurs. Instead of the requirement to save the whole recording.
- The possibility to use real-time snore prevention tools, letting the user know that snoring is occurring. This could improve sleep quality for the user and other nearby parties.

The snore detection system has been created via different Machine Learning[6] (ML) models. The models are various Neural Networks [24] (NN) that have been implemented, compared,

and analyzed. The different models created has also been trained with different feature extraction setups of the audio data to find tendencies how fast a proper preliminary prediction can be made. A broad spectrum of models are presented and a proposed model has been chosen as the best one by analyzing the results of desirable priorities presented by an external company. Priorities such as:

- A model that can distinguish between two sound events *snoring* and *non-snoring* with great performance in audio files.
- A model as uncomplicated as possible but that is sufficiently complex to deliver great performance.

2 Background

Snoring is a state during the sleep that causes noise via the breathing air. The flow of air passes through tissues in the back of the throat that leads to vibrations, this is the source of the sound. Snoring is a frequent predisposition within the population, this was concluded by an epidemiological survey in the 1980s. The survey concluded that on average 19% of the population were habitual snorers. Where 24.1% were men, and 13.7% were women. The sample size of this survey was 5713 people.[43] The snoring frequency and intensity for a person can be increased by several components that are somewhat preventable, such as:

- Stimulants, such as drugs or alcohol. They may have a relaxing effect on the throat muscles. [32]
- Nasal congestion from colds, flu or allergies. The accumulation of mucus may disturb the natural breathing function. [75]
- Weight, abdominal obesity is a great predictor of various sleep disorders [21]
- Sleep position, snoring is most common when lying on the back. [55]

Light snoring may not be dangerous or disrupt the quality of sleep. Heavy snoring may, tendencies have been found between heavy snoring and different levels of sleep apneas. [33] Sleep apnea is a breathing disorder that leads to states where shallow or no breathing occurs. The term is valid if the low-breathing-state exceed a period longer than 10 seconds.[22] There are multiple levels of sleep apneas which are defined by how many of these sessions occur per time unit. Obstructive Sleep Apnea (OSA) is the most common apnea.[22] These apneas may lead to different levels of oxygen-deprivation, which is not healthy for the human body. It can lead to multiple serious consequences such as high blood pressure, atrial fibrillation [17] , stroke[15], type II diabetes [2], other cardiovascular problems[46], and some research even points to cancer. [77]

This project has been constructed with help from a company called Urbandroid. Urbandroid is a software development company started in 2010[68]. Their most successful application is *Sleep as Android* which is a sleep-tracking tool for private use. The application runs on the operating system Android[18] for mobile devices and was released the year 2010 [47]. It currently has millions of active users over 10 million downloads (2019-02-06) and a rating of 4.3 stars (of 5) over 270 000 ratings on Google Play [30].

Sleep as Androids main purpose is to gather information around the users sleep in order for the user to improve it. The applications main feature is tracking the user's physical movement during the sleep via the device's accelerometer. Below is an image (figure 1) of a complete session with recorded data. The accelerometer data is shown in the top graph (black bracket, actigraph). The red line in the actigraph illustrates an optional recorded heart rate, via a heart rate monitor.

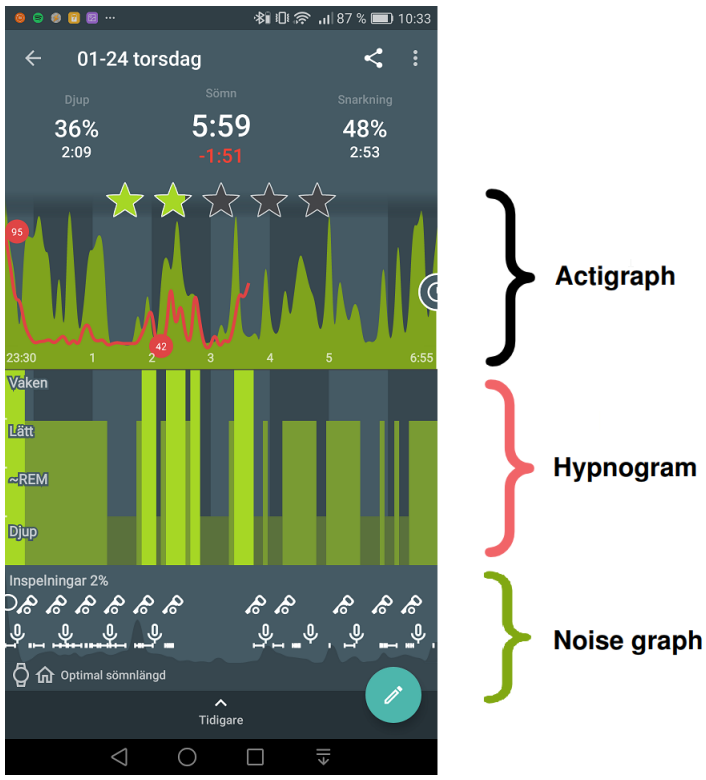

Figure 1: A screenshot of the application *Sleep as android* in the view of a sleep session

The accelerometer data is then used to mimic advanced sleep equipment to differentiate between the sleep-stages throughout the session. These approximated stages are shown as a hypnogram (red bracket) The possible sleep phases are awake, light, medium, or deep. Another feature in the application is that it records audio throughout the session via the device's microphone and represents the recorded audio as a noise graph (green bracket). This audio is analyzed in real time by an automatic system that identifies snoring and talking. If the system detects an event it is registered in the noise graph with a symbol and saved as an audio file (10 seconds long).

Sleep as androids current snore detection implementation is under constant development. Urbandroids developer team has explored many different approaches for implementing and improving this feature. Their latest implementation use a pre-processing method which generates the mel-frequency cepstral coefficients[41] (MFCC) (described in section 3.2) on recorded audio. Their implementation uses three-second long audio segments which converts the digital sound waves into a new representation of the sound as an image in the form of a heatmap (an example can be seen in figure 8). This MFCC approach is a common well-documented state-of-the-art solution that provides a compact and smooth representation of how the local frequency spectrum changes over time. These extracted heatmap images is used as a fingerprint and fed through a machine learning system that is based on a convolutional neural network[37] (CNN). A CNN is a class of deep neural networks, specialized for analyzing digital images.

2.1 Machine Learning

Machine learning (ML) is a set of algorithms and statistical models that computers use to solve given tasks without explicit instructions. Many of the fundamental algorithms and machine learning ideas origin from decades ago. Machine learning is considered as a subset of Artificial Intelligence (AI).

AI is a system that ”learns over time” without explicit instructions. The grandfather of AI is Alan Turing who is famous for inventing the Turing Test (1950) [70]. This test is used to determine if machines could be indistinguishable from humans by a human tester by answering questions. Around the same time (the 50s) Frank Rosenblatt introduced a fundamental part of machine learning and Neural Networks[24], the perceptron [57]. The theory behind the perceptron was based on a famous rule proposed by Donald Hebb in 1949[25]. Hebb's rule is simplified and summarized as: ”neurons wire together if they fire together”[42], this rule is the foundation of NN as they are today. One of the limitations with the perceptron is the inability to solve linear inseparable problems, this was pointed out in 1969 in the book Perceptrons[45]. This reduced the popularity in the research field for some decades.

In 1974 the Multilayer Perceptron (MLP) is introduced, combined with the back-propagation algorithm[74], and further popularized in 1986 [59]. These events made the field around Artificial Neural Network (ANN) popular again. Back-propagation is a learning algorithm that mimics Hebb's rule within the network structure. It iteratively changes the weights between nodes by trying to minimize a certain parameter given by a specific problem. The weights are modified based on desired output behavior.

A few years later (1989) a customized ANN for recognizing handwritten numbers for ZIP-code identification is released [38]. The Architecture of this ANN is different, this network had multiple hidden layers and used filters that convoluted over the current feature space and returned the information with reduced complexity. This was one of the ”grandfathers” of both Convolutional Neural Network (CNN)s and Deep Neural Network (DNN)s. The definition DNN is rather vague, but implies that it is a ANN, but with multiple hidden layers used on more complex problems.

In parallel, another branch of ANNs was researched called Recurrent Neural Network (RNN)s. The theory behind them was published long before in 1974 by Little [40], then later (1986) popularized by John Hopfield in [28] and by David Rumelhart (1988) [59]. These networks added a parameter of ”memory” to give the previous sequence of data relevant to the current prediction. This was preferable for sequential problems such as speech recognition, or handwritten letter recognition.

The latest accomplishments within AI and machine learning are conquering complex games with gigantic state spaces, by beating top level humans. In 2016 Googles Deepmind team created a system called AlphaGo, the system mastered the game of Go and defeated professional players.[64]
In 2018 the same team created AlphaStar[67], this system conquered the game of Starcraft 2,

also by beating professional players. Starcraft 2 is a real-time strategy incomplete information game with a state space that is many orders of magnitude larger than Go. Go's estimated state space is around 10^{170} [48]

2.2 Problem statement

- Which neural network architectures seem to be favorable for this sound event classification problem?
- The current implemented pre-processing technique uses a three second sound segment length for generating input features. Is it possible to shorten this window and still maintain a reasonable accuracy by the created networks?
- What network complexity seems reasonable to sufficiently handle this sound event classification problem? How does that change as the window length increases?
- What potential suggestions could be proposed to further develop Urbandroids current snore-detection implementation?
- Based on all findings, which created model is best one overall?

2.3 Delimitations

- Neural networks are the only techniques considered. More specifically three different neural network architectures are considered, MLP (multilayer perceptron), CNN, and RNN. No new machine learning algorithms or architectures are introduced as classifiers. This delimitation was made to compare three fundamentally different basic neural network architectures on this given problem. A comparison between how they approach the problem with the given inputs. The MLP is used as a baseline since it has the least complex structure, CNN which focuses on capturing visual patterns in images and RNN which uses previous states as inputs to the current state.
- Three different audio window lengths (100, 1000, and 2500 milliseconds) are considered in the feature extraction process of the sound events. This delimitation was made to compare three different lengths and how short the window has to be to still consist of sufficiently useful information for delivering reasonable performance. One *very short* (100 ms), *one whole sound event* (2500 ms), and one *about equal space in between* (1000 ms) windows were chosen.
- The created models will not be tested in a live environment on a mobile device. The potential of each model will be analyzed by predicting sound events on an independent testing data set. The models created are left with the possibility of exportation and usage. This delimitation was made to focus on only the machine learning model creation and pre-processing of the audio data.

- No collection of new audio data via recording devices is done. This project assembles a dataset of already recorded audio. This thesis is mainly focused on the machine learning aspect of sleep research. This delimitation was made to focus on only the machine learning model creation and pre-processing of the audio data.
- The sound events are treated like they are mutually exclusive. This is a delimitation made to simplify the machine learning model creation.

3 Theory

3.1 Machine Learning & Neural Networks

Supervised Learning
Supervised Learning (SL) is a subset of machine learning. The main purpose is to create a model that approximates an unknown function. The model will be trained by using labeled training data. The training data consists of samples with data pairs, inputs together with correct outputs. The input will be fed through the model iteratively. After each iteration the network has variables it will nudge its towards a better approximation of the unknown function. When the training phase is over the model can estimate a result based on the training phase.

Artificial Neural Network
One neural network structure is the Artificial Neural Network ANN whose behavior is intended to mimic the neural networks biologically created in our brain [24]. The neurons collectively work together to ”learn” a solution by minimizing some parameter in a given problem. If a correct pattern is presented, the connection between neurons that fired is strengthened. ANN is an umbrella term for all neural networks, these networks can be connected in many ways, have different sizes, and structures, each will yield different learning behavior. The basic building block in an ANN is a perceptron. A more advanced structure of perceptrons interconnected in many layers (at least 3) is the MLP[50]. (an illustration of a MLP can be seen in figure 2) A new component in the MLP is an activation function [44], which adds non-linear properties to the whole component so it can approximate more complex functions.

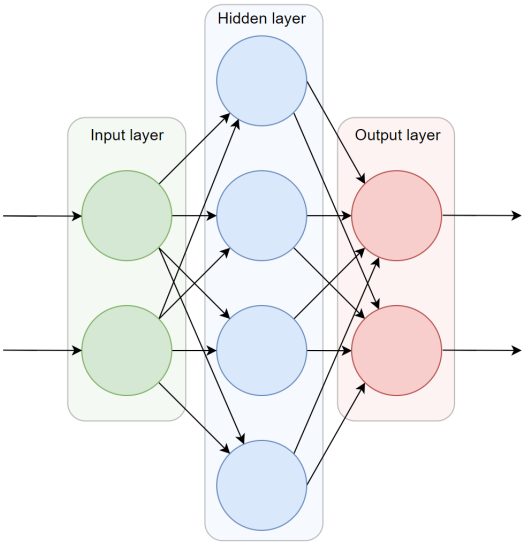

Figure 2: A multilayered perceptron with two input nodes, four hidden nodes and 2 output nodes.

Recurrent Neural Network A RNN is a type of an ANN but with a slight structural change.

Instead of a forward-only network path, the nodes in the RNN have a feedback connection between layers of the network. This structure enables the network to use its internal state as input, and therefore it uses its previous states as input when predicting the current state.

Long Short-Term Memory (LSTM) [26] is a type of neural network with a more sophisticated RNN structure. LSTM enables the possibility to multiple of dependencies over different timescales. Another advantage of LSTM is that it deals with the major problem that sometimes occur with regular RNNs, the vanishing or exploding gradient problem. This happens when the gradients tend to either 0 or infinity, via the backpropagation algorithm in the training phase. LSTM solves this by allowing gradients to flow through the network unchanged.

Deep Neural Network A DNN (illustration shown in figure 3) is a vague term that indicates the depth of a ANN and its structure. A deep ANN with multiple (generally more than 3) layers of hidden nodes connected in sequence.

