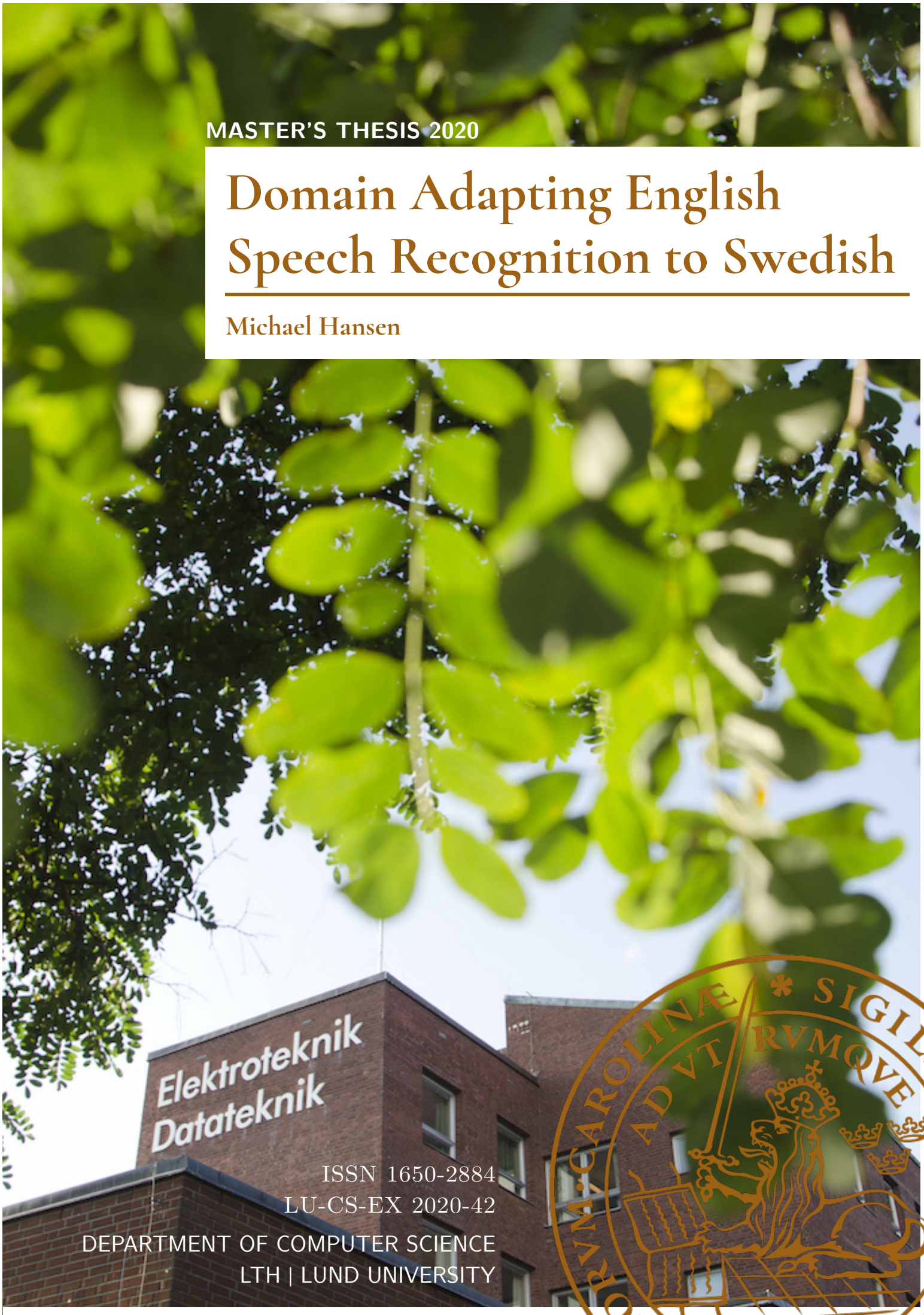


MASTER'S THESIS 2020

Domain Adapting English Speech Recognition to Swedish

Michael Hansen

ISSN 1650-2884
LU-CS-EX 2020-42

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2020-42

Domain Adapting English Speech Recognition to Swedish

Michael Hansen

Domain Adapting English Speech Recognition to Swedish

Michael Hansen
dat14mha@student.lu.se

July 1, 2020

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Pierre Nugues, pierre.nugues@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

Automatic speech recognition is a technology that enables a machine to transcribe human speech into text. Keyword spotting is a subfield of it, which restricts the problem to the identification of a few selected words.

Most current recognition algorithms require huge corpora of annotated spoken data. They result in large machine learning models that often require dedicated servers to process the audio input at a responsive speed. When reducing the vocabulary to a finite amount of key words, the model can instead be stored locally on mobile devices, eliminating the need for network connection. Nonetheless, machine learning models, even for keywords, still need a lot of data.

In 2017, Google published a dataset of ~106,000 English audio samples corresponding to a vocabulary of 35 words. Using such resources enabled researchers to make rapid progress and reach a very high recognition accuracy. Such speech corpora are expensive, however, as they take a long time to both collect and annotate. Smaller languages, like Swedish, have no such resources, restricting the advances in the field and lowering the recognition accuracy. In addition, they make the research rely on large private companies for data collection.

In this project, I developed a system to use the English dataset to train a speech recognition model, and then adapting this model to a smaller, Swedish dataset, which I collected myself. I could reach an accuracy of ~86% with a dataset 100 times smaller than the original one. This figure is to compare with the 51% accuracy of a baseline model trained without the English dataset.

Domain adaption has been used in other fields of machine learning, and this projects proves that its results are beneficial to speech recognition as well.

With this system, specific applications can be feasibly constructed without the need for a large dataset, making it easier to make mobile systems responsive and verbally interactive. This project is a starting point to use domain adaption to retrain models for new languages, and it would be interesting to see how well it works on languages from different families.

Keywords: deep residual learning, domain adaption, keyword spotting, speech recognition, machine learning, transfer learning, data collection

Acknowledgements

This project has been a completely new experience, from researching related papers, to collecting a dataset, and writing a comprehensive thesis paper. It would not have been nearly as easy without the help of my mentor Pierre Nugues. He helped guide the work and provide helpful feedback, both when I steered too far off the current goal and when I had a difficult time advancing.

I also want to thank everyone lending me their voice for the dataset. I appreciate the dedication put in to make each recording different. Thank you.

A big part of this project was carried out with the help of a dedicated computer borrowed to me by the institution of computer science at *Lunds tekniska högskola*. A lot of model training hours went smoothly thanks to that.

Contents

Chapter 1
Introduction

Speech recognition is a field of intense research, where machine learning improved greatly the recognition quality. Corporate giants, such as Google and Apple, incorporate it into their services and are at the forefront of research. This makes the field progress at an ever-quickening pace. The sophisticated models they apply, which can transcribe arbitrary sentences from utterances alone, require dedicated servers to function at a reasonable speed. This in turn requires the user to be connected to the internet in order to use them on their phones. Such pipelines come with security risks for the end user, who is required to be connected online and send potentially personal data to the servers.

A sub-field of speech recognition, called *keyword spotting* (KWS), heavily reduces the requirements of the model by requiring it to only be able to detect a limited amount of words. This limited scope allows the software to be simplified to the level that it can feasibly be run on *edge devices* such as smart phones or other *internet-of-things* (IoT) devices.

This field has also been explored to a great extent, from data pre-processing to modelling new *neural network* (NN) structures to improve accuracy and reduce the models' size. One such NN structure is *residual networks* (Tang and Lin, 2018), resNets in short.

Deep residual learning is a neural network structure inspired by *hidden Markov models*, in that it retains information from its previous state which is unprocessed, along with the processed state. The NN technique has allowed for deeper networks to be feasible. One common way to represent audio is with an image, for example a mel spectrogram.

Audio can be represented by its frequencies over time with a spectrogram. When representing human speech, and its small changes, the frequency spectrum can be converted to the mel scale to make these changes more noticeable (see Figure 2.1). The *mel-frequency cepstral coefficients* (MFCC) form another, more compact, way to represent these frequencies.

Converting the one-dimensional audio wave to a compact 2D matrix allows for the usage of *convolutional neural networks* (CNNs), which inherently connect the data points as a bigger picture, which is useful for both images and speech (Sainath and Parada, 2015; Lin et al., 2018).

In this thesis work, I describe the re-implementation of a previous resNet model structure

for KWS of English words and the adaptation of this model to Swedish using a smaller dataset. This includes the following steps:

1. Collection of a spoken dataset;
2. Construction of a resNet model in Keras;
3. Data preprocessing;
4. Adaption of the model to the domain; and finally
5. Evaluation of the model's performance.

Chapter 2
Previous Work

2.1 Google's dataset

In 2017, Google released a dataset of spoken command words in English (Warden, 2017). The dataset contained ~65,000 recordings spanning 30 words, where 10 of these words were the words to recognize and the remaining 20 were considered noise words. Google later complemented the dataset with more words and samples. The updated version of the dataset contains ~106,000 samples spanning 35 words.

The Google speech command dataset is made up of one second long 16bit WAV audio files of common commands for voice control spoken by a wide variety of English speaking people. The complete list of words is: *yes*, *no*, *up*, *down*, *left*, *right*, *on*, *off*, *stop*, and *go*.

