Computational modelling of human spoken-word recognition: the effects of pre-lexical representation quality on Fine-Tracker's modelling performance

Author: Danny Merkx, s0813400

Supervisor: dr. Odette Scharenborg

October 20th 2017

Master thesis
Artificial Intelligence - Computation in Artificial and Neural Systems
Faculty of Social Sciences

Radboud Universiteit

Abstract

Fine-Tracker is a model of human speech recognition that is able to model the use of durational cues for the disambiguation of temporarily ambiguous speech. While previous Fine-Tracker simulations were successful at modelling human behavioural data on the use of durational cues, Fine-Tracker is not a very good recogniser of speech. This study proposes to improve the quality of Fine-Tracker's pre-lexical representations by using deep convolutional neural networks for extracting the pre-lexical representations from the speech signal. The convolutional neural networks resulted in large increases in the classification accuracy of the pre-lexical level features. The improvement in the quality of the pre-lexical representations resulted in better word recognition for Fine-Tracker simulations. However, the improved word recognition did not improve Fine-Tracker's simulations of the use of durational information compared to simulations reported in previous studies.

Contents

1 Introduction

1.1 Speech recognition

Speech recognition is a complex process and yet most of us are capable of understanding speech with little difficulty. Yet there is much that we do not know about how the process of speech recognition is organised in the brain. As we cannot look directly into the brain and observe this process, research resorts to indirect methods. Most evidence today comes from behavioural psychology, neuro-science and computational modelling.

One of the difficulties in speech recognition is the fact that the speech signal is highly variable. Even two pronunciations of the same word will never be completely the same even when spoken by the same person [1]. Other sources of variability include gender, dialect, age, social status and language background [2][3]. Not only are we able to understand each other despite this variability, we are able to identify these speaker characteristics from the speech signal and use this information to aid in recognition and interpretation [3]. Now this may not seem so extraordinary because it comes so natural to us but keep in mind that such variability can quickly throw off even the most advanced automatic speech recognition systems. Take for example Alexa by Amazon, a top-of-the-line system able to recognise spoken commands and act on them. One user noticed that suddenly the performance of his new smart home system was degrading badly [4]. It turned out he bought his system during winter and when it became summer he started using his air-conditioning again. The drop in Alexa's performance was caused by the ambient noise made by the AC. Certainly it would take a very loud AC to have this effect on human speech recognition.

1.2 Computational modelling of speech recognition

Computational level theories of speech recognition try to explain how humans are able to map the highly variable speech input to the invariant representations of a lexicon. One such theory on which many influential computational models are based is the abstract theory of speech recognition [1]. This theory assumes that speech recognition is a two staged process. In the first stage the acoustic signal is mapped to a set of limited 'abstract' representations called the pre-lexical level. The pre-lexical units are then mapped to words, or the lexical level [1]. Computational level theories are implemented in computational models in order to simulate human speech recognition (HSR). By trying to reproduce human behavioural data with computational models, researchers explore the various possible implementations of a computational theory and provide data allowing for the evaluation of the theory.

Computational models based on the abstract theory of speech recognition have been successful at reproducing phenomena found in psycho-linguistic experiments. When we hear a speech signal it can initially match more than one word. There is evidence that as the speech signal enfolds, several matching

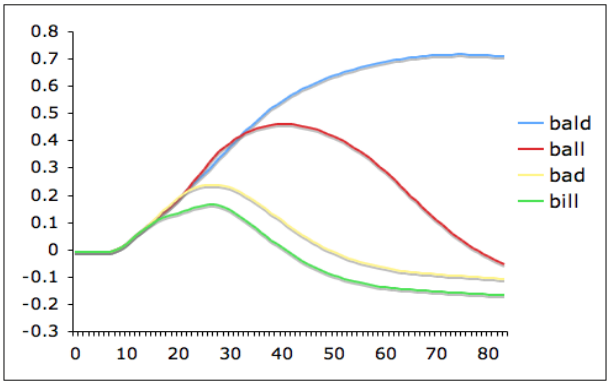

Figure 1: An example of a simulation in Trace. The input sequence is 'bald', several words compete for recognition but are inhibited when they no longer match the input sequence.

lexical representations in the brain get activated and compete for recognition [5][6][7]. As more information becomes available the problem of choosing between these word hypotheses is resolved and a word is recognised. This is called the disambiguation process. For example when hearing the word 'catnip', words sharing initial sounds such as 'catapult' will also become activated, but as the speech signal enfolds 'catapult' will no longer match the signal. Computational models of the abstract theory such as Trace and Shortlist form lexical hypotheses as the input (i.e. an abstract pre-lexical representation) comes in and words consistent with the input get activated and compete [8][9]. An example of the word activation process in Trace is shown in figure 1. The target word in the example is 'bald' and the other words compete with the target until they no longer match the input. In order to solve the disambiguation of the activated lexical hypotheses Trace has activation levels for the words in the lexicon and a lexical decision can be made by using an activation threshold [8]. Trace considers all words in its lexicon as possible word candidates and can therefore only use a relatively small lexicon [1][9]. Shortlist first compares the pre-lexical and lexical representations through an exhaustive search of its entire lexicon. Only a relatively small shortlist of the best candidate words is then considered in the disambiguation process, allowing Shortlist to work with a large lexicon [9].

1.3 Fine-Tracker

Research has shown that humans are able to use subtle details in the speech signal to disambiguate the speech signal in ways that are not predicted by the aforementioned models. Davis et al. and Salverda et al. (2003) showed that listeners were able to distinguish between a monosyllabic word and a longer word in which it is embedded such as 'ham' and 'hamster' before the end of the first syllable [10][11][12]. Salverda et al. (2003) tracked participants' eye

movements as they listened to sentences and were presented with objects on a screen. When participants listened to a word like 'hamster' in which the first syllable was replaced with a recording of the monosyllabic word 'ham' a picture of the monosyllabic word (e.g. a ham) attracted more fixations than when they listened to a recording of 'hamster' [10]. They found that this effect was modulated by the length of the sequence rather than its origin (from a mono or multi-syllabic word) so that longer sequences are more often interpreted as a monosyllabic word. This suggests that durational information contained in the speech signal is an important cue that can be used for word disambiguation.

Both the monosyllabic word and the embedded word are phonemically the same however, and computational models such as Trace and Shortlist are only be able to disambiguate the sequence at the offset of the first syllable which is when the speech signal no longer matches one of the hypotheses. As the example in figure 1 shows, the word hypotheses are equally activated until a hypothesis no longer matches the input sequence. Trace and Shortlist use this so-called post-offset mismatch to disambiguate the sequence. However, research suggests that the identification of the correct sequence can happen earlier then these models predict [10][11].

Fine-Tracker is a model of speech recognition that was built in order to investigate the role of durational information in speech recognition [13][14]. Most existing computational models of speech recognition, including Trace and Shortlist, do not have an explicit pre-lexical level. That is, they avoid the complexity of mapping the speech signal to a finite set of pre-lexical representations and assume that it can somehow be computed and instead use an artificial, often hand-crafted representation of the speech signal [1]. Fine-Tracker incorporates the extraction of the pre-lexical representations from the acoustic signal as an explicit step, allowing it to capture subtle phonetic detail such as durational information and use it for word recognition [13][14]. The units of representation in the pre-lexical level are articulatory features (AFs), which are acoustic correlates of articulatory properties of the speech signal. The AFs are estimated from the speech signal using neural network classifiers. Fine-Tracker is allowed to use the durational information by encoding durational differences into Fine-Tracker's lexical representation, meaning the lexical representation for 'ham' is different from the representation of the first syllable in 'hamster'. This allows for the enabling and disabling of the use of durational information so that the two conditions can be compared.

An additional advantage of using the acoustic signal as input to the model is that the actual stimulus materials from behavioural studies can be used. Scharenborg (2010) used the same materials as those used by Salverda et al. (2003) in order to investigate Fine-Trackers' ability to model the use of durational information [13]. Fine-Tracker was presented recordings of multi-syllabic words where the first syllable was replaced by a recording of an embedded monosyllabic word. When Fine-tracker was allowed to use durational information (i.e. durational information was included in the lexical representations) the word activations for the monosyllabic words were significantly higher than when Fine-tracker was not allowed to use durational information. Furthermore, there

was a positive correlation between the durational difference of the monosyllabic words and the embedded syllables and the modelling results of Fine-Trackers [13]. This is in line with the results of Salverda et al. (2003) that longer sequences are more often interpreted as a monosyllabic word [10].

Fine-Tracker can successfully simulate the disambiguation of ambiguous speech signals as found in humans by making use of durational cues in the speech signal [13]. While it is the first computational model that is able to do this, Fine-Tracker is still a relatively poor speech recognition system. Though Fine-Tracker was able to recognise all the words used in the experiment the correct target word was not always Fine-Tracker's top prediction. The pre-lexical to lexical mapping is dependant on the quality of the pre-lexical representations and Fine-Tracker could benefit from improved AF classification.

1.4 Research goals

The first implementation of Fine-Tracker used multi-layer perceptrons (MLP) with a single hidden layer to map the speech signal to AFs. Advances in the field of neural networks have since made it possible to train much deeper networks and deep convolutional neural networks (CNNs) have been applied to automatic speech recognition (ASR) with much success (e.g. [15][16][17]). Povey et al. (2013) achieved a reduction in relative word error rate of around 15% (various setups were tested) compared to their best Gaussian mixture model approaches [16]. In research comparing MLPs to CNNs Siniscalchi et al. (2012) showed a reduction in relative word error rate of 8.7% [15]. This is promising because it could improve the quality of the AFs that Fine-Tracker uses for the pre-lexical level. The expectation is that Fine-Tracker can benefit from replacing the shallow MLP front-end with CNNs.

In order to improve the quality of the AFs and investigate the effects on Fine-Tracker's simulations, new AF classifiers will be trained. There are seven AFs and a separate classifier will be trained for each AF. The current study uses more data to train the classifiers than used in [13]. In comparing the CNN results directly to the results reported in [13] it would not be clear if any improvement (or deterioration) is caused by the network architecture or simply because different training data was used. New baseline MLP models will be created in order to account for the effects of using different training data. These MLPs will use the architecture described in [13]. Next the CNNs are trained using the architecture described in [18]. In this study Qian and Woodland (2016) compared several CNN architectures for ASR of which the best performing architecture is implemented in this study. Furthermore, an extension of the architecture described in [18] is proposed.

After training the classifiers, the Fine-Tracker simulations reported in [13] are replicated in order to investigate the effects of improved AF classification on Fine-Tracker's performance. The pre-lexical representations of the stimulus materials are made using the newly created classifiers and used in word recognition simulations. The best performing CNN architecture is chosen for the Fine-Tracker experiments and compared to the baseline MLPs.

In summary the goals of this study are:

- Using convolutional neural networks for articulatory feature classification in order to improve the quality of articulatory feature vectors.

- Replicate the experiments reported in [13] using the new articulatory feature vectors in order to investigate the effects of articulatory feature quality on Fine-Tracker's word recognition and modelling power.

1.5 Outline

The rest of this thesis presents the methods and materials used to improve the AF classification and an analysis of the effects on Fine-Tracker's explanatory power. The next chapter gives a review of AFs and their role in ASR. The basics behind MLPs and CNNs are explained as well as the regularisation and learning rate schemes used in this study. Lastly the Fine-Tracker's word activation and competition process is discussed in more detail. Chapter 3 details the training and experiment materials and their pre-processing. Furthermore, the neural network architectures are outlined and the setup of the Fine-Tracker experiments is discussed. Chapter 4 will outline the results of the neural network training and the replication of the Fine-Tracker experiments. We end with a discussion of the implications of the experimental results and suggestions for future work.

2 Background

2.1 Articulatory features

This section will describe in more detail what articulatory features (AFs) are and how they are used in automatic speech recognition (ASR).

AFs are abstract classes that describe speech sounds in terms of properties of speech production (articulation) rather than acoustic properties of the speech signal [19]. The articulatory system consists of all organs used in producing speech (see table 1 for an overview). The active, or moving articulators are indicated with arrows indicating the direction of their movement. We distinguish several areas where the tongue can be pressed to the palate (roof of the mouth). AFs describe the configuration of the articulatory system during speech production, for example whether the vocal cords are vibrating or not [19].

Research has shown that the use of articulatory information can improve the performance of ASR systems. The speech signal is highly variable and it is hard for speech recognition models to capture this variability, AFs can help account for this variability [20][21]. Furthermore, including multiple streams of information in the ASR system can improve the noise robustness of the system

[22]. Research shows that combining a model trained on acoustic features with a model trained on AFs improves phoneme recognition rate of ASR systems. This approach is nowadays being used with convolutional neural networks and research shows word error rate improvements on well-known data sets such as Wall street journal and TIMIT [20][23].

There is a lot of diversity in how AFs are measured or estimated. AFs can be grouped into four categories; features derived by direct measurement, articulatory inversion, landmark detection based features or AF recognition [22][24].

AFs can be directly measured by physical measurement of the articulators. Examples of direct measuring techniques include X-ray, electromagnetic articulography (EMA) and the electroglottograph (EGG) [25][26][27]. While these techniques are useful for studying speech production, few corpora exist that are big enough for training an ASR system [24]. This is because it is very expensive and labour intensive to gather this kind of data. Furthermore, some techniques are invasive, limiting the potential for gathering large databases from large numbers of speakers (e.g. photoglottography [28]).

The speech signal is a product of the speech production process. That is, the movements and configuration of the articulators [29]. In articulatory inversion, the goal is to reverse this process in order to determine the configuration of articulators that produced the speech signal. This requires the creation of speaker specific articulatory-to-aucoustic mappings. Such mappings are trained on simultaneous acoustic and articulatory data for each speaker, thus the limitations of measuring the articulators apply here. To deal with this issue, recent work has focused on ways of combining the articulatory-to-acoustic mappings from multiple speakers into a speaker independent mapping [29][30].

The third approach is based on landmark detection. The goal is to detect where important acoustic events called landmarks occur in the speech signal. An example of a full ASR system is developed by Hasegawa-Johnson et al. (2005) and uses support vector machines to detect manner-change landmarks which correspond to the manner of articulation [31]. Using this approach you only get articulatory information on the speech signal near the landmarks however, and this approach must be combined with other methods to get articulatory data for sections of the speech signal where such landmarks are absent.

The approach taken in the current study is to use classification scores for AFs. In this approach a classifier is trained to recognise AFs given a small frame of speech. This method allows the estimation of AFs directly from the acoustic signal without needing any parallel measurements of articulatory data. Unlike with the landmark approach the AFs can be estimated at any temporal resolution. That is, rather than being dependent on the occurrence of landmarks the AFs are estimated at set intervals so that AFs can be created for the entire speech signal. While this classification approach does not require any knowledge about the speech production system, supervised learning techniques do require labelled training data. Typically this labelling can be acquired from phoneme labels [24]. The advantage of AF estimation is that it allows the use of AFs for corpora for which no articulatory data was measured, broadening the possibilities of using articulatory data in ASR systems.

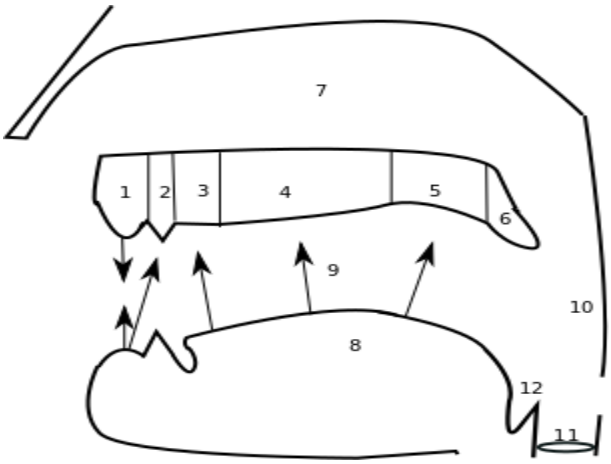

Figure 2: The human articulatory system. The roof of the mouth is divided into the various places of articulation. The arrows indicate the movement of the active articulators. 1. Lips (bilabial, labiodental), 2. teeth (labiodental), 3. alveolar ridge (alveolar), 4. hard palate (palatal), 5. velum (velar), 6. uvula, 7. nasal cavity, 8. tongue, 9. oral cavity, 10. pharynx, 11. glottis (glottal), 12. epiglottis

2.1.1 Phonemes and allophones

Before going into detail about the various AFs a short explanation of allophones and phonemes is in order. Phonemes are the smallest distinctive units of speech sounds and are divided into vowels and consonants [32][33]. Phonemes are the set of sounds that can cause a difference in the meaning of words [32][33]. That is, changing a phoneme in a word changes the meaning of the word, such as in 'bat' and 'hat' (respectively /bæt/ and /hæt/ in phonetic transcription). As the /h/ and /b/ change the meaning of the word they are phonemes. However, phonemes can vary in their surface form or realisation without changing the meaning of what is being said [2][32][33]. An example of surface form variation is caused by reduction, a phenomenon in natural speech where as someone starts to speak faster, sounds become shortened and reduced [2]. These variations are called allophones and while there are technically infinite variations in allophones or the surface forms of a phoneme, each language only has a small set of phonemes [2][32].