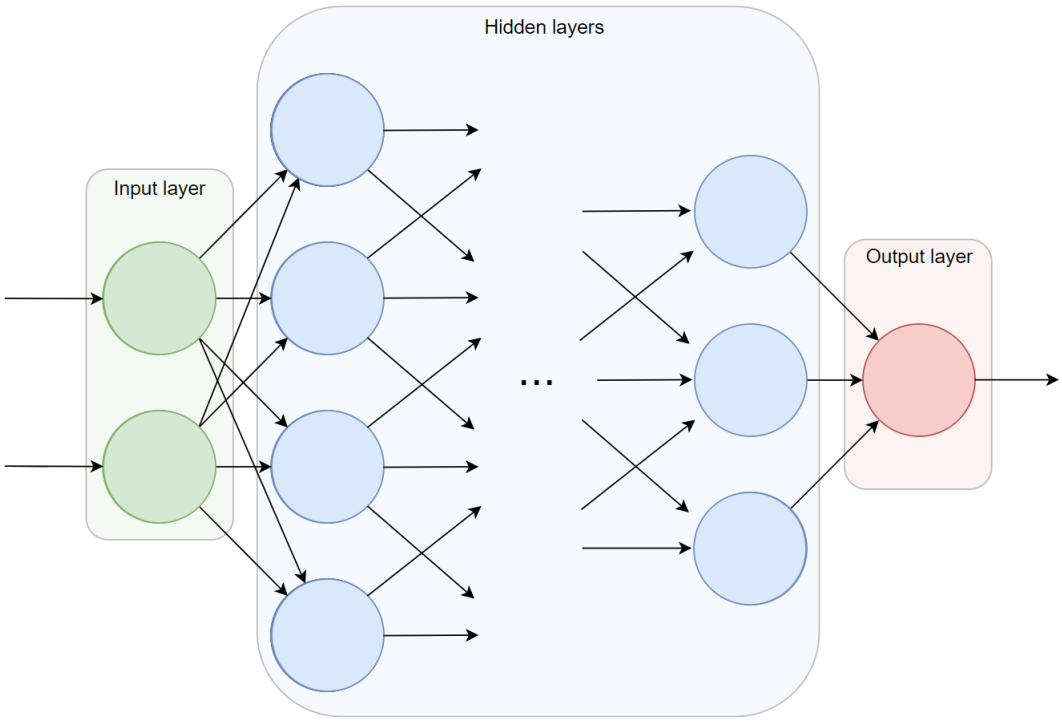

Figure 3: A Deep Neural Network architecture with multiple layers of hidden nodes.

Convolutional Neural Network CNN is a type of ANN and DNN. CNNs are preferably used on a two-dimensional feature space, such as images classification. The architecture of the CNN is different, the hidden layers may consist of different types such as convolutional, pooling or normalization. Pooling layers simplify and down-samples larger parts of the input and can use functions such as max or average. The convolutional layers use a smaller filter that ”slides” over the larger input and are used to find abstract visual patterns in the feature space. An example of a CNN is presented in figure 4.

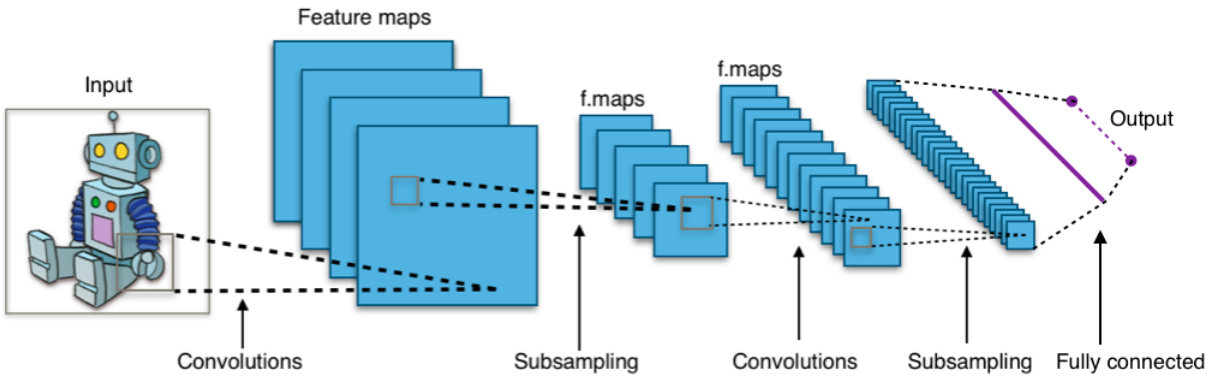

Figure 4: A Convolutional Neural Network example [3]

3.2 Mel-frequency Cepstral Coefficients

Mel-frequency cepstral coefficients (MFCCs) are a state-of-the-art method to extract features from audio signals. The method was introduced by S. Davis and P. Mermelstein in 1980. [11] This method has been continuously verified as a great sound feature extraction method. It has been widely used in various automated speech or music recognition systems. [41] [71] [10]

The main idea behind MFCCs is to mimic the human ear and its listening process. The human ear is much more sensitive to frequency shifts at the lower frequency bands. At higher frequencies, the human ear cannot distinguish between modest frequency changes. MFCCs therefore prioritizes these lower frequencies and filters out frequencies not as likely to be important. The final coefficients will be represented as a spectrogram showing the more important energies in the signal and how it changes over time. The final representations spectrogram will be a kind of a ”fingerprint” for that signal. (See figure 8)

The process to extract the MFCCs from a signal[60] starts with a given signal in the time domain. This is an analog signal saved in digital via a method called Pulse Code Modulation (PCM). (See an example in figure 5). PCM samples the amplitude of the signal at a fixed rate, the sampling rate. From this PCM representation, the signal is fractionated into 20-40ms long frames. All frames are transformed into frequency domain, using the Fast Fourier Transform (FFT) (See an example in figure 6). Next a power spectral density estimate will be calculated for all frames by applying a Mel filterbank containing n (standard is 26) triangular filters. The remaining information will be an array of n constants describing the intensity of the energies in each frame. Each value in this array is logarithmized due to the logarithmic nature of the decibel scale. Each frames array of energies are then represented as a spectrogram with pixels of different colors to represent corresponding intensity. All these created spectrograms are then stacked adjacent next to each other into one larger spectrogram. This spectrogram will now contain information about how the frequency bands energy changes over time. (See an example in figure 7)

The last step is to use the Discrete Cosine Transform (DCT) on the large spectrogram over each frame. DCT is used to distill and downsize the information in the spectrogram. This technique is widely used for e.g image and audio compression [73] [51]. The remaining coefficients are the MFCCs. (See an example in figure 8)

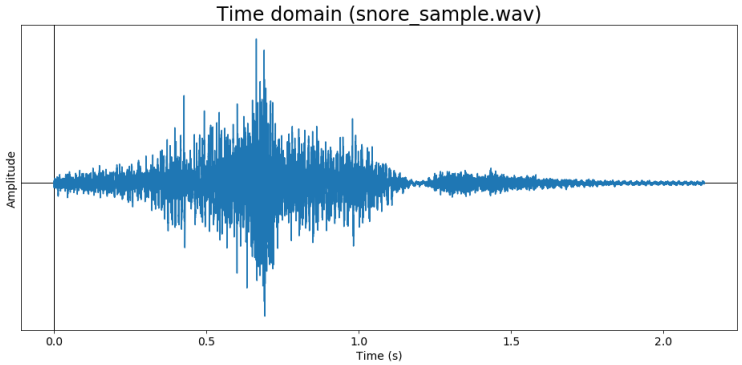

Figure 5: A sample audio file containing a snore in time domain.

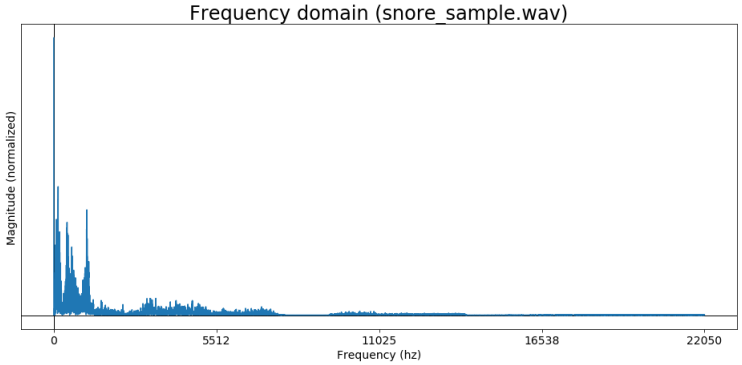

Figure 6: A sample audio file containing a snore in frequency domain.

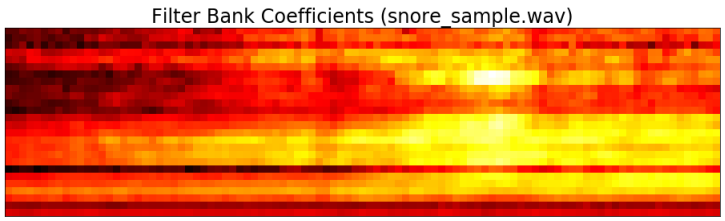

Figure 7: A sample audio file containing a snore, converted to a spectrogram consisting of filter banks coefficients.

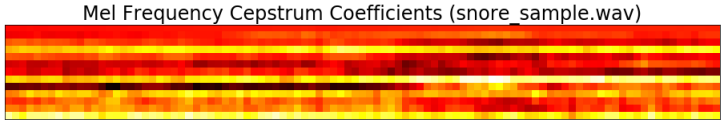

Figure 8: A sample audio file containing a snore, converted to a spectrogram consisting of the MFCCs.

Mel filterbank

This filterbank is a set of triangular filters and the distance between each filter is based on the Mel scale, created by Stevens, Volkmann, and Newman in 1937. [66] The Mel scale is approximated in figure 9. The Mel scale is based on observed equal pitch shifts comparisons and their corresponding frequency, the name Mel comes from the word melody. There is no perfect formula to convert hertz to Mel, there is one that is widely used (See equation (1)) where m equals Mels and f equals hertz.[49]

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

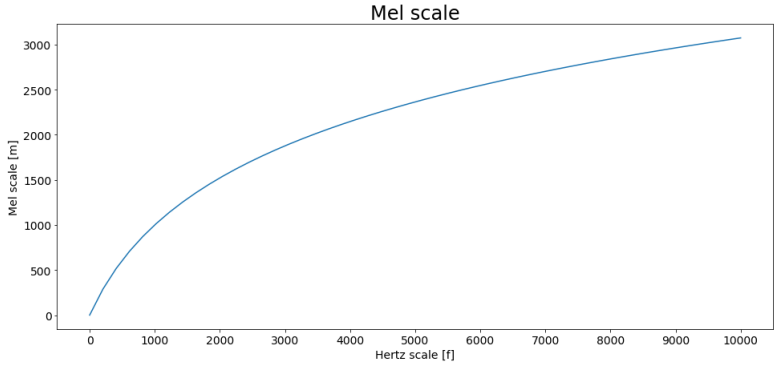

Figure 9: A graph of the relation between Mel and frequency scale.

4 Method

4.1 Libraries

The main libraries used are Keras[8] and TensorFlow[1]. TensorFlow is an open source software library used for numerical computation and it is created by the company Google [72]. Keras is a framework used for raising the abstraction level when creating Neural Network (NN). Keras is created on top of TensorFlow it can also be used with other frameworks such as Theano [5] or Microsoft Cognitive Toolkit (CNTK)[62]. The library streamlines the process of developing models with different architectures and their structure. Keras is implemented for the programming language Python [61].

4.2 Dataset

The dataset used for the experiments consists of 7673 audio files each with a length of a few seconds. The files have a total duration of 5:02:25. Each file has a label, either *Snore* or *Noise*, all files are manually verified as correctly labeled. The class Snore contains approximately 4276 different snores of various people. The class Noise contains a wide variety of environmental sounds that may occur during the night. Sounds such as different variants of static noises, sound artifacts such as humming, creaking, squeaking, created by the recording device or other electronic devices in proximity of microphone. Other sounds are ticks, coughs, sneezes, traffic, talking, distant tv, dogs, and nature. The distribution between the two classes (Noise/Snore) are 59.5%/40.5%.

The dataset was assembled from two main sources. Roughly half of the dataset origin is Google AudioSet[19] a large-scale online dataset of labeled YouTube[58] videos which were made for research. The Google AudioSet consists of 632 audio events, classes and a collection of 2,084,320 human-labeled 10-second sound clips extracted from YouTube videos. [19] One of the labeled classes in this dataset is Snoring and several other relevant classes of different noise and background sound variations.

The other source from the application Sleep as Android and the Urbandroid team. This dataset of sound files are recordings from real sessions recorded by Urbandroid users devices. The dataset is a small subset relative to the applications massive user base. A subset of these recordings is reported by users as misclassified events by the applications current snore detection model.

Table 1: An overview of the final dataset and its content

Class	Source	No. files	Duration	No. seconds
Snore	Urbandroid	1957	00:52:05	3125
	Google Audioset	2319	1:28:38	5318
	Total	4276	2:20:43	12043
Noise	Urbandroid	1854	1:32:02	5522
	Google Audioset	1543	1:09:38	4178
	Total	3397	2:41:40	9700
All		**7673**	**5:02:25**	**21743**

4.2.1 Pre-processing

The data acquisition from various sources generated a huge range in sound quality and format profiles in the assembled sound files, in terms of bit rate, bit depth, sampling rate, and the number of sound channels. Some files used uncompressed data, and some were compressed with lossy audio compression algorithms. The sound length also varied from minutes down to one second. All files were transformed into a common format profile. The format profile was:

- Duration: 1-4 seconds
- Bit depth: 32
- Sampling rate: 16kHz
- Number of channels: 1 (Mono)

These steps were made to transform the files, for synchronization so the files had the same audio profile.

