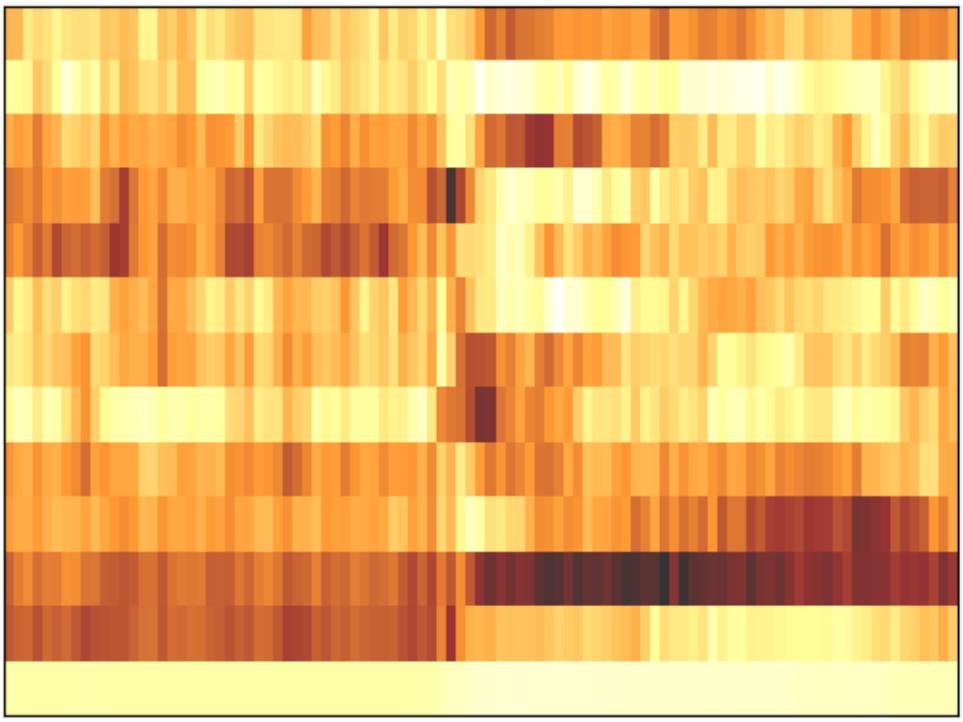

Keyword Spotting Within an Automotive Environment

A Machine Learning Approach with Analysis of Performance and Complexity

Master's thesis in Engineering Mathematics and Computational Science

NILS HAMMARÄNG GRIP

ELIAS NILSSON

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2023

www.chalmers.se

Master's thesis 2023

Keyword Spotting Within an Automotive Environment

A Machine Learning Approach with Analysis of Performance and Complexity

NILS HAMMARÄNG GRIP
ELIAS NILSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2023

Keyword Spotting Within an Automotive Environment
A Machine Learning Approach with Analysis of Performance and Complexity
NILS HAMMARÄNG GRIP
ELIAS NILSSON

Supervisor: Amir Eghbali, Aptiv
Examiner: Serik Sagitov, Mathematical sciences

Master's Thesis 2023
Department of Mathematical sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization constructed in Python of mel-frequency cepstral coefficients extracted from a one second audio segment containing an utterance of the word "down".

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Keyword Spotting Within an Automotive Environment
A Machine Learning Approach with Analysis of Performance and Complexity
NILS HAMMARÄNG GRIP
ELIAS NILSSON
Department of Mathematical sciences
Chalmers University of Technology

Abstract

The aim of this project is to implement a system to detect keywords within speech, i.e., keyword spotting (KWS), that performs well in an automotive environment. The system is based on extracting features from sound waves captured by a microphone, and machine learning (ML). The Google speech commands (GSC) dataset is used to develop the models in combination with audio book samples from the LibriSpeech (LS) dataset. The combination of these two datasets is unique and was done with the goal of increasing the robustness of the models. In addition, data augmentation and the insertion of background noise are key tools within this project, to target the system towards an automotive environment.

Aside from standard performance metrics, the complexity of the model, which will appear as a time delay for the user, is also an important aspect to enable real-time usage. Performance is examined using recorded speech and in real-world settings, both in a noise-free and a noisy automotive environment. The best-performing model was found to be a temporal convolutional residual neural network (TC-ResNet) using mel frequency cepstral coefficients (MFCC) features, which achieved an accuracy of 95.34% on the validation dataset. The model complexity is low compared to models in previous studies, with 152.7 K parameters and 3.22 M multiplications performed by the model. The model's performance is substantially lowered in an automotive environment with an average accuracy of 83.71%, but it is considered promising due to multiple possible improvements regarding capturing and filtering the speech signals by using the car's hardware instead of the laptop that was used. Due to low performance when evaluating the models on coherent speech, the suggestion is that the system should be implemented with a voice activity detection system or a "push-to-talk" button and not as a constantly ongoing process. The data collection is proposed as the main focus for future improvements, as more labeled audio segments are needed to build a more qualitative model with wider functionalities.

Keywords: Keyword spotting, machine learning, artificial neural networks, automotive environment.

Acknowledgements

To begin with, we would like to direct our thanks to our supervisor, Amir Eghbali, who has always been available for questions and helpful with all types of problems. Thanks also to our co-supervisor Emil Fahlgren for his input and guidance throughout the project. To both Amir and Emil, we appreciate all the feedback given on the multiple revisions of this report.

We would also like to thank Aptiv as a company and all coworkers within for welcoming us and making us feel at home during the thesis work. A special thanks to team manager Marcy Kirk, who has been very keen about our well-being both at Aptiv and outside the scope of our work. Our gratitude goes out to our examiner at Chalmers, Serik Sagitov, as well, for always being available if we needed it.

A thank you goes out to everyone who helped us evaluate the models by providing data in the form of speech recordings. A final thanks goes out to our families for their support, not only during this thesis work but during all of our time at Chalmers.

Nils Hammaräng Grip & Elias Nilsson, Gothenburg, May 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order.

ANN	Artificial Neural Network
ASR	Automatic Speech Recognition
Att-RNN	Attention Recurrent Neural Network
CE	Cross Entropy
CNN	Convolutional Neural Network
CRNN	Convolutional Recurrent Neural Network
DSC	Depthwise Separable Convolutional
EM	Expectation-Maximization
FC	Fully Connected
FFNN	Feed Forward Neural Network
FFT	Fast Fourier Transform
GMM	Gaussian Mixture Model
GSC	Google Speech Command
HMM	Hidden Markov Model
KDA	Keyword Detection Accuracy
KWS	Keyword Spotting
LS	LibriSpeech
LOVO	LOw Variability Orthogonal
LPC	Linear Predicative Coding
LPCC	Linear Prediction Cepstrum Coefficients
MFCC	Mel Frequency Cepstral Coefficients
MHAtt	Multi-Head Attention
MKA	Mean Keyword Accuracy
MP	Max-Pooling
ResNet	Residual Neural Network
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SNR	Signal-to-Noise Ratio
TC	Temporal Convolution
VTLP	Vocal Tract Length Perturbation

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background

The advancement of machine learning (ML) and artificial neural networks (ANN) has made voice-controlled devices ubiquitous. Especially noticeable in mobile phones and home assistants, with examples such as Apple's Siri, Amazon's Alexa, and Google's home assistant. Voice-controlled functionality is also desired in vehicles to, for example, roll up windows, set climate control, or switch radio stations. This functionality is of interest to Aptiv, a company working in the automotive industry with whom this project is being done.

A voice-controlled assistant is activated and controlled by spoken keywords. The detection of specific words within an audio stream is called keyword spotting (KWS), and is a subproblem within automatic speech recognition (ASR). The activation algorithm has historically been Hidden Markov models (HMM), but has shifted in the last decade towards artificial neural networks (ANNs) due to their superior performance [1]. However, all models rely on high-quality data, and there are resources available online, such as the Google Speech Commands (GSC) dataset, which is a standard to benchmark KWS performance [1]. However, KWS models are rarely trained on raw audio. Instead, features are extracted to provide the models with a more information dense and robust input [2].

Recently, López-Espejo et al. presented an overview of the performance of 25 ANNs on GSC and reported an accuracy of up to 98% for the best performer [1]. However, these settings were not within an automotive environment, which is the target domain for the models developed in this project. Car cabins are exposed to various noises that mask and distort the speech signal; the level of distortion depends on the car's velocity and properties. Cars traveling at ~ 110 kilometers per hour show exposure of up to 89 dB [3]. In addition to a noisy environment, at 80 dB, speech characteristics such as pitch, rate, and duration of syllables are affected [4, 5]. However, the article [3] is dated to 1981, so this project will undertake another, limited, study to evaluate car noise level in varying traffic.

Due to noise and speech alterations, the KWS model has to be robust and generalize

well. Because of this, data augmentation and the insertion of background noise are two important tools that will be implemented in this project to a greater extent than in many previous studies. At the same time, the goal is a real-time system that detects keywords as they are spoken. Therefore, the model size has to be limited, and the complexity-performance trade-off has to be analyzed. In order to do this, the two important factors that impact latency have been identified: the time to convert audio to features and, secondly, the time for the model to predict the feature's label. The complexity will be a deciding factor in the model development and final model selection.

1.2 Aim

The overall aim and goal of this project is to implement a KWS model that is sufficiently performant and responsive to be used in a real-world application within an automotive environment. To arrive at this model, these subgoals have to be achieved:

- Conduct a literature review and complementary study of the noise and sound characteristics in an automotive environment.
- Prepare data that represent speech and noise present in an automotive environment.
- Carry out a literature study of suitable methods for developing a KWS system, including ML models and feature extraction methods. Implement three promising models and feature extraction methods.
- Tune the KWS models' parameters to optimize performance.
- Conduct a complexity-performance trade-off analysis of the models, and choose a final model based on this analysis.
- Use the final model to implement a prototype KWS system and propose further improvements based on the conclusions of the project.

2

Theory

This chapter contains the theory that this project is based on, with a main focus on features and ML models which have been found relevant for KWS in the conducted literature study. In section 2.1 some basic theory of sound is presented. This is followed by a review of the methodology behind KWS in section 2.2, going through several feature extraction methods in section 2.2.2 and ML models in section 2.3, which are of interest.

2.1 Components of sound

Sound is made up of vibrations that travel through a medium. It differs from audio, a more technical term that refers to the recording and broadcasting of sound [6, 7]. However, in this project, these terms will be used interchangeably. Sound waves consist of particles, most commonly air molecules, that are compressed by an increase in pressure caused by the source of the sound [6]. Areas of compression are split up by areas of rarefaction when the pressure is decreased. The sound wave travels because kinetic energy is transferred between the particles in the medium. The intensity I of the wave decreases with the distance to the sound source d according to the inverse square law,

$$I \propto \frac{1}{d^2}.$$

There are many features of a sound wave, and some of these are displayed in figure 2.1. This figure contains a simple sinusoidal wave that can be mathematically described by the equation 2.1 [6]. Here, A is the amplitude and t is the time. The frequency f is computed as the reciprocal of the period $f = 1/T$ and is the number of cycles per second. This is measured in Hertz (Hz) and the frequency range for human hearing is around 20 Hz to 20 kHz. Frequency also correlates with pitch, where a higher frequency indicates a higher perceived pitch. The phase (ϕ) describes the displacement of the wave from the origin.

$$y(t) = A \cdot \sin(2\pi f t + \phi) \tag{2.1}$$

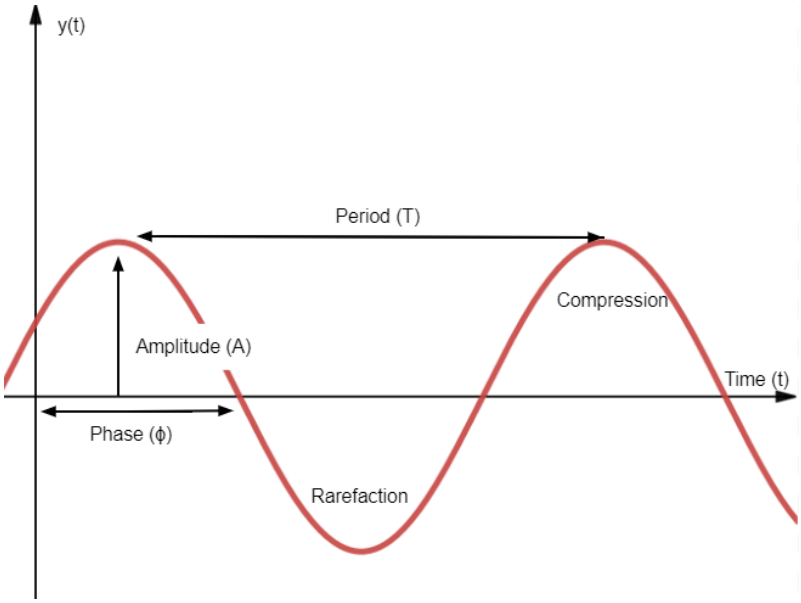

Figure 2.1: A periodic sound wave including some important features describing the wave.

The sinusoidal wave in figure 2.1 is considered periodic because of its regularly repeating character. A sum of sinusoidals can also be periodic. Periodic sounds are called tones, and these tend to have a distinguishable pitch [8]. Some examples are the sound of a spoken vowel and pure tones in music. The opposite, aperiodic sounds, are without a harmonic basis. An aperiodic sound, called noise, contains frequencies with no simple numerical relationship, and the sound wave doesn't show any distinguishable repeating pattern [8]. The term noise will be used frequently throughout this report as a sound that disturbs a desired signal, i.e., spoken words. It should be pointed out that not all sounds that will be referred to as noise will be aperiodic. Aperiodic sounds can include transient sounds, which are short single disturbance sounds, e.g., the plosive consonant sounds.

The amplitude A correlates with the loudness ("volume") of the sound and is the amount of compression and rarefaction that the particles undergo, measured in decibel (dB) [6]. A formula for computing the loudness of a sound β is shown in equation 2.2. From this, one can see that loudness is expressed as a ratio between the power of the sound P and the power of a sound at the threshold of hearing P_0 [9].

$$\beta = 10 \cdot \log_{10}\left(\frac{P}{P_0}\right) \tag{2.2}$$

A commonly used term in signal processing, such as audio processing, is signal-to-noise ratio (SNR). This is defined as the ratio between desired signal power and noise power. It is measured in dB and computed as in equation 2.2, with $P = P_{\text{signal}}$ and $P_0 = P_{\text{noise}}$ [10].

2.2 KWS

The task within this project is an example of a KWS problem. KWS is the challenge of finding one or several chosen keywords within spoken speech [1]. This is a special case within ASR, which is the problem of extracting linguistic content from audio and converting it to text. Some challenges within this area, together with tools to cope with them, are presented in section 2.2.1. This is followed by the two main parts of the KWS algorithm: feature extraction and ML modeling; see sections 2.2.2 and 2.3, respectively. A simple flow chart of KWS, which the structure of this section will follow, is shown in figure 2.2.

Figure 2.2: A flow chart describing the methodology behind KWS.

2.2.1 Data processing

Regardless of domain, the foundation of a performant ML model relies on the quality and quantity of data. ASR and KWS are subjected to several challenges in order to acquire a sufficient set of data, including problems with sparse, unreliable, and unmatched data. Methods for coping with these challenges will be described below.

2.2.1.1 Data augmentation

Data augmentation is a method that expands a set of data by introducing relatively minor perturbations that allow the main characteristics of the data to remain while still introducing variety [11]. This can be done to cope with a sparse data problem, i.e., when only a small portion of the data is labeled. Augmentation can include adding background noise, e.g., car noise, speech, or white noise, to an audio sample. This is essential to simulate the appropriate usage environment, i.e., an automotive environment in this case [1]. Moreover, one can also enable variation by changing playback speed and loudness. Another example of speech data augmentation is vocal tract length perturbation (VTLP) using mel filterbanks to map speech to new frequencies [12]. This is shown in equation 2.3, where f and f' are the new and old center frequencies, and α is a random number in the range $[0.9, 1.1]$. VTLP is a method that has been shown to improve the performance of ANNs for ASR problems [13].

$$f' = f \cdot \phi(\alpha). \tag{2.3}$$

Another facet of data augmentation is the ability to synthesize new data points using text-to-speech technology. This approach of extending the data sets has been successful in increasing model performance in a low-resource keyword setting [14]. However, diverse tones of voices are important for a performant model [15]. Thus,

combining synthetic data with techniques of perturbation could enable an extended but sufficiently varied collection of labeled data.

2.2.1.2 Data balancing

Different types of data balancing methods can be used to cope with unreliable, unbalanced, or unmatched data, which are frequent challenges in KWS [11]. Unreliable data could be of poor quality, subjected to distortions, or have missing or wrong class labels. It has been shown by Wu et al. that with a suitable data selection, a system trained on 150-hour audio data could achieve the same competitive performance as a system trained on the whole 840-hour audio data set in an ASR task [16]. Another method of data balancing is to upsample minority classes or downsample majority classes [11]. Instance-based transfer learning is another solution, where each data instance is associated with a weight of importance; see equation 2.4 [17]. The probability densities for the source and target domain are denoted $P_S(\mathbf{x})$ and $P_T(\mathbf{x})$ respectively. More important cases, e.g. keywords in KWS problems, will have a larger weight compared to less important cases.

$$\beta(\mathbf{x}) = \frac{P_S(\mathbf{x})}{P_T(\mathbf{x})}. \tag{2.4}$$

To generate a data set that better matches the userbase, one could either increase the collection of underrepresented samples or use data agglomeration from multiple sources to expand the whole dataset [18]. However, the latter presumes that the sources contain data for similar tasks and share a common feature set, and that the data is normalized if needed.

2.2.1.3 Coping with distorted audio

In real-world usage, it is unlikely for a KWS system to receive completely clear signals without distortions from noise, reverberation, multiple speakers, etc. Aside from handling this through the development of a robust ML model with slightly noisy training data, there are some hands-on coping mechanisms for these issues, which will be touched upon here.

A common issue for home assistants is reverberation, which can cause a distorted speech signal [19]. On the other hand, it has been found that reverberation times in most passenger cars are around 40 ms, which is very short in comparison to other environments [20]. Since early reverberation, with a time delay up to about 50 ms, actually increases the signal level and thus improves the perceptual quality, reverberation is not an issue to take into consideration in an automotive environment [19].

It is inevitable that the microphones will capture secondary sounds other than the target speech. To separate audio sources, one can make use of blind source separation, which uses unsupervised methods, e.g., clustering, and often spatial mixture

models to separate a speech signal into individual speakers. The model's parameters and source activity probabilities (SAP) are estimated through an Expectation-Maximization (EM) algorithm. See appendix A.1 for details of the EM algorithm. A proposed usage of this is to search for the presence of a wake-up keyword in the speech, and if it is found, the SAP can be estimated for the presence of this keyword. The source with the highest SAP is considered to contain the command intended for the home assistant [19].

Another issue, both relevant in a home and automotive environment, is that the KWS system will capture audio signals from its own responses to the user, which multichannel acoustic echo cancellation can solve [19]. It relies on adaptive filters, which estimate the acoustic paths between loudspeakers and microphones, to identify and subtract the audio from the speakers when it reaches the microphones [19]. The microphone hardware that is available at Aptiv includes an echo cancellation system, and therefore this issue will not be taken into consideration further on.

2.2.2 Audio feature extraction

Feature extraction is the process of extracting important information from a data source. By removing ineffective features, the data becomes more manageable without losing important or relevant information. Furthermore, reducing the data dimensionality enables a more concise and effective ML system with reduced training time and less computational cost [21]. Feature extraction is highly important for ML models within KWS problems since raw sound waves include plenty of redundant information and will generally benefit from feature extraction.

2.2.2.1 Mel scale

The mel scale is a range of pitches that, to the human ear, appear to be equidistant from one another, and it was developed to mimic the human perception of sound [22]. This scale provides a higher resolution at lower frequencies and an increasingly coarse resolution as the frequency increases, similar to how the human ear works. The fundamental frequencies of human speech range between 85 and 255 Hz, which are emphasized by transformation to the mel scale and thus benefit the sound models in learning what features are important [23]. Therefore, mel scale transformation is prominent in a variety of features and has been extensively utilized in the literature regarding KWS.