Google also provided *noise words* which the model may hear, but should ignore. These words are labeled *unknown*. These words are: *backward*, *bed*, *bird*, *cat*, *dog*, *follow*, *forward*, *happy*, *house*, *learn*, *Marvin*, *Sheila*, *tree*, *visual*, *wow*, *zero*, *one*, *two*, *three*, *four*, *five*, *six*, *seven*, *eight*, and *nine*. See Table 2.1 for the word count in the dataset.

Word	yes	no	up	down	left	right	on	off	stop	go
Count	4044	3941	3723	3917	3801	3778	3845	3745	3872	3880
Word	zero	one	two	three	four	five	six	seven	eight	nine
Count	4052	3890	3880	3727	3728	4052	3860	3998	3787	3934
Word	forward	backward	bed	bird	cat	dog	follow	happy	house	learn
Count	1557	1664	2014	2064	2031	2128	1579	2054	2113	1575
Word	Marvin	Sheila	tree	visual	wow					
Count	2100	2022	1759	1592	2123					

Table 2.1: The size of the Google dataset (v2). The words in the first row relate to keywords, and the following to the noise words.

Google's dataset was recorded in authentic settings with a wide variety of native English

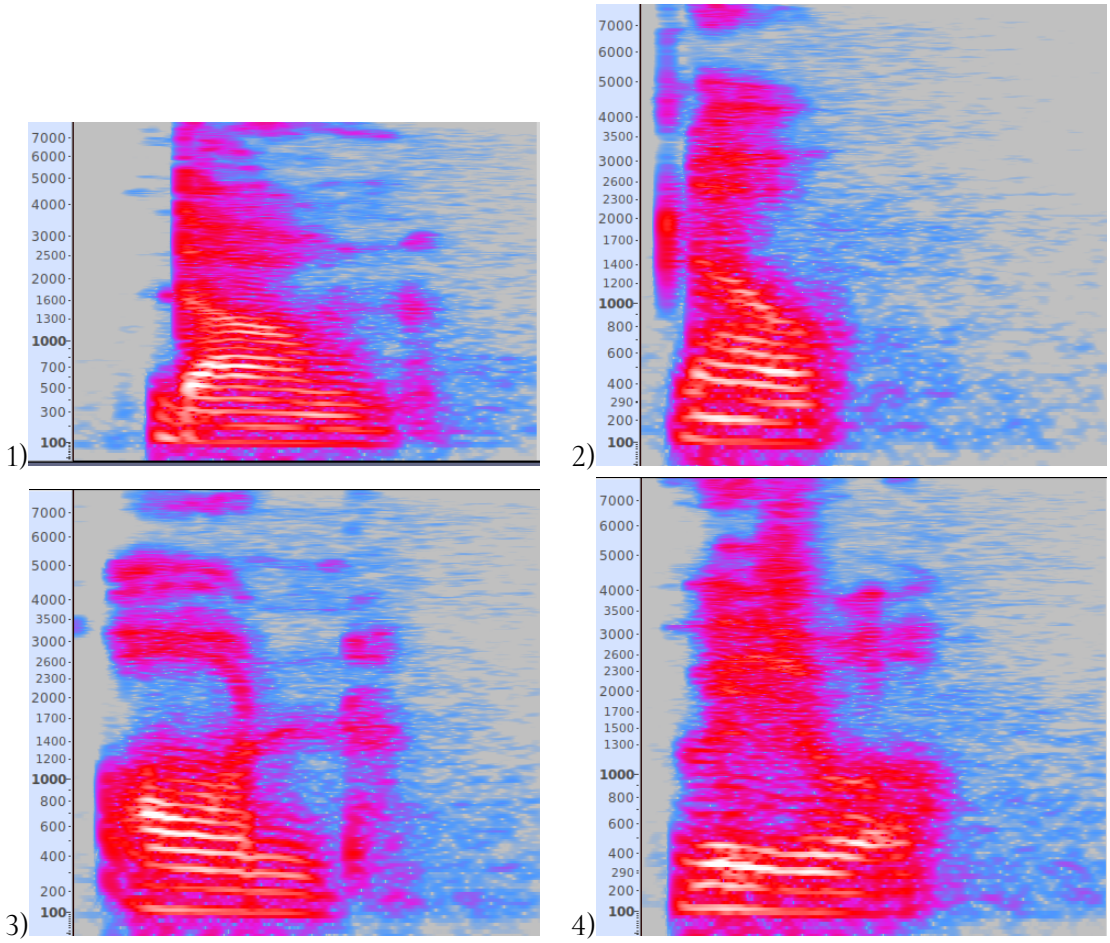


Figure 2.1: Mel spectrograms of the words down (1), no (2), on (3), visual (4).

speakers. Some recordings contained background noise and varying audio levels. Google also recorded samples of such background noise and provided them in the dataset so that it could be applied on custom audio files too, simulating a real life use case environment.

Together with the dataset the team included different lists of the dataset put in text files, which would separate the speakers across training, testing, and validation, so that one speaker would not be multiple groups, which would potentially cause an over-fit.

If a model is over-fitting while training, it means it does not train to generalise what it is seeing, but it learns to recognise exactly the data it is shown. This will yield in poor results when the model is shown new data samples.

In this thesis, I chose to have Swedish keywords and noise words reflecting the ones above in English. I could have selected them differently, for example, with numbers instead. However, this choice should not have consequences on the performance results and the conclusion of this work.

2.2 Neural Networks

Along with this dataset, Google provided programs to process the data, train a model, and test it. They also shared a program for a basic phone application, which uses the model. This app transcribes your speech and, through an interface showing the ten possible words, highlights the word that the model recognizes.

Google's model architecture consist of a neural network with convolutional layers along with matrix multiplications in a straightforward Tensorflow model. This model take a one second clip of the audio signal as input. The description of the structure can be found in the dataset paper by Warden (2017).

Neural networks, or NNs for short, are very popular machine learning structures. They take input data of a certain shape and return an output that of a certain shape which complies with what the network has learned. NN have gained a lot of traction in recent years and their performance depends on how the network and training data is structured.

In this thesis, the input is a 2D image, the MFCC of the audio (described below), which is fed into the network, and the output is one of 12 classes (the keywords and the noise).

NNs are built up of layers of *nodes* of different types, which hold *weights* with values. These weights are part of what is adjusted when the neural network is trained.

One type of node, which are used for processing images, are *convolutional neural nets* (CNNs). These utilise a so called kernel, normally of sizes 3-by-3 or 5-by-5 pixels, which it uses to apply convolutions to the input images. These types of network nodes are useful because they look at a certain pixel's neighbours in 2D space, which is where the relevance in images lie.

Connections run between these nodes and connect the node layers in different arrangements. The nodes can also be trained depending on their significance to the input-output relation.

In order to see how far off the correct answer a model is, they often include a loss function. Loss functions, and their value, represent how wrong the model prediction was.

This loss function is also used to optimise the network, using an optimiser. One example of such an optimiser is the *Adam* optimiser. Adam stands for Adaptive moment estimation, and was developed by (Kingma and Ba, 2015). It uses the loss function's past gradients to calculate its current ones. It also utilises the concept of momentum to not stop at a very local minimum.

There are several base node structures which can be used for a lot of applications. Some applications, however, require a more specific node structure – either to more effectively learn a task or to decrease the complexity or size of a NN.

One such special node structure are residual nodes.

2.3 Deep residual learning

The deeper a neural network is, the more it abstracts the data, which in turn often gives better results. However, the deeper a network is built, the more difficult it is to train. He et al. (2015) showed that using residual learning makes training deeper networks easier and more effective.

Residual networks are made up of smaller network blocks, where one connection passes through nodes, processing the input and applying weights and biases while the other connection shortcuts the processing. The processed path and the skipping identity path are then joined at the end of the block (see Figure 2.2 for example).

Tang and Lin (2018) later adapted his structure to build small models for keyword spotting effectively using Google's speech command dataset. Using the resNet structure, they managed to reach an even higher evaluation accuracy than the Google baseline.