As phonemes are a unit of speech sounds they can be described in terms of AFs. This allows us to convert phonetic transcriptions of data into articulatory feature transcriptions. An advantage of using AFs is that they can vary asynchronously allowing them to capture the variability in the surface forms of phonemes. The set of Dutch phonemes and their canonical AF vectors is shown in table 1 and is the same feature set used in [13]. The canonical AFs in the figure below are used to label the training data.

phone	manner	place	voicing	backness	height	rounding	duration-diphthong	Corpus
@	vowel	nil	voiced	central	middle	-round	short	"@"
a	vowel	nil	voiced	back	low	+round	long	"a"
A	vowel	nil	voiced	central	low	+round	short	"A", "A:", "$\tilde{A}$"
A+	vowel	nil	voiced	back	low	+round	diphthong	"A+"
e	vowel	nil	voiced	front	middle	-round	long	"e"
E	vowel	nil	voiced	front	low	-round	short	"E", "E:", "$\tilde{E}$"
E+	vowel	nil	voiced	front	low	-round	diphthong	"E+"
2	vowel	nil	voiced	central	middle	+round	long	"2"
i	vowel	nil	voiced	front	high	-round	long	"i"
I	vowel	nil	voiced	front	middle	-round	short	"I"
o	vowel	nil	voiced	back	middle	+round	long	"o"
O	vowel	nil	voiced	back	middle	+round	short	"O", "O:", "$\tilde{O}$"
u	vowel	nil	voiced	back	high	+round	short	"u"
Y+	vowel	nil	voiced	central	low	+round	diphthong	"Y+"
y	vowel	nil	voiced	central	high	+round	long	"y"
Y	vowel	nil	voiced	central	middle	+round	short	"Y", "9:", "$\tilde{Y}$"
b	plosive	bilabial	voiced	nil	nil	nil	nil	"b"
d	plosive	alveolar	voiced	nil	nil	nil	nil	"d"
f	fricative	labiodental	unvoiced	nil	nil	nil	nil	"f"
g	plosive	velar	voiced	nil	nil	nil	nil	"g"
G	fricative	velar	unvoiced	nil	nil	nil	nil	"G"
h	glide	glottal	unvoiced	nil	nil	nil	nil	"h"
j	glide	palatal	voiced	nil	nil	nil	nil	"j", "J"
k	plosive	velar	unvoiced	nil	nil	nil	nil	"k"
l	liquid	alveolar	voiced	nil	nil	nil	nil	"l"
m	nasal	bilabial	voiced	nil	nil	nil	nil	"m"
n	nasal	alveolar	voiced	nil	nil	nil	nil	"n"
N	nasal	velar	voiced	nil	nil	nil	nil	"N"
p	plosive	bilabial	unvoiced	nil	nil	nil	nil	"p"
r	retroflex	alveolar	voiced	nil	nil	nil	nil	"r"
s	fricative	alveolar	unvoiced	nil	nil	nil	nil	"s"
S	fricative	palatal	unvoiced	nil	nil	nil	nil	"S"
t	plosive	alveolar	unvoiced	nil	nil	nil	nil	"t"
v	fricative	labiodental	voiced	nil	nil	nil	nil	"v"
w	glide	labiodental	voiced	nil	nil	nil	nil	"w"
x	fricative	velar	unvoiced	nil	nil	nil	nil	"x"
z	fricative	alveolar	voiced	nil	nil	nil	nil	"z", "Z"
sil	silence	silence	unvoiced	nil	nil	nil	nil	""

Table 1: AF-to-phoneme mapping. The first column indicates the phonemes used in the current study, the last column is the phoneme notation used in the data corpus. The data corpus uses a different phoneme notation and in some cases multiple phonemes may map to one phoneme. The remaining columns indicate each phoneme's canonical AFs.

2.1.2 Voicing

The voicing feature distinguishes two types of speech sounds; voiced and unvoiced sounds. Voiced sounds are produced by vibration of the vocal cords whereas unvoiced sounds involve no vibration of the vocal cords [2][34]. In

Dutch all vowels are voiced, consonants can be realised with and without vibrating the vocal cords [32].

2.1.3 Manner and place of articulation

The place of articulation features applies only to consonants. This is because consonants are produced by constricting the airflow which is not the case for vowels [32][34][35]. There are various degrees of constriction from full to almost none. The manner of articulation feature describes the type of constriction, the point of maximum constriction along the vocal tract determines the place of articulation [2][32].

The constriction is made using the passive and active articulators. Active or moving articulators are the tongue, lips and glottis, the passive articulators are the upper teeth and the roof of the mouth [32][35]. See figure 2 for a schematic of the human articulatory system; the various places of articulation are indicated and the movement of the active articulators is indicated using arrows.

There are six places of articulation: velar, palatal, alveolar, bilabial, labiodental and glottal. The velar, palatal and alveolar consonants are made by restricting the flow of air by pressing the tongue to the roof of the mouth [2][32]. Velar consonants are made with the back of the tongue raised to the velum (the back part of the roof of the mouth). The palatal consonants are made with the tongue raised to the hard palate (the middle part of the roof the mouth). Alveolar consonants are made with the tip of the tongue pressed to the alveolar ridge (the ridge behind the upper teeth). Bilabial sounds are created by bringing both lips together. Labiodentals are articulated with the lower lip pressed to the upper teeth [2][32][34]. Lastly the glottal consonants are articulated using the glottis; that is, closing the vocal folds, to obstruct the airflow [2].

There are seven manners of articulation: Plosive, nasal, fricative, glide, liquid, retroflex and vowels. Plosives are created with a full obstruction of the airflow so that no air escapes the mouth or nose [2][32]. The airflow can be blocked with either the tongue, lips or glottis. This causes pressure to build up behind the constriction which upon release produces an audible plosion or 'popping' sound. An example of a bilabial plosive is the /p/ made with the lips pressed together. The /d/ is made by pressing the tongue to the roof of the mouth. There are no glottal plosive phonemes in Dutch though they do occur for instance in English where it is better known as the glottal stop [34].

In nasal sounds, the airflow is also fully constricted in the oral tract but air is still allowed to flow through the nasal tract [33][34]. Again we can use either the lips, as in the bilabial /m/, or the tongue, as in /n/ for example. There is no glottal nasal phoneme in Dutch.

Fricatives are created with only a partial constriction, forcing a constant flow of air around the place of articulation. The constriction of the airflow causes an audible turbulence, a characteristic 'hissing' sound, in the airflow called frication [2][32]. An example is the /f/ where air is forced through the upper teeth (labiodental) [2].

In glides or semi-vowels the tongue constricts the airflow such that the airflow is not unimpeded as in vowels, but there is too little obstruction for frication [32][34]. The glides contain the /h/, which is the only glottal phoneme in Dutch.

A liquid consonant is made with very little obstruction of the airflow and like the glides they have similarities to vowels. The liquid /l/ is articulated by allowing air to pass by the sides of the tongue [32].

Sometimes the retroflex /r/ is classified as a liquid meaning there is minimal obstruction of airflow [32][33]. However, it is useful to consider the retroflex as a separate class considering the large range of surface forms that the Dutch /r/ has depending on dialect, co-articulation and style [32][36]. According to [36] the voiced alveolar /r/, which is the canonical form used in this study, is among the most common Dutch surface forms.

Lastly the vowels are pronounced without constriction of the vocal tract and the vowels are treated as a separate class in the manner of articulation feature. As there is no constriction the place of articulation is undefined.

2.1.4 Vowel backness and height

The remaining features all concern the articulation of vowels. Vowels are pronounced with an open vocal tract meaning air is allowed to flow unimpeded unlike during the pronunciation of consonants. During the articulation of vowels, the tongue is used change the shape of the space between the oral cavity and the pharynx. This constriction can be characterised by the height of the tongue and its position relative to the back of the mouth [32][34].

The height feature indicates the position of the tongue relative to the roof of the mouth and thus the amount of space between the oral cavity and the pharynx [32]. We distinguish three degrees of tongue height which are high, middle or low. The 'backness' of vowels indicates the position where the tongue is at its highest point relative to the back of the mouth [33][34]. Again we distinguish three degrees; front, central and back.

2.1.5 Rounding

The rounding feature indicates whether the lips are rounded or un-rounded during vowel articulation [2].

2.1.6 Duration-diphthong

The duration-diphthong feature has three classes. The short and long vowels concern the duration of the vowel sounds. Short vowels are around 100ms on average in duration and long vowels around 200ms on average [32]. A diphthong is a sequence of two vowel sounds within the same syllable where the tongue moves from the first vowel to the second vowel during articulation [2][33][34]. Dutch has three diphthongs; /A+/, /E+/ and /Y+/ [32].

2.1.7 Silence

In order to account for the silence in the speech files it is modelled by the classifiers. Without explicitly having the networks model the silence, the networks will try to classify the silence as phonemes. Therefore silence will be treated as a separate 'phoneme' during training with its own canonical AFs.

2.2 Neural networks

This section will explain the basics behind MLPs and DNNs, and the learning rate schedules and regularisation techniques used in this study.

2.2.1 Multi-layer perceptrons

An artificial neural network or ANN is a type of classifier inspired by the neuronal activity of the brain [37]. Such networks can learn to recognise patterns in data given training examples. In the current study, networks are trained to estimate the AFs of a segment of speech by training them on a large amount of labelled speech samples. The goal of training such networks is for the networks to then classify previously unseen data. The basic unit in an ANN is the neuron, a unit which receives some input and sends an output based on the inputs and some activation function. Mathematically this is given by the following formula:

$$a_j = g(\Sigma_{i=0}^{n} w_{i,j} a_i + b) \tag{1}$$

Where a_j is the output of unit j, a_i is the ith input unit, $w_{i,j}$ is the connection strength or weight of the connection between i and j, b is the bias and g is the activation function. As you can see the unit's activation is determined by a weighted sum of its inputs and some activation function. The collection of all connection weights $w_{i,j}$ is also called a weight matrix. There are different types of activation functions, for instance the step function, which is 1 if the activation exceeds some threshold or 0 otherwise. Another example is the so-called rectified linear unit (ReLU), which is 0 if the activation is below 0 or simply the activation if it exceeds 0. While every input has its own connection weight, the neuron has only one bias term. The bias can be thought of as shifting the threshold of the activation function [38]. Take the aforementioned ReLU, where a negative bias makes it harder for the combined inputs to exceed zero. Conversely, a positive bias makes this easier. Below is a schematic representation of a simple artificial neuron.

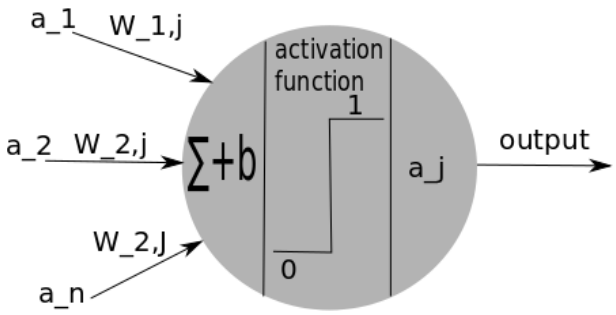

Figure 3: The neuron receives input activations which are multiplied by the weight of their respective connections. These are then summed and a bias term is added, after which the activation function (in this case the step function) determines the final output activation.

The strength of neural networks lies in combining multiple neurons into a network and combining more and more neurons allows the network to learn more complex functions [37]. Neurons are combined into so-called layers, where the output of one layer's neurons is the input of a next layer of neurons. If the neuron in figure 3 is entered into a network, a_1 through a_n would be the outputs of the n neurons in a previous layer and a_j will be fed as input into a next layer. This is called a multi-layer perceptron (MLP) which is a name for any network with more layers than just an input and an output layer. The shallowest MLP is thus a network with an input layer, one hidden layer and an output layer. The hidden layer is called hidden because unlike the input and output, the hidden layers' activations are not directly observed [38]. Figure 4 shows the architecture of a three layer MLP with an input layer, hidden layer and an output layer with two outputs. The layer structure shown below is called a fully connected layer because there is a connection between each pair of nodes in two connecting layers. The size of the weight matrix for such a layer is m by n, which are the number of incoming nodes and the number of connecting nodes respectively.

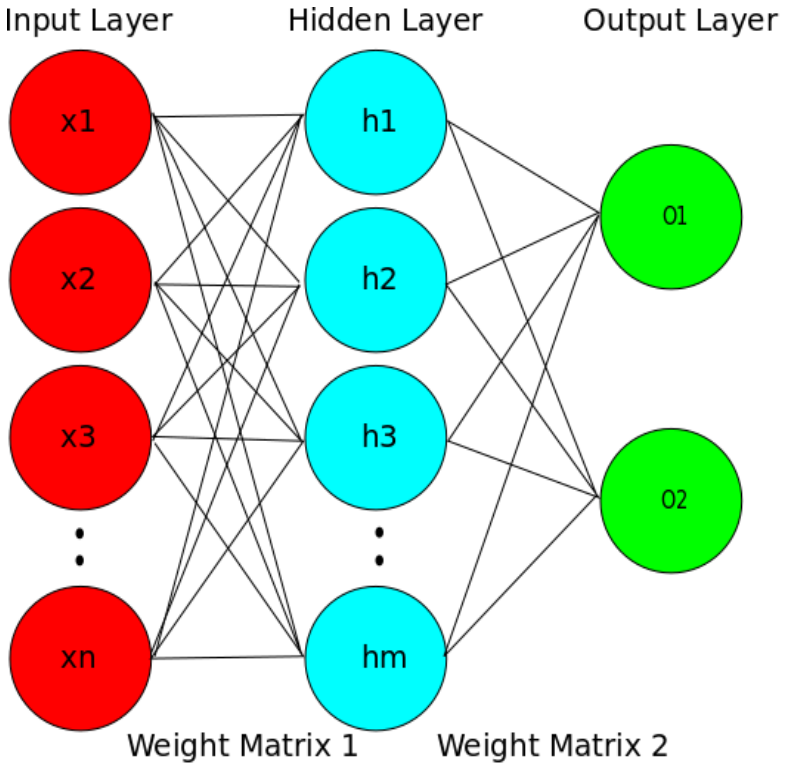

Figure 4: Example of an MLP architecture with a single hidden layer.

2.2.2 Neural network training

A network learns by adjusting the bias and the connection strengths in order to change the output of the network [38]. The network is trained by providing it with training examples for which it is known what the network's output should be and the weights are adjusted if the output is incorrect. A full pass over the training data is called an epoch and a network is typically trained for several epochs considering each training example multiple times.

In order to optimise the networks for solving a particular problem, a loss function is introduced. The loss or cost function allows for evaluation of the network performance in terms of a cost. This cost is a measure of distance between the current solution and the optimal solution, so that 0 cost indicates the optimal solution is found. The goal of training is to minimise the loss function: that is, finding a set of weights and biases with the lowest possible cost [38]. Take for example the quadratic cost function shown below:

$$C(w,b) = \frac{1}{2n}\sum_{x}(y(x)-a)^2 \tag{2}$$

Where C is the cost with respect to the weights w and biases b, n is the

number of training examples, $y(x)$ is the correct output for training example x and a is the actual network output. Unfortunately, for networks with a lot of weights the minimum of this function cannot be determined analytically [38]. Instead a technique called gradient descent is used in order to search the solution space and try to find weights and biases with as low a cost as possible. Using gradient descent, systematic updates are applied to the weights rather than randomly searching the space of possible weight sets. Each weight is changed only slightly using the following update rule:

$$\Delta(w_{j,i}) = -\eta \frac{\delta E}{\delta w_{j,i}} \tag{3}$$

Where $\Delta(w_{j,i})$ is the weight update to be applied to $w_{j,i}$, η is the learning rate which can be used to control the size of the weight updates and $\frac{\delta E}{\delta w_{i,j}}$ is the partial derivative of the loss function with respect to $w_{j,i}$. The biases are updated in a similar fashion by taking the partial derivative of the loss function with respect to the biases. For the full derivation of the loss function see [37], for now it suffices to say that it is used to determine how the cost changes when the weights and biases are changed by determining the slope of the cost function. Moving down this slope by adjusting the weights and biases decreases the cost. The learning rate η from equation 3 is used to control the size of the weight updates and can have a large influence on the training of a network.

2.2.3 Learning rate

There are a few limitations to gradient descent: for one it is not guaranteed to find the optimal solution [37][38]. As said before it is practically impossible to define the global minimum for all but toy examples. Secondly the learning rate is a hyper-parameter that can have a large influence on the parameter updates and therefore a large influence on finding a decent local minimum. It is not possible to analytically determine the optimal value for the learning rate. A low learning rate slows down convergence and makes the network prone to getting stuck in a local minimum early on during training. On the other hand a high learning rate can prevent the network from converging to a minimum (i.e. by taking large steps it is possible to skip over a minimum). However, it is not obvious what learning rate is 'too high' or 'too low'. It is common practice to train networks with a learning rate schedule that adjusts the learning rate during training [38][39][40].