Mels can be calculated as

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

where f is frequency in Hz [24]. The relation of mel to Hz is also illustrated in figure 2.3.

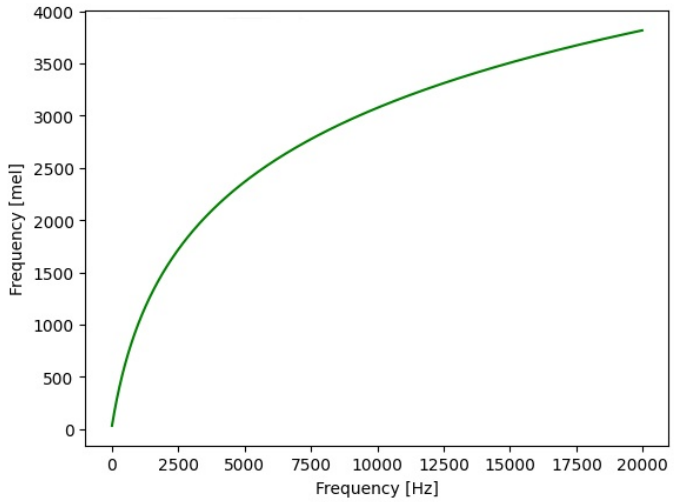

Figure 2.3: A frequency range from 20 to 20 000 Hz converted to mel scale.

2.2.2.2 Spectrogram

A spectrogram is a visual representation of the spectrum of a signal as it varies with time. An example can be seen in figure 2.4. Commonly used in KWS problems are mel spectrograms, which are these values transformed to the mel scale [1]. Mel spectrograms are computed according to the following scheme [1]:

1. The signal is partitioned into overlapping time frames with a specified window size and stride length.

2. The fast Fourier transform (FFT) is computed on each partition of the speech signal, which converts the sound wave from time domain to frequency domain. FFT is a fast algorithm for computing the discrete Fourier transform, defined as in equation 2.6 [25].

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \qquad k = 0, ..., N-1 \tag{2.6}$$

3. The entire frequency spectrum is converted to mel scale (see equation 2.5) and the spectrogram is generated by decomposing the magnitude of the signal into its components corresponding to the mel scale.

4. In some cases, the logarithm of the powers at each mel frequency is computed. This will give a log-mel spectrogram.

The methodology behind this feature extraction is also shown as a flow chart in figure 2.6. The effect that window size and stride length has on performance was evaluated in a KWS task using a ML model in [26]. When varying the window size between 20 and 50 ms and the stride length between 10 and 40 ms, the prediction

accuracy varied by about one percentage unit. The top performance was achieved with a 40 ms window size and a 20 ms stride length.

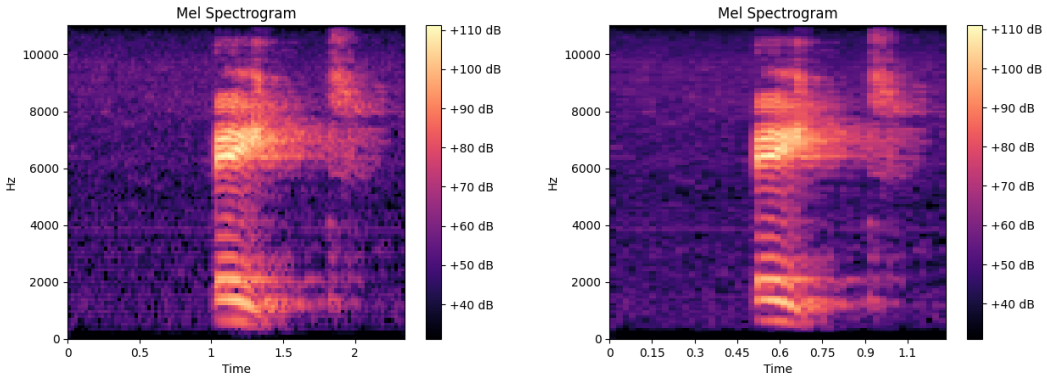

Figure 2.4: Two example figures of a spectrogram visualizing an utterance of the word "eight". The left spectrogram has a window size of 25 ms and a 10 ms stride length, while the right spectrogram has a 50 ms window size and a 20 ms stride length. This causes a slight change in the resolution of the images.

2.2.2.3 Filterbank

A filterbank is an array of bandpass filters, dividing the input signal into several parts where each one has a certain frequency sub-band of the original signal. An example of such bandpass filters can be seen in figure 2.5. When it comes to audio processing, which is a case of time-frequency signal processing, a filterbank represents the signal in a joint time-frequency domain, similar to a spectrogram described above. The difference is in the way that the features are extracted. Filterbanks are obtained by dividing the frequency domain (not the time domain) into different bandpass filters [27]. Similarly, as in the case of spectrograms, the filterbank values can be converted to the mel scale, i.e., mel filterbanks, or log scale, i.e. log filterbanks.

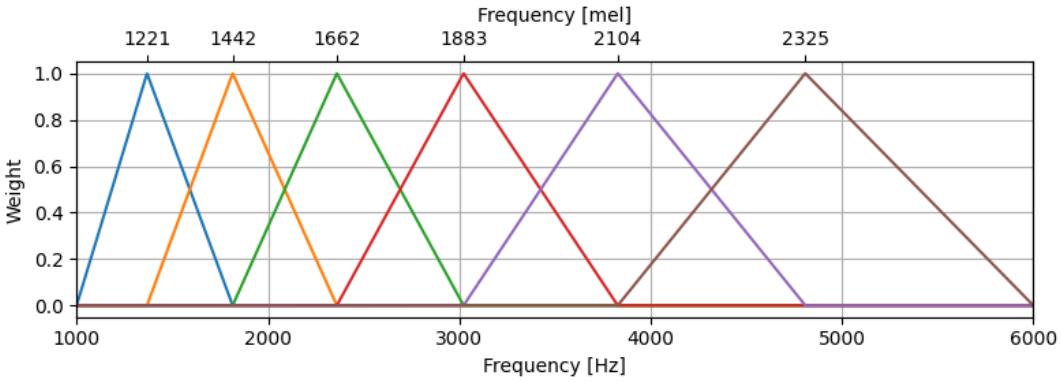

Figure 2.5: Mel filterbank with six triangular bandpass filters over a frequency range of 20 to 6000 Hz.

Figure 2.6: A flow chart describing the algorithm for extracting log-mel spectrograms and MFCC.

2.2.2.4 MFCC

Mel frequency cepstral coefficients (MFCC) are a common feature used in KWS. There are multiple ways to compute MFCC, but one way is to extend the computation of the log-mel spectrogram described in section 2.2.2.2. This is described as a flow chart in figure 2.6. From a log-mel spectrogram, MFCC are obtained by taking the discrete cosine transform (DCT) of the log-mel powers [28]. The MFCC are the amplitudes of the resulting spectrum. MFCC have the dimension (t,f), where t is the number of time steps and f is the number of features extracted from the frequencies.

2.2.2.5 Linear Predictive Coding

Linear Predicative Coding (LPC) is a feature that imitates the resonant structure of the human vocal tract, which creates the sound [29]. By considering a discrete time system with only p poles, it is possible to see the speech signal $x(n)$ as a result of the current input $e(n)$ and past samples $x(n-k)$, $k = 1, ..., p$, according to the formula in equation 2.7 [30].

$$x(n) = \sum_{k=1}^{p} a_k x(n-k) + e(n). \tag{2.7}$$

Here, p is often set to six for a speech signal since it is known that three formants (local maxima in a spectrum of frequencies) are enough to discriminate most vowels [31]. This knowledge makes LPC a suitable feature extraction method for speech.

By translating equation 2.7 to matrix form and solving it as a linear regression problem ($\boldsymbol{e} = \boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{a}$) one obtains a formula for the coefficients a_k. This solution is presented in equation 2.8 and holds if $N >> p$.

$$\boldsymbol{a} = (\boldsymbol{A}^T \boldsymbol{A})^{-1} \boldsymbol{A}^T \boldsymbol{b} \tag{2.8}$$

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_p \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} x(n) \\ \vdots \\ x(n+N) \end{bmatrix} \qquad \boldsymbol{e} = \begin{bmatrix} e(n) \\ \vdots \\ e(n+N) \end{bmatrix} \qquad \boldsymbol{A} = \begin{bmatrix} x_0 \\ \vdots \\ x_N \end{bmatrix}$$

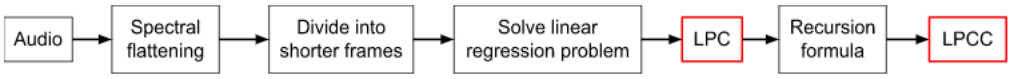

Figure 2.7: A flow chart describing the algorithm for extracting LPC and LPCC.

Using this knowledge, LPC features are computed in the following way:

1. Spectrally flattening the speech signal.

2. Dividing the signal into shorter frames with a fixed window size and stride length. The window size is denoted N in previous equations.

3. For each frame, the coefficients $\boldsymbol{a}$ are computed, and the variance of the residual vector $\mathbf{e}$, which is denoted σ^2. The LPC features are $\boldsymbol{a}$ and σ^2.

This is also described in figure 2.7, as a flow chart.

2.2.2.6 Linear Prediction Cepstrum Coefficients

Linear Prediction Cepstrum Coefficients (LPCC) are more commonly used in speech applications than the LPC features due to their improved robustness in speech recognition [32]. The LPCC features $\boldsymbol{c}$ are computed from the LPC features $[\boldsymbol{a}, \sigma^2]$ using a recursion formula displayed in equation 2.9 [33]. The notations here are the same as the ones in the formulas in the previous section. A flow chart describing the LPCC computation is found in figure 2.7.

$$\begin{aligned} c_0 &= In\sigma^2 \\ c_m &= a_m + \sum_{k=1}^{m-1} \Big(\frac{k}{m}\Big) c_k a_{m-k} \qquad 1 \leq m \leq p \\ c_m &= \sum_{k=1}^{m-1} \Big(\frac{k}{m}\Big) c_k a_{m-k} \qquad m > p \end{aligned} \tag{2.9}$$

2.2.2.7 Delta features

In previous ASR tasks, it has been found that adding delta features, which are the first-order time derivatives of the static features, can improve performance [34–36]. Delta features are computed according to equation 2.10 [37].

$$d(t) = \frac{\sum_{k=1}^{K} k\Big(c(t+k) - c(t-k)\Big)}{2\sum_{k=1}^{K} k^2}, \tag{2.10}$$

where d(t) is the delta feature at time t, c(t) is the static feature at time t, e.g., a MFCC, and K is the delta window size.

2.3 Models for KWS

The data for ML problems can essentially be divided into two types: labeled and unlabeled data. KWS tasks considers labeled data, since the data, i.e. audio samples, will have different labels, i.e. word classes. The problem for the ML model is to learn the patterns of the data that belong to each label and then correctly classify new samples where the label is unknown. This is called a classification problem.

There are many possible choices for ML models when it comes to KWS problems. Historically, HMM have been a common choice, but during the last couple of years, ANN has taken over due to its superior performance over HMM [1]. ANN is a broad class of models with varying architectures and setups. In the following section, the theory behind both HMM and different types of ANN will be presented.

2.3.1 HMM with Gaussian Mixture Models

HMM is a stochastic model used to model data that can be represented as a sequence over time, e.g., speech. In HMM, it is assumed that the states, e.g., the sequence of phonemes, are unobserved (hidden) and only the results from some probability function are observed, e.g., keyword classes [38]. HMM consists of two parts. The first is the transition model, which is based on the Markov process of discrete hidden states, and the second is the emission model, which accounts for the likelihood of observations given a hidden state. In this case, one considers Gaussian Mixture Models (GMM) as the emission probabilities [39]. GMM is a probabilistic model that combines multiple Gaussian distributions to describe a set of data. The model has three parts: mixing parameters ϕ_k, mean vectors $\boldsymbol{\mu}_k$ and covariance matrices $\boldsymbol{C}_k$. These model parameters are estimated from data in ML applications. The probability $p(\boldsymbol{x})$ from GMM, given a data point $\boldsymbol{x}$, is computed according to equation 2.11 [40].

$$
\begin{aligned}
p(\boldsymbol{x}) &= \sum_{i=1}^{K} \phi_i \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i, \mathcal{C}_i) \\
\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i, \mathcal{C}_i) &= \frac{1}{\sqrt{(2\pi)^K |\sum_i|}} \exp\Big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T \mathcal{C}_i^{-1} (\boldsymbol{x}-\boldsymbol{\mu}_i)\Big) \\
\sum_{i=1}^{K} \phi_i &= 1
\end{aligned}
\tag{2.11}
$$

Markov processes make use of the assumption that the probability of being in a state at a time t depends only on the state at the time $t-1$. Another assumption in HMM is that the probability of an output o_t only depends on the state that produced the output. The model makes use of a transition probability matrix $\boldsymbol{A}$ in which component a_{ij} gives the probability to transfer from state i to state j. Emission probabilities $\boldsymbol{B} = b_i(o_t)$ give the probability that an output o_t is generated from state i [41].

In the case of KWS, the aim is to estimate the transition probability matrix $\boldsymbol{A}$ and the emission probabilities $\boldsymbol{B}$ from data containing states (speech) and observations (keywords), in order to make new output predictions [41]. This can be done by estimating the maximum likelihood of these parameters with an EM algorithm, which is an iterative forward-backward algorithm. The details of this algorithm are presented in appendix A.1.

2.3.2 ANN

ANN is a class of commonly used ML models which have increased rapidly in popularity during the last decade in many research fields, where KWS is one [1]. An ANN can be seen as a set of algorithms that aim to discover and replicate patterns in data. The models consist of computational units z, which are called neurons or nodes. Each neuron gives a single value, and to represent data of a larger dimension, several neurons are combined to form so-called layers. The input neurons contain the input data, which is fed forward through "hidden" layers, i.e., neither input nor output layer, to an output layer, which will give the output of the task. This can be either a value in the case of regression or an output class in the case of classification, e.g., KWS. All connections between neurons in different layers are assigned weights w and each neuron has a given threshold Θ. These are the parameters that are altered in the training of the ML model, and thus the size and number of layers in the network will affect both the performance and complexity of the system. The ANN can either be fully connected (FC), called dense, i.e., all nodes in a layer are connected to all nodes in the upcoming layer, or restrictions on the number of connections can be included to decrease complexity [42].

All nodes in an ANN are considered a perceptron and their values are computed according to equation 2.12 [42].

$$z_j^p = g\Big(\sum_i z_i^{p-1} w_{ji}^{p-1,p} - \Theta_j^p\Big). \tag{2.12}$$

Here, z_j^p and z_i^{p-1} are states of nodes j and i in layer numbers p and $p-1$ respectively; see figure 2.8. Moreover, $w_{ji}^{p-1,p}$ is the weight between layers $p-1$ and p and nodes i and j in these layers. Lastly, Θ_j^p is the threshold for node j in layer p and $g(\cdot)$ is an activation function. An activation function can enable the models to find more complex patterns within the data. Commonly used activation functions can be seen in equation 2.13 [42]. To avoid the vanishing gradient problem, where ANN training practically stops due to stagnating parameter updates, it can be beneficial to choose an activation function that does not saturate at large z, such as the ReLU [42]. ReLU as an activation function and softmax as an output activation function are a common combination among KWS models presented in previous studies [43–48]. The output from softmax can be interpreted as a probability for each class in a classification task, which makes it suitable as an output activation function [42].

$$\begin{aligned} \text{Linear activation function}: \ & g(z) = z \\ \text{Softmax activation function}: \ & \sigma(z)_i = \frac{e^{\alpha z_i}}{\sum_{k=1}^{M} e^{\alpha z_k}} \\ \text{Rectified Linear Unit (ReLU) activation function}: \ & R(z) = max(0, z) \\ \text{Hyperbolic tangent activation function}: \ & g(z) = tanh(z) \end{aligned} \tag{2.13}$$

2.3.2.1 Feed forward neural network

Figure 2.8 displays an ordinary feed-forward neural network (FFNN), in which the information is transferred in one direction from the input layer, through all hidden layers, to the output layer. This network is FC, i.e., all nodes in the current layer are connected to all nodes of the next layer.

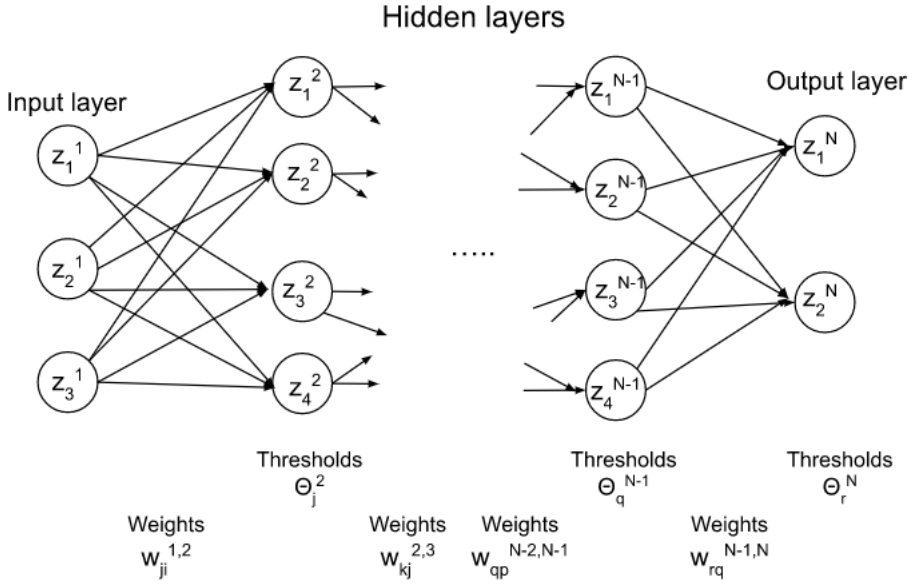

Figure 2.8: A FFNN with input layer, hidden layers and output layer. The neuron states z, weights w and thresholds Θ are defined in accordance with equation 2.12.

2.3.2.2 Convolutional neural network

Convolutional neural networks (CNN) are another type of ANNs that are commonly used in image classification due to their ability to capture spatial and temporal dependencies in an image. The most common features for KWS are two dimensional, and can thus be treated as images, so similar techniques can be applied. A CNN includes convolutional layers, which are kernels/filters, i.e., matrices, that go through the input with a specified stride and perform element-wise multiplication operations with each part of the input. An example of this is shown in figure 2.9. The convolutional layers can capture the shapes of an image, such as corners, while reducing the spatial size of the convolved feature. A pooling layer, which is often applied after a

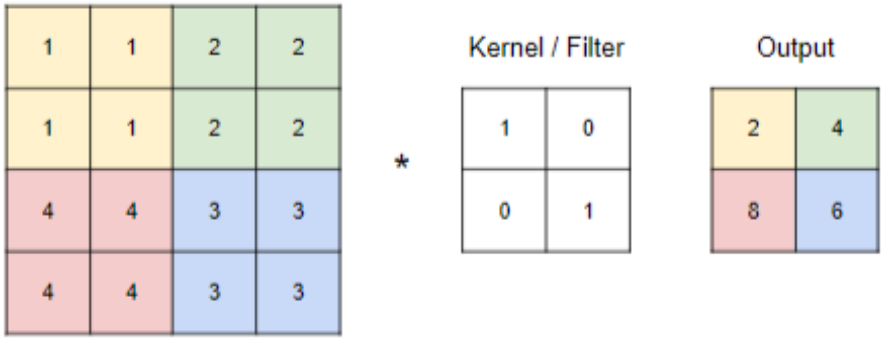

Figure 2.9: An example of a convolution, which transforms a (4,4) input space to a (2,2) output space. The kernel size is (2,2) and has a stride of (2,2). Therefore, the kernel is multiplied with the four elements of similar color in the input matrix, which gives a single value corresponding to the element of the corresponding color in the output matrix.

