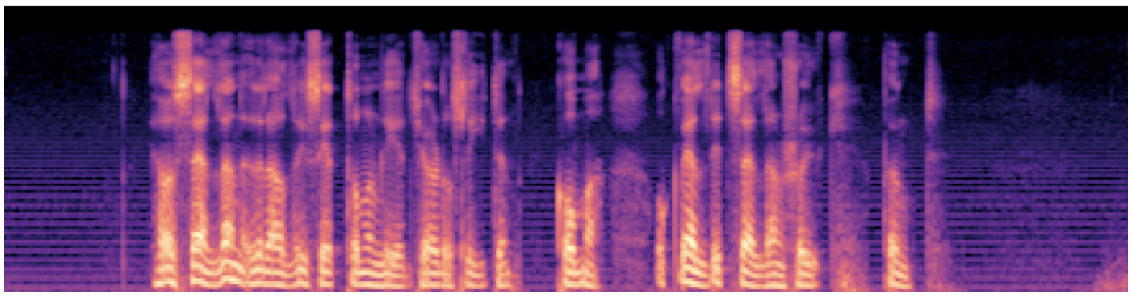

Transfer learning for domain specific automatic speech recognition in Swedish

An end-to-end approach using Mozilla's DeepSpeech

Master's thesis in Computer science and engineering

ALEXANDER HÅKANSSON
KEVIN HOOGENDIJK

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Transfer learning for domain specific automatic speech recognition in Swedish

An end-to-end approach using Mozilla's DeepSpeech

ALEXANDER HÅKANSSON
KEVIN HOOGENDIJK

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Transfer learning for domain specific automatic speech recognition in Swedish
An end-to-end approach using Mozilla's DeepSpeech
ALEXANDER HÅKANSSON
KEVIN HOOGENDIJK

Supervisor: Dana Dannélls, Språkbanken Text, Department of Swedish, University of Gothenburg
Advisor: David Fendrich, Tenfifty AB
Examiner: Carl-Johan Seger, Department of Computer Science and Engineering, Chalmers Univeristy of Techonology

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Mel-scaled spectogram of the sound from a person speaking Swedish.

Typeset in LaTeX
Gothenburg, Sweden 2020

Transfer learning for domain specific automatic speech recognition in Swedish
An end-to-end approach using Mozilla's DeepSpeech
ALEXANDER HÅKANSSON
KEVIN HOOGENDIJK
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Modern systems for automatic speech recognition (ASR), powered by artificial neural networks, make up the core in many day-to-day products and services. This development has been mostly driven by the large tech giants of the industry, which naturally has led to a large focus on developing models for the English language. If instead focusing on ASR for the Swedish language, there exists little research, and no open and available models. The lack of open and available models means that in order to create products and services relying on Swedish ASR, one needs to rely on third party commercial solutions. This becomes an issue when privacy, integrity, and cost are taken into account. Moreover, when considering specialised domains such as health care, creating a new model from scratch might not be feasible due to a lack of data.

In this thesis, the goal has been to explore how recent research in ASR can be applied to the Swedish language. In recent papers, transfer learning has been proposed as a technique to develop ASR models for languages where sufficient training data is lacking. In this thesis we aim to use the same technique to create a new state-of-the-art model for Swedish ASR, comparing against previous research on Swedish ASR as well as commercial solutions. Additionally, we explore if transfer learning can be successfully utilised to achieve even better ASR in specialised domains.

To achieve the aim of the thesis, the NST Acoustic Database for Swedish has been used to train a model based on Mozilla's DeepSpeech. Additionally, two domain specific datasets have been created as part of the thesis to explore if they can be used to fine-tune the general model for Swedish ASR in certain domains.

The resulting general model for Swedish ASR achieves a new state-of-the-art result on the test part of the NST Acoustic Database for Swedish, with a 13.80% word error rate and 4.78% character error rate. Additionally, we show that transfer learning can improve the results in specialised domain with on average 12% lower word error rate and 6% lower character error rate compared to the general Swedish model.

We conclude that recent research in ASR applies also to the Swedish language. We reaffirm that transfer learning is a powerful technique to create new ASR models based on existing ones, both for new languages and for specialised domains, with little extra effort in terms data and resources.

Keywords: automatic speech recognition, speech-to-text, transfer learning, artificial neural networks, end-to-end, DeepSpeech, specialised domains, Swedish.

Acknowledgements

First of all we would like to thank our supervisor Dana Dannélls for the invaluable guidance she has provided throughout the work with this thesis. We also want to thank our examiner Carl-Johan Seger for his contributions. Furthermore, we would like to thank Tenfifty for enabling this thesis, by providing us with an office space and for being an inspiration. Lastly, we would like to thank our friends and family for supporting us through this spring.

Alexander Håkansson, Gothenburg, June 2020

Kevin Hoogendijk, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1

Introduction

A huge amount of speech data is recorded every single day, e.g., in the form of news broadcasting or voice communication. To maximise exploitation and dissemination of this large amount of speech data, it is necessary to present it in written form. In short, if speech data could successfully be automatically transcribed into text, a lot of existing tools for text analysis could be utilised. Even if further analysis of the produced text is not desired, the transcription from speech to text is valuable in itself, in order to e.g., save time (i.e., allowing one to dictate ones thoughts instead of having to write them down), make the speech content searchable, or to simply make content more inclusive by making it available for people with hearing difficulties. This is where Automatic Speech Recognition (ASR), also known as *speech-to-text*, comes into play.

Tenfifty is a Gothenburg based company specialising in developing tailored AI solutions for their customers.[1] They have customers that are in need of automatic speech recognition for Swedish, but are constrained in terms of privacy, data integrity and cost. The lack of alternatives to existing commercial 3rd party services which can satisfy these constraints has become a problem, and progress can not be made. This thesis is a response to the challenge at hand, where possible solutions are researched.

As mentioned, there is a lack of open and ready-to-use automatic speech recognition models for the Swedish language. There are commercial services available, such as Google's *Cloud Speech-to-Text*,[2] but such models might not be an alternative for everyone due to e.g., cost or privacy constraints. The problem becomes even greater when considering automatic speech recognition in specific domains, e.g., flight radio communication or health care, where a special vocabulary or way of speaking might be used. In certain domains, such as the aforementioned health care domain, existing commercial solutions might be a particularly bad fit, since data might be very personal. An on-premises solution is preferred in these scenarios, as it preserves the integrity of the data. In order to achieve such an on-premises solution, one could naïvely create a new speech recognition model from scratch for every domain. However, such an approach would be a waste of time and resources, and might not be feasible due to limited data availability.

[1] `https://tenfifty.io`

[2] `https://cloud.google.com/speech-to-text/`

This presents a great opportunity to introduce transfer learning [1]. The idea is that utterances of speech can be similar between different applications, domains and even languages. This means that an English ASR model might also perform well on Swedish speech when additionally trained on Swedish data. Furthermore, this technique can be applied to specialised domains (e.g., health care, aviation and marine) to enable cheap, quick and well performing ASR even when data is sparse.

To the extent of our knowledge, there exist no previous research on using transfer learning to achieve automatic speech recognition for the Swedish language. Moreover, before this project there existed no openly available and ready to use model for Swedish ASR. There is work on using the Kaldi toolkit [2] for Swedish ASR [3, 4], but the resulting models have not been published. Kaldi is an open source system for speech recognition, but since its release there have been significant advances in the field of ASR [5–8]. Together with these advances, there has also been success in using transfer learning for language modelling [9], and in ASR for other languages with limited access to training data (e.g., German) [10–12]. In this thesis, we work with Swedish speech data. By utilising new and promising techniques in the field of ASR, we achieve similar improvements.

Based on recent research, such as the *wav2letter++* model by Facebook [13],[3] or the *DeepSpeech* [5] implementation by Mozilla,[4] we wanted to develop a new state-of-the-art model for Swedish ASR. Mozilla's DeepSpeech is implemented using TensorFlow,[5] an open source framework for machine learning, while wav2letter++ is implemented in C++ using the open-source machine learning library *flashlight*.[6] In a paper from 2017, an ASR model for the German language was created by first training a wav2letter++ model on English, and then used transfer learning to fine-tune the model on German [10]. Mozilla also recently demonstrated how their DeepSpeech model together with their multilingual dataset *Common Voice* could be used to create ASR models for languages with little training data available, by first pre-training on English [11].

In previous work on Swedish ASR, a public domain dataset called the *NST Acoustic Database for Swedish* was used [3].[7] The work by Kullmann [3] explored a couple of different variants of models, and achieved at best a word error rate (defined in Section 2.6) of 16% on the test part of the NST Acoustic Database for Swedish. As part of this thesis, we achieve an improved word error rate on the NST Acoustic Database for Swedish of 14%. Additionally, we evaluate Google's Cloud Speech-to-Text service on this dataset to conclude that the model developed as part of this thesis defines a new state-of-the-art result. Finally, we show how transfer learning can be used to get improved results in specialised domains.

[3] `https://github.com/facebookresearch/wav2letter`
[4] `https://github.com/mozilla/DeepSpeech`
[5] `https://www.tensorflow.org`
[6] `https://github.com/facebookresearch/flashlight`
[7] `https://www.nb.no/sprakbanken/show?serial=oai%3Anb.no%3Asbr-16&lang=en`

1.1 Aim of thesis

This thesis aims to address two problems:

1. can we create a state-of-the-art automatic speech recognition model for Swedish, following methods that have been proven successful for other languages, and

2. can we use transfer learning to achieve even better automatic speech recognition in certain specialised domains.

The models developed as part of the project should be able to, given some audio file of spoken Swedish, automatically transcribe the speech into text.

1.2 Rationale

In order to achieve the aim of the thesis, there were some problems which had to be approached and solved. These problems can be divided into two major milestones that each contribute to achieve the final goal of Swedish ASR using transfer learning. First of all, we created a general model for Swedish ASR. The general model would create a foundation for the following steps of the project, and utilised recent research on ASR. We will refer to this model throughout the thesis as the *general Swedish model*.

After creating the general Swedish model, which addressed our first aim, we explored whether we could use this model as a foundation for transfer learning. This was used to reach the second aim of the thesis, which was to explore if we could get an improved ASR system on a specific domain using transfer learning.

1.2.1 General Swedish model

Creating the general Swedish model, i.e., a general ASR model for Swedish, was the most crucial problem to solve. Our goal was to improve on previous work in the area of Swedish ASR, so we had to achieve a lower word error rate (defined in Section 2.6.2) on the NST Acoustic database for Swedish compared to Kullmann [3], i.e., lower than 15.97%. Preferably, we wanted to see some significant and non trivial improvement compared to Kullmann [3]. Additionally, we wanted to compare the general Swedish model to commercial services, namely Google's Cloud Speech-to-Text service.

1.2.2 Domain specific model

In order to achieve the second aim of the thesis project, we had to explore approaches for creating domain specific ASR models. These models had to utilise transfer learning, and be based on the general Swedish model. The goal was to achieve an improvement over the general Swedish model when used on the specialised domain data. In order for the results to be meaningful, we decided that the domain specific data had to be much smaller in size compared to the data used to train the general

Swedish model. In previous research, transfer learning was successfully used to create an ASR model for Slovenian with just five hours of data [11]. Because of this, we decided to limit how large the domain specific datasets should be, such that they consisted of at most roughly 5 hours of speech, which is about 1% of the NST Acoustic database for Swedish.

1.3 Scope

This thesis project was limited to 20 weeks, which also limited the scope of what could be achieved. Natural language processing in general, and automatic speech recognition in particular, are very active fields of research. There is new theory released all the time, but due to the limitation in time, we focused on using techniques that have proven successful. It was also a top priority that implementations either existed and were available, or were trivial to implement, as time was limited.

For automatic speech recognition, there are primarily two different open-source models that fit the project: Mozilla's DeepSpeech (see Section 2.5.2) and wav2letter++. Both models allow the usage of transfer learning, and optimally we would have liked to use and compare both models; however, this was not feasible in the given time frame. In this project, we thus decided to use Mozilla's DeepSpeech model as a foundation for the model development. The decision to go with Mozilla's DeepSpeech was made after researching both models and comparing pros and cons of them, as well as after discussions with our supervisor and advisor. Our familiarity with TensorFlow and the Python ecosystem (which Mozilla's DeepSpeech is based on) contributed a lot to the decision. Mozilla's DeepSpeech also seems to have a larger community around it in general, which includes several hands-on examples on using it for transfer learning. A con with wav2letter++ is also that it requires a lexicon with all words that it should support, in addition to a language model. This added unwanted complexity compared to Mozilla's DeepSpeech.

As described further in Section 2.5, a typical system for automatic speech recognition involves using a language model. Mozilla's DeepSpeech is no exception, and has out-of-the-box support for the KenLM language model [14]. In theory though, support for other language models could be implemented to work with Mozilla's DeepSpeech. It would have been interesting to try out different language models, such as BERT [9]. In the work by Shin et al. [15], BERT is successfully used as a language model in the context of ASR. Due to the limited time however, we deemed it infeasible to focus our efforts on implementing support for using BERT together with Mozilla's DeepSpeech — this was especially the case considering there is out-of-the-box support for KenLM.

1.4 Limitations

When training models as part of this thesis project, we were limited to a CUDA enabled GTX 1080 Ti graphics card with 11 GB of memory. The graphics card was available for use throughout the entire project. Additionally, we had a budget from

Tenfifty for use on Google Cloud computing resources, which was used to evaluate Google's Cloud Text-to-Speech service.

Since the NST Acoustic Database for Swedish is publicly available, the general Swedish model developed using this data can be and is published.[8] However, due to licensing issues, the domain specific datasets created as part of this thesis project can not be published. Instead we must refer anyone interested to download and process the source data themselves, following our method. Because of this, the models based on these domain specific datasets are also not published. However, we argue that the biggest value is obtained from having open access to the general Swedish model though, as this enables further transfer learning for potential gains in performance, and other use cases.

1.5 Ethical considerations

We have identified two potential ethical problems that follow from developing an efficient ASR system for Swedish. However, there are also gains to be made.

First of all, if an efficient enough system is developed, it might result in jobs being automated and replaced. Such jobs could be e.g., secretaries. This also leads to the second problem, mainly the problem of responsibility. In the event of an incorrect speech transcription by an ASR system, who is responsible? This problem might be negligible though, as it is for example rarely the secretary that has the ultimate responsibility. As an example, in a health care setting, it is most likely the medical doctor that has the ultimate responsibility for what is put into e.g., a patient's journal. This means that while utilising ASR systems, it is important to not stop proofreading.

As mentioned, there are also gains to be made. Instead of having to rely on third party services, such as Google's Cloud Speech-to-Text, one could create their own solutions. This increases data integrity, and strengthens the integrity and privacy of people whose speech need to be transcribed.

1.6 Contributions

As part of this thesis work, a new state-of-the-art result is achieved on the NST Acoustic database for Swedish, improving over previous research [3] by an absolute 2%, and over Google's Cloud Speech-to-Text by an absolute 7%.

It is also shown that ASR models, such as the one developed as part of this thesis, can with little effort be fine-tuned to achieve additional performance on specialised domains. This, together with the method for achieving a new state-of-the-art result on the NST Acoustic database for Swedish, confirms that transfer learning is a powerful and efficient method in the field of Swedish ASR. Additionally, the model developed is freely available for further research and use.

[8] `https://github.com/se-asr/`

2

Background

In this chapter, theory required to understand the thesis is presented. A background in computer science, with knowledge of machine learning as a concept, is assumed.

2.1 Artificial neural networks

An artificial neural network (ANN) is a graph based type of machine learning model, and is sometimes referred to as a deep learning model [16]. An ANN model consists of nodes, which are commonly grouped together in so called layers. Together the layers form a directed acyclic graph (DAG), starting with an input layer, passing through an arbitrary amount of so called hidden layers, which finally connect to an output layer. Figure 2.1 illustrates a simple ANN, with an input layer, a single hidden layer, and an output layer. When using an ANN for inference, input values are fed to the input layer, which then gets propagated through the network until they reach the output layer to produce an output.