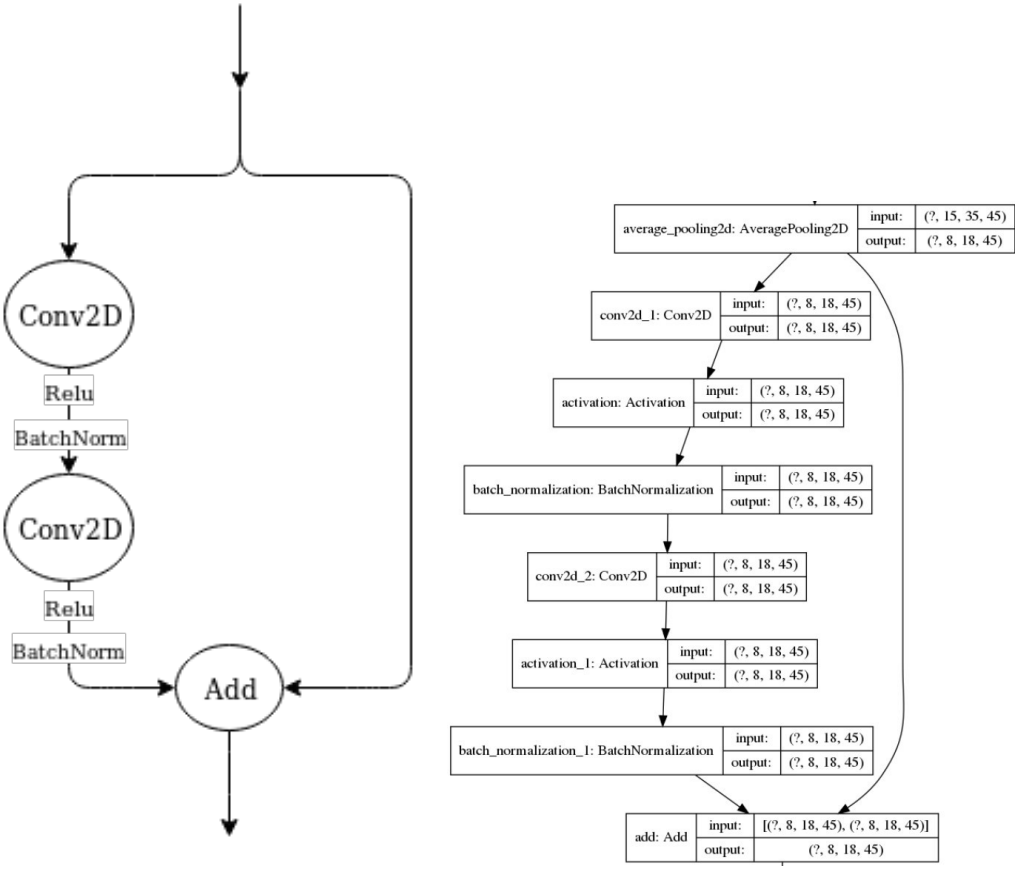


Figure 2.2: Example of a residual network node group. The left part shows the conceptual view of the structure; the right part shows the first resnet node block in the actual Keras model.

2.4 EdgeSpeechNets

Lin et al. (2018) aimed to build optimised models with regards to different parameters, such as the number of nodes in the model or accuracy. This was a way to build specialised IoT device models for different tasks. They called this model: *EdgeSpeechNets*

Lin et al. (2018) trained their models using Google's dataset as well as a resNet neural net model structure. Their results surpassed the previously documented results while also reducing the computational requirements and size of the network.

The EdgeSpeechNets utilised a generator function to develop optimised neural networks using convolutional layers, but that part is out of the scope of this project.

As a performance baseline, Lin et al. (2018) used a model described by Tang and Lin (2018).

In my thesis, I will use the resNet model structure for my models, namely the one named 'res26' in the paper by Tang and Lin (2018).

2.5 Data Representation

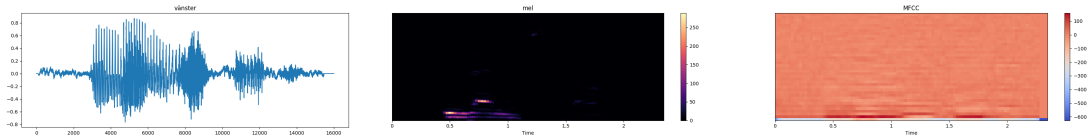


Figure 2.3: [left to right] Amplitude, mel spectrogram and MFCC representation of the Swedish word *vänster* 'left'.

We can visualize audio in many ways. Aside from showing the signal *amplitude* as a function over time, see Fig. 2.3, it can be shown as a 2D matrix, usually as an image, showing frequency energies over time. This representation is called a *spectrogram*.

Changes in human speech are usually in a relatively short span on normal spectrograms (which covers multiple kHz). When working with voice recordings, it is more beneficial to visualise it in a *mel spectrogram* (see Fig. 2.3), where the distance between frequencies is more in line with what humans perceive them to be. The mel scale better displays the frequency range where speech and vocal changes occur. Eq. 2.1 shows the function to convert frequency f to mel scale:

$$M(f) = 1125 \cdot \ln(1 + \frac{f}{700}) \tag{2.1}$$

A mel-frequency cepstrum (MFC) is a way to represent sound by its mel scale power spectrum. The power spectrum can be derived by windowing short time frames of the signal, and calculating the windows' respective frequency power distributions. As an example, the window size used in this paper is 30ms and the time hop between each frame sampling is 10ms.

An MFC is made up of *mel-frequency cepstral coefficients* (MFCCs) and since it is derived from a mel spectrogram, it is useful for speech feature extraction. It is also compact because the window framing joins 480 samples (in the case of a 16kHz sample rate and 30ms window size) into one column.

2.6 Data Augmentation

A way to cope with small datasets is to augment them, making the number of unique samples near never-ending. This is done by changing aspects or details in the picture to the degree

that it's a brand new sample, while still maintaining the structure of the object in the sample intact, be that a picture of a dog or an utterance of the word *left*.

Keras preprocessing, a sub-module of Tensorflow, has a generator class, which augments the image samples in a dataset and feeds the output to train the model. Something similar to this, but for audio, will be required to make the most of the small dataset this paper uses.

Park et al. (2019) and Ko et al. (2018) describe useful parameters for this augmentation. Quantities such as the pitch, phoneme dilation, and time shifts are examples of variables that the augmentation can adjust. The parameters are explored further below.

2.7 Augmenting Speech Data

The speech representation suggests analogies with images to improve or extend the dataset. Speech data is processed as 2D matrices, which could be interpreted as images. This indicates that transformations like the ones used on images for image classification could be used to augment the spectrograms.

When thinking of variance on spoken words, rotation and scale may not mean that much. Dilation, intensity, and translation are more realistic variables. A syllable can be stretched and shrunken, and the amplitude of the speaker can be higher and lower as well as shifted earlier or later within the given window.

So to make the model independent of the audio level and tempo of the speaker, this was an aspect of the data augmentation to consider. Ko et al. (2018) and Park et al. (2019) confirmed these augmentation ideas. Outside of these, the standard time offset and noise application, which were present in all of the keyword spotting papers, are also used in the augmentation.

Ko et al. (2018) discussed extending datasets by augmenting the data with one fixed parameter of the following with a certain magnitude. Their data is processed as MFCCs. They write about speeding up the recording, *speed perturbation*, affecting both duration and pitch. They also test adjusting audio tempo, *tempo perturbation*, which emulated how quickly something is spoken without affecting the pitch of the speech. They finally try the effectiveness of VTLP, *vocal tract length perturbation*, which is warping part of the audio after it has been normalised through *vocal tract length normalization*.

Park et al. (2019) augmented data directly on the input features, which in their case are mel spectrograms. They applied a random magnitude of each augmentation. One such augmentation is time warping. The two other augmentations are time and frequency masking, where they blank out parts of the input spectrogram.

Chapter 3
Architecture

3.1 Implementing the Model

The first step of the project was to reproduce the model used in EdgeSpeechNets (Lin et al., 2018) using the residual network structure (see Fig. 3.1) coming from the papers by Tang and Lin (2017, 2018). I did this with *Keras*, a high-level machine learning API in Python built on Tensorflow.

This model takes as input the *mel frequency cepstrum coefficients* (MFCC) of one second of audio. The frame size used for the MFCC window function groups every 30ms together, and that window jumps 10ms fowards for each grouping. This means there will be an overlap of 20ms, which is 67%, between neighbouring groupings. Before applying the calculations, the preprocessing restricts the frequency range to 20-4000 Hz, following the preprocessing in Lin et al. (2018). I centered and padded the frames with the Python module *librosa*. The shape of the resulting MFCC is $40 \times 101 \times 1$ (reshaped to work as an image).

The model trained on the MFCCs created using the librosa Python module was difficult to deploy in the Android app, which is built using Java. This was because the Java implementation of librosa's MFCC code, from GitHub (Fu, 2020), does not generate results identical to the Python code.

The first difference was that the Python code used floats with 32-bit precision (except for complex values which used 64 bits), while the original Java implementation used doubles, which use a 64-bit precision. The Python code had some additional zero value solutions which were not present in the Java code, as well as the window function for the short-time Fourier transform differed from the Python implementation in regards to zero padding and size. The Java code's method comment also stated to use *discrete cosine transform* (DCT) type 3, while the default type in Python was type 2.

To bypass those differences, I chose the Java code as the only preprocessing code for both the model training and the Android app. To access it in the Python code, I created a pipeline using the *py4j* library. Py4j works by creating a local server through which the Java code is

accessible for Python code.

This spectrum serves as input to the neural net, shown in Figures 3.1 and 3.2, output to a softmax function with 12 labels:

- The keywords (10),
- A label for background noise (labeled silence) and
- One for the noise words (labeled background_noise).