1. (Google Audio set only), a Python script was executed to batch download all relevant video files.
2. (Google Audio set only), a Python script was executed to convert all downloaded files from video to audio.
3. (Google Audio set only) Files were 10 seconds long and contained the labeled sound somewhere within that 10 seconds window. The relevant part of 1-5 second was cut out manually and the rest of the file was discarded.
4. File listened to and verified as a correct label.
5. File was downsampled to a rate of 16kHz. Since the least common denominator in sampling rates was 16 kHz. The rate of 16 kHz is more than enough when using the feature extraction of MFCCs[52] for human sounds.

6. Sound channels were reduced to mono since many of the files missed dual channels.
7. (Snore class files only) Silence was truncated to remove unnecessary information within the files, values below a certain dB were removed. To speed up the training process.
8. Bit depth of all files was set to 32bit.
9. File length was cut down into segments of 5s and less. This means that many files were split into smaller equal parts.
10. Files with the label Snore were split into one snore-event per file.
11. If the file now had a duration of less than one second it was removed.
12. After these steps 7673 files remained.

4.2.2 Feature engineering

Features were extracted from the training data audio clips by generating the MFCCs. The parameters were generated using the settings for the procedure based on the paper[78] where the authors compared different MFCCs setups. The different hyper-parameters in the setups are the number of filters, the shape of the filters, filter spacing. These settings were chosen.

Number of filters in filterbank: 26
Filter type: Triangular filters
The filters are warped along its frequency axis and spaced according to the Mel scale (See equation (1)).
Number of generated cepstrum coefficients: 13
Frame length: 10ms

Based on these settings three different feature engineering setups were created and used. They are distinguishable by the length of the audio input in the MFCCs conversion. Three lengths were chosen, 100ms, 1000ms, and 2500ms. The audio length directly determines the size of the extracted features used for the created model. The MFCC generation technique will create a matrix A of the size:

$$A_{y \times x} \tag{2}$$

where y is the number of generated cepstrum coefficients and x is calculated via equation (3).

$$\frac{a}{f} - 1 \tag{3}$$

where a is the audio length and f is the frame length.

Since the number of cepstrum is 13 and the frame step is 10ms the different sizes of the created input matrices can be seen in Table 2. An example of each of the input feature matrices for the three setups are also visually represented as images in Figures: 10, 11 and, 12)

Table 2: The three different audio length setups and their generated output matrices in the MFCC feature extraction.

Audio length (ms)	**Extracted Feature Size**
100	9x13
1000	99x13
2500	249x13

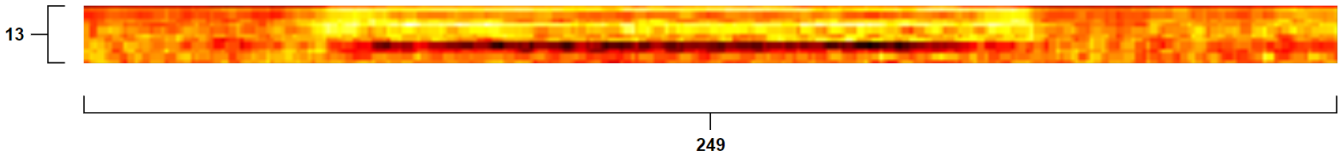

Figure 10: MFCC as features extracted from a 2500ms audio clip, illustrated as an heatmap. (This sample audioclip is from the training data.)

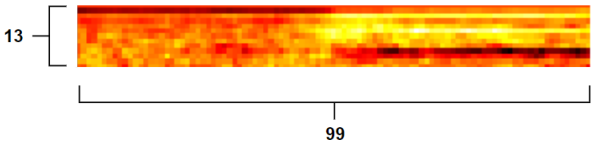

Figure 11: MFCC as features extracted from a 1000ms audio clip, illustrated as an heatmap. (This sample audioclip is from the training data.

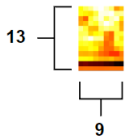

Figure 12: MFCC as features extracted from a 100ms audio clip, illustrated as an heatmap. (This sample audioclip is from the training data.

4.3 System Design

The dataset was split into three pieces 80%, 10%, and 10%. 80% of the data for training the model, 10% for validating and then a 10% as a final test set. The validation set was used in the

training phase and the test set was used to evaluate the results of the finished trained models as an independent data set of unseen data. The class imbalance was approximately 59.5%40.5% (Noise/Snore). The imbalance was handled by considering the probability distribution each time a random sample was drawn from the training data.

4.3.1 Training and Validation

The three different feature extraction methods created three separate variants of input/output mapping that would be fed through the model. Considering that the whole training dataset consisted of 5 hours of audio, the potential to use more input/output mapping samples for the shorter clips was exploited. This created a linear relationship between the length of the chosen window and the total number of samples to use. As the window shortened the samples increased. Below in table 3 each window and its number of potential samples in training data are shown.

Table 3: How the window length and number of samples relates to each other in the training/-validation dataset.

Audio length (ms)	**No. samples in window**	**Training on No. samples**	**Validating on No. samples**	**Total No. samples**
100	1600	291425	32381	323806
1000	16000	29122	3236	32380
2500	40000	14985	1665	16650

The number of samples in each setup was decided by:

$$int(\frac{t*2}{n}) \tag{4}$$

Where t is the total no of samples in all the training data, and n is the number of samples in that window.

That calculated sample size will be the iteration count by this algorithm:

1. Select a class at random, consider the class distribution
2. Select a random file with that class from the training data
3. Select an interval with within that file
4. Calculate MFCC of that selected sound segment
5. Append created MFCC as input matrix
6. Append output label from the selected file

4.3.2 Prediction

The test set was assembled by selecting 800 random files of the whole set (10.4%) from the dataset. The distribution between the two classes were 50-50. 400 files for the class Snore and 400 files for the class Noise. Two different versions of prediction were made, sample-wise or file-wise. Sample-wise a single segment of window length in a file from the test dataset would be compared to the matching label of that file. File-wise a mean would be calculated by sliding and predicting over the whole file, sample by sample and the final mean would decide which class the whole file should be predicted as. As the audio window length shortens it is possible to use more samples in the test set. In table 4 it is shown how the audio window size and the number of samples per file in the data set changed with different MFCC setups.

Table 4: Overview of how the number of samples in the test set change dependent on the different MFCC.

Audio window length (ms)	**No. files in test data**	**No. samples in test data**	**Samples per file**
100	800	12016	15.02
1000	800	1438	1.7975
2500	158	158	1

The algorithm for predicting is:

1. Select all files in the test set one by one
2. Select audio in that whole file, segment by segment (window length size)
3. Calculate MFCC of that selected sound segment
4. Predict class probability based on that calculated segment with the trained model
5. (file-wise) Calculate mean of class probabilities for that file
6. (file-wise) Highest probability will decide which label is predicted

4.4 Implementation

The experiments were conducted with a single python file *model-script* that executed all the steps from reading the audio data to creating a model, training it, and predicting results. This script was used by another script one abstraction level higher, a *main-script*.

This main script created a batch of networks that were automatically created and evaluated sequentially. In these batch mode runs detailed results for each model were saved into a folder and an overview of that model appended into a text file.

The model-script implementation roughly consists of these steps.

1. Read all files in *training* directory, which is the training audio files and saves information about them into a variable.
2. Build the input/output mapping from the training data audio files
3. Create the structure of all possible models considered
4. Select a model and trains it on the training data.
5. (a) Read all files in *prediction* directory.
   (b) Build the input/output mapping from the test data audio files.
   (c) Evaluate prediction accuracy sample-wise
   (d) Evaluate prediction accuracy file-wise
6. Log detailed results into files and graphs.

4.4.1 Algorithms, metrics and methods

Some elements were common throughout all architectures. All networks used Rectified Linear Unit (ReLU) as the activation function for all layers except the output layer. The activation function ReLU was introduced in [23] and made famous for Deep neural networks in [20] which presented a breakthrough within activation functions for neural networks. The author argues why ReLU has advantages compared to sigmoid functions to e.g solve a deficiency that neural networks have been struggling with called the vanishing gradient problem. The vanishing gradient problem is a state in the training phase where the calculated gradient for updating a weight becomes vanishingly small and this will prevent the network effectively from further training. Currently (in 2017) ReLU is the most popular activation function. [54] There are other variants of ReLU such as Softplus which is a smoother version.

$$f(x) = max(0, x)$$

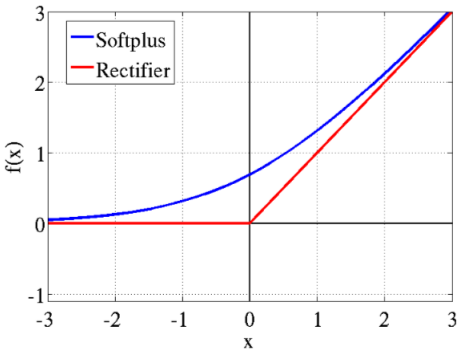

Figure 13: An illustration of ReLU and Softplus. [20]

The activation function for the output layer was chosen as Softmax (seen in equation 5). Softmax or normalized exponential function is a non-linear activation function similar to the sigmoid function but for multi-class problems. Softmax casts the target vector of real numbers into a probability distribution.

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \tag{5}$$

Categorical cross-entropy[12] was used as the loss function for all the networks (seen in equation 6). Intuitively regular cross-entropy can be thought as finding the shortest distance between two vectors and categorical cross-entropy outputs a probability distribution over C classes. It is primarily used for multi-class classification, but can also be used for binary classifications problems.

$$CE = -\sum_i^C t_i log(s_i) \tag{6}$$

Accuracy was used as evaluation metric (seen in equation 7). Accuracy is calculated by adding True Positives (TP) and True Negatives (TN) and divide that by the total number of samples. (The different terms can intuitively be distinguishable in a confusion matrix. (see Table 8)

Table 5: A confusion matrix

		True label	
		Positive	Negative
Predicted label	Positive	True Positive (TP)	False Positives (FP)
	Negative	False Negative (FN)	True Negative (TN)

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

Adam (Adaptive Moment Estimation)[35] was used as the optimizer-algorithm for all the networks. Adam is an adaptive learning rate optimization algorithm and is an extension to the classical Stochastic Gradient Descent (SGD) introduced 1951[56]. The algorithm is a combination of Stochastic Gradient Descent with momentum (AdaGrad[14]), and another SGD extension called Root Mean Square Propagation (RMSprop[69]) which finds individual learning rates for different parameters.

Early Stopping[53] was used in the training phase of all created networks. Early stopping is used to prevent overfitting on the training data and to minimize unnecessary training. Early stopping approximates the generalization error on the trained network each iteration, by calculating a validation error by using the validation subset of data. If the validation error does

not decrease over p number of epochs the training phase will halt. Where p is the selected *patience*. For these experiments, patience was set to 5. A hard cap of 100 epochs was set, so if Early Stopping did not trigger the training phase would not exceed 100 epochs.

Batch Normalization[31] was introduced for improving speed, performance, and some overfitting prevention is. Batch Normalization is argued to work by reducing the internal covariate shift which means that the weights are normalized between layers in the network. Some argue that there is disharmony between Dropout and Batch Normalization [39], therefore many networks were created with a combination of one, the other, or both techniques.

Dropout[65] is a technique introduced as a simplistic and effective method to force the network to be more generalized and robust. It works by sometimes randomly turning off nodes at each iteration in the training phase, this is to prevent overtraining. In the experiments Dropout was used with a probability of 50%.

4.4.2 Models considered

To solve this problem a variety of models was used. The considered neural network architecture was MLP, CNN, and RNN. Since there are two classes the baseline is 50%, which is just to either predict everything as one class or choose by random. The network sizes and structure were varied in a large spectrum of complexity with different hyperparameter settings. The network's number of hidden nodes in each layer was chosen as integers as result from the formula 2^N, where N is any positive integer. An overview of all networks considered is shown in Table 3, 4, and 5. Since there were three different feature engineering setups, the tables (3, 4, and 5) show one-third of the final set of created networks.

Feed forward deep neural network structures
Seven different networks with a feed-forward structure were created (see Table **??**) with increased complexity and depth. From a simple one layered network with 16 nodes to a deep network structure with seven hidden layers with a first hidden layer of 512 nodes ranging down to 8 nodes before the output layer. Each of the seven different created networks had two twin networks created, one with Batch Normalization between each layer, the last one with both Batch Normalization and Dropout. This produced 21 trained networks all shown in Table **??**.

Table 6: The created multilayer perceptron networks networks

Type	**Number**	**Depth**	**No. of hidden nodes in layer** *(d=dropout, bn=batch normalization)*						
			1	**2**	**3**	**4**	**5**	**6**	**7**
MLP	#1	1	16						
MLP	#1	1	16dbn						
MLP	#2	1	64						
MLP	#2	1	64dbn						
MLP	#3	2	64	16					
MLP	#3	2	64dbn	16bn					
MLP	#3	2	64bn	16bn					
MLP	#4	3	64	32	16				
MLP	#4	3	64dbn	32bn	16bn	16			
MLP	#4	3	64bn	32bn	16bn	16bn			
MLP	#5	4	64	32	16	8			
MLP	#5	4	64dbn	32bn	16bn	8			
MLP	#5	4	64bn	32bn	16bn	8bn			
MLP	#6	5	128	64	32	16	8		
MLP	#6	5	128dbn	64bn	32dbn	16	8		
MLP	#6	5	128bn	64bn	32bn	16bn	8bn		
MLP	#7	6	256	128	64	32	16	8	
MLP	#7	6	256dbn	128bn	64dbn	32bn	16	8	
MLP	#7	6	256bn	128bn	64bn	32bn	16bn	8bn	
MLP	#8	7	512	256	128	64	32	16	8
MLP	#8	7	512dbn	256bn	128dbn	64bn	32dbn	16	8

Convolutional neural network structures

Five different networks with the convolutional structure were created (see Table 6) with increased complexity and depth. Ranging from a simple two-layered network to an 10-layer-deep network. The first half of the layers of each network are convoluted feature map layers and the second half is fully connected layers. Each network (except number #5) had two networks with identical node structure created, but with an added regularization methods (Dropout & Batch Normalization). This produced 12 trained networks all shown in Table 6.