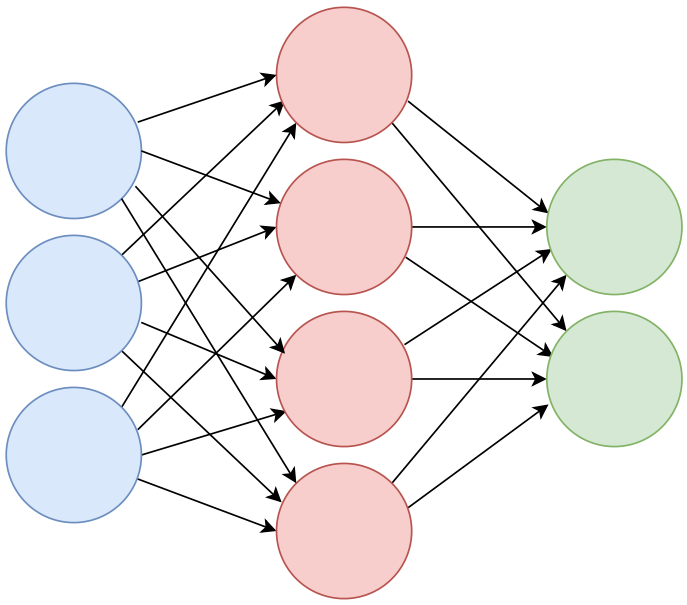

Figure 2.1: An illustration of a simple artificial neural network. Each layer is indicated by a unique color. The blue layer is the input layer, the red layer is a hidden layer, the green layer is the output layer.

There are different types of artificial neural networks [16]. In the most simplest type, feed-forward neural networks, every node in one layer is connected to every other node in the next layer. The ANN in Figure 2.1 is an example of a feed-forward neural network. Common to all types of ANN however is that each of these connections have a weight value associated with them, where the weight determines the impact of values passing through it. During the training of an ANN, the aim is to optimise these weights, as they directly affect the output of the model.

Looking at an individual node in the graph, its input is a weighted sum of the output from preceding nodes, where each node pair has a unique weight value. The weighted sum is then transformed using a so called activation function, before being sent as input to subsequent nodes. A common activation function is the rectified linear unit (ReLU), defined in Equation 2.1.

$$f(x) = max(0, x) \tag{2.1}$$

The activation function used on the nodes in the output layer during inference can differ a lot, depending on the use case for the ANN model. It is also common to simply not use any activation function at all on the output layer's nodes, letting the model output raw values directly. However, when using an ANN model for a classification task with multiple possible classes, the softmax activation function is commonly used. The softmax function is defined in Equation 2.2, where $\vec{x}$ is the vector of all node outputs, and $\vec{x}_i$ is the output of the i-th node. The softmax function has the effect that all outputs are normalised such that they are in the range $[0, 1]$, and such that $\sum_{i=1}^{n} \vec{x}_i = 1$, forming a probability space.

$$f(\vec{x}_i) = \frac{e^{\vec{x}_i}}{\sum_{j=1}^{n} e^{\vec{x}_j}} \tag{2.2}$$

2.1.1 Training

When training (or optimising) an ANN model, a so called loss function is applied to the output of the model, instead of an activation function. The purpose of a loss function is to calculate a loss, also referred to as a cost, which is a measure of how correct the output of the model was compared to some ground truth. The loss is used by an optimiser based on stochastic gradient descent to update the weights of the ANN in order to hopefully improve its future performance. There are different types of optimisers, but the Adam optimiser [17] is by far one of the most widely used ones today. As with activation functions, there are multiple different loss functions to choose from, depending on the purpose of the model. However, a loss function of particular interest in this thesis is the Connectionist Temporal Classification (CTC) loss function [18]. The CTC loss function is of particular use when working with sequential data, such as audio or text, as it can take a sequence of inputs to produce a sequence of classifications.

In general, when training an ANN model, it is fed training data. Each data point in a training dataset can either be sent one at a time through the network, optimising the network after each one, or it can be sent in batches, using so called *mini-batching* [19, 20]. Going through all data points in a training dataset is referred to as an *epoch*. An ANN model can be trained through multiple epochs, meaning it processes the training dataset multiple times. As the ANN model is trained on a training dataset, it gets better and better at making correct predictions for it. To make sure that the model is actually generalising, and not just getting good at the examples in the training dataset, a separate so called development dataset is commonly used. The phenomena where a model stops generalising and gets specialised on a specific dataset is referred to as overfitting, and is something you generally want to avoid.

One technique to avoid overfitting is referred to as *early stopping* [21]. When using early stopping, the average loss for the development set is calculated after each epoch, but the weights of the network are not updated. As the average loss for the development set stops improving over multiple epochs, overfitting has happened. When this happens, we can stop training of the model, and the model weights where the loss on the development set was the lowest is used.

Another technique to avoid overfitting, which can be used in conjunction with early stopping, is so called *dropout* [22]. When training a model with dropout, for every node there is a probability p that the node is *dropped*, meaning it will not contribute to the network for that particular training instance. The intuition behind dropout is that it reduces the risk of the network becoming reliant on individual nodes. The parameter p is typically fixed for each layer in the network, and is commonly referred to as the dropout rate. During inference, dropout is disabled.

2.1.2 Recurrent neural networks

As mentioned, there are different types of artificial neural networks, and the recurrent neural network (RNN) is an additional type of ANN which is common in the context of natural language processing (NLP). One particular feature of recurrent neural networks which makes them interesting is that they are able to capture context, in the sense that they take into account the input that has come before the current one. When using an RNN for e.g., text classification, this means that when the model is fed a word that is part of a sentence, it will be able to take into account preceding words in that sentence as well.

A core component of an RNN is the so called *unit*, and is what enables the context awareness. There are multiple types of RNN units, but a common characteristic is that they have two inputs, namely the current input, and the previous input [23, 24]. A commonly used unit is the Long short-term memory (LSTM) [23] unit. The LSTM unit consists of weighted gates, which regulate e.g., how much of the previous input should be taken into account.

2.2 Transfer learning

Transfer learning, as the name suggests, is the process of transferring knowledge from one machine learning model to another, which may or may not be used for the same task as the original model [1]. *Transfer learning* is in itself not referring to any specific technique for achieving such a knowledge transfer, and is more of an umbrella term. In this project, a specific technique for transfer learning called *fine-tuning* is the primary focus. Fine-tuning has been used very successfully in the field of image classification for a long time, primarily based on ImageNet [25, 26], and has recently shown promise in the field of ASR [10, 11].

Fine-tuning, or sometimes called model adaptation, is a transfer learning technique where the learned parameters from a machine learning model are re-used in a new model [1, 10, 27, 28]. When re-using the parameters, they can either be frozen such that they are not updated during training, or they can be adjustable such that they can be updated during training. When doing fine-tuning on artificial neural networks, the weights of the nodes in layers are transferred from a source model to the new model.

The use of fine-tuning is often motivated by a lack of training data for a specific task [10, 11, 28]. By using fine-tuning, one hopes that a model trained on a similar task can be used as a base for a new model, and as such the new model can leverage knowledge that might overlap between the two tasks [1]. In Kunze et al. [10] the assumption is that English is similar to German, so by using a model pre-trained on English and fine-tuning it on German, better results can be achieved compared to training a model on German from scratch, since the available amount of training data for German is scarce.

As mentioned, when re-using weights, they can either be frozen or adjustable. Often the choice of freezing weights or not is made on a per-layer basis [10, 11]. Besides simply freezing layers, one or more layers can also be removed completely. The last layer can be removed when e.g., the model's predicted labels are not desired for a new task [1]. In the context of ASR, a model for the English language might predict letters ranging from a–z, but when training a model for Swedish it would most likely be desirable to also include the letters *å*, *ä*, and *ö*. To achieve this, the last layer in the model for English can be replaced with a new layer that has room for the extra letters.

2.3 Sound

Sound is an important part of any system working with speech. The way sound is represented and processed can impact the performance of systems as well as enable certain kinds of analysis [29]. Sound is created by vibrations that are transferred through different media, such as air or water. The vibrations are often described as a signal of oscillating waves. The amplitude of the sound wave corresponds to the power of the sound, i.e., how loud it is. The frequency of the sound wave corresponds

to the, so called, pitch of the sound. Pitch is the quality of sound that makes it possible to judge sound as *higher* or *lower*. For example, a sub-woofer commonly outputs sound that is low pitched while a fire alarm often outputs sound that is high pitched.

Most sound is composed of several frequencies with different amplitudes. These frequencies and their individual amplitudes can be separated and distinguished through a method called Discrete Fourier Transform (DFT). Given any sound wave, a DFT can be used to approximate it using a number of sinus waves with different frequencies and amplitudes. By doing this the sound wave can be represented by a combination of components which can be analysed individually or as a whole. Figure 2.2 illustrates how a sound wave can be approximated with several sinus waves at different frequencies and amplitudes. In the figure, the waves are projected on the frequency-axis to show the amplitude of each frequency. The output is often represented by a vector, of which the length depends on the sampling rate of the DFT. The output in the figure could, depending on the sampling rate, be represented by the vector $X = [0, 4, 0, 2, 0, 2, 0, 1, 0, 1, 0]^T$. If the sampling rate halved, the output vector would instead be $X = [4, 2, 2, 1, 1]^T$. Each value in the vector is the amplitude of a sinus wave with a specific frequency, given by the index of the vector. The 4 could, for example, be the amplitude of a 1 kHz sinus wave, the 2 could then be the amplitude of a 2 kHz sinus wave, and so on.

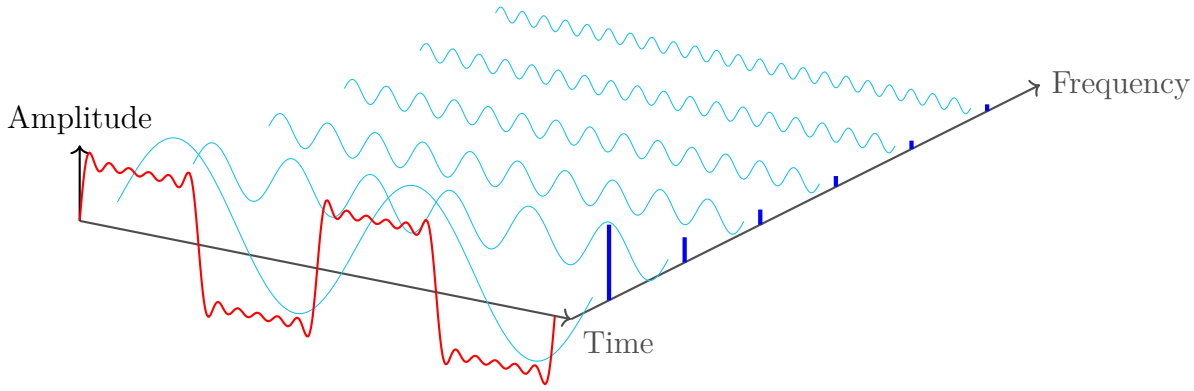

Figure 2.2: A visualisation of a DFT on a sound wave. The sound wave is shown in red. The different sinus waves contributing to the sound wave are shown in cyan. The blue bars are the result of the DFT and indicate the amplitude of the sinus wave for each frequency.

The following subsections further describe different parts of sound analysis and representations that are commonly used in the context of ASR.

2.3.1 Mel scale

The Mel scale was devised to measure the magnitude of difference in perceived pitch [30]. It describes how the perception of distance between two pitches differs at different frequencies. Figure 2.3 illustrates the characteristic signature of the Mel scale for frequencies between 0 and 8000 Hz. As can be seen in the figure, after the

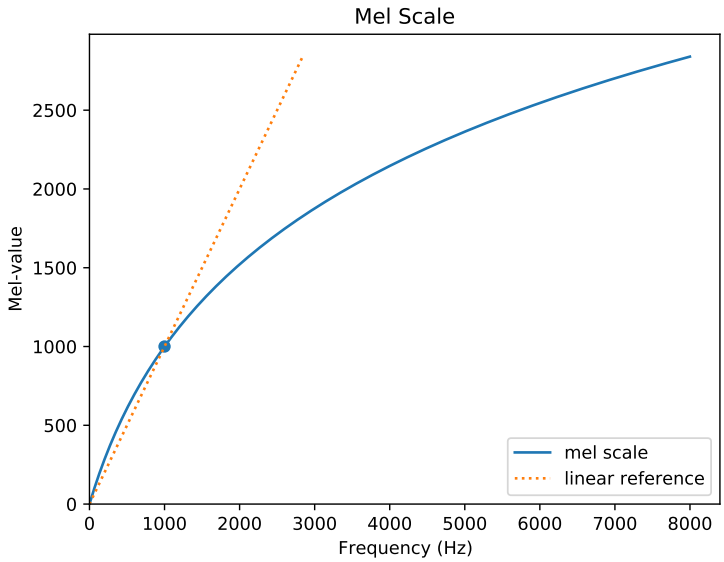

Figure 2.3: Mel values compared to their corresponding frequencies. The dot illustrates the break point at which the perception of pitch difference is lower than the actual change in frequency.

breaking point, 1000 Hz, the perceived difference in pitch becomes lower than the actual difference in frequency. It decreases to the point where a 100 Hz difference between two frequencies is indistinguishable. For example, the frequency difference between 100 and 200 Hz is equal to the difference between 10 100 and 10 200 Hz; however, as the Mel scale suggests, the way humans perceive these sound waves is very different. To our ears the difference in pitch between 100 and 200 Hz is very obvious, while the difference between 10 100 and 10 200 Hz is indistinguishable. So while the differences in frequency are identical they are perceived differently depending on the magnitude of the frequency.

The Mel scale is not an exact correlation, but an approximation, and was initially formulated empirically through quantitative experiments. Although not precise, the Mel value for a frequency f is commonly calculated as shown in Equation 2.3, where f denotes the frequency in Hz. The inverse of this equation can be used to get the frequency given a Mel value.

$$\text{Mel-scale}(f) = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right) \tag{2.3}$$

2.3.2 Mel-Frequency Cepstral Coefficients

Mel-Frequency Cepstral Coefficients (MFCCs) are commonly used as a representation of sound for speech recognition systems [6, 11, 31–33]. MFCCs combine the property of the Mel-scale that provide human-like perception of sound, together with a prioritisation of the sound properties that matter most for speech. MFCCs also compress the sound, which means that the MFCC representations of sound takes

up less storage space while capturing sufficient information for machine learning purposes.

Sound waves can be transformed into MFCCs through a number of steps, of which some might change in order depending on actual implementation.

The first step in this process is the DFT. To be able to apply the DFT, the sound wave needs to be consistent in the time frame that is analysed — i.e., not changing too much. Figure 2.2 is a good example of a consistent signal. Speech is typically not consistent but if the sound is split into small enough parts, each part can be considered consistent. These parts are called *windows* and are typically about 20 ms long. The DFT is applied to the windows individually and the result for each window is a vector of amplitudes for each of the frequencies that contribute to the the sound of that window.

The next step is to apply the Mel scale, which maps frequencies to Mel scaled values. Additionally, a logarithmic trade-off for the amplitude is implemented. Similarly to how the Mel scale models the perception of pitch (frequency), the perception of loudness (amplitude) is often described with a logarithmic model [34]. Without applying the log function a change in amplitude in the lower frequency regions would have a higher impact than the same change in the higher regions, which does not match how humans perceive sound.

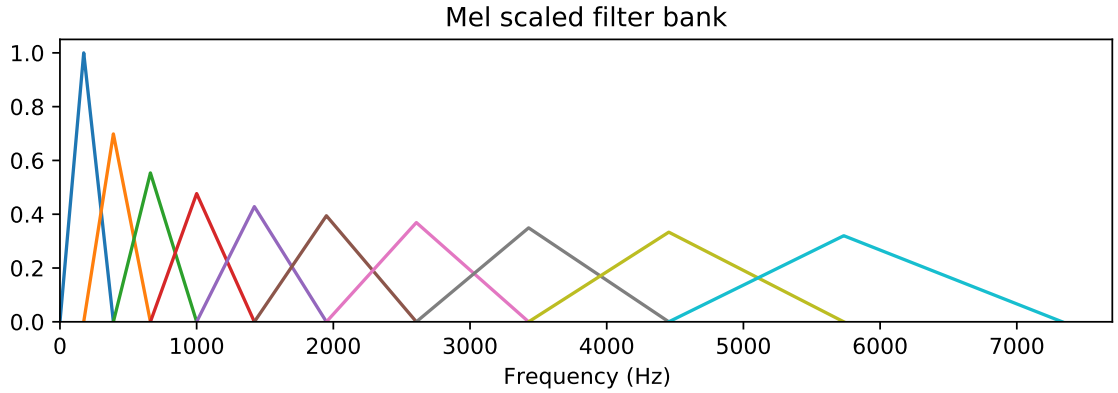

Figure 2.4: A Mel-scaled set of filters with a linear spacing of 250 Mel. Each color represents one filter.