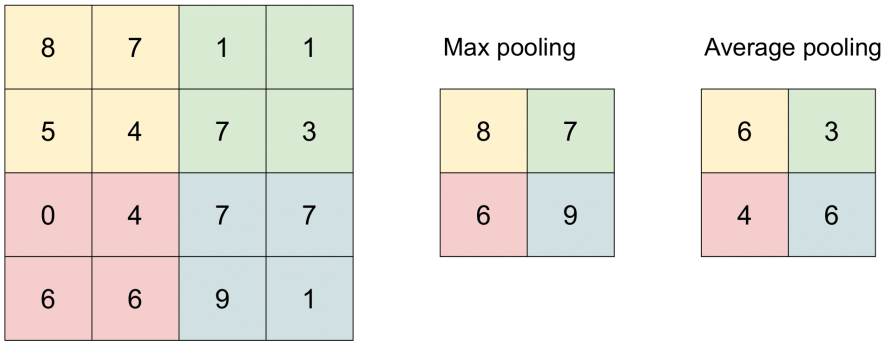

Figure 2.10: An example of a of pooling, which transforms a (4,4) input space to a (2,2) output space. The feature map is (2,2) and has a stride of (2,2). Therefore, the maximum (or average) value of the four elements of similar color is transformed into a single value corresponding to the element of the corresponding color in the output matrices.

convolutional layer, also possesses the latter ability as well as the ability to extract dominant features. This type of layer also makes use of a kernel, but instead of a matrix multiplication, the maximum, in the case of max pooling (MP), or average, in the case of average pooling, is taken from the proportion of the input covered by the kernel; see figure 2.10. A CNN ends with a flattening of the output before it is fed to an ordinary feed-forward network for classification. [42]

Aside from ordinary convolutional layers, there are also other types of convolutional layers, such as depthwise separable convolutional (DSC) layers. This type of convolution consists of a depthwise convolution, i.e., a spatial convolution, followed by a pointwise convolution, which projects the previous output onto a new channel space [49].

2.3.2.3 Temporal convolutional network

The idea of CNN is to use convolutions to successively and iteratively combine lower-level features into higher order concepts. Many common features extracted from mono audio signals, like MFCC, spectrograms, and filterbanks, are two-dimensional. This makes it possible to use traditional image networks with two-dimensional convolutions. However, a small field of reception means it is difficult to capture relationships on both ends of the frequency spectrum with a relatively shallow ANN. Therefore, an idea is to focus on audio signals as time series and thereby introduce convolution in the temporal dimension [46]. For example, MFCC features have the dimension (t, f, c), where t, f, and c is the time, feature and channel dimensions, respectively. However, when using temporal convolution (TC) and a single channel, the time and feature dimensions are switched, resulting in the dimension (t, 1, f). Thus, instead of feeding a two-dimensional image, the TC is fed one-dimensional sequential data with f features. The first main advantage of this approach is a larger receptive field, as all the lower-level features are building the higher-level concepts. This results in better model performance for a relatively shallow network. The second benefit is that TC reduces the feature map size, thereby decreasing the total computational complexity [46].

2.3.2.4 Recurrent neural network

Recurrent neural networks (RNN) not only make use of the ordinary feed-forward structure, but also recurrent connections by transmitting information from a processing node back into the network. Thus, one can unravel the network not only in space but also in time; see figure 2.11. Since this can be used to model sequential data, these networks are often used in text or speech applications [42]. A RNN can be extended to be bidirectional, which consists of two RNNs out of which one takes the input in the forward direction and one in the backward direction [50].

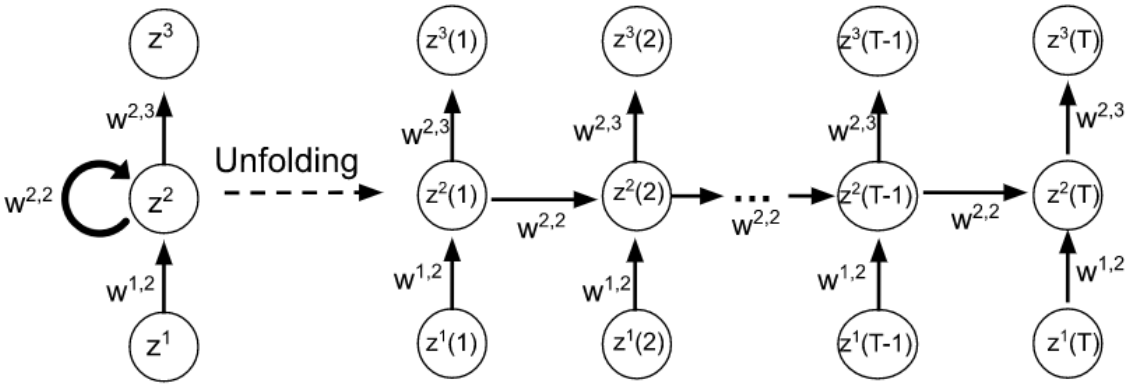

Figure 2.11: An example image of a RNN with one neuron in each layer. Layer 1 and 3 is the input and output layer respectively. Note that a time index is added as the network is unfolded and that the same weight is used between each of the time steps.

A drawback of an ordinary RNN is that long-term dependencies will disappear since

the information has to be transferred through many units [51]. More complex units, called Long Short-Term Memory (LSTM) units, were introduced in 1997 to cope with this [52]. The idea is a cell state that runs through the entire chain of recurrent units with only minor additions or removals of information at each unit through so-called gates [51]. In addition to the cell state, there is another information transfer from each unit, which is both an output and a transfer to the next unit. This is similar to the single outputs from each unit in an ordinary RNN. A slightly simpler unit, which still makes use of LSTM's core idea with a cell state and an additional output, is the Gated Recurrent Unit (GRU), introduced by Cho et al. in 2014 [53]. GRU makes use of two gates, named update gate p_t and reset gate r_t. These are updated according to equation 2.14 [54].

$$\begin{aligned} p_t &= \sigma(\mathbf{W}_z z_t^1 + \mathbf{U}_z z_{t-1}^2 + \mathbf{b}_z) \\ r_t &= \sigma(\mathbf{W}_r z_t^1 + \mathbf{U}_r z_{t-1}^2 + \mathbf{b}_r) \end{aligned} \tag{2.14}$$

The current hidden state z_t^2 is computed according to the formula in equation 2.15, where $\odot$ denotes elementwise multiplication and $g(\cdot)$ an activation function. All $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{b}$ are trainable parameters.

$$\begin{aligned} z_t^2 &= (1 - p_t) \odot z^2 + p_t \odot \hat{h}_t \\ \hat{h}_t &= g(\mathbf{W}_h z_t^1 + \mathbf{U}_h(r_t \odot h_{t-1}) + \mathbf{b}_h) \end{aligned} \tag{2.15}$$

Another type of RNN that has shown promising results within KWS is a convolutional RNN (CRNN), which makes use of both convolutional and recurrent layers [48].

2.3.2.5 Attention neural network

The attention mechanism has been used together with RNN in a previous KWS task, forming an Attention-RNN (Att-RNN) model [45]. The idea behind the attention mechanism is to introduce increased attention to more important parts of the data [50]. An attention layer takes in three inputs: keys $\mathbf{K}$ and queries $\mathbf{Q}$ with dimension d_k, and values $\mathbf{V}$ with dimension d_v. The output from the attention layer is computed according to the formula in equation 2.16 [55]. The keys, queries, and values can either come from different sources or from the same source, and in such a case, one would obtain a self-attention layer.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \sigma\Big(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\Big)\mathbf{V} \tag{2.16}$$

The attention mechanism can be divided into h different layers, or heads, where the total dimensionality of the model is divided among the heads, thus not generating

an increase in complexity [55]. This type of model is called Multi-Head Attention (MHAtt) model and the formula for this is presented in equation 2.17. The $\mathbf{W}$ matrices will linearly project the queries, keys, and values to dimensions d_k, d_k, and d_v, respectively.

$$\begin{aligned} \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{concatenate}(\text{head}_1, ..., \text{head}_h)\mathbf{W}^O \\ \text{head}_i &= \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \end{aligned} \tag{2.17}$$

2.3.2.6 Residual neural network

Another common architecture is the residual neural network (ResNet), which introduces the idea of shortcut connections that bypass some of the network layers. A shortcut connection, see figure 2.12, reduces the vanishing gradient problem facing deep neural networks by reintroducing information from previous layers [42]. This makes it easier for the network to update its trainable parameters to possibly increase the model's performance. The effect is especially true when faced with a large number of layers. A typical residual network architecture has initial layers followed by a number of residual blocks, and finally the output layers [56, 57]. The shortcut connections are located in the residual blocks.

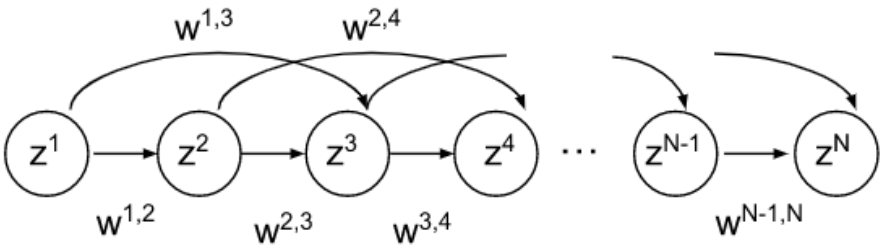

Figure 2.12: An example image of a RNN. For simplicity, each layer only contains a single neuron. In this example, the shortcut connections skip one layer each. In reality, the length of the shortcuts can differ from model to model.

2.3.3 Training an ANN

The first step in training an ANN is to initialize the weights and thresholds. There are different approaches to this, and a suitable and common method is to sample values randomly close to zero, e.g., using the normal distribution [58]. Furthermore, training data is divided into smaller portions, named minibatches. A minibatch is used to compute the loss between true labels and the model's predicted labels, where errors are propagated from the output layer backwards through the network, updating the values of the weights and thresholds [42]. The model is trained on all minibatches to update its parameters, and iterating through all minibatches is called an epoch. The number of epochs is how many times the model has worked on the entire data set. There are multiple methods of updating the model's weights

and thresholds. A common optimization algorithm is stochastic gradient descent (SGD), which updates the weights and thresholds as

$$\delta w_{pn} = -\eta \frac{\partial \mathcal{L}}{\partial w_{pn}} \qquad \delta \Theta_p = -\eta \frac{\partial \mathcal{L}}{\partial \Theta_p}, \tag{2.18}$$

where η denotes a learning rate, the loss function is denoted $\mathcal{L}$, the threshold for layer p is Θ_p and the weights between layers n and p are w_{pn} [42]. The weights and thresholds are updated by adding the increments computed in equation 2.18 to the previous parameter values.

2.3.3.1 Loss functions

In order to estimate the parameters of an ANN, a loss function is used to define optimization criteria. The most commonly used loss function in the training of KWS models is cross-entropy (CE) loss [1]. This is defined as

$$\mathcal{L}_{CE} = -\sum_{i} \sum_{n=1}^{N} l_n^{\{i\}} \log\left(\mathbf{y}_n^{\{i\}}\right). \tag{2.19}$$

Here $l_n^{\{i\}}$ is the true label and $\mathbf{y}_n^{\{i\}}$ is the posterior of the n-th class, given the input $\mathbf{X}_{\{i\}}$, and i is a discrete time segment [1].

There have been studies on how to improve the performance of KWS systems by changing the loss function. Both MP loss and low variability orthogonal (LOVO) loss have been found to improve performance compared to CE loss [59, 60]. The theory behind these loss functions is presented in appendix A.2. However, these loss functions are not yet well established, and therefore there is not an easy method for implementing them. Therefore, a demarcation was made in this project to focus on CE loss in order to focus on the more crucial parts of the project. The same choice of loss function has been made in several previous KWS studies [43, 44, 47, 61]. The results from previous studies in this area will be described in the upcoming chapter.

2.3.3.2 Regularization techniques

ANNs can contain billions of parameters that should be updated, like GPT-3's 175 billion parameters [62]. This could increase the risk of overfitting, which happens when a network learns the patterns of the training data too well and has difficulties predicting new data. However, there are regularization techniques to reduce overfitting such as adding normalization and dropout layers [63]. A dropout layer will make the model randomly disregard a given fraction of neurons and only update the other neurons' parameters [42]. This will create a slightly different ANN for each training round, which is something that has been found to improve performance [63, 64]. Another technique, early stopping, reduces the risk of overfitting by keeping track of the prediction metrics of unseen data, e.g., validation accuracy or validation loss, and thereby stopping the training before these values deteriorate [42].

2.3.3.3 Learning rate

A larger learning rate η lets the network avoid a loss function's shallow local minimas, but this leads to large variations in the loss function value and difficulties in the convergence of the model's weights [42]. An adaptive learning rule can be a solution to this. One variation is adding another term to the weight increments, as in equation 2.20. This variant of SGD is called SGD with momentum, where $\alpha \geq 0$ is a momentum constant and t is a time index.

$$\delta w_{pn}^{t} = \eta \frac{\partial \mathcal{L}}{\partial w_{pn}^{t}} + \alpha \delta w_{pn}^{t-1} \tag{2.20}$$

In previous studies on KWS, it is common to reduce the learning rate after a fixed number of training iterations or when the performance reaches a plateau [44–48, 59]. The initial learning rates and decay schemes vary based on the task and other settings.

In a study exploring the effects of batch size and learning rate, it was found that these parameters have a significant impact on model performance [65]. There is also a correlation between the two. For a lower learning rate, a larger batch size is suitable, and vice versa. Batch sizes used in previous KWS tasks are, e.g., 64 [45, 47, 48], 100 [44, 46], and 256 [59]. Also, it has been shown that larger batch sizes tend to worsen model generalization compared to smaller batch sizes [66].

2.3.3.4 Optimization algorithms

Apart from SGD, there are other optimization algorithms to train ANNs. One is RMSProp, in which one uses a moving average of the magnitudes of the recent gradients to adapt the learning rate [67]. The difference from SGD is the factor $\frac{1}{\sqrt{G_t}}$, see an example for weight increments in equation 2.21. The formula for G_t is shown in equation 2.22, where β is a moving average parameter commonly set to 0.9.

$$\delta w_{pn}^{t} = \frac{\eta}{\sqrt{G_t}} \frac{\partial \mathcal{L}}{\partial w_{pn}} \tag{2.21}$$

$$G_t = \beta G_{t-1} + (1-\beta)\Big(\frac{\partial \mathcal{L}}{\partial w_{pn}}\Big)^2 \tag{2.22}$$

A third alternative as optimization algorithm is the Adam optimizer, which has been used in previous KWS studies [44, 45, 48, 68]. It is a mixture of SGD with momentum and RMSprop. The increment formula for the model weights in the Adam algorithm can be seen in equation 2.23 [69]. The formulas for $\hat{m}_t$ and $\hat{v}_t$ are displayed in equation 2.24. Default values for the new parameters are $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

$$\delta w_{pn} = \eta(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}) \tag{2.23}$$

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \qquad & m_t &= \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial \mathcal{L}}{\partial w_{pn}} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \qquad & v_t &= \beta_2 v_{t-1} + (1-\beta_2)\Big(\frac{\partial \mathcal{L}}{\partial w_{pn}}\Big)^2 \end{aligned} \tag{2.24}$$

In the original article where Adam was presented, promising results were shown with both improved performance and complexity over the standard SGD algorithm [69]. In a KWS task performed with a CNN and spectrogram features, the Adam optimizer was found to converge faster than SGD but with the same performance [26]. On the other hand, a study from 2017 showed that the Adam optimizer does not generalize as well as SGD with momentum [70]. A variant of the standard Adam algorithm is Nesterov momentum Adam (Nadam), which is similar to RMSProp but uses Nesterov momentum instead [71]. Lastly, AdaMax is a variant that makes use of the infinity norm and was presented in the same paper as the standard algorithm [69]. The pros and cons of the optimizers imply that several optimizers should be tried in order to find the most suitable for KWS within an automotive environment. In this project, SGD, RMSProp, and Adam will be tested.

3

Results from previous studies

Several previous studies have been made within the area of KWS, mostly comparing the performance of different ANN models. Section 3.1 contains a presentation of some of these results, while section 3.2 contains some results from comparisons of different feature extraction methods.

3.1 Comparing ML models

In a study from 2018, Yundong Zhang et al. evaluated multiple ANN models on KWS with ten keywords from the GSC dataset and two additional classes: "other" and "silence" [44]. This specific KWS task will be referred to as the "standard KWS task", as this identical setup has been replicated in multiple studies and seems to be a standard setup in KWS. In [44], 10 MFCC features were extracted using a 40 ms frame length and a 20 ms stride length. Results from Zheng et al. and similar studies on the same task have been summarized in table 3.1. Some augmentation techniques have been applied, of which adding audio noise is one. The type and level of this noise have nevertheless not been presented, which should be taken into consideration when evaluating these results.

In a KWS task similar to the standard one but with two additional keyword classes, an Att-RNN with mel-scale spectrogram features achieved 96.9% ($\pm$0.2%) accuracy [45]. It should be noted, that no augmentation, and particularly no background noise, was added to the audio segments. In [43] this model is extended to include four multi-head attentions, i.e., a MHAtt-RNN, and the LSTM units are replaced with GRU units, which improved performance to 98.0% accuracy.

Many previous studies within KWS have made use of MFCC features and the GSC dataset. An exception is the work done by Tara N. Sainath and Carolina Parada, who use log-mel filterbank features and a completely new dataset with two word phrases to investigate different CNN models [61]. It is found that when limiting multiplications to 500K, a CNN model with shifting convolutional filters in frequency gives over 27% relative improvement in performance over a baseline FFNN. A CNN using pooling in time is found to improve performance with 41% compared to the baseline when limiting the number of parameters to 250K. Since this KWS task is

Table 3.1: Performance from previous studies with different ANN models. All results, except ∗, come from a KWS task performed on the GSC dataset, with 12 classes including "other" and "silence". M = million, and K = thousand. The "+ stride" denotes a network with striding larger than 1; otherwise, striding = (1,1) has been used. The values missing in the table have not been presented by the source. MFCC were used if nothing else is specified, and † and ‡ refers to spectrogram and raw waveform features, respectively.

Model	Accuracy [%]	Parameters [K]	Operations [M]
FFNN	86.7 [44]	—	—
	91.2 [43]	447	—
GRU	94.7 [44]	—	48.4
	97.2 [43]	593	—
LSTM	94.8 [44]	—	48.4
	97.5 [43]	—	—
CNN	96.0 [43]	606	—
CNN + stride	92.7 [44]	—	25.3
	95.6 [43]	529	—
	92.3† [26]	63.8	—
	97.4‡ [68]	∼700	—
CRNN	95.0 [44]	441	19.3
	97.5 [43]	467	—
DSCNN	96.9 [43]	490	—
DSCNN + stride	95.4 [44]	—	56.9
	97.0 [43]	485	—
TC-ResNet	96.6 [46]	305	13.4
	96.1 [46]	66	3.0
	97.4 [43]	365	—
Att-RNN*	96.9† [45]	202	—
MHAtt-RNN	98.0 [43]	743	—

completely different from the ones using the GSC dataset and no clear performance results are presented, it is difficult to compare these models with the previously presented ones.

TC-ResNet architectures have been evaluated in many previous studies, being the main focus in [46]. The results are presented in table 3.1 and here it is clear that these models are very promising within KWS. Just as for the other models in table 3.1, there is a trade-off between the size of the networks and their performance, where a large increase in size typically only yields a small performance increase. When increasing the TC-ResNet from 8 layers to 14, and including a multiplier of 1.5, performance is increased from 96.1 to 96.6 %, and the model size is increased from 66 K to 305 K [46]. The question is how considerable these differences are in a real-world application.