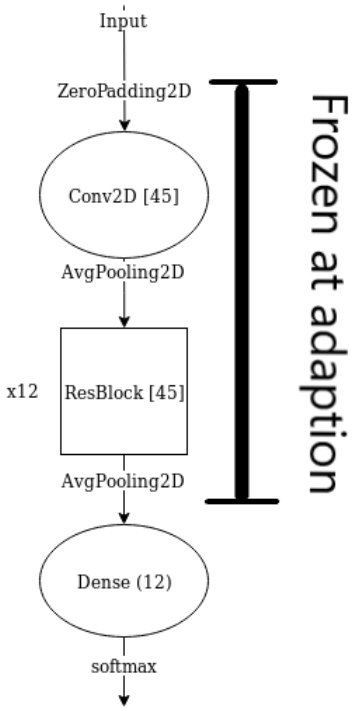


Figure 3.1: A diagram of the Keras model built.

With Fig. 3.1 as reference, the model structure can be split up into the start, with a zero padding layer, a convolutional layer and an average pooling layer, followed by the middle consisting of the 12 ResNet blocks and ending with the average pooling and dense layer.

When adapting the model to Swedish and re-training it, all layers up to the ending layers were frozen. Only the head was re-trained. See Figure 3.1.

3.2 Mobile apps

3.2.1 Speech Command App

As a way to help software development and research, Google has released a speech command Android application which uses a Tensorflow Lite model trained on their dataset, along with its code.

Before we can deploy a model onto a mobile device, we need to be convert it to a Tensorflow Lite model. There are functions to generate such models within the Python module:

`Tensorflow.Lite.TFLiteConverter`

It has functions to convert a Keras model which is what is used in this project.

The application consists of a grid of the 10 keywords, text for the sampling frequency, and how many threads Tensorflow is using for data inference. The microphone is continuously listening for speech and, when one of the keywords is identified, its corresponding label is highlighted along with the probability of this label being correct. The word box is highlighted for one second and then returns to the original layout.

Under the hood, the recording part of the app has its own thread, as does the recognition part of it.

3.2.2 Swedish Model Test App

I modified the source code to this application example and I used it with the Swedish model to test the model's usability.

I updated the label grid so it contained the Swedish labels instead of the English. Along with this, I edited the part of the functionality which handles the label highlighting so it linked properly to the new Swedish label indices. The pre-existing preprocessing in the code was to normalise the recorded signal before feeding it into the model. I rerwrote this part of the code to use the new MFCC preprocessing code.

To this base app's functionality, I added some debugging features described in the result chapter about it, and optimised the heavier-than-before preprocessing step to its own thread (seen with the "Translation Time" value in the app).

3.3 AudioDataGenerator

I made two versions of the Generator class:

1. One which returns a Python generator that operates similarly to Keras' `ImageDataGenerator` class and

2. A version which implements Tensorflow's `Sequence` class.

The two classes differ in the setup. The first version only needs one instance and then creates generators when one of its methods is called. The other is more compact to use, but needs a new class object for every generator (train / validation / testing). After writing the Sequence implementation of the generator, I moved to using that over the previous iteration, because it was easier to use. I simply provided a list of file paths, along with configurations such as augmentation and subset type, when creating the object. The code also felt more structured., since when making it I had the first generator class to compare to.

Table 3.1 shows a list of the augmentation parameters. These parameters are restricted because after a certain point, the original audio is either too obscured to be audible or too shifted to be realistic (when listening to pitch for example). Some restrictions were found in papers (cited); others were chosen based on how they sounded. Within the ranges the values were chosen uniformly.

Parameter	Range	Default value
Pitch	([-1, 1), (-1, 1])	(1., 1.)
Time stretch	([0, 1], [0, 1])	(1., 1.)
Time shift (100ms)	{True, False}	False
Time mask	{None, [0.0, 0.5]}	None
Noise probability	[0.0, 1.0]	0.80
Percentage extra background noise	[0.0, 1.0]	0.01

Table 3.1: The different augmentation parameters in the generators. The methods are using the Librosa Python library. Random values are chosen uniformly.

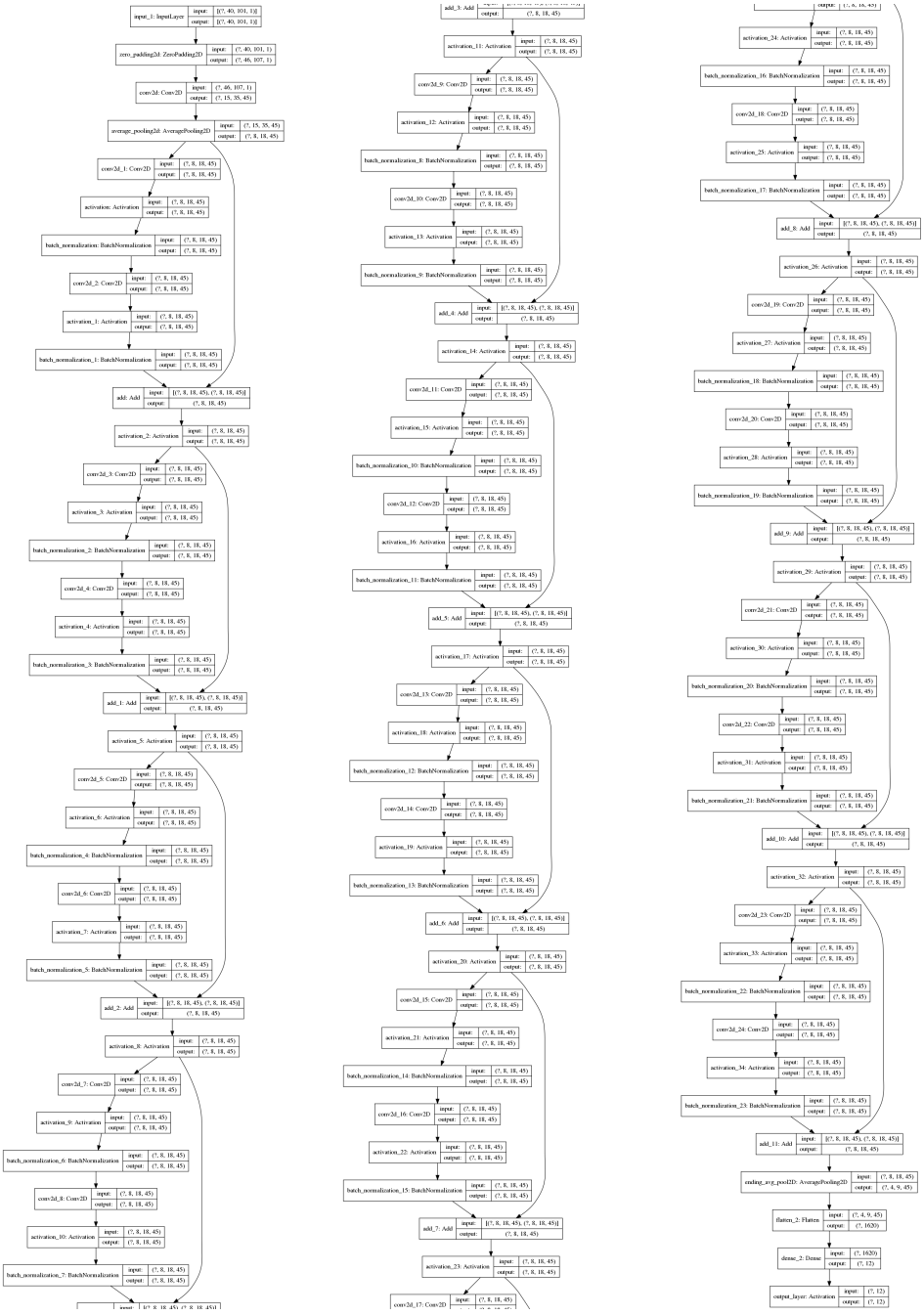


Figure 3.2: Visualization of the neural net structure. Split pieces goes left to right, up to down. Each cluster is the same as what is shown in Fig. 2.2.

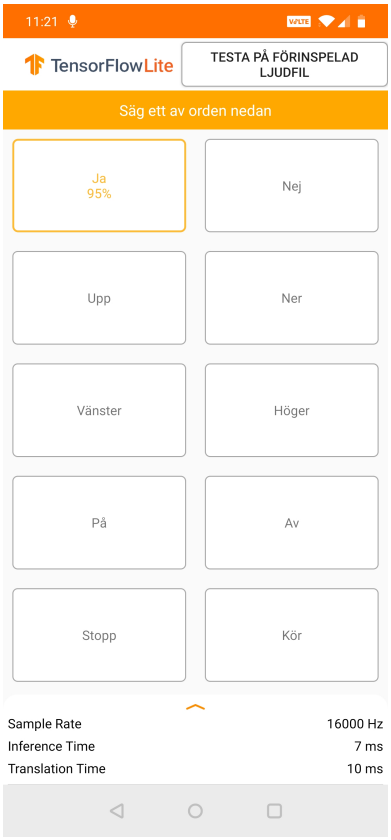


Figure 3.3: A screenshot of the modified speech command app using the Swedish model and MFCC preprocessing.

Chapter 4
Collection of a Swedish Speech Corpus

4.1 Motivation

Speech recognition, like most machine learning fields, needs a lot of data to train a model. Google has the resources to collect very large datasets including their speech command dataset in English.

For this project, I needed a dataset in Swedish. To the best of my knowledge, when I started, there was no such dataset available and I had to collect one.

Given limited time and resources, I obviously did not have the possibility to collect a dataset of the size of Google's. This explains the goal of the project, which was to adapt a model from English using a *smaller* dataset in Swedish.