While testing various learning rates can be a solution, several alternatives to the fixed learning rate have been developed. The techniques used in the current study are learning rate decay and Nesterov momentum. Learning rate decay decreases the learning rate during training as given by the following equation:

$$\eta_{n+1} = \eta_n \times d \tag{4}$$

Where the learning rate in epoch $n+1$ is the previous learning rate multiplied by the decay factor d. A higher learning rate prevents the network from getting

stuck in a local minimum during the first epochs and this promotes exploration of the solution space. The learning rate decreases with every epoch which allows the weights to converge to a minimum.

Another schema for adjusting the learning rate is Nesterov momentum given by:

$$v_{n+1} = m \times v_n - \eta \nabla(\theta + v_n) \tag{5}$$

Where v is the velocity at update step n, m is a new hyper-parameter called momentum (where $0 \leq m \leq 1$) and ∇ is the gradient with respect to $\theta + v_n$ and θ is any learnable parameter such as a weight or bias [60] [39].

The update is given by:

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{6}$$

The momentum technique tends to update the parameters along the previous update direction (building up velocity in the direction of previous updates), preventing the so-called zig-zag pattern along the gradient that gradient descent is prone to and promoting quicker convergence [39]. The downside is of course the introduction of the new hyper-parameter momentum which has to be chosen. These learning rate schedules can be combined so that weight decay decreases the base learning rate after every epoch and the Nesterov technique allows the update steps to gather momentum (momentum is added to the learning rate based on the past few training examples, it does not adjust the base learning rate).

2.2.4 Convolutional neural networks

Many types of data such as images and speech data are highly spatially or temporally correlated [41][42]. A downside of MLPs is that they ignore the topology of the input data. The order of the inputs to an MLP does not make a difference even though the pixel order in an image certainly contains information (e.g. randomly scrambling the pixels of an image would not matter to an MLP). The same goes for instance for the temporal order of a speech signal. When an MLP is presented with a segment of speech, it ignores the order of the sequence thus not exploiting any information contained in the temporal order of the signal.

Convolutional neural networks (CNNs) are able to extract and exploit such local features by looking at a small receptive field (i.e. activations are calculated over a small patch of the input) [41][43]. An example of a convolutional layer is shown in figure 5. CNNs use something called a filter or kernel, shown in the middle of figure 5, which slides over the input features [38][44]. The receptive field in this case is 3 by 3 and its movement across the input space is indicated by the coloured outlines. When a filter is applied to a receptive field, the input features are multiplied with the filter weights and summed after which a bias and activation function are applied. The resulting activations are then collected in a feature map, shown on the right with colours matching the receptive fields. As indicated in the example this procedure retains the topology of the feature

space. Each convolutional layer has a configurable number of such filters, with each filter in a layer resulting in its own output feature map. The step size of the receptive field is called the stride, in the example the stride is 1 on both dimensions. It is common practice to use convolutional layers in order to extract the local features from the data after which one or more fully connected layers are applied before the output layer. CNNs are often used in image classification for tasks such as face recognition, object detection and automatic image annotation but lately is also seeing increased use in ASR [15][18][45][46][47].

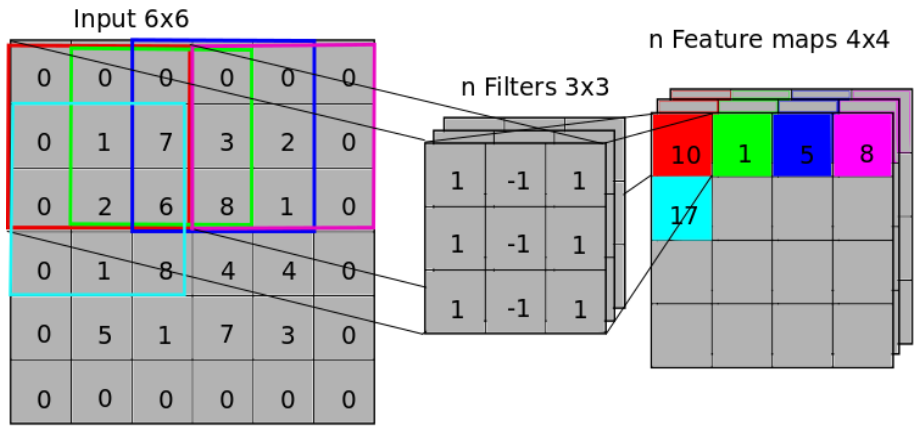

Figure 5: A schematic of a convolutional layer with two dimensional input. The filter slides over the input in both directions. The activations of the receptive fields are collected in a feature map shown on the right side.

The convolution results in output feature maps that are smaller than the original input. In order to prevent the feature maps from decreasing in size and to better take advantage of the information on the borders of the input, zero padding can be applied [17]. When using zero padding, a border of zero values is added to the input. Zero padding is shown in figure 5 where a border of zeros is applied along both axes of the input. The original un-padded input size was four by four and without padding the resulting feature maps would have been two by two. By using zero padding, the feature map retains the same size as the un-padded input.

Besides being able to exploit the local properties of speech and image data, the CNN has the advantage of being able to deal with larger input sizes. In a fully connected layer, each input has a unique connection weight to each neuron in the next layer. The size of the weight matrix is thus n by m where n is

the number of inputs and m the number of neurons in the connecting layer. The number of weights thus increases quickly with the size of the input. One of the advantages of the convolutional layer is that the filter weights and bias are shared by the entire input [38]. As seen in figure 5, the same 9 weights are applied to the entire input. In convolutional neural networks the number of weights is determined by the size of the receptive fields and the number of filters. For filters of size 3 by 3, each filter added to the convolutional layer would only add another 9 weights and an extra bias no matter the size of the input. This allows for larger input sizes without increasing the number of parameters to be learned [41].

2.2.5 Pooling

It is common practice in CNNs to perform down-sampling after a few convolutional layers which is implemented in a so-called pooling layer. Pooling can be thought of as a feature selection procedure where only the most useful information is retained [48]. There are several options for pooling layers such as max-pooling and mean pooling [49]. The max-pooling operation outputs the maximum value in the receptive field, the mean-pooling outputs the average of its receptive field. Figure 6 shows how max-pooling works. The input shown on the left is divided into non-overlapping receptive fields of size 2 by 2 and of each field only the maximum value is retained in the output. Not only does down-sampling drastically reduce the input size (a pooling size of two by two reduces the amount of data points by 75%) and thus the computational burden, it also makes the network more robust to translational variance in the input (e.g. a rotation of the input features)[48][49]. Take for example the red receptive field in figure 6, no matter how features in this field are rotated, the result of the max-pooling remains 3.

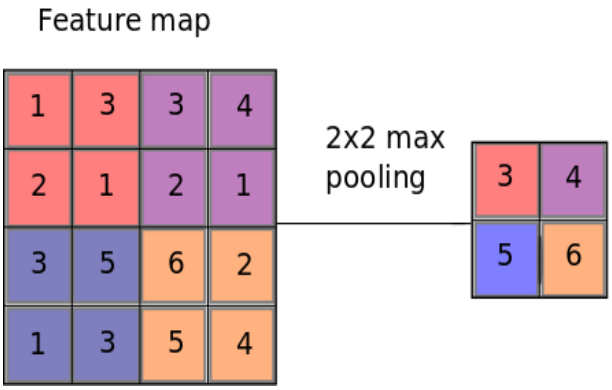

Figure 6: Max pooling of size two by two; the input is divided into four receptive fields. Only the maximum value of each receptive field is retained in the output feature map.

2.2.6 Regularisation

Overfitting is what happens when a model is trained so that it starts to adapt to the training data rather than learning to solve the classification problem [38]. That is, the network may 'memorise' the training data-set gaining near perfect classification scores on that data but terrible scores on an unseen set [40]. As the labels to the training data are already known, from a practical view it is only interesting to use the network to classify an unseen data set. As such the network should be able generalise well to unseen data [38]. Regularisation is a term for a set of techniques that are meant to reduce overfitting.

In order to reduce overfitting, the following techniques are used in this study. Dropout is a relatively simple but powerful technique [38][44][43]. The basic principle of dropout is to 'drop' a percentage of randomly chosen neurons by fixing their output to be 0 during training. The training data is passed through the network, the weights are updated and in the next iteration a new random set of neurons is dropped. This means a different network architecture is trained at each iteration. When applying the network to unseen data all neurons are kept in the network, acting rather like an ensemble of classifiers [38][44]. Furthermore, dropout prevents neurons from co-adapting too much [43][50]. In other words, because of the dropout the network cannot rely on the connection to a neuron or piece of the input features to be present and should not be completely thrown off if such as feature is missing or its value is not what is expected.

For CNNs, dropout can be applied as described above by randomly dropping activations in the feature maps. However, in [50] the authors implement a new type of dropout for CNNs called spatial dropout. The input to a CNN is usually locally correlated, for instance in the pixels of an image or the frequency spectrum of speech signals. They found that dropping a small amount of features in the receptive field of the kernel did not work because the remaining activations are correlated with those that were dropped. In spatial dropout, entire feature maps are dropped so that feature maps are either fully active or fully fixed to zero and they found this to improve the performance of their CNNs [50]. In the current study, spatial dropout is used in the convolutional layers and regular dropout is applied to fully connected layers.

A relatively new method is batch normalisation. It acts both as a regulariser and allows for higher learning rates to be used during training [48]. During training, networks experience a change in the distribution of the inputs to the internal nodes called internal covariate shift [42][51]. This means that small changes in the lower layers can have large effects on the distribution of the inputs to the higher layers and these effects become larger in the higher layers. The higher layers are not only dependent on their own connections and biases but also on those of all the layers below them and they will have to adapt to changes in the distribution of their inputs. In practice this problem is countered by setting a low learning rate, allowing only small changes to the weights to be made, but this slows down learning.

In [51] Arpit et al. (2016) propose a technique to normalise the input to each layer in order to keep the distribution of the inputs more stable during

training. Of course normalisation does not work on a single training example and therefore they use mini-batch training where a 'batch' of inputs is fed to network rather than one input at a time (hence the name batch normalisation). The training samples in each batch are normalised after each layer so that the inputs to the next layer have zero mean and unit variance.

The downside is that normalisation affects what information a layer can represent and as such it would limit the network in what it can learn [51]. Take for instance the sigmoid activation function shown below in figure 7. If the inputs to this function all have zero mean and unit variance most of them will be in the (near) linear part of the activation function indicated by the vertical lines and almost never reaching the tails of this function.

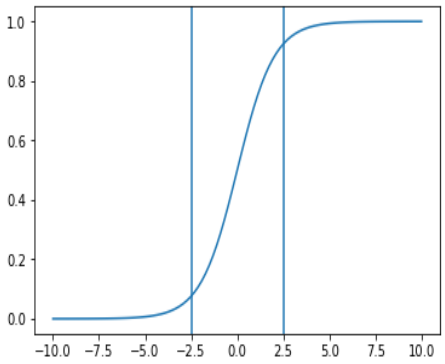

Figure 7: Plot of the sigmoid activation function. Normalised input will be mostly within the indicated range.

Therefore they add two parameters to the batch normalisation operation so that the output of the normalisation is given by:

$$y^k = \gamma^k x^k + \beta^k \tag{7}$$

Where x^k is the kth normalised value. γ and β are two parameters that are introduced in to counter the representational restrictions mentioned before. These parameters are updated along with the other network weights and biases. Note that these parameters even allow the normalised value to be restored to the original value if γ and β are the standard deviation and average respectively. The network could theoretically learn to do this if it were the optimal thing to do [51]. The authors found that batch normalisation allows for larger learning rates to be used. Furthermore, dropout could be eliminated or at least the dropout rate could be drastically lowered [51].

3 Methods

3.1 Data

3.1.1 Corpus Spoken Dutch

Neural network models perform best on data that is as close as possible to the data they were trained on. The speech data for the Fine-Tracker experiments consists of high quality recordings of read speech by a Dutch speaker. Therefore, the MLPs and CNNs used in this study are trained on the read speech component of the Corpus Spoken Dutch (CGN, Corpus Gesproken Nederlands) which was released in 2004. The CGN is a large corpus of recordings of Dutch and Flemish speech, roughly two thirds and one third of the data respectively, comprising over 9 million words. The database has several components besides the read speech component covering different types of speech such as spontaneous conversation, telephone dialogue, lectures and news-bulletins.

3.1.2 Read speech component

The read speech component of the corpus contains relatively clean and high quality recordings, as the speech consists of non-spontaneous monologues. This type of speech most closely resembles the speech that will be used for the experiments.

This component contains 903.043 words of which 551.624 come from Dutch speakers and 351.419 from Flemish speakers. The networks will only be trained on the set of Dutch speakers consisting of 561 recordings for a total of about 64 hours of speech. The Dutch part of the read speech component contains recordings from 324 unique speakers which means some of the speakers appear in multiple recordings. Note that each file only ever contains speech from one speaker.

3.1.3 Data split

It is standard practice to split the training data into three distinct sets; a training, validation and a test set. The classifiers will be trained on the training set. The validation set can be used to see if a model is still improving during training and not over-fitting on the training data. Furthermore, the validation set is used to tune the model's hyper-parameters such as the drop-out rate and the learning rate. The performance of the final model will be evaluated on a held out test set.

The data set was split into a training (79.5%), validation (10.58%) and test set (9.92%) while keeping the characteristics of the speakers in each set roughly equal. The split is roughly 80/10/10 but not exactly because the audio files all have a different length. The speaker characteristics taken into account are sex, level of education and age. Age was binned into four bins; 24-50, 51-59, 60-69 and 70-89[1]. Each speaker appeared in only one of the sets. The table below

[1]The bins were created according to the 25th, 50th, 75th and 100th percentile.

shows the speaker characteristics of the training, validation and test sets.

	Speakers	Sex (%)		Education (%)				Age %			
		Male	Female	High	Middle	Low	Unknown	24-50	51-59	60-69	70-89
Training	259	44.4	55.6	81.85	16.22	0.77	1.16	24.71	25.48	27.41	22.39
Test	33	45.46	54.54	78.78	18.18	0	3.03	30.30	21.21	24.24	24.24
Validation	32	43.75	56.25	78.12	18.75	0	3.12	25	28.12	25	21.87

Table 2: The distribution of the speaker characteristics for the data split used for training the neural networks.

3.2 Acoustic features

The audio files in the database contain raw speech signals. The raw speech signal is converted into acoustic feature vectors with each vector representing the acoustic features of a small segment of speech. In order to do this, the speech signal is framed using 25ms analysis windows with a 5ms shift. The acoustic features are computed and labelled at the frame level.

One of the most commonly used acoustic feature is the Mel-frequency cepstral coefficient or MFCC [2]. Similar to Scharenborg (2010), the baseline MLPs use MFCCs augmented with first and second temporal derivatives (also known as delta and double delta coefficients) [13]. The MFCCs are created using the standards described by the European Telecommunications Standards Institute (ETSI) in [52]. The delta and double delta coefficients were created as described in [2]. The block diagram in figure 8 shows the pre-processing pipeline as it was implemented for this study. See appendix A for a detailed description of the pre-processing steps and acoustic features in figure 8.

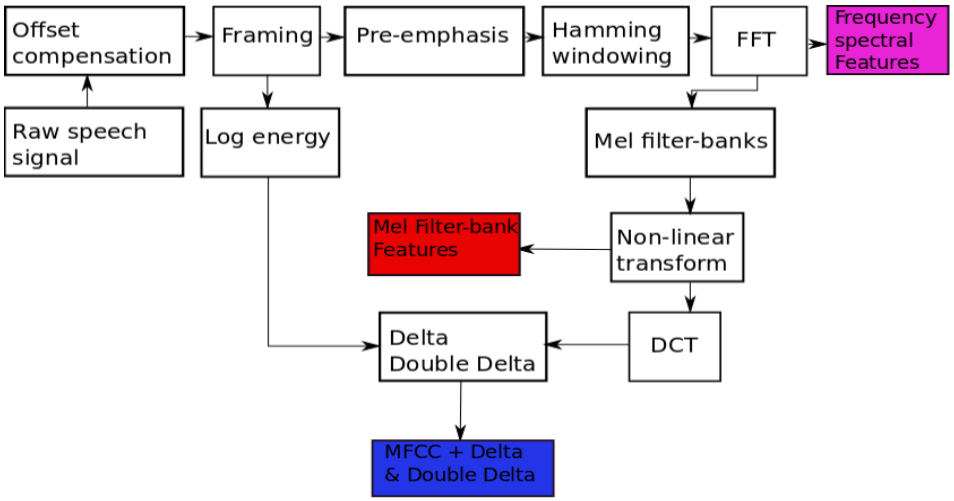

Figure 8: Block diagram of the pre-processing pipeline. The coloured blocks indicate the acoustic features. FFT stands for fast Fourier transform, DCT stands for discrete cosine transform.