3.2 Comparing feature extraction methods

Apart from focusing on the models, comparisons between feature extraction methods have also been performed within ASR tasks. In a speech dysfluencies classification task, i.e., stuttering recognition, with k nearest neighbor and linear discriminant analysis, LPCC is found to outperform MFCC slightly, with the best accuracy of 94.51% compared to 92.55% for MFCC [72]. The same performance difference is found in a fixed phrase speaker verification task. LPCC features achieve 87.04% accuracy and MFCC 84.45% accuracy, but LPCC has much greater time complexity at 4.66 seconds against 0.93 seconds, respectively [73]. In a speech activity detection task, the same complexity difference is found [74]. More specifically, the ANN reaches the minimal error rate after 18 epochs using MFCC and after 37 epochs using LPCC. However, the testing error for the ANN does not show any significant difference between the two feature extraction methods. Quite different results are presented in a study where MFCC and LPCC are used in a speech recognition system using principle component analysis. The results from this show far better performance from MFCC, with a larger recognition rate 87.52% against 78.08%, and greater robustness [75]. Delta features (see section 2.2.2.7) were also tested and found to improve the performance. From the above, one can notice that the performance difference between MFCC and LPCC varies with the type of task. Even though LPCC is found to outperform MFCC in several cases, the large usage of MFCC in previous KWS tasks indicates that this is a feasible choice. Both feature extraction types will be investigated in this project.

There is a possibility to use trainable features where the features are updated through backpropagation training, just like ANN. Multiple experiments on a KWS task with the GSC dataset showed that there is no significant improvement in performance with these learnable features compared to handcrafted features such as MFCC and mel filterbanks [76]. Neglecting feature extraction and simply using raw waveforms have also been investigated, e.g., in a previous master's thesis by Patrick Jansson in 2018. He achieves a very high validation accuracy of 97.4% on the standard KWS task using a CNN [68]. However, as one can see in table 3.1, the CNN has a relatively large number of parameters. There is also some ambiguity in the level of noise that was added to the data, which of course has an impact on performance.

4

Automotive sound environment

The following chapter contains theory and findings regarding the automotive sound environment, which are found to be relevant for KWS within this project. The information presented here will be used in the development of the KWS models further on.

4.1 Previous KWS findings

In a previous study from 2022, car noise was added to the GSC dataset with an SNR randomly between -10 and 30. A ResNet with architecture taken from [77] achieved 90.4, 94.1, 95.8, and 97.4 validation accuracy with SNR set to -10, -5, 0, and 20 respectively [78]. These results were obtained with a new type of regularizer, combining a CE loss and a contrastive loss, which improved the results. However, the aim of that study was not a real-world implementation, and no restrictions on model complexity was made. A different approach presented in a study from 1995 added a mathematical noise compensation algorithm [79]. This improved accuracy on the TIDIGITS dataset [80] with artificial noise for an HMM GMM model from 92.3 % to 95.8 % with 10 dB SNR and from 76.1 % to 94.4% with 0 dB SNR. However, in a real-world driving scenario on the highway with unknown SNR, the accuracy only reached 66.1%. This could imply that the difference between noisy audio and adding artificial noise could be a crucial factor for performance in a real-life application. The study in [78] also shows a decline in performance when the model is tested on data with SNR lower than the training levels and with noise different from the training data. This emphasizes the importance of training the ML model in circumstances similar to those in the target domain. Thus, for this project, it is important to find the target circumstances, mainly the SNR level, in an automotive environment.

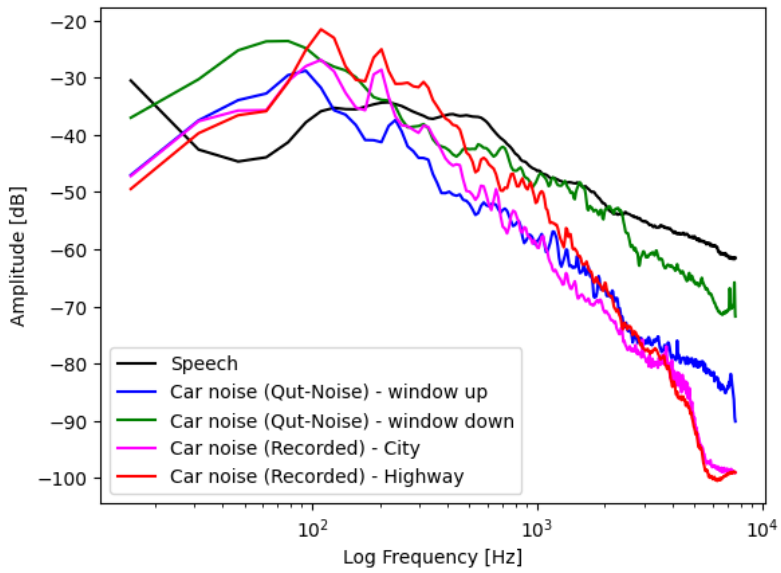

Figure 4.1: Spectrum of sound levels at different frequencies for two different sounds: speech and car noise. The x-axis displays log-scaled frequency and the y-axis amplitude in dB, with the loudest sound as reference. One can observe that the two different sound types have different curve shapes, with car noise having a larger amount of lower frequencies compared to speech.

4.2 Background noise

The background sound in an automotive environment differs from other types of sound, e.g., speech. In figure 4.1, a spectrum displaying the amount of sound at different frequencies for car noise and speech is shown. The car noise is divided into four different curves. Two of them are sounds taken from the QUT-NOISE dataset, one with car sounds recorded with the windows up and the other with the windows down. The two others come from recordings performed in this project in a Volvo XC60 with the windows up, driving in a city and highway environment, respectively. These recordings were performed with mobile phone microphones placed close to the car's built-in microphones at the ceiling. By observing the graphs in figure 4.1, it is noticeable that the car noises have similar shapes, with the most sound within 70 to 200 Hz. Moreover, one can also notice that three of these curves have peaks at around 110 and 200 Hz. The speech sound, which is taken from the LibriSpeech (LS) dataset [81], differs from the car sound as it has less energy at lower frequencies and more energy at higher frequencies. These differences imply that car noise should be less disturbing for ML than speech noise and probably also music noise. These differences could also imply that a ML model could benefit from focusing on higher frequencies than the fundamental frequencies between 85 and 255 Hz, at least within an automotive environment [23].

In figure 4.2, spectrums of different types of noise in an automotive environment are displayed, recorded in a Volvo XC60 with a mobile phone microphone. The most noticeable thing to observe from this is that all curves have peaks at 110 and 200

Hz, regardless of what kind of noise the car sound is mixed with. These peaks are characteristic of car noise and arise from the engine, tires, and other road noises. The precise values for the peaks are most likely dependent on the car, e.g., the design and interior of the car cabin. This is supported by the results in [82], where trucks are revealed to generate significant energy around 125 and 250 Hz. The differences in peaks between the QUT-NOISE recordings and the recordings done in this project, seen in figure 4.1, also implies this. If one could beforehand identify the peaks of the car noise, like the ones in figure 4.2, it would be possible to remove much of the sound at these frequencies and thus substantially lower the background noise. The risk with this approach is that it could also remove relevant parts of the speech, which will have a negative effect on the performance. Therefore, and due to limited resources, no signal processing will be actively used in this project. Instead, focus will be on processing the data in a way that will enable the model to learn the relevant characteristics of the speech despite background noise.

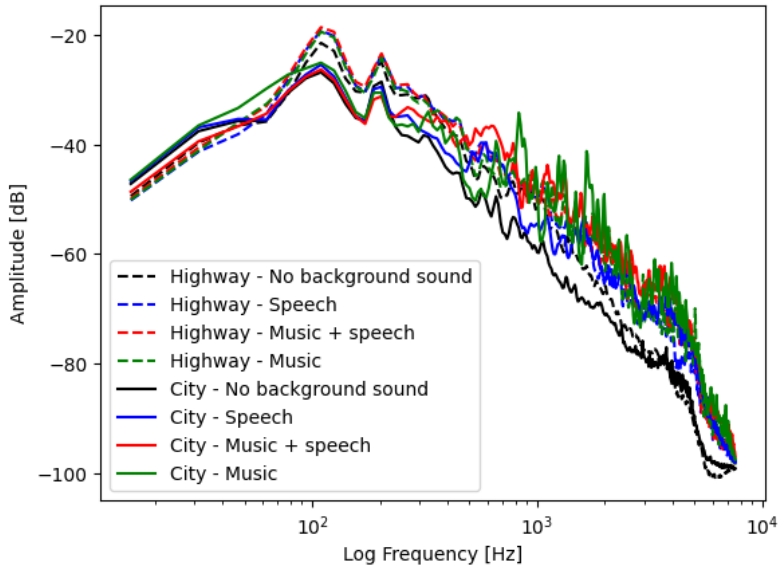

Figure 4.2: Spectrum of sound levels at different frequencies for different types of noise in an automotive environment. The x-axis displays log-scaled frequency and the y-axis amplitude in dB, with the loudest sound as reference. One can observe that all curves have two peaks at around 110 Hz and 200 Hz.

To put the recordings performed in this project into context, a small comparison between the mobile phone microphone and a car microphone was made. Sound was recorded simultaneously by the two devices, and a spectrum of the result was made, which is presented in figure 4.3. From this, one can draw the conclusion that the car microphone captures less sound at low frequencies, which will substantially lower the noise and thus improve the KWS performance.

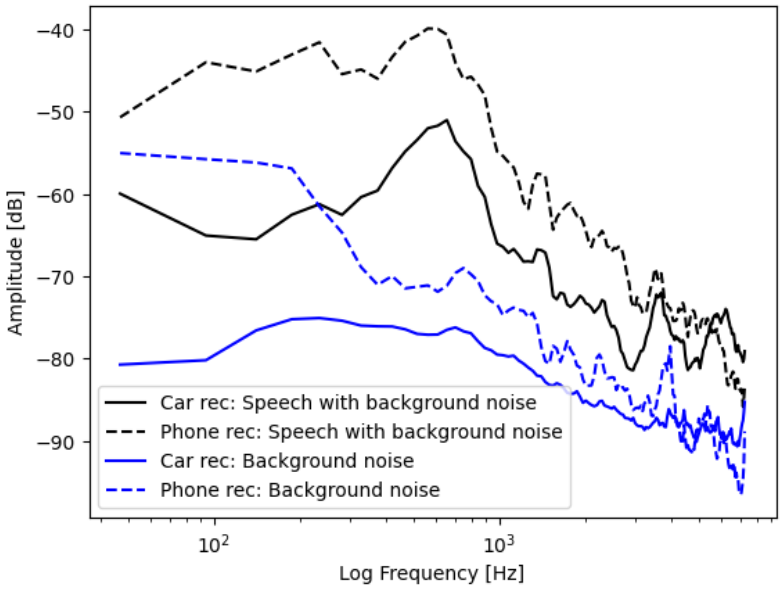

Figure 4.3: Spectrum of sound levels at different frequencies recorded with two different recording devices. The background noise was recorded car noise from the QUT-Noise dataset, which was played on a computer in the background. The "car" recording device is a microphone that is currently being available on the market. One can observe that the mobile phone microphone captures much more of the lower frequencies compared to the car microphone.

4.3 SNR in an automotive environment

It has been shown that performance decreases when models are trained and used in different environments [78, 79, 83]. Thus, in order to be suitable for real world usage and particularly the automotive environment, car noise should be added to the training data to successfully build a performant model. In addition to ordinary car noise from the road, engine etc., background speech and background music are two common sources of noise that occur in an automotive environment. To fairly scale the added noise, it is relevant to find a suitable SNR level for the automotive environment. This has been investigated by J. Freudenberger et al. in 2010 [84]. Their study aimed to investigate noise reduction techniques in an automotive environment. As a part of this, SNR levels were computed from recordings in three different settings: driving at 100 km/h, driving at 140 km/h, and driving with the defroster on. The measured SNR levels vary between -0.7 and 10.8 dB depending on the driving setting, microphone placement, and height of the speaker. By using different combining techniques for noise reduction, the SNR could be increased to between 4.4 and 20.9 dB [84]. The SNR was lowest for short speakers, i.e., people farthest away from the microphones, and when driving at 140 km/h. Driving at 100 km/h and using the defroster resulted in similar SNR levels.

5

KWS modelling

The following chapter describes the procedure to collect, preprocess, and augment audio data, the architecture and implementation of the three selected KWS models, and how the models were evaluated in regards to performance and complexity. All parts of the programming have been conducted in Python.

5.1 Data sources

The project modelling is based on the GSC dataset version 2 acquired from [85]. It contains 35 different words spoken by 2618 persons, totaling 105 829 audio samples. Each sample is stored in a WAVE format with a duration of about one second and a sample rate of 16 kHz and is encoded as linear 16-bit single-channel PCM values.

The GSC dataset was partly developed with the purpose of enabling the creation and comparison of different ML models [86], and it has been used in several previous studies, as mentioned in chapter 3. In the original article [86], an accuracy of 88.2% was presented for the best-performing model on the standard KWS task. All 35 words in the GSC dataset can be seen in table 5.1, and the number of samples for each word is between 1557 and 4052. The ten keywords chosen in the standard KWS task, as well as in this project, are presented in bold.

Table 5.1: All 35 words that are included in the GSC dataset version 2, where the ten chosen keywords are marked in bold. The number in parenthesis beside each word refers to the number of instances (in thousands) of that word.

backward (1.7)	bed (2.0)	bird (2.1)	cat (2.0)	dog (2.1)
eight (3.8)	five (4.1)	follow (1.6)	forward (1.6)	four (3.7)
happy (2.1)	house (2.1)	learn (1.6)	marvin (2.1)	nine (3.9)
one (3.9)	seven (4.0)	sheila (2.0)	six (3.9)	three (3.7)
tree (1.8)	two (3.9)	visual (1.6)	wow (2.1)	zero (4.1)
down (3.9)	**go (3.9)**	**left (3.8)**	**no (3.9)**	**off (3.7)**
on (3.8)	**right (3.8)**	**stop (3.9)**	**up (3.7)**	**yes (4.0)**

Unlike many previous studies, complementing speech data is acquired from the LS ASR corpus [81]. It was developed to build ASR systems and contains longer seg-

ments from LibriVox's public domain audio books with a 16 kHz sample rate. More specifically, the version "train-clean-360" is used, as it does not contain background noise. It has a total of 363.6 hours of speech from 921 different speakers, where 482 are male and 439 are female, corresponding to 189.6 and 174.0 hours of speech, respectively.

Moreover, additional sources provided different types of background noise. Car noise was acquired from the QUT-NOISE dataset, available from [87]. This source contains background noise from various acoustic scenes, but since the purpose is to build models that perform within an automotive environment, only background noise from this setting was selected. This car noise was recorded on a single day in a single car and includes highway, city, and suburban driving with the car windows rolled up and down [87]. The audio was sampled at 48 kHz and 16 bits per sample. The audio was resampled to 16 kHz to match the GSC and LS data, and periods of silence were manually removed by the software Audacity, which is available from [88]. This reduced the total recording time by 45 minutes to 2 hours and 15 minutes.

Furthermore, background noise in the form of speech was sampled from each speaker in LS, and music noise was sampled from eight songs of different genres, provided by the Free Music Archive under the Creative Commons copyright license [89].

5.2 Data processing

The purpose of LS is to extend the "other" class that otherwise consists of the lumped-together non-keywords from GSC. There is a transcript for each audio file in LS that is checked for keywords, so only audio segments without keywords are sampled from. Therefore, the LS samples include random speech, but certainly without the selected keywords. So, the sampled content from LS can be added to the GSC "other" class without risking adding data containing keywords. Additionally, to guarantee data conformity, segments shorter than one second were padded with silence, and longer segments were cut off and trimmed.

The audio files are loaded and converted using the Python library `Librosa` [90]. The library has functionality that reads an audio file and produces an array of the audio time series. It has settings to ensure single channel audio and a 16 kHz sample rate.

The number of one second segments in LS could reach 1.3 million, not mentioning the possibility of overlapping segments. The GSC dataset is less than a tenth of that, and even less when selecting only some keywords. This is therefore considered a sparse data problem where relatively few samples have a label. To ameliorate this situation, a subset of LS is selected. And, primarily, keyword samples are augmented by altering loudness, pitch, speed, shifting, and VTLP to increase the sample size and variation of the labeled data using the library `nlpaug`. The volume is multiplied with a factor in $[0.8, 1.2]$, and speed by a factor in $[0.9, 1.1]$. The shift is set to 100 ms either forward or backward, as in [43]. The VTLP and pitch factor are randomly in $[0.9, 1.1]$ and $[-10, 10]$, respectively. These settings have been tested and found

to make reasonable alterations both in experiments performed in this project and previous studies [91]. The augmented keyword samples will be added to the dataset as new data. The augmentation of the "other" data will be done in place, i.e., without extending the dataset. This will minimize the difference in the number of keywords and non-keywords while creating a more varied dataset, which can hopefully lead to a more robust model.

The target group is both male and female, speaking English with a variety of pronunciations. However, the majority of recordings are from native English speakers, which could imply discrepancies in pronunciation between the user base and sources. This is a problem of unmatched data. The problem has been partly reduced by using data agglomeration using two data sources. Additional sources could have been used, but they would only be used to extend the "other" class and they would quickly face diminishing returns. In this case, data augmentation is considered more promising than implicitly introducing variations in the data.

In order to target the system towards usage in an automotive environment, background noise of different types is added to the speech data. The type and level of background noise coincide with the findings in chapter 4 and thus include regular car noise, background speech, and music. All data that is overlayed with noise will have car noise added. One third of the data will also get added speech noise, and one third will get added music noise. Silence data was created by randomly sampling white noise with low amplitude and adding background noise in accordance with what was previously mentioned.

The data processing which has been described above is also summarized in table 5.2.

Table 5.2: Summary of the data processing which has been done in this project, and the goal which the methods intend to solve. The factors/parameter values used for the data augmentation and noise overlaying are also presented in the rightmost column.

Goal	Solution	Factor
Expand dataset from only containing one word segments.	Adding LS data	-
Adjust the model for automotive audio environment.	Adding noise	SNR = [0,20]
Teach the model to find keywords in all parts of an audio segment.	Shift alteration	±100 ms
Adjust the model for different speakers, pronunciations and other speech variations.	Pitch alteration	[-10,10]
	VTLP	[0.9,1.1]
	Volume alteration	[0.8,1.2]
	Speed alteration	[0.9,1.1]

5.3 Model training

The data is divided into three parts: train, test, and validation data, with a fraction of 80, 10, 10 %, respectively. The training data is used to train the models, and the progress is evaluated on the test data throughout the training. When the training is completed, the final model is evaluated on the until-now unseen validation data. Therefore, the performance metrics computed on the validation data are considered the most reliable since they will take into account the generalization ability of the model. Further on, all performance metrics presented are computed on the validation dataset unless anything else is specified.

As described in section 2.3.3, countless parameters can be tuned when training an ANN. Based on previous results, some of these were fixed, while others were tuned between different values in order to find the most suitable combination. All studies presented in chapter 3 used ReLU as activation functions until the output layer, where softmax was used, and this project uses the same concept. The batch size is fixed at 256, while the initial learning rate will be tuned between a few values. This choice is based on the fact that these two parameters correlate with each other, so it is only necessary to vary one [65]. The learning rate is also reduced by a factor of 10 on plateaus to iteratively converge to optimal network parameters.

The upper limit of training epochs is fixed at 200, and early stopping is implemented to avoid overfitting. The limit of 200 epochs is found to be more than sufficient for the training to converge. To avoid overfitting, the models also include dropout layers. The fraction of network connections to drop will be varied within the parameter tuning since dropout has rarely been used, or at least presented, in previous studies. There are multiple optimizers that were described in section 2.3.3. Both Adam and SGD with momentum have been used in previous studies, and RMSProp is also found to be a promising method. Therefore, in this project, these three optimizers will be considered in the final models.

CE loss is chosen throughout the whole project, mainly due to its frequent usage in previous studies and the fact that MP loss and LOVO loss have yet to have simple implementations to our knowledge. The keywords are assigned doubled weights in the loss function compared to the non-keyword samples, to emphasize their importance. This can be compared to instance-based transfer learning described in section 2.2.1.2. It also showed improved performance compared to using uniform weights.

5.4 Choice of feature extraction method

There are several feature extraction methods available for KWS modeling and many possible parameters for each. In order to select the method to use in the final model tuning, a smaller examination was made for each setting. A simpler experimental setup was used compared to the full task described further on, in order to reduce run times and since it was found sufficient for this purpose.