I chose to select words to record based on the first version of Google's dataset, which at that time contained 30 words. Fig. 2.1 shows the list of these words, where the first row contains the command words.

I directly translated all but the names *Marvin* and *Sheila*, which did not seem necessary nor useful to retrain in Swedish. The Swedish command words are *upp*, *ner*, *ja*, *nej*, *kör*, *stopp*, *av*, *på*, *höger*, *vänster*, and the noise words to ignore are *säng*, *fågel*, *katt*, *hund*, *glad*, *hus*, *Michael*, *hej*, *träd*, *wow*, *noll*, *ett*, *två*, *tre*, *fyra*, *fem*, *sex*, *sju*, *åtta*, *nio*.

4.2 Collection Procedure

When looking for a collection method, I prioritised the simplicity of the recording, gathering and sending process, as well as how well the data samples would be organised, i.e. how the folder structure would be when I downloaded them from the data collection server. Having all of the files in one folder with generic names like "kör 4" would not be helpful when collecting the files for training, as I had already formed the data generator class to label the files based on how they were separated in Google's dataset, one folder per word.

4.2.1 Google Forms

In a first attempt, I created a Google Form. Using this way, I could add one question per word, and when I chose that a file needed to be uploaded to answer that question, one folder for each answer/word was created in the Google Drive folder that collects the answers. This gave me the folder structure I wanted so that it was simple to train a model.

This kept the file samples well organised and, being Google Forms, it gave the added benefit that participants' samples were named after their recorder. I could then easily separate speakers.

I collected the audio files through a URL to a free online tool to easily save audio. However, this solution was not optimal. Some uploads did not get uploaded properly to the form and some were wrongly formatted in regards to file type or resolution, creating more overhead preprocessing of the data before being usable.

In addition, there was no real restriction on the duration of the recordings. I only gave instructions, in text, at the top of the Google Form, so the files were not precisely one second, but varied slightly in duration.

4.2.2 Recording Application

The first collection enabled me to collect a dataset of about 560 recordings. I decided then to write a dedicated application that would eliminate the shortcomings of Google Forms. This new method had the goal to be easier to use, since I was not sure how many more samples I would need. I also wanted as many participants as possible to be able to help me record without limitations in technical know-how or limitations in technical equipment to provide a good audio recording.

Application Architecture. In Fig. 4.1 the state machine of the recording app is shown. The first grid shown is the key words.

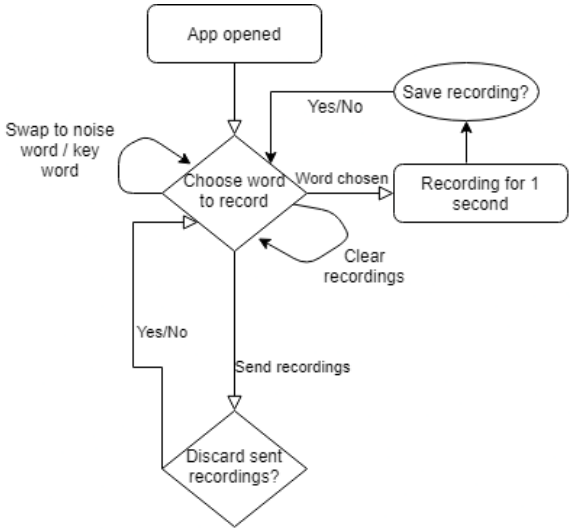


Figure 4.1: Flow diagram of the processes that make up the recording app.

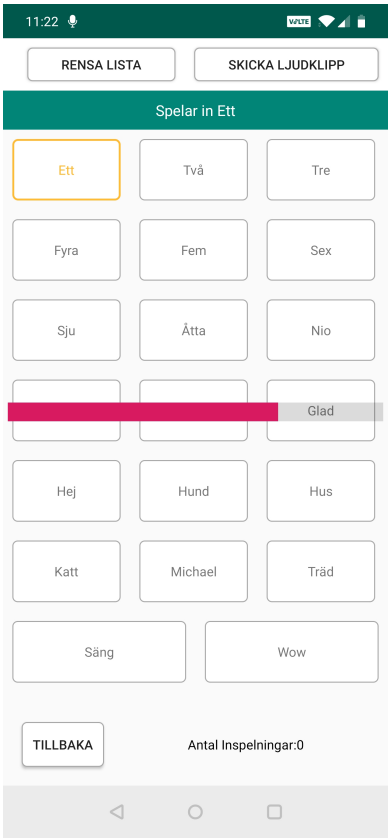


Figure 4.2: A screenshot of the recording application in the middle of recording the word *ett*.

Interface and Description of Use. Figure 4.2 shows a screenshot of the recording application that runs on the Android operating system.

The application has two pages with grids consisting of Swedish words. The first page contains the keywords and the second page contains the noise words. By pressing one of the buttons, the application will record for one second the selected word.

When the application has completed the recording, it will play the audio back and a prompt will ask the user if s/he would like to keep the recording. This recording process can be repeated as many times as needed. The number of recorded samples is tracked in the bottom right of the application screen.

There is also a button to delete all of the recorded samples, in the top left of the app. Next to this button, there is another button to send the recordings to me. When pressed, the recordings will be concatenated into one WAV file, an ordered list of the recordings will be created, and an email with those two files attached will be generated, with the receiver being my email address.

After the send action is concluded, a prompt will ask if the user wants to clear the files that were sent. This is in case someone changed their mind and decided to not send the recordings yet.

4.3 Data Analysis

The format of the data samples followed that of the English dataset. They were one second long, 16 bit WAV files recorded with a sample rate of 16 kHz.

- The gender of the speakers were three female and six male.
- There were some variety to the accents spoken, but most were Scanian.
- There were five speakers in the ages $[20, 30]$, two in the ages $[31, 40]$ and two in the ages 41 and up.
- All words are limited to one second, i.e. 16,000 sample points.
- The 10 command words have 60 or more samples each, while the 20 noise words have 30 or more. Table 4.1 shows the exact number of samples broken down per word.

Word	Av	Höger	Ja	Kör	Nej	Ner	På	Stopp	Upp	Vänster
Count	66	64	63	63	61	61	64	65	68	60
Word	Ett	Två	Tre	Fyra	Fem	Sex	Sju	Åtta	Nio	Noll
Count	35	30	30	30	30	30	30	30	32	36
Word	Säng	Fågel	Katt	Hund	Glad	Hus	Michael	Hej	Träd	Wow
Count	30	31	34	37	30	30	30	30	30	31

Table 4.1: The number of samples in the Swedish dataset. The first word row is for the keywords and their amounts. The other rows describe the noise words.

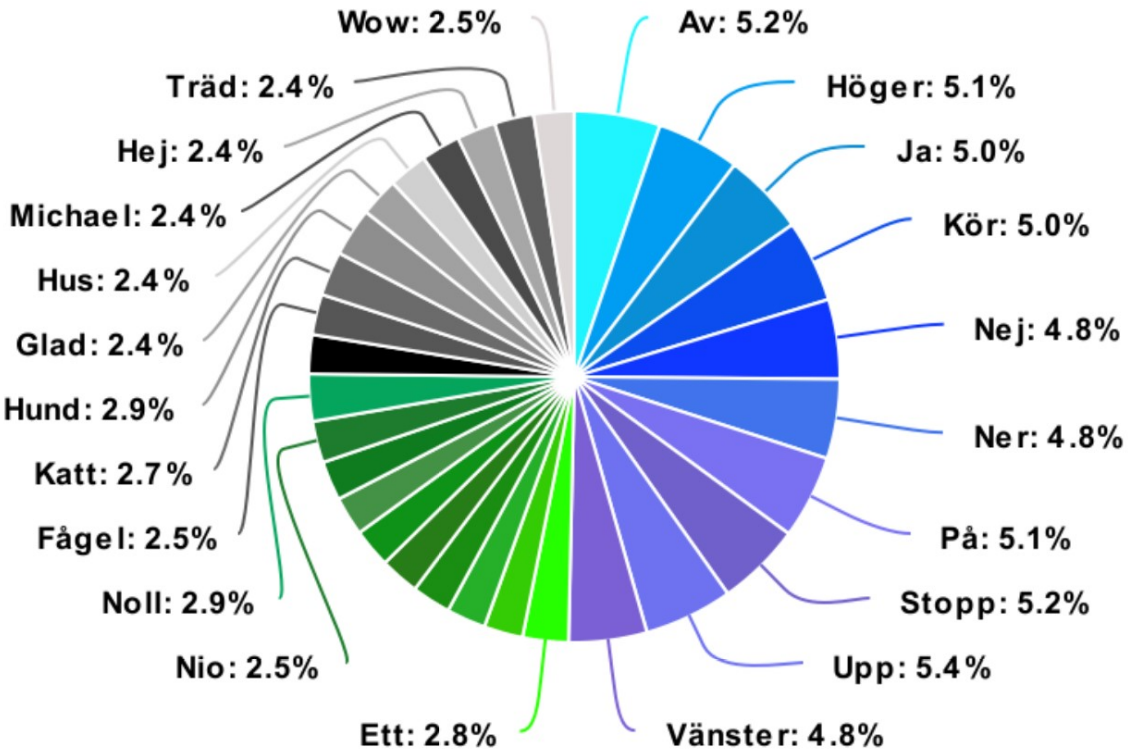


Figure 4.3: Sample distribution. Blue hued slices belong to keywords, green hued are numbers (noise words), gray slices are other noise words.