The CNN architecture as described by Qian and Woodland in [18] are trained using Mel filter-bank features. This is because MFCCs are not well suited for use with CNNs. As discussed in section 2.2.4, the receptive field of CNNs allows the network to extract information from the topology of the data. Running the pipeline up to the DCT conversion results in Mel filter-bank features, indicated in red in figure 10. The Mel filter-bank features are correlated filterbank energies which are ordered on the Hz scale. MFCCs are created by applying the discrete cosine transform (DCT) to the Mel filter-bank features. One of the effects of the discrete cosine transform (DCT) is that it de-correlates the filter-bank energies and as such the receptive field can not derive information from the order of the MFCC coefficients.

The Mel filter-bank features are created by applying Mel filters to the frequency spectral features (indicated in purple in figure 10). These frequency spectral features, or FFT bins, are the result of applying the fast Fourier transform (FFT) to the speech signal. An extension to the architecture described in [18] is proposed where the Mel filtering operation is implemented as a layer in the CNNs. These CNNs take the frequency spectral features as input and the Mel filters are optimised as part of the network training.

3.3 Transcriptions

In order to train NNs for AF recognition a ground truth labelling is required in conjunction with the speech data. AF transcriptions of the speech data are required to label the data but these are not provided for CGN. However, hand-verified orthographic transcriptions and a lexicon with phonetic transcriptions of

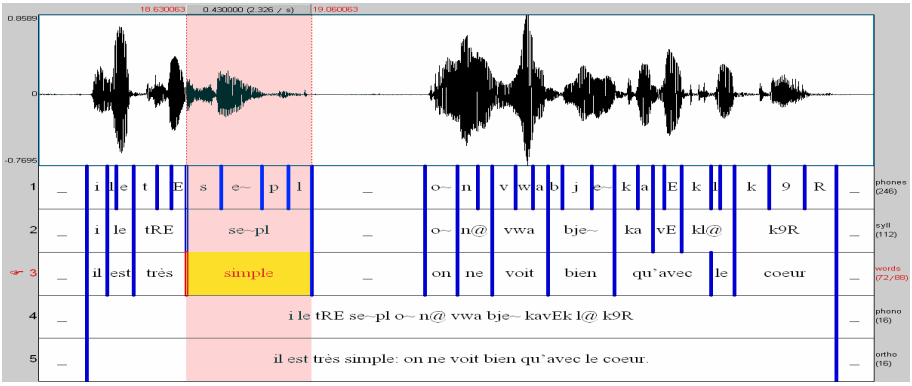

Figure 9: Alignment of transcripts to speech signal in Praat. Multiple levels are shown here from the full sentence level to word and phoneme level alignments.

all the words are available. Given the hand-verified orthographic transcripts and a lexicon which maps orthographic representation to phonetic representation, it is possible to perform a so-called forced alignment and create automatically generated phonetic transcriptions. AF transcriptions can then be made using the canonical AFs in table 2. Automatically generated phonetic transcriptions of the CGN material are available but these are not used because advances in ASR have since made it possible to create better forced alignments.

3.3.1 Forced Alignment

Forced alignment takes in a speech signal and an orthographic transcript, and tries to assign phoneme boundaries that best match the speech signal based on a lexicon and an acoustic model. For example, from the existing orthographic transcript we know which part of the speech signal contains the word hamster. From the lexicon we know the phonetic form of hamster is hAmst@r. Now the alignment is a matter of assigning phoneme boundaries within a small segment of the speech signal. Figure 9 shows an example of a speech signal with aligned transcriptions. Line 3 shows the orthographic transcription aligned on the word level with the phonetic transcriptions of the words on line 2. Line 1 shows the phonetic transcription aligned at the level of the individual phonemes. Creating such a transcription is the goal of forced alignment.

Forced alignments were made using Kaldi, a toolkit for speech recognition [16]. An existing lexicon was used, however, approximately 200 words in the transcriptions (all mispronunciations) did not appear in the lexicon. The CGN automatic phonetic transcriptions were used to complete the lexicon.

The alignments were made using GMM-HMMs (Gaussian mixture model-hidden Markov models) which are a component of many state of the art ASR systems (e.g. [17]). Figure 10 shows an example of a GMM-HMM for the word ham. Each phone is modelled by three states, one for the beginning, one for the middle and one for the final part of the phone (indicated by b, m, f in figure

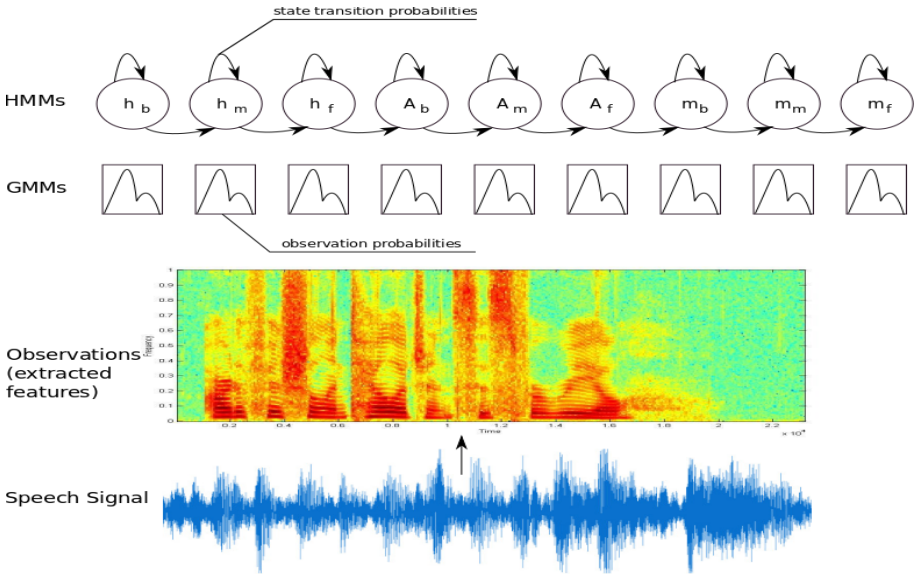

Figure 10: GMM-HMM for the word ham.

10). Each state has two transition probabilities; one for remaining in the current state and one for the transition to the next state. Acoustic features computed from the speech signal are the observations. The states of the model are 'hidden' but each HMM state emits an observation in the form of the acoustic features. The probabilities for each state to emit a particular observation are determined by the GMMs. Each observation sequence is modelled by a separate GMM-HMM with its own transition probabilities, the GMMs for the phone states are shared by all GMM-HMMs. The goal of forced alignment is to find the sequence of hidden states that is most likely to have emitted the observed signal, this is process is called decoding [2]. Rather than evaluating every possible hidden state sequence this is done using the Viterbi algorithm (see [2] ch. 6 for a full explanation of Viterbi).

Training a GMM-HMM model is the task of determining state transition probabilities and emission probabilities that maximise the likelihood of the observed data given a hidden state sequence [2]. This is done using the Baum-Welch algorithm (see [2] ch. 6 for a full explanation of Baum-Welch). The hidden state sequence and the transition and emission probabilities are unknown so the algorithm starts with an estimation. This initial estimation is an equally spaced alignment, transition probabilities of 0.5 and GMMs with the mean and variance for each Gaussian set to the global mean and variance for the entire data-set [2]. The estimations are then iteratively improved using the Baum-Welch algorithm.

First a monophone model, in which no context of the phonemes is consid-

ered, is trained on MFCCs. Trained models are to bootstrap the training of a new model. That is, the alignment, emission probabilities and transition probabilities of the monophone model can be used as the initial estimation for more complex models.

Subsequently various triphone models are trained. Triphone models model the phones' left and right context as such taking into account the pronunciation variability of the phones due to coarticulation [22][24]. First delta-based triphones are trained, then delta + delta-delta based triphones, LDA-MLLT based triphones and SAT triphones. Each new model is bootstrapped using the previous model.

Delta and delta-delta are the first and second temporal derivatives of the MFCCs respectively. First a model is trained only on MFCCs with delta features and in the next model delta-delta features are added. LDA-MLLT stands for linear discriminant analysis - maximum likelihood linear transform [53][54]. LDA is used to reduce the dimensionality of the input, MLLT decorrelates the input. LDA-MLLT transformed MFCC features are shown to improve word error rates for GMM-HMM-based ASR systems [16][55]. SAT stands for speaker adaptive training [56] which adapts the GMM-HMMs for speaker variation. SAT is shown to improve word error rates for GMM-HMM-based ASR systems in [16].

3.4 Networks

This section describes the architecture, input data and parameter settings of the neural networks that were trained.

3.4.1 Multi-layer perceptron architecture

The baseline MLPs were implemented as described in [13] and consisted of an input layer, a fully connected hidden layer and a soft-max output layer. A total of seven networks were trained, one for each AF. The hidden layer has a hyperbolic tangent non-linearity. Scharenborg (2010) based the number of hidden nodes for each MLP on tuning experiments [13]. The number of output nodes is equal to the number of classes of each AF. No drop-out or batch normalisation was used. Table 3 shows the number of hidden nodes and output nodes for each AF as used in [13].

Articulatory feature	#hidden nodes	#output nodes
Manner	300	8
Place	200	8
Voicing	100	2
Backness	200	4
Height	250	4
Rounding	200	3
Duration-diphthong	200	4

Table 3: The number of hidden nodes and output nodes per AF.

The MLPs were trained on MFCC acoustic features; 12 cepstral coefficients plus log energy and delta and double delta features for a vector of 39 features. The network receives 11 consecutive frames as input with the middle frame being the frame to classify [13]. This brings the networks' input size to 11 by 39.

The networks are trained using Nesterov momentum with a learning rate of 0.01 and momentum of 0.9, a learning rate decay of 0.5 per epoch and a batch size of 512. After each epoch the network performance is evaluated on the validation set and training is stopped when the validation accuracy starts to drop.

3.4.2 Convolutional neural network architecture

The CNN architecture is implemented as described in [18]. The input feature vectors consisted of the Mel filtered signal. As with the MLPs, the network were fed 11 consecutive frames, with the middle frame being classified. Seven networks were trained, one for each AF.

The filter-bank features were created using 64 Mel filters resulting in an equal number of filter-bank energies. The input was chosen to be an exponent of two which means the input can be neatly down-sampled by max-pooling, without having to either apply zero padding or discard the border. This brings the input size to 11 by 64.

Figure 11 shows an overview of the CNN architecture. The architecture consists of six blocks. The first five each consist of two convolutional layers with ReLU non-linearity followed by a max-pooling layer. Padding is applied to the borders of the convolutions so that the layers' output size is equal to its input size. Spatial dropout of 20% is applied to the convolutional layers. Furthermore, batch normalisation is applied in between the convolution and the non-linearity.

The size of the convolutional filters is three by three throughout the network with a stride of one in both directions. The number of feature maps increases with depth. The first two convolutional layers have 64 feature maps, the next four have 128 and the last four layers have 256. The first two max-pooling layers are of size one by two as pooling is only applied in the frequency dimension. The next three pooling layers are of size two by two now being applied in both

the time and the frequency dimension[2]. The stride of the pooling layers is equal to the pooling size in both directions. The first two pooling layers in the time dimension (the third and fourth pooling layers) apply zero padding on the time axis to make the size of the time dimension appropriate for max-pooling of size two. This is because sub-sampling along a dimension whose size is not divisible by the size of the pooling filter means that the border will be discarded.

After the convolutional layers, a block of four fully connected layers is applied. Each of these layers has 2048 hidden nodes, a drop-out of 40% and batch normalisation before the ReLU non-linearity is applied. The output layer is a soft-max layer with the number of output nodes varying per network depending on the AF.

[2]Because the time dimension is only size 11 it can not be sub-sampled five times.

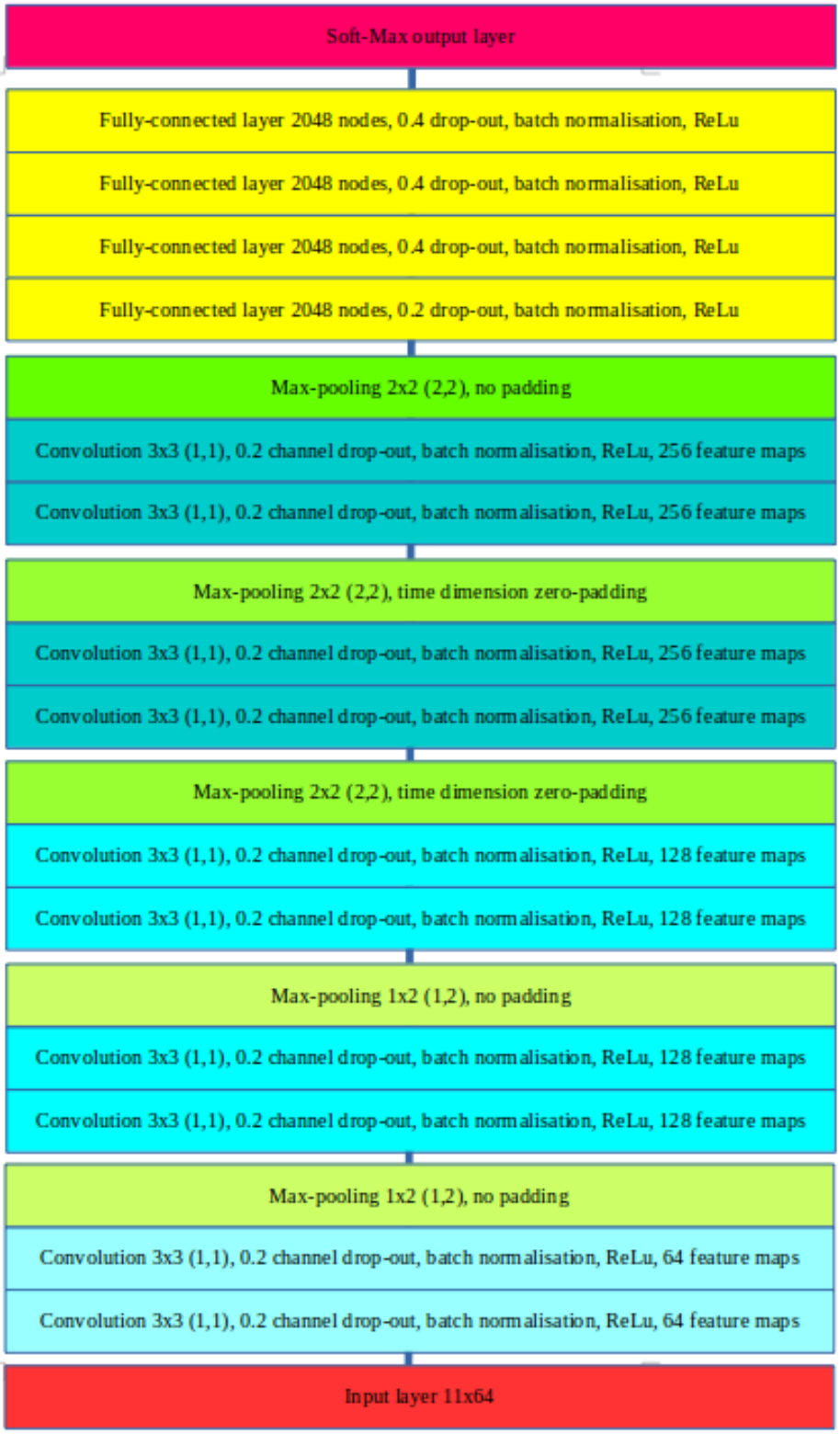

Figure 11: Overview of the DNN architecture used in this project.

The networks are trained using Nesterov momentum with a learning rate of 0.01 and momentum of 0.9 a learning rate decay of 0.5 and a batch size of 512. A pilot test showed that the networks were very close to their final performance even after a single epoch. The first epochs resulted in some small improvements with only marginal improvements in the order of 0.1% after four epochs. Considering that the networks perform well after a small amount of epochs and the long training time for each epoch, the maximum amount of epochs was set to five. The validation set accuracy is still used as an early stopping criterion.

3.4.3 Trainable filter-bank architecture

The second CNN architecture is a proposed extension of the architecture described in the previous section. The Mel filters used to extract the Mel filter-bank energies from the frequency spectral features were implemented as a convolutional network layer that was inserted right after the input layer. By including the Mel filters in the CNNs and updating their weights along with the rest of the network, the networks can potentially optimise the filters for the classification of a particular AF.

The Mel filters consist of filter coefficients forming a configurable number of triangular filters. These half overlapping filters cover the entire frequency spectrum and each filter collects energy from its own part of the spectrum. The application of the Mel filters is done by multiplying the frequency spectral features with the filter coefficients and summing the results over the entire filter.

This is exactly what a convolutional filter does and as such the Mel filtering operation can be implemented as a convolutional network layer. This layer has 64 filters so that the result of the layer will be 64 Mel filter-bank features as used in the previous section. The weights of the filters are preset to the Mel filter-bank coefficients (see appendix A for the creation of these coefficients). The convolution is basically the same as explained in section 2.2.4 except that they have a stride of 0 in the frequency dimension because the filter size is equal to size of the frequency spectral features. The filters slide only over the 11 input frames. Figure 12 shows how this layer is slightly different from that shown in figure 5.