The ten keyword classes of the GSC dataset, seen in table 5.1, were selected, and the remaining words formed an "other" class. Half the content was sampled for each word, and an additional 5000 samples were added from LS. Lastly, 5000 "silence" samples were added. Noise was added to all samples with an SNR of 10 dB, and no further augmentation was performed.

Four feature extraction methods were examined: MFCC, LPCC, mel-spectrogram, and mel-filterbank. All features can be considered time series, so delta features, see section 2.2.2.7, could be applied to any of them. However, MFCC and LPCC can specify the number of coefficients explicitly, while filterbank and spectrogram select the number of filters or the FFT window size, respectively. Nevertheless, in this experiment, for each extraction method, the two parameters described above—whether to include delta features and the number of features or FFT window size—will vary between two values. Different window sizes and stride lengths were also examined, as this had an impact on performance in [26]. The settings are tested using the TC-ResNet model with architecture as in [46]. The model training uses an Adam optimizer with an initial learning rate of 0.01, a dropout fraction of 0.5, and CE loss.

The preliminary experiment resulted in MFCC with 13 features and deltas, i.e., 26 features per time step in total. The best results were achieved with a 25 ms window size and a 10 ms stride length, which resulted in a feature size of $(99, 26)$ for each one-second audio segment. The results from the experiments are presented in appendix A.3.1. MFCC achieved substantially larger accuracy compared to the other feature extraction methods. The choice of this method is also in line with many previous KWS studies, as seen in chapter 3. All settings for MFCC yielded quite similar performance, but since it has previously been found that delta features have a positive effect on performance, a slightly more complex feature set was chosen. The increase in complexity, i.e., number of operations and MFCC size, here is also considered very small in comparison with the complexity of an ANN. An increase in feature number could also be beneficial for a more complex problem, such as the final KWS problem, which will be described in the following section.

5.5 Evaluation of KWS models

In this section, the methodology behind finding the final model for the task of this project is described. This includes a description of the KWS problem that is considered, the model architectures and how training parameters have been tuned, as well as how the models have been evaluated towards the proposed area of use.

5.5.1 Final data settings

The following data settings were selected to provide the best conditions for the KWS models to learn the feature-label relations: All available data were used for the 10 keywords, resulting in a total of 38 546 one second audio segments. From the non-keywords of GSC, 1400 samples were selected, summing to 35 000 in the "other" class. An additional 35 000 samples were added from LS, and finally 20 000 "silence" samples were added for a total of 128 546 data points.

Moreover, noise with the same settings as in 5.2, with an SNR in the range of 0 to 20 dB was added to 80 percent of the data. This SNR range was considered reasonable based on the findings in chapter 4, especially since it is possible to add noise reduction techniques. A wider SNR range, as well as some instances without noise, is also considered a beneficial variation in order to develop a robust model. To engender more variations, 20 % of the data was augmented with VTLP, speed, pitch, volume, and shift with the factors described in section 5.2. This extended the keyword samples by about 7000 additional samples, while keeping the number of "other" instances intact by performing this augmentation in-place.

5.5.2 Model architectures

This section contains information about the model architectures for the three chosen models. Several other models were also investigated throughout the project, e.g., a HMM-GMM. In this implementation, twelve different HMM-GMM models were created, one for each class, and the training data belonging to each class was given to the corresponding model. By feeding this into the models, the transition probability matrix and emission probabilities could be estimated for each class. In the evaluation, the new data is given to each of the twelve models, and the one that is the best fit will be the predicted class for the data point in question. The Python package `hmm_learn` was used to implement the above-mentioned method [92]. The results from the HMM-GMM were unsatisfactory with an accuracy of about 70%, which led to a shift of focus to ANN models. The performance is difficult to put into context as HMM-GMM models are presently uncommon for KWS nowadays and therefore has not been applied widely to this task. However, the low accuracy is supported by [93], where it is stated that HMM-GMM models have a limited performance due to inability to produce non-linear models.

The ANN models were implemented with `Tensorflow` and `Keras`, which have all types of network layers implemented [94, 95]. The final selected models were a TC-

ResNet, a DSCNN, and an Att-RNN, which will be described in detail below.

5.5.2.1 TC-ResNet

TC-ResNet models have been discovered to be well-performing for KWS regarding performance and complexity compared to models of similar size [43,46,96]. The TC-ResNet model used in this project was originally implemented by Hyperconnect [46], which uses the CNN architecture with shortcut connections from [56]. The version used in this project is TC-ResNet8-1.5, which has an initial convolution layer with 24 kernels, followed by three residual blocks with 36, 48, and 72 kernels, respectively. An illustration of the architecture is displayed in figure 5.1.

The residual block performs two processes in parallel and combines them at the end. The larger process has two 1-D convolutions in the temporal dimension with a kernel of size 9. Between these two layers are a batch normalization and ReLU activation layer, and after the second convolution, another layer of batch normalization follows. The second process is skip connection, which consists of a 1-D convolution, batch normalization, and ReLU activation layer. After the three residual blocks, the model is completed by average pooling, and a FC, softmax output layer.

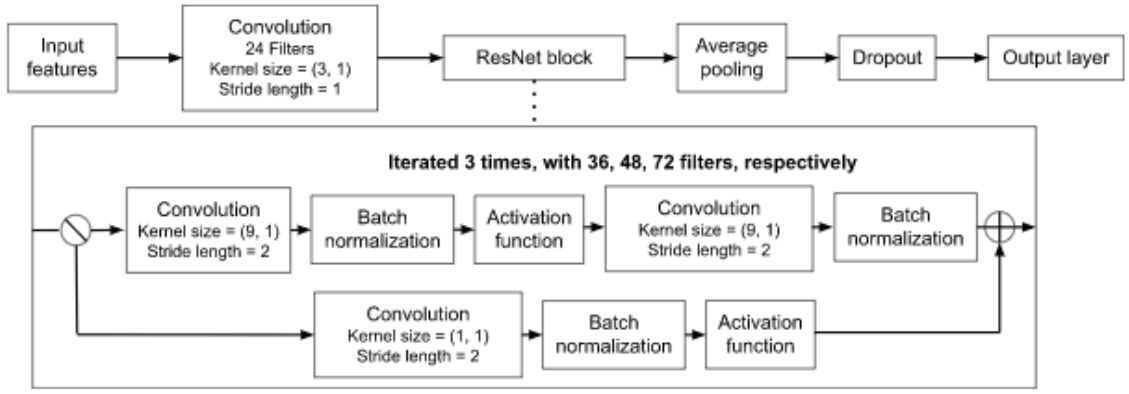

Figure 5.1: The architecture for the TC-ResNet model with 24, 36, 48, and 72 filters. The dimensions for the two-dimensional parameters are (Time, Feature).

5.5.2.2 DSCNN

Based on the work and promising results of DSCNN models in [43, 44], a model with a similar architecture was implemented in this project. The model architecture starts with an ordinary convolutional layer. This is followed by a DS convolutional block, which contains a separable convolutional layer, a normalization layer, and a ReLU activation function. It is repeated a number of times within the model. After the last DS block, an average pooling and dropout layer are stacked before a final FC output layer with a softmax activation function finishes the model.

To find the most suitable model configuration, 16 different settings were examined. These were all combinations of changing four parameters; the number of CNN filters, the CNN stride length, the number of DS filters, and the number of DS layers.

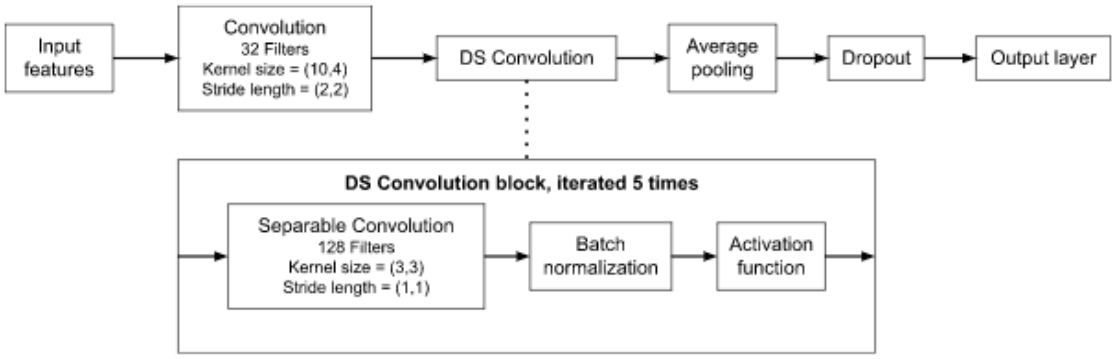

Figure 5.2: The architecture for the DSCNN model. The dimensions for the two-dimensional parameters are (Time, Feature).

The kernel sizes and DS stride length were fixed, based on the architecture in [44]. All models were evaluated on the full problem with the final data configurations described above. The model architecture tuning resulted in a final model with 32 convolutional filters, stride length (2,2), 128 DS filters, and five DS layers. The full results from this are presented in appendix A.3.2 and the final model architecture is presented in figure 5.2.

5.5.2.3 Att-RNN

The third and last model implemented is an Att-RNN model, which achieved the highest accuracy in table 3.1 and thus shows great promise for this task as well. Note that the model uses multiple heads, but will throughout the report be denoted Att-RNN despite this. The model architecture is based on the one presented in [45], but with some alterations. Two convolutional layers are used, with batch normalization after each of them. After this, the data is processed through two bidirectional RNN layers with 64 GRU units each. The output from this is given as the values and keys to an attention layer, while the query vector is obtained through the output from a FC layer with the same number of neurons as the input dimension. The model is finished off with a flattening, dropout, and FC output layer with softmax activation function. A difference from the original model in [45] is that the FC layer after the attention layer has been removed, since it will greatly reduce complexity and was found to not have any significant effect on performance in this specific task.

Different numbers of filters for the first convolutional layer, as well as other kernel sizes and stride lengths for both convolutional layers, were attempted. In addition, the number of heads was altered between two and four, to investigate the effect of this on performance. This resulted in an evaluation of 16 different model architectures. The most performant model included 32 convolutional filters, a kernel size of (5,1), stride length (1,1), and four attention heads. The results from this are presented in full extent in appendix A.3.2 and the final model architecture is displayed in figure 5.3.

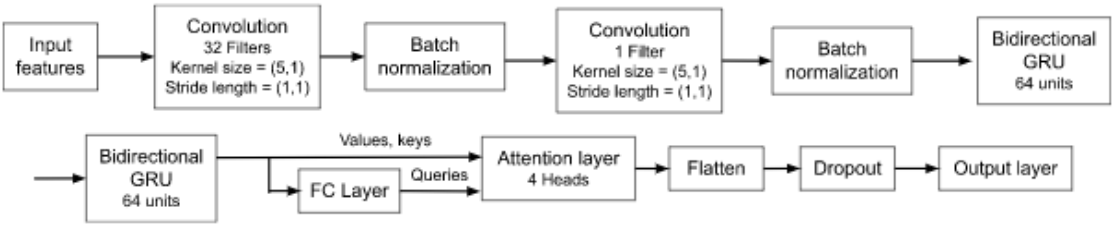

Figure 5.3: The architecture for the Att-RNN model. The dimensions for the two-dimensional parameters are (Time, Feature).

5.5.3 Parameter tuning

There is an extensive configuration space when selecting a model's parameters. Parameter tuning is an iterative process where experiments are conducted and the results provide the fundamentals for the next experiment. For an easier overhead, tuning process, and experiment tracking, the Python package `talos` was used [97]. Three different training parameters, dropout fraction, learning rate and optimizer, were tuned between 3 values, resulting in 27 configurations for each of the three chosen models.

5.6 Prototyping a KWS system

As previously stated, Python and its ecosystem was used for implementation. While this has many benefits, one disadvantage is that Python is natively single-threaded, which means that a real-time KWS system built in Python generally cannot record audio, create features, and perform inference at the same time. As a workaround, the system alternates between recording and processing data, with some audio lost between recordings, as illustrated in figure 5.4.

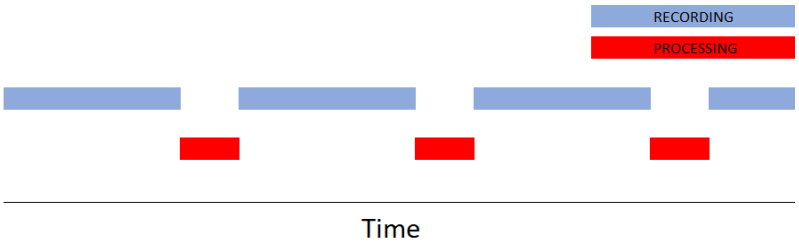

Figure 5.4: Visualization of how a thread alternates processes from recording audio to processing the audio and performing model inference in a real-time KWS system. The gaps in the "Recording" graph illustrate lost audio, which could impact the model's ability to correctly predict the speech.

To overcome this limitation and reduce latency and audio losses, the demonstration system developed here records a two-second audio segment, which is then partitioned into several overlapping segments of one second each for processing and inference. This ensures that at least one segment includes the entire spoken word. All predictions will output a probability for each class from the softmax activation function.

A probability threshold is used for all keywords, and the keyword with the highest probability that exceeds this bound will be returned. If no keyword is predicted with a higher probability than this threshold, the system will return the class with the highest probability out of "other" and "silence". The reason for this probability threshold is to make the system more or less biased against non-keywords and by extension increase performance. The effect of the prediction threshold on performance is therefore something that will be examined.

The recording in the demo system is initiated by pressing a button, and there are twelve buttons corresponding to each keyword and non-keyword class. This allows the system to store the true and predicted words for evaluating the model's performance in various scenarios, which will be described further on. However, in a real-world application it will, of course, not be necessary to have one button for each keyword. It is important to note that the demo system runs on a laptop without known noise reduction and omnidirectional microphones, which means that it picks up sounds from all directions equally. In addition, in a real automotive environment, beamforming microphones are used to increase SNR and hence improve performance. Therefore, the results from the demo should be viewed as the lower limit of expected performance.

The standard setup for KWS, especially on GSC, is to classify one-second segments. It is possible to classify a word as it is spoken, i.e., using audio segments shorter than a second. This approach would work for most of the well performing models, as seen implemented by Google's research team [98]. However, for models that implement bidirectional layers, such as the Att-RNN implemented in this project, streaming mode is not possible. This is because the model requires the complete features as the bidirectional layer uses an RNN layer, such as LSTM or GRU, to traverse the features forwards and backwards, thereby requiring the entire one second segment [50, 98]. This could be important to take into consideration when choosing the final model, in case the goal is to implement it in streaming mode. Out of the three models described above, Att-RNN is the only one not suitable for streaming mode.

The KWS task in this project essentially only considers one-word audio segments, apart from the LS data in the "other" class. This can be extended to detection of phrases or sentences with several words. Under the assumption that each word is below one second, the method of dividing the longer speech sequence into overlapping segments of one second each can be kept. A prediction is made on each of these segments, and the keyword with the highest probability above the probability threshold is kept. By assuming that no word is said twice in a row, which is unlikely in a real sentence, a keyword that is found can be disregarded in the prediction of the next word. One could also incorporate an adaptable probability threshold for the keywords by comparing them with phrases that the system is expecting. For example, a phrase that a KWS system could expect in an automotive environment is "left window up". After recognizing "left" and "window", the threshold for the last keyword could be lowered, and the system could only look at the prediction probabilities for "up" and "down". When finding either of these keywords with a high

enough probability, the system could terminate, as it does not expect the phrase to continue.

The initial demo version described above makes use of a fixed audio recording length. A more suitable implementation for real-world usage is to make use of a voice activity detection system, which will activate the KWS system when speech is recognized. An equivalent functionality is a "push-to-talk" button, which is used to manually start and stop the KWS system. The latter is implemented in an updated demo version, which enables a dynamic recording length.

5.7 Performance - complexity analysis

The parameter tuning is evaluated solely on the validation dataset. For the final three models, the comparison is extended to include more tests and measurements. Three separate tests are conducted. The first is to perform prediction on 10 000 speech files from the LS dataset. Each file will be labeled either "other" if no keyword occurs or with the keyword that occurs in the speech, based on the transcript corresponding to the file.

The second test is predictions made on two second recordings made in an ordinary office environment. 1564 recordings were made, divided among 15 different speakers. The number of instances in each of the twelve classes was evenly balanced. The third test is the same as the previous one, but done in a moving car, more specifically a Volvo XC60. The car recordings are done in six different settings, with approximately equally many recordings for each setting. In total, 1012 two-second recordings were made. The settings are all combinations of city driving and highway driving with no extra background noise, background music, or background speech. All recordings were made using a laptop microphone, which is considered worse than the real microphones in a vehicle due to the fact that microphones in the car are optimized to improve SNR on the speech signals coming from the passenger or the driver. The number of instances for each class are relatively balanced between the twelve classes.

A final model is chosen based on both performance and complexity. The goal is, as mentioned previously, to select the most performant model with low enough complexity to be able to function in real-time.

5.7.1 Performance analysis

To compare the model configurations and training parameter tuning, performance metrics on the validation dataset are used. A random seed is applied to make sure the validation dataset is the same in each run so that the results are comparable. In addition to ordinary accuracy, two other performance metrics are developed and used for evaluation. The first, mean keyword accuracy (MKA), is the accuracy among all keyword instances, i.e., removing all data belonging to the classes "other" and "silence". This metric will be a measure of how well the model learns to distinguish

between all keywords. To complement this, a metric for the ability to distinguish between any keyword and non-keywords, called keyword detection accuracy (KDA), is implemented. This is simply a binary accuracy computation, where all keyword classes form the first class while "other" and "silence" form the second class.

In addition to the three performance metrics above, macro-averaged precision and recall will be used when comparing the performance of the final three models. The formulas for precision and recall are presented in equation 5.1. In these formulas, TP = true positives, TN = true negatives, FP = false positives, and FN = false negatives. Here we consider the positives to be a single class containing all keywords and the negatives to be a single class containing all non-keywords. Basically, macro-average means that precision and recall will be the arithmetic mean for all individual classes.

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP + FP} \\ \text{Recall} &= \frac{TP}{TP + FN} \end{aligned} \tag{5.1}$$

To evaluate the performance in more detail, class accuracies are used to investigate the models' ability to correctly classify the different words. These values are computed by dividing all correct predictions of the specific class by all occurrences of that class. A confusion matrix is also a useful figure with a similar purpose. It shows prediction quality through a table of predicted labels compared to the true labels. An example can be seen in table 5.3.

Table 5.3: Confusion matrix with three classes: 0, 1, and 2. The true labels for a class is distributed along its row, while the distribution along the columns depend on the model predictions. Thus, the upper-left to lower-right diagonal represent the correctly predicted labels.

True label \ Predicted label	0	1	2
0	14	2	0
1	0	27	5
2	3	2	22

5.7.2 Complexity analysis

To investigate complexity, the model size is used, i.e., the number of parameters in the model, as well as the number of multiplications done in one prediction by the model. The number of parameters and multiplications in a model depends on the number of layers and the size of each layer. In practice, the parameter and multiplication count are computed using `Tensorflow` and the `profiler` package, respectively. However, the following section contains theoretical computational details behind the number of parameters and multiplications in different types of ANN layers. Lastly, the complexity of MFCC feature extraction is briefly discussed.

5.7.2.1 Complexity of different ANN layers

Different ANN layers will have different complexity due to discrepancies in size and functionality. Below are theoretical details behind the complexity of different layers used in the three ANN models described above.

FC layers

FC layers have $p \cdot c + c$ number of parameters, where p is the number of neurons in the previous layer and c is the number of neurons in the current layer. The number of multiplications in this type of layer will be equal to the number of weights, i.e., $p \cdot c$, since each weight is applied only once.

Convolutional layers

In a convolutional layer, the number of parameters are $w \cdot h \cdot f_1 \cdot f_2$, where (w, h) is the size of each kernel and f_1 and f_2 are the number of filters in the previous and current layer, respectively. In a convolutional layer, each kernel, and thus weight, is applied multiple times, namely the same number of times as the output size (W, H) from the layer. This leads to $w \cdot h \cdot f_1 \cdot f_2 \cdot W \cdot H$ number of multiplications for each convolutional layer.