Table 7: The created convolutional neural networks

Type	**Number**	**Depth**	**No. hidden nodes in layer** *(d=dropout, bn=batch normalization, c=convolutional, p=max pooling)*									
			1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
CNN	#1	2	c16	16								
CNN	#1	2	c16bnpd	16bn								
CNN	#1	4	c16bn	16bn								
CNN	#2	4	c16	c32	32	16						
CNN	#2	4	c16bn	c32bnpd	32bn	16bn						
CNN	#2	4	c16bn	c32bnp	32bn	16bn						
CNN	#3	6	c16	c32	c64	64	32	16				
CNN	#3	6	c16bn	c32bnd	c64bnpd	64bn	32bn	16bn				
CNN	#3	6	c16bn	c32bn	c64bnp	64bn	32bn	16bn				
CNN	#4	8	c16	c32	c64	c128	128	64	32	16		
CNN	#4	8	c16bn	c32bnd	c64bn	c128bndp	128bn	64bn	32bn	16bn		
CNN	#4	8	c16bn	c32bn	c64bn	c128bnp	128bn	64bn	32bn	16bn		
CNN	#5	10	c16	c32	c64	c128	c256	256	128	64	32	16

Recurrent deep neural network structures

Three different networks with the recurrent structure were created (see Table 8) with increased complexity and depth. Ranging from a simple two-layered network to a 6-layer-deep network with LSTM and time disturbed layers. The first half of the layers of each network are LSTM layers and the second half is time disturbed fully connected layers. Time disturbed layers needed because we are using the RNN structure. They are used to keep a one-to-one relationship between the input and output with recurring connection from previos layers. Each network had two networks with identical node structure created, but with added regularization methods (Dropout & Batch Normalization). This produced 9 trained networks all shown in Table 8.

Table 8: The created recurrent neural networks

Type	**Number**	**Depth**	**Number of hidden nodes in layer** (lstm=long short term memory, d=dropout, bn=batch normalization, td=time distrubuted)					
Layers			**1**	**2**	**3**	**4**	**5**	**6**
RNN	#1	2	lstm64	td16				
RNN	#1	2	lstm64b	td32				
RNN	#1	2	lstm64bn	td16bn				
RNN	#2	4	lstm64	lstm16	td32	td16		
RNN	#2	4	lstm64bn	lstm16bn	td32dbn	td16bn		
RNN	#2	4	lstm64bn	lstm16bn	td32bn	td16bn		
RNN	#3	6	lstm128	lstm64	lstm16	td64	td32	td16
RNN	#3	6	lstm128bn	lstm64bn	lstm16bn	td64dbn	td32bn	td16bn
RNN	#3	6	lstm128bn	lstm64bn	lstm16bn	td64bn	td32bn	td16bn

5 Results

Six tables were created to overview the final results as raw data (seen in Appendix as tables 9 through 14). All created networks (MLP, CNN, and RNN) are shown, and each table contains one of the three feature extraction methods based on the window sizes used (100 ms, 1000 ms, and 2500 ms). Each row corresponds to one created network and its relevant details. The columns in the matrix represent details about the corresponding network. A brief description of each column is found below this paragraph. Some of the columns in the matrix use background colors. These column colors are a scale from low (as red) to a high value (as green), compared to other values that are surrounded within a distinct black box. Some model rows are marked with a color to highlight that model, for smoother references.

- *Row*. Integer. An identifier for a unique reference to a single network.
- *Architecture*. MLP, CNN, or RNN. This is the used neural network architecture for this model.
- *Depth*. Integer. This number summarizes how deep the network structure is, excluding the input and output layer.
- *Window length*. Float. The window length used in the audio segment for the feature extraction method as seconds.
- *Summary of layers*. A summary of each layer in each model.
  Each layer in the network is contained between each /. The first value represents the input size as 2D matrix. All other integers are number of nodes in that layer. The letter abbreviations correspond to what type of layer it is. (*d* means that dropout was used after this layer. *c* means that this layer is a convolutional layer. *st* means that this layer is a long short term memory layer. *td* means that this layer is time distributed.) After all the layers visible in this summary there is an output layer that consists of two nodes which is not included. This layer represents the possible choices for the model to classify (Snore and Non-Snore).
- *Total training time*. Float. This is the total time it took for the model to train in minutes. Including the five epochs after early stopped triggered.
- *Training time divided by the total number of epochs*. Float. This is the mean training time for each epoch. This measurement is used to evaluate models against each other if their epochs diverge too much.
- *Dropout*. Yes or No. Answers if Dropout was used or not in this model.
- *Batch normalization*. Yes or No. Answers if Batch Normalization was used or not in this model.

- *Trainable parameters*. Integer. The number of independent variables (weights) that are adjusted each epoch.
- *Epoch, early stop triggered*. Integer. The epoch when the validation loss peaked and early stopping triggered and halted the experiment.
- *Accuracy - Train*. Percentage. The final accuracy calculated on the training set when the training halted.
- *Accuracy - Validation*. Percentage. The final accuracy calculated on the validation set when the training halted.
- *Accuracy - Test - by sample*. Percentage. This is the total accuracy calculated for all samples in all files in the test set.
- *Accuracy - Test - by file*. Percentage. This is the mean accuracy calculated for all samples in each file for all files in the test set.
- *Model size* Integer as kilobytes. This is the saved models final exported file size for future usage.

A total of 129 created neural networks were created. Each of them represented as a row in the tables 9 through 14. The six tables consists of three different subsets of 43 similar networks, with the same structure but built with different feature extractions setups. The networks depths varied from three to thirteen layers with the least complex one of a size of 30 kilobytes to the largest of 576 582 kilobytes. The smallest network had 514 trainable parameters and the largest network had almost 50 million trainable parameters. The training time varied from the trivial network of a few seconds per epoch to roughly 32 hours to train over 16 epochs which are approximately two hours per epoch. This most complex network was the deepest CNN network with a 2.5-second window length. This is the largest model created and its performance was not beyond average, ranking at 45 of 129 (by file) and 6 of 129 (by sample). The network with the highest prediction accuracy by sample was network #105 (shown as a red row in table 13) with an accuracy of 93.038%. The network with the highest prediction accuracy by file had a 97% accuracy, this was network #33 (shown as a blue in table 10).

To evaluate the architectures against each other these sequence of actions were made to create a new table; choose all models created by each window size one at the time, sort them by accuracy on all predicted samples created by the test files in descending order, and select the top 10 models at with the highest percentage results. These steps produced the ten best performing models within all three window lengths. Performance was compared by accuracy sample-wise.

To evaluate the three different window lengths (MFCC extraction setups) against each other two different sequence of actions were made; A first table was created by selecting a subset of the created models via these steps; select all models created by all window sizes, sort them by accuracy on all predicted test files in descending order, and select the top 20 models at with

the highest percentage results. The best model with 100 ms window and these preferences was #33 (shown as a blue in table 10) with an accuracy of 90.450% (by sample) and 97.000% (by file) which is also the best performing model of all.

The second table was produced by the same selection steps but instead of sorting the models by accuracy on all predicted test files the models were sorted by accuracy on all predicted samples. The best model is #105 (shown as a red in table 13) that has an accuracy of 93.038% (by sample) and 93.038% (by file). This model uses a window length of 2500ms.

Overfitting tendencies
An interesting aspect of the results is to estimate the overfitting of the various models. This could be done in several ways. One way was to view the validation accuracy when the training halted and compare it to the accuracy of each sample. This metric is shown as the column *Diff between Val Acc and By Sample Acc* in the result tables (9 through 14). Another way to detect overtraining can be too examine the number of iterations it took for the network to converge to a solution, that can be seen in the number of epochs used in the training phase before early stop triggered.

5.1 The model with the best potential

The overall best performing network created was network #29 (shown as a purple in table 10). Network #29's structure is shown in figure 14. The network has a total of 7 hidden layers where three are convolutional layers, one max pooling layer, and three fully connected layers. It has a total of 125,202 trainable parameters and uses both Dropout (twice) and Batch normalization (after each layer) as regularization methods with the size of 1555 kB.

Network #29's training phase went on for 17 epochs before early stopping triggered, putting its accuracy performance peak at epoch #12. An overview of the training phase and its epochs are shown in figure 15, and in figure 16 the validation loss is shown.

After network 29 was created it was evaluated on the independent test set that consists of 12016 samples within 800 files the network managed to predict correctly 17275 of 19015 samples which corresponds to 90.85% prediction accuracy by sample. When using these sample predictions within each file to calculate an accuracy file-wise it correctly predicts 771 of 800 files, which is 96.38%. Confusion matrices for network 29 are shown in figures 17, 18, 19, and 20).

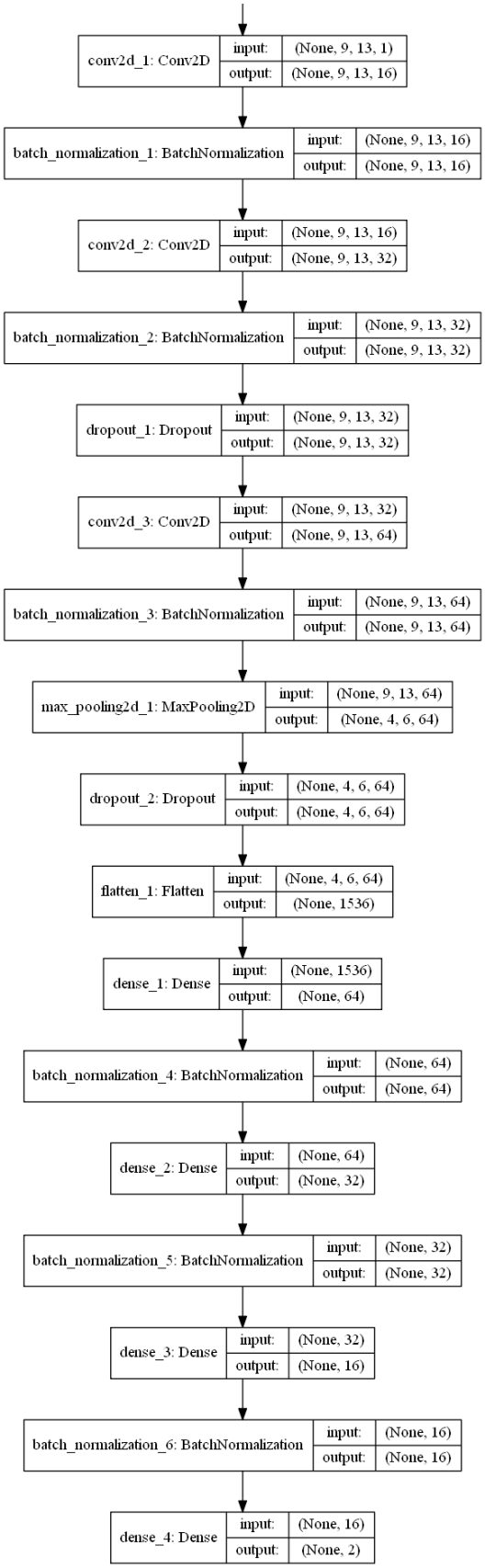

Figure 14: A detailed structure of network #29

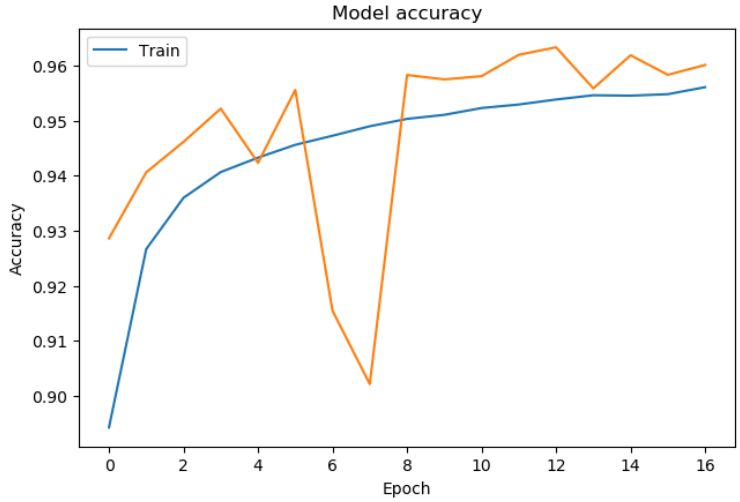

Figure 15: The accuracy on the training and validation data performance during the training phase of network #29, over no. epochs

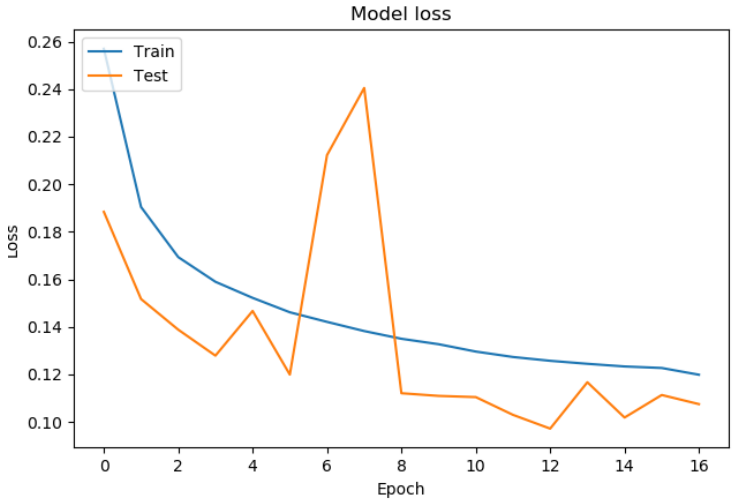

Figure 16: A visualization of the calculated loss on the training and validation data performance during the training phase of network #29, over no. epochs