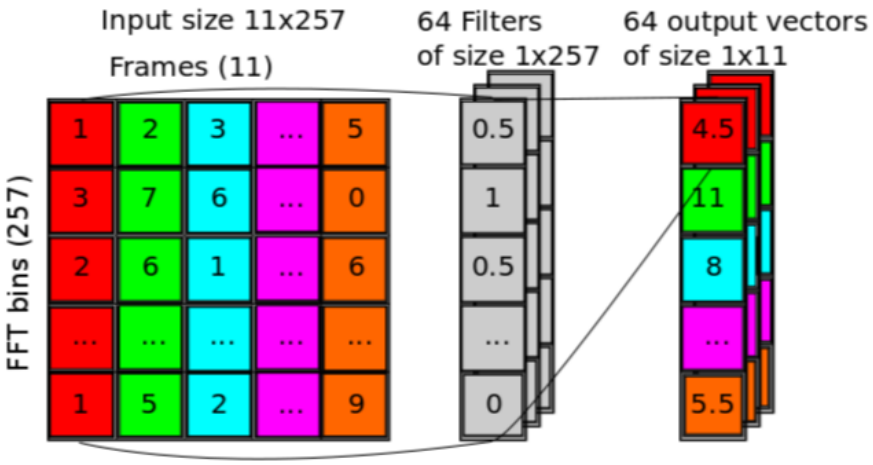

Figure 12: Schematic of the Mel-filtering operation implemented as a convolution network layer. The convolutional filters are as wide as the FFT size and slide over the time dimension.

After taking the natural logarithm of this layers' output, the result is equal to the Mel filter-bank features. However, the filters are now incorporated into the CNN and their weights will be trained along with the rest of the network.

The size of the input is 257 frequency spectral features by 11 frames. The output size of the new layer is 11 by 1 with 64 feature maps, which is reshaped to 11 by 64 corresponding to the number of frames and filter-banks. The result can now be used as input to the network shown in figure 14.

The networks were trained using Nesterov momentum with a learning rate of 0.01 and momentum of 0.9 a learning rate decay of 0.5 and a batch size of 512. The maximum number of epochs was set to five while using the validation set accuracy as an early stopping criterion.

3.5 Fine-Tracker materials

The acoustic stimuli used in the Fine-Tracker simulations were provided by O. Scharenborg and are the same as those used in [13]. These stimuli came from the spoken sentences from the experiments by Salverda et al. (2003) and were cut manually so that they only contain the target word [10][13]. The stimuli consist of 28 multi-syllabic target words of which the first syllable is also a an embedded monosyllabic word, such as 'ham' in 'hamster'. There are two conditions for every word: the first is the MONO condition in which the first syllable is cross-spliced from a recording of the embedded word (e.g. 'ham'). In the CARRIER condition the first syllable is cross-spliced from another recording of the target word.

Acoustic features were computed for the stimuli as described in section 3.2. The acoustic features were then passed through the trained networks resulting

in 33 posterior probabilities, one for each class of each AF, for every 5 ms of speech. These AF vectors serve as the pre-lexical level representations of the speech signal. These AF vectors are the input materials for Fine-Tracker. The inputs are mapped onto the lexical representations according to the activation and competition process described in more detail section 3.6.

The durational information is hard-coded in Fine-Tracker's lexicon. There are two lexicons, one with and one without durational information (the 'canonical' and the 'duration' lexicon respectively). In the canonical lexicon the lexical feature representations for the embedded words and the first syllable of the target words are identical. The phonemes of the lexical representations are represented by each phoneme's canonical AF vector (see table 1 for the phoneme to AF mapping). In the duration lexicon the lexical representations for the embedded word and the first syllable of the target word are different. To accommodate the use of durational information each phoneme in the embedded words was represented by two identical AF vectors in the duration lexicon. An example of how durational information is encoded in the lexicon is shown in figure 13. The lexicon used in this study contained only the target words and the embedded words for a total lexicon size of 56.

Canonical lexicon

ham	AF vector
h	100101...
A	010111...
m	101000...
hamster	AF vector
h	100101...
A	010111...
m	101000...
s	001100...
t	111001...
@	101010...
r	101001...

Duration lexicon

ham	AF vector
h	100101...
h	100101...
A	010111...
A	010111...
m	101000...
m	101000...
hamster	AF vector
h	100101...
A	010111...
m	101000...
s	001100...
t	111001...
@	101010...
r	101001...

Figure 13: Lexical representations of ham and hamster in the canonical lexicon (left) and the duration lexicon (right). In the duration lexicon the embedded word is represented by two identical AF vectors for each phoneme.

3.6 Word activation and competition process

Fine-Tracker's word activation and competition process is implemented as a probabilistic word search [13]. This process maps the pre-lexical representations (i.e. the AF vectors derived from the acoustic signal) onto the lexical representations (i.e. canonical AF representations of words). The lexicon is

represented as a tree of feature vectors. When a node in the lexicon is accessed all words starting with the same sequence of feature vectors become equally activated. The word search algorithm is a breadth first search of the lexical tree.

The word search starts at the root of the lexical tree and child nodes are created by two mechanisms [13]. The first is to take a step in the input but not in the lexical tree. This results in multiple pre-lexical feature vectors being mapped to one lexical feature vector. The word search can also take a step in both the input and the lexical tree. Once the end of a word has been reached, the search starts over at the root of the lexical tree. This loop is repeated until the end of the input sequence has been reached.

The goal of the word search is to find the cheapest path through the search space. The cost of each node is given by a distance measure that indicates the goodness-of-fit between the pre-lexical and lexical feature vectors. The distance measure used by Fine-Tracker is the averaged squared distance (ASD) given by:

$$ASD = \frac{\sum_{AdmComp}(Lexval - Preval)^2}{\#Admissible}$$

Where $Lexval$ and $Preval$ are the lexical and pre-lexical feature vectors. The distance is only calculated over the 'admissible' AFs. For example for a lexical representation of a consonant, the vowel height features 'low', 'mid' and 'high' are inadmissible and not included in computing the distance.

The word score it then calculated by:

$$word\_score = \sum_{prelex\_feature\_vectors} SV + \alpha ASD$$

Where ASD is the averaged squared distance, α is a weight for the distance measure, and SV can be either SI (step-in-input) or SIL (step-in-input-and-lexicon) a value indicating the cost of these two actions.

Furthermore, Fine-Tracker has a word entrance penalty (the cost of starting a new word) and a word-not-finished penalty. The word-not-finished penalty is a penalty applied at the end of the input sequence to any search paths ending in an unfinished word. Fine-Tracker outputs an N-best list of predictions after every frame of the input sequence. This allows for the evaluation of the word activations over time. At the end of the input sequence a list of final predictions is made where any word-not-finished penalties are applied. The table below lists the settings for the parameters discussed above. These parameter settings were taken from [13] in order to allow for a fair comparison with previous results.

word entrance penalty	.5
word-not-finished penalty	10
α	1
step-in-input	0.2
step-in-input-and-lexicon	0.0

Table 4: Fine-Tracker parameter settings

3.7 Fine-Tracker simulation setup

The goal of the Fine-tracker simulations is to model the findings by Salverda et al. (2003) that pictures representing an embedded word attracted relatively more eye fixations from listeners in the MONO condition than in the CARRIER condition [10]. They claimed that this effect was governed by the duration of stimulus sequence so that longer sequences are more often interpreted as a monosyllabic word.

Fine-Tracker was tested in two conditions, that is, with and without the ability to use durational information. This is done by letting Fine-Tracker use the canonical lexicon (without durational information) or the adapted duration lexicon. Considering the behavioural results by Salverda et al. (2003) and the results of the previous Fine-Tracker simulations by Scharenborg (2010), the expectation is that word activations for the embedded words are higher in the MONO condition than the word activations for the embedded words in the CARRIER condition [10][13].

Firstly Fine-Tracker's word recognition performance is evaluated. In previous Fine-Tracker simulations reported by Scharenborg (2010) the correct target word was not always Fine-Tracker's top prediction. The expectation is that improving the quality of the AF vectors that are input to Fine-Tracker will improve improve Fine-Tracker's word recognition performance such that the target word is more often Fine-Tracker's top prediction.

In order to evaluate Fine-Tracker's simulation performance, the word activations of the embedded words over time are compared on the MONO and CARRIER conditions. In a correct simulation, the word activation of the embedded word is higher in the MONO condition than in the CARRIER condition. The effect of using durational information is investigated by comparing the number of correct simulations with and without durational information. If durational information can indeed be used to disambiguate the embedded and target words, using the duration lexicon should result in more correct simulations than using the canonical lexicon. The results for the MLPs and CNNs are compared in order to investigate the effects of better AF classification on the simulation performance of Fine-Tracker. As Fine-Tracker was made in order to model human speech recognition, the simulation results are then compared to the human behavioural data reported by Salverda et. al. (2003) [10].

4 Results

This chapter starts with the analysis of the different DNN architectures. First the new baseline MLPs are compared to the results reported by Scharenborg (2010) [13]. Next the performance of the three DNN architectures outlined in the previous chapter is compared. Lastly the Fine-Tracker simulations made with the baseline MLPs and the best performing CNN architecture are analysed.

4.1 Baseline performance

The new baseline MLPs were created in order to account for any effects caused by the differences in training data used in the current study and in the training data used in [13]. The AF classification scores for the new baseline MLPs and the results reported by Scharenborg (2010) are shown in table 5 [13].

Articulatory feature	baseline MLP accuracy (%)	Scharenborg 2010 accuracy (%)
Manner	73.98	76.6
Place	72.74	76.2
Voicing	90.18	89.3
Backness	81.10	77
Height	81.82	82.5
Rounding	83.68	79.6
Duration-diphthong	80.33	79

Table 5: Classification accuracy on the test set of the MLPs trained on MFFCs. The left column are the results of the new baseline MLPs, the right column shows the results reported by Scharenborg (2010) [13].

As you can see in the table above, the performance of the new MLPs is similar to results reported in [13]. The newly trained baseline MLPs perform better on backness and rounding, worse on manner and place of articulation with minor differences on the other features. So simply using more data by itself did not improve AF classification.

4.2 Comparison of MLPs and CNNs

Next the baseline results are compared to the results of the two CNN architectures. First the relative size of each class label per AF is analysed. The relative size of the majority class can be thought of as the performance of a naive classifier which assigns the majority label to all the data (a chance level performance). Furthermore, large class imbalances can cause a classifier to become biased towards the majority class. The relative size of each class label per AF is shown below in table 6. The largest class for each AF is indicated in boldface. It is notable that every AF except voicing has a clear majority class that is much larger than the other classes.

Articulatory feature	Class size (%)							
Manner	vowel **29.81**	plosive 13.28	fricative 11.38	glide 3.44	liquid 2.68	nasal 8.06	retroflex 4.01	silence 27.32
Place	nil **29.81**	bilabial 4.19	alveolar 26.29	labiodental 3.80	velar 6.24	glottal 1.49	palatal 0.83	silence 27.32
Voicing	voiced **53.50**	unvoiced 46.50						
Backness	central 11.12	back 8.48	front 10.21	nil **70.19**				
Height	mid 15.55	low 11.54	high 2,72	nil **70.19**				
Rounding	unrounded 16.75	rounded 13.06	nil **70.19**					
Duration-diphthong	short 16.11	long 10.46	diphthong 3.24	nil **70.19**				

Table 6: Relative sizes of the classes within each AF.

Table 7 shows the classification results per AF for the baseline and the two CNN architectures. The first row indicates the chance level performance based on the size of the majority class for each AF and the second row indicates the performance of the baseline MLPs. CNN indicates the CNN architecture described in [18], CNN Mf indicates the architecture extended with the Mel filter layer. The classification results of all models were well above chance level performance. Furthermore, both CNN architectures are a clear improvement over the MLPs on every AF. The CNN architecture had the best performance on all AFs. The bottom rows of the table show the absolute and relative improvement of the best results over the baseline MLPs, with the biggest improvements on manner and place of articulation and the smallest improvement on voicing. As the CNN architecture outperformed the CNN Mf architecture on every AF, further analyses and the Fine-Tracker simulations were done for the baseline MLPs and the CNN architecture only (an analysis and discussion of the CNN Mf architecture and the trained Mel filter-banks can be found in appendix B).

	Metric	Articulatory feature						
		manner	place	voicing	backness	height	rounding	duration-diphthong
	chance level %	29.81	29.81	53.50	70.19	70.19	70.19	70.19
MLP baseline	accuracy (%)	73.98	72.74	90.18	81.10	81.82	83.68	80.33
CNN	accuracy (%)	**86.90**	**86.28**	**93.52**	**89.20**	**89.17**	**90.63**	**88.18**
CNN Mf	accuracy (%)	84.58	84.89	92.87	86.57	86.88	88.55	86.53
	abs improvement (%)	12.92	13.54	3.34	8.10	7.35	6.95	7.85
	rel improvement (%)	17.46	18.61	3.70	9.99	8.98	8.30	9.77

Table 7: Table of classification results per AF for the baseline MLPs, CNNs and the CNNs extended with trainable Mel-filters. The last two rows indicate the absolute and relative improvements of the best results (printed in boldface) over the MLP baseline results.

Next we look at the classification performance for each class label per AF, in order to see whether the classification performance is biased towards the majority class. Figure 14 and 15 show the normalised confusion matrices for the baseline MLPs and CNNs respectively. The scores on the diagonals indicate the classification performance for each individual class label. The scores off the diagonal indicate the confusion, that is, the percentage of examples that were miss-classified as another class.

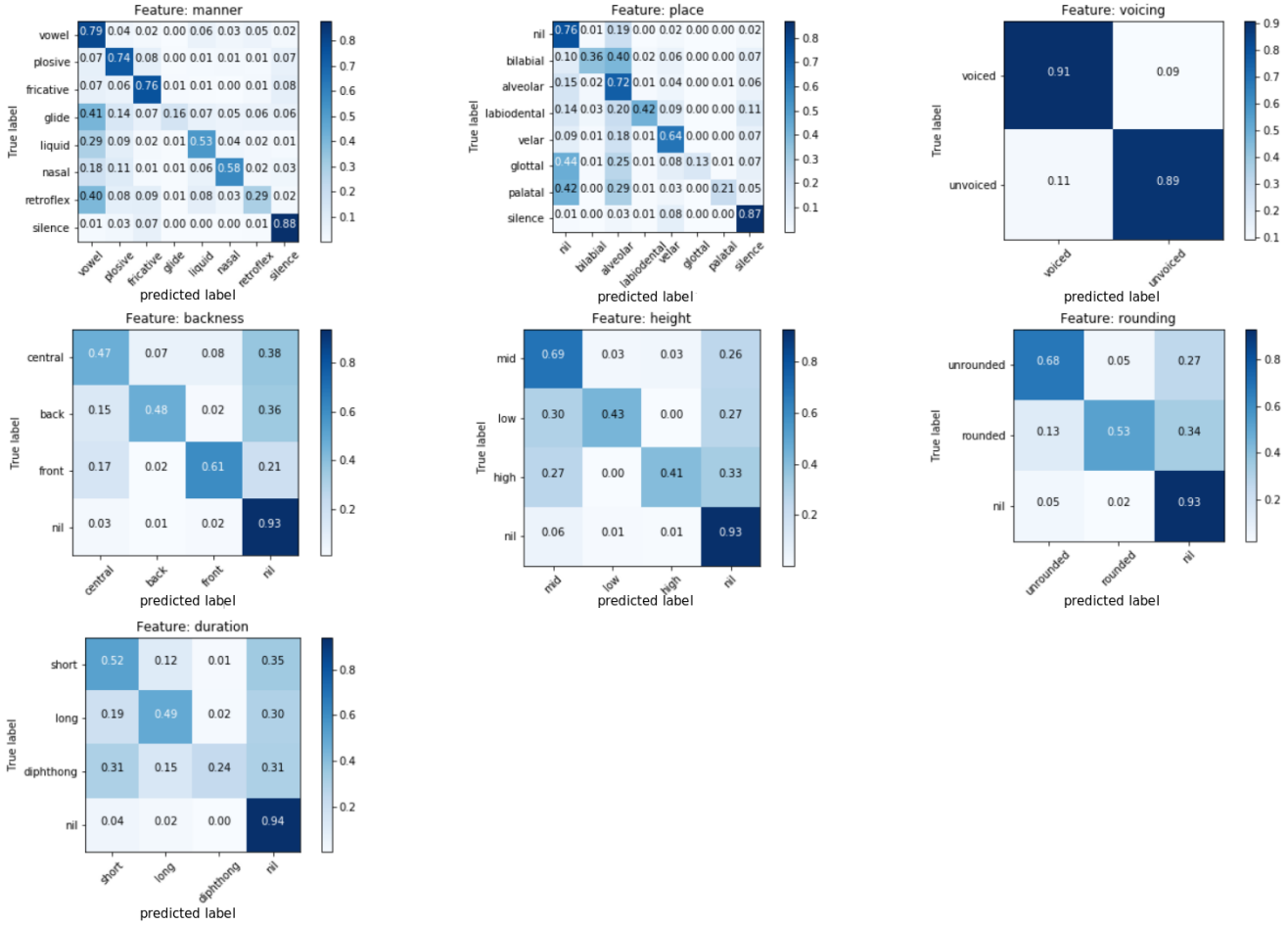

Figure 14: Normalised confusion matrices for the MLPs. The diagonal corresponds to the per class accuracy.