DSC layers

In the case of a DSC layer, the convolution is divided into two parts; depthwise- and pointwise convolution. This leads to a reduced number of parameters and multiplications. More specifically, the depthwise convolution will contribute with $(w \cdot h \cdot f_2)$ parameters and the pointwise convolution with $f_1 \cdot f_2$ parameters, summing to $(w \cdot h + f_1) \cdot f_2$ parameters. The total number of multiplications will be $(w \cdot h \cdot f_2 \cdot W \cdot H) + (f_1 \cdot f_2 \cdot W \cdot H)$, which is a substantial reduction in operations compared to an ordinary convolutional layer. This is the main benefit of DSC layers.

GRU layers

Another layer used in this project is the GRU, and the math behind this type of layer was presented in equations 2.14 and 2.15. Here we note that there are three parameters $\mathbf{W}, \mathbf{U}, \mathbf{b}$ and each occur three times. The $\mathbf{W}$ matrices will be of size (v, u), where v is the input feature size and u is the number of units in the GRU. The matrix $\mathbf{U}$ will have size (u, u) and $\mathbf{b}$ has size u, leading to the total number of parameters $3(u^2 + vu + u)$. This number will be doubled in the case of a bidirectional GRU layer. All parameters are applied to each part of the input, leading to $3(u^2 + vu + u) \cdot T$ number of multiplications, where T is the input sequence length, for an ordinary GRU layer.

Attention layers

An attention layer with multiple heads consist of four $\mathbf{W}$ matrices, as seen in equation 2.17. The matrices $\mathbf{W}_i^Q$ have dimension (d_{model}, d_k), $\mathbf{W}_i^K$ have dimension (d_{model}, d_k), W_i^V have dimension (d_{model}, d_v), and W^O have dimension (hd_k, d_{model}) [55]. Since $i = 1, ..., h$ and in this case $d_k = d_v = d_{\text{model}}/h$, the total number of parameters will be

$$h(3d_{\text{model}}d_k) + hd_kd_{\text{model}} = 4d_{\text{model}}^2,$$

where d_{model} is the output dimension. Each of these parameters are applied once, and in addition there are $2d_{\text{model}} + 1$ multiplications in each attention computation; see equation 2.16. This computation is done h times which results in a total of $4d_{\text{model}}^2 + 2hd_{\text{model}} + h$ multiplications in this layer.

Other layers

There are also other types of layers that have been used by the models in this project, i.e., input layer, pooling layers, normalization layers, dropout layers, and flattening layers. These do not have any learnable parameters and make use of very few multiplications in comparison to the previously mentioned layers. Therefore, these can often be neglected in the complexity analysis of ANN.

5.7.2.2 Complexity of feature extraction

Producing MFCC features involves several steps, which were explained in section 2.2.2. The FFT has a complexity of $O(MN\log_2(N))$, where N is the window size of the Fourier transform and M is the number of windows that the one-second speech segment is divided into [99]. In this project $N = 512$ and $M = 99$. The log computation will add some complexity, while re-scaling and spectrogram creation have low complexity and can therefore be neglected. The last computationally heavy part of the MFCC extraction is the DCT, which has complexity $O(MN^2)$, but can be reduced to $O(MNlog(N))$ by factorising the computation in a similar manner as the FFT algorithm [100]. The resulting complexity of the feature extraction is thus $O(MN\log_2(N))$. Complexity can therefore be decreased by changing window size or stride length, i.e., altering M, or by changing N. The feature size will also affect the size and complexity of the ANN, and could therefore also be used to lower complexity. Although feature extraction is assumed to account for a negligible part of the total system complexity in comparison to model prediction, and this hypothesis will be investigated further using computational times for feature extraction and prediction.

6

Results

In this chapter, the results for the three chosen models are presented, both in terms of performance and complexity. Due to major difficulty in recognizing silence in new recorded audio samples, this class has been added to the "other" class in all evaluations except for the one done on the validation data. This difficulty implies that too much white noise was added to the silence samples, which rendered this data too different from true silence. This is not considered a problem, though, since the most important part of the task is the distinction between different keywords and non-keywords.

6.1 Performance evaluation

The parameter tuning for the three chosen models is presented in appendix A.3.3. Noticeable is that there were large variations among the results for the Att-RNN model, where many of the parameter combinations rendered the model completely useless. The optimal parameter combination for TC-ResNet was a dropout fraction of 0.7, an initial learning rate of 0.01, and Adam as the optimizer. For DSCNN, this combination was having the dropout at 0.3, the learning rate at 0.01, and RMSProp, while 0.7, 0.01, and SGD with momentum were optimal for Att-RNN. While a learning rate at 0.01 seems to be the most suitable choice, no other parameter choice was found to be a universally optimal choice regarding optimizer and dropout for all model types. This shows the importance of always investigating and conducting parameter tuning when implementing a new type of model or ML task.

Table 6.1: Performance and complexity for the final three models. The performance metrics are based on the validation dataset. Size and Mult represent the number of parameters and multiplications, respectively.

Model	Performance					Complexity	
	MKA	KDA	Accuracy	Precision	Recall	Size [K]	Mult. [M]
TC-ResNet	89.80	96.42	95.34	97.38	94.87	152.7	3.22
DSCNN	87.35	95.44	94.28	96.95	93.56	80.6	16.35
Att-RNN	87.55	95.78	94.21	96.71	94.52	244.5	2.34

The performance results for the best parameter combination for each of the three models are summarized in table 6.1. From this, it is clear that TC-ResNet is the most performant model out of these three for this particular task. Comparing the results with those in chapter 3, one can note a slight decrease in performance for the models presented here. This is not unexpected since this task is considered more difficult than in most previous studies. Background noise, data agglomeration, and data augmentation are increasing the disturbance in the data, which could explain these results. On the other hand, these disturbances should yield an increase in performance in a real-world application.

Table 6.2: Performance for the models in the car evaluation, using a probability threshold of 10% for the keywords. The metrics were computed with the "silence" and "other" classes joined together. The highest value in each row is marked in bold.

Metric	Setting	TC-ResNet	DSCNN	Att-RNN
Accuracy	City	**86.98**	76.92	79.49
	Highway	**80.04**	74.11	61.07
	Total	**83.71**	75.72	70.29
MKA	City	**91.49**	83.69	82.51
	Highway	**81.09**	77.78	60.43
	Total	**86.51**	80.95	71.48
KDA	City	**89.15**	86.39	86.59
	Highway	85.57	**86.96**	73.72
	Total	**87.46**	86.77	80.16
Precision	City	**92.99**	89.33	92.77
	Highway	**94.64**	91.42	91.40
	Total	**93.78**	90.35	92.15
Recall	City	94.09	**95.04**	91.02
	Highway	87.71	**93.14**	75.59
	Total	91.01	**94.20**	83.31

Regarding the real-world performance, one can study the results in tables 6.2 and 6.3, where the performance metrics from the recordings made in a car and in an office environment are presented. TC-ResNet continues to outperform the other two models, and the difference in performance is larger than previously seen. This implies that the TC-ResNet is a much more robust model that can handle new and different data better than the other two. These results also infer the importance of multiple tests in the evaluation process. This has been lacking in many previous studies, e.g., [43–45, 47, 68], and is something that should be taken into consideration when evaluating those results.

Looking at table 6.2, one can notice a significant decrease in performance in a

highway environment compared to city driving for most metrics and models. In chapter 4, a hypothesis was presented that car noise might not have a significant impact on performance due to energy peaks at different frequencies for car sound and speech. But, the results argues against this hypothesis. However, as speech and music were included in the noise, that might be a factor causing the declined performance. An idea could be to use the findings in chapter 4 to tune the noise reduction in an automotive environment. In addition, the built-in microphones in a vehicle are directed to capture speech from the driver and passenger and have improved signal processing that is adapted for the specific environment. Therefore, one could expect improved results for an embedded KWS system compared to the prototype system. The declining performance in a highway environment could also imply that even more noise should have been added to the training data. It could also result from a difference between artificially added noise and real noise, which was found to have a significant negative effect on performance in [78].

Table 6.3: Performance for the models in the office evaluation, using a probability threshold of 50%. The metrics were computed with the "silence" and "other" classes joined together. The highest value in each row is marked in bold.

Metric	TC-ResNet	DSCNN	Att-RNN
Accuracy	**92.33**	89.83	89.19
MKA	**94.39**	91.58	91.12
KDA	**95.01**	92.20	93.73
Precision	**96.55**	96.28	96.07
Recall	**97.57**	94.39	96.51

The performance in a real-life environment with low background noise is quite high for all models, as seen in table 6.3. Accuracy is lower than that for the validation data, while the other metrics are comparable or even improved in this test. Since the balance between the classes is different in the validation data and recorded data, one should be careful when doing a direct comparison between these results. Especially the "other" class constitutes a much larger fraction of the validation dataset, leading to improved accuracy. However, the results show that the models perform well in a real-world application with low disturbance and prove once again that TC-ResNet is the most suitable model for this purpose. In general, all models had significantly higher performance in the office environment compared to the car in regards to accuracy and recall, while precision was slightly decreased. High precision is important in a KWS system, as we wish to keep the number of false positives low, i.e., minimize misclassifying a non-keyword as a keyword. Though the importance will depend on the type of system that is in use. In the case of a system that is turned on by pressing a button, one can expect very few non-keywords, and thus precision does not need to be emphasized in the same manner as for a system that is always turned on.

The results from the evaluation of coherent speech from the LS data are shown

in table 6.4. A clear decrease in performance is observed compared to the other results. This indicates that neither of the models will work particularly well as a home assistant that is always on, as it will recognize keywords when no keyword is uttered. In fairness, though, the keywords used in this project are very short and often not significant in coherent speech, which they are part of in the LS data. This will greatly increase the difficulty for the models. On the other hand, the models, at least TC-ResNet, perform relatively well for the new recordings made in this project, as seen in the results mentioned above. Therefore, the system should be suitable for an application where it is turned on and off by the press of a button and when the speech is only intended as commands to the system.

Table 6.4: Performance for the models in the LS evaluation, using 10 000 audio files and a probability threshold of 50%. The metrics were computed with the "silence" and "other" classes joined together. The highest value in each row is marked in bold.

Metric	TC-ResNet	DSCNN	Att-RNN
Accuracy	**56.01**	53.60	54.31
MKA	13.42	**17.40**	14.14
KDA	**61.87**	61.22	60.85
Precision	**48.53**	47.79	46.69
Recall	29.06	**37.74**	31.60

6.1.1 Effect of the prediction threshold on performance

In figure 6.1, graphs of accuracy, precision, and recall are presented for different prediction thresholds for the office evaluation. The accuracy graphs almost resemble parabolic equations, with maxima around the threshold between 30 to 70 percent. Outside of this threshold range, a larger percentage of the keywords are mistaken for another keyword or a non-keyword. When the threshold level is too high, a larger part of the keywords will be mistaken for other words, as the certainty of the keywords is insufficient to reach the threshold level. On the other hand, when the threshold is too small, the wrong keyword might be detected, especially between similar keywords such as "on" and "off". In other words, these maxima and threshold boundaries could argue towards having a threshold in the range of 30 to 70 % in a real-world KWS system. In previous works on a similar task, a fixed threshold was found to be optimal at 50% [101]. However, more sophisticated algorithms, such as keyword specific thresholding and sum-to-one score normalization could be used to produce dynamic and keyword specific thresholds to improve the ability to detect keywords [102].

The accuracy graphs for the in-car evaluation are quite different than the in-office graphs; see figure 6.2. One explanation for this can be the noise and level of SNR in an automotive environment, which is the main difference between these two tests. Here, a low probability threshold seems to be most beneficial for performance. This difference suggests that the probability threshold should be tuned for the task and setting at hand, e.g., the level of noise in the specific vehicle in which the system is implemented.

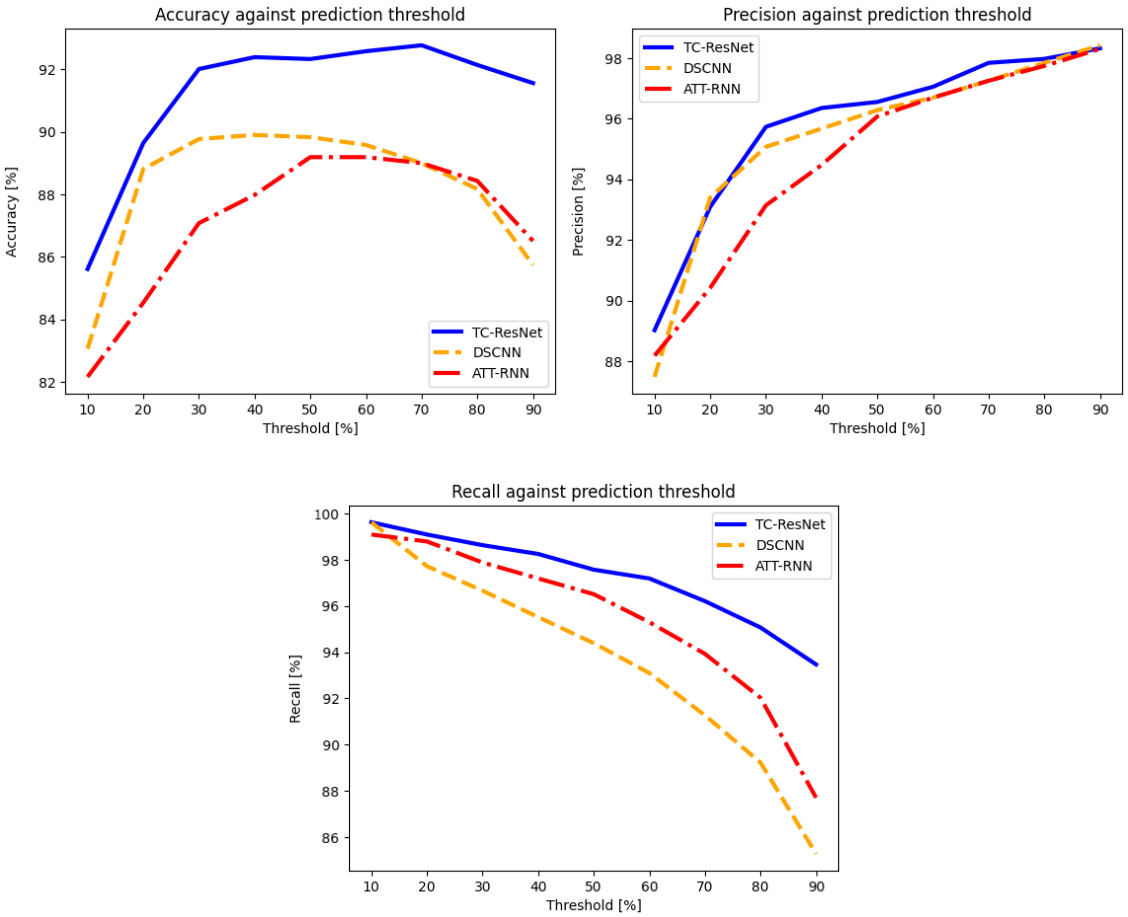

Figure 6.1: Accuracy, precision, and recall for an increasing prediction threshold for the in-office evaluation for the three different models. The metrics were computed with the "silence" and "other" class joined together.

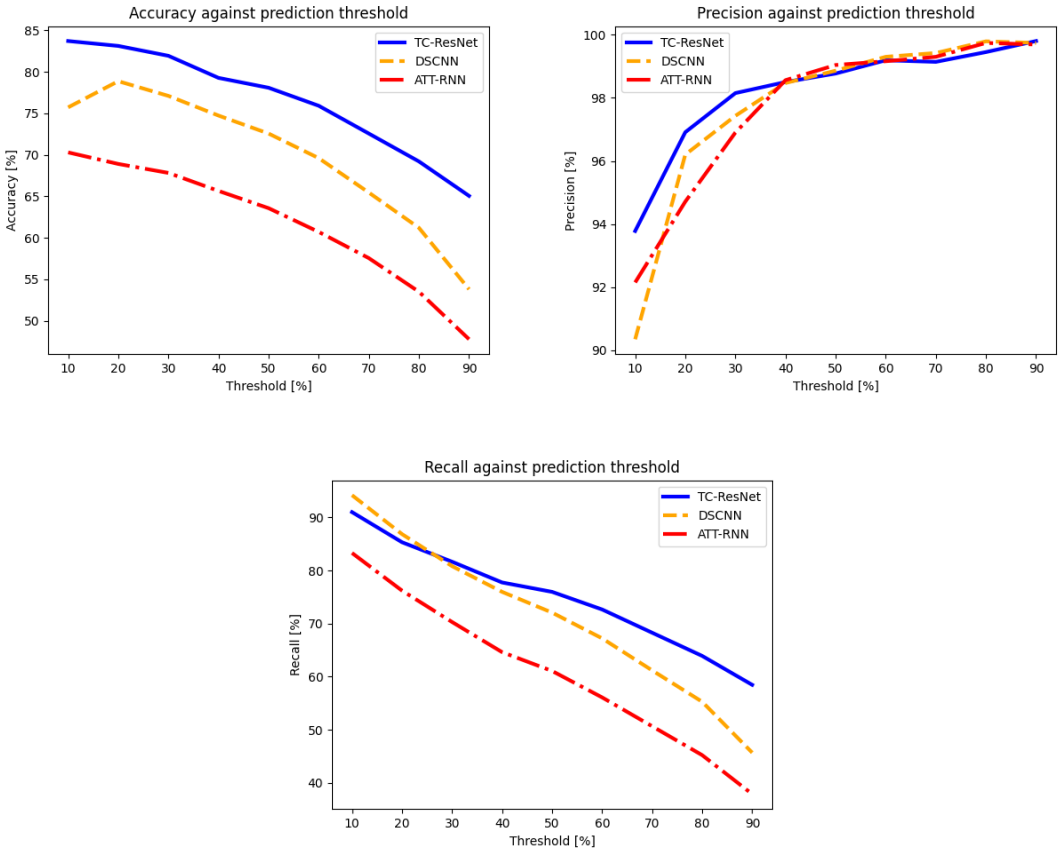

Figure 6.2: Accuracy, precision, and recall for an increasing prediction threshold for the in-car evaluation, for the three different models. The metrics were computed with the "silence" and "other" class joined together.

6.2 Complexity evaluation

The complexities for the three models are presented in table 6.1. The complexity is low for all models in comparison to the model complexities presented in chapter 3. This indicates that, from a complexity perspective, any of the models render a suitable final choice. One can notice that model size and number of multiplications do not always correlate, since the DSCNN is the smallest model but includes the most multiplications. This comes from the fact that convolutional layers, which are the foundation of DSCNN, make use of a few parameters that are applied many times.

The prediction time for a one-second audio segment on an ordinary 1.90 GHz CPU laptop is approximately 80 ms, and the MFCC feature extraction time takes around 4 ms. These computational times were computed using the Python `time` package. The results are in line with the hypothesis that feature extraction is only a negligible part of the total system complexity. It is found that the prediction time varies somewhat and is greatly impacted by the hardware and software that the model is running on. Therefore, the exact prediction time cannot be determined with certainty. Nonetheless, in most cases, the complete duration for the system to perform inference on a single one-second audio segment will be considered instant by the user since the limit for this is about 100 ms [103]. This assumes that the system is implemented in a streaming mode and instantaneously processes one-second-long audio segments. As mentioned, this comes with additional overhead in Python and should therefore be a topic for future development.

6.3 Further analysis of the final model

From the performance and complexity evaluation presented above, it is clear that TC-ResNet should be the final choice of model due to its superior performance and low complexity. In table 6.5, accuracy for all classes evaluated on the validation data, as well as in-office and in-car data, is presented. These results show which words the model performs well and poorly for. For example, "yes" and "right" are two words that the model recognizes relatively well in all settings, while "up" and "go" are more difficult. Interestingly, the difficulty for some words varies with the settings, e.g., "down". This could come from pronunciation differences or background noise having a diverse impact on different words. One can also notice a very high accuracy for the "other" class in the validation data. This strengthens the previously mentioned hypothesis that the difference in class balance could cause the higher accuracy for validation data compared to in-office data.