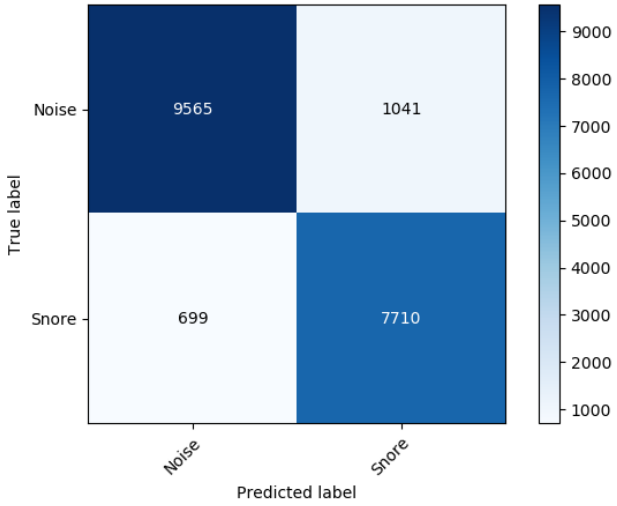

Figure 17: A confusion matrix for the network #29, showing all samples predicted in test set

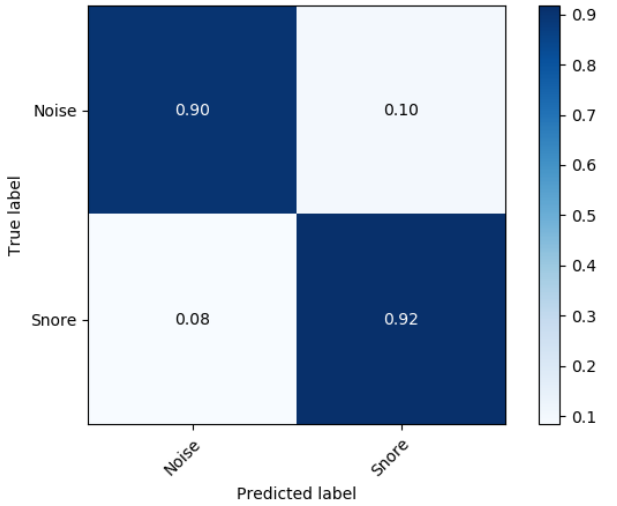

Figure 18: A normalized confusion matrix for the network #29, showing all samples predicted in test set

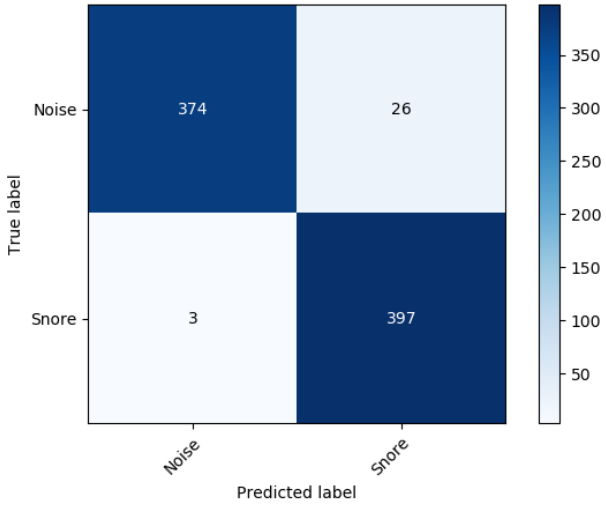

Figure 19: A confusion matrix for the network #29, showing all files predicted in test set

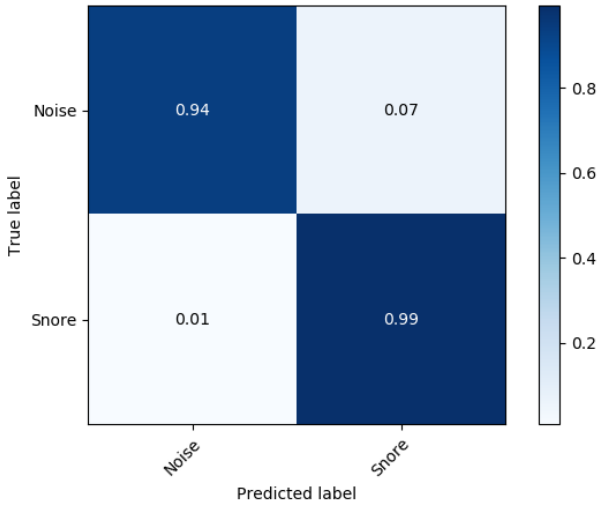

Figure 20: A normalized confusion matrix for the network #29, showing all files predicted in test set

6 Related work

Automatic detection of whole night snoring events using a microphone [9]
This was a study that recorded sounds during a polysomnography (PSG) session using a microphone on a one-meter distance. Forty-two subjects generated over 76600 acoustic episodes that were classified into snore and non-snore episodes. The average classification accuracy was 98.4% based on a ten-fold cross-validation technique. The signal used was a 44.k1 (PCM) kHz down-sampled to 16kHz and 16 bits per sample. For feature selection, a 10-fold cross-validation method was used to reduce feature complexity. This research is similar to this thesis but with the main difference that a AdaBoost classifier was used for the final prediction. There are some other differences with this research, that would not suit our desired models purpose and preferences such as:

- The model will probably have a hard time classifying external random environmental sounds that the model has never heard.
- The same high-quality microphone used in all acoustic episodes samples in the training data. In contrast to using the model by a large range of different devices that all will generate different sound artifacts.
- The microphone was always placed in the same place with the same acoustic, one meter above the subject. In an uncontrolled environment, common people all have different recording setups. The recording device will be placed somewhere around the bed in different rooms with different acoustics.

Recurrent Neural Network for Classification of Snoring and Non-Snoring Sound Events [4]
This is a study made in 2018 to classify snore and non-snore events. The study had 20 subjects referred to clinical sleep recording 70 cm from the top end of the bed. Features were extracted from the samples using the Mel-frequency cepstral coefficient technique. The features were fed through a Recurrent Neural Network and the proposed method had an accuracy of 95%. This study uses the same feature extraction and the same algorithms but suffers from the clinical problem discussed in the last section. The controlled lab environment enhances one of the found problems, too similar data in the dataset. Contrary to popular belief noise is not always a bad thing.

Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection [7]
This is a study made in 2018 to classify multiple overlapping sound events any given time instance. A CRNN (a combination of CNNs and RNNs) was created and compared to regular CNNs/RNNs/MLP and GMM baselines on four different datasets containing everyday sound events. The convolutional neural network was used to extract higher level features and the recurrent neural network used for learning the long term temporal context in the audio signals. Features were extracted from the samples using the Mel-frequency cepstral coefficient technique.

This approach to handling sounds is more natural and reasonable since sounds in the real world might overlap. In this thesis, the events are treated like they are mutually exclusive as a delimitation. The same pre-processing technique is with similar hyperparameters, and the window length of 100ms is used.

7 Discussion

Answers to the problem statement questions introduced in section 2.2 are presented one by one.

7.0.1 Which neural network architectures seem to be favorable for this sound event classification problem?

the CNN structure seems to outperform both MLP and RNN under most circumstances. The results indicate that the superior network architecture for this specific problem is the convolutional neural network. There is a common theme in all three feature extraction contexts by comparing sample accuracy. This is concluded by sorting the different networks window sizes by sample accuracy. In the 100ms window size category, 9 of 12 networks are CNN. In the 100ms window size category, 9 of 12 networks are CNN. In the 100ms window size category, 7 of 12 networks are CNN. From these results I draw the conclusion that CNN is dominating the other architectures. This probably means that there are visual patterns to be found and exploited for performance in the MFCCs generated from the audio files. This is CNN's specialty.

7.0.2 The current implemented pre-processing technique uses a three-second sound segment length for generating input features. Is it possible to shorten this window and still maintain a reasonable performance by the created networks?

Shorter window is superior in accuracy performance compared to a longer window. This is true if an ensemble of models is used and the results are produced by multiple predictions over the audio file.

The results indicate that if the models are compared by only predicting once per file (sample-wise) there are advantages to use a longer window. Since the best and the majority of the best models sorted by accuracy sample-wise are the models with a longer window. This conclusion is drawn from observing the top 10 networks with the best sample accuracy independent of window length. Only 1 of 10 networks uses 100ms, 4 of 10 uses 1000ms, and 5 of 10 uses 2500ms. In the top 20 networks sorted by best sample accuracy, the 100ms window size networks still manages to claim six of the placements. This is remarkable since the information contained in that small window is a fraction (117/1287= 9.1% and 117/3237= 3.6%) of the other window sizes. The best model with 100 ms window is #28 (shown as a green row in table 10) with an accuracy of 91.054% (by sample) and 96.875% (by file).

One major advantage of the smaller window sizes is that as the window shrinks, the possibility to do multiple predictions per file opens up. When multiple predictions are done by creating an ensemble of models by sliding over the file as many times as possible (file length divided by window length) without overlap. The mean of all created predictions is calculated and used as the final prediction. The performance of this ensemble of simpler models is on par with and often superior to the longer window models. This performance is summarized in the accuracy by file column. Of the top 20 networks with the highest accuracy file-wise, 19 of them uses

a window length of 100 ms. This indicates that a shorter window is superior to produce the highest accuracy by file.

7.0.3 What network complexity seems reasonable to sufficiently handle this sound event classification problem? How does that change as the window length increases?

A CNN architecture with a 2500 ms window created a network with roughly 3 to 4 million trainable parameters. The same architecture with a 100 ms window length produced 150,000 to 500,000 trainable parameters. Both these solutions were created with a structure of 6 to 8 hidden layers in the neural networks. Accuracy sample-wise was used to evaluate what network complexity that was preferred for better results. For MFCCs created with a window length of 2500ms, a CNN seemed to require approximately 6 to 8 hidden layers to produce optimal results. This network structure created a network with around 3-4 million trainable parameters which are (if exported to file) around 40 Megabytes large. This is based on the best performing networks (top 10) with the highest accuracy by sample and a window length of 2500 ms. The majority (4/7) of the networks of those CNN networks has this depth.

For a window with length 100 ms the CNN structure requires approximately 6 to 8 hidden layers. This corresponds to around 150,000 to 500,000 trainable parameters which are (if exported to file) around 6 Megabyte. It seems counter-intuitive why the same number of hidden layers would produce a less complex network, that answer is found in the window size. Since a smaller window size produces a smaller input to the network which yields significant fewer parameters to be calculated. This is based on the top 10 created networks with the highest accuracy by sample. where the majority (6/9) of the CNN networks has this depth.

7.0.4 What potential suggestions could be purposed to further develop Urbandroids current snore-detection implementation?

Use same architecture, structure, and depth. Try a shorter window length feature extraction setup. To keep the same architecture (CNN) and depth (6-8 hidden layers) of the network, might be a good idea. To reduce some of its complexity (remove 1-2 layers), use a shorter window size combined with a sliding mean calculation over the sound event. This to reduce the file size of the exported model, which has to be shipped with the application. Use both Dropout and Batch normalization as regularization methods.

7.0.5 Present the best solution

The proposed best model was #29 (shown as a purple row in table 10). To find and crown the best network is a difficult task, there are many aspects that have to be considered. Good network performance is characterized as a network with strong generalization performance on unseen data. In this context, another important factor is the size of the model which is a direct relation to the complexity of the model, since it will be run on a mobile device with limited resources (CPU-speed and APK-size). To find this best network a thorough analysis has to be done on the results to find tendencies that indicate what hyperparameters and techniques this

network consists of. This is done in this thesis by looking at the independent hyperparameters and techniques one by one and an attempt is made to analyze and find what indicates good results to this particular problem.

The first variable to consider is what architecture to use, CNN was concluded to to be favorable. The second variable is the window length used in the feature extraction process when the MFCCs are generated. It was concluded that a smaller window would be preferable since it would generate a smaller model with better performance. Therefore a window of 100ms is preferred. Another advantage of a window of 100ms is that the trainable parameters are kept to a minimum, that leads to a smaller exported model size. When choosing preferable complexity for the model structure it was concluded that the extremes did not yield the best results. The smallest trivial models and the largest most complex models are not preferred. The model has to be sufficiently complex to capture the patterns in the data and not to complex that it will overfit on the data. Therefore a model with a complexity somewhere in the middle, that would be approximately 6 to 8 hidden layers, is recommended. The results indicate that regularization methods help in this problem by generating more robust models. They seemed to prevent overtraining and speeding up the training process. The number of epochs can be valuable to consider in the screening process, the value should not be too high (over 20).

The main parameter of how good the network generalizes is to look at how well the model accuracy performance on the independent test set is. Therefore the file-wise accuracy is the best predictor. The sample-wise accuracy is also important to consider when the accuracy is evaluated. Considering these mentioned factors the network crowned as the winner is network number #29. The network is described in detailed in section 5.1. There were several top-performing networks that could've been chosen as the winner, but ultimately #29 seemed to be the best.

Controlled vs Uncontrolled environment
Another important aspect of this thesis is the usage of an uncontrolled environment. Many other similar studies of the same problem uses controlled lab environments. A controlled environment means that research used a fixed setup where only the person is the independent variable that changes. A fixed setup could mean the same recording room for the same sound acoustics. A room with no or minimal background noise. The same recording device for each recording which means the same noise artifacts. This recording device could be a highly advanced microphone without or minimal noise artifacts. The device is often located on the exact same position, a common approach is 1 meter straight above the subject. In this thesis, an artificial dataset is created with a variety of recording setups and persons.

Training Time The training of all the neural networks was made on a CPU (Intel(R) Core(TM) i5-3450 CPU @ 3.10GHz, 3101 MHz with 4 cores and 4 logical processors). The training time varied from lowest around 5.0 minutes per epoch to the highest 178.1 minutes per epoch. Performance could've probably been gained if a GPU was used and handled the operations, sadly there was not a supported GPU available. The official Keras library does not support AMD Radeon graphics cards at the moment.