Word	Avg. amplitude	Word	Avg. amplitude
av	0.0319915	två	0.025331382
nio	0.028247776	ner	0.029569779
hund	0.030639486	upp	0.015508879
kör	0.03212368	ett	0.017379973
katt	0.016292006	glad	0.03131938
michael	0.029442405	fyra	0.032550946
fågel	0.028528877	höger	0.03549839
tre	0.024380192	fem	0.028214794
åtta	0.027719777	på	0.02318398
hej	0.025882196	ja	0.030171564
stopp	0.024081478	vänster	0.026099792
nej	0.021604436	träd	0.027674261
sex	0.014786018	sju	0.028273618
noll	0.030434398	säng	0.02333186
hus	0.024855552	wow	0.03628734

Table 4.2: Mean absolute value of amplitudes of the word recordings. Audio was normalized to the range [-1, 1].

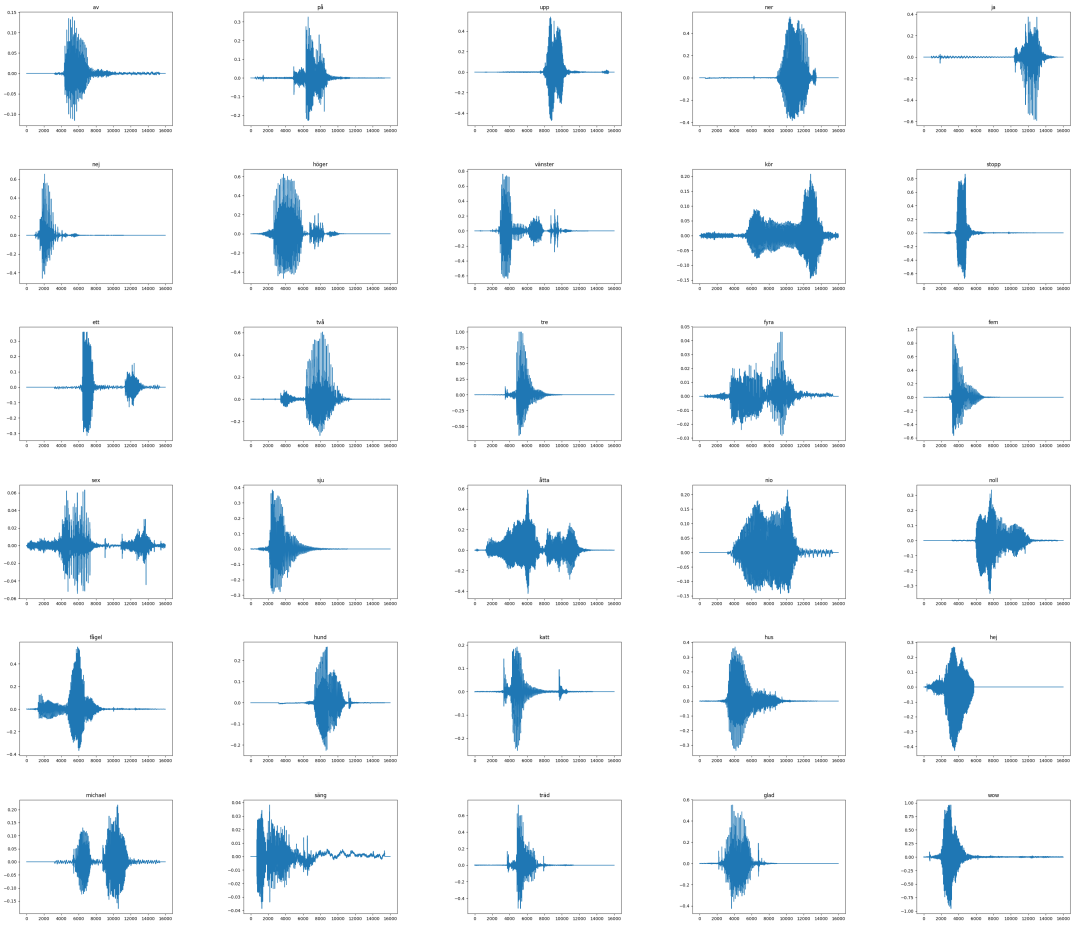


Table 4.3: Examples of audio signals from the dataset.

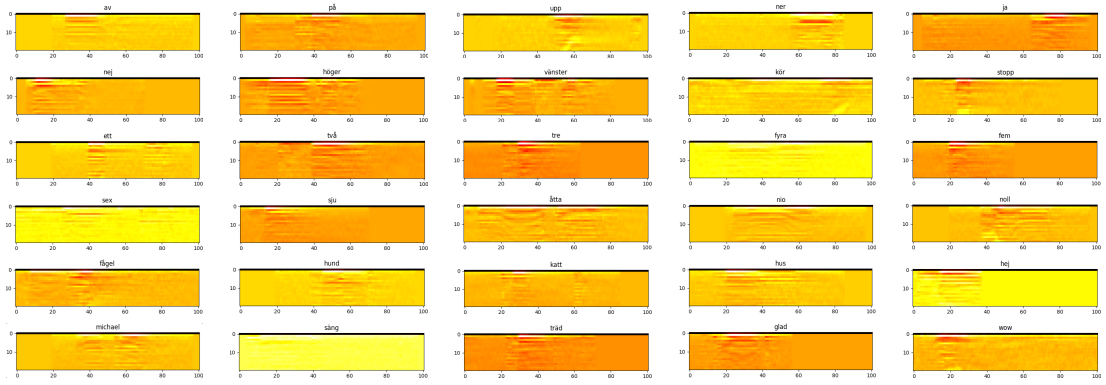


Table 4.4: Examples of MFCCs of recordings from the dataset.

Chapter 5
Results

5.1 Training an English Model with Google's dataset

After training the English keyword spotting model on the full dataset for 50 epochs (without converging), the accuracy was on average 99.4% between the two preprocessing resolutions. For the English model, I used the Adam optimizer as it seems to let the model reach a higher accuracy faster initially.

I trained on the English data using two different preprocessing settings for MFCC construction:

1. One higher resolution using 2048 *fast Fourier transform* (FFT) frequencies and 128 mel frequencies, and

2. One faster model with 480 FFT and 40 mels.

I used the same resolution for the Swedish models as for the English one, making a total of two model types. The final confusion matrices are shown in Tables 5.1 and 5.2.

5.1.1 Training a Swedish Model

Augmenting the Data. With the Swedish dataset being so much smaller than the original one (≈1%), training and data usage needed to be augmented in order to be effective. I tried different methods to make the most of the data, and eventually, I wrote a data generator along the lines of Keras' `ImageDataGenerator`, which augments the audio data.

With inspiration from Park et al. (2019) and Ko et al. (2018), I chose as parameters: *pitch range*, *time dilation*, *time shift*, *time masking*, *noise probability* as well as likelihood for the generator to add a sample of solely background audio.

silence	_unknown_	yes	no	up	down	left	right	on	off	stop	go
1.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.997	0.	0.	0.	0.001	0.	0.	0.001	0.001	0.	0.
0.	0.005	0.993	0.	0.	0.	0.002	0.	0.	0.	0.	0.
0.	0.012	0.	0.983	0.	0.	0.002	0.	0.	0.	0.	0.002
0.	0.012	0.	0.	0.979	0.	0.	0.	0.002	0.007	0.	0.
0.	0.025	0.	0.	0.	0.975	0.	0.	0.	0.	0.	0.
0.	0.01	0.	0.	0.	0.	0.99	0.	0.	0.	0.	0.
0.	0.018	0.003	0.	0.	0.003	0.	0.977	0.	0.	0.	0.
0.	0.015	0.	0.	0.	0.	0.	0.	0.982	0.003	0.	0.
0.	0.012	0.	0.	0.007	0.	0.	0.	0.002	0.978	0.	0.
0.	0.005	0.	0.	0.002	0.	0.	0.	0.	0.	0.993	0.
0.	0.015	0.	0.002	0.	0.002	0.	0.	0.	0.	0.	0.98

Table 5.1: Low resolution Keras model trained on the entire Google dataset. The labels are *_silence_* (background noise), *_unknown_* (noise words), *yes*, *no*, *up*, *down*, *left*, *right*, *on*, *off*, *stop* and *go*. Accuracy: 99.1%.

silence	_unknown_	yes	no	up	down	left	right	on	off	stop	go
0.91	0.09	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	1.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	1.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.002	0.	0.998	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.005	0.	0.	0.993	0.	0.	0.	0.	0.	0.002	0.
0.	0.007	0.	0.	0.	0.993	0.	0.	0.	0.	0.	0.
0.	0.005	0.	0.	0.	0.	0.995	0.	0.	0.	0.	0.
0.	0.013	0.	0.	0.	0.	0.	0.987	0.	0.	0.	0.
0.	0.01	0.	0.	0.	0.	0.	0.	0.99	0.	0.	0.
0.	0.005	0.	0.	0.	0.	0.	0.	0.	0.995	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	1.	0.
0.	0.012	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.988

Table 5.2: High resolution Keras model trained on the entire Google dataset. The labels are *_silence_* (background noise), *_unknown_*, *yes*, *no*, *up*, *down*, *left*, *right*, *on*, *off*, *stop* and *go*. Accuracy: 99.7%.