Diving into more detail, the confusion matrices in figure 6.3 give information about what classes are often mixed up by the system. Most commonly, the keywords are mixed up with the "other" class. Some keywords that are often difficult to distinguish between are "off" and "up", "no" and "go", and "on" and "off". From these results, it is clear that one should carefully choose the selected keywords for the KWS system

Table 6.5: Class accuracies for each class with the TC-ResNet model, where the "other" and "silence" classes have been combined.

	Class										
	Down	Go	Left	No	Off	On	Right	Stop	Up	Yes	Other
Validation	88.78	85.81	92.62	89.03	85.39	85.53	91.83	90.50	88.69	94.26	98.12
Office	89.63	95.35	98.45	94.81	95.38	98.47	96.92	93.89	86.36	94.85	81.30
Car	93.98	73.49	80.95	80.95	83.13	85.71	86.90	76.09	79.76	96.43	91.67

to achieve the most satisfactory results. It is not unlikely that for example Apple's "Siri" and Amazon's "Alexa" are keywords chosen from such an analysis.

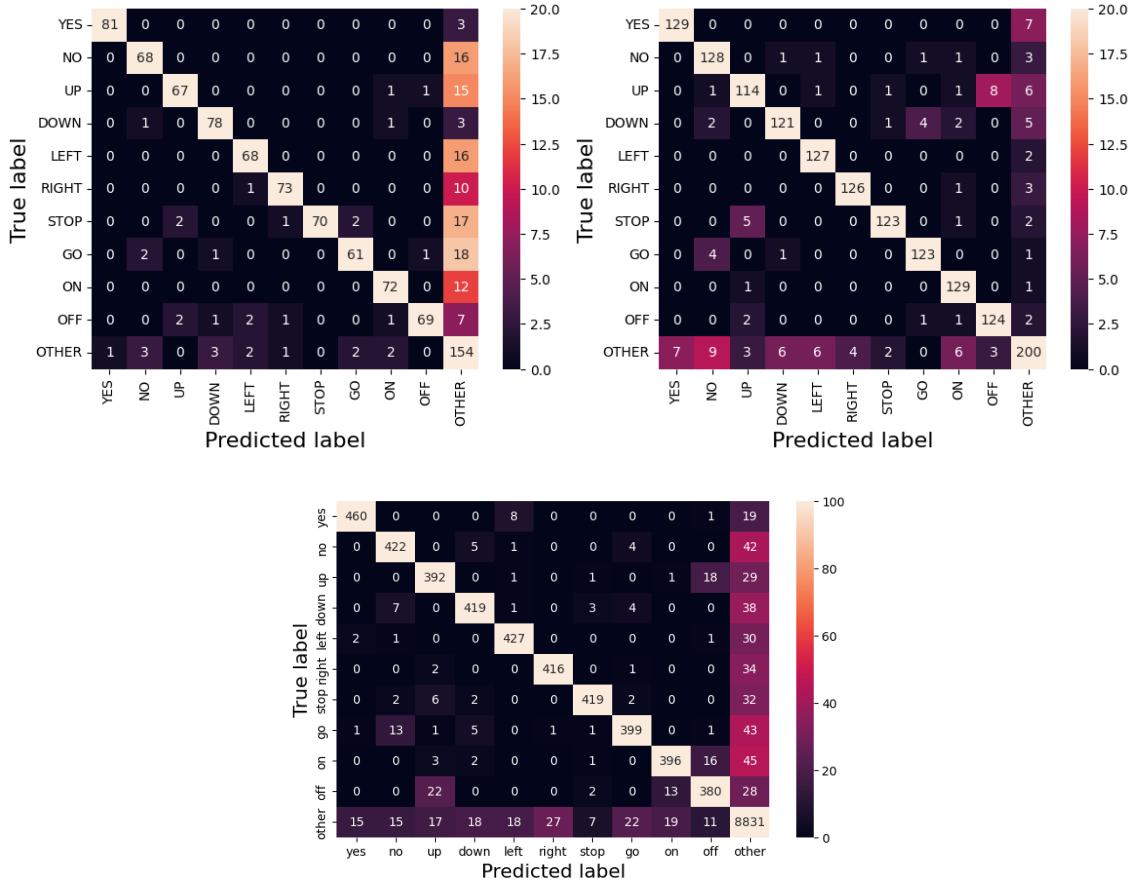

Figure 6.3: Confusion matrices for the TC-ResNet model, evaluated on in-car speech (top left), in-office speech (top right), and validation data (bottom). The cells are colored according to the corresponding scale. The "silence" and "other" classes have been joined together.

7

Conclusions and future work

This project had the main goal of developing a KWS system targeted for an automotive environment. Throughout the project, several ML models have been implemented, tested and compared, where a TC-ResNet model with MFCC features achieved the highest performance while maintaining low complexity. In the previous chapter, it was shown that this model performed well both on the validation dataset and in a low-noise real-life environment. The performance declined in an automotive environment with more noise but is still considered promising due to possible improvements regarding the microphone, noise reduction, and signal processing in a real implementation.

When evaluating the model on coherent speech taken from audio books, i.e., the LS dataset, it is found that the KWS system performs quite poorly with low accuracy as well as precision and recall. From these results, we conclude that the system developed here is not suitable as an ongoing home assistant but should rather be implemented with a starting button to initiate the KWS process. Commonly, a voice activity detection system is used to initiate the KWS system, which is something that should be attempted in future work. The evaluation process with multiple tests, including the LS test on coherent speech, is also quite unique and was found to show larger differences in performance than a simple evaluation on a validation dataset. Therefore, we conclude that this is a framework suitable for others to implement as well, as it will give more insight into the model's performance and robustness.

Aside from performance, complexity has also been an important factor in the development of the KWS system. The chosen TC-ResNet model has a model size and multiplication count on the lower end of the previous results presented in chapter 3, which also had the goal of developing a small-footprint KWS model. Therefore, there is certainty in the fact that the model developed here could work well in a real-time application. This is further supported by the prediction time undercutting 100 ms, which should definitely be improved using better software and hardware in a real-life implementation. A possibility for future work is to perform such an implementation. An implementation in a multithreaded programming language, e.g., C or C++, would enable a streaming mode for the model, which would increase the system quality.

In the results, it has been shown that the probability threshold implemented in the system is an important factor for performance and should therefore be carefully tuned for the targeted domain for optimal quality. A possible improvement for the future is to implement keyword specific thresholding and investigate the effect of this. In addition, the keywords that are chosen for the system could also have a substantial impact on performance and should thus be chosen wisely. The keywords chosen in this project are heavily based on previous studies and the GSC dataset. In order to further develop the system, more keywords are needed to be able to create short sentences of instructions targeted for the system. This requires extensive data collection and data labeling, which have not been possible within the scope of this project. To create a more comprehensive system, a suggestion would be to collect data on all phonemes and either train the ML model on these or on words created by combining the phonemes in different ways.

In summary, the aim of this project has been achieved since a well-performing KWS system with low complexity targeted for an automotive environment has been investigated and implemented. However, there are numerous improvements that can and should be explored before this system is ready to be put into real-world use, of which extensive data collection is the main key to success.

Bibliography

[1] López-Espejo I, Tan ZH, Hansen JHL, Jensen J. Deep Spoken Keyword Spotting: An Overview; 2022. Available from: https://doi.org/10.1109/ACCESS.2021.3139508.

[2] Kinnunen T, Li H. An overview of text-independent speaker recognition: From features to supervectors; 2010. Available from: https://doi.org/10.1016/j.specom.2009.08.009.

[3] Simpson M, Bruce R. Noise in America: the Extent of the Noise Problem. EPA; 1981.

[4] Bottalico P, Piper RN, Legner B. Lombard effect, intelligibility, ambient noise, and willingness to spend time and money in a restaurant amongst older adults; 2022. Available from: https://doi.org/10.1038/s41598-022-10414-6.

[5] Summers WV, Pisoni DB, Bernacki RH, Pedlow RI, Stokes MA. Effects of noise on speech production: Acoustic and perceptual analyses; 1988. Available from: https://doi.org/10.1121/1.396660.

[6] Berg, R E . Sound. Encyclopedia Britannica; 2022. [Cited 2023-04-11]. Online. Available from: https://www.britannica.com/science/sound-physics.

[7] Cambridge Dictionary. Audio. Cambridge University Press Assessment; 2023. [Cited 2023-04-13]. Online. Available from: https://dictionary.cambridge.org/dictionary/english/audio.

[8] Fry DB. In: Periodic and aperiodic sounds. Cambridge Textbooks in Linguistics. Cambridge University Press; 1979. p. 82–88. Available from: https://doi.org/10.1017/CBO9781139165747.008.

[9] Montrose MI. In: Appendix A: The Decibel; 1999. p. 291-3. Available from: https://doi.org/10.1002/047172310X.app1.

[10] Papadopoulos P, Tsiartas A, Gibson J, Narayanan S. A supervised signal-to-noise ratio estimation of speech signals. In: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE; 2014. p. 8237-41. Available from: https://doi.org/10.1109/ICASSP.2014.6855207.

[11] Zhang Z, Cummins N, Schuller B. Advanced Data Exploitation in Speech Analysis: An overview; 2017. Available from: https://doi.org/10.1109/MSP.2017.2699358.

[12] Kim C, Shin M, Garg A, Gowda D. Improved Vocal Tract Length Perturbation for a State-of-the-Art End-to-End Speech Recognition System; 2019. p. 739-43. Available from: https://doi.org/10.21437/Interspeech.2019-3227.

[13] Jaitly N, Hinton E. Vocal Tract Length Perturbation (VTLP) improves speech recognition; 2013. Available from: http://www.cs.toronto.edu/~ndjaitly/jaitly-icml13.pdf.

[14] Lin J, Kilgour K, Roblek D, Sharifi M. Training Keyword Spotters with Limited and Synthesized Speech Data. In: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE; 2020. Available from: https://doi.org/10.1109/icassp40776.2020.9053193.

[15] Sharma E, Ye G, Wei W, Zhao R, Tian Y, Wu J, et al. Adaptation of RNN Transducer with Text-To-Speech Technology for Keyword Spotting. In: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE; 2020. Available from: https://doi.org/10.1109/icassp40776.2020.9053191.

[16] Wu Y, Zhang R, Rudnicky A. Data selection for speech recognition. In: 2007 IEEE Workshop on Automatic Speech Recognition Understanding (ASRU); 2007. p. 562-5. Available from: https://doi.org/10.1109/ASRU.2007.4430173.

[17] Shimodaira H. Improving predictive inference under covariate shift by weighting the log-likelihood function; 2000. Available from: https://doi.org/10.1016/S0378-3758(00)00115-4.

[18] Schuller B, Vlasenko B, Eyben F, Wöllmer M, Stuhlsatz A, Wendemuth A, et al.. Cross-Corpus Acoustic Emotion Recognition: Variances and Strategies; 2010. Available from: https://doi.org/10.1109/T-AFFC.2010.8.

[19] Haeb-Umbach R, Watanabe S, Nakatani T, Bacchiani M, Hoffmeister B, Seltzer ML, et al.. Speech Processing for Digital Home Assistants: Combining Signal Processing With Deep-Learning Techniques; 2019. Available from: https://doi.org/10.1109/MSP.2019.2918706.

[20] Lee SK, Yu DJ. Measurement of reverberation time of a passenger car utilizing the wavelet filter bank; 2005. Available from: https://doi.org/10.1243/095440705X6686.

[21] Cremona C, Yang S, Dieleman L. Overview of feature extraction techniques. vol. 2; 2009. Available from: https://www.researchgate.net/profile/Christian-Cremona/publication/279176561_Overview_of_feature_extraction_techniques/links/55e5c83b08aecb1a7ccd5ada/Overview-of-feature-extraction-techniques.pdf.

[22] Pedersen P. The Mel Scale. [Duke University Press, Yale University Department of Music]; 1965. Available from: https://doi.org/10.2307/843164.

[23] Baken RJ, Orlikoff RF. Clinical measurement of speech and voice. Singular Thomson Learning; 2000.

[24] O'Shaughnessy D. In: Speech Communications: Human and Machine; 2000. p. 173-227. Available from: https://doi.org/10.1109/9780470546475.

[25] Cochran WT, Cooley JW, Favin DL, Helms HD, Kaenel RA, Lang WW, et al.. What is the fast Fourier transform?; 1967. Available from: https://doi.org/10.1109/PROC.1967.5957.

[26] Gouda SK, Kanetkar S, Harrison D, Warmuth MK. Speech Recognition: Keyword Spotting Through Image Recognition; 2020. Available from: https://doi.org/10.48550/arXiv.1803.03759.

[27] Sarangi S, Sahidullah M, Saha G. Optimization of data-driven filterbank for automatic speaker verification. Elsevier BV; 2020. Available from: https://doi.org/10.1016/j.dsp.2020.102795.

[28] Sahidullah M, Saha G. Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition; 2012. Available from: https://doi.org/10.1016/j.specom.2011.11.004.

[29] Gupta H, Gupta D. LPC and LPCC method of feature extraction in Speech Recognition System. In: 2016 6th International Conference - Cloud System and Big Data Engineering (Confluence); 2016. p. 498-502. Available from: https://doi.org/10.1109/CONFLUENCE.2016.7508171.

[30] Kim HS. Linear Predictive Coding is All-Pole Resonance Modeling. Center for Computer Research in Music and Acoustics, Stanford University;. [Cited 2023-02-19]. Online. Available from: https://ccrma.stanford.edu/~hskim08/lpc/.

[31] Deterding D. The Formants of Monophthong Vowels in Standard Southern British English Pronunciation; 1997. Available from: https://doi.org/10.1017/S0025100300005417.

[32] Dhanalakshmi P, Palanivel S, Ramalingam V. Classification of audio signals using SVM and RBFNN; 2009. Available from: https://doi.org/10.1016/j.eswa.2008.06.126.

[33] Chia Ai O, Hariharan M, Yaacob S, Sin Chee L. Classification of speech dysfluencies with MFCC and LPCC features; 2012. Available from: https://doi.org/10.1016/j.eswa.2011.07.065.

[34] Jayanna HS, Prasanna SRM. Fuzzy Vector Quantization for speaker recognition under limited data conditions. In: TENCON 2008 - 2008 IEEE Region 10 Conference; 2008. p. 1-4. Available from: https://doi.org/10.1109/TENCON.2008.4766453.

[35] Memon S, Lech M, Maddage N. Speaker Verification Based on Different Vector Quantization Techniques with Gaussian Mixture Models. In: 2009 Third International Conference on Network and System Security; 2009. p. 403-8. Available from: https://doi.org/10.1109/NSS.2009.19.

[36] Hossan MA, Memon S, Gregory MA. A novel approach for MFCC feature extraction. In: 2010 4th International Conference on Signal Processing and Communication Systems; 2010. p. 1-5. Available from: https://doi.org/10.1109/ICSPCS.2010.5709752.

[37] The HTK book version 3.4. Cambridge University Engineering Department; 2006. Available from: http://speech.ee.ntu.edu.tw/homework/DSP_HW2-1/htkbook.pdf.

[38] Eddy SR. Hidden Markov models; 1996. Available from: https://doi.org/10.1016/S0959-440X(96)80056-X.

[39] Jung Y, Park J. Scalable Hybrid HMM with Gaussian Process Emission for

Sequential Time-series Data Clustering; 2020. Available from: https://doi.org/10.48550/arXiv.2001.01917.

[40] Najar F, Bourouis S, Bouguila N, Belghith S. A Comparison Between Different Gaussian-Based Mixture Models. In: 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA); 2017. p. 704-8. Available from: https://doi.org/10.1109/AICCSA.2017.108.

[41] Daniel Jurafsky JHM. Speech and Language Processing; 2023. .

[42] Mehlig B. Machine learning with neural networks : an introduction for scientists and engineers. Cambridge (GB): Cambridge University Press; 2022.

[43] Rybakov O, Kononenko N, Subrahmanya N, Visontai M, Laurenzo S. Streaming Keyword Spotting on Mobile Devices. In: Interspeech 2020. ISCA; 2020. Available from: https://doi.org/10.21437/interspeech.2020-1003.

[44] Zhang Y, Suda N, Lai L, Chandra V. Hello Edge: Keyword Spotting on Microcontrollers; 2017. Available from: https://doi.org/10.48550/arXiv.1711.07128.

[45] de Andrade DC, Leo S, Viana MLDS, Bernkopf C. A neural attention model for speech command recognition; 2018. Available from: https://doi.org/10.48550/arXiv.1808.08929.

[46] Choi S, Seo S, Shin B, Byun H, Kersner M, Kim B, et al. Temporal convolution for real-time keyword spotting on mobile devices. Interspeech 2019. 2019. Available from: https://doi.org/10.21437/interspeech.2019-1363.

[47] Tang R, Lin J. Deep Residual Learning for Small-Footprint Keyword Spotting. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP); 2018. p. 5484-8. Available from: https://doi.org/10.1109/ICASSP.2018.8462688.

[48] Arik SO, Kliegl M, Child R, Hestness J, Gibiansky A, Fougner C, et al.. Convolutional Recurrent Neural Networks for Small-Footprint Keyword Spotting; 2017. Available from: https://doi.org/10.48550/arXiv.1703.05390.

[49] Chollet F. Xception: Deep Learning with Depthwise Separable Convolutions; 2017. Available from: https://doi.org/10.48550/arXiv.1610.02357.

[50] Bahdanau D, Cho K, Bengio Y. Neural Machine Translation by Jointly Learning to Align and Translate; 2016. Available from: https://doi.org/10.48550/arXiv.1409.0473.

[51] Olah, C. Understanding LSTM networks;. [Cited 2023-04-27]. Online. Available from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[52] Hochreiter S, Schmidhuber J. Long Short-term Memory. Neural computation. 1997 12;9:1735-80. Available from: https://doi.org/10.1162/neco.1997.9.8.1735.

[53] Cho K, van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, et al.. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation; 2014. Available from: https://doi.org/10.48550/arXiv.1406.1078.

[54] Dey R, Salem FM. Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks; 2017. Available from: https://doi.org/10.48550/arXiv.1701.05923.

[55] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al.. Attention Is All You Need; 2017. Available from: https://doi.org/10.48550/arXiv.1706.03762.

[56] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016. p. 770-8. Available from: https://doi.org/10.1109/CVPR.2016.90.

[57] Mathworks. Train Residual Network for Image Classification;. [Cited 2023-03-03]. Online. Available from: https://www.mathworks.com/help/deeplearning/ug/train-residual-network-for-image-classification.html.

[58] Boulila W, Driss M, Al-Sarem M, Saeed F, Krichen M. Weight Initialization Techniques for Deep Learning Algorithms in Remote Sensing: Recent Trends and Future Perspectives; 2021. Available from: https://doi.org/10.48550/arXiv.2102.07004.

[59] Sun M, Raju A, Tucker G, Panchapagesan S, Fu G, Mandal A, et al. Max-Pooling Loss Training of Long Short-Term Memory Networks for Small-Footprint Keyword Spotting; 2016. p. 474-80. Available from: https://doi.org/10.1109/SLT.2016.7846306.

[60] Kim D, Ko K, Han DK, Ko H. Discriminatory and Orthogonal Feature Learning for Noise Robust Keyword Spotting. IEEE Signal Processing Letters. 2022;29:1913-7. Available from: https://doi.org/10.1109/LSP.2022.3203911.

[61] Sainath T, Parada C. Convolutional Neural Networks for Small-Footprint Keyword Spotting. In: Interspeech; 2015. Available from: https://doi.org/10.48550/arXiv.1703.05390.

[62] Li C. OpenAI's GPT-3 Language model: A technical overview. Lambda, Inc.; 2022. Available from: https://lambdalabs.com/blog/demystifying-gpt-3.

[63] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 2014;15(56):1929-58. Available from: http://jmlr.org/papers/v15/srivastava14a.html.

[64] Nielsen MA. Neural Networks and Deep Learning. 2015.

[65] Kandel I, Castelli M. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset; 2020. Available from: https://doi.org/10.1016/j.icte.2020.04.010.

[66] Keskar NS, Mudigere D, Nocedal J, Smelyanskiy M, Tang PTP. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima; 2017. Available from: https://doi.org/10.48550/arXiv.1609.04836.

