DTU Civil Engineering
DTU compute

Deep voice conversion

02466 Project work

Bachelor of Artificial Intelligence and Data science

Deep voice conversion

Project work - Bachelor of Artificial Intelligence and Data science
June, 2020

By
Gustav Gamst Larsen (s180820), Lukas Leindals (s183920), Peter Grønning (s183922)

Cover photo: Gustav Larsen, 2020 with materials used under fair use from Stefani Reynolds, Philip Davali/Ritzau Scanpix, Ida Guldbæk Arentsen/Ritzau Scanpix & Pixabay

Abstract

The popularity of deep fakes (faking of voice or video with deep neural networks) are drastically increasing for entertainment as well as political interference's and when people are beginning to speculate more and more on the wrong uses of such a technology, thoughts of concern about identity as well as unethical uses starts to emerge.

Voice conversion seem to be more accessible as time progresses, but the theory and methods behind them are complicated and time consuming to understand. This paper wishes to describe the necessary theory needed to perform voice conversion and investigate how well state-of-the-art voice conversion models can be implemented. Furthermore, a discussion of the ethical considerations that need to be taken into account when manipulating with peoples voices.

This paper uses the state-of-the-art models of AutoVC [1] and StarGAN [2] to replicate these methods to perform convincing voice conversion. The voices that will undergo conversion will be political people in and formerly in power from Denmark and The United States. These conversions will then be tested on the public using a survey, creating a general "fooling rate" to investigate the general performance of the methods as well as measure the quality and similarity between original voices and their impersonator. The methods will also be tested on the amount of training data given.

The results showed that the AutoVC model performs best across the conversion categories fooling 13% of the participants against StarGANs 7%. The quality seemed to have an effect on the overall naturalness score, and the AutoVC had a similarity MOS of 2.14 while StarGAN had a similarity MOS of 0.84 on a scale from 0 to 5. It was not possible to conclude that the amount of training data had any significant effect on the performance of the models.

In conclusion this paper finds the AutoVC to be the best model for voice conversion. The results were, nevertheless, not as impressive as the literature suggested, which was partly due to lack of certain knowledge of the models and the idea of a world where our voices cannot be trusted still exists in the future.

Contents

1 Introduction

The rapidly evolving technology around deep learning and artificial intelligence is incredible to say the least, and with all of this exciting technology, we try to create a better tomorrow. But technologies that try to alter our reality or twist our minds can also seem frightening and evil to the human mind. Nevertheless, as Margrethe Vestager, the executive vice president of EU has said in a tweet: "*Artificial intelligence is not good or bad in itself: It all depends on why and how it is used. Let's enable the best possible use and control the risks that AI may pose to our values - no harm, no discrimination!*".[1] Certain technology can be used for mischief, but if we do not explore this technology we will not know how to eventually combat it in the future. A field where deep learning is being used for fun, but where media is speculating if it will be used for evil are the field of so called Deepfakes. The guardian made an article on the fact that Facebook has decided to "*ban deepfake videos in run-up to US election*", because they could mislead viewers to believe that some politicians said words and statements that they have never uttered. This happens as a video of the US government house speaker, Nancy Pelosi, had a deepfake video going around Facebook showing her going through a speech to what looked like she was drunk while doing it [3].

That video would be an example of both video and voice deepfakes. This paper will focus on the technology surrounding voice-related deepfakes. Voice-deepfakes can just as well as video-deepfakes create confusion in the human mind. Our voice is something we can recognise and can in some ways be seen as a fingerprint, defining who we are. Identity theft is highly illegal and imitating a person in power is highly unethical and illegal as well. In Chapter 2 the ethical considerations regarding voiced conversion will be discussed. Chapter 3 presents the data used in this project. Chapter 4 introduces theories behind the state-of-the-art models suggested in Chapter 5. Chapter 6 will provide an overview of the experimental design used, followed by Chapter 7 which will follow up on the experiment outcomes. At last, Chapter 8 will look into and discuss these results as well as possible enhancements for future related work.

1.1 Scope

The scope of this paper revolves around investigating state-of-the-art Voice Conversion (VC) models and will test them on the basis of training data used for natural voice conversion, as well as try to convert voices of famous speakers convincingly. In this paper the two state-of-the-art VC models, AutoVC [1] and StarGAN [2], will be implemented and tested. Furthermore, the results obtained will be compared to the results of the original papers. Conversion speech will be synthesised using different vocoders.
The success criteria of this project would be to implement the models in such a way that

[1] https://twitter.com/vestager/status/1230087490415087616

the final product would sound convincing to the human ear by testing our conversions with an experiment/survey.

1.2 Research Questions and hypothesis

In this paper, the purpose is to understand, implement and execute voice conversion in a limited time frame and to discover the best possible outcome using as few resources as possible. Research questions have been created to keep focus on the main goal and to stay in scope of the project:

- Q1: How do state-of-the-art models alter voices and how can these be implemented?
- Q2: How well do the state-of-the-art methods perform when converting the voices of famous speakers?
- Q3: What ethical and lawful restrictions exists in the field of deep fakes?

This paper hypothesises that it is possible to implement voice conversion from the state-of-the-art literature, and to do so in a convincing way.

1.3 Related work

This paper mainly focuses on two methods, AutoVC and StarGAN. These two methods will be heavily inspired from similar work ([1], [2]) with slight changes which will be described in the methods chapter. StarGAN-VC originates from the papers [2] and [4]. It is a GAN that tries to learn multiple mappings using only one generator, instead of a cross-domain model. In the first StarGAN paper ([2]) it is proven that StarGAN is capable of performing Non-parallel many to many voice conversion with reasonably realistic sounding speech with only several minutes of training [2]. Later the authors of [2] improved the model in regards to a better architecture and training objectives in [5]. However, this paper will focus on the original paper ([2]) for voice conversion. The AutoVC paper [1] focuses on zero-shot voice style transfer and auto-encoder loss. The AutoVC follows a auto-encoder framework and is trained on said encoders loss. The simple auto-encoder structure proposed actually outperforms StarGAN in non parallel many-to-many voice conversion tasks and is the first to perform zero-shot voice conversion [1].

2 Ethical considerations

Before jumping into the world of voice inspired deep-fakes, a greater scope of how a technology such as this can be manipulated for better and for worse has to be realised. The technology is new, and has already been used for worse. An article from the wall street journal described how fraudsters used this technology to mimic the voice of a CEO of an U.K.-based energy firm, and make his/her assistant transfer a lot of money to a certain bank account, convincing the assistant that it was the Hungarian department of the firms bank account [6]. This is one miss use of this newfound technology and it can only be assumed that more people in the future will exploit deep fakes for mischief and crime. The ethical consideration for what can be shared and cannot be shared to violate some ethical standards will be discussed and thought about in this chapter.

2.1 Data ethics

To implement voice conversion a lot of data has to be used. Since the final product of this paper wishes to be able to convert the voices of famous people such as Donald Trump, videos or other sources of the presidents voice would be needed to teach the models his acoustic features in order to reproduce these. Care needs to be taken when using sensitive data such as speech to avoid violating the speakers rights. Many sources of speech data such as movies and songs are mostly under copyright laws and not available for use, fortunately many data sets are available and have already been anonymised, meaning general data protection regulation (GDPR) would not be violated. This paper uses the voices of former and present world leaders to execute voice conversion. Data is obtained from websites putting political speech available legally.
However, when dealing with a subject such as voice conversion, obeying copyright laws and GDPR is not necessarily enough to act ethically correct.

2.2 Unethical complications

Deontology
Deontological ethics is the ethical theory based on the thought that the morality of an action is more important than the consequences of said action[1]. This project could heavily be discussed under Deontology as such a product could use the voice of a person as an object to accomplish a goal. When combining ethics and artificial intelligence, we mostly talk about how the technology and algorithms can come out biased in a way that is unethical (e.g. the COMPAS algorithm[2]), but for this project it is more about the uses that such a technology can be used for, if gotten into the wrong hands. In the article [6] we see that the fraudsters used the CEO as an object to reach the goal, money. In Chapter 1 the possibility of election interference's was also named, again being

[1] https://en.wikipedia.org/wiki/Deontological_ethics
[2] https://en.wikipedia.org/wiki/COMPAS_(software)

an example of how voice deep fakes can be used in an unethical manner. The actions in these examples are fraud and manipulation which both are considered unethical in the public opinion. However, from a more extreme deontological point of view, voice conversion in itself could be considered unethical, since the action of imitating a voice might not be morally justified. A persons own voice is such a highly integrated part of their outward identity that imitating a voice can be seen as huge violation of that individuals rights to have their own identity. Deontology could be used in this situation to say that creating a deep fake is unethical in the first place, and therefore deep fakes should never be created, whatever the end-goal/consequences are from creating it in the first place. No matter how beneficial the end-goal might be a voice and hence an identity will be used as an object to reach this goal.

Principle of rights and Consequentialism
Consequentialism can be seen as the opposite of deontology. Here the consequences are seen as the ultimate basis for judgement, not the actions[3]. The technology of VC is not only harmful it can also be put to good use; restoring the original voice of speech impaired, simplify communication and make voice assistants more human. If the consequence of a deep fake is, that a dangerous man is not elected president, is it bad thing? If the consequences are good, VC can not be seen as unethical in the eyes of a consequentialist. If your will is good and the outcomes are good, why should your freedom to do VC be taken away by e.g. ilegalising it? The principle of rights was formulated by Immanuel Kant from the idea that the people entrusted the government to make laws and since the people gave the government that freedom, the government should not be able to make laws restricting the freedom of the public. [4] This include the peoples freedom to explore and use new technology in an ethical way.

Although we are not there yet, the media and the general public seems to find deep fakes amusing but also rather dangerous, and it is not difficult to see the potential dangers coming up ahead. But with the ethical standpoint going in opposite directions and the possible consequences of VC remains unknown, it is difficult to make laws against it. This project is entirely for learning purposes as well as realising how easy it may be to produce such technology. None of the authors of this paper condone or wish to inspire unethical uses.

2.3 Who owns your voice?

> *Your voice is such a personal thing. It's like a fingerprint. It's your intellectual property. And suddenly, it's no longer yours. And I'm not blaming anyone — we signed a contract to do the work — but I don't think we had any clue of the ramifications.* - Susan Bennett (The original voice of Apple's Iphone voice assistant, Siri.)

When impersonating others voices, you will need a recording of their voice to begin with. These recordings can go under copyright and privacy laws. According to the new EU

[3] Consequentialism
[4] Wikipedia: Immanuel Kant

GDPR law, both the person being recorded and recorder must consent to the recording or else it is an invasion of privacy and is considered an illegal action. But here is the tricky part, if consent to record your voice is given to someone, do they own your voice? This grey area has never really been adapted into copyright laws as of yet. Because it surely seems like we are owners of our own voices, but when is it okay for others to use yours? Susan Bennett which is quoted in the beginning of this subsection did a recording gig for a company formerly known as ScanSoft to do a lot of recordings, but she did not know that her voice was going to become the voice of one of the most popular phones in the world. Bennett also states that she gave them permission to "use her voice anywhere", having no idea what the recording would be used for.

This raises the question, if someone has given their consent to being recorded and that recording becomes public domain, does the public then own your voice?

2.3.1 Who is liable?

In law theory, this can still be quite tricky. If we look at a similar subject where many are beginning to see the trickery of this kind of concept and the legal repercussion there may be, it could be in deep fake pornography. Here it is difficult to find the victim, wrongdoer and liability. Is the victim the one being put on a naked porn actor or actress, or is it the person(s) acting that is under attack? The individuals in the porn did not give their consent to be manipulated by AI, they could raise the claims. Is the wrongdoer the one who created the original video, the deep fake or the one who uploaded the video? And is the liability in the ones creating the video, distributing it and/or storing it?

The victim in porn could both be the target as well as the source. When thinking about it, the most obvious victim when looking at voice deep fakes would be the target, the one faked to say potentially bad stuff. If someone copied their ex-lovers voice to leave a voice message to all of his/her friends saying she/he hates them, the victim in this case would be the ex-lover. But would there necessarily be any legal way to ban this. The problem lies that banning a deep fake voice could be seen as censorship against the freedom of speech. If the sound was from a recording where consent was given, there is not much you can do in this scenario[7].

Who is the wrongdoer? As of now, it is not illegal to make deep fakes. From an deontological standpoint, copying someones voice for personal gain would be unethical, but it is not illegal to do so. This also complicates further who has the liability, since there legally does not seem to be a violation of any laws.

This leaves us in a blind alley. It seems as of now, there is no legal repercussions for using any ones voice for personal use as long as they have given consent to be recorded. This could easily explain how Scansoft could record Bennett's voice without her having any idea of the scale in which those recordings were to be used, because in reality they did not need to tell her.

2.4 Ethical standpoints

This paper also collects some data to be able to do voice conversion, as well as try to evaluate the conversions using an experiment. The voice conversions will be done merely to evaluate the current state-of-the-art models and not for any other purposes. For the experiment, data collection is also needed to evaluate these. The data is collected under a set of rules the authors wish to work under, described and formulated by dataethics.eu [8]. The rules that are relevant to this paper are as follows:

- **Data minimisation:** Not all data is relevant data. To gather greats amount of unnecessary data can be dangerous if hacked or breached. Therefore only the relevant data for the experiment is being collected to drive conclusions and perhaps find co-founders along the way.
- **Anonymisation:** The data collected should not in anyway be able to be traced back to the person the data is gathered from. It should not be possible to trace back to the exact person who has filled out the experiment.
- **Consent:** It is important that the participant is aware of the usage of their data and that they consent to it being using for research purposes. If a participant finds out later about other uses of certain data, it can be crucial to their way of experiencing experiments in the future and may scare them away for ever participating again.
- **Storing the data:** The data should be stored in such a way that no one that should get hold of it can. An optimal way should be by downloading the data locally as soon as possible and then delete it from its online location.

3 Data

3.1 Voice

Voice is sounds made by humans (or animals) in the vocal tract. The simplified story of human speech generation is as follows; air is pressed from the lungs through the vocal cords forcing them to vibrate, which creates a voice with a specific pitch (the auditory sensation of a wave frequency[1]). The articulators such as the tongue, lips and cheeks are then used to form and filter the voice into sounds which in the listeners ears are decoded into meaningful speech[2].
Every single person has his or hers unique speaking characteristics - average pitch, articulation, speaking rate etc. which enable us to distinguish between persons solely based on their speech. Sounds (including speech) can be quantified into four parameters: frequency (often measured in Hz), energy / power (often measured in dB), time (s) and the aperiodic component of the signal.
When a continuous sound is sampled/recorded correctly (with respect to the Shannon/Nyquist sampling theorem) no information is lost and one is able to fully reconstruct the signal [9]. The analogue audio signal is often depicted as a waveform as shown in figure 3.1.

It is not unthinkable that in the future many-to-many VC is implemented, making anyone able to sound like another person from a catalogue of voices with the press of a button. To implement this, voice data on a target speaker is needed. Therefore choosing celebrities would be ideal since clips of interviews exists in lots, and could easily be used. The ethical considerations which was discussed in chapter 2 have been taken into account when choosing target speakers.

3.2 Data collection

Before voices can be converted data is needed. There is multiple ways to the approach of acquiring data for voices either by recording themselves or finding data online. This section discusses what the authors ended up doing and the legal hurdles along the way.

3.2.1 The VCTK-Corpus

The main data set used in this paper is the VCTK-Corpus under the Open Data Commons Attribution License (ODC-By) v1.0.[3] The corpus includes speech data by 109 native English speakers (both males and females) with various accents. Each speaker reads about 400 sentences resulting in about 44 hours of utterances. All speech data is recorded in a similar setup in a hemi-anechoic chamber, with 96kHz sampling frequency at 24 bits [10]. In this paper the VCTK-Corpus is mainly used for training the models.

[1] Wikipedie: Pitch (music)
[2] Wikipedia: Human Voice
[3] Open Data Commons Attribution License (ODC-By) v1.0

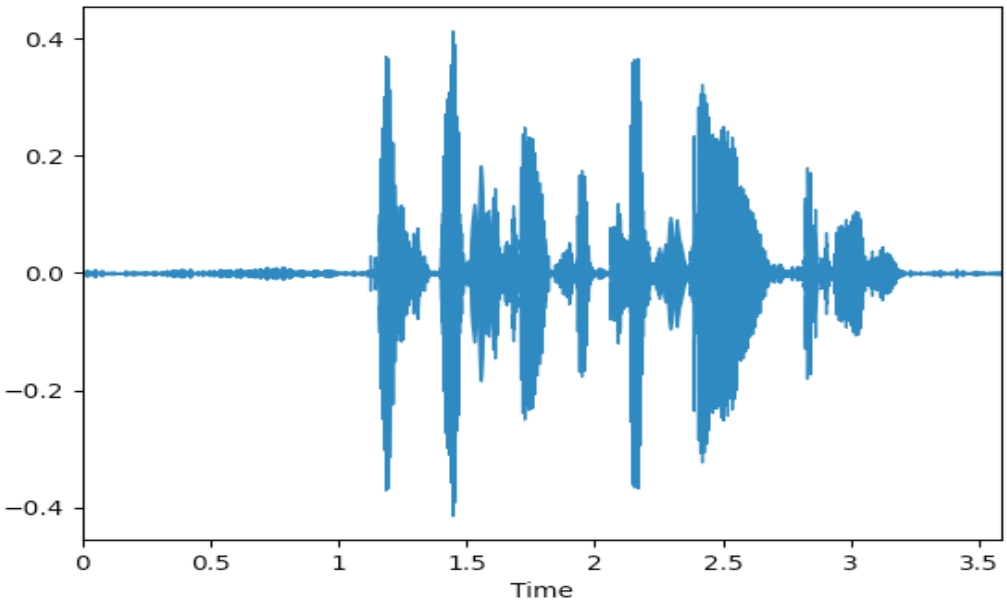

Figure 3.1: Waveform of a random sample from the VCTK Corpus with the amplitude as the y-axis, created using the librosa python library

The VCTK-Corpus is also used for voice conversion in the literature papers [1] which justifies the use in this paper as well.

3.2.2 Accessing Data

To test on more speakers like celebrities, data needs to be obtained from interviews, movies or like wise. However, copyright laws can have a great affect on how much data that can be acquired if any. One large source of such data is YouTube, which contains billions of videos with the voices of famous speakers. YouTube's terms of service states: *"It is not allowed to: Access, reproduce, download, distribute, transmit, broadcast, display, sell, license, modify or otherwise use any part of the Service or any Content except: a) as specifically permitted by the Service; (b) with the prior written permission of YouTube and, if applicable, the respective rights holders or (c) as permitted by applicable law;"*[4]. This means that the easy choice to access sound files from celebrities is illegal, and will therefore not be used for data gathering. To convert to a celebrity voice, data will therefore have to be obtained in another fashion. Luckily a website with non-copyrighted audio files was found.

3.2.3 Political Speakers

The data used for voice conversion was found from recorded speeches from American and Danish politicians. The voices of the 4 last sitting prime ministers of Denmark

[4] Youtube Terms of Service

(Mette Frederiksen, Lars Løkke Rasmussen, Helle Thorning Schmidt and Anders Fogh Rasmussen) were used and consisted mostly of their new years speeches to the nation (one video was from a COVID-19 press meeting) which were downloaded freely from the ministry's own homepage [11]. The speeches in English was taken from former US president Barack Obama and his wife and former first lady Michelle Obama, former secretary of state Hillary Clinton and sitting US president Donald Trump. These speeches were downloaded from an online speech bank website, Americanrhetoric.com, which hosts audio clips of some of the most important and memorable speeches from American politics through the years. Americanrhetoric has some soundfiles that are under copyright which cannot be downloaded, therefore the non-copyrighted files were used. The files downloaded will not be possible to download for the public through this projects GitHub repository, but everyone could in theory go download the same soundfiles as used in this paper from the two websites [12].

The data is then processed into 5 second sound files to be used for training and testing of the StarGAN and AutoVC model which will be described in the Chapter 5.

4 Methods

4.1 Types of voice conversion

4.1.1 Problem formulation

Before getting started a formal definition of voice conversion and its assumptions are needed. Following the example in [1] and [13] a speech segment $\mathbf{X}_k = (x_0, \ldots, x_{T-1})$ of T time dependent features belongs to the speaker k. X_k could for instance be a spectrogram.
It is assumed that each speaker has a unique speaker identity U e.g. average pitch, speaking frequency etc. U is represented as random variable drawn from the speaker population $p_U(\cdot)$. U_k is the speaking 'style' of the speaker k. Following is assumed with regards to speaker identity.

Assumption 1 *Given two speech segments X_a and X_b. If $a = b$ then $U_a = U_b$ and if $a \neq b$ then $U_b \neq U_a$. I.e. speech from the same speaker has the same characteristics and speech of different speakers has different characteristics.*

Furthermore the content of speech i.e. the utterance, is denoted Z and is a random process. Z_i is e.g. a specific sentence. The speech segment $\mathbf{X}_k$ is thus a random process sampled from the 'speech distribution' conditioned on U and Z resulting in speech of the speaker k.

$$\mathbf{X}_k \sim p_X(\cdot|U_k, Z_i) \tag{4.1}$$

Now given a source speaker s and a target speaker t, the main goal of voice conversion is to learn a mapping function, which transfers the content of the source to the target but maintains the identity of the target.
The ideal voice conversion is defined as follows: given two sets of variables (U_s, Z_0) and (U_t, Z_1) and a mapping function $p_{X_s \to X_t}(\cdot)$ the set (U_t, Z_0) is created and results in $\hat{X}_t$ as were it sampled from the true speaker distribution $p_X(\cdot|U_t, Z_0)$

$$p_{X_s \to X_t}(\cdot|(U_s, Z_0), (U_t, Z_1)) = p_X(\cdot|U_t, Z_0) \tag{4.2}$$

4.1.2 Parallel and non-parallel

The parallel voice conversion framework is a type of voice conversion in which the same utterance from a target speaker and a source speaker is used. This gives us the easiest way to make voice conversion, but also the most limiting, since we are taking the same words and converting the voices and utterances of exactly these words, making the model not very great at converting new utterances.
The non parallel is a bit more valuable in its outcome, since here conversion is performed using different utterances from the target speaker and source speaker. From this method, it should be possible to have any kind of target speaker utterances data and potentially be able to speak as the target speaker with any sentences desired[14].

4.1.3 many-to-many / one-to-many etc.

In data science, one-to-many is described as a record in one table which is related to many in another table. Whereas many-to-many relationship is multiple records in one table are related to multiple records in another table. In voice conversion table one would be the source speakers and table two the target speaker. So one-to-many means having one source speaker whom it is possible to convert into many target speakers, and vice versa for many-to-one. Many-to-many voice conversion wants to make many source speaker sound like many target speakers. As one could imagine, the network would need to train longer with more data when it needs to perform many-to-many than one-to-many, since learning multiple source speakers and how to convert them into multiple target speakers is not supposedly easy. In this project capabilities of many-to-many voice conversion will be tested.

4.2 Feature Extraction

Speech data is represented as analogue sampled signals in a time-series waveform as seen in figure 3.1. The data in its raw form is rather troublesome to work with, since each data point represents the amplitude of the signal at a specific time. The acoustic and linguistic features are extracted first in most (if not every) paper regarding VC ([1], [2]) and this paper is no different. In the paper describing AutoVC the speech features used represent the frequency and power of the signal in a closed time interval [1]. Concatenating these intervals into a matrix $\mathbf{X} = (\mathbf{x_0} \dots \mathbf{x}_{T-1})$ where each column represent a feature vector $\mathbf{x_t}$ at time-interval t yields the final input to the AutoVC model. To do this a mapping from the discrete time domain unto a discrete frequency domain is needed. A common practice of this is the Fourier transformation described in section 4.2.1.

A different approach used in [2] is to decompose speech into 3 parts: *fundamental frequency* (pitch) F_0, *spectral envelopes* and *aperiodicity*. This approach is also the one used for the vocoder WORLD [15]. A spectral envelope describes the vocal tract characteristics and aperiodicity refers to the aperiodic elements of speech i.e. sound emitted without use of the vocal cords, e.g. tongue clicking.

4.2.1 The discrete Fourier transformation

The Fourier transformation is named after the french mathematician Jean Baptiste Joseph Fourier who is famous for his studies of approximations of periodic functions as a sum of sines and cosines - this sum is referred to as a Fourier series. Each part x of the series can be described by Eulers formula (eq. 4.3)

$$e^{in\gamma x} = cos(n\omega x) + isin(n\omega x) \tag{4.3}$$

i is the imaginary unit and $\omega = \frac{2\pi}{N}$.

Now, the data used in this paper is discrete sequences $\{\mathbf{x}_n\} = \{x_0, \dots, x_{N-1}\}$ of N data points sampled with a sampling rate F_s from the continuous signal. Due to this the

discrete Fourier transformation (DFT) (eq. 4.4) has to be used, which 'measures' the amount of each frequencies in a signal.

$$\hat{x}_k = \sum_{n=0}^{N-1} x_n e^{-i\omega kn} \tag{4.4}$$

where k is the number of possible frequencies in the signal. The sequence $\{\widehat{\mathbf{x}}\} = \{x_0, \ldots, x_{N-1}\}$ consists of N complex numbers each corresponding to a frequency k. It can be interpret as the cross correlation between the signal and the frequency. By Fouriers theorem the signal can be understood as a linear combination of harmonic functions meaning the DFT takes the projection of the signal unto a basis of these.
The DFT can be expressed as a linear system of equations

$$\mathbf{U}\mathbf{x} = \widehat{\mathbf{x}} \tag{4.5}$$

where $\mathbf{U}$ is a $N \times N$ symmetric (and if scaled correctly) basis matrix.

$$\mathbf{U} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & e^{i\omega} & e^{-i2\omega} & \ldots & e^{-i(N-1)\omega} \\ 1 & e^{-i2\omega} & e^{-i4\omega} & \ldots & e^{-i2(N-1)\omega} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e^{-i(N-1)\omega} & e^{-i2(N-1)\omega} & \ldots & e^{-i(N-1)^2\omega} \end{bmatrix} \tag{4.6}$$

Row k represents N samples from a period of the harmonic function with frequency $\frac{2\pi k}{N}$. A signal with frequency $\frac{2\pi k}{N}$ will therefore have a high cross correlation with row k resulting in a complex number with a large magnitude (absolute value) - the magnitude of the $\hat{x}_k$ scaled with the number of samples N is the amplitude A_k of the corresponding frequency

$$A_k = \frac{1}{N}\sqrt{Re(\hat{x}_k)^2 + Im(\hat{x}_k)^2} \tag{4.7}$$

With the DFT one can find the amplitude of a frequency present in a signal, the signal just has to be sampled with a sampling frequency equal to or above the Shannon-Nyquist sampling rate. Frequencies with $k < 2F_s$ cannot be reconstructed, where F_s is the sampling frequency, meaning the sampling frequency has to be at least twice as large as the highest frequency in the signal. The squared amplitude is known as power. Doing this for every frequency in the spectrum yields the power spectrum.
There is still one issue with DFT which needs to be addressed: The entire time domain of the signal is transformed to be discrete, whereas it is desired to extract features of short intervals of time.

Short Time Fourier Transformation
In order to extract the time dependent features short time Fourier transformation (STFT) is applied. STFT is just like the DTF but it uses a window function w which is positive within and zero-valued outside of a specific interval m[1], e.g. the Hann Window function depicted in figure 4.1

[1] Wikipedia: Window Function

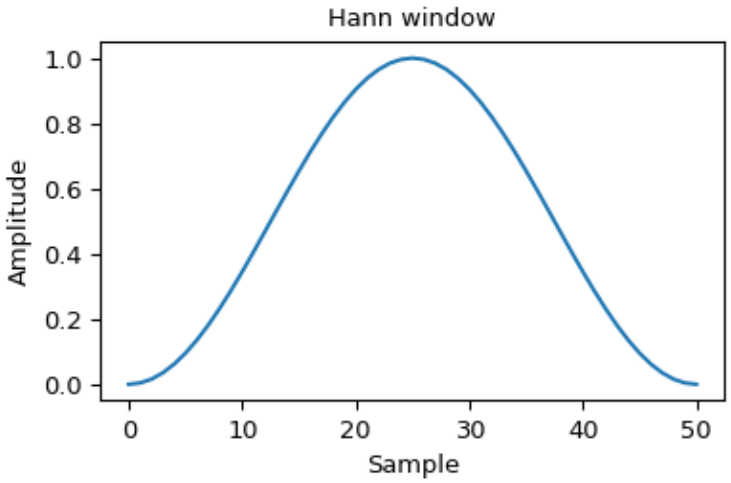

Figure 4.1: Hann Window function. From Olli Niemitalo - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=77818388

The signal is broken into intervals/frames of equal lengths, e.g. 5 ms or 1024 samples. A window function with center in the middle of the interval is then applied. The length of the window function varies, but it is most common to have intervals and a window function of the same length. This allows the signal to be broken into pieces of shorter time intervals, in which the energy of different frequencies can be captured. The STFT looks as follows

$$\hat{x}_{k,m} = \sum_{n=0}^{L-1} x_n w(n, L) e^{-i\omega kn} \tag{4.8}$$

where L is the length of the interval. Say, the STFT frames are set to $N = 1024$ and likewise for the window function. The DFT is then applied on the interval $m_0 = [0 : N] = [0 : 1024]$ after which the interval is shifted with a 'hop length' R, for instance $R = 256$ forces the next interval to be $m_1 = [0 + R : N + R] = [256 : 1280]$ resulting in a slight overlapping of intervals to 'smoothen' the transition to the next interval (see figure 4.2).

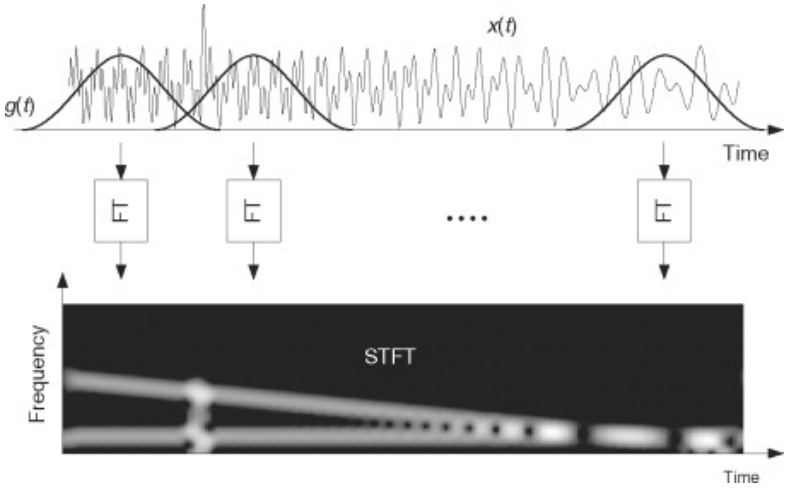

Figure 4.2: STFT example. Notice the overlapping windows. From Nasser Kehtarnavaz: short-time-fourier-transform

In short the STFT breaks the signal into $\lfloor \frac{N}{R} \rfloor$ overlapping time intervals and applies the DFT on each, resulting in a matrix with $\frac{L}{2} + 1$ rows, each corresponding to a discrete frequency and $\lfloor \frac{N}{R} \rfloor$ columns each corresponding to a time interval. The element $\hat{x}_{k,m}$ is the complex number of the k'th frequency at the m'th time interval.
If the window interval is set to be 1024, the hop length 256 and the sample rate 16kHz, then each time frame (STFT window) captures $\frac{1024}{16000} = 64ms$ of the signal and there is a $\frac{256}{16000} = 16ms$ step from the beginning of each window interval.

The result of the STFT is a spectrogram, which consists of the spectral envelopes describing the vocal tract characteristics (see figure 4.3).

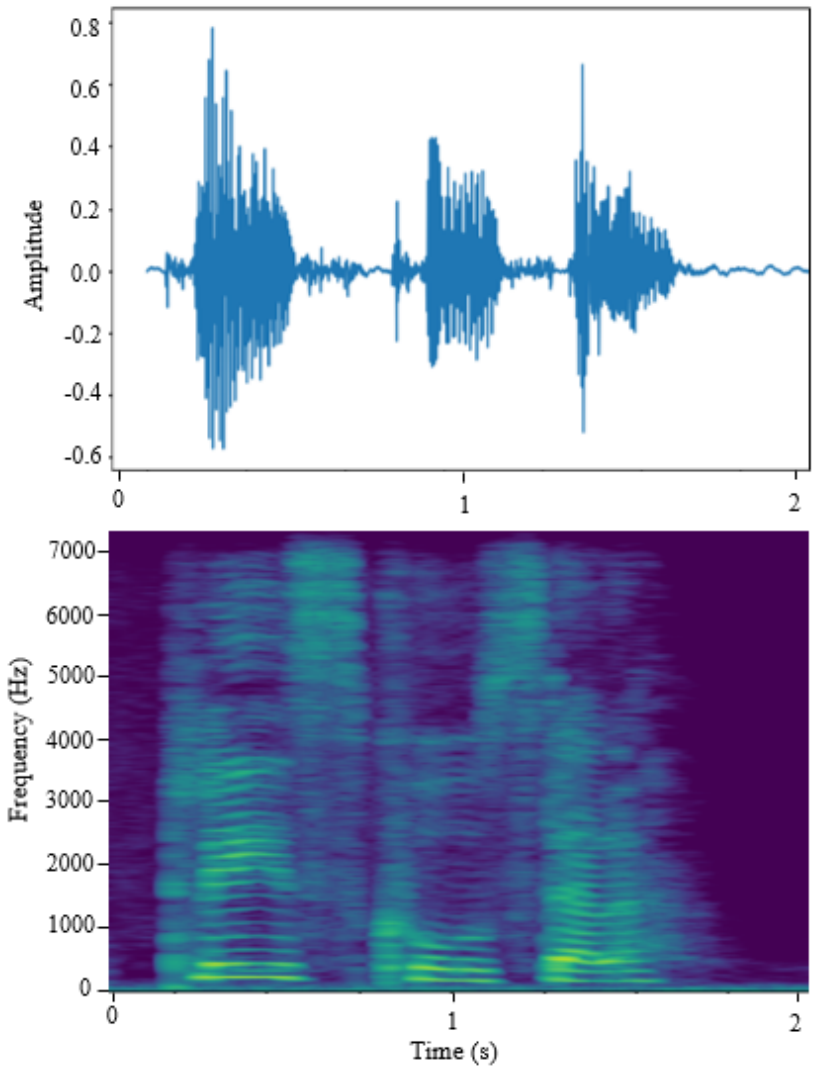

Figure 4.3: *STFT*: A Waveform and its' corresponding spectrogram

4.2.2 The Mel Scale

Given 1000 samples of a 1 second signal it follows directly from the Shannon-Nyquist theorem that frequencies up to 500 Hz can be reconstructed. Fortunately the fundamental frequency of male voice is from 85 to 180 Hz and from 165 to 255 for females[2], which makes it possible to sample voice signal at a relatively low sampling frequency without distorting the underlying information. However the Hz scale is linear and not very analogue to the human perception of sound. A classic approach is to transform Hz into the non-linear mel-scale which is more closely related to the auditory sensation

[2] Wikipedia: Voice Frequency

in human ears, since it is build around pitches being judged by listeners to be equal in distance from one another[3], using equation (4.9) [16]. Transforming the STFT unto the mel-scale yields in a mel-spectrogram that looks similar to the spectrogram in figure 4.3 except the frequency is replaced with mels.

$$m = 2595 \log_{10}\left(1 + \frac{f_{Hz}}{700}\right) \tag{4.9}$$

$$f = 700(10^{\frac{m}{2595}} - 1) \tag{4.10}$$

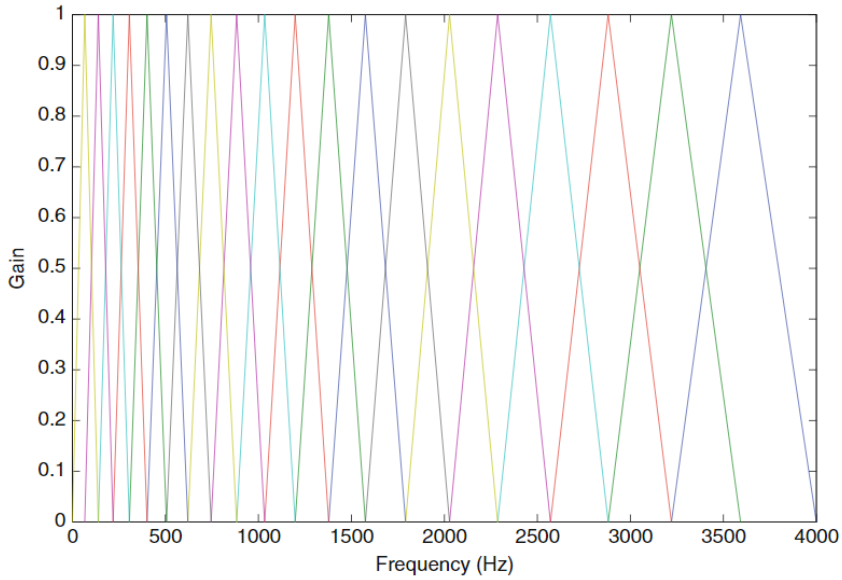

Figure 4.4: Mel filterbank. Each triangle represent one of the mels in the filter bank, i.e. a row in the matrix **H**. From [17]

.

The Hz to mel transformation is often made easy by computing a triangular mel filterbank. This is a set of m triangular filters, with m being the number of mels of the signal. This filterbank is computed with equation (4.11) and visualised in figure 4.4.

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \frac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leq k < f(m) \\ 1 & k = f(m) \\ \frac{f(m+1) - k}{f(m+1) - f(m)} & f(m) < k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{4.11}$$

[3] Wikipedia: Mel Scale

where k corresponds to a frequency, m is mel, and f is found in (eq. 4.10). This filter bank $\mathbf{H}$ can be represented as a $M \times N$ matrix, with M being the number of mels in the filterbank and N being the number of frequency bands. Given a discrete STFT signal $\mathbf{X}$ as a $N \times T$ matrix, with N being the number of frequency bands and T the number of time intervals, the mel scaled signal (often referred to as the mel spectrogram) can be found as the matrix-matrix product [17]:

$$\mathbf{X}_{mel} = |\mathbf{X}|^2 \cdot \mathbf{H}^T \tag{4.12}$$

4.2.3 Fundamental Frequency Estimation

There exists several techniques for estimating the fundamental frequency F_0 of a voice. The technique used in [2] and this project is an algorithm known as Harvest [18]. When estimating the fundamental frequency of a speech segment the segment is divided into frames of a few ms and each frame is analysed for F_0. Combining F_0 for each frame yields what is known as the F_0 contour (see figure 4.5). What Harvest does is to apply a set of bandpass filters with different center frequencies ω_c to each frame. The filter is designed to detect the fundamental frequency candidates. The nearer a candidate is to ω_c of the filter the more likely the candidate is to be the fundamental frequency. Harvest only includes candidates in the range of $w_c \pm 10\%$ for the further analysis. Furthermore, if the fundamental frequency is discovered by a filter it will also be discovered by filters with similar center frequencies. Candidates which are not discovered within a bandwidth of $\omega_c \pm 10\%$ are rejected. Due to the periodic nature of voiced speech the F_0 contour is not expected to change rapidly. If a change from one frame to another happens above a threshold it is counted as an unvoiced part of the speech segment and as a result removed [18]. The F_0 contour is also refined and the most reliable F_0 candidates are chosen as described in [18].

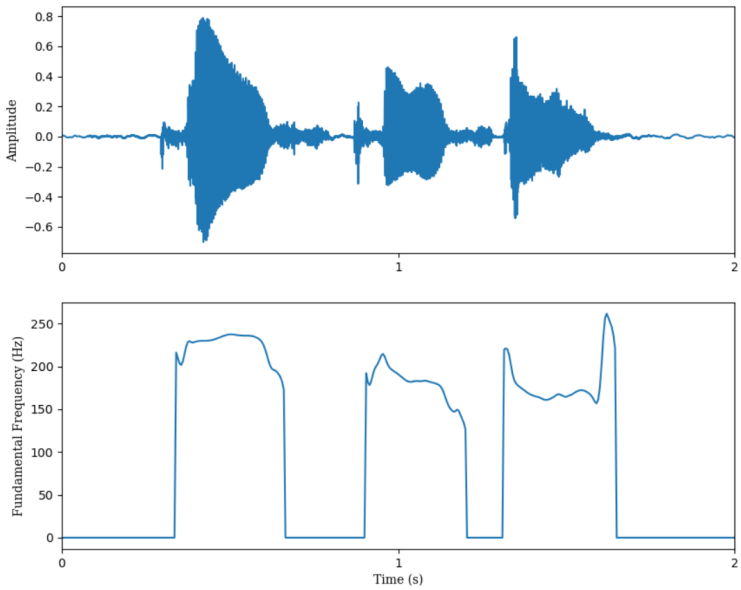

Figure 4.5: *Harvest Algorithm:* Input waveform and the output F_0 contour of the Harvest Algorithm. Made with PyWORLD.

4.2.4 Spectral Envelope Estimation

Instead of using STFT to compute the spectral envelopes the authors of [2] and [15] use the CheapTrick Algorithm described in [19]. CheapTrick proposes additional ideas to DFT in order to improve the estimation of the spectral envelopes. Instead of setting the window length to the same number of samples as done in standard DFT, the window length adapts to the F_0 contour by defining the length of it as $3T_0$ with T_0 being the period of F_0 in the given frame. DFT is performed on the windowed signal and the power spectrum for each frame is calculated. The result $P_s(\omega)$ (ω is frequencies) is then filtered to avoid any zeros in the power spectrum [19]. The final output of the algorithm is found in equation (4.13) [19] and visualised in figure 4.6.

$$P_l(\omega) = exp\left(\mathcal{F}\left[\frac{sin(\pi f_0, t)}{\pi f_0 t} \cdot (q_0 + 2q_1 cos\left(\frac{2\pi t}{T_0}\right)) \cdot \mathcal{F}^{-1}[log(P_s(\omega)]\right]\right) \tag{4.13}$$

with $q_0 = 1.18$, $q_1 = -0.09$, $\mathcal{F}$ denoting DFT and $\mathcal{F}^{-1}$ its' inverse:

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} \hat{x}_k \cdot e^{i\omega kn} \tag{4.14}$$

with $\omega = \frac{2\pi}{N}$.

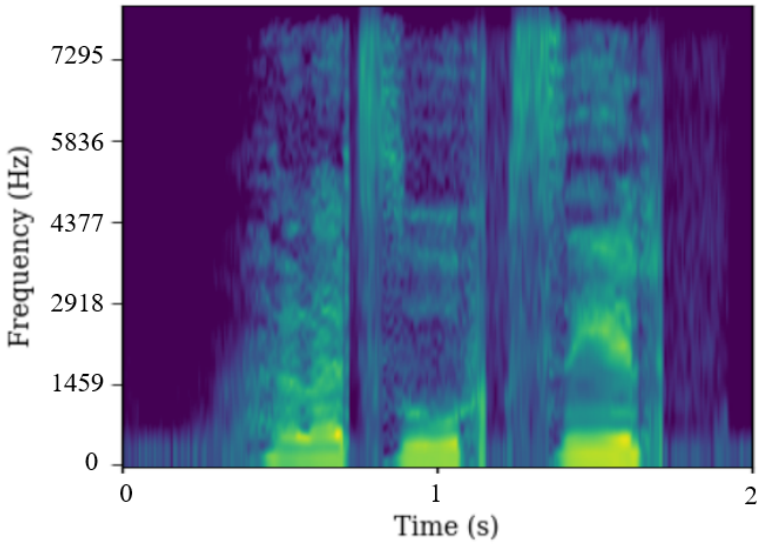

Figure 4.6: *CheapTrick:* The output spectral envelopes for each frame for an input time series, where each column is a spectral envelope. Made with PyWORLD.

Adjacent elements within each spectral envelope (as in figure 4.6) tend to be highly correlated indicating that they carry a lot of unnecessary information. To counter this a discrete cosine transformation is applied to each spectral envelope. This is done by first transferring the spectral envelopes onto a mel basis (using a filter bank), represent it on a log-scale and afterwards apply a discrete cosine transformation (DCT). The DCT is related to the DFT, however, it only uses real numbers. By applying DCT the spectral envelope is compressed in a way, in which the result still carries the most important information. The result is known as mel frequency cepstrum coefficients (MFCC). This compression is done in the following way, with $s(m)$ being the spectral envelope element at mel m [17]:

$$c(n) = \sum_{m=0}^{M-1} s(m) cos\left(\frac{\pi n(m+0.5)}{M}\right), n = 0, 1, 2, ..., C-1 \tag{4.15}$$

where C is the number of MFCC's. The results is shown below in figure 4.7 when using equation (4.15) for each time signal.

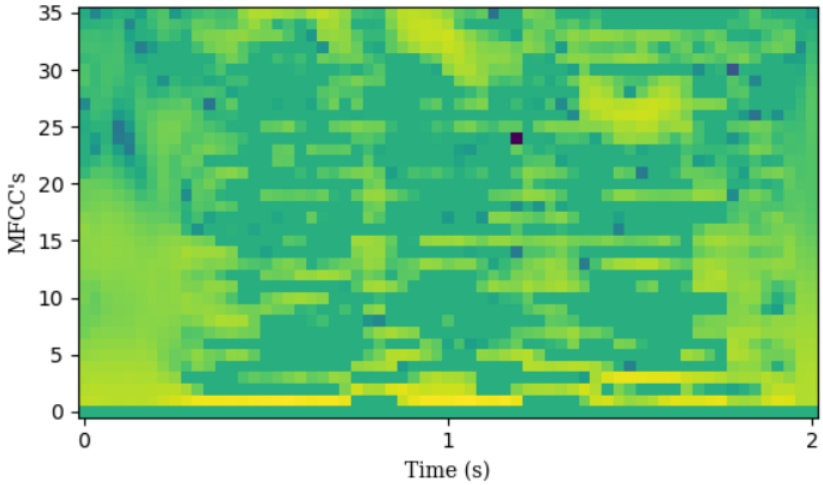

Figure 4.7: The MFCCs for the spectral envelopes shown in figure 4.6, where each column (spectral envelope) represents the logarithmic (for visualisation purposes) output of equation (4.15) for each MFCC used. Made with PyWORLD.

4.2.5 Aperiodicity estimation

Aperiodicity is an vital part for modern speech synthesising algorithms such as WORLD and STRAIGHT [15]. The core idea behind these vocoders is to mix vocal tract information (spectral envelopes) with the aperiodic components (aperiodicity). There exists various aperiodicity estimators, but the concept of aperiodicity estimation used in StarGAN and hence this project comes from the D4C algorithm [20]. In [20] they measure aperiodicity for several frequency bands at voiced sections of speech defined by F_0 contour information and define the frequency band aperiodicity as the power ratio between the total power of the signal and the power of the sine wave with the corresponding band frequency. This means aperiodicity is measured in dB.
A visualisation of the relationship between input waveform, spectral envelopes, F_0 contour and aperiodicity bands found in figure 4.8.

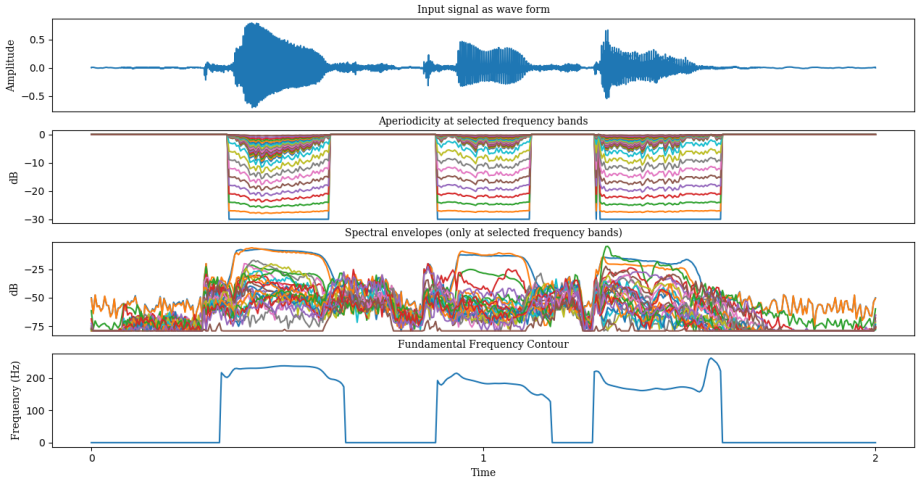

Figure 4.8: The relationship between *input waveform*, *Aperiodicity*, *spectral enelopes* and F_0 contour (from top to bottom). The 3 lower plots are all created with the PyWORLD module with the algorithms *D4C* [20], *CheapTrick* [19] and *Harvest* [18] respectively.

4.3 Building Blocks of Networks

This section introduces the main building blocks of the models used in this paper and will give a basic understanding of these.

4.3.1 Up- and down sampling

Up- and down sampling are simple tools to extract important latent information and discard what is less crucial. It can be understood as a compression of data. The most simple down sampling technique when working with sequential data (such as sound) is to just sample every k'th time-step. Up sampling is afterwards applied by copying these time steps to scale the data up to dimensions matching the original input.
In deep learning, up- and down sampling can be applied with linear layers of different input/output size - down sampling when passing information to layers of lower dimensions and up sampling when passing information to higher layer dimensions.
Both RNN including LSTM and CNN do down and up sampling, e.g. in regular CNN convolutions without padding do down sampling and transposed convolutions do upsampling. Hence these LSTM and CNN layers works well when the goal is to squeeze out information and afterwards upscale using only the most important information.

4.3.2 Convolutional Neural Network

A convolutional neural network (CNN) is an artificial neural network (ANN) that is especially good to detect patterns. They use kernels (or filters) which are able to detect different patterns such as edges or corners in images. This can also be done with speech if it is represented as a spectrograms or MFCC's.

A kernel is a mathematical object often a vector, a matrix or a tensor depending on the dimensions of the task. They come in a great variety, each suited to a special detection task.[4] The kernel is moved over the input and computes the cross-correlation, which is the sum of element wise multiplication of kernel and input at each position. Below is a 1d example of cross correlation with a kernel ($\star$ denotes cross correlation) [21]. Here a kernel of size 3 is used on an input of size 5 with a stride of 1. Thus the first element of the output becomes the sum of the element wise multiplication between the kernel and the first 3 values of the input. For the second element of the output, the kernel is multiplied element wise with the 2nd to 4th element and so on.

$$\underbrace{\begin{bmatrix}1\\2\\0\\3\\4\end{bmatrix}}_{input} \star \underbrace{\begin{bmatrix}1\\0\\1\end{bmatrix}}_{kernel} = \underbrace{\begin{bmatrix}1\cdot 1+2\cdot 0+0\cdot 1\\2\cdot 1+0\cdot 0+3\cdot 1\\0\cdot 1+3\cdot 0+4\cdot 1\end{bmatrix}}_{output} = \underbrace{\begin{bmatrix}1\\5\\4\end{bmatrix}}_{output}$$

In the 2d case, imagine a matrix starting from the upper left corner of the input moving out it as a typewriter. CNN's treat these kernels as weights in a neural network and thereby learns which kernel to use when training. Often many kernels are used at each layer, each capturing different patterns in the input. The number of kernels and their size decides the output dimension of that layer. Terminology connected to CNN's will be explained below.

Kernel size
The size of the vector / matrix used as kernel. In the example above a kernel size of 3 is used. If the kernel is 2d the size would e.g. be 3×3. Often a 2d kernel is square but this is not a necessity [22].

Padding
In the example above no padding is performed. Without padding the output dimension is reduced compared to the input dimension. If one wish to obtain same output dimension as input dimension padding can be done [22]. Below is seen an example of 0-padding where zeros are added to the end points of the input. Here a padding size of 1 is used as 1 zero as added in both ends.

$$\underbrace{\begin{bmatrix}0\\1\\2\\0\\3\\4\\0\end{bmatrix}}_{input} \star \underbrace{\begin{bmatrix}1\\0\\1\end{bmatrix}}_{kernel} = \underbrace{\begin{bmatrix}0\cdot 1+1\cdot 0+2\cdot 1\\1\cdot 1+2\cdot 0+0\cdot 1\\2\cdot 1+0\cdot 0+3\cdot 1\\0\cdot 1+3\cdot 0+4\cdot 1\\3\cdot 1+4\cdot 0+0\cdot 1\end{bmatrix}}_{output} = \underbrace{\begin{bmatrix}2\\1\\5\\4\\3\end{bmatrix}}_{output}$$

[4] Kernels - Wikipedia

With 2d input the entire rim will be padded and the size of the padding can also be adjusted as desired. Zero padding $[n, m]$ means n zeros on both sides of the width dimension and m zeros on both sides of the height dimension.

Stride

The stride is how far to move the kernel at each step. In the above examples a stride of 1 is used. In the 2d example strides takes the form $[n, m]$, which means move n fields right when moving horizontally and m fields down when moving vertically [22].

Channels

RGB-images have 3-colour channels making a colour image not 2d but actually 3d, thus a 3d kernel is needed when handling RGB-images. One could also imagine 1d data with multiple channels, for instance data with a temporal dimension and channels corresponding to independent observations. This data would look 2-dimensional, but patterns between the independent observations are not expected, only along the temporal axis [23]. For this reason 1d Convolution is preferred when working with multi-channel temporal data (such as speech) [23].
This means that nd-convolutions is not about the dimensions of the kernel but how the kernel is moved over the input [23]. 1d-Convolutions move the kernel only one direction, whereas 2d-Convolutions move the kernel as a typewriter.
A 2d-Convolution of a RGB-image with a 3d-Kernel would produce an output with 1 channel. If more output channels are desired a higher number of kernels can be used, each performing its own 2d-Convolution and then stacking the outputs [22].

Transposed Convolution

Convolutions without padding reduce the size of the input data. Sometimes it is desired to scale up the output to the original input size. This can be done with a transposed convolutional layer (or a de-convolutional layer). Transposed convolutions aim to reverse the process of standard convolutions. If the stride, padding and kernel size of the convolution is known, the original input can almost be restored. A transposed convolutional layer uses padding as a regular convolutional layer, but the stride does not change the step-size of the kernel, which in this case is always 1. Instead the stride expands the input by adding zeros between the input elements. Transposed convolutional layers calculates the padding p' and stride s' from the original padding p and stride s of the convolutional layer and the kernel size k [24].

$$p' = k - p - 1$$
$$s' = s - 1$$

Below an example of a 2d input with 2d convolution with $s = 1$, $p = 0$ and $k = 3 \times 3$ and its corresponding transposed convolutional layer

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{Original input}} \Rightarrow \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}}_{\text{Convolution output}} \Rightarrow \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{De-Convolution input}} \Rightarrow \underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{De-Convolution output}}$$

4.3.3 Recurrent Neural Networks

The recurrent neural network is a network where the nodes form a directed graph along a temporal sequence, making it exhibit temporal dynamic behaviour. RNN's comes in many variants, but most commonly a RNN has neuron-like nodes organised into layers.[5] RNN's are just like regular ANN's, but each node has a direct connection to the next node 4.9. The recurrent part is because the network loops a message to affect the next output. A RNN can be thought of as a number of copies of the same network where information is sent to the successor network, making RNN ideal for sequential and time dependent data such as speech[25]. This however, has the downside of not being able to run in parallel as the output related to the previous input is needed when evaluating the next input.

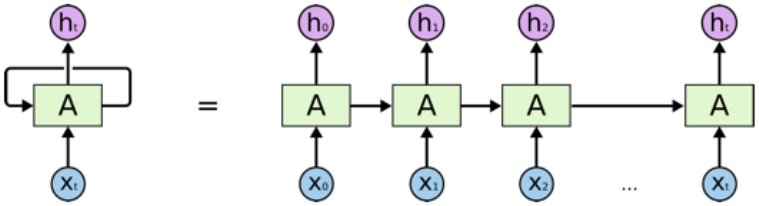

Figure 4.9: A RNN network visualised. From Christopher Olah: https://colah.github.io/posts/2015-08-Understanding-lstms/

The effect of using RNN's when processing speech can be shown with a simplified example. Imagine the input data X to be a spectrogram of voice saying *"She eats apples"* and the output to be the given sentence. If X is split into three parts (one for each word) x_0, x_1 and x_2 as in figure 4.9, the information behind choosing h_0 and h_1 helps to guide the RNN to come with a better prediction of h_2 since this word depends on the previous[25].

Each nodes calculates the output by the following function[25]:

$$h_t = tanh(W_t \cdot [h_{t-1}, x_t] + b_t) \tag{4.16}$$

[5] Recurrent Neural Network - Wikipedia

$[h_{t_1}, x_t]$ indicates concatenation of h_{t-1} and x_t, b_t is the bias computed and W_t are the weights of the network in node t.

In this project a variant of RNN known as Long Short Time Memory LSTM is used.

4.3.4 Long Short Term Memory

An issue with RNN's is they have a hard time handling long-term dependencies, meaning they forget. If vital information for predicting the last word in a sentence is given in the beginning of the sentence a basic RNN structure as in figure 4.9 is not capable of knowing that this early information is important to remember.

The LSTM network however, has feedback connections in its architecture making it good at processing entire sequences of data like speech. These feedback connections are represented as different gates which helps the LSTM network to chose which information to remember and which to forget[25].

A standard LSTM network has three of these gates in each cell and in addition it has a cell state C_t which is depicted as the upper vertical line in figure 4.10. The cell state is transferred directly to the neighbour cell and all the way down the network with only minor changes to it in each cell. This allows a flow of information through the network[25].

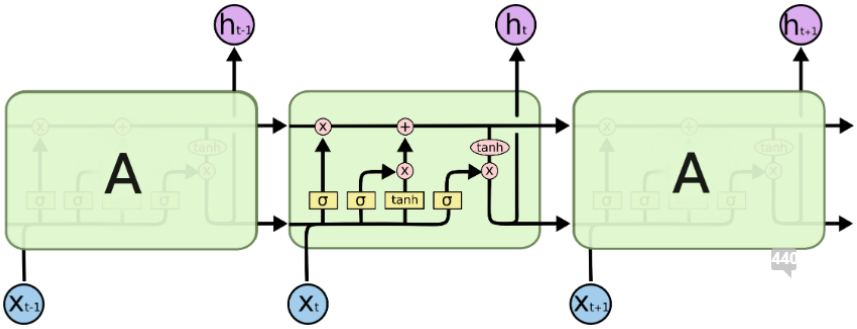

Figure 4.10: An unrolled LSTM network. Arrows are vector transfer, yellow squares are ANN layers, red circles are element-wise operations, merging arrows are concatenations and splitting arrows are copying. From Christopher Olah: https://colah.github.io/posts/2015-08-Understanding-lstms/

Every time an ANN layer (yellow square in figure 4.10) goes to a element-wise operation (red circle) it corresponds to a gate. The first gate (starting from left in figure 4.10) is for forgetting. The output of the previous cell is used to decide what to forget from the previous cell state. This is done by weighting the information of C_{t-1} with numbers between 0 and 1. These numbers come from the first ANN layer in the cell with the sigmoid activation function (eq. 4.17)[25].

$$f_t = \sigma(W_{tf}[h_{t-1}, x_t] + b_{tf}) \tag{4.17}$$

The second gate is for remembering; deciding what information to store in the cell state C_t. This is done by first calculating the RNN output g_t as in (eq. 4.16) and then scale this output by a second ANN layer with the sigmoid activation function (eq. 4.18)[25].

$$i_t = \sigma(W_{ti}[h_{t-1}, x_t] + b_{ti}) \tag{4.18}$$

The cell state is then updated by first forgetting by element-wise multiplication with f_t and afterwards store the new information by adding $i_t * g_t$

$$C_t = C_{t-1} * f_t + i_t * g_t \tag{4.19}$$

where * is element-wise multiplication.

The final gate is for deciding what to output. The updated cell state C_t is passed through a tanh activation function whose output is again element wise multiplied with an output o_t from an ANN with sigmoid activation function in order to decide the importance of the output elements[25].

$$o_t = \sigma\left(W_{to}([h_{t-1}, x_t] + b_{to}\right) \tag{4.20}$$

$$h_t = tanh(C_t) * o_t \tag{4.21}$$

LSTM can remember the most relevant previous predictions and can learn to identify if a prediction in the next time step should be given a negative value for the next prediction giving the previous chosen prediction.
For speech, LSTM could be good at translating speech data, since the predictions that is stored in its short term memory can affect later predictions.

LSTM networks can be stacked in layers in which the output of one layer is used as input in the next layer.

Bidirectional Long Short Term Memory
A bidirectional long short term memory (BiLSTM) can pass information through a forward pass and a backward pass, making it possible to get information on what the next prediction should be given both past and future inputs. Imagine figure 4.10 with a second almost identical LSTM network above the first with the directions reversed, meaning it starts with input x_{n-1} and ends with input x_0.

This gives two outputs for each input x_t a forward output (the regular LSTM output) and a backward output from the second LSTM network (with the direction reversed)

$$\begin{aligned} \text{Forward output:} \quad & \overrightarrow{h}_t \\ \text{Backward output:} \quad & \overleftarrow{h}_t \end{aligned} \tag{4.22}$$

In short, a BiLSTM can simultaneously acquire information from past and future states with the help of two LSTM networks and preserve that information. The Unidirectional LSTM can only preserve information from past states to help with its predictions.

The forward and backward output can be summed together and passed through an activation function, which yields a single final output for each input.

Gated Recurrent Unit
A slightly more complicated variation of LSTM's is the gated recurrent unit (GRU). It merges the hidden state h_t with the cell state C_t into a single state h_t and connects the forget- and update gate more directly. A GRU cell is showed below in figure 4.11 and the update process is as follows [25].

$$\begin{aligned} z_t &= \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \\ r_t &= \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \\ \tilde{h}_t &= \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \end{aligned} \tag{4.23}$$

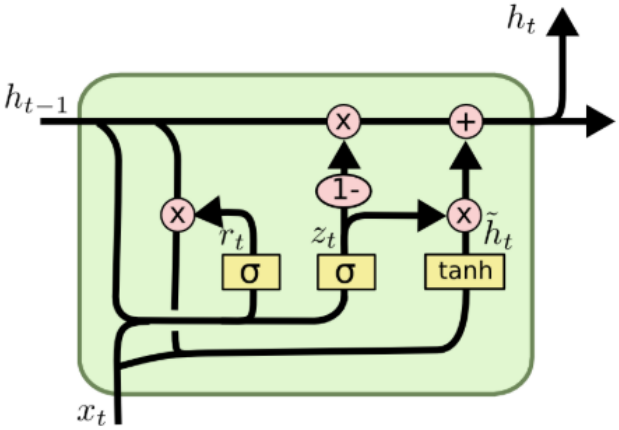

Figure 4.11: *Gated Recurrent Unit:* A variant of LSTM. From Christopher Olah: https://colah.github.io/posts/2015-08-Understanding-lstms/

4.3.5 Adam Optimiser

Adam is a nickname for Adaptive Moment Estimation given to the optimisation algorithm by the authors of [26]. Adam is a variant of stochastic gradient descend which as classic gradient descend computes the gradient of the loss function $J(\theta)$ with respect to the model parameters and moves with step size α in downward direction:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta J(\theta) \tag{4.24}$$

Instead of using the full available dataset to compute the loss SGD only uses a smaller batch, which reduces computation but also makes the optimisation fluctuate [27].
To help SGD converge we add momentum m_t, which in the Adam optimiser is calculated as a decaying average of past gradients g:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{4.25}$$

m_0 is initialised as a zero-array. Momentum uses information of past update steps to keep the direction of the SGD more steady. It can be thought of as a ball rolling downhill; building up momentum; making it difficult to push off its direction. The decaying average of the momentum is used to 'forget' early update steps, which otherwise will keep adding maybe inconvenient momentum [27].
In addition to momentum Adam also has an adaptive learning rate. This will give a higher learning rate for more crucial parameters and a lower learning rate for more irrelevant parameters. In Adam the adaptive learning rate v_t is stored as a decaying average of squared gradients g^2:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4.26}$$

v_0 is initialised as a zero-array. v_t is added to the update step to penalise parameters differently, again by weighting information of more recent updates higher.
The authors of [26] notes that m_t and v_t are biased toward zero. To counteract this they add bias correction:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

The full Adam optimisation step using momentum and adaptive learning rate is as follows [26]:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{4.27}$$

The ϵ is added to avoid division by zero errors. The algorithm parameters are set to default by the authors of [26] to:

$$\beta_1 = 0.9, \quad \beta_2 = 0.999, \quad \epsilon = 10^{-8} \tag{4.28}$$

Learning Rate Scheduling
The learning rate parameter α of an optimiser (such as Adam) has an important impact on the convergence properties of the optimisation. Starting with a relatively high learning rate and reducing it as we go is known as *annealing learning rate* and is often very beneficial for the training of ANN's. The type of learning rate decay varies and there is no clear answer to which to use. In the training of the models AutoVC ([1]) and StarGAN ([2]) annealing learning rates are used.

4.3.6 Batch Normalisation

To accelerate the training of ANN's, the authors of [28] proposes a technique called batch normalisation (BN). They address the issue of *Internal Covariate Shift* which is the change in distribution of network activations (i.e. layer output) when training due to change in network parameters [28]. This means the network layers have to adapt to various distributions of input, which slows down the training progress. To

counter this the authors suggest normalising each dimension of a d-dimensional input $x = (x^{(1)}, ..., x^{(d)})$ as seen in (4.29) [28].

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}\left[x^{(k)}\right]}{\sqrt{\mathrm{Var}\left[x^{(k)}\right]}} \tag{4.29}$$

However, normalising the input of a layer has a downside of changing what that layer is able to represent. For this reason each input $x^{(k)}$ is scaled and shifted using the learnable parameters $\gamma^{(k)}$ and $\beta^{(k)}$ [28].

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)} \tag{4.30}$$

When using a variant of stochastic gradient descend (SGD) (such as Adam) BN uses 'mini-batch statistics' to estimate mean and variance of each activation [28]. Let x_{tihw} be the $tihw$ element of an input tensor, where h and w is the span of the spatial dimensions, i is the feature channels and t is the index of the element in the mini batch. For instance a batch of T RGB-images with 3 channels, $I = 3$ and H and W are the height and width of the images. Batch normalisation with mini batch statistics looks as follows [29]:

$$\begin{aligned}
\widehat{x}_{tihw} &= \frac{x_{tihw} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \\
\mu_i &= \frac{1}{HWT}\sum_{t=1}^{T}\sum_{w=1}^{W}\sum_{h=1}^{H} x_{tiwh} \\
\sigma_i^2 &= \frac{1}{HWT}\sum_{t=1}^{T}\sum_{w=1}^{W}\sum_{h=1}^{H} (x_{tiwh} - \mu_i)^2 \\
y_{tihw} &= \widehat{x}_{tihw}\gamma_{tihw} + \beta_{tiwh}
\end{aligned}$$

where the constant ϵ is added for numerical stability [28].
The authors of [29] prove that replacing batch normalisation in generative networks (such as GAN's see 5.2.1) with a different normalisation technique instance normalisation (IN), "dramatically" improves the performance of certain ANN's [29]. IN is highly related to BN, but instead of normalising over the entire batch, each element in the batch (e.g. each image) is normalised independently. IN looks as follows [29]:

$$\begin{aligned}
\widehat{x}_{tihw} &= \frac{x_{tihw} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}} \\
\mu_i &= \frac{1}{HW}\sum_{w=1}^{W}\sum_{h=1}^{H} x_{tiwh} \\
\sigma_i^2 &= \frac{1}{HW}\sum_{w=1}^{W}\sum_{h=1}^{H} (x_{tiwh} - \mu_{ti})^2 \\
y_{tihw} &= \widehat{x}_{tihw}\gamma_{tihw} + \beta_{tiwh}
\end{aligned}$$

4.3.7 Residual Blocks

The methods used in this project also applies residual blocks/residual connections. Residual blocks are handy for deep networks as it can help solve around a problem know as a Degradation problem. The Degradation problem is a tendency seen in training of deep neural networks where a network at a point of applying more layers (which should in fact make the network more complex and accurate), starts to decrease in accuracy [30].

The reason why residual blocks or residual learning is called what it is, is because of the way that residual learning learns from a residual function instead of the true distribution. A normal neural network block wishes to learn the true distribution $H(x)$ from a input x. The difference aka. the residual function is thus [30]:

$$\mathcal{F}(x) = Output - Input = H(x) - x \tag{4.31}$$

This can then be rearranged to find the true distribution:

$$H(x) = \mathcal{F}(x) + x \tag{4.32}$$

So this residual block wishes to find the true distribution through the residual function, meaning that these blocks does not try to learn the true output but the residual instead.

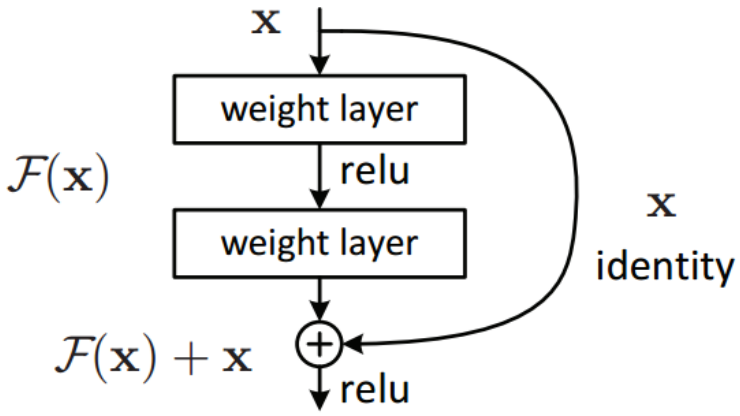

Figure 4.12: A simple single residual block. x is the input and identity that gets send around certain layers to feed and combine with our residual function to find the true distribution. From Residual blocks — Building blocks of ResNet

Residual connections is the principle of skipping training of a few layers for layers to come to learn the identity function directly. Skip connections are great because they allow us to propagate larger gradients to initial layers, giving us the ability to train deeper networks[6]. This is because residual learning fixes the problem of vanishing gradient

[6] Residual blocks — Building blocks of ResNet

since this problem tend to become severe with an increasing number of layers using back-propagation. The vanishing gradient problem describes how ANN's (trained with gradient based learning and back-propagation) tend to stop training since the weights of the networks tend to become vanishingly small when updated from a partial derivative of the error function with respect to the current weight[7].

So, in a way residual learning works kinda opposite of a gate, and more like a highway. This highway can be build in different ways to achieve the best result with a bit of trial and error, but the flow can skip layers to avoid increasing inaccuracy of the network. Residual connections also makes the network dynamic in the sense that we do not know the optimal amount of layers, and we therefore can skip the training of the layers that do not improve the accuracy, making the overall training faster, optimising the netowrk in general[8].

4.3.8 Activation functions

The output of ANN layers is often passed through an activation function. In this project 3 different activation functions are used: *sigmoid* which forces the output to be between 0 and 1, *tanh* which forces the output to be between -1 and 1 and *ReLU* which is 0 for negative values (see figure 4.13)

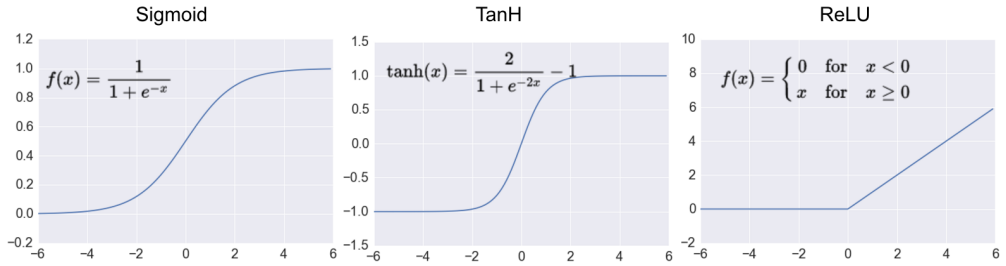

Figure 4.13: The 3 activation functions used in this project where the x-axis represents the input value and the y-axis the output value. From https://www.quora.com/What-is-meant-by-activation-function

[7] Vanishing gradient problem
[8] Residual blocks — Building blocks of ResNet

5 Models

The GAN and AutoEncoder seemed to be the best choices for voice conversion to jump into. Besides it being State-of-the-art right now, many papers try to perform voice conversion based on these methods. With transfer-learning, high end goals can allegedly be achieved using these methods. There are some risks for the results end up poor, but that lies in the two methods not being exactly perfect. GAN's are very difficult to train since it takes a lot of time and its convergence property seems to be fragile. Autoencoder is easier to train, but can easily over-smooth the conversion output, not guaranteeing distribution matching[1]. The methods may fool the discriminators, but only time will tell if the conversions sounds real to the human ear.

The models seem in literature to be the ones gaining the most popularity, which is why these two models are being looked into in this paper.

5.1 The AutoVC model

5.1.1 AutoEncoder

An AutoEncoder (AE) is an unsupervised learning technique, which uses a Neural Network (NN) to learn the features of an input x. The (AE) consist of an encoder and a decoder. The encoder learns a function to code an embedding of the input $E(x)$ and the decoder learns a function to translate this embedding back into x; $D(E(x)) = \hat{x}$, where $\hat{x}$ is the reconstruction of x. This task seems unprofitable in theory, which it also can be in practice with a poor design of the AE. However, what is of interest is not the output of the AE, but the latent representation of x that the encoder learns [31].
Besides of feature learning, AE's have been used for dimension reduction just as PCA has. AE and PCA are highly related and it can be shown that AE with one hidden layer, linear activation functions and a squared error function has weights which span the same subspace as the two first principal components [32]. But if the activation function is non-linear the AE is able to capture non-linear relations in latent space [31].
A simple AE with in- and output size N and a hidden layer H (often referred to as the bottleneck) can be seen as a N-H-N network. The weights N-H work as the encoder and the weights H-N as the decoder (see figure 5.1) [33]. Though an one-layer AE is a common approach it is however not a requirement and deep structures for the encoder and decoder might be advantageous.
The AE learns simply by minimising a loss function (often referred to as reconstruction loss), which measures the similarity between the input and output (e.g. the squared euclidean norm)[31]

$$\min \mathcal{L}(x, D(E(x))) = \min ||x - D(E(x))||_2^2 \tag{5.1}$$

The trick is to design the bottleneck in such a way that the network learns to represent the most salient features of x, for instance by reducing the dimension of H.

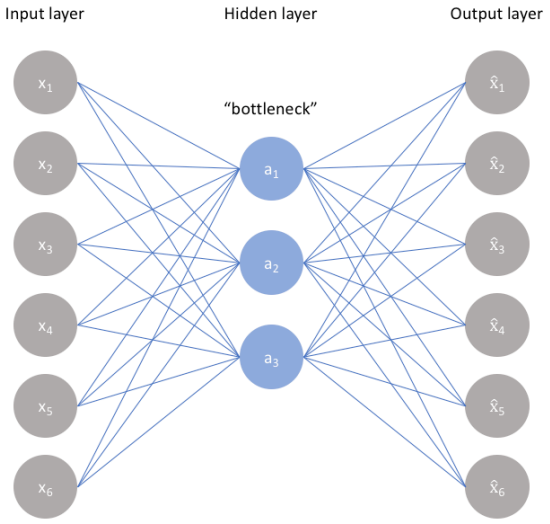

Figure 5.1: Architecture of a simple Autoencoder, from "Jeremy Jordan: Introduction to Autoencoders"[33]

5.1.2 AutoVC

The framework of AE is simple, yet it is a powerful tool for voice conversion as proven in [1]. The code and hence the architecture of AutoVC will be used to replicate the results of [1] and then be applied to answer some of this papers research questions (Q1 and Q2). The authors of [1]s' justification for the use of an AE for voice conversion will be summarised and the architecture of AutoVC will be described in this section.

Model Justification

AutoVC uses two encoders. The first encoder, $E_U(\cdot)$, is carefully designed and trained to extract a latent representation of the speaker identity U and the second encoder, $E_Z(\cdot)$, wishes to extract the latent representation of the content Z. Following directly from assumption 1:

Assumption 2 *Given a trained speaker identity encoder $E_U(\cdot)$ and two speech segment X_a and X_b then the following holds: if $a = b$ then $E_U(X_a) = E_U(X_b)$ and if $a \neq b$ then $E_U(X_a) \neq E_U(X_b)$*

If $E_Z(\cdot)$ has a bottleneck designed exactly wide enough to contain all content information Z, but narrow enough to disregard speaker independent information U, then by concatenating $E_Z(X_{source})$ with $E_U(X_{target})$ (will be denoted X_s and X_t respectively), trained with the training scheme in next section, the decoding should yield perfect voice conversion under the given assumptions[1].

Loss function and training

The speaker identity encoder $E_U(\cdot)$ is trained on the generalised end to end (GE2E) loss [34], which aims to maximise the similarity among latent representations of utterances by the same speaker and minimise the similarity among those of different speakers. A

speaker identity encoder trained on the GE2E loss should therefore agree with assumption 2 [1].
GE2E loss is defined as follows; Let $\mathbf{X}_{ij}$ be the jth speech segment of the ith speaker. The output of the encoder network $f(\mathbf{X}_{ij}, \mathbf{w})$ where $\mathbf{w}$ are the weights, is normalised using the L2 normalisation, which results in the embedding vector $\mathbf{e}_{ij}$:

$$\mathbf{e}_{ij} = \frac{f(\mathbf{X}_{ij}, \mathbf{w})}{||f(\mathbf{X}_{ij}, \mathbf{w})||^2} \tag{5.2}$$

The centroid $\mathbf{c}_i$ is the mean of the embedding vectors for speaker i and can be referred to as the 'voice-print' of speaker i.

$$\mathbf{c}_i^{(-j)} = \frac{1}{M-1} \sum_{\substack{m=1 \\ m \neq j}}^{M} \mathbf{e}_{im} \tag{5.3}$$

The $j'th$ speech segment is left out in the centroid when computing the similarity matrix (below) for the $j'th$ speech segment to avoid bias. Afterwards the similarity matrix $\mathbf{S}$ as the scaled cosine similarity between embedding vectors $\mathbf{e}_{im}$ and all centroids $\mathbf{c}_k$.

$$\mathbf{S}_{ij,k} = \begin{cases} w \cdot \cos\left(\mathbf{e}_{ij}, \mathbf{c}_i^{(-j)}\right) + b & \text{if } k = i \\ w \cdot \cos\left(\mathbf{e}_{ij}, \mathbf{c}_k\right) + b & \text{otherwise} \end{cases} \tag{5.4}$$

where w and b are learnable parameters with $w > 0$ - a good choice of initialisation of these is (10, -5) respectively [34]. Think of $S_{ij,k}$ as a $I \times M \times K$ matrix, with I being the total number of speakers, M being the total number of speech segments for each person and $K = I$. Each row in the spatial dimension K carries the elements of the similarity matrix, $S_{ij,k}$.

For each centroid $k = 1 \dots N$ a softmax is put on the similarity matrix

$$\mathbf{S}_{ij,k} = \frac{e^{\mathbf{S}_{ij,k}}}{\sum_{i=1}^{N} \sum_{j=1}^{M} e^{\mathbf{S}_{ij,k}}} \tag{5.5}$$

The loss of each embedding vector is then defined as

$$\mathcal{L}(\mathbf{e}_{ij}) = -\mathbf{S}_{ij,i} + \log \sum_{k=1}^{N} e^{\mathbf{S}_{ij,k}} \tag{5.6}$$

The negative term drags the embedding near the centroid it belongs to whereas the positive term pushes it away from the other centroids as shown i figure 5.2 [34]. The GE2E loss is defined as the total sum of the losses of the embedding vectors.

$$\mathcal{L}_{GE2E}(\mathbf{S}) = \sum_{j,i} \mathcal{L}(\mathbf{e}_{ij}) \tag{5.7}$$

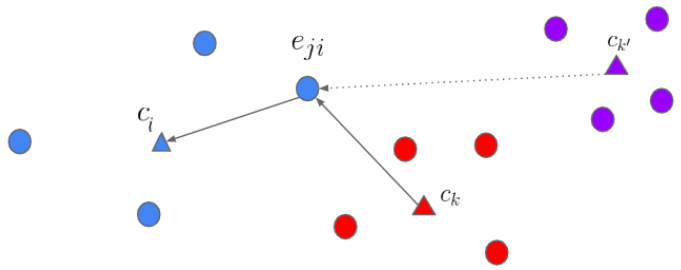

Figure 5.2: Illustration of how GE2E loss works, from *Wan et. al: "Generalized End-to-End Loss for speaker verification"*[34]

With a trained Speaker Identity encoder AutoVC can be trained. The trick is to design the bottleneck dimension of the content encoder $E_Z(\cdot)$ to lose as much information as possible and still be able to reconstruct the signal. The content encoder will then focus on the content information since speaker information is provided by the speaker identity encoder $E_U(\cdot)$. If the bottleneck is perfectly designed then perfect reconstruction can be achieved (under the given assumptions) and the content encoder disregards all information about the speaker identity [1].
AutoVC is trained with the following loss functions; The reconstruction loss is defined as the squared euclidean distance of the reconstructed speech segment $\hat{\mathbf{X}}_{1\to1}$ and the original speech segment $\mathbf{X}_1$ [1].

$$L_{recon} = \mathbb{E}\left[||\hat{\mathbf{X}}_{1\to1} - \mathbf{X}_1||_2^2\right] \tag{5.8}$$

The content reconstruction loss is defined as the 1-norm of the difference between content encoding $E_Z(\hat{\mathbf{X}}_{1\to1})$ of the reconstructed signal and content encoding $E_Z(\mathbf{X}_1)$ of the original signal [1].

$$L_{content} = \mathbb{E}\left[||E_Z(\hat{\mathbf{X}}_{1\to1}) - E_Z(\mathbf{X}_1)||_1\right] \tag{5.9}$$

The decoder has no information about the speaker identity, before it is applied by the speaker identity encoder, when reconstructing speech. This indicates if

$$p_{X_s\to X_t}(\cdot|E_Z(X_s), E_U(X_s)) = p_X(\cdot|Z_s, U_s) \tag{5.10}$$

then

$$p_{X_s\to X_t}(\cdot|E_Z(X_s), E_U(X_t)) = p_X(\cdot|Z_s, U_t) \tag{5.11}$$

A third reconstruction loss is applied to improve convergence [1]. Before AutoVC outputs a speech segment $\hat{\mathbf{X}}$ an initial estimate of the converted speech $\tilde{\mathbf{X}}$ is produced before the final part of the decoder (as described in the next section). This loss is defined as [1]

$$\mathcal{L}_{recon2} = \mathbb{E}\left[||\tilde{\mathbf{X}}_{1\to1} - \mathbf{X}||_2^2\right] \tag{5.12}$$

To summarise the AutoVC training tries to minimise a weighted sum of these three loss functions using the Adam optimiser.

$$\min \ (L_{recon} + \mu L_{recon2} + \lambda L_{content}) \tag{5.13}$$

5.1.3 Architecture

The architecture of AutoVC is comprised of 4 modules: a *speaker identity encoder*, a *content encoder*, a *decoder* and a *postnet*. The *speaker identity encoder* is independent of the rest, whereas the remaining 3 modules forms the Autoencoder.

Speaker identity encoder

The *speaker identity encoder* is vaguely different from the one proposed in the AutoVC paper [1]. It consists of 3 stacked LSTM layers, with a hidden layer (output) of size 256. The output is then projected onto a fully connected linear layer of size 256 with the ReLU activation function followed by a L2-normalisation (see figure 5.3b).
The total number of trainable parameters in the *speaker identity encoder* is 1,423,618.

Content encoder

The *content encoder* follows the framework as in [1]. First, the input is fed into three 1d Convolutional layers with a output channel size of 512, a ReLU activation function in each layer and applied with BN for each layer. Secondly, the CNN output is propagated through 2 stacked BiLSTM layers with output channel size 32 (see figure 5.3a). Finally, the forward output of the last BiLSTM layer is downsampled along the temporal axis by only keeping the time steps $\{31, 63, 95, ...\}$ (see figure 5.3d-1) and the backward outputs is downsampled by only keeping the time steps $\{0, 32, 64, ...\}$ (see figure 5.3d-2). The weights are all initialised using *normalised initialisation* [35]

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right] \tag{5.14}$$

with n_j being the number of weights of the j'th layer.

Decoder and Postnet

The *decoder* firstly upsamples the content encoder outputs by the upsampling schemes visualised in figure 5.3d-1 and -2, followed by a concatenation of these upsampled outputs and the output of the speaker identity encoder. This is then propagated through one LSTM layer with output channel size 512, three 1d Convolutional layers each with output channel size 512, ReLU activation function and batch normalisation, two LSTM layers with output channel size 1024 and finally projected unto a fully connected linear layer of output channel size 80 (see figure 5.3c).
The *Postnet* is a residual block at the end of the decoder comprised of four 1d Convolutional layers with output channel size 512, a tanh activation function and batch normalisation followed by a single 1d Convolutional layer with output channel size 80.
The total number of trainable parameters in AutoVC is 28,422,464

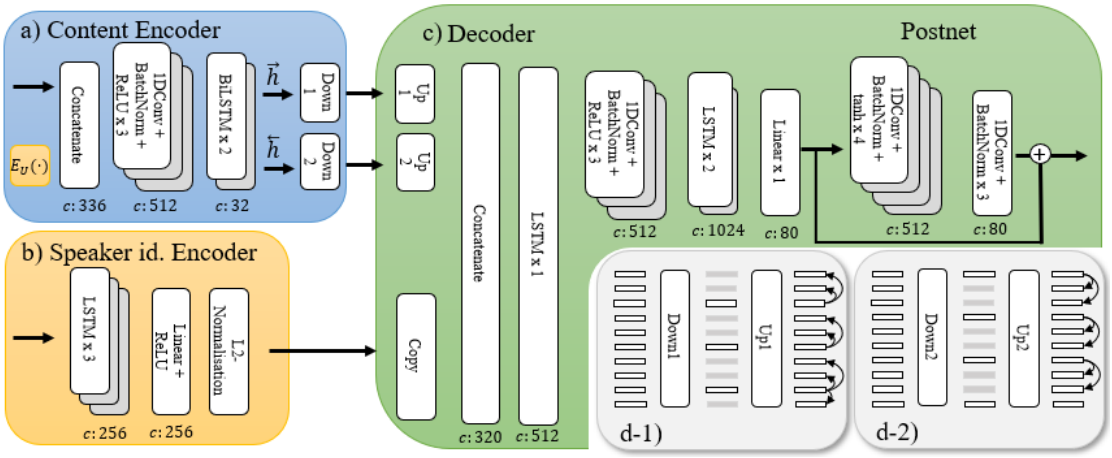

Figure 5.3: *AutoVC Architecture:* The 1d Convolutional layers all have kernel size 5, stride 1 and padding 2, thus maintaining the shape of the input. c denotes output channel size, arrows are vector transfer. Modified from the original paper [1].

5.2 The StarGAN model

5.2.1 Generative Adversarial Network (GAN) - TODO

A generative adversarial network (GAN) comes in many forms such as Deep convolutional GANs, AdaGANs and cycle GANs[1]. This section serves the purpose of introducing the general ideas behind a GAN.

The objective of a GAN is to generate samples that are not real, but look exactly like real samples. This is done with a generator and a discriminator. To put things into an art perspective, the generator can be seen as an art forger trying to make forged paintings that looks just like the real ones and the discriminator can be seen as an art distributor who has to tell if the painting is real or not. In the beginning neither the art forger or the art distributor are very good at their job. The forger becomes better by trying to make forgeries and see whether or not they are good enough to fool the art distributor and he will have to become better each time as the art distributor becomes better at detecting forgeries. The art distributor becomes better by looking at both real and fake paintings and learns to distinguish between the two. This is exactly what happens with a GAN as the generator generates samples that have never been seen before and passes them to the discriminator, which has to tell if it is real or fake. This *adversarial* relationship helps both parts becoming better at their job, since the generator gets feedback from the discriminator on how to make the samples appear more real and the discriminator gets fake training samples that looks more real and will thereby improve, hence the name[36].

The discriminator

The discriminator is simply a classifier which will typically be some type of neural network with an architecture fitted to the task at hand. The data fed to the discriminator is

[1] https://deephunt.in/the-gan-zoo-79597dc8c347

made up of real and fake data. The real data X serves as the positive samples obtained from a data set and the fake data $G(z)$ serves as negative samples created by the generator. When training the discriminator a discriminator loss is used. This loss penalises the discriminator for doing any false positives or false negatives, meaning it has done a misclassification. The discriminator loss will then be used to update the weights of the discriminator by using backpropagation as seen in figure 5.4.

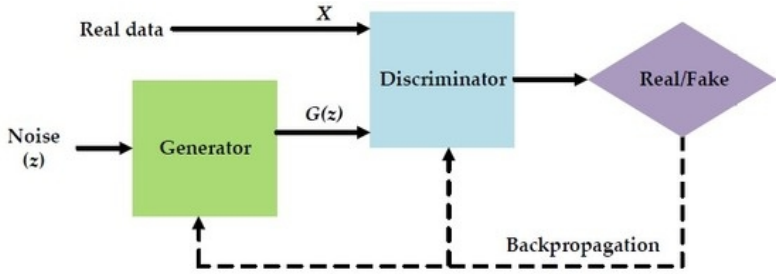

Figure 5.4: The general structure of a GAN, taken from [37]

The generator

The goal of the generator is to generate samples that the discriminator classifies as real. To generate an output the generator is fed some random noise z sampled from an arbitrary distribution and the generator network will then transform this in to a meaningful sample $G(z)$. To train the generator a generator loss is needed. This loss is based on whether or not the samples generated are classified as real or fake. Since this final output depends on the discriminator the backpropagation has to go through the discriminator network as well as the generator network which is seen in figure 5.4. As it would be difficult to hit a moving target in form of a discriminator being updated in the generator training and the purpose is to train the generator and not the discriminator only the weights of the generator network is updated in the backpropagation[36].

GAN training

As a GAN consists of two networks the training has to be split in two, so the discriminator has a chance to learn the flaws of the generator and the generator is not trying to hit the moving target of a discriminator that keeps changing. To do this the training has to balance between training the two networks which is done by first training the discriminator for one or more epochs followed by the generator being trained for one or more epochs. One of the most troublesome parts of this training is to decide when the GAN training has converged. When the generator improves the discriminator gets worse at classifying. If it comes to the point where the generator is trained to perfection, the discriminator will have a 50 % chance at guessing correct, as it will be random whether it is correct or not. This poses a problem, as the discriminator feedback will get less meaningful as it becomes more and more random what label it gives the samples. This will make the generator worse as the training will be based on random feedback. The convergence will therefore not be stable[36].

5.2.2 StarGAN

StarGAN is a state-of-the-art method when it comes to voice conversion (VC). It separates itself from other VC GAN methods such as CycleGAN [38] by only needing one generator to learn many-to-many mappings across different attribute domains. An attribute domain can e.g. be female speakers, age of the speaker or a specific speaker identity. Normal cross-domain models has to learn a one-to-one mapping for each pair of domains that the generator has to convert between. This results in the need for many generator models, e.g. 12 different generators would be needed for 4 different domains as seen in figure 5.5 opposed to StarGAN which only needs 1.

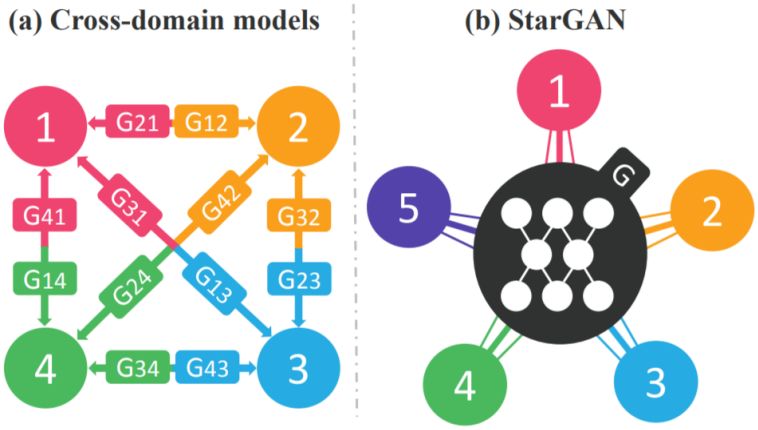

Figure 5.5: Comparison of a normal cross-domain model and StarGAN, taken from [4].

StarGAN was first introduced for images in the paper "*StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation*"[4] and was then adapted to be used for VC in the paper "*StarGAN-VC: Non-parallel many-to-many voice conversion with star generative adversarial networks*"[2]. The name StarGAN refers to the fact that the model structure looks like a star as seen in figure 5.5 as multiple domains can be mapped with a single generator in the center.

Preprocessing of data

The speech segments used as training data are resampled to 16000 Hz and divided into intervals/frames of 5 ms before the Harvest Algorithm [18] and the CheapTrick algorithm [19] are applied to every frame, yielding a F_0 contours and spectral envelopes for every speaker. 36 mel frequency cepstrum coefficients (MFCC) are then extracted from the spectral envelopes by using the discrete cosine transformation. The result is the acoustic feature vector for each time frame and can be visualised as in figure 4.7. During training only inputs of 256 frames are used, meaning utterances are sliced to match this size.
In addition to this, aperiodicity bands of every speakers utterances are extracted with D4C [20] for later synthesis with the WORLD vocoder.

The generator

The generator G takes an acoustic feature vector $\mathbf{x} \in \mathbb{R}^{Q \times N}$ along with a target attribute label c as input, where Q is the feature dimension, N is the length of the temporal

dimension (see figure 4.7 and c is an one-hot representation of the speaker label as this is the only feature used. From these two inputs G can generate a new acoustic feature vector $\hat{\mathbf{y}} = G(\mathbf{x}, c)$. The goal of the generator is to make $\hat{\mathbf{y}}$ sound like it came from the speaker with attribute c. When training the generator, it is given a random acoustic feature $\mathbf{x}$ of attribute c' along with a random target attribute label c which is used to generate a fake sample to give the discriminator as seen in figure 5.6.

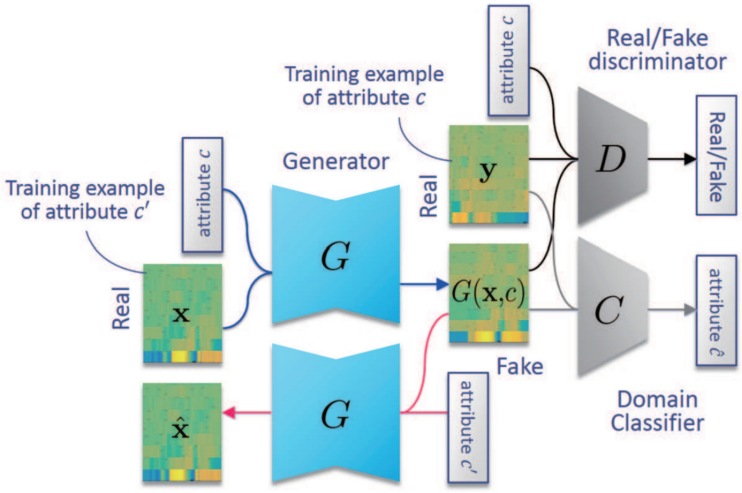

Figure 5.6: The training process of StarGAN, taken from [2]

The discriminator

The discriminator in StarGAN uses a normal real/fake discriminator D to give a probability $D(\mathbf{y}, c)$ that $\mathbf{y}$ is a real speech feature. This does however only tell something about how real the speech sounds and not how likely it was to come from the target speaker c. To address this a domain classifier C is introduced, which gives a probability distribution $p_C(c|\mathbf{y})$ for which class/domain $\mathbf{y}$ belongs to. The discriminator trains on both real samples belonging to attribute c and fake generated samples made to appear like it has attribute c as seen in figure 5.6. The auxiliary input of c separates StarGAN from a normal GAN which only tries to generate an output that looks real. By introducing the domain classifier and c as a second input, the model can be trained with only one generator instead one for each pair of domains to convert between [2]. A PatchGAN is used for both the real/fake discriminator and domain classifier. This separates itself from a normal discriminator classifying patches of an arbitrary size. The PatchGAN network will therefore output multiple values for different patches of the acoustic feature and tell how likely the patch is to be real along with the probability of each class. By taking the mean, an overall decision of the acoustic features realness and class can be decided by the discriminator[39].

Training
StarGAN can now be trained like a normal GAN where the generator has to fool the real/fake discriminator along with the domain classifier, the real/fake discriminator is trained to detect which acoustic features are real and the domain classifier is trained to predict the correct class of the acoustic feature. The generator is trained once for every time the discriminator has been trained 5 times. To optimise the networks used for these tasks some loss functions for each of these is introduced, for C a domain classification loss is used, for D an adversarial loss is used and for G an adversarial loss, a reconstruction loss and a domain classification loss is used. As C and D are both part of the discriminator these to losses are added along with a gradient penalty loss. The losses will typically consist of an expectation, as G and D takes in mini-batches of acoustic features and attribute labels and the loss has to reflect an arbitrary input[2].

Adversarial loss The adversarial loss $\mathcal{L}_{adv}$ tells something about how real the acoustic feature sounds. This means D must have a low adversarial loss in order for it to being good at classifying what is real and what is fake when being shown both real and generated acoustic features and G will try to generate an acoustic feature that D finds real. Since the adversarial loss applies to both the discriminator and the generator we denote the different losses by $\mathcal{L}^D_{adv}$ and $\mathcal{L}^G_{adv}$. The generator loss is simply given by the expected value of the real/fake discriminator on a generated sample

$$\mathcal{L}^G_{adv}(G) = -\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}),c\sim p(c)}\left[D(G(\mathbf{x},c))\right] \tag{5.15}$$

As the objective is to minimise the loss, the negative value of the output is used as the loss. For the discriminator the loss consists of how well the discriminator identifies real acoustic features as real and how well it recognises fake acoustic features made by the generator

$$\mathcal{L}^D_{adv}(D) = -\mathbb{E}_{c\sim p(c),\mathbf{y}\sim p(\mathbf{y}|c)}\left[D(\mathbf{y},c)\right] + \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}),c\sim p(c)}\left[D(G(\mathbf{x},c),c)\right] \tag{5.16}$$

Just as before the loss is defined as negative for the value which should be maximised, meaning the discriminators ability to recognise real acoustic features as real.

Domain classification loss The domain classification loss $\mathcal{L}_{cls}$ is the loss gained when misclassifying the domain label c. Here C wants to classify the domain of real acoustic features correctly and G wants to generate acoustic features that C will classify as the target domain label c. The domain classification loss used is the so called cross-entropy loss. To use this loss, the output of C has to be converted to a probability distribution $p_C(\mathbf{x})$ using a softmax function. To get the cross-entropy loss the real class distribution $p(c)$ is used as well, this is just a vector where there is 100 % probability of the correct class c. The cross entropy loss is then defined as [2]

[2] Loss Functions

$$\mathcal{L}_{cls} = -p(c) \log p_C(\mathbf{x}) \tag{5.17}$$

Since $p(c)$ will be 0 where the class is wrong, the expression can be reduced to simply looking at the probability that C identified the class of $\mathbf{x}$ to be the correct one

$$\mathcal{L}_{cls} = -\log p_C(c|\mathbf{x}) \tag{5.18}$$

For the generator the input acoustic feature vector is the generated sample $G(\mathbf{x}, c)$ and the domain classification loss of the generator thus becomes

$$\mathcal{L}^G_{\text{cls}}(G) = -\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}), c\sim p(c)} \left[\log p_C(c|G(\mathbf{x}, c))\right] \tag{5.19}$$

as the expected value is used to take different inputs into account and the negative part can be moved outside the expectation due to the multiplication rule for independt expected values. By using the same process of thoughts a similar loss can be defined for the discriminator, where the generated input is simply replaced by the input of a real acoustic feature $\mathbf{y}$

$$\mathcal{L}^C_{\text{cls}}(C) = -\mathbb{E}_{c\sim p(c), \mathbf{y}\sim p(\mathbf{y}|c)} \left[\log p_C(c|\mathbf{y})\right] \tag{5.20}$$

The probability can be between 0 and 1, meaning the loss will range from 0 to ∞, where 0 symbolises a perfect classification, as the discriminator will be 100 % sure on the correct class[2].

Reconstruction loss As the adversarial loss only makes sure that the acoustic feature sounds real, a reconstruction loss $\mathcal{L}_{rec}$ is introduced to make sure the linguistic features of the source speaker remains unchanged. If only the adversarial loss was considered there would be multiple generated outputs that sounded like the target speaker, thus by adding the reconstruction loss the generator becomes a bijective function, meaning the input maps to exactly one output and this output can only be produced by the same exact input. This means that the generator should be able to reproduce the original input by taking the generated acoustic feature $\hat{\mathbf{y}}$ and the original speaker label *c'*. The reconstructed acoustic feature is denoted $\hat{x}$ and the reconstruction loss can be expressed by

$$\mathcal{L}_{rec}(G) = \mathbb{E}_{c'\sim p(c), \mathbf{x}\sim p(\mathbf{x}|c'), c\sim p(c)} \left[\left\| G\left(G(\mathbf{x}, c), c'\right) - \mathbf{x}\right\|_1\right] \tag{5.21}$$

where $\mathbf{x}$ is the original input, c is the target attribute label and *c'* is the original attribute label. It is seen that the reconstruction loss is simply the expected value of the absolute difference between the reconstructed acoustic feature $G(G(\mathbf{x}, c), c')$ and the original acoustic feature $\mathbf{x}$[2].

Gradient penalty loss The purpose of the gradient penalty loss is to make sure the discriminator is a 1-Lipschitz function and thereby is constrained by the Lipschitz constraint. The discriminator is a 1-Lipschitz function if the following holds for two input acoustic features $\mathbf{x}_1$ and $\mathbf{x}_2$:

$$|D(\mathbf{x}_1, c_1) - D(\mathbf{x}_2, c_2)| \cdot |\mathbf{x}_1 - \mathbf{x}_2| \leq 1 \tag{5.22}$$

where $\mathbf{x}_1$ - $\mathbf{x}_2$ is the average frame-wise difference. This makes sure the rate of which the predictions $D(\mathbf{x}_1, c_1)$ and $D(\mathbf{x}_2, c_2)$ changes does not get too high, e.g. the absolute value of the gradient does not become larger than 1 anywhere.

The Lipschitz constraint addresses the vanishing and exploding gradient problem[40]. The exploding gradient is a problem where the gradients are larger than 1 throughout the network and when multiplied, exponential growth occur, resulting in extremely large weights[41]. The vanishing gradient problem, is the problem of the discriminator becoming too good and the loss function approaches 0, meaning the gradient will be zero[42].

To define the gradient penalty loss $\mathcal{L}_{gp}$ a randomly weighted average $\hat{\mathbf{z}}$ between the real acoustic feature $\mathbf{y}$ and the generated acoustic feature $\hat{\mathbf{y}}$ is made for each sample in the batch

$$\hat{\mathbf{z}} = \alpha \mathbf{y} + (1 - \alpha)\hat{\mathbf{y}} \tag{5.23}$$

where α defines how much of the mixed acoustic feature should originate from the real sample. This value is chosen by random for each pair of real and generated acoustic features and the random weighted average will thereby interpolate between the real and fake acoustic features as seen in figure 5.7.[43]

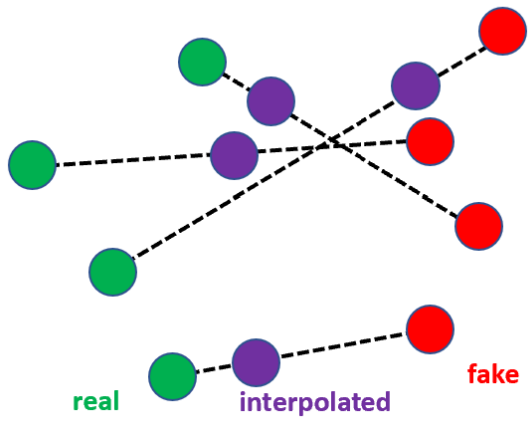

Figure 5.7: The interpolated acoustic feature (purple) will be a mix of the real (green) and fake (red) acoustic features from the training batch

The gradient penalty loss is now defined as the squared difference between the L2-norm of the gradient of the real/fake prediction wrt. the input and 1

$$\mathcal{L}_{gp}(D) = \mathbb{E}\left[\left(\left|\left|\frac{\mathrm{d}D(\hat{\mathbf{z}})}{\mathrm{d}\hat{\mathbf{z}}}\right|\right|_2 - 1\right)^2\right] \tag{5.24}$$

where the expected value is used to represent the entire batch[43].

Full loss definitions The different losses can now be put together for both the generator and discriminator

$$\mathcal{L}_G(G) = \mathcal{L}^G_{adv}(G) + \lambda_{rec}\mathcal{L}_{rec}(G) + \lambda_{cls}\mathcal{L}^G_{cls}(G) \tag{5.25}$$

$$\mathcal{L}_D(D) = \mathcal{L}^D_{adv}(D) + \lambda_{cls}\mathcal{L}^D_{cls}(D) + \lambda_{gp}\mathcal{L}_{gp}(D) \tag{5.26}$$

where $\lambda_{rec} \geq 0, \lambda_{cls} \geq 0$ and $\lambda_{gp} \geq 0$ are used to weight the importance of the different loss types relative to the adversarial loss.

Conversion process
With a trained generator voice conversion is possible. The generator has not only learned to generate realistic acoustic features, but also acoustic features within domains because of the domain classification loss and reconstructable linguistic content due to the reconstruction loss. The generator should thus be able to transform the acoustic features

of the source to the ones of the target whilst maintaining the linguistic features of the source speech.
The source MFCC's are converted to match the target's vocal features. The converted MFCC's are then transformed back to spectral envelopes with the inverse cosine transformation (eq. 5.27).

$$s(m) = \frac{1}{2}c_(0) + \sum_{n=1}^{N-1} c_(n) \cos\left(\frac{\pi m(n+0.5)}{N}\right), \quad m = 0, 1, 2, ..., M-1 \tag{5.27}$$

The source pitch is afterwards converted to match the targets pitch by using the logarithm Gaussian normalised transformation (eq. 5.28) [2]

$$\hat{F}_{0t} = \exp\left(\frac{log(F_{0s}) - \mathbb{E}[log(F_{0s})]}{\sqrt{Var(log(F_{0s}))}} \cdot \sqrt{Var(log(F_{0t}))} + \mathbb{E}[log(F_{0t})]\right) \tag{5.28}$$

With subset s and t denoting source and target respectively and means and variance are estimated empirically with the training data available. The logarithm used is masked, i.e. it ignores zeroes.
Converted spectral envelopes and pitch are used together with the aperiodicity of the source speech in the WORLD vocoder to synthesise the converted speech.

Architecture
The architecture of StarGAN used in this project is slightly different from that of the original StarGAN-VC model. The Generator consists of an input layer, a downsample network, a bottleneck network, an upsample network and an output layer (see figure 5.8). The input layer and downsample network consist of 2d Convolutional layers followed by instance normalisation and a ReLU activation function. The downsample layers have kernel size, padding and stride designed to half each spatial dimension while the channel dimension doubles. The bottleneck network is comprised of 6 residual blocks of 2d Convolutional layers, designed to maintain the input dimensions. The upsample layers are Transposed 2d Convolutional layers which doubles each spatial input dimension and halves the channel dimension, thus restoring the original input dimension. These are followed by instance normalisation and ReLU activation function (see figure 5.8).
The total number of trainable parameters in the Generator is 9,258,432.
The discriminator i consist of 5 downsampling layers of 2d Convolutions followed by leaky ReLU activation function, which is just like ReLU but with a small slope for negative values. Each layer halves the spatial dimension and doubles the channel dimension. The downsampled output is fed to 2 different 2d Convolutional layers, one, the Real Fake classifier, reducing the channel dimension to 1, and a second, the Domain Classifier reducing the channel dimension to the number of speakers. The full architectures of the generator and the discriminator are shown in figure 5.8
The total number of trainable parameters in the Discriminator is 12,110,784.

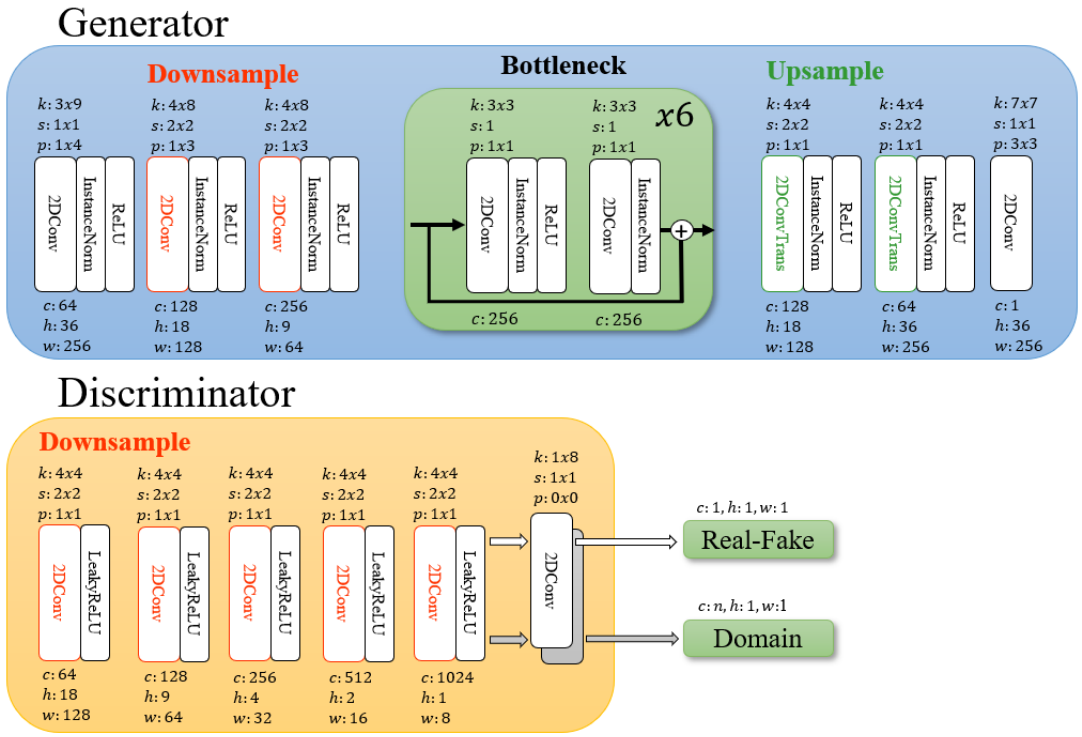

Figure 5.8: *StarGAN:* The architectures of the Generator and the Discriminator of StarGAN. Input dimensions are here assumed to be $[1x36x256]$. k is kernel size, s is stride, p is padding, c is output channel dimension, h is output height dimension and w is output width dimension.

5.3 Vocoders

After the source speaker has been transformed into a target speaker, the sound has to be synthesised in order to create a sound output rather than sound features, that can be hard to interpret.

Vocoders are methods of synthesising human voice. The conventional vocoders are based on the decomposition of human voice into periodic and aperiodic signals and then reconstructing the voice from these. In recent years deep neural networks have proved their impressing abilities as vocoders.

In this project both a conventional vocoder WORLD (2016) and an ANN vocoder WaveRNN (2018) will be used.

5.3.1 WaveNet

Perhaps the most popular variant of a vocoder is a data driven vocoder, such as WaveNet. WaveNet is a deep neural network for generating audio waveforms that operates directly on raw audio wave form, which means we can skip the steps of using speech-to-text and vice versa. Before this audio was generated by sampling the phonetic sounds you

needed recorded sounds from a large corpus and concatenate them together to form the sound you wanted. This however resulted in audio that sounded very "robotic", as many people probably have experienced when listening to machine generated audio that possesses very little of the rhythm and flow that humans talk with. The idea behind WaveNet is to output the probability distribution that generated the training data, that is the raw audio which created the raw signal. The WaveNet was first introduced by Googles DeepMind in the paper "*WaveNet: A generative model for raw audio*" [44] in September 2016. A year later in November 2017 it was improved by a lot of the same people and published in the paper "*Parallel WaveNet: Fast High-Fidelity Speech Synthesis*" [45] where the process was made more than 1000 times faster and thereby opened possibilities for more real time applications as it expanded its sampling rate to 24 kHz meaning it could generate 24000 audio samples pr. second. It was in fact used to generate voices for Google Assistant. WaveNet is used as the vocoder in the original AutoVC paper [1]. However, the architecture of wavenet is a very deep network requiring an enormous amount of computations to produce an output. Training such a network and improving its generation speed is troublesome. Once again, a year later in 2018, Google Deepmind introduced a much sparse model for high fidelity audio sampling called WaveRNN [46]. They prove that WaveRNN is able of generating sound 24 kHz 16-bit audio 4 times faster than real time (4 second audio sampled every second) and with no significant difference in its' quality compared to WaveNet [46].
Due to the sparse architecture and speed of WaveRNN this will replace WaveNet as the vocoder coupled with AutoVC in this project.

WaveRNN
Sequential generative models achieve state-of-the-art performance within natural language processing [46]. Especially RNN's and LSTM's. By passing information on to subsequent layers these models learn the joint probability distribution of the data as a product of conditional probabilities over each sample [46]. WaveRNN introduce several ideas to increase the efficiency of these models without harming the quality. [46] The first idea is to reduce the number of computations for each sample. The second is to batch the input in order to make computations more parallel-like. WaveRNN is structured in a way, that allows splitting the input into batches with overlapping ends, which afterwards can be unfolded and put together again. Batching often reduces computation speed because weights are reused [46]. It is proven that batched sampling dramatically improves the synthesis speed of WaveRNN [46].
WaveRNN is a single layer RNN with gates similar to GRU's. This project depends on a public implementation of WaveRNN[3] featuring an upsampling and residual network. This WaveRNN has a slightly different architecture than the one described in the original paper [46].

Upsampling and Residual Networks
The main purpose of the two 'pre-networks' is to scale the input mel spectrogram to a temporal dimension matching the length of the output waveform, since each WaveRNN cell outputs only one temporal feature for each temporal input.

[3] Fatchord Github

The Residual network consists of 10 residual blocks of two 1d Convolutional layers followed by batch normalisation.
The Upsampling network consists of three upsampling convolutional layers, each followed with a stretch function stretching the outputs size.
Outputs of the Upsampling and Residual networks are fed to the WaveRNN, with the Residual network output split 4-ways along the spatial dimension giving each WaveRNN cell 5 inputs.

WaveRNN Cell
The WaveRNN combines two GRU's used as Residual blocks followed by two linear layers with ReLU activation functions and a final linear layer. The GRU's work as a conveyor belt for the cell state. The final output is sampled from a Logistic Mixture Distribution with parameters estimated from the hidden state. The output and the cell state are fed into the next cell. The structure of the WaveRNN cell is depicted below in figure 5.9.

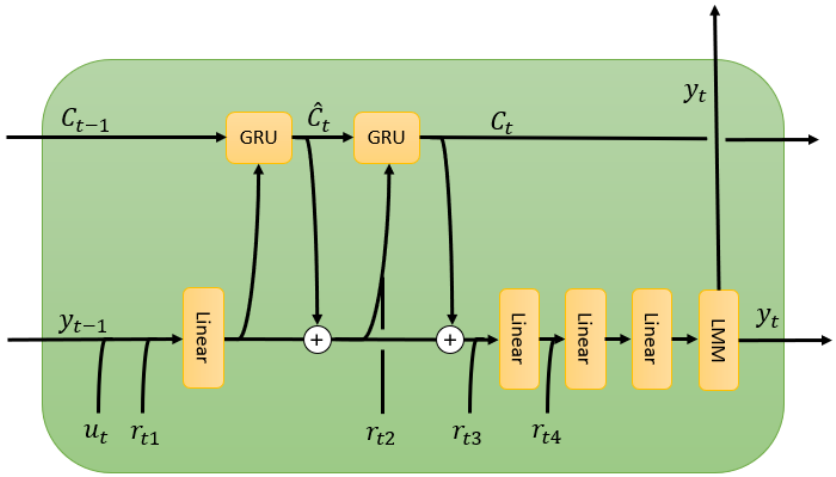

Figure 5.9: *WaveRNN*: u_t is input from upsample network, r_t is input from residual network, C_t is cell state, y_t is output and *LMM* denotes Logistic Mixture Model.

5.3.2 WORLD

WORLD is state-of-the-art within the conventional vocoder regime as it has real time implementation potential and outperforms other conventional vocoders in synthesis quality [15].
A slight modification of WORLD is used as the vocoder coupled with StarGAN [2] and will also be used with StarGAN in this project. WORLD differs from WaveRNN since it is not data driven but rather an algorithm using vocal cord characteristics, aperiodicity and a fundamental frequency in order to synthesise human voice. It uses the algorithms *CheapTrick* [19] to extract the spectral envelopes as vocal chord characteristics, *D4C*

[20] for aperiodicity estimation and *harvest* [18] for estimating the F_0 contour.
WORLD restores the vocal chord vibrations (the waveform) by a convolution of spectral envelope information and aperiodicity information. When restoring a waveform from a spectrogram the temporal positions of the vocal chords vibrations must be determined since much temporal information is lost when applying the DFT. WORLD uses the F_0 contour information to estimate these temporal positions.

5.4 Implementation of the models

The models introduced in this paper was implemented in Python using PyTorch. All experiments were performed using the DTU HPC clusters and training was performed on a Tesla V100 GPU[4] as the code supports GPU training on CUDA. The Python version and packages used in this paper is seen below and the full code can be found at this papers GitHub repository[5] for further clarifications.

AutoVC

- Python 3.6.2
- PyTorch 1.4.0
- librosa
- tqdm
- torchvision
- webrtcvad

StarGAN

- Python 3.6.2
- PyTorch 1.4.0
- librosa
- tqdm
- torchvision 0.5.0
- tensorboardX
- tensorboard
- SoX[6]
- PyWORLD

5.4.1 Implementation of AutoVC

The implementation of the content encoder and decoder of AutoVC in this paper follows the one proposed by the authors of [1] with slight modifications. The source code is obtained from the GitHub Repository[7] by Kaizhi Qian and modified under the MIT License[8] provided by the owner of the repository.

Source code to the speaker identity encoder is borrowed from the Real Time Voice Cloning GitHub Repository[9] and used under the MIT License[10] provided by the owner of the repository.

The vocoder used for with AutoVC is the Fatchord PyTorch implemention of WaveRNN [46] from this GitHub Repository. The code is used under the MIT License[11] provided by the owner of the repository.

[4] https://www.hpc.dtu.dk/?page_id=2759
[5] https://github.com/Fagprojekt-Deep-voice-conversion/Deep_voice_conversion
[7] https://github.com/auspicious3000/autovc
[8] https://github.com/auspicious3000/autovc/blob/master/LICENSE
[9] https://github.com/CorentinJ/Real-Time-Voice-Cloning
[10] https://github.com/CorentinJ/Real-Time-Voice-Cloning/blob/master/LICENSE.txt
[11] https://github.com/fatchord/WaveRNN/blob/master/LICENSE.txt

Preprocessing
As input the *speaker identity encoder* takes 40 channel mel spectrograms, with a sampling rate of 16000 Hz, STFT intervals of length 400 samples and a hop size of 160 samples.

The input to the *content encoder* is as follows: 80 channel mel spectrograms, with a sampling rate of 16000 Hz, STFT intervals of length 1024 samples and a hop size of 256 samples, concatenated with the speaker identity embedding of a second mel spectrogram belonging to the same speech segment along the channel axis. The second spectrogram has the specifications fit to the speaker identity encoder. All spectograms are created using the librosa library in python. The mel spectrograms X to the *content encoder* are transformed into dB by the following formula

$$X_{dB} = 20 \cdot log_{10}\left(max(1e-5, |X|)\right) - dB_{ref} \tag{5.29}$$

with the reference power dB_{ref} set to -16.

Training
The *speaker identity encoder* is pretrained using an Adam optimiser on the GE2E loss, with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and learning rate $\alpha = 1e-4$ with a batch size of 64 for 1.56 M steps, on the VoxCeleb[12] and the LibriSpeech[13] datasets (english voice only) [47]. It needs to be emphasised that the *speaker identity encoder* is trained independently of AutoVC and all credits go to the authors of [47] and the pretrained model is found in the belonging Github Repository.

The *content encoder* and *decoder* are trained on the loss function

$$\min \ (L_{recon} + \mu L_{recon2} + \lambda L_{content}) \tag{5.30}$$

with $\mu = 1$ and $\lambda = 1$, and L_{recon}, $L_{content}$ and L_{recon2} are defined respectively in eq. 5.8, eq. 5.9 and eq. 5.12.

The model is trained on mini batches of size 2 using an Adam optimiser with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and an initial learning rate $\alpha_0 = 1e-3$. Furthermore, Noam learning rate decay (eq. 5.31) is used as a annealing learning rate schedule. This starts by increasing the learning rate linearly for some training steps (in this case 4000) and afterwards decaying the learning rate exponentially. This is visualised in figure 5.10 and described by formula as:

$$\alpha_{t+1} = \alpha_0 \cdot 4000^{0.5} \cdot \min(t \cdot 4000^{-1.5}, t^{-0,5}) \tag{5.31}$$

In addition gradient clipping is applied, forcing values of the gradient to be within the interval $[-1; 1]$ by setting values below the interval to -1 and values above to 1. Finally an exponentially decaying average of the network parameters θ (eq. 5.32) is saved at each step during training and used for evaluation (i.e. in the experiment).

$$\widehat{\theta}_{t+1} = 0.9999 \cdot \widehat{\theta}_t + (1 - 0.9999) \cdot \theta_t \tag{5.32}$$

[12] http://www.robots.ox.ac.uk/~vgg/data/voxceleb/
[13] http://www.openslr.org/12

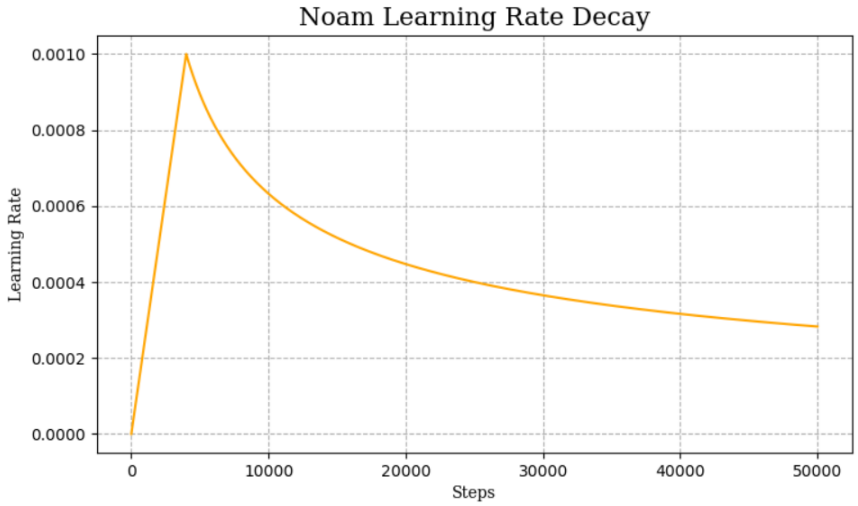

Figure 5.10

AutoVC is trained on the full VCTK-corpus with a batch size of 2 for 200k steps (about 9 epochs) with the training procedure described above. To test the convergence properties of the models, 4 different seeds was tested on which is visualised in figure 5.11. From the plot, the models seems to converge rather quick across different seeds. Since the model seems to perform equally well during training across all the tested seeds, seed 40 is chosen arbitrarily for the experiment.

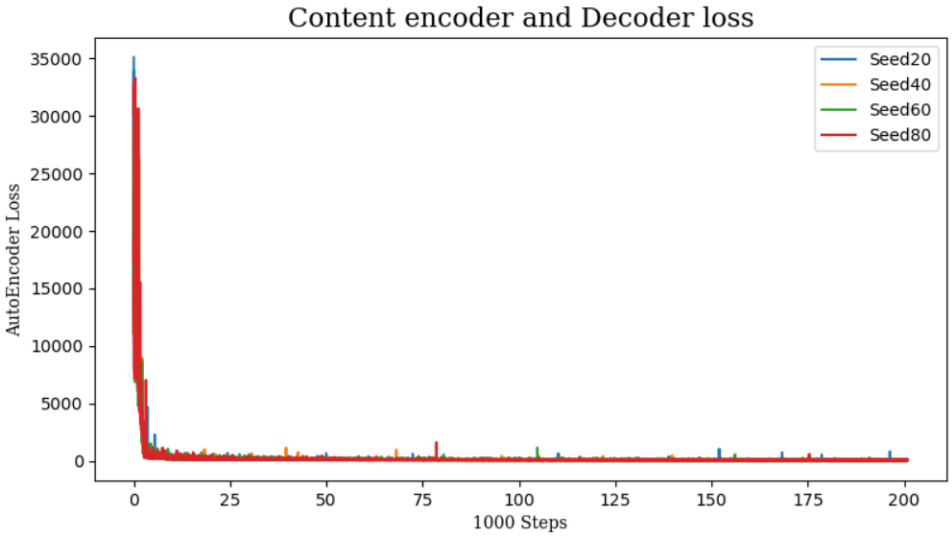

Figure 5.11: AutoVC training loss for 4 different seeds

Conversion
AutoVC needs a speaker identity embedding provided by the speaker identity encoder and an utterance. The utterance for conversion follow the prepossessing scheme as when training. The identity embedding of a speaker is the mean embedding of all identity embeddings belonging to that speaker, referred to as the voice print in [34]. Conversion is then simply performed by providing AutoVC with a source utterance and a target speaker identity voice print.

5.4.2 Implementation of StarGAN

The StarGAN model described in section 5.2.2 uses [2] as a large motivation all though some changes has been made, including a change of some the loss functions used. This paper uses the source code from liusongxiang's implementation on GitHub[14] which is adapted to work for the purpose of this paper.

Preprocessing
To preprocess the data, all sound files for a certain speaker is placed in a folder with the speakers name and these are then resampled to 16000 Hz in a new folder with same structure. Here folders for the test speakers and the VCTK-Corpus are stored together. In order for the model not to be trained on all data, leaving some for test conversions, the data is split into a train and test set. For each file the corresponding normalised spectral envelope created with 36 MFCC is stored as a NumPy file and saved in either the train or test folder according to the train-test split made. Furthermore, the mean and standard deviation of the spectral envelope (MFCC) along with the mean and standard deviation of the fundamental frequency for each speaker is denoted the speaker stats and are saved for the conversion process.

Training
To train the generator and discriminator, the training process described in section 5.2.2 was implemented with PyTorch. The discriminator was build to both output the real/-fake discriminator output aswell as the domain classification of a training batch size of 32, which was used to calculate the different losses. In this paper the sum of speakers from the VCTK-Corpus and the test speakers is 117, meaning there are 117 different domains, making the StarGAN structure especially useful due to the many domains. The discriminator uses LeakyReLU with a slope value of 0.01 for the negative values and the discriminator is trained by calculating the adversarial loss, domain classification loss and gradient penalty loss and adding these after regularising with specified values of λ to get the discriminator loss $\mathcal{L}_D(D)$. The λ values used were $\lambda_{cls} = 10$ and $\lambda_{gp} = 10$. Then the weights are updated using the Adam optimiser with $\beta_1 = 0.5$, $\beta_2 = 0.999$ and a learning rate of $\alpha = 0.0001$. Each time one train batch goes through either discriminator or generator training, it is called a step. For every time the discriminator was trained for 5 steps, the generator was trained for 1. The generator was trained by calculating the generator loss $\mathcal{L}_G(G)$, which was made up of an adversarial loss, a reconstruction loss and a classification loss that were added with regularising parameters λ aswell. The λ values used for the generator was $\lambda_{rec} = 10$ and $\lambda_{cls} = 10$. The weights could then

[14] https://github.com/liusongxiang/StarGAN-Voice-Conversion

again be updated with an Adam optimiser that used the same parameter values as the one for the discriminator. To get an annealing learning rate, the learning rates were updated every 1000 step for the last 100000 steps. This was done by simply subtracting $\frac{\alpha}{100000} = \frac{0.0001}{100000} = 1e^{-9}$ from the learning rate each time the learning rate was updated.

Conversion

The speaker stats are loaded for both the source and target speaker along with the sound file to be converted. The spectral envelope, fundamental frequency and aperiodicity are then extracted from the file and the spectral envelope is normalised before being fed into the generator. The generator will then use the normalised spectral envelope and the target speaker label to generate a converted spectral envelope. Now the converted spectral envelope and normalised fundamental frequency can be unnormalised with the target stats and passed to the World vocoder along with the source aperiodicity to create the converted sound file.

Getting data for the experiments

To get the final model to use for converting the test sound files, three different models were trained on 30 min of each test speaker with the seeds 1000, 2000 and 3000. When looking at figure 5.12 no model seems to be significantly better than the other, therefore the model trained with seed 1000 is chosen arbitrarily. Furthermore, the model trained for 175000 steps is chosen as this seems like the step where the model is closest to somewhat of a convergence. For the experiment regarding the amount of training data needed, two models with 10 and 20 min of test speakers are trained with seed 1000.

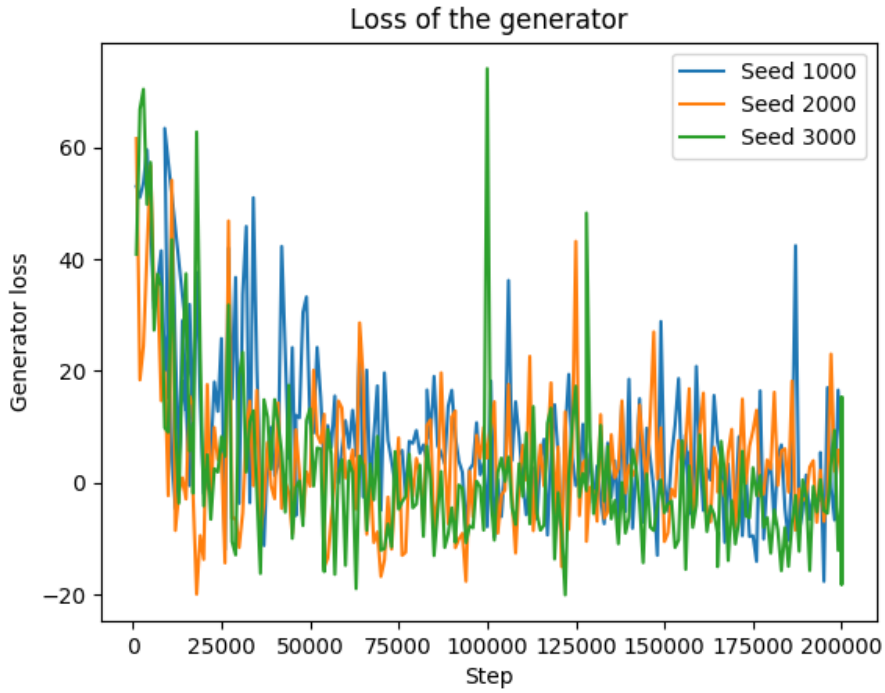

Figure 5.12: The generator loss for 3 different seeds with 30 min of each test speaker and the VCTK-Corpus as train data

To get some converted sound files for the test speakers to use for the experiments, 12 sound files were handpicked from the test speakers to avoid files without speech and converted with the chosen models.

6 Experiment

To evaluate the models some experiments were designed to get an subjective idea about how well the models performed. For this a survey was created as a R shiny app that presented the participants with various sound files and questions and the results were saved with the participants consent. To make sure the participant gave consent to use the data obtained, a check off box was provided on the survey front page as seen in appendix A.2 figure A.1.

6.1 Experimental Design

The experiment is designed in a way to acquire data for the quality of and how convincing the voice conversions were by ratings from human listeners. The tests look at conversions within the Danish and English language and compares the conversions of AutoVC and StarGAN between the female and male gender

To test do this, the experiment is divided into 2 sub experiments, the first called ”the similarity experiment” and the second ”the fooling experiment”. Here the setup of the similarity experiment was used to evaluate the effect of the amount of training data as well called ”the amount of training data experiment”.

For both the similarity and fooling experiment conversions was tested both across and within the male and female gender giving 4 possible combinations. This was done both within the Danish and English language resulting in a total of 16 different possible conversions when both models were used.

The participants of the experiments were asked to give some basic information such as:

- Age
- Gender (M/F/other)
- Time and date of submitting

6.1.1 The similarity experiment

To investigate how convincing a voice conversion was compared to a real recording the similarity experiment was used. This gives a subjective measure of how well the models converted the voices.

This experiment investigates this by having 2 audio files from the same speaker, one converted from a source and the other a real recording from the target speaker. The recording from the target had been synthesised by a vocoder matching the model of that question to reduce a bias created by using differnet vocoders for the two models. The participants were presented with slider from 0-5 to answer how similar the two files sound. 0 indicated high confidence that the two voices *did not* originate from the same persons and 5 indicated high confidence the two voices *did* originate from the same

person. A sub question was also attached where the participant were asked to judge the sound quality of a sound clip. This was done to determine if the converted sound clips overall quality were good or if they could have been better as this could induce a bias.

Not only converted voices but also a baseline was tested in this experiment. The baseline was made with non converted voices synthesised with the vocoders to reduce the bias introduced by the vocoders when testing the conversion quality. In addition this also tested the participant judgement of voices when they actual belong to the same speaker. The overall layout of the similarity experiment is shown in figure 6.1 where an arrow from M to F meant a male was converted to a female. This resulted in 16 questions for the two models and 8 for the baseline models giving a total of 24 questions.

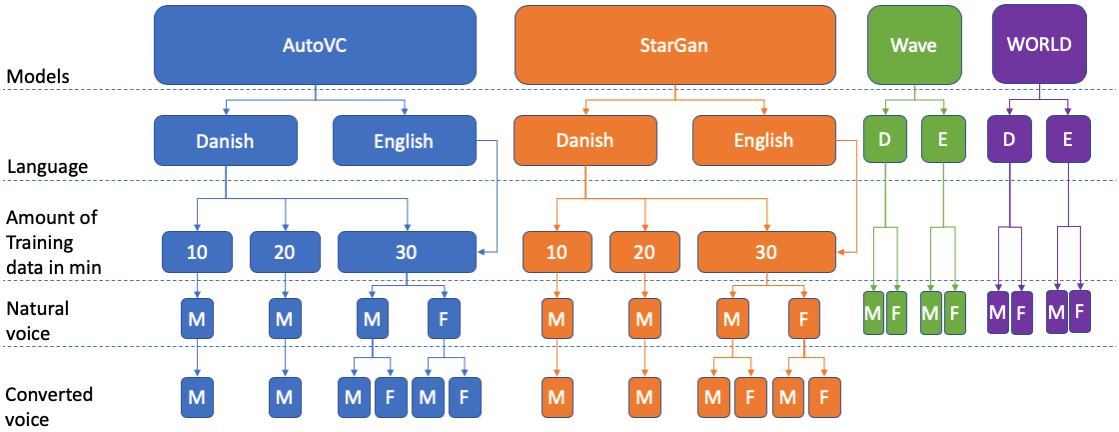

Figure 6.1: The structure of the similarity experiment

6.1.2 The Fooling experiment

The experiment was designed to look into how well voice conversion for the two models works in terms of ”fooling” the public and hence works as a variation of a Turing test. Each participant was presented with each of the 16 possible conversions as seen in figure 6.2, meaning there was 16 questions for this experiment. In each questions a converted speech will be paired up with a true speech from a speaker. The true speech will be synthesised by the vocoder matching the model tested to reduce the bias of the vocoder. Both the synthesised and converted fiels was chosen at random from 10-12 possible utterances for each speaker. This was done since comparing exactly the same sentences could make it easier to distinguish between real and fake. This will in resulted in a ”fooling rate” giving a percentage of how well the voice conversion for each model in each category is at fooling the general public.

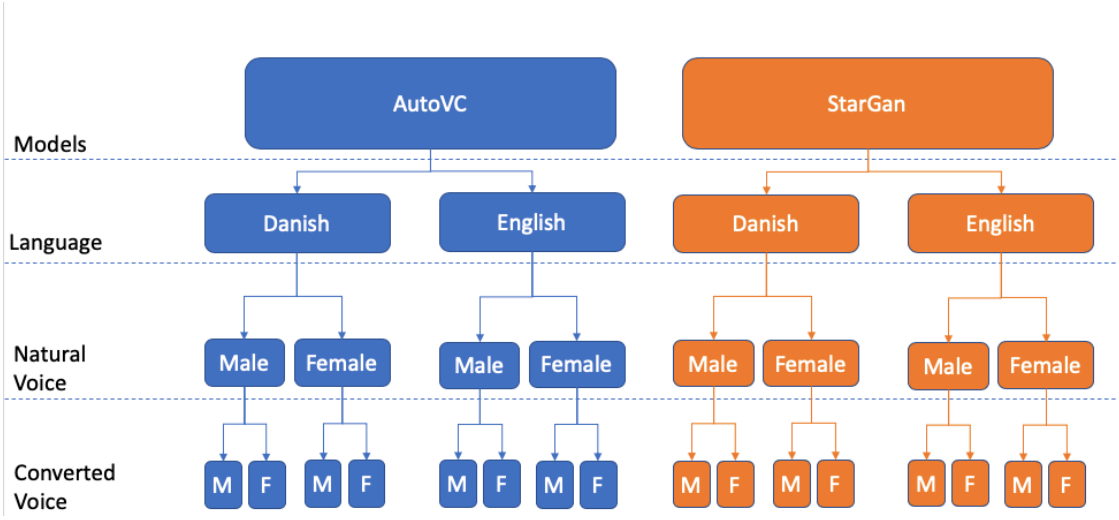

Figure 6.2: The structure of the Fooling experiment

6.1.3 The amount of training data experiment

As this paper wished to look into the effect of using different amounts of training data for the models, an experiment similar to the similarity experiment was designed. This had the same setup as the similarity experiment. However this did not include the vocoder baseline as this was experiment only concerned itself about the difference between models trained on different amounts of data and not the overall succes. To reduce the amount of questions for the participants and possible speaker biases, this only involved conversions from Anders Fogh Rasmussen and Lars Lykke and vice versa as indicated by figure 6.1. Since the Danish male to male conversion for a model trained on 30 min of data was included in the similarity experiment, only 4 questions were added, where each was a combination of either a model trained on 10 or 20 min and the model used.

6.2 Statistical tests

For the statistical handling of the results, R was used to easily process the data.

6.2.1 Fooling Score - Binomial Distribution

The fooling score results are binomially distributed with the total number of success measured as the number of people fooled. In the test people were assigned with random converted and real voices within the same category. This random assignment of test elements make the McNemar test not suited for comparing the two models' "foolness" scores, hence the statistical evaluation will rely on Jeffreys intervals as 95 % binomial proportion confidence intervals.

6.2.2 Similarity and Quality score - Mean Opinion Score

The data is categorical $y_i \in \{0, 1, 2, 3, 4, 5, 6\}$ with y_i as user score and hence not believed to be normally distributed. To check this assumptions the Shapiro Wilk test was used on

the different groups of data and showed that none of the groups were normally distributed as assumed (see the appendix A.3).

Since participants rated similarity and quality of the sound clips with a slider from 0 to 5 the mean opinion score (MOS) is an ideal choice of statistical test. The MOS has through time been used for speech and audio quality, especially for telephone quality by subjects sitting in a quiet room [1]. The same thing was done in the literature [1], which is why it is also being used here to look at the same aspect of the converted sounds naturalness. The MOS wants to represent the overall quality/opinion of performance of a system. By giving a score varying from 0-5 (0 equals inaudible and 5 excellent naturalness), it can be easier to visualise how well the conversion has performed in the eyes of the public. The MOS is defined as (with n_i being the amount of user rating for category i):

$$\hat{Y} = \sum_{i=1}^{k} i\frac{n_i}{n} \tag{6.1}$$

Confidence interval estimators of MOS can be rather troublesome since often the data is not normally distributed and it is bounded with the score range. In [48] different confidence interval estimators based on the binomial and multinomial distribution are proposed including a variation of the Jeffreys interval [48]. They also show that bootstrapping is a solid confidence estimator especially for larger sample sizes. The statistical evaluation of the Similarity and quality scores will rely on bootstrapping as a confidence interval estimator. Due to the data not being normally distributed group means can not be compared using ANOVA.

6.2.3 Confidence Intervals and Plots

Jeffreys Intervals

The 95 % Jeffreys intervals were calculated using R's BinomCI. The formula for upper and lower bounds are given as

$$\theta_L = \text{cdf}_B^{-1}\left(\frac{\alpha}{2}|m+\frac{1}{2}a, b\right), \quad \text{cdf}_B^{-1}\left(1-\frac{\alpha}{2}|m+\frac{1}{2}a, b\right), \quad a = m+\frac{1}{2}, b = n-m+\frac{1}{2}$$

with n being the number of test subjects, m number of people fooled and $\alpha = 0.05$.

Bootstrapped confidence interval estimator

The means of similarity and quality ratings for each conversion type were bootstrapped by sampling with replacement $1e6$ number of times. From this distribution of means lower and upper confidence bounds are chosen as the $\frac{\alpha}{2}$ and the $1-\frac{\alpha}{2}$ percentiles respectively.

Plots

To get a better visual representation of the results plots for each experiment were created.

[1] Mean Opinon Score

Normal bar plots were created with the 95% confidence interval applied as to visualise the potential error.
In addition to this a scatter plot of mean opinion scores grouped of similarity grouped over the quality ratings was created to visualise the effect of quality on the similarity ratings.

7 Results

In this section the results of the experiments are presented. To make the plots more neat, some abbreviations are introduced. For the similarity experiment and fool test, the conversion types are denoted "$language - gender_1 - gender_2$" meaning $gender_1$ was converted to $gender_2$ in a certain language. The language options were "D" for Danish and "E" for English and for the genders "F" was used for female and "M" for male. With this convention the abbreviation "DFM" means that the Danish female speakers were converted to the male Danish speakers. For the vocoders "DF" simply means the Danish female speakers went through the vocoder without any conversion. In the experiment regarding the effect of the amount of training data from the test speakers used, "10 min" simply means, that data samples corresponding to 10 minutes were used in the training of the model.

7.1 The similarity experiment

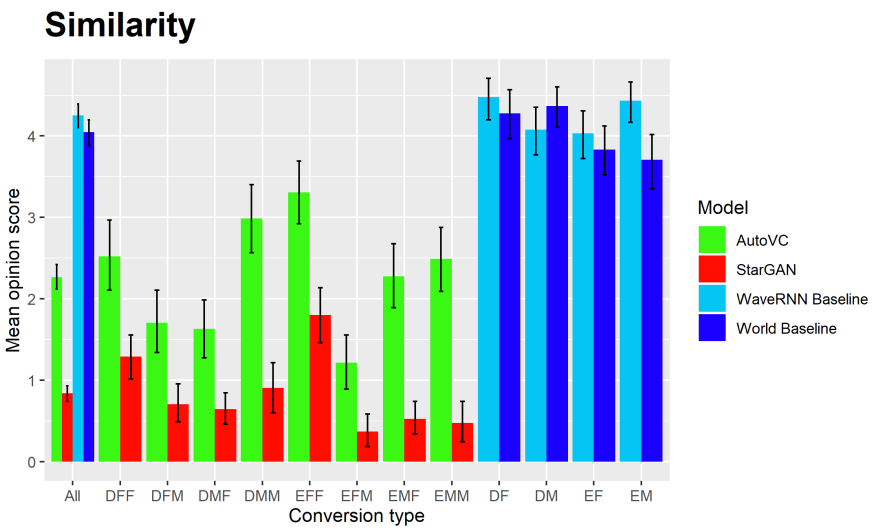

Figure 7.1: The similarity results of the similarity experiment comparing the different models, the explicit values can be seen in table A.4 in appendix A.4

Figure 7.1 shows MOS of similarity ratings between a converted and a original sample from a test speaker with the respective confidence intervals. When comparing the MOS for all conversion types ("All" in figure 7.1), AutoVC acheived a MOS of 2.27, while StarGAN got 0.84, WaveRNN baseline 4.25 and World baseline 4.05 (see appendix A.4), indicating AutoVC performed best of the two implemented models and that the vocoders did not have a significant difference in performance.

It also shows that AutoVC performs significantly better than StarGAN within each conversion type, as the confidence intervals do not overlap and no AutoVC conversion is below the mean (”All”) MOS of StarGAN. The only thing in favour of StarGAN is the WORLD vocoder being better in the danish male category, however, not significantly.

AutoVC performs best when the gender domain remains the same as the source, as all these MOS scores are larger than the average (”All”) opposed to the conversions where the gender is changed and the MOS is below average (see A.4).

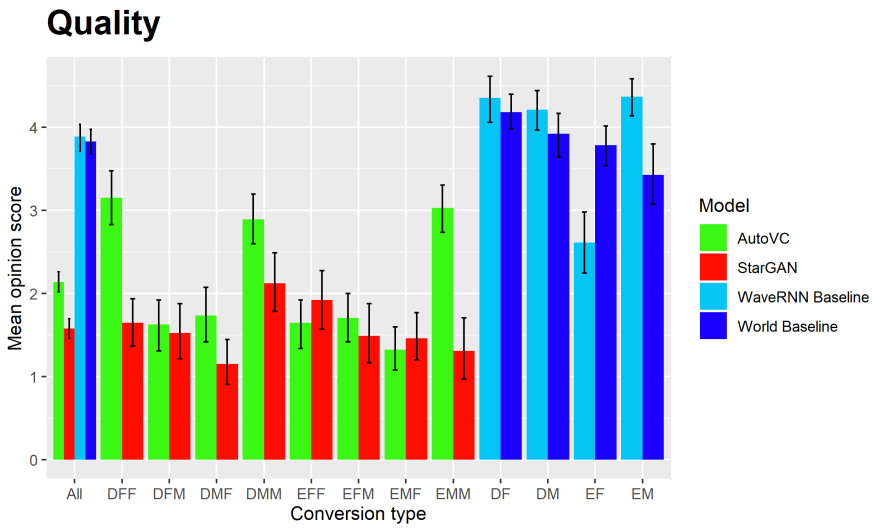

Figure 7.2: The quality results of the similarity experiment comparing the different models, the explicit values can be seen in table A.5 in appendix A.4

Figure 7.2 shows the MOS of the quality rating of a converted sample from one of the test speakers. When comparing the mean scores for all conversion types (”All” in figure 7.2), AutoVC achieved a MOS 2.14, while StarGAN got 1.58, WaveRNN baseline 3.89 and World baseline 3.83 (see appendix A.4), indicating AutoVC performed the best of the two implemented models and that the vocoders did not have a significant difference in performance.

AutoVC outperforms StarGAN in most of the conversion types when it comes to quality, but not as significantly as when it came to the similarity results. StarGAN performs slightly better for ”EFF” and ”EMF”, however, not significantly. StarGANs vocoder WORLD performs significantly better then WaveRNN for English female conversions and the opposite is true for English males.

The quality of the conversions seems to stand out AutoVC, when it comes to male-male conversions for both languages and female-female conversions in Danish. These conversions performs significantly better than all other conversions.

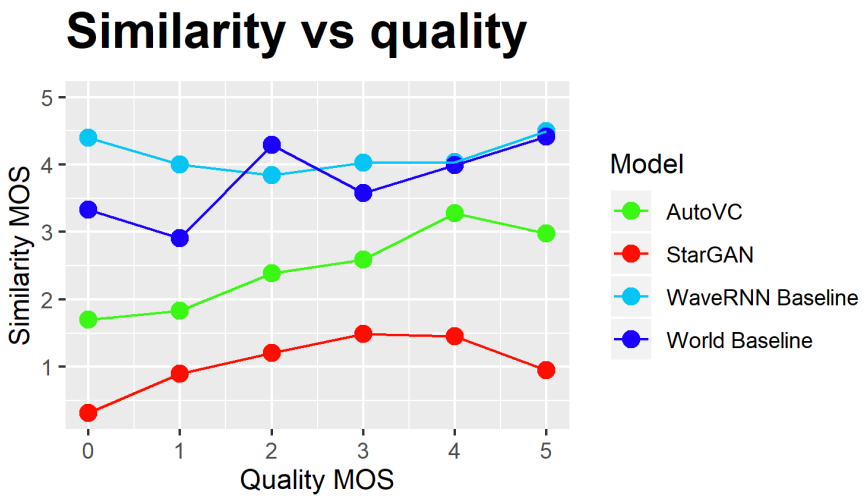

Figure 7.3: The similarity scores compared to the quality scores

Figure 7.3 shows the mean similarity score for each quality opinion score. Generally the baseline models seems to be more similar and therefore more natural sounding regardless of the quality. Only AutoVC shows a small trend of getting better similarity results, when the quality is better, nevertheless, this is not a large trend and none of the of the models seems to gain anything from having a better quality.

7.2 The fool test

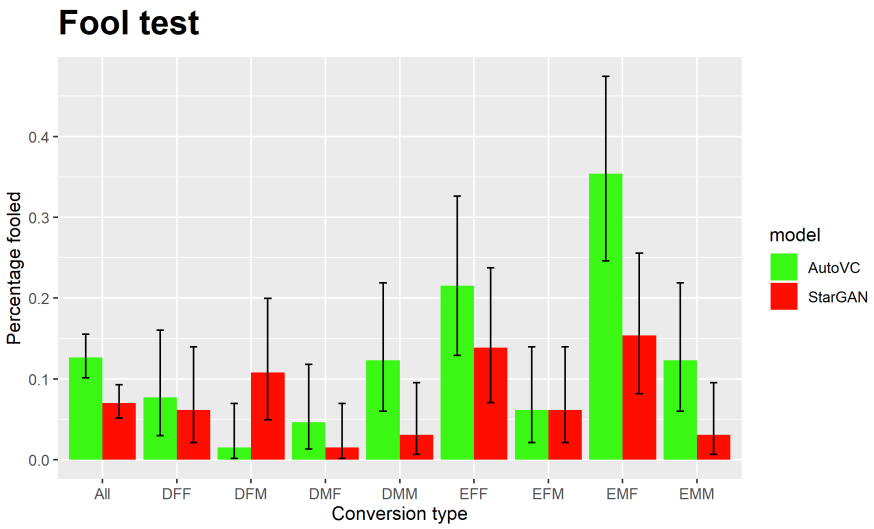

Figure 7.4: The result of the fool test, the explicit values can be seen in table A.6 in appendix A.4

Figure 7.4 shows the percentage fooled of each category for both AutoVC and StarGan with the applied confidence intervals. When comparing the mean score of all categories (”All in figure 7.4), AutoVC fooled 13 % and StarGAN fooled 7 %. Furthermore, as seen in figure 7.4 the greatest ”fooler” was the conversion from a English speaking male to female (EMF) using AutoVC where 23 out of 65 (35%) participants choose the converted audio file as being the natural one (see appendix A.4). It also seems from the bar plot that AutoVC is superior compared to StarGan across nearly every conversion although the confidence intervals show many of these are not significant. In total AutoVC fools significantly more people than StarGAN

Some of the conversions has confidence intervals that includes 0, which means these might not have fooled anyone.

7.3 The amount of training data experiment

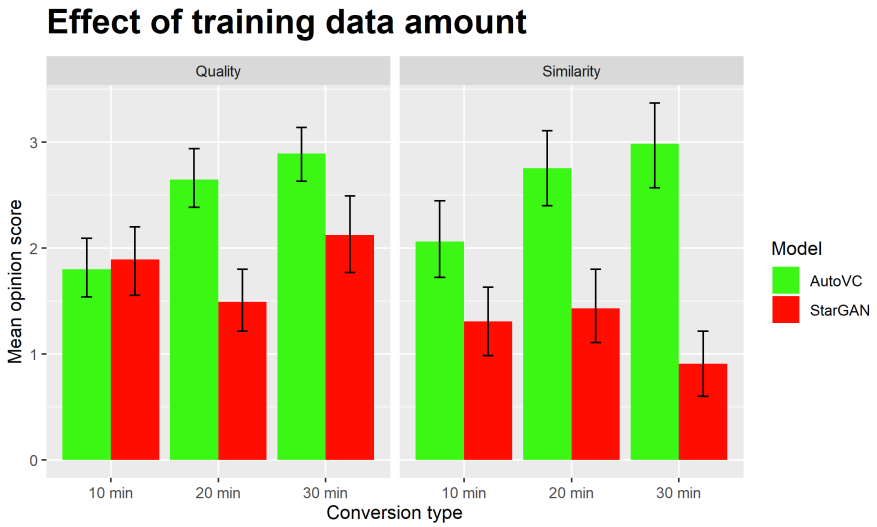

Figure 7.5: The effect of training on different amounts of data for the AutoVC and StarGAN model, the explicit values can be seen in table A.4 in appendix A.4

The effect of training data shown in figure 7.5 shows how the models did in the two different MOS's when trained on different amount of data. This plot indicates a growing trend of both the similarity and quality when it comes to AutoVC and a random trend for StarGAN when comparing the MOS for models trained on different amounts of training data from the test speakers.

8 Discussion

8.1 The similarity experiment

The similarity test results compared to the AutoVC paper [1] is more unfavourable here. The AutoVC paper has around all their similarity scores above the middle rating where the results from this experiment in general is right below the middle rating. StarGAN also performs worse than the results in [1]. AutoVC outperforms StarGAN significantly in both quality and similarity. This agrees with the results in [1].
A reason for the lower quality might be due to original quality of the data. AutoVC originally uses the VCTK-corpus which all is recorded within a hemianechoic chamber, whereas the data used in this project consists of political speeches with more noise. Furthermore, the authors of the papers did not provide detailed code as the code for AutoVC had missing parts such as the speaker identity encoder and a training function and the code for StarGAN was implemented by external sources where modifications to e.g. the architecture was made. This lead to creating a greater patchwork of code, trying to replicate the models found in the literature, which induced inconsistencies between the model from the literature and the coded model used for this paper. Interestingly both AutoVC and StarGAN peaks in similarity ratings in *"EFF"*, which is a category where the Baseline *WaveRNN* drops dramatically in quality. This indicates high similarity might not be because of good conversions but rather low original audio quality since it is hard to recognise the real voices due to noise. It is also the English female category in which the most distorted audio clips are found. Acquiring more 'clean' data or reducing the noise levels could have had an effect on the overall outcome of the similarity experiment. The models might have performed better and it would have been a more fair evaluation for the test subjects. In addition Michelle Obama's and Hillary Clinton's voice might not be that familiar as the ones of Barack Obama and Donald Trump making the similarity question more difficult in the English females category.

Besides the bias of the voice being synthesised after conversion both models perform significantly worse than the baselines in both quality and similarity, indicating the conversion process significantly reduces the voice quality. The vocoders prove their worth as human voice synthesisers with high quality ratings and they are not significantly different, which can be seen as a win to the conventional vocoder in a time where deep learning is conquering the field of vocoders.

It was hypothesised that good quality meant high similarity ratings, since the converted speech would sound more natural then. However, no trend seems to be found in figure 7.3.

8.2 Fooling rates

In general, the models performed below 20% in each of the conversion categories when it came to the fooling test shown in figure 7.4. It is worth mentioning, that if we had

perfect voice conversion the result should be a fool percentage of around 50% since it at this point would be a pure guess for a participant whether sound A or B represented the natural recording. It is only for the English male to female conversion where both StarGAN but especially AutoVC does their best performance. This category consists of Barack Obama and Donald Trump audio files being converted into either Hillary Clinton or Michelle Obama sound files. A reason for this spike in performance may be due to the English female voices may be the ones people are having the most difficulty recognising assuming the majority of our participants being Danish. It was also discovered that for some reason when the Michelle Obama audio file went through the vocoder, the quality of the file went down a lot. It became really distorted which resulted in the conversions as well as baseline sounds distorted as seen in low performance of the baseline model for WaveRNN and English females in figure 7.2. This means that there is a possibility that when the baseline and a converted file are being tested against each other, they may both sound distorted, which could result in a difficulty spike for finding the non converted audio file.

8.3 The amount of training data experiment

The growing trend of both quality and similarity in figure 7.5 for AutoVC, qualitatively seems to have similar slopes, which might mean that both of these rely on the training data amount. However, this is more likely an outcome of the trained model becoming better with more training data and thereby outputting a converted voice that is both of better quality and more similar to the target speaker. Furthermore, the confidence intervals are overlapping between almost all of the scores and a significant conclusion can thereby not be made. This applies to StarGAN as well. Therefore it is far from guaranteed that adding more training data would give better results. To investigate this further, 3 models is too few models to discover a significant trend and more models would have to be trained. Due to the time frame of this project this was not done.

One of the reasons why AutoVC might have a trend of becoming better with more training data and StarGAN did not, could be found in the way the two models are trained. AutoVC is first trained on solely the VCTK-Corpus and afterwards on the test speakers whereas StarGAN is trained on both training sets from the beginning. This means AutoVC has a second wave of training where the weights of the networks are adjusted to only the test speakers while StarGANs weights are adjusted to all speakers at once. It could also be the reason why AutoVC is significantly better for most conversion types. To further investigate this hypothesis, one could have made a test on how the models perform on the original speakers from the VCTK-Corpus, as this hypothesis would suggest the weights of StarGAN being adjusted to all speakers, where AutoVC is more focused on the test speakers used. Here StarGAN should perform better and AutoVC worse compared to the results obtained in this paper, meaning the gap between the two would decrease, if this is one of the reasons this trend appears. Another way to make sure StarGAN is not affected by that many different speaker identities, could be to use more data from fewer speakers, such that StarGANs generator does not have to learn as many mappings between attribute labels and thereby a less complicated knowledge of

domain mappings would be needed. The AutoVC paper [1] trains the StarGAN model on the VCTK-Corpus although the original StarGAN paper [2] only used 6 male and 6 female speakers for conversion, which could indicate that less speakers used in the training could be an advantage.

8.4 Speaker bias

Since the voices of famous speakers are being used slight bias may have been introduced. It is assumed that people in general know of these people and their voices. This could easily make it way easier to deduct if a voice is converted. It is likely, that when a famous voice just sounds a little bit different from what we are used to hear it we would generally classify it as a fake. On the other hand, when having this approach the results of a fooling is also more impact full as it shows that a voice that is known well can be replicated. This could very well be a factor of a generally lower similarity rate across the board than seen in the AutoVC paper [1]. In the AutoVC paper test subjects are presented with speaker from the VCTK-corpus and hence most likely unknown to the subjects. Distinguishing between unknown speakers is more difficult and would hence lead to higher similarity scores. Unfortunately the authors of [1] do not test a general baseline of how well people in reality are when it comes to distinguishing between two speakers.

8.5 Zero-shot conversion

AutoVC is not only the model which performs best overall it is perhaps also the more simple. Not in the number of trainable parameters, which in total is approximately 29.8 M in AutoVC compared to approximately 21.3 M in StarGAN, but in the way it tackles the problem. GAN training is not known to be stable which is emphasised in figure 5.6 and StarGAN requires domain knowledge limiting it to only known speakers. AutoVC converges quickly and stable as seen in figure 5.11 and the idea of using an Autoencoder for voice conversion is theoretically justified [1] and liberates the conversion from being within specific domains. Ideally, AutoVC should be capable of perhaps the strongest form of voice conversion, the *zero-shot* [1]. This type of voice conversion is without any domain knowledge, i.e. neither source nor target has been seen during training.

This paper compares AutoVC and StarGAN and since StarGAN is not capable of zero shot this is not investigated. Furthermore a few subjective tests with AutoVC zero shot showed that proper conversions were not achievable. However, to some degree zero shot was actually investigated since the speaker identity encoder was never trained on any of the speakers, neither the ones in the training nor in the test. Still it had the ability to extract speaker identity of unknown subjects which the similarity experiment proves. Training the speaker identity encoder on test subjects would most likely increase the performance of AutoVC and the lack of doing so in this paper, might explain partly why AutoVC does not performs as well as in the original paper [1].
Originally the idea of the speaker encoder was used for speaker verification to distinguish between different speakers based on a 256x1 vector [34]. AutoVC shows how much more

information is actually available in those numbers - a speaker identity could be recreated based on them.