The application of the Mel scale and log function is often combined with a compression. These three function to ensure that the features of the sound that are most important are extracted. Compression is accomplished through sampling, by extracting a selection of data points from a dataset. Sampling single points would, in this case, result in very misleading features. Instead, sample filters are used to sum the amplitudes in an area around the sample point to capture the characteristics of the sound at that part of the frequency spectrum. These filters are most commonly triangular filters centered around a sample point, with linearly decreasing activation in each direction. However, other filter types can be used as well, depending on implementation. As illustrated in Figure 2.4, the filters are spaced

at proportional steps according to the Mel scale and logarithmically decreasing in weight. This means that, if they are plotted on a frequency axis, as in the figure, there are more sample points in the lower frequency area and fewer sample points in the higher frequency area. If plotted on the Mel scale, however, the difference between each filter would be constant. The triangular filters are commonly compiled as shown in Equation 2.4 [35] where f_k is frequency sample points distributed at equal distances according to the Mel scale and h is the frequency.

$$w_{k,h} = \begin{cases} \frac{h-f_{k-1}}{f_k-f_{k-1}} & \textbf{if } f_{k-1} < h < f_k \\ \frac{f_{k+1}-h}{f_{k+1}-f_k} & \textbf{if } f_k < h < f_{k+1} \\ 0 & \textbf{otherwise} \end{cases} \tag{2.4}$$

Finally a Discrete Cosine Transform (DCT) is applied to the output of the filters. DCTs are similar to DFTs, except they use the cosine function instead of the sinus function.

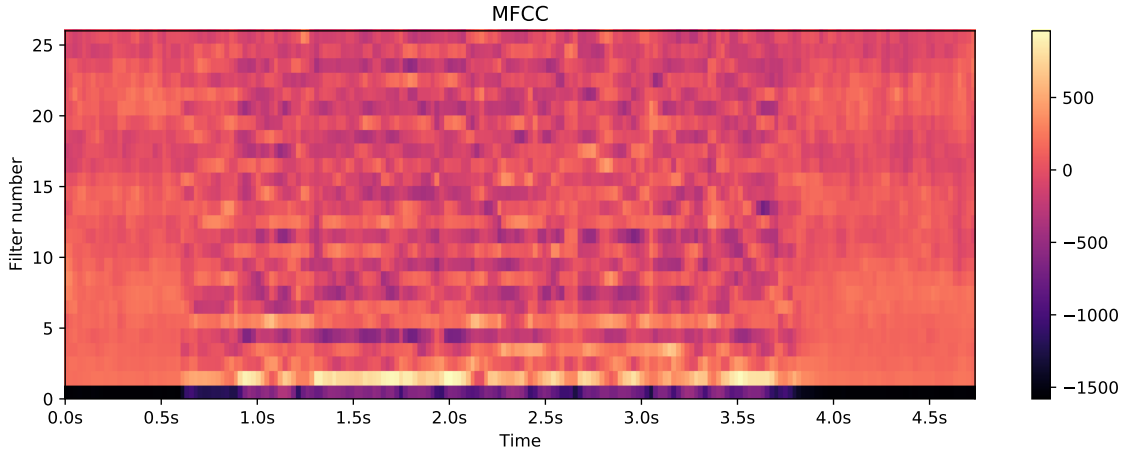

Figure 2.5: MFCC representation of a short audio file from the NST Dataset.

The final result of putting a window through all of these transformations is a vector with the same number of values as the number of filters. The values correspond to the activation of each filter. If we have 4 filters, each vector will have 4 values. To represent a piece of sound, the vectors for all windows are vertically concatenated with each other to form a matrix, like the one in Figure 2.5. The figure shows sound represented as MFCCs, where the height is equal to the number of filters and the width is equal to the number of samples from the audio. If the audio file is 400ms long and the window size is 20ms the width will be 20.

2.4 Language modeling

A language model (LM) is a model of a natural language, such as Swedish or English. In this work we focus on statistical language models that are used by digital systems to modify texts, such that they adhere to the language. Given, for example,

a sequence of words, a language model can predict the next word in the sequence based on the language that it represents. A statistical language model has no notion of grammar, instead it relies on the number of occurrences of words in the language [36]. As an example, the word sequence *a banana is very* is, intuitively, more often followed by the word *sweet* than the word *toxic*. Language models can be useful in many situations related to natural language processing. For example, when correcting words in a sentence or scoring the likelihood of whole sentences and deciding which one of two (or more) that is the most probable one.

It can be difficult to distinguish between someone saying *this guy* and *the sky*, since the pronunciation can sound very similar for some people. Depending on the surrounding words, a language model can make a qualified prediction for one or the other. Previous work has been able to significantly improve automatic speech recognition results by using a language model [10, 31, 32, 37, 38].

A common way of modeling languages is with n-grams [14, 39]. An n-gram is in general a sequence of n items, and in the context of this thesis a sequence of n words. Given a sequence of words one could extract all possible n-grams from it and count the number of occurrences for each n-gram. For example, extracting all 3-grams (trigrams) from the sentence *The fox and the dog* would result in the trigrams *The fox and*, *fox and the*, and *and the dog*.

One toolkit for compiling n-gram language models is KenLM [14]. It has successfully been used in several automatic speech recognition systems [5, 10, 13]. KenLM was designed to perform faster and use less memory than popular models at the time [14]. When it was published it outperformed previous models, such as the SRILM [39] and IRSTLM [40], in these aspects [14]. Both SRILM and IRSTLM are implementations of what was at the time state-of-the-art algorithms.

When compiling a KenLM model, a parameter called *order* has to be specified. The order describes what kind of n-gram models it consists of. Since it is difficult to find all possible sequences of words for a specific order, statistical language models often approximate probabilities using lower order n-gram models [36]. A 3-gram could be approximated using a 2-gram and a 1-gram model. Using this approach, it is theoretically possible to give a probability for a specific 3-gram even though the exact sequence has never been seen during training of the language model.

The resulting language model will contain n-grams from n = 1 to n = order. Throughout this thesis, we will describe these models as {order}-gram models (e.g., 3-gram) but what it really means is a combined model with all n-grams up to that order. For example, what we call a 3-gram LM is not a model with only 3-grams but instead a combination of 1-gram, 2-gram and 3-gram models.

2.5 Automatic speech recognition

Automatic speech recognition, or ASR for short, is a technology that enables human–computer interaction through speech, and is an active field of research that has been

studied for several decades [33]. A more everyday term for ASR is simply *speech-to-text*, which most people have likely heard of at some point. In recent years, ASR systems have become a core component in consumer products like digital assistants such as *Google Assistant* or *Siri*, voice search such as *Google Voice Search*, and in-car infotainment systems. ASR systems can also aid in human–human interaction, by e.g., transcribing lectures to make content indexed and searchable by students.

2.5.1 General architecture

A typical ASR system includes four major components, as shown in Figure 2.6 [33]. The signal processing and feature extraction component is responsible for transforming an audio signal into a format suitable for the acoustic model (AM). This includes extracting features from the audio signal that might be of interest for the AM. The AM then combines its knowledge about speech (i.e., acoustics and phonetics) with the features from the previous step, and produces a score for the features. The language model, as described in Section 2.4, estimates a probability, or score, of a sequence of words. Finally, the hypothesis search component combines the AM score and the LM score to produce an hypothesis for what was spoken in the input audio signal.

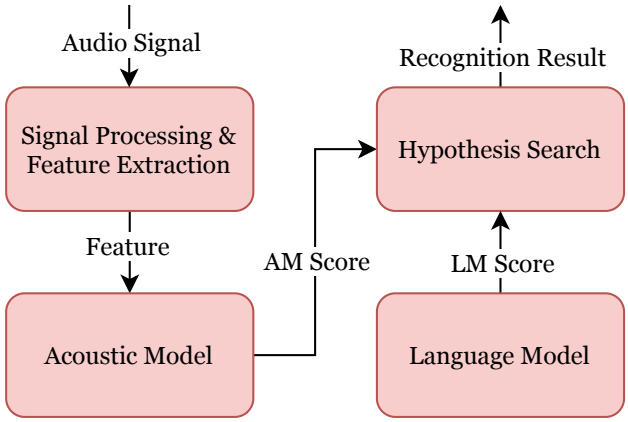

Figure 2.6: Architecture of a typical automatic speech recognition system.

The details of how these four components should be implemented are not specified, and recently so called *end-to-end* systems which combines multiple of these components have emerged. As an example, both DeepSpeech [5] and wav2letter [6] are two end-to-end models, based on artificial neural networks, combining the signal processing and feature extraction, acoustic model, and hypothesis search, while making the language model optional. An advantage with the end-to-end approach to ASR is that it eliminates the need for linguistic resources and expertise [5, 6, 11, 13, 41]. It has also been shown that this approach outperforms more conventional ASR systems based on hidden Markov models [5, 7, 8, 10, 37, 42]. An example of a model that is not end-to-end, and is primarily based on hidden Markov models, is the Kaldi toolkit [2], which previously defined state-of-the-art for ASR [43], and has been used to develop ASR systems for Swedish and German [3, 44, 45].

2.5.2 Mozilla's DeepSpeech

Mozilla's DeepSpeech is an open-source implementation of the DeepSpeech model presented by Hannun et al. [5],[1] which is a model for end-to-end ASR. As Mozilla's DeepSpeech is open-source, and is continually improved, the model is not identical to the one by Hannun et al. [5]. In this thesis, the released version `v0.6.1` of Mozilla's DeepSpeech is used, and a simplified version of its model architecture is shown in Figure 2.7.

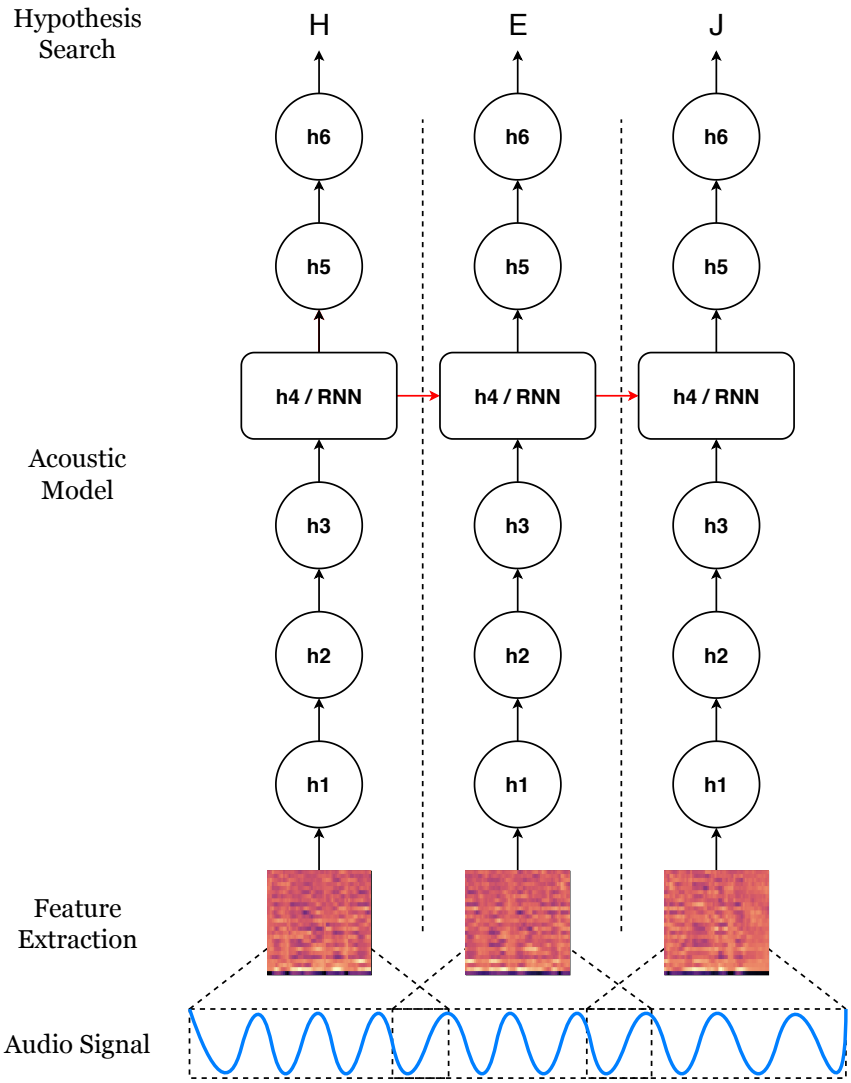

Figure 2.7: Simplified architecture of Mozilla's DeepSpeech model. Each circle represents a layer of several nodes.

As seen in Figure 2.7, Mozilla's DeepSpeech model is and end-to-end model that contains the three most important components of a typical ASR system: The signal processing and feature extraction, acoustic model, and hypothesis search components. The model is an artificial neural network which in total consists of six layers.

[1] `https://github.com/mozilla/DeepSpeech`

It takes a raw audio signal as input, and outputs a text transcription for the speech in the audio.

The model splits the raw audio signal into slightly overlapping windows of 32 milliseconds each, and extracts Mel-Frequency Cepstral Coefficients (MFCCs, described in Section 2.3.2). Each of these windows represents a so called *time step*, of which each serves as input to the artificial neural network. The input is first fed through three feed-forward fully connected layers, with 2048 nodes each. Each of these three layers use the ReLU [46] activation function. The fourth layer is a uni-directional RNN layer using 2048 LSTM cells [23]. The fifth layer is another feed-forward fully connected layer, with 2048 nodes using the ReLU activation function. The sixth and last layer is the output layer, where the number of nodes is equal to the size of the alphabet of the target language. As an example, if the target language is e.g., English, the alphabet would consist of the characters a–z, making up a total of 26 characters. The output layer would then consist of 26 nodes, where each node represents one of the characters. The output layer outputs so called *logits* for each represented character. A logit is a value in the range $[-\infty, \infty]$, and works as a raw representation of the probability that a specific character is uttered, where a higher value means the probability is higher. In statistics, a logit is typically calculated as shown in Equation 2.5, where p is the probability.

$$logit = log\left(\frac{p}{1-p}\right) \tag{2.5}$$

When using Mozilla's DeepSpeech model for inference, the softmax function is applied to each of the n logits from the output layer. The softmax function is defined in Equation 2.2, where $\vec{x}$ is the vector of all logits, and $\vec{x}_i$ is the logit for the i-th character in the alphabet. The softmax function forms a probability space, where each character gets a probability in the range $[0, 1]$, and the sum of the probabilities for all characters is 1. The character with the highest probability will be predicted by the model. Applying this to all time steps, the result will be a sequence of predicted characters, which when combined makes up the predicted transcript for the input sound signal.

The model predicts transcripts based on the probabilities of characters, and has no understanding of the actual language itself, it only predicts characters based on sounds. This means that the resulting transcript can contain spelling mistakes and sequences of words that does not make any sense when put together. To counteract this, Mozilla's DeepSpeech supports using the KenLM [14] language model to correct the transcription. In short, it works by feeding the output transcription from Mozilla's DeepSpeech into KenLM, which then outputs a finalised and corrected transcript. The details of how language models work are described in Section 2.4.

For the training of Mozilla's DeepSpeech model, the Connectionist Temporal Classification (CTC) loss function [18] is used together with the Adam optimizer [17]. The choice of using the Adam optimizer is an example of where Mozilla's DeepSpeech differs from the work by Hannun et al. [5]. In Hannun et al. [5] an optimizer

based on Nesterov's Accelerated Gradient Descent (AGD) [47] was used, rather than the Adam optimizer used in Mozilla's DeepSpeech. Mozilla's DeepSpeech utilises mini-batching [19, 20] and dropout [22], both of which are easily configurable when training a new model.

Table 2.1: Hyperparameters used to pre-train an English model using Mozilla's DeepSpeech.

Parameter	Value
Mini-batch Size	128
Learning Rate	0.0001
Dropout	0.2

Together with its implementation, Mozilla has also released a pre-trained DeepSpeech model for the English language, free of use. The pre-trained model is trained on approximately 5500 hours of transcribed English speech, and achieves a 7.5% WER on the LibriSpeech [48] test dataset. When training this model, the hyperparameters shown in Table 2.1 were used. In [31], a DeepSpeech model for German was created using 302 hours of transcribed German speech, and in the original DeepSpeech paper [5], 7380 hours of transcribed English speech is used.

2.6 Evaluation

When developing any sort of model, it is crucial to be able to evaluate how good it is in a systematic and reproducible manner. In the evaluation of models for ASR, the word error rate (defined in Section 2.6.2) and character error rate (defined in Section 2.6.3) are two commonly used measurements, where an as low error rate as possible is desired [3, 10, 11]. The word error rate and character error rate are defined below, but in short they both compare a predicted output text from a model, given some input sound, with the actual transcription of that sound, and calculate how much they differ from each other.

By using the same metrics for evaluation, it is possible to compare two different works against each other. Metric results are comparable if they are calculated for the same dataset (a so called test set), i.e., the output text of one model is compared to the same reference text as the other model was tested on. In fact, this is how most research is done in the field, and realistically, it is not meaningful to compare evaluation results with results reported by others unless all results are based on the same dataset.

When looking at tasks other than pure transcription of speech, such as real world applications for e.g., speech driven navigation, relying on metrics like word error rate for evaluation might not be enough [49]. In real world context, it is important to involve and combine human evaluations to evaluate naturalness and usability. Such evaluations are typically costly, as human time is expensive, and are usually performed by letting a human use and rate a system.

2.6.1 Levenshtein distance

The *Levenshtein distance*, sometimes referred to as *edit distance*, is measurement for comparing the similarity between two sequences of characters (i.e., strings) [50, 51]. It calculates how many edits that would be required in order to change one of the sequences into the other.

In Equation 2.6, the Levenshtein distance between two sequences a and b is formally defined, and calculated as $\mathcal{L}_{a,b}(|a|, |b|)$, where $|a|$ and $|b|$ is the length of a and b respectively. In the equation, a_i refers to the element at index i of sequence a, where both a and b are 1-indexed (i.e., starting at index 1).

$$\mathcal{L}_{a,b}(i,j) = \begin{cases} max(i,j) & \textbf{if } min(i,j) = 0 \\ min \begin{cases} \mathcal{L}_{a,b}(i-1,j) + 1 \\ \mathcal{L}_{a,b}(i,j-1) + 1 \\ \mathcal{L}_{a,b}(i-1,j-1) + \begin{cases} 1 & \textbf{if } a_i \neq b_j \\ 0 & \textbf{otherwise} \end{cases} \end{cases} & \textbf{otherwise} \end{cases} \tag{2.6}$$

As mentioned, the Levenshtein distance is commonly used to compare string sequences, where each character in the string is considered. However, the Levenshtein distance can be used for any sequence of elements, where the sequence is represented as an ordered set. In the case of strings, the sequence is simply an ordered set of all characters in a string.

2.6.2 Word Error Rate

Given a sound file X containing speech, with a corresponding true transcription $\hat{y}$, a model predicts a transcription y for X. To calculate the word error rate (WER), the words in y are put in an ordered set A, and the words in $\hat{y}$ are put in an ordered set B. As an example, if y is *hello world*, then A is $\{hello, world\}$. As shown in Equation 2.7, the WER is then defined as the Levenshtein distance $\mathcal{L}_{A,B}(|A|, |B|)$ between A and B, divided by the number of words in (or size of) B, denoted $|B|$.

$$WER = \frac{\mathcal{L}_{A,B}(|A|, |B|)}{|B|} \tag{2.7}$$

2.6.3 Character Error Rate

The character error rate (CER) is defined similarly to WER, as described in Section 2.6.2, but on a per-character basis instead of per-word. This means that each character in y is put in the ordered set A, and each character of $\hat{y}$ is put in the ordered set B. As an example, if y is *hello world*, then A is $\{h, e, l, l, o, \ , w, o, r, l, d\}$.

3

Datasets

Data is an integral part of machine learning, and in this chapter the datasets used as part of this project are presented in detail.

3.1 NST Acoustic Database for Swedish

The *NST Acoustic Database for Swedish (16kHz)*,[1] from now on referred to as the *NST Dataset*, is a dataset of Swedish speech with corresponding validated transcriptions. It was created specifically with speech recognition and dictation in mind. The dataset was created around the year 2000 by a Norwegian company named *Nordisk språkteknologi holding AS*, or *NST* for short. The company NST went bankrupt in 2003, but the NST Dataset has since then been made freely available to the public domain through the National Library of Norway, by directions from the Royal Norwegian Ministry of Culture.

Table 3.1: Dialects represented in the NST Acoustic Database for Swedish. Statistics are reported by the official documentation of the NST Dataset.

Region	**Part of dataset**
Stockholm and vicinity	19.87%
Northern Sweden (Norrland)	12.15%
West South Sweden (Västra sydsverige)	11.40%
Gothenburg and vicinity	10.77%
Central Sweden (Mellansverige)	10.53%
West Gothland (Västergötland)	7.85%
East Gothland (Östergötland)	7.69%
East South Sweden (Östra sydsverige)	7.29%
Dalarna and vicinity	6.79%
West Sweden (Västsverige)	5.65%

The NST Dataset is divided into two parts and consist of around 500 hours of transcribed Swedish speech in total. The first part is for training, and consist of around 400 hours of speech, and the remaining part is for testing. There is no overlap

[1] `https://www.nb.no/sprakbanken/show?serial=oai%3Anb.no%3Asbr-16&lang=en`

between the two parts. Additionally, no speaker in the training part is present in the test part, and vice versa.

As mentioned, the NST Dataset was created with speech recognition in mind, and as such has a large mix of speakers. There are a total of 1000 speakers represented in the dataset, and the dataset covers 10 major different dialects, as shown in Table 3.1. The speakers' ages range from 16–77, and there are 54% female and 46% male speakers. All recordings are made in a closed office environment, and have little background noise.

Table 3.2: Properties of the sound files in the NST Acoustic Database for Swedish.

Sample rate:	16 000 Hz
Resolution:	16 bit
Channels:	2 (stereo)
Encoding:	Linear PCM (LPCM)

The transcripts of the dataset are divided into multiple text files with a `.spl` file extension and Latin-1 (ISO 8859-1) encoding. Each such text file holds transcribed sentences for a single speaker, together with metadata about that speaker. The metadata includes a unique anonymous ID for the speaker, their age, sex and the region where they were raised (used to deduce dialect). For each sentence in the transcript, there is a reference to a sound file where the speaker says that sentence. The sound files have a `.wav` file extension, and has the properties described in Table 3.2.

3.2 Common Voice

Common Voice [11] is a large multilingual dataset of transcribed speech, created by Mozilla, made freely available under a Creative Commons Zero (CC0) license.[2] It is a community driven project, where anyone can contribute by either recording themselves reading given sentences, or validating other peoples recordings. As such, it is a constantly growing dataset with (at the time of writing) 40 different languages and over 3400 hours of validated speech in total.

Table 3.3: Properties of the sound files in the Common Voice dataset.

Sample rate:	48 000 Hz
Resolution:	16 bit
Channels:	1 (mono)
Encoding:	MPEG-3

The Swedish part of the Common Voice dataset is the only one of relevance for the aim of this thesis project. Common Voice consists of five hours of validated Swedish speech, by a total of 99 unique speakers. Comparing this with the English part of

[2] `https://www.mozilla.org`

Common Voice, with a total of 1118 hours of validated speech, or the NST Dataset described in Section 3.1, the Swedish part of Common Voice appears rather small. It is optional for contributors to disclose gender, and the statistic is entirely self reported. Out of those who have disclosed their gender for the Swedish part of the Common Voice dataset, 90% are male.

As anyone can contribute to the Common Voice dataset, using whatever recording equipment they have available, the sound quality of the speech in Common Voice can differ quite a lot. Most of the speech is recorded in an environment with a lot of background noise, in contrast to the recordings of the NST Dataset. All sound files have a `.mp3` file extension, and the sound properties are further described in Table 3.3.

3.3 Lunchekot Dataset

Lunchekot is a Swedish news radio program that is broadcast every day at lunch at 12:30 Swedish time. Data from this program has been used to create a dataset in the news broadcasting domain, which will from now on be referred to as the *Lunchekot Dataset*. Each episode of Lunchekot is 25 minutes long and for the main part contains what is considered the most important news for the day. Towards the end of each episode there are two short distinct segments with domestic news and sports news. The program is mainly recorded in studio where the host and some of the reporters and experts reside. In the case of interviews, the host or reporter is often in the studio while the interviewee is recorded from a phone call or in an environment with some background noise. In the first and more general part of the program, one often finds longer reports about the main news for the day, both from abroad and domestic. The domestic news and sports segments tend to keep the different elements shorter.

Some news are presented directly by the host, for every other element the host usually makes a hand-over. The hand-over is typically on the form *Now x will talk about y*, where x is the name of a reporter, and y is some news subject. After an element, the host typically either thank the reporter or, if not done before the start of the element, explains who was talking. In this case, the reporter typically says something along the lines of *Thanks for that x!*, where x is the name of the reporter. This format makes it easy to identify who is speaking, which was useful when creating the dataset.

Table 3.4: Properties of the sound files in the Lunchekot Dataset.

Sample rate:	16 000 Hz
Resolution:	16 bit
Channels:	1 (mono)
Encoding:	Linear PCM (LPCM)

The tempo of the broadcast is quite high, probably as there is a lot to present. Sometimes *Sveriges Radio*, the company behind Lunchekot, has longer reports on

various subjects. These are often shortly presented in Lunchekot, and the listener is then referred to the longer report if they are interested.

The Lunchekot program is freely available on Sveriges Radios website.[3] When compiling this dataset, a selection of 14 episodes spread out over 6 months, November 2019 to April 2020, were downloaded. This resulting dataset contains 5 hours and 2 minutes of speech with corresponding transcription and speaker annotation. In total there are 229 unique speakers, out of which 148 are male and 81 are female. Some speakers, mostly reporters, experts and hosts, appear in multiple episodes. Other speakers, mostly interviewees, appear in only a single episode.

Table 3.4 shows properties of the sound files in the Lunchekot Dataset. Since this dataset was devised as part of this thesis project, the sound files were deliberately created with the specific properties shown in the table.

Sound quality in this dataset is varying. The parts recorded in the Lunchekot studio are of high audio quality, with little distortion and background noise. These parts also make up the large majority of the dataset. Sound from interviews often have a lot of background noise and worse quality, but are in most cases still intelligible. Some interviews have a lot static noise, and are hard to interpret even for us humans, but they are quite rare.

3.4 Sports Dataset

In order to explore an even narrower domain than that of news broadcasting, the sports segment of the Lunchekot Dataset was extracted to form a new dataset; the *Sports Dataset*. This dataset consists of just the sports segment from Lunchekot, and could be considered to be in the domain of sports. As it is extracted from the Lunchekot Dataset, it has the same sound properties as shown in Table 3.4.

Looking at the distribution of speakers in the Sports Dataset, it has 29 unique speakers. Out of these, 21 are male and 8 are female. In total, this dataset consists of 18 minutes of transcribed speech, annotated with speaker and gender information. The sound quality of the Sports Dataset is mostly of good quality, with little background noise. This follows from the fact that most of the sound is recorded in the Lunchekot studio.

[3] `https://sverigesradio.se/ekot`

4

Method

In the following sections the method that has been used is described. Each of the main sections describes an important part of the project. Each of these parts is described in a more or less chronological order, in which they were completed.

First the datasets are downloaded and processed so that they can be used with Mozilla's DeepSpeech, following the process described in Section 4.1. In Section 4.2 a language model based on KenLM is then created from open text corpora available online. Using the language model and the processed datasets, Mozilla's DeepSpeech is used to create models for ASR. In Section 4.3, a general Swedish model is created, and in Section 4.4 the general Swedish model is used together with transfer learning to create domain specific models.

4.1 Dataset processing

Before any of the datasets can be used for training or testing of models, they have to be downloaded and processed into a suitable format.

All sound files containing speech need to be converted into a usable format. This follows from the fact that the DeepSpeech implementation by Mozilla is used, and it comes with some requirements. The desired sound properties are described in Table 4.1, and they differ from both the NST Dataset and Common Voice.

Table 4.1: Sound file requirements by Mozilla's DeepSpeech model.

Sample rate:	16 000 Hz
Resolution:	16 bit
Channels:	1 (mono)
Encoding:	Linear PCM (LPCM)

The transcripts have to be parsed from the original datasets, and be converted into a format that makes them usable together with Mozilla's DeepSpeech. Mozilla's DeepSpeech implementation wants the data in the form of comma-separated values (CSV) in a `.csv` file. Each line in the file, except for the first one, corresponds to one data point (a single spoken sentence). Each data point is formatted as a tuple of a path to the sound file where the sentence is spoken, the file size of the sound file in

bytes, and the transcription of the spoken sentence. The first line in the file contains a header for each column, namely `wav_filename`, `wav_filesize` and `transcript`.

Transcriptions are only allowed to contain a special set of characters, called an alphabet. For Swedish, we define the alphabet as the letter a–z, plus the letters *å*, *ä*, and *ö*. Any transcripts that contain characters outside the alphabet either have to be discarded, or the invalid characters have to be converted into a character in the alphabet. Transcripts are also made all lowercase, as the alphabet would otherwise need to contain all capitalized letters as well, meaning that e.g., the letters *A* and *a* would be treated as non-equivalent.

4.1.1 Processing the NST Dataset

Retrieving the NST Dataset is as straightforward as downloading four compressed archives from the National Library of Norway's website.[1] One of the compressed archives contains the test part of the NST Dataset, and the other three archives contain the training part. To make further processing easier, the contents of all three training parts are extracted into a directory called `train`, and the test part into a directory called `test`.

The properties of the sound files in the NST Dataset almost perfectly matches the requirements of Mozilla's DeepSpeech. The only mismatch is in the number of channels, where the sound files of the NST Dataset are stereo, but mono is required. Because of this, all sound files in the NST Dataset are converted to mono, by mixing the two channels together. Mixing the sound files into a single channel was done using the open-source utility *FFmpeg*.[2]

All transcripts of the NST Dataset are spread out over multiple different files, as described in Section 3.1, and thus need to be extracted. For each `.spl` file, speaker metadata was extracted from the top of the file using simple pattern matching. All the validated transcripts are placed between the two headers `[Validation States]` and `[End]`, and each of the transcripts are separated by a newline character. Each line between the two headers could thus be treated as a transcript, and parsed individually. Each such line includes a reference to a sound file, and the transcription of the spoken Swedish.

In order to make future processing steps easier, all transcripts were saved into a single `.csv` file, combining the transcript with all metadata known about it. Each transcript together with its sound file reference and speaker metadata was saved, with each comma-separated line containing all the information for a transcript. In addition to just saving a reference to each transcript's sound file, the file size and duration in seconds for the sound file was calculated and saved in the `.csv` file. The duration of each sound file was determined programmatically using an open-source utility called *SoX*.[3]

[1] `https://www.nb.no/sprakbanken/show?serial=oai%3Anb.no%3Asbr-16&lang=en`
[2] `https://www.ffmpeg.org/`
[3] `http://sox.sourceforge.net/`

Splitting the dataset

The dataset needs to be split into a training, development, and test part, as is common when developing machine learning models [6, 37, 52, 53]. The NST Dataset already comes with a test part, making up about 20% of the total data. This test set is left intact, such that our results can be compared with other work also using the NST Dataset, such as Kullmann [3]. The remaining training part from the NST Dataset is split into a new training part and a development part. The aim is to split the combined data such that the training part makes up 70%, the development part 10%, and the test part 20%. To achieve this, the training part of the NST Dataset is split into a new training part that is roughly 87.5%, and the remaining 12.5% goes into the development part.

When splitting the dataset, we wanted to maintain a balance in gender, duration, and dialect between the subsets as close as possible to the one in the original dataset. Also, most importantly, no speaker should appear in more than one of the three parts, as that might lead to overfitting to a specific speaker [10]. As the test part is fixed, we can not influence it, but we can influence the creation of the new training part and the development part. In order to maintain the desired balance when splitting the datasets, a simple randomized algorithm was developed, as displayed in Algorithm 4.1.

Algorithm 4.1 Randomized algorithm for splitting the NST Dataset

Require: data := *training part from NST Dataset*
  train, dev := {}
  thresholds := {10%, 0.1%, 0.1%} *// {dialect, duration, gender}*
  while not balanced **do**
    // Randomly split the data into two parts, based on speakers
    train, dev := $randomize(data, 87.5\%, 12.5\%)$
    // Check if the random split is balanced
    balanced := $is\_balanced(train, dev, thresholds)$
  end while
  return train, dev

The algorithm described in Algorithm 4.1 randomly splits the training part of the NST Dataset until it finds a split that satisfies the requirements. The splitting is done based on the speakers, such that 87.5% of all speakers will be in the new training part, and 12.5% of the speakers will be in the development part, with no overlap. After the randomized split, the resulting training and development subsets are automatically checked as part of the algorithm to make sure that they are balanced in terms of dialect, gender, and duration. When determining if a certain variable (e.g., dialect) is balanced, we allow for some margin of error, referred to in the algorithm as a threshold. For the gender and duration balance, we set the threshold to 0.1%. Since the difference between male and female in the NST Dataset is roughly 10%, setting a threshold of 0.1% means a difference in balance of e.g., 9.9% in either of the new parts is accepted. For the dialect balance, we set the threshold to 10%, since the total balance in the NST Dataset is not so good to

begin with, as seen in Table 3.1, making it hard to find an accepted split with lower thresholds.

The resulting training and development splits are saved in the files `train.csv` and `dev.csv` respectively. In these files, for each data point there is a file path, the file size, and the transcript of a sound file. These files are according to the specifications of Mozilla's DeepSpeech. Before these files are saved however, the data is also filtered according to the process described in the next section, and normalised according to the process described in Section 4.1.3.

Filtering bad data

Some of the data in the NST Dataset is not suitable for use with Mozilla's DeepSpeech. There is an upper limit of 10 seconds for the duration of the sound files. This only applies when training the model, but means that files longer than 10 seconds need to be filtered from the final training and development set.

There are some sound files that do not match their corresponding transcript. This is detectable from the fact that the duration is too short compared to the text transcript. Mozilla's DeepSpeech has a built-in detection for the case of too short sound files, so they are easy to detect. We simply ran all the training and test data through Mozilla's DeepSpeech, and took note of all sound files that were deemed bad, and filtered these out.

The NST Dataset contains some transcripts with non-standard Swedish characters, such as *è*, *ü*, *î*, and *ÿ*, as well as some transcriptions with the text *tyst under denna inspelning* (which is Swedish for *silent during this recording*) together with a silent sound file. All of these transcripts were removed from both the training and test part of the NST Dataset.

4.1.2 Processing Common Voice

The Common Voice dataset is downloaded from its official website.[4] Data for multiple different languages are available, but only the Swedish data is of interest for this project.

In contrast to the sound files from the NST Dataset, the sound files from Common Voice requires more processing before they are compatible with Mozilla's DeepSpeech. The sound files from the Common Voice dataset have both the wrong sample rate and the wrong encoding. Luckily, as the Common Voice dataset has been used together with Mozilla's DeepSpeech before, there exists an official script for processing all sound files. The official script is part of the DeepSpeech repository by Mozilla on GitHub.[5] It uses the open-source utility *SoX* for all sound file processing.[6]

[4] `https://voice.mozilla.org/en/datasets`
[5] `https://github.com/mozilla/DeepSpeech`
[6] `http://sox.sourceforge.net/`

The official processing script does some filtering of data not suitable for Mozilla's DeepSpeech. First of all, it only extracts the transcripts and sound files that have been validated by multiple users. It also filters out sound files that are too long for Mozilla's DeepSpeech. Finally, it also filters files that have a mismatch between the transcript and the actual sound file, i.e., if the duration of the sound file is too short for the provided transcription.

In order to use Common Voice as a means of evaluating the models developed as part of this project, all of the validated data was used. That means that no splitting into training, development, and test is made — there is instead just a test set.

4.1.3 Normalisation of transcripts

As previously mentioned, transcriptions are only allowed to contain a special set of characters. This means that the transcriptions for the datasets had to be normalised to fit the allowed alphabet. This normalisation is enforced by Mozilla's DeepSpeech, as it will crash if it detects characters which are not allowed.

Table 4.2: Character substitutions made when normalising dataset transcripts. The single blank space character (UTF-8 `0x20`) is denoted `BLANK`, and a character removal (i.e., the substitution with nothing) is denoted `REMOVED`.

Original	Substitution
-	BLANK
_	BLANK
.	REMOVED
,	REMOVED
;	REMOVED
?	REMOVED
!	REMOVED
:	REMOVED
"	REMOVED
[,], (,)	REMOVED
|	REMOVED
\, /	BLANK
é, è	e
&	och
%	procent

Normalising the transcripts is easy from a technical point of view, but considerably harder from a linguistic point of view. The actual normalisation is just a matter of searching for disallowed characters, and either replacing them or removing them according to some rules, but these rules need to be defined. It is important that the rules are defined in such a way that a transformation does not create a mismatch between the transcript and what is actually being said in the sound file, as this might impact the final model in a negative way. All character substitutions that were made are shown in Table 4.2.

Numbers and digits would usually have to be normalised into words, as there are no digits in the alphabet. In the NST Dataset, this has already been done, where e.g., the number 97470 is transcribed as *nittiosju tusen fyrahundrasjuttio*. As far as we know, there are no numbers in the Common Voice dataset. In summary, no normalisation of numbers and digits is required for the dataset transcriptions.

4.2 Language model

When doing ASR using models such as DeepSpeech [5] or wav2letter [6], a language model can typically be used to improve the accuracy of the inference [6, 10, 37]. As, to the extent of our knowledge, there are no openly available language models for Swedish compatible with Mozilla's DeepSpeech, a language model had to be created from scratch.

For compatibility with Mozilla's DeepSpeech, the language model is created using the KenLM [14] toolkit. The KenLM toolkit comes with some tools that can be used to create a KenLM language model. Most notably, the `lmplz` and `build_binary` tools are provided. The `lmplz` tool is used to create the actual n-grams based language model given some text corpus, and the `build_binary` tool converts the language model into an efficient binary representation.

Before a language model could be created using `lmplz`, we had to create a text corpus that the model could be based on. Previous research has typically used massive open text corpora online, such as Wikipedia and the Europarl corpus [10, 31], so we also opted for this approach. Besides using the Swedish version of Wikipedia, and the Swedish part of Europarl, we have also used transcripts from speeches in the Swedish Parliament, and news articles from the Swedish newspaper *Göteborgs-Posten*.

Table 4.3: Text corpora used to create Swedish language model.

Name	**Size**	**Source**
Swedish Wikipedia	2.1 GB (46.80%)	Wikipedia
Swedish Europarl	202 MB (4.57%)	Språkbanken
Swedish Parliament	697 MB (15.81%)	Språkbanken
Göteborgs-Posten	1.5 GB (32.82%)	Språkbanken
Total	4.4 GB (100%)	

See Table 4.3 for a list of the corpora used, and how large they are in relation to each other. The Swedish version of Wikipedia is downloaded from the official Wikipedia database dump.[7] The Swedish Europarl corpus, the Swedish Parliament speech transcripts, and the texts from Göteborgs-Posten are downloaded from *Språkbanken Text*,[8] the division of the National Language Bank.

None of the corpora can be used in their original form, and therefore require some pre-processing before being fed into the KenLM toolkit. Both the corpora from

[7] `https://dumps.wikimedia.org/`
[8] `https://spraakbanken.gu.se/`

Språkbanken, and the Swedish version of Wikipedia are originally in a structured XML format, containing metadata. For the language model, only the actual Swedish sentences are of interest, and the KenLM toolkit expects a single plain text file of all text. In order to extract the plain text articles, removing structured data such as tables etc., from Wikipedia, the open-source tool *WikiExtractor* was used.[9] To extract plain text from the XML files from Språkbanken, we had to implement our own simple script.

Additionally, a language model which includes a corpora based on the training part of the NST Dataset was created. This language model was created using all of the corpora from Table 4.3, plus all the transcripts from the training part of the NST Dataset.

4.2.1 Corpora normalisation

Previous research suggests that simply using the plain texts from the corpora is not enough, but the sentences should also be normalised such that they are closer to spoken language [31]. In Agarwal et al. [31], a tool called MaryTTS was used to do the normalisation.[10] The normalisation steps includes expansion of abbreviations, and conversion of digits and numbers into words (e.g. 97470 into *ninety seven thousand four hundred seventy*). On the official website of MaryTTS, support for the Swedish language is claimed, but after examining the tool further no such support seems to exist.

As Swedish support in MaryTTS is lacking, we had to implement our own normalisation script, with inspiration from MaryTTS. As it would likely be an entire additional thesis project to implement an entire text normalisation system for Swedish, we had to focus our efforts. We primarily focused on implementing number to word normalisation, with a word representation close to the one found in the NST Dataset, where e.g., 97470 is normalised as *nittiosju tusen fyrahundrasjuttio* (*ninety seven thousand four hundred seventy*). Additionally, we wanted to make sure to expand some common abbreviations, as shown in Table 4.4. When doing this, we also normalised square and cubic units, such that e.g., m^2 is normalised to *kvadratmeter* (*square meters*) and dm^3 is normalised to *kubikdecimeter* (*cubic decimeters*). This type of normalisation was implemented using regular expressions and search-and-replace. In addition to normalisation of numbers and abbreviations, the same substitutions as described in Section 4.1.3 were performed, shown in Table 4.2.

Before applying the number to word normalisation described, we realised that due to how numbers are typically written, they themselves often had to be pre-normalised before being normalised into words. The patterns observed were numbers with decimals, e.g., $3,14$ (decimals written with comma in Swedish), number ranges, e.g., $18-65$, and numbers being separated by whitespace, e.g., 97 470. These were pre-normalised as *3 komma 14* (*3 comma 14*), *18 till 65* (*18 to 65*), and 97400 respectively.

[9] `https://github.com/attardi/wikiextractor`
[10] `http://mary.dfki.de`

Table 4.4: Normalisation of Swedish abbreviations.

Abbreviation	Normalised
bl.a.	bland annat (*including*)
t.ex.	till exempel (*for example*)
osv.	och så vidare (*and so on*)
mm	millimeter
cm	centimeter
dm	decimeter
km	kilometer
m	meter
ml	milliliter
cl	centiliter
dl	deciliter
l	liter

An issue encountered with number to word normalisation is that of correctly normalising years and centuries. With our implementation, e.g., the year 1923 is normalised as *ett tusen niohundra tjugotre* (*one thousand nine hundred twenty three*) instead of *nittonhundra tjugotre* (*nineteen twenty three*). This does not correspond to how a person would typically pronounce a year in Swedish. The same problem exists in Swedish when dealing with centuries, such as *1900-talet* (*20th century*), where with our implementation the normalisation would incorrectly be *ett tusen niohundra talet* instead of *nittonhundra talet*.

4.2.2 Compilation

After all corpora was normalised and combined, a Swedish language model could be compiled using the KenLM toolkit. The `lmplz` tool allows specifying n, when creating the n-grams model. Both a 3-grams and a 5-grams model were created from the combined corpora, so that the effect of n could be examined. Two additional 5-grams models based on just Wikipedia were also created, one from the normalised version and one from the un-normalised one. These were created such that the effect of normalisation and a larger corpora could be examined.

Once the language models had been created, they all had to be converted into a binary and memory optimized trie data structure. This is required by Mozilla's implementation of DeepSpeech. The `build_binary` tool in the KenLM toolkit was used to create binary trie representations of the language models. Mozilla's DeepSpeech also comes with a tool called `generate_trie`, which takes the binary trie from KenLM as input. It also takes an alphabet, i.e., a list of allowed characters (as described in Section 4.1). When using Mozilla's DeepSpeech, it requires both the binary trie from the `build_binary` tool, and the output trie from the `generate_trie` tool in order to utilise the language model during inference.

4.3 Developing the general Swedish model

The general Swedish model was developed using Mozilla's DeepSpeech. The processed NST Dataset was primarily used for training the model. Besides typical hyperparameter tuning, two different approaches to training were explored. The first, and most straightforward approach, was to train a DeepSpeech model from scratch. In this case, the model was only trained on the training part of the NST Dataset. The second approach was to fine-tune the released pre-trained English model on the NST Dataset, and is described further in Section 4.3.1. The model trained from scratch serves as a baseline to evaluate the model trained using transfer learning.

When exploring hyperparameters for the model, mini-batch size, dropout rate, and learning rate were considered. As a starting point, the parameters that were used to train the publicly available pre-trained English model for Mozilla's DeepSpeech were copied. These parameters are shown in Table 2.1. However, due to hardware memory limitations, a mini-batch size of 128 could not be used, so 64 was used as a starting point instead. Different combinations of hyperparameter values were manually explored.

In order to know when to stop training of a particular model, early stopping [21] was used. After each epoch of training, the average loss was calculated over the development part of the processed NST Dataset. If there had been no improvements in loss for the past four epochs, training was stopped.

4.3.1 Transfer learning from English

As mentioned, the second approach explored was to use transfer learning from a pre-trained English model, so called fine-tuning (see Section 2.2). Transfer learning has previously successfully been used when developing ASR systems for languages with limited training data [10–12], which is what motivated the exploration of this approach. Instead of first training a DeepSpeech model for English ourselves, the pre-trained English model by Mozilla was used.

A problem with the pre-trained English model was that it was trained using a different alphabet. In particular, it was trained to support the '-character, and it did not recognise the Swedish letters *å*, *ä* and *ö*. In order to mitigate this problem, the output layer of the model was replaced with a new layer with a size matching our desired alphabet. This however meant that the weights from the original output layer were discarded and not transferred, but previous work indicate that the output layer is not as important as the previous layers [10, 11].

After making the architectural changes required to continue training with Swedish data, training models using transfer learning was made in the same fashion as when training models from scratch. The only difference would be that the models utilising transfer learning would first load the weights from the English model, before starting training on the Swedish data. None of the layers were frozen during the training.

4.3.2 Evaluation

After the training of each model was complete, the WER, CER, and average loss was calculated on the development part of the processed NST Dataset. These measurements were then used to determine a best performing model. After hyperparameter tuning, and exploring the two different approaches, a best performing model was determined. This model was then used to calculate the WER, CER and average loss on the test part of the NST Dataset, in order to get an evaluation result that could be compared to Kullmann [3]. Multiple different language models were tested in order to evaluate how different language models affected the WER and CER.

Besides comparing against Kullmann [3], we also wanted to compare the model against commercial services, such as Google's Cloud Speech-to-Text. In order to make such a comparison, the WER and CER had to be calculated for the test part of the NST Dataset, using Google's Cloud Speech-to-Text. To calculate this, each of the sound files from the test part was sent to the service to get a prediction. The predictions were then compared to the actual transcripts, and the WER and CER could be calculated. However, Google's Cloud Speech-to-Text service transcribes numbers with actual digits, making the transcription incompatible with the alphabet used throughout this project. To make for a fair comparison, we also normalised all numbers into their corresponding words, such that e.g., 24 becomes *tjugofyra* (twenty four), similarly to the number normalisation described in Section 4.1.3 and Section 4.2.1.

Finally the best performing models are evaluated in terms of WER and CER against the Common Voice dataset for Swedish. This is done in order to get some estimate of how the models generalise outside of the NST Dataset domain.

4.4 Developing the domain specific models

In order to create domain specific ASR models, domain specific datasets were required. The datasets were created as part of this project, and the data for them was collected manually. Two datasets, and thus two different domain specific ASR models, were created — the *Lunchekot Dataset* within the news broadcasting domain, and the *Sports Dataset* within the sports domain.

Once the domain specific datasets were in place, transfer learning based on the general Swedish model was explored. More specifically, the general Swedish model was fine-tuned on the domain specific datasets.

Finally, after hyperparameter tuning of the new domain specific models, the general Swedish model was compared to the new domain specific models on the test part of each respective dataset.

4.4.1 Dataset creation

In order to create a domain specific dataset, a source of domain specific speech data was needed. As mentioned in Section 1.2.2, the dataset used to train the domain specific model should be significantly smaller than the one used for the general model, in order to successfully reach the aim of the thesis. After exploring alternatives, the decision fell on using the daily radio news show called *Lunchekot* from the Swedish public service radio company *Sveriges Radio*.[11]

Based on the data from Lunchekot, two datasets were created. The first dataset contains all of the speech data from Lunchekot, making up a very broad domain. The other dataset only contains a subset of the Lunchekot data, namely the sports segment, where exclusively sports are discussed. The second dataset is considered much more specific, or narrow, in the domain aspect. Additionally, it is much smaller in size, as it is made up of a smaller subset of the first dataset.

As mentioned in Section 3.3, a total of 14 episodes of Lunchekot were arbitrarily selected, such that there was at least a couple of days between each episode. Each episode was manually cut into multiple shorter sound clips. The cutting was made such that each new sound clip contained just a single sentence from a single speaker, and such that the sound clip was shorter than 10 seconds. As some sentences were spoken for longer than 10 seconds, they sometimes had to be cut into two parts. This cutting was made in a way such that the sentences were as semantically meaningful as possible when split into two separate parts. Any non Swedish speech was removed, with exceptions for names, locations, and common English loan words and expressions used as part of an otherwise Swedish sentence. The importance of keeping the sound clips shorter than 10 seconds comes from the limit, during training, in Mozilla's DeepSpeech.

After cutting the episodes of Lunchekot into sound clips of sentences, each sentence was manually transcribed by the authors. The sentences were transcribed by listening to each sentence, and writing the transcription manually in a text file, where each sentence was on a separate line. While transcribing, all numbers were written as text. After transcribing each sound clip, all transcriptions were normalised and validated by at least one other person in addition to the person who originally transcribed it. Any disagreement in the transcriptions were settled by bringing in a third person, such that the majority would decide on the transcription to use. The authors as well as all additional third persons consulted during the transcription process have Swedish as their native language.

All transcriptions together with their corresponding sound clips were also annotated with a speaker ID and the gender of the speaker. The speaker ID was mostly straightforward to extract, due to the format of Lunchekot. In the program, a host usually says who just spoke or is about to speak. The gender of the speaker was annotated both by researching the name of the speaker, and by listening to the speakers voice. Like with the transcriptions themselves, these annotations were

[11] `https://sverigesradio.se`

entered manually in a text file. The annotation was made to aid with splitting the datasets, which was the next and final step.

Both datasets had to be split into a training, development, and test part, similarly to how the NST Dataset was split. Just like with the NST Dataset, we wanted the datasets to be split into 70% training, 10% development, and 20% test. Making the actual split could be done using the same method as for the NST Dataset, namely using Algorithm 4.1. This was possible as the transcriptions were annotated with gender and speaker ID. Since there is no information about dialects however, this parameter was not taken into consideration in the algorithm. When splitting the first dataset, the thresholds had to be updated to 10% for the gender distribution, and 1% for the duration distribution. The updated thresholds were required as a split could not be found with the same threshold parameters as for the NST Dataset. For the second dataset, the sports specific one, the gender and speaker ID parameters were also ignored when making the split, as the total number of unique speakers was very small. This meant that the second dataset was split only based on duration.

4.4.2 Fine-tuning the general Swedish model

After having both dataset in place, the domain specific models could be created. As stated in Section 1.1, one aim of this thesis is to explore if transfer learning can be used to achieve better results in specialised domains. To do this, the general Swedish model was used as the basis for transfer learning in the form of fine-tuning.

The general Swedish model was used as a base for both of the domain specific models. The models were fine-tuned on the training part of the general Lunchekot Dataset and the Sports Dataset respectively. Similar to how training of the general Swedish model was performed, the model was evaluated against the development part of the datasets after each epoch of training.

Multiple values for the hyperparameters learning rate, dropout, and mini-batch size were explored for both models, similarly to how hyperparameter search for the general Swedish model was performed. As the domain specific datasets are significantly smaller in size compared to the NST Dataset, training of the models was very quick. This meant that thousands of different hyperparameter combinations could be tested in just a day, compared to roughly one per day when training the general Swedish model on the NST Dataset. To aid the process of hyperparameter search given the quick training time, a simple script which randomly generated and tested hyperparameters was used. Random search for hyperparameter optimization has been shown to be more efficient than manual or grid search approaches, when feasible [54].

4.4.3 Evaluation

While developing the domain specific models, after the training of each model variant was completed, the WER, CER and average loss was calculated on the development part of the dataset of which the model was trained on. Similarly to how the general

Swedish model was evaluated, the best performing model variant for each dataset was selected. The best performing variant was then tested in terms of WER, CER and average loss on the test part of the dataset on which it was trained on.

In order to compare the domain specific model to the general Swedish model, the general Swedish model was also evaluated on the test part of the general Lunchekot Dataset and the Sports Dataset. This allows determining whether or not transfer learning could indeed be used to create a better performing domain specific model, evaluated on two different domain specific datasets.

Similarly to the evaluation of the general Swedish model in section 4.3.2, multiple different language models were evaluated. In order to do this, two additional language models were created. These two language models were based on the best performing language model for the general Swedish model, but also included the transcripts from the training part of both the domain specific datasets respectively.

5

Results

This chapter presents results obtained from experiments conducted in the project. The structure closely follows that of Chapter 4, to make it easier to follow along.

5.1 Dataset processing

Dataset processing was done according to the methods described in Section 4.1. The results of processing each of the datasets are described below.

5.1.1 NST Dataset

Using the algorithm described in Section 4.1.1, the training part of the NST Dataset (referred to as *NST Original Training* in Table 5.1) could successfully be split into a new training part, as well as a development part. These new splits are referred to as *NST Train* and *NST Dev* respectively. Statistics for the resulting splits are shown in Table 5.1, together with statistics for the original training and test part. The test part is referred to as *NST Test*.

Table 5.1: Resulting datasets after processing the NST Dataset.

Dataset	**Length (h)**	**Male (%)**	**Female (%)**
NST Original Training	392	45	55
— NST Train	343	45	55
— NST Dev	49	45	55
NST Test	103	49	51
Total	495	46	54

When processing the NST Dataset, some files were filtered out. Most importantly, some files had to be filtered out of the NST Test set. In total, NST Test contains 73047 files, out of which 87 had to be filtered out. Table 5.2 lists how many files had to be removed, grouped by the reason for removal, together with their fraction of the total size of NST Test. A total of 83 files were removed as they only contained silence. These were the files that had the transcription *tyst under denna inspelning* (which is Swedish for *silent during this recording*), as described in Section 4.1.1. Finally, 2 files were filtered out as their transcriptions were too short to match

the duration of their sound file, and 2 files were filtered out as their transcriptions contained unwanted characters not in the alphabet. See Appendix A for a complete description of filtered files.

Table 5.2: The number of files filtered out of the total size of NST Test, grouped by the reason for filtration.

Reason	**Number of files**	**Part of total (%)**
Silent recording	83	0.11
Duration too short for transcript	2	< 0.01
Unwanted character	2	< 0.01
Total removed	87	0.12
Total	73047	100

5.1.2 Common Voice

After using the processing script that comes with Mozilla's DeepSpeech for processing the Swedish part of the Common Voice dataset, roughly five hours of data remained. These five hours corresponds to the validated part of the dataset. The remaining 15% of the dataset was left out as it had not been validated.

5.2 Language model

Three language models were created, two using 5-grams, and one using 3-grams. All language models are based on the corpora mentioned in Table 4.3, following the method described in Section 4.2. Additionally, one of the 5-grams based language models also include a corpora based on the training part of the NST Dataset. The language models are described in Table 5.3. The storage space required grows exponentially when increasing the order of the language models. The 3-gram model uses 2.6GB while both of the 5-gram models use about 12GB.

Table 5.3: Resulting language models and the corpora used to create each of them, as indicated with a check-mark.

LM Name	**Swedish Wikipedia**	**Swedish Europarl**	**Swedish Parliament**	**Göteborgs-Posten**	**NST Training**
5-grams	✓	✓	✓	✓	
5-grams + NST	✓	✓	✓	✓	✓
3-grams	✓	✓	✓	✓	

5.3 General Swedish model

Below are the results from testing the best performing models trained from scratch and trained using transfer learning from a pre-trained English model. The results

show that the model that utilises transfer learning outperforms the model trained from scratch, with an absolute difference of over 5% in WER.

The results from evaluating the models on the Swedish part of Common Voice shows a far worse WER and CER for both the model trained from scratch and the model utilising transfer learning when compared to the results on NST Test. However, comparing the results on NST Test to Google's Cloud Speech-to-Text shows that the best model is on par with one of the best commercial services available, as described in Section 5.3.3.

Both the model trained from scratch and the model utilising transfer learning took on average about 12 hours to train.

5.3.1 Training from scratch

The best performing model trained from scratch used the hyperparameters shown in Table 5.4. These are the same as the for the pre-trained English model from Mozilla, with the exception of the changed mini-batch size. Since training a model from scratch is the most straightforward and typical approach, these results are regarded as a baseline for comparison and evaluation of the model trained using transfer learning.

Table 5.4: Hyperparameters for best performing model trained from scratch.

Parameter	Value
Mini-batch Size	64
Learning Rate	0.0001
Dropout	0.2

In Table 5.5, the results from testing the model on NST Dev, NST Test, and Common Voice are shown, using different language models. The best result in terms of WER and CER for each dataset is underlined. The overall best language model, in terms of on average lowest WER over all datasets, is highlighted in bold font. The lowest WER on NST Test achieved using the ASR model trained from scratch was 19%, and used the 5-grams + NST language model. This language model was also the overall best performing language model.

Table 5.5: Test results for ASR model trained from scratch. The best WER and CER for each dataset is underlined. The on average best language model is in bold font.

	NST Test		Common Voice	
Language Model	**WER (%)**	**CER (%)**	**WER (%)**	**CER (%)**
No language model	39.22	11.33	83.17	43.08
5-grams	19.27	07.10	68.08	41.59
5-grams + NST	<u>19.01</u>	<u>07.05</u>	68.08	<u>41.58</u>
3-grams	19.42	07.16	<u>68.04</u>	<u>41.58</u>

5.3.2 Transfer learning from English

Hyperparameters for the model trained using transfer learning based on a pre-trained English model are shown in Table 5.6.

Table 5.6: Hyperparameters for best performing model trained using transfer learning.

Parameter	Value
Mini-batch Size	64
Learning Rate	0.0001
Dropout	0.3

In Table 5.7, the results from testing the model on NST Dev, NST Test, and Common Voice are shown, using different language models. The best result in terms of WER and CER for each dataset is underlined. The overall best language model, in terms of average lowest WER over all datasets, is highlighted in bold font. The lowest WER achieved, on NST Test using the transfer learning model was 14%, and used the 5-grams + NST language model. This language model was also the overall best performing language model.

Table 5.7: Test results for ASR model trained using transfer learning. The best WER and CER for each dataset is underlined. The on average best language model is in bold font. Improvements over each result in Table 5.5 are highlighted and in parentheses.

	NST Test		Common Voice	
Language Model	**WER (%)**	**CER (%)**	**WER (%)**	**CER (%)**
No language model	27.55 (-11.67)	07.56 (-3.77)	67.44 (-20.73)	32.17 (-10.91)
5-grams	14.03 (-5.24)	04.82 (-2.28)	52.21 (-15.87)	30.64 (-10.95)
5-grams + NST	13.80 (-5.21)	04.78 (-2.27)	52.21 (-15.87)	30.63 (-10.95)
3-grams	14.16 (-5.26)	04.87 (-2.29)	52.22 (-15.82)	30.66 (-10.92)

5.3.3 Evaluating commercial services

After letting Google's Cloud Speech-to-Text predict transcriptions for NST Test, a WER and CER of 27% and 17%, respectively, could be calculated. After normalising numbers produced by Google's Cloud Speech-to-text, the WER and CER were improved to 21% and 9%, respectively. Even after normalisation, the error rate is significantly higher than for the ASR model using transfer learning developed as part of this project. Table 5.8 shows a transcription by Google's Cloud Speech-to-Text, alongside with its normalisation and the ground truth. It is clear that the decision to group some numbers together has a large impact on the final normalised prediction compared to the ground truth. A rough estimate of the NST Test set indicates that about 10% of the transcriptions are just numbers; however, removing these when evaluating Google's Cloud Speech-to-Text service did not yield any significant improvement.

Table 5.8: An example of a predicted transcription by Google's Cloud Speech-to-Text together with its normalisation, compared to the ground truth.

Prediction	4128 22 616
Normalised Prediction	fyra tusen etthundratjugoåtta tjugotvå sexhundrasexton
Ground Truth	fyra ett två åtta tjugotvå sexhundrasexton

The transcriptions of NST Test produced by Google's Cloud Speech-to-Text service are published.[1] This enables future researchers to use these for comparison and evaluation, without having to spend more money on Google's service.

5.4 Domain specific models

The following subsections describe the results from developing two domain specific models, the *Lunchekot domain model* and the *Sports domain model*.

5.4.1 Datasets

The domain specific datasets were created according to the described method in Section 4.4.1. In total, just over 5 hours of speech was transcribed, validated and annotated from Lunchekot by the authors. These five hours of speech make up the Lunchekot Dataset and is split into 70% training, 10% development, and 20% test. From the Lunchekot Dataset, the sports segments of each episode were extracted to form a second domain specific dataset, the Sports Dataset. The Sports Dataset is just 18 minutes of speech in total and is also split into 70% training, 10% development, and 20% test. Both the Lunchekot Dataset and the Sports Dataset are described in further detail in Section 3.3 and Section 3.4 respectively.

5.4.2 Language models

Besides using the best performing language model from the general Swedish model, the *5-grams + NST* language model, two additional language models were created. One 5-grams language model which also includes a corpora based on the training part of the Lunchekot Dataset, and one 5-grams model which also includes the training part of the Sports Dataset. Both of the two new language models also include the corpora used for the 5-grams + NST language model. The two new language models are referred to as *5-grams + NST + Lunchekot* and *5-grams + NST + Sports* respectively.

5.4.3 Lunchekot domain model

The hyperparameters for the best performing model variant fine-tuned on the Lunchekot Dataset are shown in Table 5.9. This model is based on the best performing general

[1] `https://github.com/se-asr/nst-google`

Swedish model, as described in Section 5.3.2. Fine-tuning of the domain specific model was completed in a couple of minutes, compared to the multiple hours of training required for the general Swedish model.

Table 5.9: Hyperparameters for best performing domain specific model fine-tuned on the Lunchekot Dataset.

Parameter	Value
Mini-batch Size	37
Learning Rate	0.00011
Dropout	0.172

In Table 5.10, the results from testing the model in terms of WER and CER on the test part of the Lunchekot Dataset are shown. A comparison is also made with the results from testing the best performing general Swedish model on the test part of the Lunchekot Dataset. As can be seen in the table, the domain specific model yields an absolute improvement of on average 12% in terms of WER, and 6% in terms of CER, compared to the general Swedish model.

Table 5.10: Test results for the Lunchekot domain model compared to test results for the general Swedish model. Results are from testing on the test part of the Lunchekot Dataset. The best WER and CER are underlined.

	Language Model	WER (%)	CER (%)
General Swedish model	No language model	58.30	20.38
	5-grams + NST	36.36	15.91
	5-grams + NST + Lunchekot	35.95	15.80
Lunchekot domain model	No language model	45.87	14.31
	5-grams + NST	25.26	10.35
	5-grams + NST + Lunchekot	<u>24.72</u>	<u>10.13</u>

The first example in Table 5.11 shows an interesting transcription predicted by the Lunchekot domain model on the Lunchekot Dataset. The predicted transcription gets a 100% word error rate, but only a 4% character error rate. As can be seen in the table, the only error is a missing *s*, resulting in the compound word being separated into two. The remaining examples in the table show typical errors, as well as a perfect transcription with zero word error rate.

5.4.4 Sports domain model

The hyperparameters for the best performing model variant fine-tuned on the Sports Dataset are shown in Table 5.12. This model is based on the best performing general Swedish model, as described in Section 5.3.2. Fine-tuning of the domain specific model was completed in a couple of minutes, compared to the multiple hours of training required for the general Swedish model.

In Table 5.13, the results from testing the model in terms of WER and CER on the test part of the Sports Dataset are shown. A comparison is also made with the

Table 5.11: Examples of interesting transcriptions made by the Lunchekot domain model on the Lunchekot Dataset.

Ground Truth	eller samtalsbehandlingar
Prediction	eller samtal behandlingar
WER	100.00 %
CER	4.00 %
Ground Truth	hela regelverket behöver uppdateras
Prediction	hela regelverket för häver uppdateras
WER	50.00 %
CER	14.29 %
Ground Truth	ericssons styrelse föreslår att utdelningen ökar ordentligt
Prediction	eriksson styrelse föreslår att utdelningen ökar ordentligt
WER	14.29 %
CER	3.39 %
Ground Truth	en säkerhetsåtgärd för att undvika smittspridning
Prediction	en säkerhetsåtgärd för att undvika smittspridning
WER	0.00 %
CER	0.00 %

Table 5.12: Hyperparameters for best performing domain specific model fine-tuned on the Sports Dataset.

Parameter	Value
Mini-batch Size	26
Learning Rate	0.00016
Dropout	0.282

results from testing the best performing general Swedish model on the test part of the Sports Dataset. As can be seen in the table, the domain specific model yields an absolute improvement of on average 9% in terms of WER, and 4% in terms of CER, compared to the general Swedish model.

Some examples of transcriptions made by the Sports domain model on the Sports Dataset are shown in Table 5.14. The table shows an example of a perfect transcription (WER is 0%), an okay transcription (WER is 33%), and a transcription which is bad (WER is 67%). Looking at more examples, the cause of errors seems to be rather arbitrary. There are examples of 20 word long sentences scoring a perfect 0% WER, but there are also examples of much shorter sentences which are completely wrong. A lot of the times, a name is just misspelled, or a word is missing a letter such as when the word inflection is wrong. It sometimes gets rather obscure names of either teams or people correct, and sometimes fails a lot. An example of a name it correctly transcribes is *Linda Hofstad Helleland*, which would not typically be considered common last names.

Table 5.13: Test results for the Sports domain model compared to test results for the general Swedish model. Results are from testing on the test part of the Sports Dataset. The best WER and CER are underlined.

	Language Model	**WER (%)**	**CER (%)**
General Swedish model	No language model	62.12	20.59
	5-grams + NST	40.04	16.19
	5-grams + NST + Sports	28.37	13.14
Sports domain model	No language model	52.24	16.49
	5-grams + NST	32.32	12.33
	5-grams + NST + Sports	<u>19.03</u>	<u>08.56</u>

Table 5.14: Examples of a perfect, okay, and bad transcription made by the Sports domain model on the Sports Dataset.

Ground Truth	och svenskt missflyt har det även varit i skidskytte
Prediction	och svenskt missflyt har det även varit i skidskytte
WER	0.00 %
CER	0.00 %
Ground Truth	tror inte zlatans karriär är över
Prediction	tror inte slattats karriär över
WER	33.33 %
CER	18.18 %
Ground Truth	kvart över två i eftermiddag i p fyra extra avslöjas årets kandidater
Prediction	juventus två eftermiddagen ipfyra extra varslas årets kandidater
WER	66.67 %
CER	27.94%

6

Discussion

In this chapter, some of the key results are highlighted and discussed in further detail, in relation to the methodology and constraints of the project. Each section can be considered a key takeaway or point of discussion.

6.1 The impact of language models

Language models have been considered for each model presented throughout this thesis. In each case, the use of a language model improved the performance significantly, as can be seen in Table 5.5, Table 5.7, Table 5.10 and Table 5.13. The tables show that using any language model is considerably better than not using a language model at all, both in terms of WER and CER.

That being said, the difference between the models that have been used is minimal. First of all, the order of the model had a small impact; there was less than a 1% absolute improvement between using 3-grams and 5-grams. Considering the small improvement and the exponential growth in storage space required, as described in Section 5.2, it is not obvious that a greater order n-grams is always desired. As an example, in this thesis we decided to not use a higher order than 5-grams due to storage limitations. The small improvement when going from 3-grams to 5-grams also implies that 3-grams are enough in most cases.

Another observation is that increasing the size of the set of corpora used to create the language model can be an effective technique to increase its performance. This effect is clear when looking at the evaluation results of the general Swedish model and the domain specific models, shown in the tables mentioned in the previous paragraph. Adding the training and development parts of the NST Dataset to the language model improves the evaluation results of the general Swedish model. Additionally adding the training and development parts of the domain specific datasets respectively improves the domain specific models evaluation results even further. The improvement can be seen for all models, but it is especially noticeable for the sports specific domain model in Table 5.13. This table shows an improvement from 32% to 19% by simply adding the training and development part of the Sports Dataset to the language model. However, this dataset is also special in that it was not split with the speaker IDs in mind, as is further discussed in Section 6.5. Nonetheless, it shows the impact of the language model and how, depending on the use case, adding

context to the language model could improve performance significantly. In a domain where very specific phrases and terms are used, adding these to the language model can be very beneficial.

As described in Section 4.2.1, there are obvious errors in our approach to normalising the language model corpora. These issues are primarily related to how years are typically spoken in relation to how they are written. Since the approach considered in this thesis does not deal with these issues at all, it is highly likely that the ASR models developed are rather bad at transcribing speech talking about years and dates. However, this is not something we have found direct evidence of. The lists of special characters and abbreviations normalised, as described in Table 4.2 and Table 4.4 respectively, are far from complete considering the whole Swedish language. This could also negatively impact the performance of our models. Some of the most common cases in everyday speech are handled though, and should at least lead to some improvement compared to not doing this normalisation at all. In the work by Agarwal et al. [31], a tool called MaryTTS was used to normalise German.[1] To the extent of our knowledge, no such tool exists for the Swedish language. It is likely that model performance could be increased further if a more powerful normalisation tool existed, one that could support more special cases, expand more abbreviations and make the text more similar to spoken language.

6.2 A general model for Swedish ASR

When developing the general Swedish model, two different approaches were considered, both using Mozilla's DeepSpeech. The first and most straightforward approach was to train a model completely from scratch, and the second approach was to utilise transfer learning. Comparing the results from using both approaches, as seen in Table 5.5 and Table 5.7, it is clear that the transfer learning approach significantly outperforms training a model from scratch. These results seem to show that transfer learning is indeed a powerful method also for Swedish ASR. This was expected considering recent research on using transfer learning for other languages in the same language family as the Swedish language [10, 11]. To the extent of our knowledge, the results obtained as part of this thesis are the first using transfer learning in the field of Swedish ASR.

Although the general Swedish model does achieve a new state-of-the-art result, as discussed further in Section 6.3, it leaves a lot to be desired. The most disappointing results were achieved when evaluating the model on the Swedish part of the Common Voice dataset. The best WER achieved on the Common Voice dataset is as high as 52%, which is almost four times as high as the result achieved on NST Test. A big difference between the speech data in Common Voice and NST Test is the sound quality. The sound in Common Voice has a lot of background noise while the sound in NST Test is of high quality with little background noise. Looking at the results on the Lunchekot Dataset and Sports Dataset, the general Swedish model achieves 36% and 28% WER respectively. Common for both these datasets

[1] `http://mary.dfki.de`

is that they have a mix of speech data, with varying amounts of background noise. An interpretation of these results is that the general Swedish model has issues with lower quality sound containing much background noise. Although a disappointing observation, it is not so surprising considering that the NST Dataset used to fine-tune the model consists of speech data with little background noise. If more large transcribed Swedish datasets, like the NST Dataset, would have been available, the model would likely get better overall results. Such a dataset could have been used either instead of the NST Dataset or in addition to it. A dataset with a balanced mix between noisy and non-noisy speech data would be preferable, as the model could then likely learn to deal with noise in a better way. However, there is no such dataset. As of today, the best hope of collecting more data for Swedish ASR seems to be through contributing to the Swedish part of Common Voice — anyone can contribute, and the infrastructure to do so is already in place.

Regarding the hyperparameters used to train the best performing general Swedish model, as seen in Table 5.6, they do not differ much from the default parameters of Mozilla's DeepSpeech. The only difference is the use of 30% dropout rate rather than the default 20%. One explanation to this is that extensive hyperparameter tuning has likely been performed by the team behind Mozilla's DeepSpeech, so it is not surprising that the hyperparameters they recommend are good. Another explanation is that the hyperparameter tuning performed as part of this thesis was rather limited, due to time and resource constraints. The true explanation is likely a mix of both. It is also notable that the only hyperparameters considered for tuning in this thesis were the dropout rate, mini-batch size, and learning rate. These three hyperparameters are strictly related to the training process, and do not actually affect the shape or size of the model. A simple explanation for this is that since the model utilises transfer learning it is constrained by the shape and size of the model from which it is fine-tuning on, in this case the official model for English published by Mozilla's DeepSpeech.

6.3 Evaluation results on the NST Dataset

The general Swedish model was primarily evaluated by comparing the WER and CER achieved on the test part of the NST Dataset with the results achieved in previous research and by commercial services. The best WER and CER achieved by the model developed as part of this thesis is 14% and 5% respectively, as shown in Table 5.7. This is an absolute improvement in WER over previous research [3] with 2%, and over Google's Cloud Speech-to-Text with 7%. This could be considered a new state-of-the-art result on the NST Dataset, and shows that the model developed as part of this project is competitive compared to commercial services, there are however some concerns regarding the evaluation methodology.

A major problem encountered when evaluating Google's Cloud Speech-to-Text service on the test part of the NST Dataset was that there was an alphabet mismatch. In particular, Google's Cloud Speech-to-Text service predicted transcriptions containing numbers represented by actual digits, rather than in their normalised word

form (e.g., the number 1 was actually transcribed as *1*, rather than *one*). This is in contrast to how numbers have been treated throughout the rest of the project, and to how numbers are treated in the NST Dataset, where all numbers have been written in their normalised form. As a result of this, the WER and CER initially achieved by this service was 27% and 17%. This did not make for a fair comparison though, as in most cases the numbers were correctly transcribed, but were just presented in an incompatible way. To make for a more fair comparison all predictions were normalised and a better result of 21% WER and 9% CER was achieved. The normalisation did not eliminate all issues though, and an example of an interesting case illustrating this is shown in Table 5.8. Here the service correctly identified and transcribed all numbers, but while doing so it also decided to group some of the numbers together. Because of this decision, the prediction is wrong compared to the ground truth. As the error is due to an incorrect decision by Google's Cloud Text-to-Speech, and not due to an incorrect normalisation, we argue that the results are fair and that the comparison is valid. This claim is further strengthened by the fact that removing all numbers when evaluating did not yield any significant improvement.

Another, although more subtle, problem encountered during evaluation is related to the filtering made on the NST Dataset. This problem is primarily related to the comparison with the result achieved by Kullmann [3]. Because of limitations in DeepSpeech, as mentioned in Section 4.1.1, some files had to be removed from the NST Dataset. This includes removing files from the test part of the NST Dataset. As seen in Table 5.2, 0.12% of the files had to be removed from the test part of the NST Dataset. This means that the data on which we evaluated the general Swedish model might not be the exact same as the data used during the evaluation by Kullmann [3]. Unfortunately, Kullmann [3] does not go into much detail about any processing or filtration made. We do however suspect that some filtration must have been made, since 83 sound files were just silent with a transcription stating that they are in fact silent. This together with the fact that the total fraction of files removed is only 0.12% anyways means our comparison should still be considered fair and usable.

In summary, even though there are some noteworthy things to consider in regards to the evaluation on the NST Dataset, it is fair to state that we created a state-of-the-art ASR model for Swedish and this model presents a new baseline. In doing so, it also fulfills one of the main goals of this thesis.

6.4 Transfer learning for specific domains

One of the main goals of this thesis is to explore if transfer learning could be utilised to get improved results on specialised domains. To analyse this, two domain specific models were created: one in the news broadcasting domain and one in the sports domain. The Lunchekot Dataset was created and used to train the Lunchekot domain model in the news broadcasting domain. The Sports Dataset was created and used to train the Sports domain model in the sports domain. The former

domain can be considered broader and more general in the sense that it uses a wider vocabulary and touches upon a wider range of subjects, while the latter is more narrow and specific to a smaller vocabulary and range of subjects.

Both domain specific models significantly outperforms the general Swedish model on their respective domain specific test set. As can be seen in Table 5.10, the Lunchekot domain model got a best WER of 25% while the best WER achieved by the general Swedish was 36%. Similarly, as seen in Table 5.13, the Sports domain models achieved a WER of 19% while the general Swedish model gets 28%. These results seem to confirm the hypothesis that transfer learning can indeed be used to achieve improved results on specialised domains; however, due to the small size of the domain specific test sets, it is hard to draw too bold conclusions. It is hard to claim that the small test sets are representative of the whole domain, and further analysis on separate datasets is likely required.

Since the domain specific datasets are significantly smaller in size compared to the NST Dataset, the domain specific models could be trained significantly faster than the general Swedish model. While the latter model took approximately 12 hours to train, the domain specific models took less than 30 minutes. Taking this into consideration, the improvements achieved are even more impressive. It means that you can get a significant improvement on specialised domains using few extra resources and time. The fast training time also enabled more extensive and randomised hyperparameter search. The fact that the hyperparameter search was randomised also explains why the hyperparameters shown in Table 5.9 and Table 5.12 are so different from the hyperparameters used to train the general Swedish model and the default hyperparameters recommended by Mozilla's DeepSpeech.

Even though both domain specific models achieve improved results, mistakes are still made. A lot of the mistakes made are reoccurring, and Table 5.11 and Table 5.14 show some interesting examples. Names seem to be an especially common cause of errors, which is not so surprising. In most cases the model gets the name right, but uses an incorrect spelling. Even for a native speaker it is impossible to know how some names are supposed to be written, since spelling can differ without a change in pronunciation. Another common error is missing to add an *s* to the end of a noun, either to indicate possession or to form a compound word. Interestingly enough, as seen in Table 5.11, the compound word *säkerhetsåtgärd* is correctly transcribed, while the very similar *samtalsbehandlingar* is not. What causes these errors is unclear, and further analysis is required — unfortunately, due to time constraints, this analysis could not be made as part of this project. These transcriptions are still easy to understand though, as it is in most cases obvious that there is just an *s* missing. Some of them are however grammatically incorrect, and can not be excused in the same way as the errors related to names were. A native speaker would not make these types of errors. Finally, an interesting observation is that failing to combine two words into a compound word can have a very large impact on the WER, while the effect on the CER is minimal. This is especially clear in the first example of Table 5.11.

6.5 Creating the domain specific datasets

Due to a lack of available Swedish datasets in general, and domain specific datasets in particular, the datasets used to train the domain specific models had to be created as part of this thesis. The lack of domain specific datasets forced us to create the datasets ourselves, as they were essential to explore for accomplishing one of the main goals of the thesis. The lack of datasets might also explain why there is little research on this subject, as creating datasets turned out to be a very time consuming task.

As the domain specific datasets were created manually, there is an obvious risk of human errors affecting the final evaluation. As an example, due to human errors, a transcription in either dataset might be incorrect, meaning that it does not actually reflect what is being said in the sound file. This might result in the model getting an error even though it correctly transcribed what was being said, as the ground truth was wrong. To minimise these errors, we made sure to validate all transcriptions. We are however certain that not all errors were avoided, as this is rarely the case. This problem is not as big of an issue as it might seem though, as we were only interested in the relative improvement between the general Swedish model and the domain specific models. Even though there might have been errors, the evaluation process was the same for the general Swedish model and the domain specific models, making it a fair comparison.

Besides possible incorrect transcriptions, human error might have contributed to wrong annotations in terms of speaker ID and gender. The annotations were also manually created, and are thus prone to error. As an example, most speaker ID annotations could easily be made by listening to the program hosts presenting each speaker, but there were cases when the identity of a speaker could not be determined. For all speakers which could not be identified, a randomised and unique ID was assigned. It might be the case though that one or more of these unidentified speakers were previously correctly identified in some other part, meaning that the same speaker could be represented by multiple speaker IDs. A similar risk of error is present when considering the annotation of gender. The gender of a speaker was annotated purely based on either the name of the speaker, or how the speaker sounded. This is a far from perfect approach, as the name of a person or the sound of a person's voice does not have to be an indication of its gender, and could be considered discriminating. It was, however, the most straightforward approach, and any errors caused by it are negligible.

The purpose of annotating the datasets was to aid in the process of splitting them into training, development, and test sets. However, when creating the Sports Dataset, the speaker ID was not taken into consideration during the split. This design had to be made as we were otherwise unable to create a desirable split of 70% training, 10% development, and 20% test data. The cause of this is the fact that the total Sports Dataset is just 18 minutes in total, with just 29 unique speakers, meaning the degree of freedom to find a good split is very limited. The fact that the same speaker might appear in any of the three subsets might help explain why

there is such a significant improvement when using the 5-grams + NST + Sports language model compared to just the 5-grams + NST model, as seen in Table 5.13. An explanation is that if a certain speaker appears in more than one subset, the specific subject that the speaker is talking about also appears in more than one subset. This means that by simply including the training data in the language model, words or phrases that also might appear in the test set could be captured.

7

Conclusion

Conclusions based on the results and discussion of this thesis are presented in Section 7.1. Based on the conclusions, we highlight some areas that would be interesting for future research, presented in Section 7.2.

7.1 Conclusions

The first clear conclusion that can be made, given the results achieved as part of this thesis, is that transfer learning is a very effective technique in the field of ASR. As part of this thesis, transfer learning was successfully used both to create a model for Swedish, based on an English model, and to create domain specific models within the Swedish language. The results from creating a Swedish model based on an English model seems to confirm the findings by Kunze et al. [10] and Ardila et al. [11], namely that transfer learning can be successfully used for this purpose. This thesis is, to the extent of our knowledge, the first using transfer learning in the field of Swedish ASR. Additionally, the fact that transfer learning could be used so effectively to achieve better ASR in specialised domains should make it an interesting approach when developing products or services in specialised domains or use cases. As previously mentioned though, further analysis and evaluation on separate datasets in the same domain is required before any final conclusion can be made about it.

This thesis additionally highlights the importance of using a language model when developing ASR systems. This seems to be an accepted and widely used theory, but is worth reiterating. Improvements gained when using a language model compared to not using a language model is not the most interesting finding, as this was expected given previous research. Rather, it is worth highlighting the improvements that were gained by simply extending the set of corpora used when creating the language model to include domain specific text. This technique requires no additional training of the actual ASR model, but can improve evaluation results.

As a final remark about the general Swedish model developed as part of this thesis, it is clear that it is not sufficiently good at dealing with speech data containing background noise. This becomes clear when looking at the evaluation results on Common Voice, and the worse results on the Lunchekot Dataset and Sports Dataset compared to the evaluation result on NST Test. We believe that if there was significantly more transcribed Swedish speech data available, the results could be improved. However,

there is an obvious lack of available training data for Swedish ASR. The best available dataset seems to be the NST Dataset, at around 500 hours of transcribed speech, but this dataset is 20 years old. Projects like Common Voice could be a solution in the long run but as of now only contains five hours of validated data for Swedish. Unfortunately, the domain specific datasets developed as part of this thesis could not be published due to licensing issues with Sveriges Radio.

7.2 Future work

Below are some suggestions to topics that could prove valuable if researched further.

Artificial datasets

As previously stated, there is a lack of training data for Swedish ASR. The process of creating a dataset suitable for training an ASR model is however time consuming. A potential solution which does not involve manually transcribing more speech data is to artificially and automatically create more data. It has been shown that simply augmenting existing data can effectively improve results [32]. By using this approach, one could get around the problem with lacking data by augmenting data from the NST Dataset. An example which could perhaps improve evaluation results on speech with background noise would be to artificially insert background noise in the training data of the NST Dataset.

Alternative language models

In this thesis, the KenLM language model [14] was used. KenLM was first introduced by Heafield [14] in 2011, but since then a lot has happened in the field of language modeling. As an example, the BERT language model which was recently published has achieved state-of-the-art results [9]. In a recent paper by Shin et al. [15], BERT was successfully used in the context of ASR. Perhaps by using BERT, or other alternative language models, even better results could be achieved than those presented as part of this thesis.

Improved corpora normalisation

There are some errors with the way Swedish text corpora were normalised as part of this thesis. As previously discussed, there does not seem to exist any tools for performing the needed normalisation of Swedish text. If such a tool, like the MaryTTS tool previously highlighted, would exist for Swedish it is probable that results could be improved further. Such a tool could also be valuable in the field of speech synthesis [55].

Bibliography

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

[2] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, *et al.*, "The Kaldi speech recognition toolkit", in *IEEE 2011 workshop on automatic speech recognition and understanding*, IEEE Signal Processing Society, 2011.

[3] E. Kullmann, "Speech to Text for Swedish using KALDI", Master's thesis, KTH, Optimization and Systems Theory, 2016.

[4] Z. Mossberg, "Achieving Automatic Speech Recognition for Swedish using the Kaldi toolkit", Master's thesis, KTH, School of Computer Science and Communication (CSC), 2016.

[5] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, *Deep Speech: Scaling up end-to-end speech recognition*, 2014. arXiv: `1412.5567 [cs.CL]`.

[6] R. Collobert, C. Puhrsch, and G. Synnaeve, "Wav2letter: An end-to-end convnet-based speech recognition system", *arXiv preprint arXiv:1609.03193*, 2016.

[7] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, *et al.*, "State-of-the-art speech recognition with sequence-to-sequence models", in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2018, pp. 4774–4778.

[8] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang, *et al.*, "Streaming End-to-end Speech Recognition For Mobile Devices", in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2019, pp. 6381–6385.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", *arXiv preprint arXiv:1810.04805*, 2018.

[10] J. Kunze, L. Kirsch, I. Kurenkov, A. Krug, J. Johannsmeier, and S. Stober, "Transfer Learning for Speech Recognition on a Budget", in *Proceedings of the 2nd Workshop on Representation Learning for NLP*, Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 168–177.

DOI: 10.18653/v1/W17-2620. [Online]. Available: https://www.aclweb.org/anthology/W17-2620.

[11] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber, "Common Voice: A Massively-Multilingual Speech Corpus", *arXiv preprint arXiv:1912.06670*, 2019.

[12] S. Bansal, H. Kamper, K. Livescu, A. Lopez, and S. Goldwater, "Pre-training on high-resource speech recognition improves low-resource speech-to-text translation", *arXiv preprint arXiv:1809.01431*, 2018.

[13] V. Pratap, A. Hannun, Q. Xu, J. Cai, J. Kahn, G. Synnaeve, V. Liptchinsky, and R. Collobert, "Wav2letter++: The fastest open-source speech recognition system", *CoRR*, vol. abs/1812.07625, 2018. [Online]. Available: https://arxiv.org/abs/1812.07625.

[14] K. Heafield, "KenLM: Faster and smaller language model queries", in *Proceedings of the sixth workshop on statistical machine translation*, Association for Computational Linguistics, 2011, pp. 187–197.

[15] J. Shin, Y. Lee, and K. Jung, "Effective sentence scoring method using BERT for speech recognition", in *Asian Conference on Machine Learning*, 2019, pp. 1081–1093.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

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

[18] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks", in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 369–376.

[19] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization", in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 661–670.

[20] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan, "Better mini-batch algorithms via accelerated gradient methods", in *Advances in neural information processing systems*, 2011, pp. 1647–1655.

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

[22] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout", in *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, 2013, pp. 8609–8613.

[23] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM", 1999.

[24] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation", *arXiv preprint arXiv:1406.1078*, 2014.

[25] M. Huh, P. Agrawal, and A. A. Efros, "What makes ImageNet good for transfer learning?", *arXiv preprint arXiv:1608.08614*, 2016.

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

[27] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris, "SpotTune: transfer learning through adaptive fine-tuning", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4805–4814.

[28] S. J. Pan and Q. Yang, "A survey on transfer learning", *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

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

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

[31] A. Agarwal and T. Zesch, "German End-to-end Speech Recognition based on DeepSpeech", in *Preliminary proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019): Long Papers*, Erlangen, Germany: German Society for Computational Linguistics & Language Technology, 2019, pp. 111–119.

[32] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition", *arXiv preprint arXiv:1904.08779*, 2019.

[33] D. Yu and L. Deng, "Automatic Speech Recognition", in. Springer, 2016, pp. 1–9.

[34] M. Epstein and J. Marozeau, "Loudness and intenstity coding", *The Oxford Handbook of Auditory Science–Hearing*, vol. 3, pp. 45–69, 2010.

[35] J. W. Picone, "Signal modeling techniques in speech recognition", *Proceedings of the IEEE*, vol. 81, no. 9, pp. 1215–1247, 1993.

[36] L. R. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer, "A tree-based statistical language model for natural language speech recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 7, pp. 1001–1008, 1989.

[37] G. Synnaeve, Q. Xu, J. Kahn, E. Grave, T. Likhomanenko, V. Pratap, A. Sriram, V. Liptchinsky, and R. Collobert, "End-to-end ASR: From Supervised to Semi-Supervised Learning with Modern Architectures", *arXiv preprint arXiv:1911.08460*, 2019.

[38] J. Kahn, A. Lee, and A. Hannun, "Self-training for end-to-end speech recognition", *arXiv preprint arXiv:1909.09116*, 2019.

[39] A. Stolcke, "SRILM-an extensible language modeling toolkit", in *Seventh international conference on spoken language processing*, 2002.

[40] M. Federico, N. Bertoldi, and M. Cettolo, "IRSTLM: An open source toolkit for handling large scale language models", in *Ninth Annual Conference of the International Speech Communication Association*, 2008.

[41] J. Meyer, "Multi-Task and Transfer Learning in Low-Resource Speech Recognition", PhD dissertation, 2019.

[42] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups", *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[43] C. Gaida, P. Lange, R. Petrick, P. Proba, A. Malatawy, and D. Suendermann-Oeft, "Comparing open-source speech recognition toolkits", in *11th International Workshop on Natural Language Processing and Cognitive Science*, 2014.

[44] B. Milde and A. Köhn, "Open source automatic speech recognition for German", in *Speech Communication; 13th ITG-Symposium*, VDE, 2018, pp. 1–5.

[45] M. Stadtschnitzer, J. Schwenninger, D. Stein, and J. Köhler, "Exploiting the large-scale German Broadcast Corpus to boost the Fraunhofer IAIS Speech Recognition System.", in *LREC*, 2014, pp. 3887–3890.

[46] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines", in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[47] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.

[48] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books", in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2015, pp. 5206–5210.

[49] B. Favre, K. Cheung, S. Kazemian, A. Lee, Y. Liu, C. Munteanu, A. Nenkova, D. Ochei, G. Penn, S. Tratz, *et al.*, "Automatic human utility evaluation of ASR systems: Does WER really predict performance?", in *INTERSPEECH*, 2013, pp. 3463–3467.

[50] T. Kohonen, "Self-organization and associative memory", in. Springer Science & Business Media, 2012, vol. 8, p. 66.

[51] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals", in *Soviet physics doklady*, vol. 10, 1966, pp. 707–710.

[52] W. Wu, R. J. May, H. R. Maier, and G. C. Dandy, "A benchmarking approach for comparing data splitting methods for modeling water resources parameters using artificial neural networks", *Water Resources Research*, vol. 49, no. 11, pp. 7598–7614, 2013.

[53] Z. Reitermanova, "Data splitting", in *WDS*, vol. 10, 2010, pp. 31–36.

[54] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization", *Journal of machine learning research*, vol. 13, no. Feb, pp. 281–305, 2012.

[55] M. Gaved, "Pronunciation and Text Normalisation in Applied Text-to-Speech Systems", in *Third European Conference on Speech Communication and Technology*, 1993.

A

Files removed from the NST Dataset

The NST Dataset (NST Training and NST Test combined) consists of 380617 files with corresponding transcriptions. Out of these, 10197 files had to be removed as a result of the filtering described in Section 4.1.1. There were 1082 files that were just silent, and were removed because of this. These were the files that had the transcription *tyst under denna inspelning* (which is Swedish for *silent during this recording*), as described in Section 4.1.1. A large number of files, 9097, were removed from the training part as they were longer than or equal to 10 seconds, which is a limitation in DeepSpeech during training. The remaining 18 files are described in Table A.1 and Table A.2. The file names in these tables have the prefixes *train* and *test*, which means that they refer to a file in the training or test part of the NST Dataset respectively. The number of files together with the reason for removal of these files are displayed in Table A.3, Table A.4 and Table 5.2 for the NST Dataset as a whole, NST Training and NST Test respectively.

Table A.1 shows files that were removed because their duration was either too long or too short to match their transcriptions. Mozilla's DeepSpeech contains a check for this, and trying to use a file which does not pass the check makes Mozilla's DeepSpeech crash with an error.

In Table A.2, sound files which were removed because of their transcriptions are shown. In this case, the transcriptions included characters not part of the defined alphabet, see Section 4.1.1.

Table A.3 and Table A.4 describes the reasons why files got removed from the NST Dataset as a whole and from NST Training respectively. These tables shows that 2.68% and 3.29% of all files were removed from the NST Dataset as a whole and NST Training respectively. Table 5.2 shows the same data for NST Test, from this dataset 0.12% of the files have been removed.

Table A.1: Sound files removed from the NST Dataset due to being too long or too short for their transcriptions.

File name
train/Stasjon7/100799/adb_0467/speech/scr0467/07/04670706/r4670580/u0580189.wav
train/Stasjon7/100899/adb_0467/speech/scr0467/07/04670707/r4670672/u0672205.wav
train/Stasjon7/210799/adb_0467/speech/scr0467/07/04670707/r4670619/u0619087.wav
train/Stasjon7/160799/adb_0467/speech/scr0467/07/04670706/r4670598/u0598102.wav
train/Stasjon2/120799/adb_0467/speech/scr0467/02/04670202/r4670123/u0123069.wav
train/Stasjon5/220799/adb_0467/speech/scr0467/05/04670505/r4670441/u0441079.wav
train/Stasjon6/270799/adb_0467/speech/scr0467/06/04670606/r4670536/u0536161.wav
train/Stasjon20/191099/adb_0467_2/speech/scr0467/20/04672001/r4670086/u0086079.wav
train/Stasjon3/120799/adb_0467/speech/scr0467/03/04670303/r4670225/u0225002.wav
test/Stasjon20/110100_del2/adb_0468/speech/scr0468/20/04682001/r4680023/u0023342.wav
test/Stasjon20/191299/adb_0468/speech/scr0468/20/04682001/r4680013/u0013872.wav

Table A.2: Sound files removed from the NST Dataset as their transcriptions contained unwanted characters.

File name
train/Stasjon1/020899/adb_0467/speech/scr0467/01/04670101/r4670077/u0077089.wav
train/Stasjon1/020899/adb_0467/speech/scr0467/01/04670101/r4670076/u0076121.wav
train/Stasjon6/290799/adb_0467/speech/scr0467/06/04670606/r4670544/u0544107.wav
train/Stasjon6/190799/adb_0467/speech/scr0467/06/04670606/r4670519/u0519118.wav
train/Stasjon3/210799/adb_0467/speech/scr0467/03/04670303/r4670249/u0249117.wav
test/Stasjon20/160100/adb_0468/speech/scr0468/20/04682001/r4680028/u0028784.wav
test/Stasjon20/060100/adb_0468/speech/scr0468/20/04682001/r4680019/u0019397.wav

Table A.3: The number of files removed from the NST Dataset as a whole, grouped by the reason for filtration.

Reason	**Count**	**Part of total (%)**
Duration too long	9097	2.39
Silent recording	1082	0.28
Duration not matching transcription	11	< 0.1
Unwanted character	7	< 0.1
Total removed	10197	2.68
Total	380615	100

Table A.4: The number of files removed from NST Training, grouped by the reason for filtration.

Reason	**Count**	**Part of total (%)**
Duration too long	9097	2.96
Silent recording	999	0.32
Duration not matching transcription	9	< 0.1
Unwanted character	5	< 0.1
Total removed	10110	3.29
Total	307568	100

B

Acronyms

AGD	Accelerated Stochastic Gradient Descent
AM	Acoustic Model
ANN	Artificial Neural Network
ASR	Automatic Speech Recognition
BERT	Bidirectional Encoder Representations from Transformers
CER	Character Error Rate
CSV	Comma-separated Values
CTC	Connectionist Temporal Classification
DAG	Directed Acyclic Graph
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
ID	Identification
LM	Language Model
LPCM	Linear Pulse-Code Modulation
LSTM	Long Short-Term Memory
MFCC	Mel Frequency Cepstral Coefficients
NLP	Natural Language Processing
NST	Norsk Språkteknologi
RNN	Recurrent Neural Network
ReLU	Rectified Linear Unit
TTS	Text-to-Speech
WER	Word Error Rate
XML	Extensible Markup Language