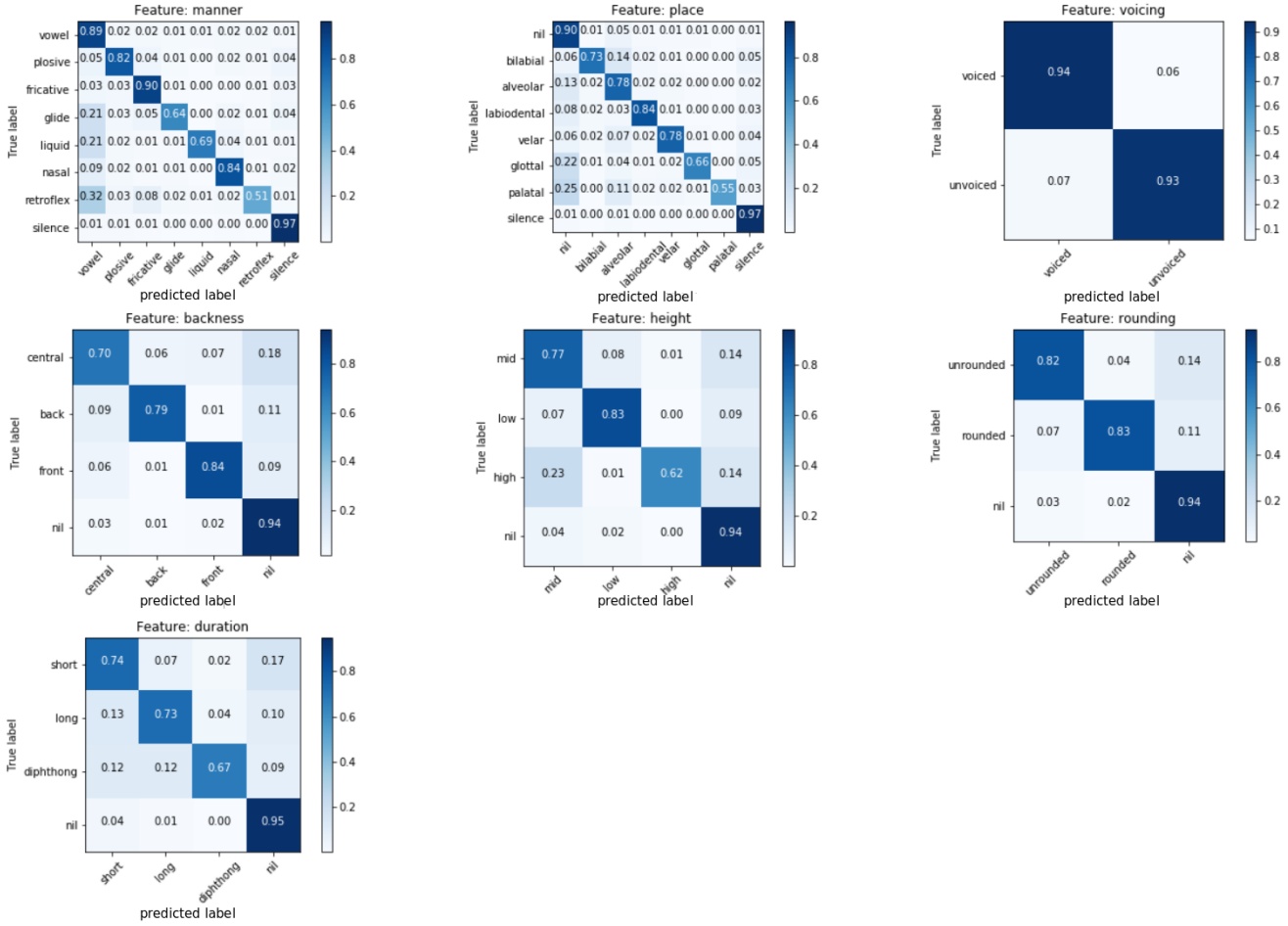

Figure 15: Normalised confusion matrices for the CNNs. The diagonal corresponds to the per class accuracy

The bias towards majority classes is biggest in the baseline MLPs, where 10 out of 33 labels are more often miss-classified than correctly classified. The most extreme cases are in the manner and place classifiers. Over 40% of the occurrences of the glide and retroflex for manner and the glottal and palatal for place are miss-classified as the majority label.

The CNNs are an improvement over the baseline on every single class, with larger relative improvements on the minority classes than on the majority classes. For example, classification accuracy of the glottal increased from 13% to 66% and the confusion with the majority class decreased from 44% to 22% compared to the baseline. For the CNNs all 33 class labels were more often correctly classified than miss-classified.

4.3 Fine-Tracker simulation results

In order to simulate speech recognition, Fine-Tracker has to be able to correctly identify the target and embedded words. The table below lists the number of words that were correctly recognised by Fine-Tracker for each condition and DNN architecture.

Model	condition	lexicon	embedded words correct	target words correct
Scharenborg 2010	MONO	canonical	28	28
		duration	28	28
	CARRIER	canonical	28	28
		duration	28	28
MLP	MONO	canonical	28	28
		duration	28	28
	CARRIER	canonical	27	27
		duration	27	27
CNN	MONO	canonical	28	28
		duration	28	28
	CARRIER	canonical	28	28
		duration	28	28

Table 8: Table listing Fine-Tracker's word recognition results. The first column indicates the models used to create the AF vectors, the second column lists the stimulus condition and the third column indicates which lexicon was used.

Using the CNN AF vectors all 28 target and 28 embedded words were correctly recognised for both conditions and lexicons. For the MLP AF vectors, all 28 target and 28 embedded words were correctly recognised only in the MONO conditions for both lexicons. In the CARRIER condition, 27 words were correctly recognised. For both lexicons, the embedded word 'ham' was not recognised and the target word 'lampekap' was not recognised. Note that these results indicate whether a word was recognised at all at some point during the acoustic input.

The word recognition performance is further evaluated by looking at the list of final predictions at the end of the acoustic input. The table below lists the number of simulations for which the target word was in the final 50 best list of predictions for each condition and lexicon. The number between brackets is the number of times the target word was the top prediction.

condition	Scharenborg 2010		MLP		CNN	
	canonical	duration	canonical	duration	canonical	duration
MONO	27 (20)	25 (15)	24 (16)	19 (11)	28 (23)	23 (19)
CARRIER	23 (21)	22 (16)	22 (19)	22 (16)	28 (23)	21 (18)

Table 9: Table listing the number of simulations where the target word was among the final predictions for each condition and lexicon. The number between brackets indicates how many times the target word was the top prediction.

The word recognition was worst for the MLPs both in terms of words that were not in the final 50 best list and the number of words that were the top prediction. Furthermore, the word recognition of the CNNs is an improvement over the results reported by Scharenborg (2010) [13]. For the canonical lexicon all words appeared in the final predictions and 23 out of 28 words were the top prediction. However, for the durational lexicon the number of words appearing in the final results is slightly lower than the results reported by Scharenborg (2010) [13]. The CNN still performs better in terms of words that appeared in the final results as the top prediction.

In order to investigate the modelling ability of Fine-Tracker, the word activations are compared over time for the MONO and the CARRIER condition using the canonical lexicon and the duration lexicon. Table 10 shows in which condition (MONO or CARRIER) the embedded word had the highest activation over time. The decision is based on a comparison of the word activations over time; the 'winning' condition had the highest activation for the largest part of the stimulus.

embedded / target	Scharenborg 2010		MLP		CNN		Effect size human data
	canonical	duration	canonical	duration	canonical	duration	
bij / beitel	CARRIER	MONO	MONO	MONO	CARRIER	MONO	.19
blik / bliksem	CARRIER	CARRIER	MONO	MONO	MONO	MONO	.01
bok / bokser	MONO	MONO	MONO	MONO	MONO	MONO	.12
ei / eikel	CARRIER	CARRIER	CARRIER	CARRIER	CARRIER	CARRIER	.01
ham / hamster	CARRIER	CARRIER	X	X	MONO	MONO	.01
hen / hendel	CARRIER	MONO	CARRIER	MONO	CARRIER	CARRIER	-.30
kan / kandelaar	CARRIER	MONO	MONO	MONO	CARRIER	CARRIER	.22
kei / kijker	CARRIER	MONO	MONO	MONO	CARRIER	MONO	-.01
knip / knipsel	MONO	MONO	CARRIER	MONO	MONO	MONO	-.03
koe / koekepan	CARRIER	CARRIER	MONO	MONO	CARRIER	CARRIER	.23
kok / cocktail	CARRIER	CARRIER	CARRIER	CARRIER	MONO	MONO	.07
kom / compact-disc	MONO	MONO	MONO	MONO	MONO	MONO	.08
la / lama	CARRIER	MONO	CARRIER	CARRIER	CARRIER	CARRIER	.05
lam / lampekap	MONO	MONO	CARRIER	CARRIER	CARRIER	CARRIER	-.08
lei / leiding	CARRIER	CARRIER	MONO	MONO	CARRIER	CARRIER	-.02
man / mantel	MONO	MONO	CARRIER	CARRIER	CARRIER	MONO	.09
pan / panda	CARRIER	CARRIER	CARRIER	MONO	CARRIER	CARRIER	.03
pen / panty	MONO	MONO	MONO	CARRIER	CARRIER	CARRIER	-.03
pin / pinda	MONO	MONO	CARRIER	MONO	CARRIER	MONO	-.10
ree / regenton	CARRIER	MONO	CARRIER	CARRIER	CARRIER	MONO	.14
roos / rooster	MONO	MONO	MONO	MONO	MONO	CARRIER	-.01
schil / schilder	MONO	MONO	CARRIER	MONO	CARRIER	CARRIER	.20
sla / slager	CARRIER	CARRIER	MONO	MONO	CARRIER	CARRIER	.10
snor / snorker	CARRIER	MONO	MONO	CARRIER	CARRIER	MONO	.00
tak / taxi	CARRIER	CARRIER	CARRIER	CARRIER	CARRIER	MONO	.26
thee /tegel	CARRIER	CARRIER	CARRIER	CARRIER	CARRIER	MONO	.13
tor / torso	CARRIER	MONO	CARRIER	MONO	MONO	MONO	-.02
zee / zebra	CARRIER	CARRIER	CARRIER	CARRIER	MONO	CARRIER	.12
MONO total	9	17	12	16	9	15	18

Table 10: The results of the Fine-Tracker simulations. The first column lists the embedded and target words. The next three columns list the results by Scharenborg (2010) and the results of the current study using the MLP and DNN AF vectors respectively [13]. The last column indicates the effect size of the human data which is the difference between the two conditions in average proportions of eye fixations to the embedded words. The bottom row shows the total number of times that the MONO condition had the highest word activation over time.

The table above show the results for both the MLPs and CNNs along with the results reported by Scharenborg in [13]. Canonical and duration refer to the use of the canonical lexicon (i.e. Fine-Tracker cannot use durational informa-

tion) and the duration lexicon (Fine-Tracker can use durational information). MONO and CARRIER indicate in which condition the embedded word had the highest activation over time. As noted earlier the word 'ham' was not completely recognised by Fine-Tracker using the MLP AF vector and as such this word is excluded for the MLPs. For the canonical lexicon, the MONO condition won 12 times for the MLPs and 10 times for the CNNs. When the duration lexicon was used, these numbers increased to 16 for both the MLPs and CNNs. These numbers are similar to those reported by Scharenborg (2010) (9 and 17 for the canonical and duration lexicon respectively). A one-tailed McNemar for paired samples (without continuity correction) was done in order to see if the effect of durational information is significant. The stimuli are paired for the canonical and duration condition with a 1 indicating when MONO won and a 0 for CARRIER. The test for the MLP results was not significant ($\chi^2 = 2.0$, p = 0.0786). The test for the CNN results was significant ($\chi^2 = 3.6$, p = 0.0288). However the results for the more conservative McNemar test with continuity correction was not significant for the CNNs ($\chi^2 = 2.5$, p = 0.0569).

In line with the results of Salverda et al. (2003) the effect of durational information was not equal for all stimuli [10]. The last column of the table shows the effect size on the human data as a difference in the average proportion of eye fixations to the image representing the embedded word. A positive number indicates more fixations on the embedded word in the MONO condition. The effect was positive for 18 of the stimuli. While there are some similarities between the Fine-Tracker simulations and the human data the Fine-Tracker results do not agree with the human data on every stimulus. The Fine-Tracker results agree with the human data on 10 out of 18 cases for the CNNs and 9 out of 18 times for the MLPs and 8 out of 18 times for the results reported by Scharenborg (2010) [13].

5 Discussion

The goal of this study was to improve the classification of articulatory features and investigate the effects of articulatory feature quality on Fine-Tracker's word recognition and simulation performance. To this end three types of neural networks were trained. Firstly new MLP baselines were trained according to the architecture described in [13]. The baseline was created in order to account for the fact that this study used more data to train the models. However, simply using more training data did not increase the AF classification performance. There were only minor differences in the classification accuracy but if anything the new baselines performed slightly worse than the MLPs reported in [13]. This result is unexpected as using more training data typically increases the performance and generalisability of DNNs. A possible explanation is that the number of hidden nodes for each AF was optimised for the data-set used in [13] by tuning experiments. The optimal number of hidden nodes may be different for the data-set used in the current study.

In order to improve the AF classification two CNN architectures were in-

vestigated. The first architecture was implemented as described in [18]. Furthermore, an extension to this network, where part of the pre-processing is integrated as a convolutional layer was implemented. Both CNN architectures were a clear improvement over the baseline MLPs. The basic CNN architecture gave the best classification results for all AFs with relative improvements of up to 18.61% over the baseline. While the basic CNN architecture outperformed the extended CNN architecture the differences were only minor. Research shows that the random initialisation of the network weights can have a large effect on the performance of DNNs, especially when they are trained with momentum [57]. It is possible that the CNN Mf architecture would have performed better with a different initialisation of the network weights.

Further investigation of the baseline MLPs and the best performing CNNs showed that the improvements in classification accuracy were bigger for the minority classes than for the majority classes. There were clear majority classes for six of the seven AFs and an investigation of the confusion matrices showed that the classifiers were biased towards the majority class. This bias was so large for the MLPs that some classes were more often miss-classified as the majority label than they were correctly classified.

This bias decreased for the CNNs where all classes were more often correctly classified than miss-classified. Confusion with the majority class decreased on all AFs. This is an important improvement of the quality of the AF vectors. For instance the majority class for backness, height, rounding and duration-diphthong is 'nil', a label assigned to all consonants and silence. While distinguishing between vowels on the one hand and consonants and silence on the other is necessary in order to be able to distinguish between the different characteristics of vowels, the interesting information such as vowel height is actually captured by the minority labels. If these classifiers are biased towards the majority class they are all fairly accurate at detecting consonants but they provide less accurate information on the characteristics of the vowels. In this sense the CNNs are even more of an improvement than the overall accuracy would suggest.

Speech recognition simulations were carried in order to investigate the effects of improved AF classification on Fine-Tracker's word recognition and modelling performance. The simulations used the acoustic material from the original behavioural studies by Salverda et al. (2003) [10]. As expected, the higher quality of the AF vectors resulted in improved word recognition performance. For the canonical lexicon, all of the target words appeared in Fine-Tracker's final predictions with 23 words appearing as the top prediction which is the best recognition performance reported so far.

The simulations also showed that using durational information allows Fine-Tracker to distinguish the embedded words from their respective target words. For both the MLPS and CNNs, the use of durational information resulted in more correct simulations than without durational information. This is in line with results reported by Scharenborg (2010). However, this difference was not significant for the MLPs and only significant for the CNNs when using the less conservative mcNemar test without continuity correction. The simulations

were also compared to the human behavioural data in order to investigate Fine-Tracker's ability to model the behavioural data. This showed that the both the MLPs and CNNs agreed with the human data more often than the simulations reported by Scharenborg (2010), the differences were very small however [13].

The results suggest that to some degree Fine-Tracker's modelling performance is dependant on the quality of the AF vectors. The MLPs, which showed the worst AF classification and word recognition results, also showed the worst modelling results. However, even though the CNNs showed large improvements on the classification of the AFs and the best word recognition results, they did not increase Fine-Tracker's ability to model human behavioural data compared to the results by Scharenborg (2010) [13]. It is not clear why the improved AF classification and word recognition does not result in better modelling of the use of durational information. In order to make a fair comparison to previous Fine-Tracker results the simulations setting as used by Scharenborg (2010) were used [13]. Perhaps these parameters need to be tuned to the new AF vectors in order to allow Fine-Tracker to make better use of the increased quality of the AF vectors.