7.1 Limitations and Weaknesses

The window length trade-off creates sample imbalance
Since the window frame is 10ms, the smallest possible window length in the feature extraction is 10ms. Each additional 10ms segment of window length will generate another column of coefficients into the feature matrix. This means that as the window length increases as the feature matrix grows. Since the training data is limited, a shorter window length will yield in more training samples. This creates a trade-off. The same amount of training data will generate more independent samples with a shorter window length but as the window length decreases each sample will contain less information that is supposed to be detected by the generated model. Since the quantity of data is a huge factor for generating robust models, more data is better. This might be a problem in this thesis since the simpler models get a huge data imbalance advantage since a 25 times shorter window will yield 25 times more data. That could be a reason for the great performance by the shorter window models. A fair approach might have been to let each window length have that factor normalized so all window length had the same number of samples.

Dataset
The dataset assembled consisted of two sources with different strengths and weaknesses. The dataset was created in an attempt to mitigate the drawbacks from both and a try to bring out the best form of both to create a rich and prosperous dataset.

Overall the dataset might have been to easy for the networks to learn. An indication of this phenomenon is when the networks converge close to 100% accuracy in the training phase. An effort could have been made to find more data or make the data more hard to learn with some kind of noise injection. Both approaches would probably have generated more robust models.

The Urbandroid dataset was collected from Sleep as Android users. This part of the dataset are the most reliable sounds for this project. The sound files were recorded on mobile devices that run the Operating system Android, the same as the final implemented model will be run on. This ensures that the same recording device was used. Many of these supplied recordings were also reported misclassifications by users. Therefore these recordings are extra valuable for creating a new better model. The Google Audio dataset consists of labeled Youtube videos. Each supplied data point was only a link to a video and time references as a range of where the labeled sound occurs. The major advantages of this dataset were: the large number of classes and samples, the wide variety of different people in different environments, different recording devices, and various recording setups.

The major weaknesses of this dataset were:

- Huge spectrum of sound profiles in the recordings.
- Some recordings were compressed with various lossy compression algorithms.
- Non-normalized sound levels between files.

- Time cost to convert data points to raw audio files. First, download video and convert to audio file.
- Time cost to find and extract the wanted sound from the 10-second recording.
- A large part of the recordings were unusable, consisted of unusable data such as wrong labels, to a low volume to be used.

Audio file length
A decision was made to have all training files as a length between 1 and 5 seconds. This was primarily made to have one snore event per file. Then to create a somewhat equal distributed dataset the same amount of noise-files was created by splitting long noise files into 1 to 5 seconds as the Snore files. This was probably not needed and could presumably be solved some other way with accompanying advantages.

Number of classes
A decision to create a binary classifier was made. In retrospective, more classes to identify could've been better for the models. This because of the tight range of accuracy between the best and worst models. With more classes, the room of improvement potential could have been larger as many training accuracies converged to 100%.

Data split
A three-way split was chosen for the dataset. 80% training data 10% validation data, and 10% test data. In hindsight, a more favorable setup could have been to focus more on the test set by creating a larger more extensive dataset for testing. This purpose would be to really confirm the generalization performance and the validity of the generated results of all created models. In these produced results many models had great accuracy with a narrow range. This change might have broadened the absolute values of results and minimize the randomness involved.

Testing and Prediction
Since the length of the files in the whole dataset was one to five seconds long, the independent test got smaller (details in table 4) when a window of 2500 ms was used. It is not possible to predict anything on files shorter in duration than the minimum length, so all files less than 2.5 seconds files were discarded. This is a weakness of this thesis and can obviously affect the end results since the smaller set of files decreases the robustness of the results and increases the random element. This could be mitigated by using files only 2500ms and longer in the dataset.

Another problem not properly dealt with in this thesis is the fact that there is a difference between a models performance on digital recordings as audio files stored on a computer, versus a live setting where a microphone records live analog data. In the short term sound spectrum, it differs heavily from what you get from a microphone, and this could be crucial for performance in a live setting. This problem could've been mitigated by creating a small proof of concept application where the created model could be implemented and run, with the purpose to evaluate its performance once more in a production environment.

Pre-processing and Feature extraction
MFCC setup The window lengths chosen was 100ms 1000 ms and 2500 ms. Since the models converged in the high 90s even a smaller window length would've been interesting to explore. When does the accuracy really start to decline? How far is it possible to push it?

8 Future work

The potential to expand this work is possible in many ways. There are many steps along the process from an event in a digital audio file to a created final prediction model, and some of the steps have endless combinations of hyperparameters to tweak. This means that there is always some area to further explore and dig deeper into. E.g a more extensive exploration of what hidden node structure of the networks is optimal. In this thesis, the network layers were contained to the power of two. What is the optimal structure for the networks with the most potential? And in what order, since there are many layers in these deep networks these independent variables create many possible combinations.

Expand classes
A great way to further mitigate the randomness problem with a binary classifier is to extend the model by expanding the number of classes to consider and identify. This would also open up the features if used in a sleep application. Sound events that would be relevant in this context such as:

- Coughing and sneezing. Identify users who coughing/sneezing, to monitor users health during sleep and log the events. This could be used to identifying tendencies over time and find a potential correlation between users health and the overall sleep quality or other parameters.
- Baby sounds such as screams and talking. This could be useful to identify if present babies are not sleeping. This could be used to log baby sleep statistics and find potential correlations and patterns. It is also a useful feature for the parents to know how baby behaves during the night.
- Other environmental sounds

More neural network structures
One great extension to this thesis is to investigate the potential of other neural network structures. E.g. a Gated Recurrent Unit (GRU)[13] structure or a Convolutional Recurrent Neural Network (CRNN)[34] structure. GRU is a type of LSTM network with a simpler structure. It was introduced in 2014 and seems to have great potential. CRNN is a combined CNN and RNN, where the CNN would be used to find local patterns in a visual domain and extract features. The RNN would then be used on top of the extracted features for the final prediction.

Dataset
Since more data is preferable within this domain of machine learning, a larger dataset with more width and depth would probably yield better results. The width and depth would be in the form of more data samples of even different people snoring and a greater range of possible environmental sounds.

Alternative pre-processing methods
Another way to extract features from the audio files, one example would be to use openSMILE[16]. SMILE is an acronym for speech and music interpretation by large-space extraction. An audio analysis process for extracting a 6573-dimensional feature set from audio segments. This would generate a larger input to the classifier which means that there is more information to find patterns within.

Extensive Window size comparison
A more extensive window size comparison, that would answer questions like: Where is the sweet-spot when information in each sample starts to decline? How near 10ms is it possible to still get a good reasonable accuracy? When do MLP networks start to catch up on CNN performance wise?

9 Conclusion

This report presents a broad spectrum of models to a subset of all possible solutions to this huge domain of parameters. The solutions were all trained to solve a particular sound problem of recognizing snores within audio data. Models were created as neural networks of three different architectures, with a large range of complexity. These networks based on features extracted from three different MFCC setups, using three different audio window lengths. The models created were compared against each other to find tendencies and potential improvements for Urbandroids current implementation. The current implementation uses the common pre-processing technique extracting the MFCCs, with an audio window length of 3 seconds fed into a CNN.

Sound classification in digital audio is a difficult task to handle, especially when the detection is supposed to work in uncontrolled and noisy environments. In uncontrolled environments such as using different recording equipment at different distances and classifying sounds created by different people, it becomes much harder.

A common approach to sound classification (current state-of-the-art in speech recognition) is to use MFCC as features from a 3-second window of audio fed into a CNN to achieve great performance. This is a broad solution which introduces countless independent parameters in the different stages of the process to tweak, such as.
- Endless representation combinations in the sound profiles, that describes how the analog sound is represented as a digital file.
- Hyperparameter choices used in the different steps of the feature extraction process in the MFCC creation.
- Choices concerning Neural network architectures, their structures, and hyperparameters.

The conclusion of these experiments indicate which neural network architectures seems to be desirable for this task and what complexity they need to have. Another important indication found is how the audio window length in the feature extraction process affects the results. The most important finding was that a shortening of the audio window length of the MFCC extraction and reducing the complexity of the network may lead to great advantages such as smaller exportable model size, a faster classification estimate created by the model, and it yields a less computationally expensive model. In conclusion a smaller more quick and agile model with great performance.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[2] W. K. Al-Delaimy, J. E. Manson, W. C. Willett, M. J. Stampfer, and F. B. Hu, "Snoring as a risk factor for type ii diabetes mellitus: a prospective study," *American journal of epidemiology*, vol. 155, no. 5, pp. 387–393, 2002.

[3] Aphex34, "Typical cnn," 2015, [Online; accessed Mars 27, 2019]. [Online]. Available: https://commons.wikimedia.org/wiki/File:Typical_cnn.png

[4] B. Arsenali, J. van Dijk, O. Ouweltjes, B. den Brinker, D. Pevernagie, R. Krijn, M. van Gilst, and S. Overeem, "Recurrent neural network for classification of snoring and non-snoring sound events," in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2018, pp. 328–331.

[5] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math compiler in python," in *Proc. 9th Python in Science Conf*, vol. 1, 2010.

[6] C. M. Bishop, *Pattern recognition and machine learning*. Springer Science+ Business Media, 2006.

[7] E. Cakır, G. Parascandolo, T. Heittola, H. Huttunen, and T. Virtanen, "Convolutional recurrent neural networks for polyphonic sound event detection," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 6, pp. 1291–1303, 2017.

[8] F. Chollet *et al.*, "Keras: The python deep learning library," *Astrophysics Source Code Library*, 2018.

[9] E. Dafna, A. Tarasiuk, and Y. Zigel, "Automatic detection of whole night snoring events using non-contact microphone," *PloS one*, vol. 8, no. 12, p. e84139, 2013.

[10] N. Dave, "Feature extraction methods lpc, plp and mfcc in speech recognition," *International journal for advance research in engineering and technology*, vol. 1, no. 6, pp. 1–4, 2013.

[11] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE transactions on acoustics, speech, and signal processing*, vol. 28, no. 4, pp. 357–366, 1980.

[12] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, no. 1, pp. 19–67, 2005.

[13] R. Dey and F. M. Salemt, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 2017, pp. 1597–1600.

[14] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[15] M. E. Dyken and K. B. Im, "Obstructive sleep apnea and stroke," *Chest*, vol. 136, no. 6, pp. 1668–1677, 2009.

[16] F. Eyben, M. Wöllmer, and B. Schuller, "Opensmile: the munich versatile and fast open-source audio feature extractor," in *Proceedings of the 18th ACM international conference on Multimedia*. ACM, 2010, pp. 1459–1462.

[17] A. S. Gami, D. O. Hodge, R. M. Herges, E. J. Olson, J. Nykodym, T. Kara, and V. K. Somers, "Obstructive sleep apnea, obesity, and the risk of incident atrial fibrillation," *Journal of the American College of Cardiology*, vol. 49, no. 5, pp. 565–571, 2007.

[18] N. Gandhewar and R. Sheikh, "Google android: An emerging software platform for mobile devices," *International Journal on Computer Science and Engineering*, vol. 1, no. 1, pp. 12–17, 2010.

[19] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio set: An ontology and human-labeled dataset for audio events," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 776–780.

[20] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[21] R. Grunstein, I. Wilcox, T.-S. Yang, Y. Gould, and J. Hedner, "Snoring and sleep apnoea in men: association with central obesity and hypertension." *International journal of obesity and related metabolic disorders: journal of the International Association for the Study of Obesity*, vol. 17, no. 9, pp. 533–540, 1993.

[22] C. Guilleminault, A. Tilkian, and W. C. Dement, "The sleep apnea syndromes," *Annual review of medicine*, vol. 27, no. 1, pp. 465–484, 1976.

[23] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, p. 947, 2000.

[24] S. Haykin, *Neural networks*. Prentice hall New York, 1994, vol. 2.

[25] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] V. Hoffstein, "Is snoring dangerous to your health?" *Sleep*, vol. 19, no. 6, pp. 506–516, 1996.

[28] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[29] V. Ibáñez, J. Silva, and O. Cauli, "A survey on sleep assessment methods," *PeerJ*, vol. 6, p. e4849, 2018.

[30] G. Inc, "Google play, googles own digital distribution service." https://play.google.com, [Online; Visited 2019-02-06].

[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[32] F. G. Issa and C. E. Sullivan, "Alcohol, snoring and sleep apnea." *Journal of Neurology, Neurosurgery & Psychiatry*, vol. 45, no. 4, pp. 353–359, 1982.

[33] P. JENNUM and A. SJØL, "Epidemiology of snoring and obstructive sleep apnoea in a danish population, age 30–60," *Journal of sleep research*, vol. 1, no. 4, pp. 240–244, 1992.

[34] C.-C. Kao, W. Wang, M. Sun, and C. Wang, "R-crnn: Region-based convolutional recurrent neural network for audio event detection," *arXiv preprint arXiv:1808.06627*, 2018.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[36] M. Knuiman, A. James, M. Divitini, and H. Bartholomew, "Longitudinal study of risk factors for habitual snoring in a general adult population: the busselton health study," *Chest*, vol. 130, no. 6, pp. 1779–1783, 2006.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[38] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[39] X. Li, S. Chen, X. Hu, and J. Yang, "Understanding the disharmony between dropout and batch normalization by variance shift," *arXiv preprint arXiv:1801.05134*, 2018.

[40] I. M. Little and J. A. Mirrlees, "Project appraisal and planning for developing countries," *New York*, 1974.

[41] B. Logan *et al.*, "Mel frequency cepstral coefficients for music modeling." in *ISMIR*, vol. 270, 2000, pp. 1–11.

[42] S. Lowel and W. Singer, "Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity," *Science*, vol. 255, no. 5041, pp. 209–212, 1992.

[43] E. Lugaresi, F. Cirignotta, G. Coccagna, and C. Piana, "Some epidemiological data on snoring and cardiocirculatory disturbances," *Sleep*, vol. 3, no. 3-4, pp. 221–224, 1980.