[67] Geoffrey Hinton KS Nitish Srivastava. Lecture 6a Overview of mini-batch

gradient descent; 2012. Available from: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[68] Jansson P. Single-word speech recognition with Convolutional Neural Networks on raw waveforms; 2018. Available from: https://www.theseus.fi/handle/10024/144982.

[69] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization; 2017. Available from: https://doi.org/10.48550/arXiv.1412.6980.

[70] Wilson AC, Roelofs R, Stern M, Srebro N, Recht B. The Marginal Value of Adaptive Gradient Methods in Machine Learning; 2018. Available from: https://doi.org/10.48550/arXiv.1705.08292.

[71] Dozat T. Incorporating Nesterov Momentum into Adam; 2016. Available from: https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ.

[72] Chia Ai O, Hariharan M, Yaacob S, Sin Chee L. Classification of speech dysfluencies with MFCC and LPCC features; 2012. Available from: https://doi.org/10.1016/j.eswa.2011.07.065.

[73] Misra S, Das T, Saha P, Baruah U, Laskar RH. Comparison of MFCC and LPCC for a fixed phrase speaker verification system, time complexity and failure analysis. In: 2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]; 2015. p. 1-4. Available from: https://doi.org/10.1109/ICCPCT.2015.7159307.

[74] Eltiraifi O, Elbasheer E, Nawari M. A Comparative Study of MFCC and LPCC Features For Speech Activity Detection Using Deep Belief Network. In: 2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE); 2018. p. 1-5. Available from: https://doi.org/10.1109/ICCCEEE.2018.8515821.

[75] Jing X, Ma J, Zhao J, Yang H. Speaker recognition based on principal component analysis of LPCC and MFCC. In: 2014 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC); 2014. p. 403-8. Available from: https://doi.org/10.1109/ICSPCC.2014.6986224.

[76] López-Espejo I, Tan ZH, Jensen J. Exploring Filterbank Learning for Keyword Spotting. In: 2020 28th European Signal Processing Conference (EUSIPCO); 2021. p. 331-5. Available from: https://doi.org/10.23919/Eusipco47968.2020.9287772.

[77] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition; 2015. Available from: https://doi.org/10.48550/arXiv.1512.03385.

[78] Ng D, Yip JQ, Surana T, Yang Z, Zhang C, Ma Y, et al.. I2CR: Improving Noise Robustness on Keyword Spotting Using Inter-Intra Contrastive Regularization; 2022. Available from: https://doi.org/10.48550/arXiv.2209.06360.

[79] Yang R, Haavisto P. Noise compensation for speech recognition in car noise environments. In: 1995 International Conference on Acoustics, Speech, and Signal Processing. vol. 1; 1995. p. 433-6 vol.1. Available from: https://doi.org/10.1109/ICASSP.1995.479621.

[80] R Gary Leonard, George R Doddington. TIDIGITS; 1993. [Accessed 2023-04-

20]. Online. Available from: https://catalog.ldc.upenn.edu/LDC93S10.

[81] Panayotov V. LibriSpeech ASR corpus; 2014. Available from: https://www.openslr.org/12.

[82] Long M. 10 - Sound Transmission in Buildings. In: Long M, editor. Architectural Acoustics (Second Edition). second edition ed. Boston: Academic Press; 2014. p. 383-415. Available from: https://www.sciencedirect.com/science/article/pii/B9780123982582000106.

[83] Hanson BA, Applebaum TH, Junqua JC. Spectral Dynamics for speech recognition under adverse conditions; 1996. Available from: https://doi.org/10.1007/978-1-4613-1367-0_14.

[84] Freudenberger J, Stenzel (EURASIP Member) S, Venditti (EURASIP Member) B. Microphone diversity combining for in-car applications; 2010. Available from: https://doi.org/10.1155/2010/509541.

[85] Warden P. Speech Commands Dataset Version 2; 2018. Available from: http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz.

[86] Warden P. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition; 2018. Available from: https://doi.org/10.48550/arXiv.1804.03209.

[87] Dean D, Sridharan S, Vogt R, Mason M. The QUT-NOISE-TIMIT corpus for evaluation of voice activity detection algorithms. In: Hirose K, Nakamura S, Kaboyashi T, editors. Proceedings of the 11th Annual Conference of the International Speech Communication Association. CD Rom: International Speech Communication Association; 2010. p. 3110-3. Available from: https://eprints.qut.edu.au/38144/.

[88] Audacity. Audacity; 2023. Accessed 2023-04-03. Online. Available from: https://www.audacityteam.org/.

[89] Tribe of Noise. Free Music Archive; 2023. Accessed 2023-04-03. Online. Available from: https://freemusicarchive.org/home.

[90] McFee B, McVicar M, Faronbi D, Roman I, Gover M, Balke S, et al.. librosa/librosa: 0.10.0.post2. Zenodo; 2023. Online. Available from: https://doi.org/10.5281/zenodo.7746972.

[91] Ko T, Peddinti V, Povey D, Khudanpur S. Audio augmentation for speech recognition. In: Proc. Interspeech 2015; 2015. p. 3586-9.

[92] Hmmlearn developers. hmmlearn;. [Cited 2023-03-03]. Online. Available from: https://hmmlearn.readthedocs.io/.

[93] Liu L, Yang M, Gao X, Liu Q, Yuan Z, Zhou J. Keyword spotting techniques to improve the recognition accuracy of user-defined keywords. Neural Networks. 2021;139:237-45. Available from: https://doi.org/10.1016/j.neunet.2021.03.012.

[94] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al.. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems; 2015. Software available from tensorflow.org. Available from: https://www.tensorflow.org/.

[95] Keras;. Available from: https://keras.io/.

[96] Zhang B, Li W, Li Q, Zhuang W, Chu X, Wang Y. AutoKWS: Keyword Spotting with Differentiable Architecture Search; 2021.

[97] Kotila M. Autonomio Talos [Computer software];. Available from: http://github.com/autonomio/talos.

[98] Team GR. kws streaming;. Https://github.com/google-research/google-research.

[99] Rajaby E, Sayedi SM. A structured review of sparse fast Fourier transform algorithms. Digital Signal Processing. 2022;123:103403. Available from: https://doi.org/10.1016/j.dsp.2022.103403.

[100] Pang CY, Zhou ZW, Guo GC. Quantum Discrete Cosine Transform for Image Compression; 2006. Available from: https://doi.org/10.48550/arXiv.quant-ph/0601043.

[101] Vergyri D, Shafran I, Stolcke A, Gadde V, Akbacak M, Roark B, et al. The SRI/OGI 2006 spoken term detection system; 2007. p. 2393-6.

[102] Wang Y, Metze F. An in-depth comparison of keyword specific thresholding and sum-to-one score normalization. Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH. 2014 01:2474-8.

[103] Nielsen J. Response Times: The 3 Important Limits. 1993 January. Available from: https://www.nngroup.com/articles/response-times-3-important-limits/.

[104] Moon TK. The expectation-maximization algorithm; 1996. Available from: https://doi.org/10.1109/79.543975.

A

Appendix

A.1 The EM algorithm

The EM algorithm has the following methodology [104]:

1. Initialize transition matrix **A** and emission probabilities **B**.
2. Expectation step:

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \qquad \forall t, j.$$
$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \qquad \forall t, i, j.$$

3. Maximization step.

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{k=1}^{N}\xi_t(i,k)}.$$
$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t=v_k}^{T}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)}.$$

4. Iterate 2-3 until convergence.

In the equations above, $\gamma_t(j)$ is the probability of being in state j at time t and $\xi_t(i,j)$ is the probability of being in state i at time t and state j at time $t+1$. The backward probability $\beta_t(j)$ is the probability of seeing the observations from time $t+1$ to the end, given state j at time t.

A.2 MP and LOVO loss functions

MP loss is computed according to the formula in equation A.1, where one considers a minibatch of data with P number of keyword samples. In this minibatch, L is all indices of the input feature segments belonging to a class that is not a keyword and $\mathbf{y}_p^*$ is the largest target posterior corresponding to the p-th keyword sample [59].

$$\mathcal{L}_{MP} = -\sum_{i \in \mathrm{L}} \sum_{n=1}^{N} l_n^{\{i\}} \log\left(\mathbf{y}_n^{\{i\}}\right) - \sum_{p=1}^{P} \log\left(\mathbf{y}_p^*\right) \tag{A.1}$$

The formula for the LOVO loss function can be seen in equation A.2. $\mathcal{L}_M$ denotes the triplet loss:

$$\mathcal{L}_M = \sum_{i=1}^{M} \sum_{j=1}^{M} y_i * ||H_i - H_j||_2^2 + \alpha. \tag{A.2}$$

In this formula, α is the margin of the loss, suggested to be set to one by the authors, and M is the size of the minibatch. Furthermore, y_i returns 1 if H_i and H_j are of the same class and -1 otherwise. H_i is computed using feature averaging to the temporal dimension (TAP) with w_1 and w_2 being learning variables for the metric learning:

$$H_i = TAP(Conv(max(0, (Conv(x_i', w_1)))), w_2),$$

where "Conv" denotes a convolution. Moving on, $\mathcal{L}_I$ is the minimization of distance between the embedding vectors and their respective centroids:

$$\mathcal{L}_I = \frac{1}{C} \sum_{k=1}^{C} \sum_{i=1}^{M_k} ||E_i^k - \overline{E^k}||_2^2,$$

with E_i^k being the k-th class subset, M_k the sample size, C the number of classes and $\overline{E^k}$ the class centroid. The inter-class orthogonal loss is denoted $\mathcal{L}_O$ and computed using the spectral norm ($SN(\cdot)$):

$$\mathcal{L}_O = SN(p(M_{IM}) + e^{-M_{DM}} - I).$$

$p(\cdot)$ is a matrix filter that maps all diagonal elements to zero. M_{IM} and M_{DM} are the covariance of E' and a class-wise distance matrix respectively, and these are defined in the following way:

$$\begin{aligned} M_{IM} &= \frac{1}{C-1}(E' - \overline{E'})^T(E' - \overline{E'}) \\ M_{DM} &= \left[||E' - \overline{E^1}||_2^2, ||E' - \overline{E^2}||_2^2, ..., ||E' - \overline{E^c}||_2^2\right] \\ E' &= [\overline{E^1}, \overline{E^2}, ..., \overline{E^c}] \end{aligned}$$

$\overline{E'}$ is the overall mean vector of all centroids. Lastly $\mathcal{L}_{CE}$ is the CE loss previously presented.

A.3 Extensive results

A.3.1 Evaluation of feature extraction methods

Table A.1: Validation accuracy using different types of feature extraction methods with some parameter alterations. A TC-ResNet model was used, and the KWS problem was slightly simplified, as explained in section 5.4, in order to reduce time consumption. The best performing feature configuration is marked in bold.

Feature extraction method	Deltas	# Features	Accuracy
MFCC		13	94.80
		20	95.01
	Δ	**26**	**95.04**
		40	94.83
LPCC		6	78.36
		10	75.48
	Δ	12	78.23
		20	75.44
Filterbank		26	83.13
		36	86.55
	Δ	52	78.48
		72	77.56
Feature extraction method	Deltas	FFT Window size	Accuracy
Spectrogram		512	75.23
		1024	77.54
		2048	85.12
	Δ	512	74.03
		1024	86.35
		2048	75.35

A.3.2 Model architecture tuning

Table A.3: Performance for different DS-CNN models, when running with Adam optimizer, learning rate 0.1, dropout 0.5, batch size 256 and only 6 epochs. Model size is the number of parameters in the ANN, and mult is short for the number of multiplications performed in a single prediction. The best performing model, i.e. the one chosen for further investigation, is presented in bold.

CNN Filters	CNN Stride	DS Filters	# DS Layers	Size [K]	Mult [M]	MKA	KDA	Accuracy
32	(1,1)	64	4	19.7	27.29	69.41	89.48	87.27
			5	24.7	32.17	61.50	86.65	81.44
		128	4	62.4	86.52	73.34	88.50	85.63
			5	80.6	104.81	70.86	89.89	87.90
	(2,2)	64	4	19.7	5.06	64.57	87.04	84.43
			5	24.7	5.39	65.80	88.73	85.40
		128	4	62.4	15.12	68.20	89.47	87.70
			5	**80.6**	**16.35**	**74.75**	**90.68**	**88.58**
64	(1,1)	64	4	23.4	34.29	70.58	88.88	86.25
			5	28.4	39.17	68.33	88.67	85.87
		128	4	68.1	97.31	70.49	89.95	87.50
			5	86.3	115.60	61.73	85.73	82.74
	(2,2)	64	4	23.4	6.77	65.97	88.61	85.95
			5	28.4	7.09	68.70	89.21	85.82
		128	4	68.1	17.70	71.70	89.36	86.91
			5	86.3	18.93	65.97	88.61	85.95

Table A.4: Performance for different Att-RNN models, when running with Adam optimizer, learning rate 0.01, dropout 0.5, batch size 256 and only 6 epochs. Model size is the number of parameters in the ANN, and mult is short for the number of multiplications performed in a single prediction. The best performing model, i.e. the one chosen for further investigation, is presented in bold.

CNN Filters	CNN KS	CNN SL	Heads	Model size [K]	Mult [M]	MKA	KDA	Accuracy
10	(10,4)	(1,1)	4	244.8	3.55	81.43	93.73	92.04
			2	243.0	3.50	69.17	90.82	88.07
		(2,2)	4	149.6	0.69	81.19	93.12	90.97
			2	147.8	0.68	76.35	92.71	89.73
	(5,1)	(1,1)	4	244.1	1.75	83.33	93.33	90.38
			2	242.3	1.70	83.42	94.47	92.67
		(2,2)	4	148.9	0.40	74.51	91.41	87.69
			2	147.1	0.40	58.26	87.22	83.42
32	(10,4)	(1,1)	4	246.7	8.11	81.17	94.21	92.61
			2	244.9	8.06	82.19	94.51	92.26
		(2,2)	4	151.5	1.42	71.63	90.58	87.19
			2	149.6	1.42	78.81	92.61	90.64
	(5,1)	**(1,1)**	**4**	**244.5**	**2.34**	**84.57**	**94.97**	**93.04**
			2	242.6	2.29	82.06	94.37	92.57
		(2,2)	4	149.3	0.50	70.04	89.81	86.21
			2	147.4	0.49	74.23	91.47	87.42

A.3.3 ANN training parameter tuning

Table A.5: Model performance for the TC-ResNet model in order of descending validation accuracy for various training parameter configurations. LR_init is the initial learning rate.

Epochs	Dropout	Optimizer	LR_init	MKA	KDA	Accuracy
27	0.7	Adam	0.010	89.80	96.42	95.34
25	0.3	Adam	0.010	89.78	96.39	95.17
22	0.3	RMSProp	0.005	88.95	96.30	95.06
22	0.7	RMSProp	0.005	88.76	96.15	95.02
30	0.7	RMSProp	0.010	88.39	96.24	94.97
27	0.5	RMSProp	0.010	88.69	96.05	94.84
22	0.3	Adam	0.001	88.05	96.13	94.77
28	0.7	SGD	0.010	88.22	96.13	94.75
22	0.5	RMSProp	0.005	87.79	95.85	94.64
22	0.7	RMSProp	0.001	87.51	95.87	94.53
23	0.5	SGD	0.010	88.39	95.98	94.53
22	0.3	RMSProp	0.001	88.11	95.68	94.44
25	0.5	Adam	0.010	88.20	95.68	94.35
20	0.7	Adam	0.005	88.24	95.87	94.35
27	0.7	SGD	0.005	87.89	95.76	94.25
23	0.5	SGD	0.005	87.46	95.77	94.11
20	0.7	Adam	0.001	87.35	95.62	94.09
27	0.3	SGD	0.005	87.92	95.73	94.03
20	0.3	Adam	0.005	86.47	95.66	93.97
20	0.3	RMSProp	0.010	87.92	95.20	93.84
19	0.5	Adam	0.001	86.64	95.70	93.75
16	0.5	RMSProp	0.001	87.03	95.30	93.65
23	0.5	Adam	0.005	87.72	95.39	93.61
35	0.3	SGD	0.001	85.91	95.08	93.50
19	0.3	SGD	0.010	87.09	95.17	93.45
35	0.5	SGD	0.001	86.06	95.11	93.29
40	0.7	SGD	0.001	85.80	95.08	93.11

Table A.6: Model performance for the DSCNN model in order of descending validation accuracy for various training parameter configurations. LR_init is the initial learning rate.

Epochs	Dropout	Optimizer	LR_init	MKA	KDA	Accuracy
24	0.3	RMSProp	0.010	87.35	95.44	94.28
24	0.5	RMSProp	0.010	86.53	95.41	94.12
37	0.5	SGD	0.100	86.38	95.24	94.02
27	0.7	RMSProp	0.010	85.80	95.22	93.98
37	0.3	Adam	0.100	85.21	95.05	93.87
29	0.3	Adam	0.010	86.40	95.04	93.86
28	0.5	Adam	0.010	85.28	95.08	93.85
25	0.3	SGD	0.100	85.95	94.99	93.82
24	0.7	Adam	0.010	85.06	94.95	93.81
26	0.7	SGD	0.100	85.26	94.94	93.74
38	0.7	SGD	0.010	85.11	94.71	93.53
28	0.7	Adam	0.001	85.11	94.61	93.41
28	0.5	Adam	0.001	85.02	94.72	93.34
31	0.5	SGD	0.010	84.67	94.58	93.34
26	0.5	RMSProp	0.001	84.87	94.58	93.31
26	0.3	RMSProp	0.001	85.04	94.70	93.27
29	0.5	Adam	0.100	83.38	94.42	93.19
34	0.3	RMSProp	0.100	83.38	94.28	93.12
26	0.7	RMSProp	0.001	84.24	94.42	93.11
30	0.3	Adam	0.001	85.26	94.47	93.04
31	0.3	SGD	0.010	83.90	94.35	92.90
47	0.7	Adam	0.100	80.61	93.48	92.37
29	0.5	RMSProp	0.100	80.00	93.07	91.99
42	0.5	SGD	0.001	80.93	93.26	91.77
50	0.7	SGD	0.001	77.93	92.64	91.02
74	0.3	SGD	0.001	79.18	92.75	91.01
50	0.7	RMSProp	0.100	70.28	90.76	88.88

Table A.7: Model performance for the Att-RNN model in order of descending validation accuracy for various training parameter configurations. LR_init is the initial learning rate. Note that some parameter configurations with learning rate 0.1 has not been performed, as this it was found early that this was an unsuitable choice of parameter value.

Epochs	Dropout	Optimizer	LR_init	MKA	KDA	Accuracy
23	0.7	SGD	0.010	87.55	95.78	94.21
13	0.7	Adam	0.001	87.44	95.90	94.20
13	0.5	Adam	0.001	87.74	95.79	94.18
12	0.7	RMSProp	0.001	85.39	95.06	93.40
15	0.5	Adam	0.001	83.79	94.69	93.26
11	0.5	Adam	0.010	85.11	94.93	93.12
50	0.7	SGD	0.001	85.97	94.92	93.10
11	0.7	Adam	0.010	84.63	94.60	93.01
24	0.5	SGD	0.010	84.85	94.69	92.77
50	0.5	SGD	0.001	83.83	94.25	92.37
9	0.7	RMSProp	0.010	73.74	90.26	85.31
9	0.5	RMSProp	0.010	77.99	88.13	82.83
12	0.3	Adam	0.010	0.00	66.07	65.90
63	0.3	SGD	0.001	0.00	66.04	65.75
21	0.3	Adam	0.100	0.00	66.05	51.37
13	0.3	Adam	0.001	0.00	66.05	51.37
6	0.3	SGD	0.100	0.00	66.05	51.37
37	0.3	RMSProp	0.100	0.00	66.05	51.37
11	0.3	RMSProp	0.010	0.00	66.05	51.37
12	0.3	RMSProp	0.001	0.00	66.05	51.37
6	0.7	SGD	0.100	10.55	33.95	3.58
22	0.3	SGD	0.010	10.01	33.95	3.40

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY