NTNU

Norwegian University of
Science and Technology

Voice Conversion using Deep Learning

Jon Magnus Momrak Haug

Master of Science in Communication Technology
Submission date: July 2017
Supervisor: Torbjørn Svendsen, IES

Norwegian University of Science and Technology
Department of Electronic Systems

I would like to give my sincere thanks to Torbjrn Svendsen for all the help he has provided while I have been working on this master.
I also want to thank all of my family and friends, and especially my mother and father for always being the most loving and supporting parents I know. Without you completing this degree would not be possible.

Summary

This thesis aims to implement a voice conversion system that transforms one persons voice into another persons voice. Mel Frequency Cepstral Coefficients are used as coefficients for one set of tests, while STRAIGHT spectrogram is tried out as another set of features. The system is built using an artificial neural network approach when mapping the features from one speaker to the other.

Training the system is done using first around 300 sentences from 6 speakers that will not be used for testing. This builds a speaker independet stacked autoencoder that is used as a pre-training for the complete network. The encoder and decoder part of the stacked autoencoder is then separated by a shallow artificial neural network mapping layer, mapping features from the source speaker to the target speaker. This is done using only 2, or 70 sentences each from these 2 speakers. Finally, the complete network when combining the stacked autoencoder with the shallow artificial neural network is trained, also using 2 or 70 sentences.

The performance of the individual autoencoders, the complete stacked autoencoder and the complete network has been tested using mel cepstral distortion.

The complete network, when putting everything together was unable to train properly, thus only intermiediate tests are available.

Table of Contents

List of Tables

List of Figures

Abbreviations

Symbol	=	definition
VC	=	Voice Conversion
VT	=	Voice Transformation
NN	=	Neural Networks
ANN	=	Artificial Neural Networks
MFCC	=	Mel Frequency Cepstral Coefficients
VCC	=	Voice Conversion Challenge
GMM	=	Gaussian Mixture Model
mel-CD	=	Mel Cepstral Distortion
DTW	=	Dynamic Time Warping
RELU	=	Rectifiec Linear Unit

Chapter 1

Introduction

This master thesis is a continuation of work done last semester within the field of *voice conversion* (VC). The work that was done last semester functioned as an introduction to the field. The majority of effort was put into a literature study in order to gain familiarity with fundamental theory needed to implement a VC system, which is the goal of this thesis.

This first chapter will introduce what the problem is, along with the motivation behind looking into the field. Then the thesis scope will be given, and lastly the outline of the rest of the thesis is presented.

1.1 General Problem Description

Speech is one of the most important ways we communicate in today's society. It is an effective way to express ourselves and to share information quickly, be that in a conversation face-to-face, over the phone, listening to the radio, or by some other mean.

Speech as a communication form is not only a way of conveying the message we want to. The speech signal also contains cues about the speakers mood, gender, age, etc. It also contains what might be some combination of the these cues resulting in us being able to get the identity of the person uttering the speech.

The term *voice transformation* (VT) refers to the various modifications that is applicable to the sound produced by a person (Stylianou, 2009). The modifications we apply seeks to alter the speech signal in a way that keeps the message content intact while changing other properties of the signal. These changes might include making the person sound older or younger, having a man sound like a woman, changing the perceived emotion of the person (Kawanami et al., 2003) or similar modifications.

In this thesis the focus lies on a subset of VT, namely that of VC. VC differs from VT by making modifications in order to make the speech of one person (the source speaker) sound like it was uttered by another specific speaker (the target speaker) (Mohammadi and Kain, 2017).

1.2 Motivation

VC has applications in a broad specter of areas, and there is an active community of researchers in the field contributing to the advances being made. In 2016 the first ever *Voice Conversion Challenge* (VCC) was held. Here 17 research teams from around the world contributed their VC system to the objective committee of the VCC which compared them. This way, for the first time at this scale it was possible to compare the systems in a more fair manner than what has previously been the case where the different teams might use different corpuses with different amount of data available etc.

So why is VC an interesting area to research? As mentioned there are a variety of applications where a VC symstem can be used. Below is listed some of these applications:

***Text-To-Speech* (TTS) Systems** A major application of VC is within training of new voices for TTS systems. This can be a time consuming and high cost process if a new voice has to be trained and optimized from scratch. Using VC we can exploit voices that are already trained and optimized in the system and use VC to obtain a new voice using much less data from the desired new speaker (Kain and Macon, 1998b; Duxans, 2006).

Entertainment Perhaps the one application that comes first to mind. The entertainment buisness could largely benefit from advances within the field (Stylianou, 2009). When dubbing a movie it would be possible to have only one or a few people doing all of the dubbing. Using VC their voices could then be converted into different voices. This could also be the case in games based on movies where it might be too expensive having the actors from the movie come and read its part.

Medical Applications People that are unable to speak can generate synthetic speech with their own voices instead of some built-in standard voice (Nakamura et al., 2012). Also for people with articulation disorders VC could help to improve everyday life by aiding them when speaking (Aihara et al., 2014).

Speech-to-speech translation As a part of a larger system VC can be used in a system that translates the speech from one language to another, while still preserving the speakers identity (Bonafonte et al., 2006).

There are many positive applications where VC can be applied. It is however also worth mentioning an application where VC can be abused. VC presents an issue for systems where authentication is provided through voice identity (biometric voice authentication systems) (Pellom and Hansen, 1999). This is another active research area that arises as a consequence of the advances made in VC (Wu and Li, 2013).

There are other application areas where VC can be used as well, but these are some of the main ones that are responsible for continued research.

1.3 Thesis Scope and Focus

The VC research field has existed for quite some time, and different approaches have been suggested for the task. The first approach was done using a vector quantization and

codebook mapping approach (Abe et al., 1990). The next proposed approach suggested using *artificial neutral networks* (ANN), and after this using *Gaussian Mixture Models* (GMM) was suggested (Stylianou et al., 1998; Kain and Macon, 1998a). Both GMM and ANN systems are still popular today, and by looking at the systems that were submitted to the aforementioned VCC (Chen et al., 2016a) we can see just that. Most of the systems use either a GMM based mapping, or some flavor of an ANN based mapping.

In this thesis the implemented system was supposed to be based on a state-of-the-art VC system. More specifically one of the ANN systems submitted to the VCC. However, due to a constraint introduced by the library used to build the ANN it was necessary to abandon this particular approach and rather try to implement a previous slightly more primitive version of the same system.

An attempt to improve this system was made by changing the features being mapped from the source to the target from *Mel Frequency Cepstral Coefficients* (MFCC) to a spectrogram obtained from a STRAIGHT analysis. The results that are presented in Chapter 5 are all objective test results using among other things mel cepstral distortion.

1.4 Thesis Outline

The background information and theory needed to follow this thesis is presented in Chapter 2. Some topics that are already mentioned here in the introduction like the source-filter model will be covered in more depth there. Also ANNs and an overview of a VC system will be presented. Following the theory is the materials and methods chapter covering the tools used and choices for using them together with introducing the objective evaluation method. In Chapter 4 the different experiments that were conducted and their results are shown. Lastly in Chapter 5 these experiments and results are discussed and a conclusion is given. This last chapter also contains a future work section where some directions for future work are suggested.

Chapter 2

Theoretical Background

This chapter presents the theory behind VC. Reading this will hopefully give enough information about the field in order to follow the rest of the thesis.

In order to perform VC we need to represent speech in a mathematical way that allows us to change the characteristics of the signal. Section 2.1 shortly introduces how speech physically is produced, then using this knowledge of the physical speech production we see how it leads to one way of representing speech mathematically. In Section 2.2 the different types of information contained in a speech signal, along with how we can discriminate between individuals through these characteristics is mentioned. This section essentially describes what makes voices differ. The next section will serve as an introduction to ANNs. The ANN made in this thesis is a crucial part of the VC system, and having some basic knowledge of ANNs is therefore essential to follow the rest of the thesis. At the end of this chapter a general VC system will be presented. Some information about each step in the system, and the typical modifications done is also provided.

2.1 Speech Production

In order to make modifications to speech, as is the goal of VC, we first need to know how speech is produced(Stylianou, 2009). One fundamental building block in the field is therefore to learn about how speech is produced on a physical level. We can then utilize this knowledge in order to make a mathematical model or representation we can work with.

Human speech production starts from the lungs, where air is pressed out through the glottis. It then continues through the vocal tract and exits out from the oral and nasal cavities (Svendsen, 2014; Gopi, 2014) (see Figure 2.2 for a view of the organs related to speech production). The position of the *articulators* (lips, jaw, tongue, etc.) alter the shape of the vocal tract, thus affecting the air flowing from the lungs on its way out. This is how different sounds are produced (Kain, 2001). Different resonant frequencies in our speech are associated with different shapes that the vocal tract takes on. These resonant frequencies are called *formants*. Formants can be seen as the building blocks for our speech.