[44] H. N. Mhaskar and C. A. Micchelli, "How to choose an activation function," in *Advances in Neural Information Processing Systems*, 1994, pp. 319–326.

[45] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[46] P. G. Norton and E. V. Dunn, "Snoring as a risk factor for disease: an epidemiological survey." *Br Med J (Clin Res Ed)*, vol. 291, no. 6496, pp. 630–632, 1985.

[47] A. A. Ong and M. B. Gillespie, "Overview of smartphone applications for sleep analysis," *World journal of otorhinolaryngology-head and neck surgery*, vol. 2, no. 1, pp. 45–49, 2016.

[48] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 5, no. 4, pp. 293–311, 2013.

[49] D. O'shaughnessy, *Speech communication: human and machine*. Universities press, 1987.

[50] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Transactions on neural networks*, vol. 3, no. 5, pp. 683–697, 1992.

[51] M. Patil, A. Gupta, A. Varma, and S. Salil, "Audio and speech compression using dct and dwt techniques," *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 2, no. 5, pp. 1712–1719, 2013.

[52] K. J. Piczak, "The details that matter: Frequency resolution of spectrograms in acoustic scene classification," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017)*, 2017.

[53] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[54] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[55] M. Ravesloot, J. Van Maanen, L. Dun, and N. De Vries, "The undervalued potential of positional therapy in position-dependent snoring and obstructive sleep apnea—a review of the literature," *Sleep and Breathing*, vol. 17, no. 1, pp. 39–49, 2013.

[56] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[57] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[58] R. Rowell, *YouTube: Company and Its Founders: Company and Its Founders*. ABDO, 2011.

[59] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[60] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition," *Speech Communication*, vol. 54, no. 4, pp. 543–565, 2012.

[61] M. F. Sanner *et al.*, "Python: a programming language for software integration and development," *J Mol Graph Model*, vol. 17, no. 1, pp. 57–61, 1999.

[62] F. Seide and A. Agarwal, "Cntk: Microsoft's open-source deep-learning toolkit," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 2135–2135.

[63] A. V. Shelgikar, P. F. Anderson, and M. R. Stephens, "Sleep tracking, wearable technology, and opportunities for research and clinical care," *Chest*, vol. 150, no. 3, pp. 732–743, 2016.

[64] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–, Oct. 2017. [Online]. Available: http://dx.doi.org/10.1038/nature24270

[65] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[66] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.

[67] A. Team, "Alphastar: Mastering the real-time strategy game starcraft ii," 2019, [Online; accessed Mars 27, 2019]. [Online]. Available: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/

[68] U. Team, "About urbandroid." https://team.urbandroid.org/, [Online; Visited 2019-02-18].

[69] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop, coursera: Neural networks for machine learning," *University of Toronto, Technical Report*, 2012.

[70] A. M. Turing, "Computing machinery and intelligence," in *Parsing the Turing Test*. Springer, 2009, pp. 23–65.

[71] R. Vergin, D. O'shaughnessy, and A. Farhat, "Generalized mel frequency cepstral coefficients for large-vocabulary speaker-independent continuous-speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 5, pp. 525–532, 1999.

[72] D. Vise, "The google story," *Strategic Direction*, vol. 23, no. 10, 2007.

[73] A. B. Watson, "Image compression using the discrete cosine transform," *Mathematica journal*, vol. 4, no. 1, p. 81, 1994.

[74] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard University, 1974.

[75] T. Young, L. Finn, H. Kim *et al.*, "Nasal obstruction as a risk factor for sleep-disordered breathing," *Journal of Allergy and Clinical Immunology*, vol. 99, no. 2, pp. S757–S762, 1997.

[76] T. Young, M. Palta, J. Dempsey, J. Skatrud, S. Weber, and S. Badr, "The occurrence of sleep-disordered breathing among middle-aged adults," *New England Journal of Medicine*, vol. 328, no. 17, pp. 1230–1235, 1993.

[77] X. Zhang, E. L. Giovannucci, K. Wu, X. Gao, F. Hu, S. Ogino, E. S. Schernhammer, C. S. Fuchs, S. Redline, W. C. Willett *et al.*, "Associations of self-reported sleep duration and snoring with colorectal cancer risk in men and women," *Sleep*, vol. 36, no. 5, pp. 681–688, 2013.

[78] F. Zheng, G. Zhang, and Z. Song, "Comparison of different implementations of mfcc," *Journal of Computer science and Technology*, vol. 16, no. 6, pp. 582–589, 2001.

Table 9: Overview of the networks created with 100ms sec window length

Row	Structure	Depth	Summary of layers	Total Training Time (min)	Training Time/Epoch (min)	Dropout	Batch Normalization	Trainable Parameters	Epoch, early stop triggered	Accuracy (%)				Model Size (kB)	Diff between Val Acc & By Sample Acc
										Training Phase		Test Phase			
										Train	Validation	*by sample*	by file		
1	MLP	0,1	9x13/16	4,7	0,2	Yes	No	514	21	88,127	88,472	80,489	88,250	30	7,9825
2	MLP	0,1	9x13/16d	3,2	0,2	Yes	Yes	578	9	86,789	88,669	79,516	88,500	95	9,1531
3	MLP	0,1	9x13/64	6,4	0,2	No	No	2050	26	90,820	91,183	82,977	91,500	48	8,2065
4	MLP	0,1	9x13/64d	7,8	0,4	Yes	Yes	2306	13	87,721	89,429	81,015	90,000	55	8,4140
5	MLP	0,1	9x13/64/16	4,7	0,2	No	No	2226	14	92,061	92,557	86,826	95,500	54	5,7312
6	MLP	0,1	9x13/64d/16	2,7	0,3	Yes	Yes	2482	3	88,180	87,104	78,112	83,750	64	8,9915
7	MLP	0,1	9x13/64/16	2,7	0,3	No	Yes	2546	4	90,400	90,059	82,814	91,500	69	7,2454
8	MLP	0,1	9x13/64/32/16	6,2	0,3	No	No	3794	16	93,456	93,941	87,289	94,875	79	6,6519
9	MLP	0,1	9x13/64d/32d/16	10,8	0,5	Yes	Yes	4178	17	90,142	90,575	83,760	90,375	95	6,8145
10	MLP	0,1	9x13/64/32/16	4,0	0,5	No	Yes	4242	3	92,412	92,906	75,640	84,375	102	17,2660
11	MLP	0,1	9x13/64/32/16/8	19,9	0,4	No	No	3786	41	94,624	94,645	87,910	95,250	83	6,7355
12	MLP	0,1	9x13/64d/32/16d/8	7,2	0,5	Yes	Yes	4234	10	90,322	89,624	77,302	80,625	110	12,3214
13	MLP	0,1	9x13/64/32/16/8	13,7	1,2	No	Yes	4266	6	92,902	93,527	80,905	89,250	114	12,6225
14	MLP	0,1	9x13/128/64/32/16/8	10,2	0,7	No	No	12938	9	93,550	94,182	87,105	95,000	197	7,0769
15	MLP	0,1	9x13/128d/64/32d/16/8	14,2	1,1	Yes	Yes	13834	8	91,043	91,300	83,213	90,000	226	8,0872
16	MLP	0,1	9x13/128/64/32/16/8	39,1	1,9	No	Yes	13930	16	94,677	95,683	86,763	94,500	236	8,9196
17	MLP	0,1	9x13/256/128/64/32/16/8	27,0	1,3	No	No	47626	16	95,151	95,460	88,257	96,000	611	7,2037
18	MLP	0,1	9x13/256d/128/64d/32/16/8	25,6	2,6	Yes	Yes	49546	5	87,517	76,310	60,826	54,875	652	15,4845
19	MLP	0,1	9x13/256/128/64/32/16/8	22,8	1,6	No	Yes	49642	9	95,465	95,748	89,230	96,250	661	6,5180
20	MLP	0,1	9x13/512/256/128/64/32/16/8	104,5	5,0	No	No	182538	16	95,570	96,001	88,183	95,750	2 198	7,8177
21	MLP	0,1	9x13/512/256d/128/64d/32/16/8	42,2	5,3	Yes	Yes	186506	3	89,112	88,404	81,362	86,000	2 261	7,0416

Table 10: Overview of the networks created with 100ms sec window length

Row	Structure	Depth	Summary of layers	Total Training Time (min)	Training Time/Epoch (min)	Dropout	Batch Normalization	Trainable Parameters	Epoch, early stop triggered	Accuracy (%)				Model Size (kB)	Diff between Val Acc & By Sample Acc
										Training Phase		Test Phase			
										Train	Validation	*by sample*	by file		
22	CNN	0,1	9x13/c16/16	21,0	0,4	No	No	6354	42	92,071	92,159	84,733	92,750	104	7,4259
23	CNN	0,1	9x13/c16d/16	23,7	1,5	Yes	Yes	6482	11	90,176	92,134	85,338	93,375	117	6,7964
24	CNN	0,1	9x13/c16/16	21,5	1,3	No	Yes	6482	11	91,834	92,811	85,590	94,875	116	7,2203
25	CNN	0,1	9x13/c16/c32/32/16	37,8	1,5	No	No	29970	20	96,308	95,930	90,492	96,875	391	5,4380
26	CNN	0,1	9x13/c16/c32d/32/16	72,3	4,5	Yes	Yes	30354	11	93,969	95,346	88,967	93,875	424	6,3794
27	CNN	0,1	9x13/c16/c32/32/16	41,7	4,2	No	Yes	30354	5	94,387	94,679	85,501	91,875	423	9,1781
28	CNN	0,1	9x13/c16/c32/c64/64/32/16	93,5	3,6	No	No	124306	21	98,264	97,236	91,054	96,875	1 512	6,1816
29	CNN	0,1	9x13/c16/c32d/c64d/64/32/16	174,9	10,3	Yes	Yes	125202	12	95,385	96,334	90,849	96,375	1 555	5,4849
30	CNN	0,1	9x13/c16/c32/c64/64/32/16	123,7	13,7	No	Yes	125202	4	96,022	96,390	89,014	96,125	1 553	7,3759
31	CNN	0,1	9x13/c16/c32/c64/c128/128/64/32/16	336,0	14,0	No	No	501394	19	98,688	97,452	91,012	96,500	5 941	6,4399
32	CNN	0,1	9x13/c16/c32d/c64/c128d/128/64/32/16	419,8	30,0	Yes	Yes	503314	9	97,037	97,248	89,903	96,375	6 011	7,3457
33	CNN	0,1	9x13/c16/c32/c64/c128/128/64/32/16	380,0	31,7	No	Yes	503314	7	97,975	96,890	90,450	97,000	6 155	6,4405
34	CNN	0,1	9x13/c16/c32/c64/c128/c256/256/128/64/32/16	813,2	42,8	No	No	2009234	14	98,782	97,665	90,970	95,250	23 622	6,6950
35	RNN	0,1	9x13/st64/td16	13,4	0,4	No	No	21298	26	94,835	95,130	89,429	96,250	282	5,7005
36	RNN	0,1	9x13/st64/td32	12,6	0,4	No	No	22626	26	94,999	94,830	88,520	96,625	295	6,3107
37	RNN	0,1	9x13/st64/td16	5,8	0,6	No	Yes	21618	4	91,512	90,775	81,357	91,125	294	9,4186
38	RNN	0,1	9x13/st64/st16/td32/td16	15,5	0,6	No	No	26514	21	94,575	94,904	88,267	95,375	354	6,6373
39	RNN	0,1	9x13/st64d/st16/td32d/td16	11,1	1,1	Yes	Yes	27026	5	93,665	92,316	81,499	89,500	385	10,8177
40	RNN	0,1	9x13/st64/st16/td32/td16	8,0	0,7	No	Yes	27026	7	94,766	91,773	78,885	89,000	384	12,8879
41	RNN	0,1	9x13/st128/st64/st16/td64/td32/td16	33,7	1,8	No	No	131282	14	95,043	95,389	87,899	95,375	1 598	7,4902
42	RNN	0,1	9x13/st128/st64d/st16/td64/td32/td16	39,9	2,7	Yes	Yes	132562	10	97,104	95,485	86,369	94,750	1 648	9,1164
43	RNN	0,1	9x13/st128/st64/st16/td64/td32/td16	45,7	5,7	No	Yes	132562	3	93,883	91,609	72,774	83,625	1 647	18,8352