The original Keras `generator` class was built to be useful for many different users, and configurable to fit as many different data architectures as possible. Since the generator I built was made to aid this project specifically, the generator is simplified, and parameters such as data normalization and data shape are fixed.

The generator class has a parameter which determines how many extra data samples of background noise, or "silence", should be added to the training. For the English dataset 1% was added, which is about 1'060 clips, and for the Swedish training the value was set to 5%, adding up to about 63 samples.

Training the Model. Originally, Tang and Lin (2017) as well as Lin et al. (2018) used a stochastic gradient descent with learning rate of 0.1 and a momentum 0.9. These were however based on the bigger dataset by Google, and they applied it to mini-batches of 64 examples.

This smaller Swedish dataset uses a mini-batch of 32. In addition, changing the learning rate to 0.01 and momentum to 0.8 improved the learning effectiveness.

tystnad	_okänt_	av	höger	ja	kör	nej	ner	på	stopp	upp	vänster
0.231	0.769	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.912	0.011	0.01	0.01	0.003	0.011	0.032	0.0	0.006	0.0	0.005
0.0	0.545	0.318	0.0	0.015	0.045	0.0	0.0	0.015	0.03	0.0	0.03
0.0	0.641	0.047	0.125	0.0	0.062	0.016	0.062	0.0	0.016	0.0	0.031
0.0	0.635	0.016	0.016	0.111	0.032	0.095	0.016	0.0	0.0	0.048	0.032
0.0	0.698	0.016	0.032	0.0	0.079	0.032	0.032	0.0	0.048	0.0	0.063
0.0	0.525	0.0	0.033	0.049	0.016	0.213	0.082	0.016	0.0	0.0	0.066
0.0	0.557	0.016	0.0	0.033	0.0	0.098	0.279	0.0	0.0	0.0	0.016
0.0	0.688	0.109	0.016	0.016	0.031	0.062	0.0	0.0	0.062	0.0	0.016
0.031	0.754	0.015	0.0	0.015	0.015	0.031	0.0	0.031	0.062	0.015	0.031
0.044	0.824	0.059	0.0	0.029	0.0	0.0	0.0	0.0	0.015	0.015	0.015
0.017	0.883	0.0	0.0	0.017	0.0	0.017	0.05	0.0	0.0	0.017	0.0

Table 5.3: Confusion matrix of the Swedish model trained *without* a pre-trained stack. Acc: 51%, loss: 1.88

tystnad	_okänt_	av	höger	ja	kör	nej	ner	på	stopp	upp	vänster
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.002	0.888	0.005	0.013	0.016	0.01	0.002	0.011	0.027	0.008	0.013	0.006
0.0	0.061	0.879	0.0	0.0	0.0	0.0	0.0	0.0	0.03	0.03	0.0
0.0	0.062	0.0	0.875	0.0	0.0	0.0	0.0	0.031	0.0	0.0	0.031
0.0	0.063	0.0	0.0	0.857	0.032	0.0	0.016	0.016	0.016	0.0	0.0
0.0	0.048	0.0	0.016	0.016	0.921	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.18	0.0	0.0	0.0	0.0	0.77	0.033	0.0	0.0	0.016	0.0
0.0	0.098	0.0	0.0	0.016	0.0	0.0	0.852	0.0	0.0	0.0	0.033
0.0	0.234	0.016	0.0	0.0	0.016	0.0	0.0	0.719	0.016	0.0	0.0
0.0	0.046	0.015	0.0	0.0	0.046	0.0	0.0	0.015	0.831	0.031	0.015
0.0	0.103	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.015	0.882	0.0
0.0	0.133	0.0	0.0	0.0	0.017	0.017	0.0	0.0	0.017	0.0	0.817

Table 5.4: (Highres) Confusion matrix of the Swedish model, pre-trained on the English dataset. Acc: 86%, loss: 0.39

Influence of the Corpus Size. I investigated a possible bias for words which have more samples and I tried to determine if the accuracy improvement begins to slow down with the amount of samples. For this, I ran the Swedish training multiple times, using different subset sizes of the dataset.

Table 5.5 and Figure 5.1 show the results of these experiments. The amount of noise words were a consistent 80/20 split for each of the subset results. All tests displayed in that table use the higher resolution preprocessing, and ran with EarlyStopping looking at val_accuracy (5 epochs). This means that the model would stop its training if the validation accuracy did not improve over 5 epochs, essentially when it was no longer improving. It is set to look at validation accuracy because the validation examples are not augmented like those going in to the training phase, and so they were more consistent and reliable for the model accuracy.

model type	dataset	accuracy	loss
not pre-trained	80/20 split, kfold	47.6%	6593.2
pre-trained	10ex	71.2%	1.06
	20ex	79.4%	0.68
	30ex	82.5%	0.55
	40ex	84.5%	0.49
	50ex	86.0%	0.44
	80/20 split, kfold	85.7%	0.43

Table 5.5: [2020-04-30] Partial training of the Swedish data, average of 5 training runs. The "N"ex means N entries of each keyword class was chosen for the training dataset (noise words stayed 80/20 split). 80/20 splits mean 48 samples on training, 12 on testing. Average and standard deviation for the not pre-trained model is huge!

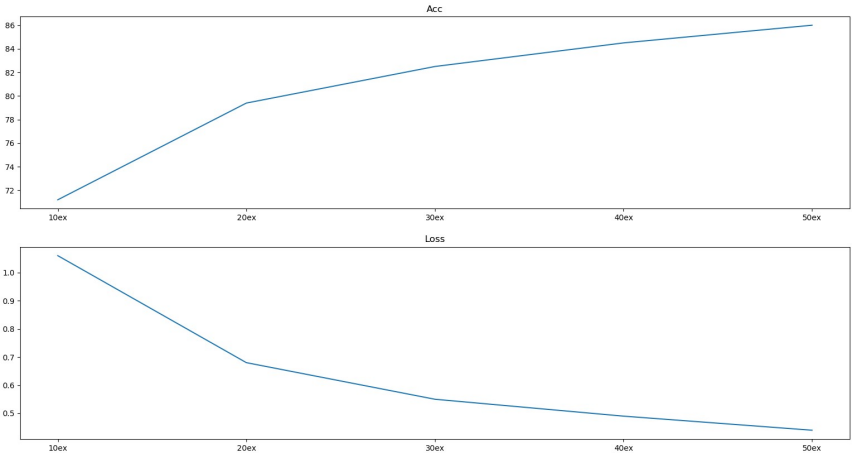


Figure 5.1: [Upper is accuracy in %, bottom is loss] The partial accuracies and losses from Table 5.5 plotted in a graph. The values for the "not pre-trained" model are omitted as they are much larger than the others.

Statistical Observations. It is worth noting that the standard deviation was rather high between each run for each subset. I believe this was caused by the selection of the samples, which was randomised. Given a train/validation split which considered the speakers, like they did in the English dataset, I think the result would be more aligned. However, given the restricted amount of speakers for this dataset, 9, this would be more difficult to accomplish. See Table 4.1 for the number of samples for each of the words in the Swedish dataset.

The trade-off between the slower and faster model types were .6% for the English model, and 8.1% for the Swedish, and the faster preprocessing is about 5 times quicker on the mobile application as well as when training.

5.2 Android Application

I wrote an Android application to embed the speech recognition engine. I started from code provided by Google for command word recognition. I modified it to work with my model – both with the labels being different (order) and with the model using MFCC as input and not the audio samples.

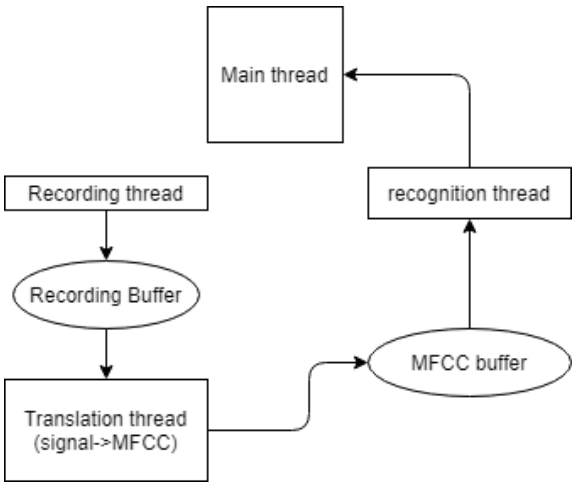


Figure 5.2: The Thread and resource architecture of the speech recognition app.

With the two models deployed on the mobile application, while the more accurate one is slower, it is still quick enough to realistically be usable: ~75ms, compared with ~15ms. These values were measured on a OnePlus 7 Pro running Android 10.

Along with the two models and an option to change in real-time, the app also got some debugging features like an image of the MFCC fed into the model, and the different probabilities. The features can be seen in Fig. 5.3.

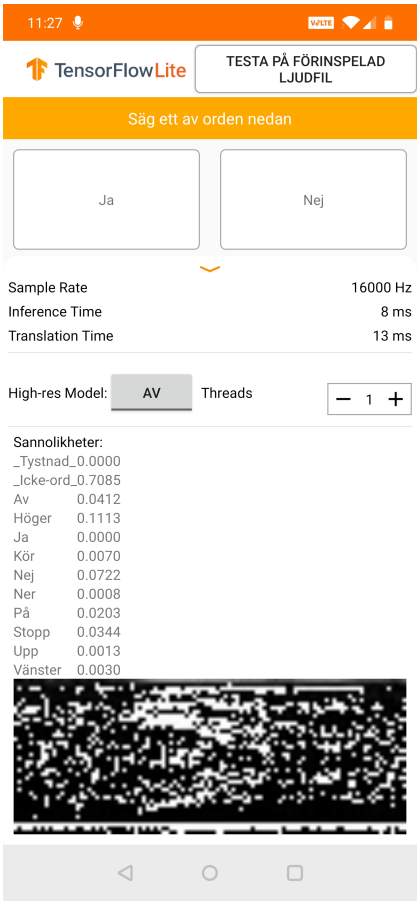


Figure 5.3: A screen shot showing the debug features of the Swedish speech commands app, displaying the MFCC, the different probabilities, and a button to toggle the high-res model.

Chapter 6
Discussion & Conclusion

The project aimed at utilising the speech command dataset released by Google to train a keyword spotting model, and then retrain this model to recognise Swedish words. I could reach an accuracy of 86% from a baseline of 51%, yielding an improvement of a 29% lower false rejection rate.

I researched the English dataset, the resnet model structure and built it in Keras. I also gathered a Swedish dataset consisting of 1261 recordings over a total of 30 words between 9 speakers, and rewrote a demo app on Android to test the model adapted to this Swedish dataset. To collect this dataset, I built another Android app, sent it out to my friends and family so that they could record the words and email them to me.

The bias of the baseline model, in Table 5.3 to lean towards the unknown label, "_okänt_", even the silence samples, can be explained that 50% of the words coming in are labeled noise words, and 5% of the dataset, about 63 audio clips, are silence

In table 5.4 the individual label accuracies are somewhat varied. This can be caused by the choice of noise words. "På" shows an accuracy of 71.9%, which can be correlated to the noise word "två" (two). The most confused label for "på" is just "_okänt_" (unknown), so it is quite likely. The same deduction can be made for the 77% accuracy for "nej" (no). Among the noise words is "hej" (hello), and with "nej" spoken without a Scanian accent is very similar to "hej".

Besides those two words the other keywords are accurate more than 80% of the guesses.

6.1 Future Work

At the moment, the model has a bias towards Scanian Swedish since most data samples come from speakers of this region. An obvious improvement would be to collect a larger corpus with more data samples and from a larger variety of speakers.

I carried out this adaptation experiment on one single language: Swedish. A further work could be to evaluate the adaptation technique I developed on other languages or word

subsets. In addition to porting the system to new languages, this would probably give insights on language differences: Do we need more samples, epochs, etc.?

Another interesting topic to pursue would be to get a better understanding of the model parameters such as the influence of the number of speakers, distribution of speakers across age, regional accents, gender, etc., number of samples per word, and adapted language. This would enable us to fine-tune the model with a greater efficiency.

Another point of interest would be to analyse the preprocessing resolution more in depth. I compared two configurations and I showed a significant difference both in performance and in computational demand. Finding a good middle ground or which parameters affect the trade-offs would be useful to look into.

In the literature, I found a couple of resnet block structures with different numbers of convolutional layers. In my experiments, I evaluated only one variant of a resnet block. Analyzing the influence of different numbers of nodes and layers on the training time and adaptability could also be interesting.

6.2 Conclusion

All in all, this project has taught me the benefits of using networks trained on large datasets to adapt speech recognition to different domains and languages. It has also provided evidence that this is a viable solution when it comes to keyword spotting within a more specific application context.

I am really happy with how well the data collection went and how good the resulting adaptation was. The fact that the smaller dataset for the specified domain still performed so well shows that this system can be applied to keyword spotting domains, even if there is no large pre-existing dataset.

Bibliography

Fu, C. (2020). Java implementations of mfcc calculation based on the python module librosa. https://github.com/chiachunfu/speech/blob/master/speechandroid/src/org/tensorflow/demo/mfcc/MFCC.java. Accessed 2020-01-09.

Google (2018). Tensorflow speech commands example. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/speech_commands/.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *https://arxiv.org/pdf/1512.03385.pdf*.

Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. *https://arxiv.org/pdf/1412.6980.pdf*.

Ko, T., Peddinti, V., Povey, D., and Khudanpur, S. (2018). Audio augmentation for speech recognition. *https://www.isca-speech.org/archive/interspeech_2015/papers/i15_3586.pdf*.

Lin, Z. Q., Chung, A. G., and Wong, A. (2018). Edgespeechnets: Highly efficient deep neural networks for speech recognition on the edge. *https://arxiv.org/abs/1810.08559*.

Nguyen1, T.-S., Stuker, S., Niehues, J., and Waibel1, A. (2019). Improving sequence-to-sequence speech recognition training with on-the-fly data augmentation. *https://arxiv.org/pdf/1910.13296.pdf*.

Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk1, E. D., and Le, Q. V. (2019). Specaugment: A simple data augmentation methodfor automatic speech recognition. *https://arxiv.org/pdf/1904.08779.pdf*.

Sainath, T. N. and Parada, C. (2015). Convolutional neural networks for small-footprint keyword spotting. *https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43969.pdf*.

Tang, R. and Lin, J. (2017). Honk: A pytorch reimplementation of convolutional neural networks for keyword spotting. *https://arxiv.org/pdf/1710.06554.pdf*.

Tang, R. and Lin, J. (2018). Deep residual learning for small-footprint keyword spotting. *https://arxiv.org/abs/1710.10361*.

Warden, P. (2017). Speech commands: A public dataset for single-word speech recognition. *https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html*.

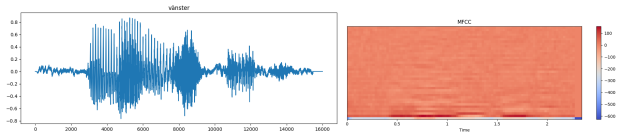
INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2020-06-15

EXAMENSARBETE Domain Adapting English Speech Recognition to Swedish
STUDENT Michael Hansen
HANDLEDARE Pierre Nugues (LTH)
EXAMINATOR Jacek Malec (LTH)

Konsten att lära en engelskförstående maskin svenska

POPULÄRVETENSKAPLIG SAMMANFATTNING **Michael Hansen**

Taligenkänning används mycket och väl idag, men det kräver mycket data för att fungera bra. I detta projekt utvecklades ett system för att träna om en engelsk taligenkänningsmodell till svenska med ett insamlat dataset 100 gånger mindre än det ursprungliga.



Automatisk taligenkänning handlar om att få en dator att översätta mänskligt tal till text. Nyckelordssökning är en fördjupning av detta, där man förenklar uppgiften till att urskilja mellan några specifika ord.

Många av dagens lösningar för taligenkänning kräver massvis med inspelningar, vilket leder till invecklade maskininlärningsmodeller som kräver egna servrar för att behandla ljudet snabbt nog för användning. Genom att minska antalet möjliga ord till ett bestämt antal kan man förenkla modellen så att den kan användas direkt på mobila enheter, vilket eliminerar behovet av internetuppkoppling. Denna sortens modeller behöver dock fortfarande en massa data att träna på.

I 2017 släppte Google ett dataset med ∼106'000 engelska ljudklipp av 35 olika ord. Med hjälp av dessa inspelningar kan forskare utveckla taligenkänningslösningar som presterar bra, visat med en hög träffsäkerhet på deras gissningar. Så stora dataset tar lång tid att samla in och kategorisera. Språk som svenska har inte möjligheten att samla in data av samma storlek, vilket begränsar framsteg och prestationen inom området på hemmaplan. Det leder också till att forskningen blir beroende av de engelska företagen för sådan datainsamling.

I detta projekt har jag utvecklat ett system som använder det engelska datasettet för att träna en taligenkänningsmodel som jag sedan anpassar till svenska ord, med ett mindre svenskt dataset som jag själv samlat in. Med denna metod nådde jag ∼86% i träffsäkerhet med en datamängd på ∼1% av den engelska. Jämför det med träffsäkerheten på 51% som fås om jag tränar modellen utan att ha använt den engelska datan innan.

Domänanpassning har använts i andra områden för maskininlärning, och detta projektet visar att dess resultat är även användbart inom taligenkänning.

Med detta system kan specialiserade program för taligenkänning byggas utan att man behöver samla in en stor mängd data, vilket gör det lättare att, till exempel, skapa mobilapplikationer med taligenkänningsfunktioner. Med detta projekt som utgångspunkt kan man lättare träna taligenkänningsmodeller för nya språk, och det skulle vara intressant att se hur systemet fungerar med språk som har andra ursprung än svenska/engelska.