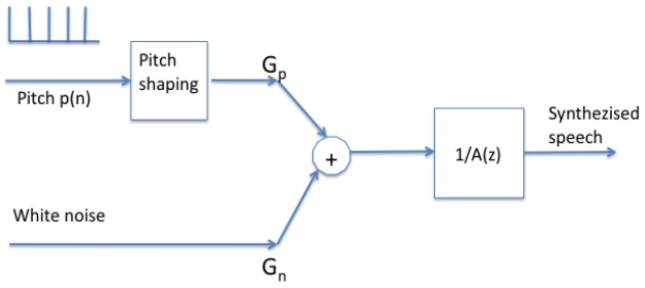


Figure 2.1: Speech synthesis using the LPC model(Svendsen, 2014)

They are important when it comes to making the different basic sounds we produce called *phonemes*(Kain, 2001). Sounds we make can be perceived as something meaningful by others by combining these phonemes, given that we understand the language. Phonemes can differ from language to language, and this can be a challenge when trying to speak another language as we are not used to hearing or uttering these specific phonemes.

2.1.1 Speech models

There are broadly two main categories of speech models, namely the source-filter models and the signal-based models (Mohammadi and Kain, 2017). Source-filter models are more flexible for modification (Mohammadi and Kain, 2017), and therefore using a model from this class might be beneficial for VC. This thesis is using a source-filter model, and therefore some details of this model is provided below. More information about signal-based models can be found in (Moulines and Charpentier, 1990) and (Valbret et al., 1992).

Source-filter model

When using a source-filter model we view speech as a combination of an *excitation signal* (or *source signal*), and a filter(Mohammadi and Kain, 2017). The source signal represents the air coming out through the vocal chords and the filter represents the vocal tract. The source signal and the filter are considered to be independent of each other, thus we can also modify them independently.

As the vocal tract constantly changes with the position of the articulators, the filter is seen as a *time-varying filter*.

The excitation signal can be classified as either a *voiced* or *unvoiced* signal. A voiced excitation signal can be modeled as an impulse-train where the distance between the pulses varies with the fundamental frequency of the speaker. The unvoiced signal on the other hand can be modeled as a noise signal. In reality this is a simplification as sounds are not only classified as purely voiced, or unvoiced, but rather it could also be some sort of combination of these. However, this simplification is useful for giving a basic understanding of one of the ways the source signal in the source-filter model can be modeled. The filter shapes the source signal (as the vocal tract ”shapes” the air) by attenuating certain

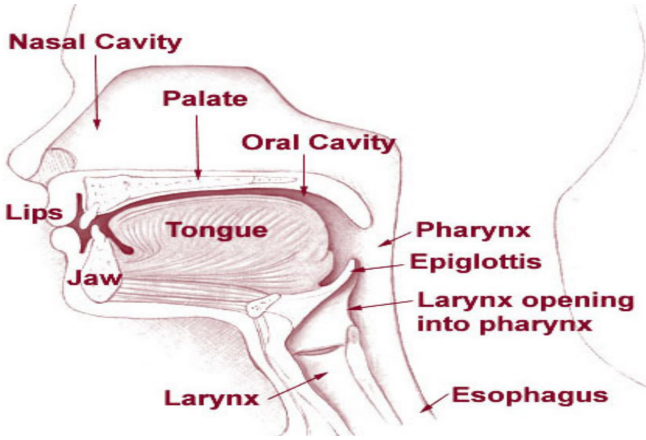


Figure 2.2: The human speech production system (Svendsen, 2014)

frequencies and emphasizing others in the spectrum of the excitation signal. This filtering results in new spectrum of the output signal with a particular spectral envelope and formant structure. Figure 2.1 shows visually a very basic speech synthesis system taking advantage of the source-filter model.

The source-filter model as we have seen tries to replicate the actual way speech is acoustically produced.

2.2 Speech Signal Characteristics

As mentioned briefly in the introduction, the speech signal contains information not only regarding the message being conveyed. Even though the message is the main information (at least in most cases), there is more information we can extract. The environment the speaker is in, the mood of the speaker, gender and age are all examples of information we can get only by listening to someone speak. Also, the identity of the speaker is part of this information. The information about the identity might be seen as a combination of cues such as gender, age and other aspects that are linked to an individual. This embedded information about the identity of the speaker is also known as *speaker characteristics*. These characteristics can be divided into three different type of cues:

Segmental cues Cues that describe the unique sound or *timbre*, of a person. Energy, fundamental frequency F_0, formant locations and bandwidth are acoustical descriptors that are included amongst the segmental cues. These cues mainly dependent of the actual physiology and physical properties of the speech organs. The emotional state of the speaker might also influence these cues (Kain, 2001).

Suprasegmental cues Cues that describe the so called *prosodic features*. The prosodic features affect the duration of phonemes, the evolution of the fundamental frequency F_0 over time (also called *intonation*), and the intensity or *stress* we put into some

phonemes in a word or a sentence. We perceive the average behaviour of the duration of phonemes, F_0, and the intensity as *rate of speech*, *average pitch* and *loudness* respectively. The suprasegmental cues are unlike segmental cues influenced by social and psychological conditions (Kain, 2001).

Linguistic cues Cues more directed to the choice of words, dialects and accents (Kain, 2001).

The linguistic cues are the hardest ones to convert of the three. Most of the research has therefor been focusing on the two other cues.

2.3 Artificial Neural Networks

Artificial Neural Networks origins from an interest in algorithms that try to mimic aspects of how the human brain works. They were very popular from the 80s until the late 90s when the popularity diminished a bit. Now, the technique has become popular again, and it is used for many state-of-the-art applications.

In (Haykin, 2004) a NN is defined as follows: *A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects:*

1. *Knowledge is acquired by the network from its environment through a learning process.*

2. *Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge*

On of the main reasons why NNs had a down period is because of the fact that it is as stated above *...massively parallel*. The computational cost of NN was not feasible if we go back to this period. As machines have become more and more powerful, and also with the introduction of GPUs into machine learning, using NNs is now more applicable. GPUs are unlike CPUs very good at performing many parallel operations (Krewell, 2009), making them particularly suitable for computations in NN.

2.3.1 Components of ANNs

ANNs are organized in *layers* where each layer contains a certain amount of *nodes*. There are mainly three types of layers, the *input layer* which is where the data to be processed is first sent in. The *output layer* which is where we get the output of the system, after the input data has been processed throughout the system. Lastly we have the so called *hidden layers*, which are all the layers in between the input and the output layer.

In figure 2.3 we see an example of a Neural Network. As already said the neural network consists of the input layer, N_i, the output layer, N_0, and arbitrary many hidden layers. Neural networks with more than one hidden layer are called *Deep Neural Networks*. The connections between the nodes are *weighted connections*. The weights are modified to best fit the input data, and there several learning rules that can be used in order to

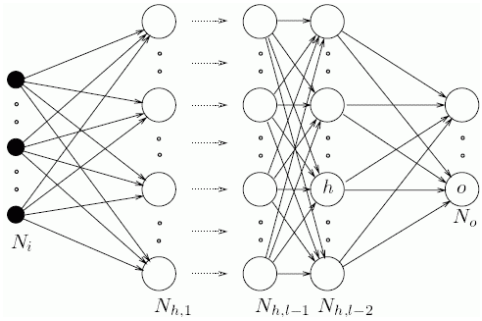


Figure 2.3: Neural Network example

decide how we should modify the weights. Each node receives the weighted input from the connections it has to the layer in front of it

$$y_k = \sum_i w_i x_i \tag{2.1}$$

Learning Process

The learning process mentioned above in the definition is, e.g., done by having some input that we want want to process and we know what the corresponding output should be. Either the output can be a specific value (*regression*) or the network can function as a *classifier*, grouping the inputs into classes (e.g., images can be classified as being a car, dog, ball etc.).

There are two main types of learning when dealing with ANNs. The *unsupervised* learning technique where we do not know the outputs, but rather try to make the network, for instance group together similar items, without necessarily knowing what these items are, we only know they have similar features. This can be be used to get some meaningful representation out of a dataset we do not know too much about initially. The other type is the *supervised* learning. Here we know what the output should be when it is processed through the network.

The data we use is sent in at the input layer and processed through the network in *batches*. Choosing the batch-size is only on of the many parameters one has to consider when building a ANN. The purpose of the batch-size can be to change how often our *optimizer* changes the weights, trying to obtain a better solution. Our optimizer also uses a *learning rate*. The learning rate determines how quickly the network is learning by deciding how large the changes the optimzer does should be.

The learning rate is used as a part of the ANN *optimizer*. There are many different optimizers to choose from, one is the *stochastic gradient descent* (SGD)(Bottou, 2010). This works by calculating the gradient after one iteration to see in which direction it should change the weights in order to reach a minima (or optimum).

To put this all a bit together, consider we have 20 28x28 images. If we lay out the 28x28 image into a vector we have a 784 dimensional vector. If the batch-size is then 5, we will process 5 of these 784 dimensional vectors before our optimizer makes a change in

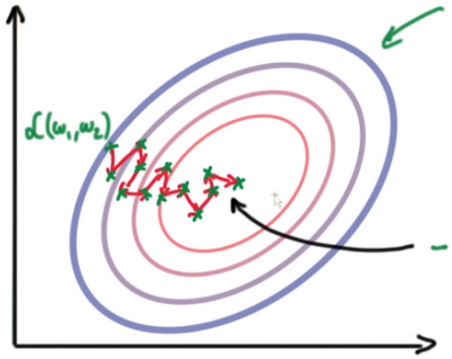


Figure 2.4: SGD updating(Skymind, 2017)

order to get close to the minima. When we have processed all 20 images we are done with the data set. This does not necessarily(almost never) mean that we are done training. One loop over the complete data set is called an *epoch*. The weights we have initially might be far away from the optimal weights that makes the network perform best. In Figure 2.4 an illustration of how the updating might work is shown. In the beginning we might start far off, but as we iterate through we change the direction of where we are going in order to get closer to the optimal solution. The learning rate decides as mentioned how far we are jumping, and thus how fast we are closing in on the solution. Choosing the learning rate is not trivial as choosing a too large value might result in that a good result is hard to obtain since it takes too large steps each time, and choosing too small value can lead to the network taking too much time to complete the training.

Often, one also choose a *learning rate decay*. With this set the learning rate will decrease after one epoch is done, or after one batch is finished, depending on the implementation. This way we can make quite large jumps in the beginning when we are likely to be far off a good solution. Then as we are getting closer and closer to the minima, our steps are getting smaller and smaller so that we can get as close to the minima as possible. These parameters are some of the reasons why there is a lot of trail and error when training ANNs. There is no formular giving you the learning rate and learning rate decay that should be chosen, so one has to try different combinations to find a good values.

The data mentioned above is called the *training set*, which is data that is used to train the network. This is usually a quite large dataset so that the network learn how to generalize what it is learning (e.g., when classifying people in a photo to female or male it is unlikely that the network will perform well if it has only seen one image of a male and a female during its training.). The more data we have the better it usually is.

In addition to the training set we also have a *validation set* or *test set* that is used in the training phase. This is data that our network has not seen and thus we can see how well the network could perform in the ”real world”. We want to be able to use the network for any kind of data we have trained it on, and if it performs well only on the training set the network is practically useless. This phenomena is called *overfitting*. That is when the network has become specialized on the training set and might perform perfectly on that,

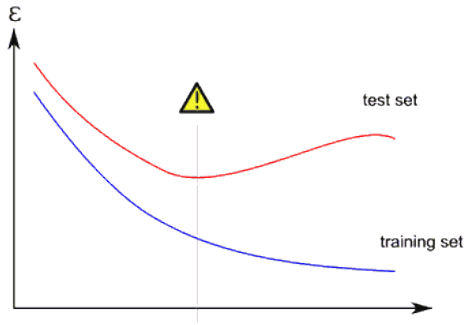


Figure 2.5: Overfitting network(mlwiki, 2017)

however, when seeing new data the network struggles.

Figure 2.5 shows a plot of the error being made on the y-axis and the number of epochs on the x-axis. The error the network makes on the training set becomes steadily lower and lower, but the error on the test set is starting to increase after some number of epochs. We want to stop the training before this happens, so choosing the number of epochs is also important. Often the network error of the training and test set is monitored during training and could be stopped when it is noticed that overfitting is starting to happen.

Another way of preventing overfitting is by using *dropout*(Srivastava et al., 2014). Doing this drops some units or neurons and its connections to the next layer disappears for that pass through the network. Using this a the network will learn to generalize more as it cannot learn the complete training set, since there are small changes to it all the time passing through. This way hopefully the network will generalize well to what we are trying to learn, thus avoiding the increase in error for the test set after some amount of epochs.

An ANN architecture used in this thesis is the *autoencoder*. An autoencoder can be seen as a semi-supervised learning technique. We use the input also as reference values at the output. That is, we are trying to alter the weights so that out network recreates its input on the output layer.

2.4 Voice Conversion

As mentioned in Chapter 1 the goal of VC is to change the voice of some speaker (source speaker) into the voice of another specific speaker (target speaker). This is usually done by modifying properties linked to the segmental and suprasegmental cues mentioned in Section 2.2.

Using the source-filter speech model we have effectively divided the problem of VC into two parts. There is the part focusing on modification of the source signal, and the other part is modifications done to the filter. In order to obtain high quality conversion we cannot restrict the modifications to only one of the two parts, we need to modify both the source signal and the filter (Stylianou, 2008).

Now, a complete general VC system will be presented. The different steps of such a

system will then be presented in a more detailed manner.

2.4.1 General Voice Conversion System Overview

A typical voice conversion system consists of the different parts we can see in Figure 2.6. The general system consists of a *training phase* and a *conversion phase*. First we will discuss the training phase which is what is usually done offline. After obtaining the conversion function that is the result of the training phase, we can do the conversion online. The training phase is the part that can take a bit of time. The actual conversion, once the conversion function is found, can be done in real-time.

Speech analysis

We can see that the first block is *Speech analysis*, this block is closely related to the last one, *Synthesis filter*. One have to choose these to match each other, that is the synthesis part has to be able to synthesize speech by using what the analysis part outputs. As this is not really a part of the actual conversion of the speech but rather a way to represent and reconstruct speech from the way it was represented, researchers typically use an external system for this task in order to focus more on the actual task, which is the conversion. A popular tool for this task is STRAIGHT, as briefly described in Section 3.1 later in the paper. Ahocoder[1] is another used tool for this purpose that uses a signal based method.

Feature extraction

In the feature extraction part of the system we extract the features we think will be a good representation for our signal. A speech signal is non-stationary so when working with a signal like this we more or less always do the actual processing frame-wise. We can look at the signal as short-time stationary, so when considering one frame we say that the signal is stationary within that time. Usually the frames will have a length of about 10-30 msec (Svendsen, 2014) and we usually have some overlap between our frames in order to catch variations better. It is from these frames we extract the features, and we end up with a feature vector per frame.

Two features that can be used are the *mel frequency cepstral coefficients* (MFCC)(Gupta et al., 2013) and the STRAIGHT spectrogram(Zhang et al., 2009).

These are the ones tested in this thesis.

Frame alignment

So we find features for each frame of the signal. Now we have a representation of the signal that contains information so that we can reconstruct the message, but also containing information about the identity of the person speaking. These *feature vectors* we obtain could, i.e., contain information about which phoneme was being uttered at this frame. In voice conversion most methods make use of parallel training data. With parallel training data, the source and target speaker utter the same sentences. There have also been approaches where the system did not require the use of parallel training data, but this has

[1] http://aholab.ehu.es/ahocoder/

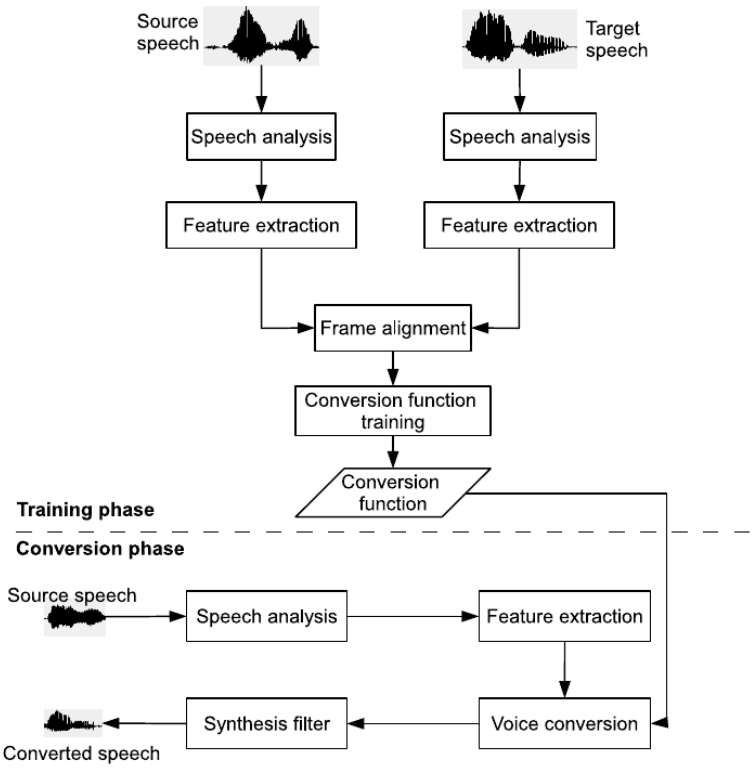


Figure 2.6: A general voice conversion system

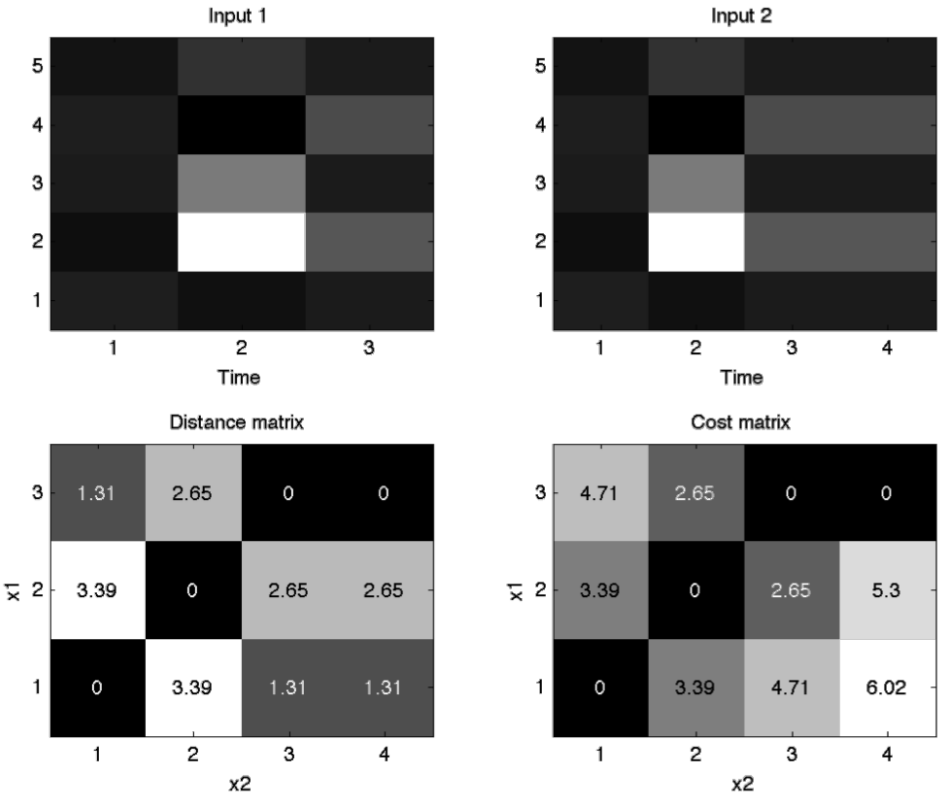


Figure 2.7: Input to DTW algorithm and output of distance and cost, showing the best alignment Forsyth (2012)

shown that parallel data provides better conversion results (Stylianou, 2008). When we have parallel data we can use a frame alignment technique, such as *dynamic time warping* (DTW), to align the frames.

DTW DTW is a dynamic programming technique initially used to compare different speech patterns in automatic speech recognition Müller (2007a). When applying DTW to two signals we look for a pattern-matching of two sequences. DTW can help us find this alignment if the sequences have timescale variations Müller (2007b). The algorithm is used a lot in voice conversion from the early approaches like in Abe et al. (1990) up until recent implementations presented in systems for the Voice Conversion Challenge 2016 Erro et al. (2016)Chen et al. (2016b)Wu et al. (2016).

Figure 2.7 is provided to help illustrate visually what DTW does. Having two sequences $\boldsymbol{X1} = [\boldsymbol{x1_1}, \boldsymbol{x1_2}, \ldots, \boldsymbol{x1_n}]$ and $\boldsymbol{X2} = [\boldsymbol{x2_1}, \boldsymbol{x2_2}, \ldots, \boldsymbol{x2_m}]$, and giving each node a distance, i.e., $d(i, j) = ||\boldsymbol{x1_i} - \boldsymbol{x2_j}||$ we want to find the path that minimizes the overall path cost which we get by summing over all distances in a proposed path $D = \sum_k d(\boldsymbol{x1_k}, \boldsymbol{x2_k})$. This optimal D path gives us the "distance"

between the two given sequences (Forsyth, 2012). For clarification, the $\boldsymbol{x1_i}$ and $\boldsymbol{x2_j}$ vectors are both $D \times 1$ dimensional, and the optimal path has to start in $(1, 1)$ and end in (n, m).

To find this optimal path dynamic programming is used and to speed up the process even more, some constraints are added. These constraints are defined as *local* and *global constraints*. The global constraint limits the area we search for the optimal path in, as we usually assume that the time shifted sequences are not shifted that much. The local constraints defines where it is possible to proceed from a node, i.e., we want to avoid going backwards to find our path. Thus we can set a constraint like this $i_{k-1} \leq i_k \quad j_{k-1} \leq j_k$ to assure the path can repeat the same frame/time, proceed to next frame/time instance, but not go back to a previous one

DTW is necessary as two people will never utter a sentence in completely the same way. One speaker might have longer breaks in between words, or put more emphasis on some parts causing the phoneme duration to increase. The alignment then seeks to map frames from one speaker to the other.

Conversion Function Training

Now that this is done the training for obtaining a conversion function starts. Here we seek to find a function that can map the characteristics of the source speaker into those of the target speaker. All the parallel utterances in the training set are used to create this mapping. After we have trained this function for all the utterances in the training set, the training phase is done and we can start converting the speech of the source.

Conversion

When the training is complete and we have obtained a conversion function, the source speaker can input any utterance he or she may like. An analysis and feature extraction of this signal is executed, and then these parameters enter the voice conversion function and they are mapped to resemble the ones of the target speaker. Lastly the go into the synthesis filter and the converted speech is generated.

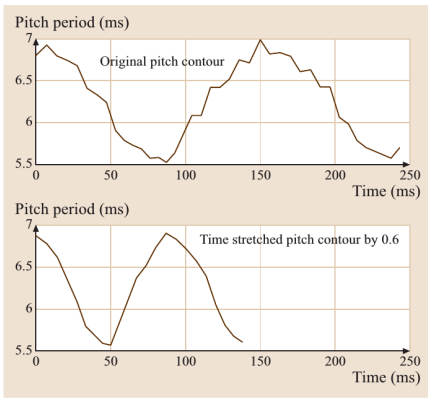
Now I will describe the different modifications we can do to a speech signal. These are the aforementioned source, and filter modifications.

2.4.2 Source Modifications

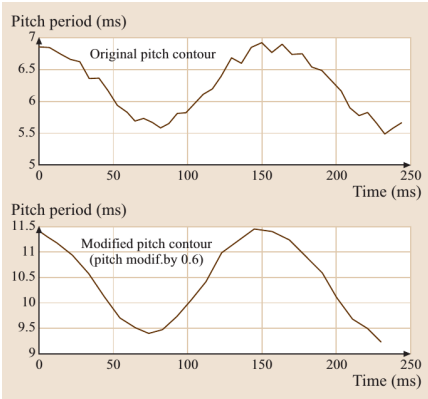
Source or *prosodic modifications* (Stylianou, 2008) are one of the two types of modifications that are best suited for voice conversion. So what exactly is it that we can modify? We can alter all the three prosodic features like this:

Time-Scale Modification

With time scaling we want to change how long the speech to be modified will be. This is useful i.e. when we want a person to sound more enthusiastic we can increase the rate at



(a) Pitch-period contour: original and time-stretched by a factor of 0.6



(b) Pitch period contour: original and modified by applying a pitch modification factor of 0.6

Figure 2.8: Visualization of time-scale and pitch-scale modifications (Stylianou, 2008)

which the person is speaking. That is to decrease the time-scale. While changing the rate with which some speech is spoken we still want to keep the perceptual quality the original speech had. This is visualized in 2.8a

Pitch modification

Pitch or fundamental frequency was mentioned in the section about speech production. With pitch modification we seek to alter the periodic excitation in the source signal, by either compressing or expanding the gap in between the periodic components. This is visualized in 2.8b

Intensity Modification

Intensity modification is thought to be the easiest of the three main source signal modifications. Put simply all we do here is assigning an intensity scaling factor at the analysis time instants.

2.4.3 Filter Modifications

Filter modifications, or spectrum modifications is what we do when modifying the filter characteristics. We make changes to the magnitude spectrum of the vocal tract system in order to change the speech signal.

Using MFCC or STRAIGHT spectrogram are two ways of representing the magnitude spectrum of a speech signal. We can then change the coefficients in these representations in order to change the spectrum, and thus changing the speech signal and how it sounds.

We distinguish filter modifications into two categories. Filter modifications for changing the perceived identity to a specific target, which is what we do in voice conversion, or

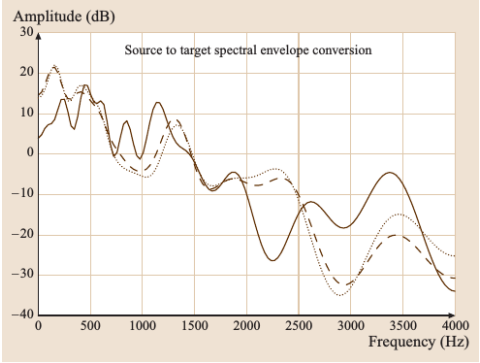


Figure 2.9: Source (solid line) to target (dashed line) filter transformation. The dashed-dot line shows the converted filter. (Stylianou, 2008)

modifications without any specific target.

Without Specific Target

In the latter case we modify the signal in a more general way. This could be, i.e., to make the speech sound like it is being uttered by a man instead of a woman, or an old person instead of a young one. It can also be to change the apparent mood of the speaker. Changes like these can be done with information we have gained from analyzing speech, i.e., after seeing the spectrum of several old and young women it is seen that the formants of an old woman are distributed at lower frequencies than those of of younger woman. Thus, in order to make an old woman sound more similar to a younger woman we can change the placements of the formants (Stylianou, 2008).

With a Specific Target

As mentioned, this is the way we want to modify the spectrum in the voice conversion case. The modifications are done in order to approximate the characteristics of target speaker in a mean-squared sense. A figure trying to illustrate this given here 2.9. Getting from the source spectrum to the target one is as mentioned above a training process where many training utterances are used. Thus we end up with an average spectrum of the target spectra used in the training process (Stylianou, 2008). This averaging is the reason why converted speech often sound *muffled*. We loose some details that would have been in the spectra, but that gets smoothed out.

2.4.4 Techniques

In todays approaches most of the research is done in the filter modification area. Both spectral and prosodic characteristics are changed, but usually quite simple techniques are used to modify the prosody. The most used technique for changing the pitch is by normalizing the source speakers mean and variance of the (log)F_0 distribution to the targets

mean and variance. (Wu, 2015). This can be done by utilizing a linear transform at the frame-level F_0 values. The new F_0 value after conversion will then be

$$F_0' = \frac{\sigma_y}{\sigma_x}(F_0 - \mu_x) + \mu_y \tag{2.2}$$

F_0 is the source speakers initial F_0 value, μ_x and μ_y are the means and the σ_x and σ_y are the standard deviations of the source speaker and target speaker training data respectively. There are other approaches as well, see (Wu, 2015) for more details.

I will mention a bit more regarding the filter modifications. There has been many proposed techniques for mapping the spectral envelope of the source into that of the target speaker. One of the first attempts used vector quantization (Abe et al., 1990). Here the idea is to have two mapping codebooks, one for each speaker. These codebooks represent the correspondences between the source and the target. Other approaches are dynamic frequency warping (DFW), and GMM based mapping. In GMM based mapping a speaker's voice is characterized by m acoustic classes. These classes represent phonetic events, i.e., vowels, nasals or fricatives. The reason for this representation is said in (Stylianou, 2008) to be this *The probabilistic modeling of an acoustic class is important since there is variability in features coming from the same class due to variations in pronunciation and co-articulation.*

Another way of doing the mapping is using ANNs. When training the ANN we can give the network the source speakers features at the input layer, and then have the target speakers features at the output layer. The we do training to find a function that does the best job of mapping the input (source) features to those at the output (target features).

Chapter 3

Tools and Methods

The work in this thesis was conducted using several frameworks and technologies combined. The different tools that were used and the reason behind choosing them will be presented here. As mentioned in Chapter 2, VC involves several separate steps, both for training and the conversion part. Different technologies was chosen for the the different parts as they all have their strengths and weaknesses. The analysis and alignment part was done using Matlab for both the MCEP and STRAIGHT attempt. The training was done using a Deep Learning library in Python. The reasoning behind the choices are presented in the following sections.

3.1 Matlab

Matlab is a well known programming language used a lot for scientific tasks. It has a huge selection of *toolboxes* containing code for specific areas (i.e., Signal Processing Toolbox and Audio System Toolbox). In addition to official toolboxes as the two mentioned ones, it is also possible to add unofficial toolboxes provided by private persons, university institutes or similarly. These are not maintained by Matlab and one have to determine the credibility of the creators.

To build the VC system from this thesis two unofficial toolboxes were used. The VOICEBOX toolbox (Brookes et al., 1997) was used for extracting MCEPs. STRAIGHT was the second toolbox used. It was used for extracting the STRAIGHT spectrogram that was proposed as the new mapping features in this thesis.

STRAIGHT STRAIGHT (*Speech Transformation and Representation using Adaptive Interpolation of weiGHTed spectrum*) is a very popular tool to use for speech processing. It is a powerful tool to use in the analysis and synthesis of a speech signal.

Also in the field of voice transformation it is very popular. Among the 17 teams that participated in the 2016 VCC no less than 11 of them used STRAIGHT as a part of their system (Chen et al., 2016a). STRAIGHT is essentially a vocoder, and it uses the source-filter speech model as a basis (Mohammadi and Kain, 2017). It

is possible to get STRAIGHT from a github repository (shuaijiang, 2014), however for the most recent version one should contact the creator of STRAIGHT, Hideki Kawahara. A colleague and I managed to get this directly from Mr. Kawahara so the version used in this implementation is this.

The description of STRAIGHT from the website is as follows: ”STRAIGHT is a tool for manipulating voice quality, timbre, pitch, speed and other attributes flexibly.” (Kawahara, 2014). The use in the VC systems is mainly for analysis and synthesis. That is at least the purpose it has filled during this thesis. Using STRAIGHT for the analysis and synthesis is very useful as it provides the user with a system that provides high quality synthesized voice. With this part covered one is able to focus on the actual VC part, not on making sure you can synthesize it properly after the conversion.

3.2 Deep Learning Library

As deep learning is the main concept used in this thesis it is necessary to find a suitable library to make the development of the system go smoother. There are a lot of things to keep in mind while deciding upon a library, and one has to evaluate which priorities are the most important for the task at hand. Is it important to have the newest and most feature rich library, or is perhaps the programming language that the library is for the most important matter. Python, being one of the most used programming languages for this kind of work was chosen as the preferred language to write in due to its simplicity. Two very popular libraries are Theano (Al-Rfou et al., 2016) and Tensorflow (Abadi et al., 2015). These languages are very powerful, but it can take some time getting to a level of confidence where you are able to be productive and go from idea to prototype/beta/testing/program quickly. Instead, there are libraries that uses these as a backend and have an easier interface to work with. The library used by the authors of the paper this thesis bases it work on, pylearn2 (Goodfellow et al., 2013), has Theano as a dependency. Using this library was considered, however there is currently no development going on in the library. Therefore it seemed best to go with another library, hopefully making the application more future proof.

Keras (Chollet et al., 2015) became the the preferred library to go with. It fulfilled the properties that was seeked in this thesis. It was also one of the three libraries suggested to use by the developers behind pylearn2. Using Keras enables you to choose between Theano and Tensorflow as the backend. Having the option of choosing these gives you the flexibility provided by these libraries while making it a lot easier to develop using the high-level API Keras provides.

3.3 Python

Python is as Matlab a very popular programming language. It is a popular choice for scientists and people doing data analysis. It was chosen in this thesis because of all the available DL libraries, and becayse of some prior knowledge of the language, lowering the bar for producing valuable code a bit.

Some packages that were used are presented below.

Numpy Numpy is a core package for scientific computing in Python. It handles high dimensional data well, and for this it is a dependency in order to use Keras.

h5py Hierarchical Data format (HDF) is a file format for storing and organizing large amounts of data, and h5py provides a way to work with this in Python. It was specifically created for sharing scientific data (Rhody and Raqueno, 2003). This was not a necessary package to use as numpy/scikit has a built-in function to load Matlab's .mat files, however the idea behind using it non the less came from wanting compatibility. Using this file format enables use of data coming from any program that can save files using this file format instead of the system only being able to handle Matlab's proprietary .mat files.

3.4 Docker

A typical problem when trying to reproduce someone elese work done with software is the following phrase: "It works on my machine". This is a typical issue that appears when working with code. Having all the necessary dependencides and packages can be a problem, but a lot of times even that is not enough. Different versions of the packages and libraries can also influence the results and thus there is a lot of small parameters that can create a sub optimal environment when one wants to reproduce someone elses work, or when working with someone else. OS-specific syntax and environment variables. Using a virtual environment was also considered, however the ease of using Docker from a terminal (as this is the common way) made it seem like a better choice for this task. A lot of the computations were done on a server where only a headless setup was available (e.g., a terminal was all that was accessible).

Docker in comparison with a virtual environment has access to all the resources available on the machine it is running on, so you do not need to allocate anything to it making it very dynamic to work with.

Using Docker allows easier reproduction of results as someone trying to reproduce the results of another team can use the exact same environment as the initial researches given that the docker container is available. This can get rid of problems or differences caused by having slightly different versions of programs and packages installed. In this particular case Tensorflow, Keras, Python and all the libraries used from within Python such as Numpy and h5py will be the same for someone running the container and the provided code should run without any modifications.

3.5 Speech Corpus

The speech corpus used for this work was collected from a Norwegian database. It consists of 12 speakers uttering the same 520 sentences. There are 6 female and 6 male speakers, thus making in well suited for testing a VC system with both inter- and intra-gender. However, during the STRAIGHT analysis, an error occured for two speakers, one male and

one female. Therefore, for this thesis we used 10 of the 12 speakers. It is recorded with $f_s = 16000$.

The following speakers are available:

Male t01, t03, t11, t13, t15, t17

Female t02, t04, t12, t14, d

3.6 Voice Conversion System Network Architecture

The VC system in this thesis is as mentioned using an ANN approach to do the feature mapping from source speaker to target speaker. There are many ways of building an ANN. The architecture used in this thesis is the same as proposed in the paper *Voice Conversion using Deep Neural Networks with Speaker-Independent Pre-Training* by (Mohammadi and Kain, 2014). The architecture will be described in this section.

This paper was chosen as a good paper to start out with and work on improving. The implementation seemed to be possible to implement in the relatively short period a master thesis is conducted.

In addition to using the MCEP features that were used in the original paper, using STRAIGHT spectrogram will also be tested as a new feature here. The STRAIGHT spectrogram has a lot more coefficients per frame, at least 257 compared to the MCEPs 24. This will hopefully enable the VC system to give a better end result.

This approach consists of multiple steps when training the system to get the conversion function. It can essentially be divided into these parts

1. Train a speaker-independent DAE on data from multiple speakers
2. Train a shallow ANN mapping compact features from source speaker to target speaker
3. Create a DNN by taking the encoding part of the DAE, then then shallow ANN, then lasly the decoding part of the DAE and cobine them. Keep the weights obtained from training the different layers.
4. Train the complete DNN on source and target speaker data.

The DAE was used to learn the general acoustical features of speech. This is the most time consuming part of building the VC system as a lot of data was used for this training process.

Around 300 sentence from 6 speakers was used for this testing.

The first part after obtaining the features that are mapped is the training of the DAE. Since we are training AEs here we do not need parallel training data in this step. The deep autoencoder consists of 3 encoding layers (and thus also 3 decoding layers). The original system this was based on extracted 24 MCEPs per frame, while the system using STRAIGHT spectrogram as features used 257 coefficients per frame. This number had to be explicitly coded into the STRAIGHT code basis, otherwise it could change from speaker to speaker, and sentence to sentence, creating a problem when there number of features per frame would be different, thus making training on these coefficients impossible. It was the file *TandemSTRAIGHTGeneralBody.m* that had to be edited. The *Fast*

Fourier Transform length was set on line 97, so somewhere this had to be overrided. The value was chosen to be 257 as this was the lowest value that did not give an error. Since the original MCEP approach only used 24 coefficients it was desirable to go with the lowest amount of coefficients possible also for the STRAIGHT approach.

The sizes of the hidden layers of the MCEP approach were 24 for the input layer, 100 for the first hidden layer, 40 for the next hidden layer, and 15 for the fully encoded layer. The activation functions used were sigmoids on all layers except the decoding layer of the first autoencoder where a linear activation function was used. Some experiments were conducted also using relu as the activation function in place of the sigmoid.

Regularization was also included in order to avoid overfitting. The first AE used Keras' *GaussianNoise* layer with a standard deviation of 0.01. The two other AEs had *Dropout* layers between the input and the hidden layer, and between the hidden layer and the output. These were set to drop 20% of the weights.

The optimizer was *stochastic gradient descent*, the $learningrate = 0.001$, cost function was *mean squared error*. These values were obtained from the paper. In addition the learning rate decay was set to 0 for the first, 2.9 for the second and 2.9 for the third AE

Each of the 3 AEs were trained for 1000 epochs and the batch size was 20. The learning rate decay mentioned above occurred at the beginning of every batch and was calculated as this

$$\text{new lr} = \text{original lr} \cdot \frac{1}{\text{decay} \cdot \text{epoch}}$$

Since the epoch counting starts at 0, it was unchanged for the first epoch.

The AEs are trained like this: first one AE is completely trained, then the input to this is encoded in it and passed on to the next AE where that AE is trained using this as input. This is also done for the last one.

The mapping ANN was used to map the encoded values of the input. The dimension of them was then 15. Testing showed that 15 was the best value of the mapping layer as well.

3.6.1 Complete DNN

After combining the DAE and the mapping ANN we have the complete DNN that will do the actual conversion. First however, we need to fine-tune it. Here the batch-size was also 20, learning rate $= 0.002$. The network was monitored during training and stopped before it started overfitting (Mohammadi and Kain, 2014).

M NEVNE NOE OM HVA DU GIKK FOR P STRAIGHT ATTEMPT SELV OM DU IKKE EGNETLIG HAR FTT GODE RESULTATER. VELGE OM BILDENE SKAL MED I TEORI ELLER HER. ER DEN SOM IVSER AT NOE ER LINERT SOM KAN VRE LITT RART HA MED I TEORIEN.

Effectively this can be viewed as a pre-trained DNN,

The goal was originally to implement a newer version of this approach by the same authors (Mohammadi and Kain, 2016). This was one of the papers submitted to the VCC. The main difference between these two approaches is that the DAE and shallow ANN combination used as a part of the mapping in (Mohammadi and Kain, 2014) is replaced by a Stacked-Join Autoencoder (SJAE). The jointness is introduced to avoid the need of the extra mapping layer. The cost function is what introduces the jointness by including

the error between the hidden layer encodings. Unfortunately, using Keras it was not trivial to create a cost function that could include all the data needed to include the jointness. Therefore, this somewhat more primitive approach using an additional mapping layer was pursued instead.

3.7 RELU

The authors of the paper I have based this on have used sigmoid as their activation function on all (except the output layers where it is linear) layers in their ANN architecture. Another popular activation function is the rectifiec linear unit (RELU) activation function.

There was done some testing with this activation function as well. The same architecture as the one mentioned above was used, the only difference was to swap the sigmoid activation function with the RELU.

3.8 Evaluation Methods

In order to compare with the attempt this is based on *mel-scaled log-spectral distance* was used as an objective,

Based on readings of the litterature *mel-cepstral distortion* (mel-CD) came out as the best choice of an objective evaluation method. This original approach used *mel-scaled log-spectral distance* in the original paper, but they have also in their more recent approaches started using mel-CD instead.

Finding the mel-CD was done for the SAE itself, and for the complete network to assess how well it seemed like the system would do in subjective tests.

The mel-CD was calculated using the following formula(Kominek et al., 2008)

$$\text{mel-CD} = \frac{10}{ln(10)}\sqrt{2\sum_{d=0}^{D}(V_d^{ref} - V_d^{conv})^2}$$

where V_d^{ref} is the reference, or targets value and V_d^{conv} is the converted value of the $d-th$ feature. In this case there were extracted 24 coefficients.

This calculation was done per frame and then averaged over all frames that were used for testing.

Chapter 4

Experiments and Results

In this chapter the different experiments conducted and the results obtained from them will be presented. There was quite a lot of testing being done in this thesis. The experiments ranged from small tests to see which learning rate decay should be chosen to experimenting with the complete system.

4.1 Network Parameter Experiments

The parameters of the network was mostly given in (Mohammadi and Kain, 2014). Some, including the learning rate decay was not. Below the results of experimention with different parameters is presented. Also some of the values and choices that were made in the original paper are tested to verify or to see how alternatives perform.

4.1.1 Learning Rate Decay

Most of the values used when building the DAE were given in (Mohammadi and Kain, 2014), but there were some parameters that needed to be found. One of these were the decay of the learning rate. It is possible to set decay easily in Keras by adding it as a parameter when training the network. However, this would change the rate after each batch. Since the training consisted of several hundred thousand frames, the learning rate would become infinitesimally small rather fast. A scheduler was then introduced in Keras, making it possible to change the learning rate only each epoch. This was changed as shown in Chapter 3.

There is no magical formula that helps you choose the decay, and therefore several different decays were tested. Different values ranging from 0.00000003 up to 3.3 were tested. These tests were done for 100 epochs. The SNRs are calculated for each AE separately. That is, first the input is encoded and then decoded. Then the encoded input is encoded again, and then decoded. Lastly we are at the third AE where the input to the second AE is encoded again, and then decoded. This was done to get an idea of how

Table 4.1: SNRs for different learning rate decays. The bolded ones are the highest for each AE.

	SNR AE1	SNR AE2	SNR AE3
decay			
0	**29.8**	28.9	20.1
0.03	26.7	29.3	25.6
1.3	20.2	30.4	36.6
2.9	18.5	**31.3**	**40.2**

Table 4.2: SNRs for different number of epochs.

	SNR AE1	SNR AE2	SNR AE3
epochs			
250	35.6	29.6	38.6
500	39.7	29.6	36.5
750	41.4	29.7	36.2
1000	42.5	29.6	37.0

well the different decays were working on each AE. Values for some chosen decays are presented in table Table 4.1.

The decay was then based on these tests chosen to be 0 for the first AE, and 2.9 for the second and the third AE. To visually give an impression of how well the different AEs are, see Figure 4.1a, Figure 4.1b and Figure 4.1c.

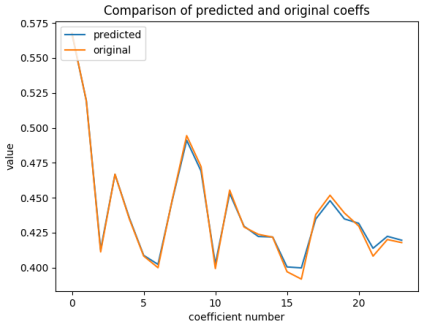
4.1.2 Number of epochs

The number of epochs was originally set to 1000. Some tests were run to see whether it was necessary with this many epochs.

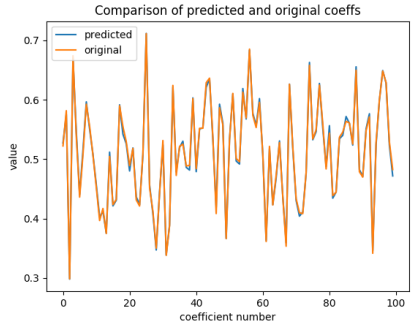
Table 4.2 shows the SNR and the mel-CD of the different AEs for different number of epochs. The decay is 0 for the first, 2.9 for the second and 2.9 for the third, as was found to be best from previous experiments.

4.2 Mapping ANN

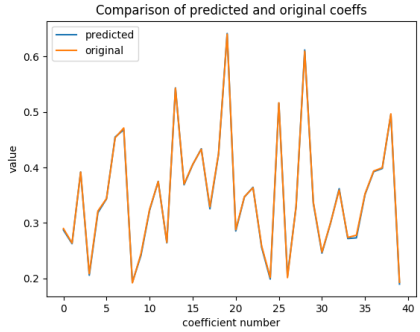
Several different combinations of hidden layer dimension and learning rates were tested for the mapping ANN layer. The results are presented in Table 4.3



(a) Original and predicted first frame, AE1



(b) Original and predicted second frame, AE2



(c) Original and predicted third frame, AE3

Table 4.3: SNR for different dimensions and learning rates on mapping ANN

	dim=5	dim=10	dim=15	dim=20	dim=30	dim=40
learning rate						
1.0	55.5	56.0	56.1	55.2	54.9	53.7
0.1	55.9	55.8	55.4	55.8	55.9	55.9
0.01	55.5	55.4	55.8	55.8	55.5	55.5
0.001	55.3	55.5	55.2	55.4	55.9	55.6
0.0001	55.8	55.7	55.5	55.6	55.3	5.3

Table 4.4: mel-CD for the different number of epochs

	mel-CD
epochs	
250	2.4321
500	2.4340
750	2.4306
1000	2.4309

Table 4.5: SNR for different learning rates using RELU activation function

	SNR AE1	SNR AE2	SNR AE3
learning rate			
0	26.8	17.8	9.1
0.003	28.0	17.4	9.5
0.009	27.4	17.3	8.0
3.9	21.6	14.0	16.8

4.3 Complete SAE

These test were done to see how the complete SAE was performing, unlike above where values for each of the AEs were presented by themselves. The values for which the above results were the best has been used. A lower value is better.

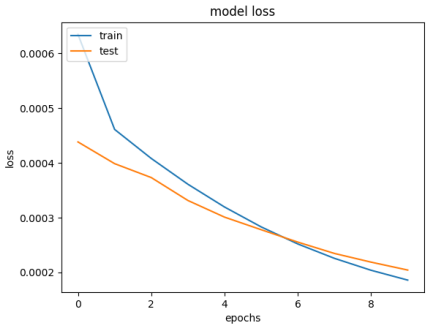
4.4 Complete Trained Network

With the network parameters set as stated in Chapter 3, and the decay as also stated right above, the complete network is trained and tested.

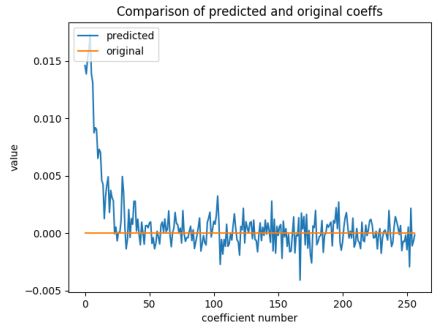
The network does for some reason not manage to change its weights during training, and thus the complete network becomes useless.

4.5 RELU

The RELU was tested with different decays as well. In Table 4.5 some chosen values are presented.



(a) Loss of training and test during network training



(b) Predicted output vs original coefficients output of the first AE

4.6 STRAIGHT Spectrogram

Using the straight spectrogram gave similar results to what we see in Figure 4.2a and Figure 4.2b for all the tested setups. Comments will be given in Chapter 5.

Chapter 5

Discussion and Conclusion

This chapter will discuss the findings from Chapter 4. After commenting on the results, the thesis as a whole will be reviewed and a conclusion will be made. Lastly future directions to take the work will be mentioned.

5.1 Discussion

The learning rate decay values that were used seems to be good choices. It is always a bit hard to say based on only using objective evaluations as these as best only as an indicator of how well the complete network will perform. Also it might seem like 1000 is a fine choice when it comes to the number of epochs. Even though a smaller amouunt gives better SNR for some combinations in the second and third AE, the first one is only getting better and better.

When training the mapping ANN there was not a big difference in any of the tested architectures. This seems strange, but it might be that the task of mapping the compact features is trivial, so that it does not really matter how this layer is built.

The complete network has been tried finished for a long time, but there have been great problems with the last fine-tuning. After building the complete network and attempting to fine-tune it it seems like non of the weights are updating. The loss during training remains constant no matter how many epochs is run. This function will be a good place to look for people wanting to get the approach working.

Using a RELU as activation function seems like a good idea. Even though only a few test were run it shows promising results, at least for the first AE. To se whether it would be a good choice for the complete network some more tests should be run. Also, it can be smart to try some different parameters as the ones used here were the exact same as in the one using sigmoids.

Finally, the STRAIGHT spectrogram approach. It seems like a good idea to use these features as we are able to reconstruct the speech very well, and with the introduciton of better ways of working with ANNs it seemed possible to get it working. This first approach is however not good. Analysing what the spectrogram is and looks like to get a deeped

understanding of it is necessary in order to go deeper into it, and to see if this is in fact a good set of features.

5.2 Conclusion

This thesis implemented a model for transforming the speech of one person into that of another. This was done by using features mapped using an ANN. Both MFCC and STRAIGHT spectrogram features were tested to as the features to be mapped.

The classic source-filter model has been in the foundation of the work so that the source and filter modifications can be done separately. The focus here was on the filter modifications.

The MFCC attempt seemed to be working well based on the visual tests seen in Chapter 4 as well the SNR values from the same chapter, but problems during training the complete network made it impossible to train, and therefore also test the complete network.

When it comes to the STRAIGHT spectrogram, it is still possible that the STRAIGHT spectrogram is suitable for a VC task, but with the visual inspections from Chapter 4 it might seem like since the values of the coefficients of the STRAIGHT spectrogram are so low at most of the frames that it is hard for the network to learn to use this in a good way.

The first approach was to directly implement the STRAIGHT spectrogram version, but due to tests like this it was abandoned in order to try to get a working MCEP version instead. This was done in order to check the general network architecture that was implemented. It is worth saying that completing a more basic approach is best to start with for someone else trying the same task as I have been working on here.

There are great limitations for the tests run in this work as it was not possible to run subjective evaluations. This should be done if possible in VC.

5.3 Future Work

Some things that were learning along the way and directions that could be beneficial to continue working in.

Using Python for everything in this thesis the MCEP and STRAIGHT features have been extracted using Matlab. It is however possible to do everything from within Python. WORLD (Morise et al., 2016) and Cheaptrick (Morise, 2015) have proposed some changes to improve slightly when compared to TANDEM-STRAIGHT. WORLD is a vocoder similarly to TANDEM-STRAIGHT, but there has been built a python wrapper[1] for it, so that one can use this package to get a spectrum representation similar to that obtained here with STRAIGHT. This could possibly also lead to an improvement in performance as WORLD focuses more on real-time processing abilities, and therefor might be faster.

Using GPU for ANN calculations during the work on this master thesis there was no available server where a GPU could be used. This leads to significantly longer

[1] https://github.com/JeremyCCHsu/Python-Wrapper-for-World-Vocoder

processing times and testing becomes a very time consuming task as there are often small changes and a lot of parameters that have to be tested to improve the system.

f_0 a obvious limitation to this system is the lack of f_0 conversion. This was done on purpose in order to focus solely on the filter modification.

VCC Corpus switch from the currently used corpus to that used in the VCC. This makes results easier to compare as the results from the VCC are publicly available.

Subjective Evaluation another obvious aspect lacking here is that there is no subjective evaluation.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from `http://tensorflow.org`.

Abe, M., Nakamura, S., Shikano, K., and Kuwabara, H. (1990). Voice conversion through vector quantization. *Journal of the Acoustical Society of Japan (E)*, 11(2):71–76.

Aihara, R., Takiguchi, T., and Ariki, Y. (2014). Individuality-preserving voice conversion for articulation disorders using dictionary selective non-negative matrix factorization. *ACL 2014*, page 29.

Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A., et al. (2016). Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*.

Bonafonte, A., Höge, H., Kiss, I., Moreno, A., Ziegenhain, U., van den Heuvel, H., Hain, H.-U., Wang, X. S., and Garcia, M.-N. (2006). Tc-star: Specifications of language resources and evaluation for speech synthesis. In *Proc. of LREC Conf*, pages 311–314.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

Brookes, M. et al. (1997). Voicebox: Speech processing toolbox for matlab. *Software, available [Mar. 2011] from `www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html`*, 47.

Chen, L., Saito, D., Toda, T., Villavicencio, F., Wester, M., Wu, Z., and Yamagishi, J. (2016a). Voice conversion challence wepage. http://vc-challenge.org/ [accessed: 28. April, 2017].

Chen, L.-H., Liu, L.-J., Ling, Z.-H., Jiang, Y., and Dai, L.-R. (2016b). The ustc system for voice conversion challenge 2016: Neural network based approaches for spectrum, aperiodicity and f0 conversion. In *INTERSPEECH*, pages 1642–1646.

Chollet, F. et al. (2015). Keras. https://github.com/fchollet/keras.

Duxans, H. (2006). Voice conversion applied to text-to-speech systems. *Unpublished Ph. D. thesis. Universitat Politecnica. Barcelona, Spain*, 4:167.

Erro, D., Alonso, A., Serrano, L., Tavarez, D., Odriozola, I., Sarasola, X., Del-Blanco, E., Sanchez, J., Saratxaga, I., Navas, E., et al. (2016). Ml parameter generation with a reformulated mge training criterion participation in the voice conversion challenge 2016. In *Proceedings of the INTERSPEECH*.

Forsyth, D. (2012). Learning time series. Available at http://luthuli.cs.uiuc.edu/~daf/courses/CS-498-DAF-PS/Lecture%2018%20-%20Time%20series,%20DTW.pdf [accessed: 05. November 2016]. Lecture note from *Applied Machine Learning* at University of Illinois i Urbana-Champaign.

Goodfellow, I. J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F., and Bengio, Y. (2013). Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.421*.

Gopi, E. S. (2014). Digital Speech Processing Using Matlab. pages 73–93.

Gupta, S., Jaafar, J., Ahmad, W. F. W., and Bansal, A. (2013). Feature extraction using mfcc. *Signal & Image Processing*, 4(4):101.

Haykin, S. (2004). Neural networks - a comprehensive foundation. *Neural Networks*, 2.

Kain, A. and Macon, M. W. (1998a). Spectral voice conversion for text-to-speech synthesis. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 1, pages 285–288. IEEE.

Kain, A. and Macon, M. W. (1998b). Text-to-speech voice adaptation from sparse training data. In *ICSLP*, volume 98, pages 2857–2850.

Kain, A. B. (2001). *High resolution voice transformation*. PhD thesis, Oregon Health & Science University.

Kawahara, H. (2014). Straight speech analysis. Accessed: 2010-09-30.

Kawanami, H., Iwami, Y., Toda, T., Saruwatari, H., and Shikano, K. (2003). Gmm-based voice conversion applied to emotional speech synthesis.

Kominek, J., Schultz, T., and Black, A. W. (2008). Synthesizer voice quality of new languages calibrated with mean mel cepstral distortion. In *SLTU*, pages 63–68.

Krewell, K. (2009). What is the difference between a cpu and a gpu? https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/.

mlwiki (2017). Overfitting. http://mlwiki.org/index.php/Overfitting [accessed: 17. June 2017].

Mohammadi, S. H. and Kain, A. (2014). Voice conversion using deep neural networks with speaker-independent pre-training. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 19–23. IEEE.

Mohammadi, S. H. and Kain, A. (2016). A voice conversion mapping function based on a stacked joint-autoencoder. In *INTERSPEECH*, pages 1647–1651.

Mohammadi, S. H. and Kain, A. (2017). An overview of voice conversion systems. *Speech Communication*.

Morise, M. (2015). Cheaptrick, a spectral envelope estimator for high-quality speech synthesis. *Speech Communication*, 67:1–7.

Morise, M., Yokomori, F., and Ozawa, K. (2016). World: A vocoder-based high-quality speech synthesis system for real-time applications. *IEICE TRANSACTIONS on Information and Systems*, 99(7):1877–1884.

Moulines, E. and Charpentier, F. (1990). Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. *Speech communication*, 9(5-6):453–467.

Müller, M. (2007a). 4 Dynamic Time Warping.

Müller, M. (2007b). Dynamic Time Warping. *Information Retrieval for Music and Motion*, pages 69–84.

Nakamura, K., Toda, T., Saruwatari, H., and Shikano, K. (2012). Speaking-aid systems using gmm-based voice conversion for electrolaryngeal speech. *Speech Communication*, 54(1):134–146.

Pellom, B. L. and Hansen, J. H. (1999). An experimental study of speaker verification sensitivity to computer voice-altered imposters. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, volume 2, pages 837–840. IEEE.

Rhody, H. and Raqueno, R. (2003). Hierarchical data format. https://www.cis.rit.edu/class/simg726/lectures/HDF/HDF_Lecture.pdf [accessed: 30. May 2017]. Lecture note from Rochester Institute of Technology.

shuaijiang (2014). Straight. https://github.com/shuaijiang [accessed: 29. May, 2017].

Skymind (2017). Deeplearning4j updaters explained. https://deeplearning4j.org/updater [accessed: 17. June 2017].

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Stylianou, Y. (2008). Voice transformation. In *Springer handbook of speech processing*, pages 489–504. Springer.

Stylianou, Y. (2009). Voice transformation: a survey. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 3585–3588. IEEE.

Stylianou, Y., Cappé, O., and Moulines, E. (1998). Continuous probabilistic transform for voice conversion. *IEEE Transactions on speech and audio processing*, 6(2):131–142.

Svendsen, T. (2014). Applied signal processing - Autoregressive modeling of speech. pages 1–17.

Valbret, H., Moulines, E., and Tubach, J.-P. (1992). Voice transformation using psola technique. *Speech communication*, 11(2-3):175–187.

Wu, Y.-C., Hwang, H.-T., Hsu, C.-C., Tsao, Y., and Wang, H.-M. (2016). Locally linear embedding for exemplar-based spectral conversion. In *Proc. INTERSPEECH*.

Wu, Z. (2015). Spectral mapping for voice conversion. Master's thesis, Nanyang Technological University.

Wu, Z. and Li, H. (2013). Voice conversion and spoofing attack on speaker verification systems. In *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2013 Asia-Pacific*, pages 1–9. IEEE.

Zhang, X., Yao, J., and He, Q. (2009). Research of straight spectrogram and difference subspace algorithm for speech recognition. In *Image and Signal Processing, 2009. CISP'09. 2nd International Congress on*, pages 1–4. IEEE.

Appendix