In conclusion the convolutional neural networks gave a remarkable improvement in the classification of articulatory features over both the baseline and previously reported results. While this led to better word recognition for Fine-Tracker simulations, it did not improve Fine-Trackers modelling capabilities regarding the use of durational information. A possible avenue for future research is to investigate whether Fine-Tracker is sensitive to the simulation parameter settings and whether these settings need to be tuned to the input.

6 Acknowledgements

I would like to thank my supervisor dr. Odette Scharenborg for including me into her research group and for all her help with this thesis. I have had more questions than I can remember and the door to your office was literally always open. I would also like to thank dr. Martha Larson and dr. Elena Marchiori for their advice and feedback.

Training all these neural networks on my laptop would have been impossible, so a special thanks to the SURF corporation for giving me access to their GPUs.

7 references

[1] Scharenborg, O., & Boves, L. (2010). Computational modelling of spoken-word recognition processes. Design choices and evaluation. Pragmatics& Cognition 18:1 (2010), 136–164.
[2] Jurafsky, D. & Martin, J. H. (2009). Speech and language processing (2nd ed.). Upper Saddle River, NJ: Pearson Education.
[3]Bent, T. & Holt, Ra. (2017). Representation of speech variability: Speech variability. Wiley Interdisciplinary Reviews: Cognitive Science. e01434. 10.1002/wcs.1434.

[4] https://www.cnet.com/how-to/common-amazon-alexa-problems-and-how-to-fix-them/ Retrieved on 26/09/2017
[5] Vroomen, J., de Gelder, B. (1997). Activation of Embedded Words in Spoken Word Recognition, 23(3), 710–720.
[6] Gow, D. & Gordon, P. C. (1995). Lexical and prelexical influences on word segmentation: Evidence from priming. Journal of experimental psychology. Human perception and performance. 21. 344-59. 10.1037//0096-1523.21.2.344.
[7] Allopenna, P., Magnuson, J. & Tanenhaus, Michael. (1998). Tracking the Time Course of Spoken Word Recognition Using Eye Movements: Evidence for Continuous Mapping Models. Journal of Memory and Language. 38. 419-439. 10.1006/jmla.1997.2558.
[8] McClellan, J. L. & Elman, J. L. (1986). The Trace Model of Speech Perception. Cognitive Psychology, 18, 1-86.
[9] Norris, D. G. (1994). Shortlist: A connectionist model of continuous speech recognition. Cognition, 52, 189-234.
[10] Salverda, A. P., Dahan, D., and McQueen, J. M. (2003). The role of prosodic boundaries in the resolution of lexical embedding in speech comprehension. Cognition 90, 51–89.
[11] Davis, M., Marslen-Wilson, W. & Gaskell, G. (2002). Leading up the lexical garden path: Segmentation and ambiguity in spoken word recognition. Journal of Experimental Psychology: Human Perception and Performance. 28. . 10.1037//0096-1523.28.1.218-244.
[12] Salverda, A. P., Dahan, D., Tanenhaus, M., Crosswhite, K., Masharov, M. & McDonough, J.. (2007). Effects of prosodically modulated subphonetic variation on lexical competition. Cognition. 105. 466-76. 10.1016/j.cognition.2006.10.008.
[13] Scharenborg, O. (2010). Modeling the use of durational information in human spoken-word recognition. J Acoust Soc Am, 127(6), 3758–3770.
[14] Weber, A., & Scharenborg, O. (2012). Models of spoken-word recognition. Wiley Interdisciplinary Reviews: Cognitive Science, 3(3), 387–401.
[15] Siniscalchi, S. M., Yu, D., Deng, L., & Lee, H. (2012). Exploiting Deep Neural Networks for Detection-Based Speech Recognition, 106, 148–157.
[16] Rath, S. P., Povey, D., Vesely K., & Cernock, C. (2013). Improved feature processing for Deep Neural Networks
[17] Abdel-Hamid, O., Mohamed, A., Jiang, H. & Penn, G. (2012). Applying Convolutional Neural Networks concepts to hybrid NN-HMM model for speech recognition. Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on. 4277-4280. 10.1109/ICASSP.2012.6288864.
[18] Qian, Y. & Woodland, P. (2016). Very Deep Convolutional Neural Networks for Robust Speech Recognition.
[19] Kirchoff, K. (1999). Robust Speech Recognition Using Articulatory Information. Dissertation. Faculty of Technology, University of Bielefeld.
[20] Mitra, V., Sivaraman, G., Nam, H., Espy-Wilson, C., Saltzman, E. & Tiede, M. (2017). Hybrid convolutional neural networks for articulatory and acoustic information based speech recognition. Speech Communication. 89. 103-112. 10.1016/j.specom.2017.03.003.

[21] Mitra, V., Wang, W., Stolcke, A., Nam, H., Richey, C., Yuan, J. & Liberman, M. (2013). Articulatory trajectories for large-vocabulary speech recognition. Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on. 7145-7149. 10.1109/ICASSP.2013.6639049.
[22] Kirchhoff, K., Fink, G. A. & Sagerer, G. (2002). Combining acoustic and articulatory feature information for robust speech recognition. Speech Communication. 37. 303-319. 10.1016/S0167-6393(01)00020-6.
[23] Badino, L., Canevari, C., Fadiga, L. & Metta, G. (2015). Integrating Articulatory Data in Deep Neural Network-based Acoustic Modeling. Computer Speech & Language. 36. . 10.1016/j.csl.2015.05.005.
[24] King, S., Frankel, J., Livescu, K., McDermott, E., Richmond, K. & Wester, M. (2007). Speech production knowledge in automatic speech recognition. The Journal of the Acoustical Society of America. 121. 723-42.
[25] Westbury, J. R., (1994). X-ray Microbeam Speech Production Database User's Handbook. Waisman Center on Mental Retardation and Human Development. University of Wisconsin, Madison, WI, USA, version 1.0 edition.
[26] Wrench, A. A., (2000). Multi-channel/multi-speaker articulatory database for continuous speech recognition research. Phonus 5, 1–13.
[27] Chen, L., Mao, X., Wei, P. & Compare, A. (2013). Speech Emotional Features Extraction Based on Electroglottograph. Neural computation. 25. 10.1162/NECO_a_00523.
[28] Yoshioka, H., Löfqvist, A., & Hirose, H. (2008) Laryngeal adjustments in the production of consonant clusters and geminates in American English. Journal of the Acoustical Society of America 70: 1615-1623, 1981.
[29] Ji, A. (2014). Speaker Independent Acoustic-to-Articulatory Inversion. Dissertation. Marquette University.
[30] Ghosh, P. & Narayanan, S. (2011). Automatic speech recognition using articulatory features from subject-independent acoustic-to-articulatory inversion. The Journal of the Acoustical Society of America. 130. EL251-7. 10.1121/1.3634122.
[31] Hasegawa-Johnson, M., Baker, J., Borys, S., Chen, K., Coogan, E., Greenberg, S., Juneja, A., Kirchhoff, K., Livescu, K., Mohan, S., Muller, J., Sönmez, M. & Wang, T. (2005). Landmark-based speech recognition: Report of the 2004 Johns Hopkins summer workshop. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing / sponsored by the Institute of Electrical and Electronics Engineers Signal Processing Society. ICASSP. 1. 1213-1216. 10.1109/ICASSP.2005.1415088.
[32] Rietveld, A. C. M. & van Heuven, V. J. (1997). Algemene fonetiek (1st ed.). Bussum, Nederland: Coutinho.
[33] Ladefoged, P. (1975). A course in Phonetics. Hartcourt Brace Jovanovich International Edition.
[34] Catford, J. C. (1977). Fundamental problems in phonetics (1st ed.). Edinburgh, Scotland: Edinburgh University Press.
[35] Mazharul Islam, A. K. M. (2015). Speaking: A Review. International Journal of Humanities and Cultural Studies. 2. 3. 423-435.

[36] Sebregts, K. (2014). The Sociophonetics and Phonology of Dutch r. Dissertation. University of Utrecht
[37] Russel, S. & Norvig, P. (2010). Artificial intelligence: a modern approac. Upper Saddle River, NJ: Pearson Education.
[38] Nielsen, M. (2017). Neural networks and deep learning. Only (and freely) available on: http://neuralnetworksanddeeplearning.com
[39] Botev, A., Lever, G. & Barber, D. (2016). Nesterov's Accelerated Gradient and Momentum as approximations to Regularised Update Descent.
[40] Svozil, D., Kvasnicka, V. & Pospíchal, J. (1997). Introduction to multilayer feed-forward neural networks. Chemometrics and Intelligent Laboratory Systems. 39. 43-62. 10.1016/S0169-7439(97)00061-0.
[41] Bengio, Y. & Lecun, Y. (1997). Convolutional Networks for Images, Speech, and Time-Series.
[42] Sainath, T. N., Mohamed, A., Kingsbury, B. & Ramabhadran, B. (2013). Deep convolutional neural networks for LVCSR. Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on. 8614-8618. 10.1109/ICASSP.2013.6639347.
[43] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 15. 1929-1958.
[44] Krizhevsky, A., Sutskever, I. & Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. 1097-1105.
[45] Jalali, A., Mallipeddi, R. & Lee, M. (2017). Sensitive Deep Convolutional Neural Network for Face Recognition at Large Standoffs with Small Dataset. Expert Systems with Applications. 87. . 10.1016/j.eswa.2017.06.025.
[46] Adeel Waris, M., Iosifidis, A. & Gabbouj, M. (2017). CNN-based Edge Filtering for Object Proposals. Neurocomputing. . 10.1016/j.neucom.2017.05.071.
[47] Jenisha, T. & Swarnalatha, P. (2016). A survey of neural network algorithms used for image annotation. 7. 236-252.
[48] Manli, S., Song, Z., Jiang, X., Pan, J. & Pang, Y. (2016). Learning Pooling for Convolutional Neural Network. Neurocomputing. 224. . 10.1016/j.neucom.2016.10.049.
[49] Li, J., Deng, L., Haeb-Umbach, R. & Gong, Y. (2015). Robust automatic speech recognition: A bridge to practical applications.
[50] Tompson, J., Goroshin, R., Jain, A., Lecun, Y. & Bregler, C. (2014). Efficient Object Localization Using Convolutional Networks.
[51] Arpit, D., Zhou, Y., Kota, B. U. & Govindaraju, V. (2016). Normalization Propagation: A Parametric Technique for Removing Internal Covariate Shift in Deep Networks.
[52] European Telecommunications Standards Institute. (2003). ETSI ES 201 108 V1.1.3. Available on www.etsi.org, retrieved on 23/11/2016.
[53] Duda, R. O., Hart, P. E. and Stork, D. G. (2000). Pattern classification. Wiley, November 2000.
[54] Gopinath, R. (1998). Maximum likelihood modeling with Gaussian distributions for classification. Proc. IEEE ICASSP, 1998, vol. 2, pp. 661–664.
[55] Psutka, J. V. (2007). Benefit of Maximum Likelihood Linear Transform (MLLT) Used at Different Levels of Covariance Matrices Clustering in ASR

Systems.
[56] Matsoukas, S., Schwartz, R., Jin, H. and Nguyen, L. (1997). Practical implementations of speaker-adaptive training. DARPA Speech Recognition Workshop, 1997.
[57] Sutskever, I., Martens, J., Dahl, G. & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. 30th International Conference on Machine Learning, ICML 2013. 1139-1147.
[58] McClellan, J. H., Schafer, R.W. & Yoder. M. A. (2003). Signal processing first (1st ed.). Upper Saddle River, NJ: Pearson

8 Appendix A

This appendix describes in more detail the pre-processing pipeline for the creation of the acoustic features. The pipeline is implemented as described by the ETSI protocol in [2]. The computation of the delta and double delta features is implemented as described in [52]. The block diagram below gives an overview of the pipeline. The pipeline is available on https://github.com/DannyMerkx /CGN_speech_recognition

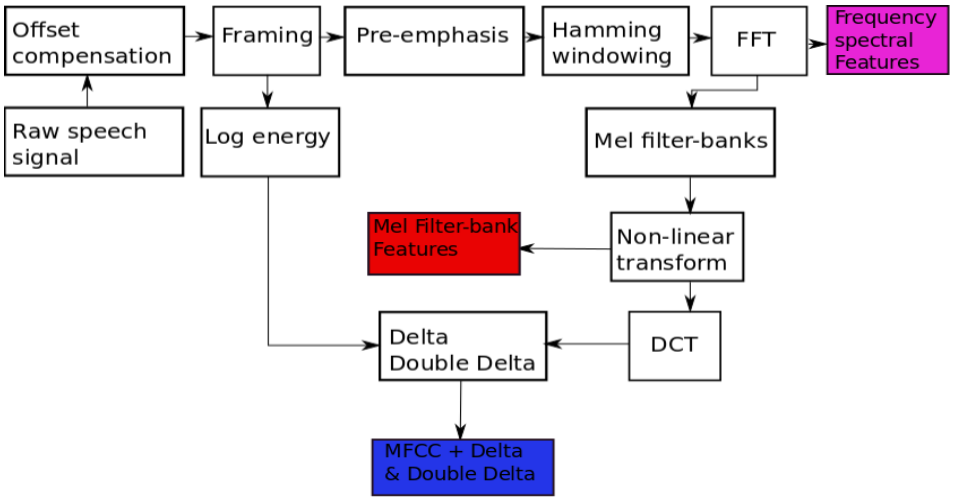

Figure 16: Block diagram of the pre-processing pipeline. The coloured blocks indicate the acoustic features. FFT stands for fast Fourier transform, DCT stands for discrete cosine transform.

8.1 Offset compensation

The first step is to apply a notch filter to remove the signal's DC offset. The notch filter is given by:

$$s_{of}(n) = s(n) - s(n-1) + 0.999 \times s_{of}(n-1) \tag{8}$$

Where s_{of} is offset free signal and $s(n)$ is input signal.

8.2 Framing

The offset free signal is then segmented into 25ms frames with 5ms frame shift (400 and 160 samples respectively at a sampling rate of 16kHz). The frame shift indicates the offset in start times of consecutive frames, and a shift lower than the frame size means that frames will partly overlap. The signal is padded with zeros if the last frame does not have enough samples left for a full frame.

8.3 Log energy

The next step is to take the natural logarithm of the frame energy as given by:

$$LogE = ln(\sum_{i=1}^{N} s_{of}(i)^2) \tag{9}$$

N is the number of samples in each frame and s_{of} is the offset free input signal. The log frame energy is later added to the MFCC feature vector.

In exceptional cases it is possible for the frame energy to be zero in which case we would take the natural logarithm of zero. This results in negative infinite, which causes problems for the cross entropy loss function during network training. Instead, the zero value will be replaced by 1e-22, a value very close to zero, the log of which results in approximately -50.

8.4 Pre-emphasis

After extracting the log energy feature, pre-emphasis is applied. The pre-emphasis filter is given by:

$$s_{pe}(n) = s_{of}(n) - 0.97 \times s_{of}(n-1) \tag{10}$$

Where n is the nth sample, s_{pe} is the pre-emphasised signal and s_{of} is the offset free input signal.

8.5 Hamming windowing

As the data is cut into 25ms frames, the signal is abruptly cut at the boundaries causing discontinuities in the signal. The FFT assumes that the finite frame of data it is given is a full period of a periodic series. What happens when the signal is discontinuous is called spectral leakage, causing artefacts looking like low amplitude peaks in the frequency domain that did not exist in the original signal [58].

Below is an example of the frequency spectrum for two different cuts of a 100Hz sine-wave, one at 1s (a full 10 periods) and the other at 0.78s (7.8 periods).

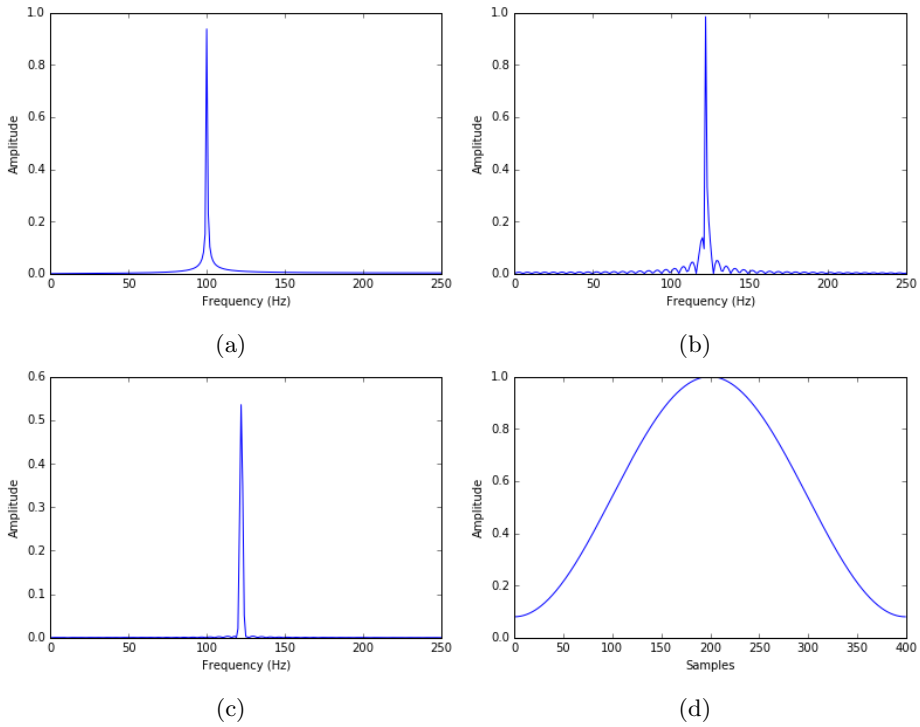

Figure 17: a) FFT of a 100Hz sine-wave without spectral leakage. b) FFT of a 100Hz sine-wave with spectral leakage due to discontinuities in the FFT input signal. c) The same signal with spectral leakage, but after applying the Hamming window. d) The Hamming window for a window of 400 samples.