Table 11: Overview of the networks created with 1000ms window length

Row	Structure	Depth	Summary of layers	Total Training Time (min)	Training Time/Epoch (min)	Dropout	Batch Normalization	Trainable Parameters	Epoch, early stop triggered	Accuracy (%)				Model Size (kB)	Diff between Val Acc & By Sample Acc
										Training Phase		Test Phase			
										Train	Validation	*by sample*	by file		
87	MLP	1	249x13/16	2,1	0,1	Yes	No	8194	23	98,198	98,498	84,810	84,810	119	13,6884
88	MLP	1	249x13/16d	1,7	0,2	Yes	Yes	8258	5	96,310	97,838	87,342	87,342	126	10,4961
89	MLP	1	249x13/64	1,0	0,1	No	No	32770	6	97,197	98,559	77,848	77,848	408	20,7105
90	MLP	1	249x13/64d	2,8	0,1	Yes	Yes	33026	37	88,137	88,728	79,879	87,250	415	8,8486
91	MLP	2	249x13/64/16	1,3	0,2	No	No	9906	3	97,010	98,619	79,747	79,747	144	18,8718
92	MLP	2	249x13/64d/16	3,9	0,5	Yes	Yes	10162	3	97,758	98,919	84,177	84,177	154	14,7417
93	MLP	2	249x13/64/16	5,4	0,7	No	Yes	10226	3	98,812	98,378	90,506	90,506	160	7,8720
94	MLP	3	249x13/64/32/16	2,7	0,1	No	No	11474	16	99,173	99,339	89,241	89,241	168	10,0988
95	MLP	3	249x13/64d/32d/16	8,4	0,6	Yes	Yes	11858	9	98,105	97,958	74,684	74,684	186	23,2744
96	MLP	3	249x13/64/32/16	4,4	0,5	No	Yes	11922	4	98,952	99,219	80,380	80,380	193	18,8395
97	MLP	4	249x13/64/32/16/8	2,4	0,1	No	No	7626	11	98,091	98,799	79,114	79,114	128	19,6849
98	MLP	4	249x13/64d/32/16d/8	3,2	0,4	Yes	Yes	8074	4	97,317	92,973	87,342	87,342	155	5,6312
99	MLP	4	249x13/64/32/16/8	5,5	0,5	No	Yes	8106	6	99,453	99,399	78,481	78,481	160	20,9184
100	MLP	5	249x13/128/64/32/16/8	10,3	0,4	No	No	16778	22	99,520	99,520	89,241	89,241	241	10,2790
101	MLP	5	249x13/128d/64/32d/16/8	16,4	2,0	Yes	Yes	17674	3	97,244	91,351	77,848	77,848	272	13,5033
102	MLP	5	249x13/128/64/32/16/8	17,8	1,1	No	Yes	17770	11	98,699	99,279	87,975	87,975	282	11,3046
103	MLP	6	249x13/256/128/64/32/16/8	10,3	0,7	No	No	51466	9	98,418	98,799	84,810	84,810	657	13,9887
104	MLP	6	249x13/256d/128/64d/32/16/8	32,3	2,9	Yes	Yes	53386	6	98,805	98,438	84,810	84,810	697	13,6283
105	MLP	6	249x13/256/128/64/32/16/8	135,3	6,1	No	Yes	53482	17	99,826	99,880	93,038	93,038	707	6,8419
106	MLP	7	249x13/512/256/128/64/32/16/8	47,6	4,0	No	No	186378	7	98,138	98,679	89,873	89,873	2 244	8,8053
107	MLP	7	249x13/512/256d/128/64d/32/16/8	54,8	9,1	Yes	Yes	190346	1	96,016	86,306	80,380	80,380	2 304	5,9266

Table 12: Overview of the networks created with 1000ms window length

Row	Structure	Depth	Summary of layers	Total Training Time (min)	Training Time/Epoch (min)	Dropout	Batch Normalization	Trainable Parameters	Epoch, early stop triggered	Accuracy (%)				Model Size (kB)	Diff between Val Acc & By Sample Acc
										Training Phase		Test Phase			
										Train	Validation	*by sample*	by file		
65	CNN	2	99x13/c16/16	22,7	0,3	No	No	75474	69	95,993	95,983	87,135	91,875	914	8,8478
66	CNN	2	99x13/c16d/16	25,0	2,1	Yes	Yes	75602	7	95,910	96,292	89,917	91,875	927	6,3752
67	CNN	2	99x13/c16/16	22,5	1,7	No	Yes	75602	8	97,177	95,735	84,771	88,000	926	10,9650
68	CNN	4	99x13/c16/c32/32/16	40,1	1,3	No	No	306450	25	97,469	96,910	88,873	92,625	3 631	8,0363
69	CNN	4	99x13/c16/c32d/32/16	67,6	5,6	Yes	Yes	306834	7	98,726	98,486	91,029	93,500	3 664	7,4566
70	CNN	4	99x13/c16/c32/32/16	64,1	4,6	No	Yes	306834	9	99,808	97,373	90,334	93,875	3 663	7,0395
71	CNN	6	99x13/c16/c32/c64/64/32/16	64,1	4,6	No	No	306834	9	99,808	97,373	90,334	93,875	14 472	7,0395
72	CNN	6	99x13/c16/c32d/c64d/64/32/16	93,4	4,7	Yes	Yes	1230226	15	98,729	98,331	90,612	93,125	14 515	7,7193
73	CNN	6	99x13/c16/c32/c64/64/32/16	144,3	16,0	No	Yes	1231122	4	98,245	94,376	85,466	87,000	14 541	8,9098
74	CNN	8	99x13/c16/c32/c64/c128/128/64/32/16	267,1	14,8	No	No	4925074	13	99,111	98,578	90,890	93,000	57 781	7,6884
75	CNN	8	99x13/c16/c32d/c64/c128d/128/64/32/16	447,2	40,7	Yes	Yes	4926994	6	99,207	98,177	92,490	94,875	57 851	5,6872
76	CNN	8	99x13/c16/c32/c64/c128/128/64/32/16	357,9	22,4	No	Yes	4926994	11	99,701	98,795	91,933	95,500	57 849	6,8616
77	CNN	10	99x13/c16/c32/c64/c128/c256/256/128/64/32/16	810,6	47,7	No	No	19703954	12	99,176	98,517	90,682	93,375	230 982	7,8352
78	RNN	2	99x13/st64/td16	39,3	0,7	No	No	24178	49	97,304	97,373	89,082	93,500	314	8,2912
79	RNN	2	99x13/st64/td32	12,3	0,6	No	No	28386	14	94,901	95,797	85,118	89,500	363	10,6791
80	RNN	2	99x13/st64/td16	18,2	1,3	No	Yes	24498	9	96,085	94,561	83,171	86,750	329	11,3901
81	RNN	4	99x13/st64/st16/td32/td16	10,9	1,0	No	No	29394	6	92,006	93,109	81,015	84,500	388	12,0935
82	RNN	4	99x13/st64d/st16/td32d/td16	13,9	1,4	Yes	Yes	29906	5	97,531	89,308	73,783	75,625	420	15,5248
83	RNN	4	99x13/st64/st16/td32/td16	21,9	1,5	No	Yes	29906	10	98,644	91,286	77,191	79,875	419	14,0950
84	RNN	6	99x13/st128/st64/st16/td64/td32/td16	159,8	3,7	No	No	134162	38	97,967	98,331	91,864	94,000	1 631	6,4676
85	RNN	6	99x13/st128/st64d/st16/td64/td32/td16	54,6	5,0	Yes	Yes	135442	6	98,359	89,061	64,882	67,875	1 683	24,1788
86	RNN	6	99x13/st128/st64/st16/td64/td32/td16	81,8	6,8	No	Yes	135442	7	98,973	94,129	81,850	85,750	1 682	12,2788

Table 13: Overview of the networks created with 2500ms window length

Row	Structure	Depth	Summary of layers	Total Training Time (min)	Training Time/Epoch (min)	Dropout	Batch Normalization	Trainable Parameters	Epoch, early stop triggered	Accuracy (%)				Model Size (kB)	Diff between Val Acc & By Sample Acc
										Training Phase		Test Phase			
										Train	Validation	*by sample*	by file		
44	MLP	1	99x13/16	2,0	0,0	Yes	No	3394	84	93,929	94,438	82,754	87,500	62	11,6838
45	MLP	1	99x13/16d	3,1	0,1	Yes	Yes	3458	31	94,211	94,963	85,953	91,250	70	9,0102
46	MLP	1	99x13/64	4,0	0,1	No	No	13570	40	94,485	95,365	84,284	90,000	183	11,0809
47	MLP	1	99x13/64d	3,8	0,4	Yes	Yes	13826	5	93,984	94,963	81,711	86,625	190	13,2522
48	MLP	2	99x13/64/16	5,0	0,1	No	No	5106	47	95,066	95,581	84,910	90,250	88	10,6714
49	MLP	2	99x13/64d/16	10,0	0,4	Yes	Yes	5362	21	95,873	96,137	89,569	91,250	98	6,5684
50	MLP	2	99x13/64/16	3,5	0,3	No	Yes	5426	6	96,257	94,623	84,075	88,625	104	10,5479
51	MLP	3	99x13/64/32/16	5,1	0,1	No	No	6674	40	95,941	96,261	86,926	92,250	112	9,3345
52	MLP	3	99x13/64d/32d/16	4,1	0,5	Yes	Yes	7058	3	93,077	93,912	81,085	84,750	130	12,8274
53	MLP	3	99x13/64/32/16	5,9	0,5	No	Yes	7122	6	97,057	75,618	67,038	67,250	137	8,5805
54	MLP	4	99x13/64/32/16/8	5,5	0,2	No	No	5226	29	95,072	96,137	86,996	91,875	100	9,1414
55	MLP	4	99x13/64d/32/16d/8	7,0	0,7	Yes	Yes	5674	5	93,984	93,140	80,042	83,000	127	13,0980
56	MLP	4	99x13/64/32/16/8	5,0	0,4	No	Yes	5706	7	97,689	96,199	81,015	82,750	132	15,1837
57	MLP	5	99x13/128/64/32/16/8	4,7	0,3	No	No	14378	11	94,880	95,272	85,814	89,375	213	9,4583
58	MLP	5	99x13/128d/64/32d/16/8	10,8	1,1	Yes	Yes	15274	5	94,702	94,036	83,310	86,625	244	10,7257
59	MLP	5	99x13/128/64/32/16/8	9,6	0,9	No	Yes	15370	6	99,118	97,188	78,929	81,875	254	18,2588
60	MLP	6	99x13/256/128/64/32/16/8	21,8	0,6	No	No	49066	30	96,388	96,786	88,873	93,375	629	7,9127
61	MLP	6	99x13/256d/128/64d/32/16/8	19,7	2,5	Yes	Yes	50986	3	93,888	94,190	84,771	87,375	669	9,4198
62	MLP	6	99x13/256/128/64/32/16/8	27,4	2,0	No	Yes	51082	9	99,574	97,868	88,873	91,625	679	8,9943
63	MLP	7	99x13/512/256/128/64/32/16/8	26,8	1,8	No	No	183978	10	94,465	95,735	85,744	91,375	2 216	9,9914
64	MLP	7	99x13/512/256d/128/64d/32/16/8	36,8	5,3	Yes	Yes	187946	2	93,699	93,665	80,042	82,750	2 276	13,6233

Table 14: Overview of the networks created with 2500ms window length

Row	Structure	Depth	Summary of layers	Total Training Time (min)	Training Time/Epoch (min)	Dropout	Batch Normalization	Trainable Parameters	Epoch, early stop triggered	Accuracy (%)				Model Size (kB)	Diff between Val Acc & By Sample Acc
										Training Phase		Test Phase			
										Train	Validation	*by sample*	by file		
108	CNN	2	249x13/c16/16	16,4	0,4	No	No	190674	33	98,372	98,559	83,544	83,544	2 264	15,0143
109	CNN	2	249x13/c16d/16	24,4	1,5	Yes	Yes	190802	11	99,506	99,339	91,139	91,139	227	8,2001
110	CNN	2	249x13/c16/16	25,0	1,9	No	Yes	190802	8	99,219	99,219	52,532	52,532	2 276	46,6876
111	CNN	4	249x13/c16/c32/32/16	39,6	1,7	No	No	767250	18	100,000	99,880	87,342	87,342	9 031	12,5381
112	CNN	4	249x13/c16/c32d/32/16	89,5	6,9	Yes	Yes	767634	8	99,907	100,000	89,873	89,873	9 064	10,1266
113	CNN	4	249x13/c16/c32/32/16	58,1	6,5	No	Yes	767634	4	99,313	99,760	85,443	85,443	9 063	14,3167
114	CNN	6	249x13/c16/c32/c64/64/32/16	101,7	6,0	No	No	3073426	12	99,526	99,940	91,139	91,139	36 072	8,8007
115	CNN	6	249x13/c16/c32d/c64d/64/32/16	879,6	51,7	Yes	Yes	3074322	12	99,873	99,940	92,405	92,405	36 115	7,5349
116	CNN	6	249x13/c16/c32/c64/64/32/16	252,6	23,0	No	Yes	3074322	6	100,000	100,000	89,241	89,241	36 113	10,7595
117	CNN	8	249x13/c16/c32/c64/c128/128/64/32/16	186,8	12,5	No	No	12297874	10	100,000	99,880	90,506	90,506	144 181	9,3736
118	CNN	8	249x13/c16/c32d/c64/c128d/128/64/32/16	1078,1	82,9	Yes	Yes	12299794	8	99,847	99,820	86,076	86,076	144 251	13,7439
119	CNN	8	249x13/c16/c32/c64/c128/128/64/32/16	503,7	56,0	No	Yes	12299794	4	99,987	99,940	88,608	88,608	144 249	11,3323
120	CNN	10	249x13/c16/c32/c64/c128/c256/256/128/64/32/16	1911,0	119,4	No	No	49195154	11	99,987	99,880	91,772	91,772	576 582	8,1077
121	RNN	2	249x13/st64/td16	63,9	1,6	No	No	28978	35	99,426	99,820	89,241	89,241	370	10,5793
122	RNN	2	249x13/st64/td32	31,3	1,6	No	No	37986	15	98,765	99,279	86,076	86,076	475	13,2033
123	RNN	2	249x13/st64/td16	15,5	1,7	No	Yes	29298	4	99,293	96,877	62,658	62,658	385	34,2186
124	RNN	4	249x13/st64/st16/td32/td16	64,6	2,0	No	No	34194	27	99,346	99,339	87,342	87,342	444	11,9976
125	RNN	4	249x13/st64d/st16/td32d/td16	17,7	1,8	Yes	Yes	34706	5	99,786	92,793	63,924	63,924	476	28,8687
126	RNN	4	249x13/st64/st16/td32/td16	17,6	2,5	No	Yes	34706	2	99,600	98,859	81,646	81,646	476	17,2133
127	RNN	6	249x13/st128/st64/st16/td64/td32/td16	97,8	4,3	No	No	138962	18	98,879	98,679	87,342	87,342	1 688	11,3369
128	RNN	6	249x13/st128/st64d/st16/td64/td32/td16	40,8	5,8	Yes	Yes	140242	2	99,326	90,030	63,291	63,291	1 739	26,7389
129	RNN	6	249x13/st128/st64/st16/td64/td32/td16	43,9	6,3	No	Yes	140242	2	99,660	92,372	77,215	77,215	1 739	15,1572