As you can see in the figure above, the FFT without spectral leakage shows a large peak at 100Hz as is to be expected. figure 17b shows the artefacts introduced by taking a frame of the 100Hz signal which does not contain an integer number of periods.

A Hamming window is applied to counter the issue of spectral leakage. The Hamming window is given by:

$$s_{hw}(n) = (0.54 \times 0.46 \cos(\frac{2\pi n}{N-1})) \times s_{pe}(n) \tag{11}$$

Where N is the length of the frame in number of samples, s_{hw} is the Hamming windowed signal and $s_{pe}(n)$ is the pre-emphasised signal.

As opposed to the so-called square window described in section 3.2.2, the Hamming window compresses the values near the boundaries of the frames to-

wards 0. As you can see in figure 2c, applying the Hamming window does not solve the issue of spectral leakage, but it does reduce the artefacts.

8.6 Fast Fourier transform

After applying the Hamming window, the signal is converted to the frequency domain using a fast Fourier transform (FFT). The FFT is faster when the number of samples in each frame is a power of two; therefore each frame is zero padded at the end of the frame to the nearest power of two which is 512 for 25ms frames at a sampling rate of 16kHz.

The FFT transform is given by:

$$s_{fft}(k) = \sum_{n=0}^{N-1} s_{hw}(n) e^{-j2\pi nk/N} \tag{12}$$

Where $s_{fft}(k)$ is frequency bin k of the transformed signal, $s_{hw}(n)$ is the Hamming windowed signal, N is the number of samples in each (padded) frame and $k = 0, ..., N-1$.

Next the two sided amplitude spectrum is computed as given by:

$$Amp(k) = \frac{1}{N} |s_{fft}(k)| \times 2 \tag{13}$$

where $Amp(k)$ is the amplitude of bin k and N is the number of samples in each (padded) frame. This takes the absolute value of the FFT and normalises it for the length of the input signal. Furthermore, it is multiplied by two to account for the fact that we only retain the positive part of the FFT.

Because the FFT is computed on a discrete-time signal, the bins are symmetrical or 'mirrored' around the Nyquist frequency [24] [58]. The first bin (the DC component) and the Nyquist frequency[3] bin are both unique; of the mirrored bins we only retain the first half. In a setup with 25ms frames and a 16kHz sampling rate this leaves us with 257 bins. Because the DC component and Nyquist frequency are unique FFT bins, they should not be multiplied by two in equation 6. The resulting FFT bins are the frequency spectral features used as input features for the extended CNN architecture.

8.7 Mel filter-banks

Next, Mel-spaced filter-banks are created and applied to the FFT amplitudes. Frequencies that are judged to be equal in perceptual distance are all equidistant

[3]The Nyquist frequency is defined as the sampling frequency divided by two. According to the Nyquist-Shannon theorem, only frequencies up to the Nyquist frequency can be reconstructed with perfect fidelity. CGN is recorded at 16kHz meaning we can reconstruct signals up to 8Khz.

on the Mel scale [2]. The Mel scale is used because it more closely resembles human sensitivity to sound than a linear scale.

The Hz to Mel conversion formula is given by:

$$m = 2595 \times \log_{10}(1 + f/700) \tag{14}$$

The filter-banks are created so that they are equally spaced along the Mel scale. The figure below displays the filters both on the Mel and Hz scale to highlight the difference.

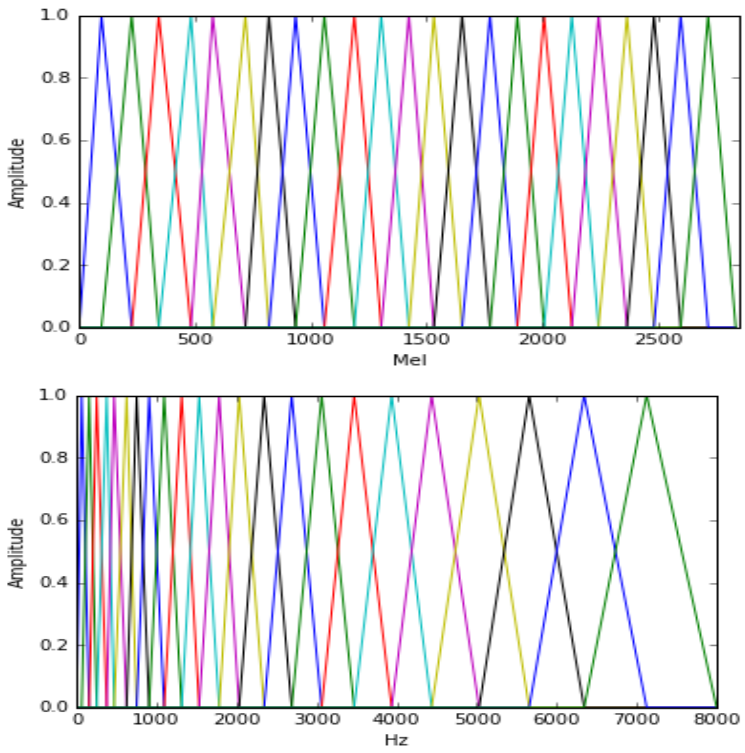

Figure 18: Upper: 23 filter-bank coefficients equally spaced on the Mel scale. Filters are half overlapping. Lower: The same filters on the Hz scale, no longer equally spaced

As you can see in figure 18, the Mel spaced filter-banks have a high resolution at low frequencies while at the higher frequencies, the filter-banks start to cover a larger range of the spectrum.

The filter-banks consist of weights, each filter has a weight for each of the FFT bins created in the previous steps. The filters weights are calculated on the Mel scale and then converted back to the frequency scale before applying them to the signal. Each filter is sensitive to its own frequency band and has

257 values, one for each of our FFT bins, of which most are zero. Applying the filters is simply a matter taking the dot product of each filter with the signal. This results in a single filter-bank energy per filter-bank. The resulting filter-bank energies are the Mel filter-bank features used as input to the CNN architecture described in [18].

8.8 Non-linear transform

We take the natural logarithm of the Mel-filtered signal because the human response to signal amplitude is roughly logarithmic [2]. As with the log energy feature calculated in section 5.3 the natural logarithm is used:

$$log\_fbank_i = ln(fbank_i) \tag{15}$$

Where i = 1,..., 23

Again the input values are clipped at a lower bound of 1e-22 on the off chance of taking the log of zero.

8.9 Discrete cosine transform

The next step in creating the MFCC features is to compute the cepstrum. This is done by applying the discrete cosine transform (DCT) to the log filter-bank features.

The ETSI protocol uses the type 2 DCT[4] which is given by:

$$MFCC_i = \sum_{n=0}^{N-1} log\_fbank_n \times \cos(\frac{\pi}{n}(n + \frac{1}{2})i) \tag{16}$$

$MFCC_i$ is the ith cepstral coefficient and N is the number of filter-banks created in section 5.7.

As described in [52] only the first 13 cepstral coefficients are kept. Furthermore, the first cepstral coefficient is replaced by the log energy computed in 5.3 [52].

8.10 Delta and double delta features

It is common in speech recognition to add so-called delta and double delta features to the MFCCs. The delta is the velocity and the double delta the acceleration. These are calculated for each of 12 MFCCs and the log energy feature and result in a feature vector of 39 features.

The delta features are calculated using the following formula:

$$d_{i,t} = \frac{\sum_{n=1}^{N} n(c_{i,t+n} - c_{i,t-n})}{2\sum_{n=1}^{N} n^2} \tag{17}$$

[4]DCT type 2 is often what is meant by the DCT. It is also the default DCT in many implementations such as those for Python and Matlab

Where $d_{i,t}$ is delta coefficient i at time t for i = 1,...,13 for coefficients $c_{i,t-n}$ through $c_{i,t+n}$. N is the parameter that determines how many frames of context are used in calculating the delta features, this parameter was set to two. The procedure for the double deltas is the same, except now the delta features are used instead of the MFCCs:

$$dd_{i,t} = \frac{\sum_{n=1}^{N} n(d_{i,t+n} - d_{i,t-n})}{2\sum_{n=1}^{N} n^2} \tag{18}$$

Looking back and ahead two frames will run into trouble at the edges of the signal. The signal is therefore padded with the first frame at the front and with the last frame at the back. The resulting MFCCs and delta and double delta features are the input features used as input to the baseline MLPs.

9 Appendix B

This appendix contains a more in depth analysis and discussion of the results of the CNN architecture extended with the Mel filter-bank layer (CNN Mf). Figure 19 shows the normalised confusion matrices for these networks. The scores on the diagonals indicate the classification performance for each class label, the scores off the diagonal indicate each label's confusion with other class labels.

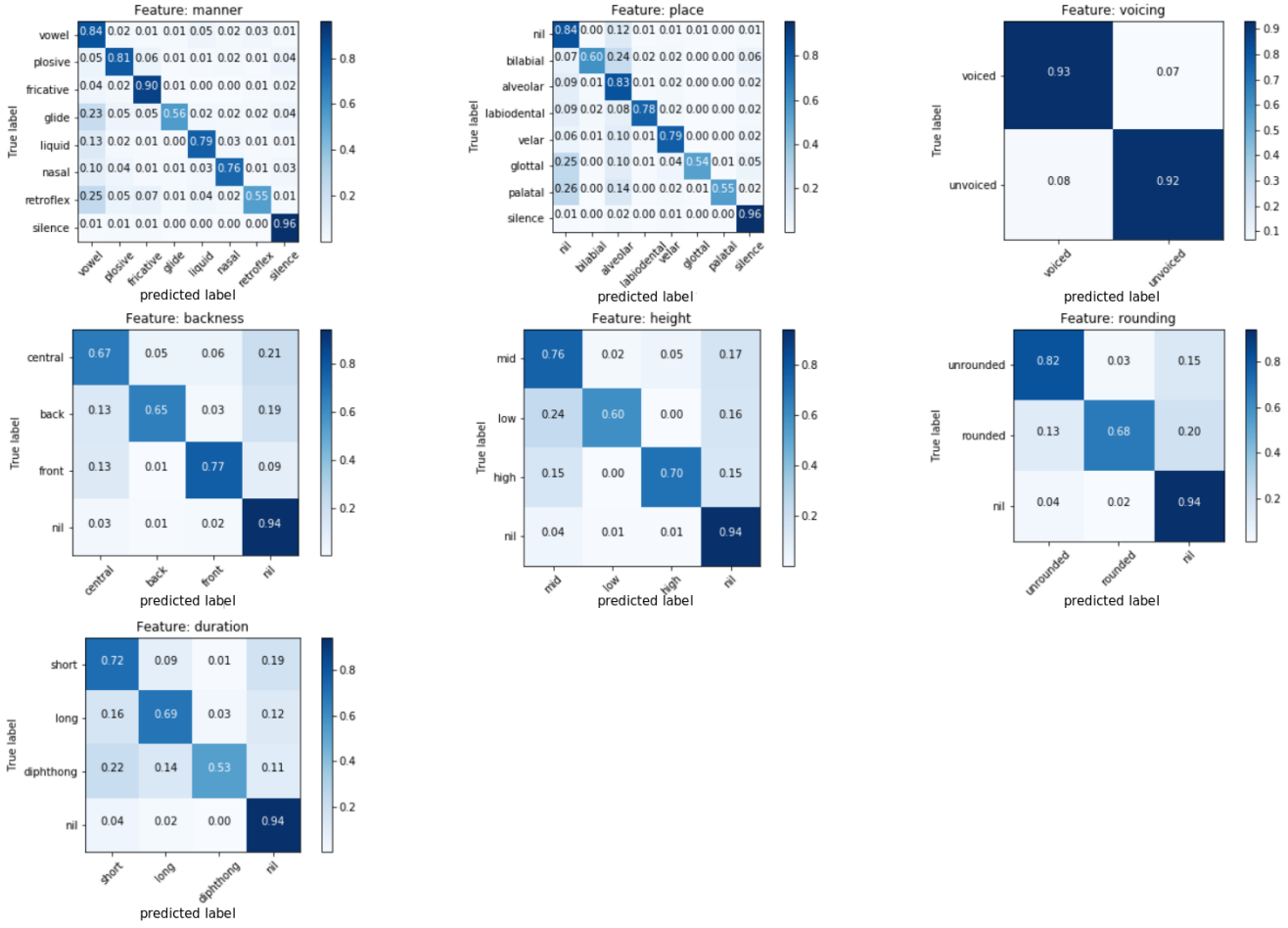

Figure 19: Normalised confusion matrices for the CNN Mf architecture. The diagonal corresponds to the per class accuracy.

While the extended networks did not outperform the basic architecture, they were not far behind with differences of only 0.6% to 2.6%. Furthermore, as the confusion matrices show these CNNs also reduced the bias towards the majority class compared to the MLPs. One possible explanation for the lower classification accuracy is that increasing the network depth can cause the network training to converge slower. The CNNs were trained for five epochs only based on pilot tests that showed that the CNNs made very little improvement beyond 4 epochs. However, this was not tested separately for the CNN Mf architecture and further investigation showed that these networks still made improvements between 0.5% to 1% in the fifth epoch. Training these networks beyond five epochs might (partly) close the gap between the CNN and CNN Mf architectures. Furthermore, the differences could simply be caused by the ran-

dom initialisation of the network weights. The differences are relatively small and it is possible that the performance of the CNN Mf architecture is closer to or higher than that of the CNN architecture under a different initialisation.

The idea beyond this architecture was that the Mel filter-banks are integrated into the network so that the filters can be optimised for the classification of the different AFs. The figure below shows the filter-banks that were learned, averaged for all classifiers.

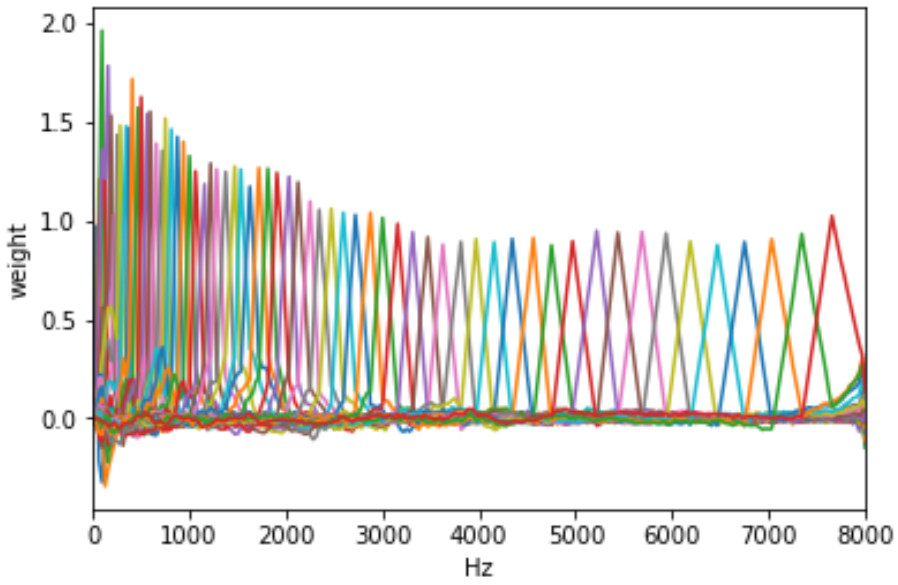

Figure 20: The learned filters averaged for all AFs.

All the filters were initialised to have on peak with a weight of one. As you can see the peaks increased most for the filters covering the low end of the amplitude spectrum increasing the relative importance of the low frequencies. This is in line with human sensitivity to sound and could indicate that the low end of the spectrum contains more valuable information about the speech signal.

However, the initialisation of the filters is already biased towards the low end of the frequency spectrum. By using Mel spaced filters, the filters have a better resolution at the low end than at the high end of the spectrum. Further testing with equally spaced filters could show whether the low end of the spectrum really contains more information that can be used for the classification of AFs or whether this is caused by a bias in the initialisation. In conclusion, frequency spectral filters are viable input features for AF classification, allowing the network to optimise the Mel filtering operation. While this approach did not outperform the basic CNN architecture, different hyper-parameter settings or more training epochs could improve the recognition results.