With zero shot, VC could simply be done between any pair of persons which would eliminate the idea of a *voice fingerprint*. Relying on voices as identities is an important part of the social nature of human beings. We know the voices of our loved ones, we respect voices of authority and we trust familiar voices. Being able to switch to any voice as simple as by flipping a switch could have devastating consequences for the society. Decision are made upon orders given by a 'voice in power' and everyone is exposed of identity theft.
Deep fakes introduces a need for improving general personal data restrictions. Even if consent is given to share personal data, consent should not simultaneously be given to synthesising personal data, which it is done with VC.

Fortunately, there seems to be much more to our identities and our voices than the state-of-the-art models are able to generate convincingly.

8.6 Future Work

For the experiment, since this was made as an online survey, it would have been nice to how the participants listened to the audio files, e.g. make them give this information. Some may have used the speakers from their computer, which in general could lead to a less clear sound opposed to listening through headphones which would make the audio go straight through your ear canal and generally has better sound quality. This might have influenced the fooling an similarity results as two sounds with good quality are easier to distinguish opposed to two sound with low quality. Ideally, the experiment would have been in person to control for this matter, but because of the COVID-19 situation and time pressure, this was not achievable.

More participants would also have been preferred. It is not easy to persuade people online to take a survey, which resulted in 65 participants in total. Of the 65 participants most are probable to be friends and family of the authors of this paper, as these are easier persuaded. This leads to a bias towards the age group taking the survey, where most participants were between 20 and 30 years old. For a better representation of how well the models performed, more participants would be needed for more narrow confidence intervals.

Our experiment also used a MOS scale from 0-5. It is worth noting that the normal use of the MOS goes from 1 to 5. This makes the results in this paper a little difficult to compare to the once in the AutoVC paper ([1]) since they have the original scale of 1-5. It is a little unclear how much of an impact this has had on the results, but it should be possible to get a legit MOS from a different scale as it just gives more possibilities to the participant.

Another minor slip up was the lack of a way to trace back the audio files given to the participant. This could have shown shown potential outliers within each group. E.g. the

hypothesis of some Michelle Obama files being worse quality than the other files could have been investigated.

Regarding the models, focus for improving AutoVC should be put on the speaker identity encoder and zero shot voice conversion. Moreover reducing the number of speakers and hence domains could perhaps increase the performance of StarGAN, which is worth looking into.

8.7 Conclusion

Two state-of-the-art models AutoVC and StarGAN and their respective state-of-the-art vocoders WaveRNN and WORLD were implemented successfully on a foundation of deep learning and digital signal processing theory. Both models were tested regarding their ability to perform VC in English and Danish within and across gender and quantitatively evaluated with a public survey which included a variation of a Turing test and public opinion ratings of the converted voices similarity with the targets voice and the quality of these. The results showed that AutoVC significantly outperformed StarGAN in almost every task, but it did not get close the baselines of synthesised speech by the vocoders. In the fool experiment, AutoVC fooled 13% of the public and StarGAN fooled 7%. In the similarity experiment, AutoVC achieved a MOS of 2.14, while StarGAN got 1.58 on a scale from 0-5. This concludes that the implemented models performed worse than the results from past literature. This is partly due to lack of information and specific details about the model implementation. The fooling rates did not leave an impression of the models being specifically convincing and this implementation of the models will not be suited for high scale VC deep fakes. Ethical considerations regarding VC are troublesome and the law full restrictions sparse. Extra care has to be taken when improving the existing technology, since our voice has been a fundamental part of our identities for such a long time that we rely on it as a fingerprint. Data has to be handled with care and many regulations have been introduced to prevent misuse of personal data, but regulations regarding synthesised personal data are yet to come.

References

[1] Kaizhi Qian et al. "AutoVC: Zero-Shot Voice Style Transfer with Only Autoencoder Loss". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, June 2019, pp. 5210–5219. URL: http://proceedings.mlr.press/v97/qian19c.html.

[2] Hirokazu Kameoka et al. "StarGAN-VC: Non-parallel many-to-many voice conversion with star generative adversarial networks". In: *CoRR* abs/1806.02169 (2018). arXiv: 1806.02169. URL: http://arxiv.org/abs/1806.02169.

[3] Alex Hern. *Facebook bans 'deepfake' videos in run-up to US election*. 2020. URL: https://www.theguardian.com/technology/2020/jan/07/facebook-bans-deepfake-videos-in-run-up-to-us-election.

[4] Yunjey Choi et al. "StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation". In: *CoRR* abs/1711.09020 (2017). arXiv: 1711.09020. URL: http://arxiv.org/abs/1711.09020.

[5] Takuhiro Kaneko et al. "StarGAN-VC2: Rethinking Conditional Methods for StarGAN-Based Voice Conversion". In: *CoRR* abs/1907.12279 (2019). arXiv: 1907.12279. URL: http://arxiv.org/abs/1907.12279.

[6] Catherine Stupp. *Fraudsters Used AI to Mimic CEO's Voice in Unusual Cybercrime Case*. 2019. URL: https://www.wsj.com/articles/fraudsters-use-ai-to-mimic-ceos-voice-in-unusual-cybercrime-case-11567157402.

[7] Edvinas Meskys et al. "Regulating deep fakes: legal and ethical considerations". In: *Journal of Intellectual Property Law & Practice* 15.1 (Jan. 2020), pp. 24–31. ISSN: 1747-1532. DOI: 10.1093/jiplp/jpz167. eprint: https://academic.oup.com/jiplp/article-pdf/15/1/24/33221931/jpz167.pdf. URL: https://doi.org/10.1093/jiplp/jpz167.

[8] ThinkDoTank Dataethics. *Data Ethics Tools for Companies and Organisations*. 2018. URL: https://dataethics.eu/tools/.

[9] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. USA: California Technical Publishing, 1997. Chap. 3. ISBN: 0966017633.

[10] Christophe Veaux, Junichi Yamagishi, and Kirsten MacDonald. *SUPERSEDED - CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit*. eng. 2017. DOI: 10.7488/DS/1994. URL: http://datashare.is.ed.ac.uk/handle/10283/2651.

[11] The danish ministry of state. *Nytårstaler*. URL: https://www.stm.dk/_a_1612.html.

[12] Michael E. Eidenmuller. *American Rhetoric*. URL: https://www.americanrhetoric.com.

[13] Maitreya Patel et al. *Ada{GAN}: Adaptive {GAN} for Many-to-Many Non-Parallel Voice Conversion*. 2020. URL: https://openreview.net/forum?id=HJlk-eHFwH.

[14] Lorenzo-Trueba JYamagishi JToda T et al. *The Voice Conversion Challenge 2018: Promoting Development of Parallel and Nonparallel Methods*. 2018. URL: https://arxiv.org/abs/1804.04262.

[15] Masanori MORISE, Fumiya YOKOMORI, and Kenji OZAWA. "WORLD: A Vocoder-Based High-Quality Speech Synthesis System for Real-Time Applications". In: *IEICE Transactions on Information and Systems* E99.D.7 (2016), pp. 1877–1884. DOI: 10.1587/transinf.2015EDP7457.

[16] James Lyons. *Mel Frequency Cepstral Coefficient (MFCC) tutorial*. URL: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/.

[17] *Speech Processing in Mobile Environments*. Springer, 2014. Chap. Appendix A.

[18] Masanori Morise. "Harvest: A High-Performance Fundamental Frequency Estimator from Speech Signals". In: (Aug. 2017), pp. 2321–2325. DOI: 10.21437/Interspeech.2017-68.

[19] Masanori Morise. "CheapTrick, a spectral envelope estimator for high-quality speech synthesis". In: *Speech Communication* 67 (Jan. 2014). DOI: 10.1016/j.specom.2014.09.003.

[20] Masanori Morise. "D4C, a band-aperiodicity estimator for high-quality speech synthesis". In: *Speech Communication* 84 (Sept. 2016). DOI: 10.1016/j.specom.2016.09.001.

[21] Divyanshu Mishra. *Convolution Vs Correlation*. 2019. URL: https://towardsdatascience.com/convolution-vs-correlation-af868b6b4fb5.

[22] Suhyun Kim. *A Beginner's Guide to Convolutional Neural Networks (CNNs)*. 2019. URL: https://towardsdatascience.com/a-beginners-guide-to-convolutional-neural-networks-cnns-14649dbddce8.

[23] Thom Lane. *Multi-Channel Convolutions explained with... MS Excel!* 2018. URL: https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108.

[24] Aqeel Anwar. *What is Transposed Convolutional Layer?* 2020. URL: https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11.

[25] Christopher Olah. *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[26] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[27] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. URL: https://ruder.io/optimizing-gradient-descent/.

[28] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[29] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Instance normalization: The missing ingredient for fast stylization". In: *arXiv preprint arXiv:1607.08022* (2016).

[30] Kaiming He et al. “Deep residual learning for image recognition”. In: (2016), pp. 770–778.
[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 14.
[32] Elad Plaut. *From Principal Subspaces to Principal Components with Linear Autoencoders*. 2018. arXiv: 1804.10253 [stat.ML].
[33] Jeremy Jordan. *Introduction to autoencoders*. 2018. URL: https://www.jeremyjordan.me/autoencoder.
[34] Li Wan et al. *Generalized End-to-End Loss for Speaker Verification*. 2017. arXiv: 1710.10467 [eess.AS].
[35] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: (2010), pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.
[36] Google. *Googles Machine Learning course on GANs*. Portions of this page are modifications from work created and shared by Google and used according to terms described in the Creative Commons 4.0 Attribution License. URL: https://developers.google.com/machine-learning/gan.
[37] Jie Feng et al. “Generative Adversarial Networks Based on Collaborative Learning and Attention Mechanism for Hyperspectral Image Classification”. In: *Remote Sensing* 12.7 (Apr. 2020), p. 1149. ISSN: 2072-4292. DOI: 10.3390/rs12071149. URL: http://dx.doi.org/10.3390/rs12071149.
[38] Takuhiro Kaneko and Hirokazu Kameoka. “Parallel-data-free voice conversion using cycle-consistent adversarial networks”. In: *arXiv preprint arXiv:1711.11293* (2017).
[39] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *CoRR* abs/1611.07004 (2016). arXiv: 1611.07004. URL: http://arxiv.org/abs/1611.07004.
[40] Kanglin Liu and Guoping Qiu. “Lipschitz constrained GANs via boundedness and continuity”. In: *Neural Computing and Applications* (May 2020). DOI: 10.1007/s00521-020-04954-z.
[41] Jason Brownlee. *A Gentle Introduction to Exploding Gradients in Neural Networks*. Aug. 2019. URL: https://machinelearningmastery.com/exploding-gradients-in-neural-networks/.
[42] Maciej Wiatrak and Stefano V. Albrecht. “Stabilizing Generative Adversarial Network Training: A Survey”. In: *ArXiv* abs/1910.00927 (2019).
[43] David Foster. *Generative Deep Learning*. URL: https://www.oreilly.com/library/view/generative-deep-learning/9781492041931/ch04.html.
[44] Aaron van den Oord et al. *WaveNet: A Generative Model for Raw Audio*. 2016. arXiv: 1609.03499 [cs.SD]. URL: https://arxiv.org/pdf/1609.03499.pdf.
[45] Aaron van den Oord et al. *Parallel WaveNet: Fast High-Fidelity Speech Synthesis*. 2017. arXiv: 1711.10433 [cs.LG]. URL: https://arxiv.org/pdf/1711.10433.pdf.
[46] Nal Kalchbrenner et al. “Efficient neural audio synthesis”. In: *arXiv preprint arXiv:1802.08435* (2018).

[47] Corentin Jemine. “Master thesis : Automatic Multispeaker Voice Cloning”. In: (2019).

[48] Robert C Streijl, Stefan Winkler, and David S Hands. “Mean opinion score (MOS) revisited: methods and applications, limitations and alternatives”. In: *Multimedia Systems* 22.2 (2016), pp. 213–227.

A Appendix

A.1 Code

The code used for this paper can be found in this GitHub repository[1]. This should only be used for studies and should not in any way be used for harmful or unethical uses.

A.2 Survey

Survey Frontpage:

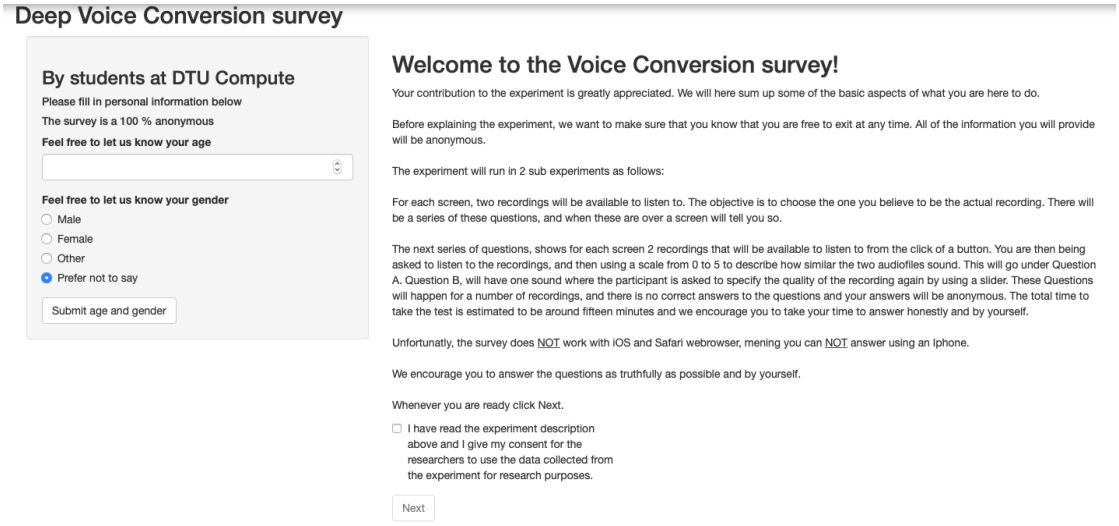

Figure A.1: Frontpage of the survey containing the information sheet

Survey Fool test:

[1] https://github.com/Fagprojekt-Deep-voice-conversion/Deep_voice_conversion

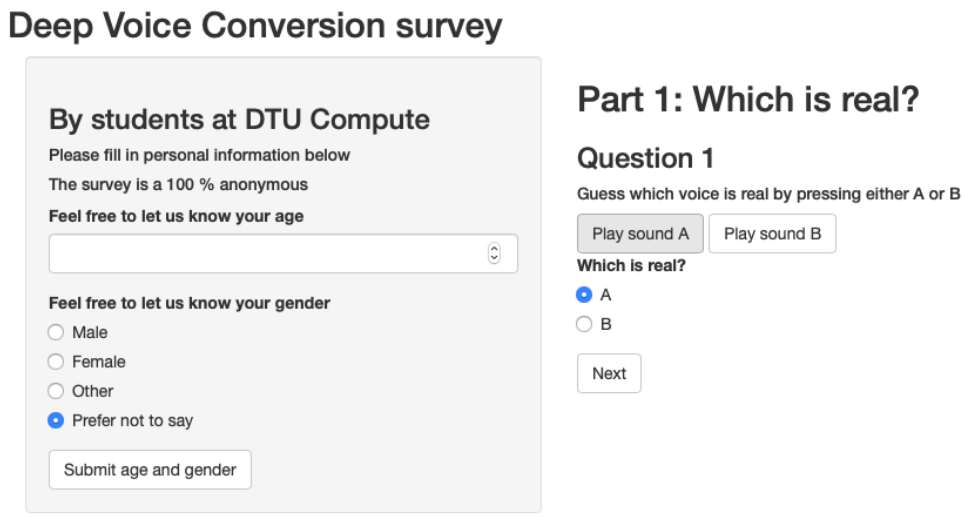

Figure A.2: First question of the survey to test for fooling rates

Survey Similarity/Quality test:

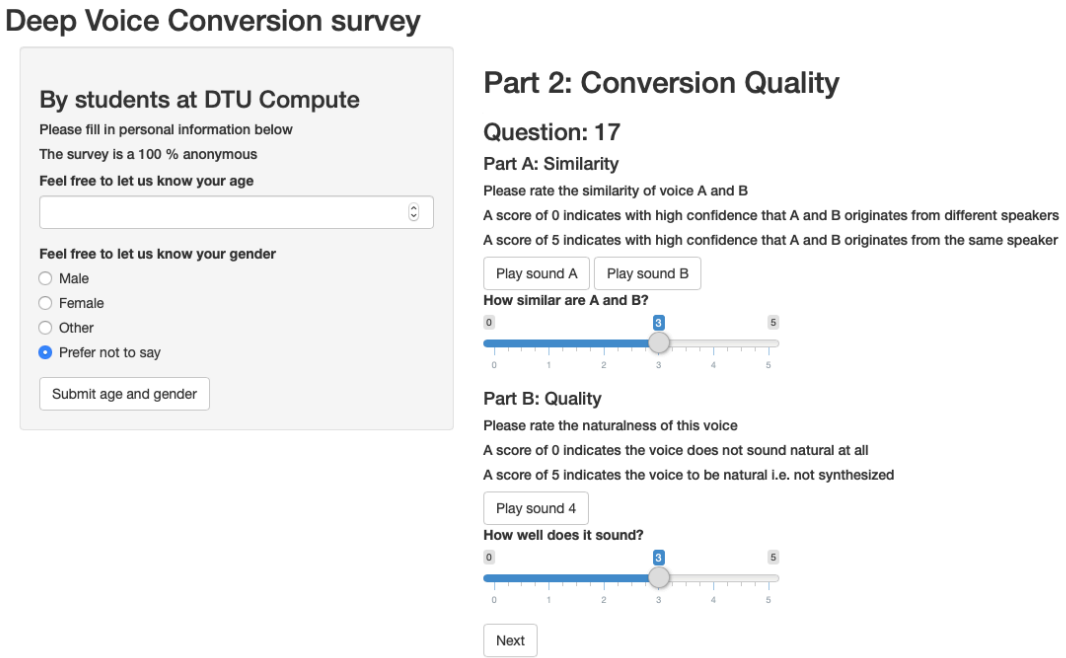

Figure A.3: Second experiment of the survey to test for similarity and naturalness of the sound

A.3 Shapiro Wilks test

Experiment	Model	P-value
10 min	AutoVC	1.97e-04
10 min	StarGAN	7.46e-07
20 min	AutoVC	3.29e-04
20 min	StarGAN	1.42e-06
30 min	AutoVC	3.23e-05
30 min	StarGAN	1.94e-09
DFF	AutoVC	6.73e-05
DFF	StarGAN	1.05e-05
DFM	AutoVC	6.26e-06
DFM	StarGAN	3.83e-10
DMF	AutoVC	4.00e-06
DMF	StarGAN	1.54e-09
DMM	AutoVC	3.23e-05
DMM	StarGAN	1.94e-09
EFF	AutoVC	1.06e-05
EFF	StarGAN	2.11e-04
EFM	AutoVC	2.67e-07
EFM	StarGAN	1.03e-13
EMF	AutoVC	2.07e-04
EMF	StarGAN	6.71e-11
EMM	AutoVC	7.06e-05
EMM	StarGAN	1.24e-12
FD	WaveRNN Baseline	1.56e-12
FD	World Baseline	3.87e-11
FE	WaveRNN Baseline	2.33e-08
FE	World Baseline	5.68e-08
MD	WaveRNN Baseline	2.87e-09
MD	World Baseline	3.09e-11
ME	WaveRNN Baseline	1.08e-11
ME	World Baseline	6.13e-07

Table A.1: The p-values for the similarity experiment when performing a Shapiro Wilks test

Experiment	Model	P-value
10 min	AutoVC	3.53e-05
10 min	StarGAN	1.63e-04
20 min	AutoVC	7.02e-04
20 min	StarGAN	5.25e-05
30 min	AutoVC	7.32e-04
30 min	StarGAN	2.65e-04
DFF	AutoVC	3.24e-04
DFF	StarGAN	5.92e-05
DFM	AutoVC	1.03e-04
DFM	StarGAN	1.88e-05
DMF	AutoVC	6.1e-06
DMF	StarGAN	1.3e-06
DMM	AutoVC	7.32e-04
DMM	StarGAN	2.65e-04
EFF	AutoVC	6.70e-06
EFF	StarGAN	8.44e-05
EFM	AutoVC	1.63e-04
EFM	StarGAN	5.43e-06
EMF	AutoVC	6.16e-06
EMF	StarGAN	7.94e-06
EMM	AutoVC	7.76e-04
EMM	StarGAN	5.93e-08
FD	WaveRNN Baseline	9.39e-12
FD	World Baseline	2.12e-08
FE	WaveRNN Baseline	2.65e-04
FE	World Baseline	7.20e-06
MD	WaveRNN Baseline	1.04e-08
MD	World Baseline	3.03e-07
ME	WaveRNN Baseline	6.02e-10
ME	World Baseline	2.20e-06

Table A.2: The p-values for the quality experiment when performing a Shapiros test

Experiment	Model	P-value
DFF	AutoVC	4.66e-16
DFF	StarGAN	1.87e-16
DFM	AutoVC	5.92e-18
DFM	StarGAN	2.38e-15
DMF	AutoVC	6.89e-17
DMF	StarGAN	5.92e-18
DMM	AutoVC	4.99e-15
DMM	StarGAN	2.26e-17
EFF	AutoVC	1.89e-13
EFF	StarGAN	9.99e-15
EFM	AutoVC	1.87e-16
EFM	StarGAN	1.87e-16
EMF	AutoVC	5.95e-12
EMF	StarGAN	1.93e-14
EMM	AutoVC	4.99e-15
EMM	StarGAN	2.26e-17

Table A.3: The p-values for the fool test experiment when performing a Shapiros test

A.4 Plot data

Conversion type	Model	Mean	Lower CI	Upper CI
10 min	AutoVC	2.06	1.72	2.45
10 min	StarGAN	1.31	0.98	1.63
20 min	AutoVC	2.75	2.40	3.11
20 min	StarGAN	1.43	1.11	1.80
30 min	AutoVC	2.98	2.57	3.37
30 min	StarGAN	0.91	0.60	1.22
DF	WaveRNN Baseline	4.48	4.20	4.71
DF	World Baseline	4.28	3.97	4.57
DFF	AutoVC	2.52	2.11	2.97
DFF	StarGAN	1.29	1.02	1.55
DFM	AutoVC	1.71	1.34	2.11
DFM	StarGAN	0.71	0.49	0.95
DM	WaveRNN Baseline	4.08	3.77	4.35
DM	World Baseline	4.37	4.11	4.60
DMF	AutoVC	1.63	1.28	1.98
DMF	StarGAN	0.65	0.46	0.85
DMM	AutoVC	2.98	2.57	3.40
DMM	StarGAN	0.91	0.60	1.22
EF	WaveRNN Baseline	4.03	3.72	4.31
EF	World Baseline	3.83	3.52	4.12
EFF	AutoVC	3.31	2.92	3.69
EFF	StarGAN	1.80	1.46	2.14
EFM	AutoVC	1.22	0.89	1.55
EFM	StarGAN	0.37	0.18	0.58
EM	WaveRNN Baseline	4.43	4.17	4.66
EM	World Baseline	3.71	3.35	4.02
EMF	AutoVC	2.28	1.89	2.68
EMF	StarGAN	0.52	0.34	0.74
EMM	AutoVC	2.49	2.09	2.88
EMM	StarGAN	0.48	0.25	0.74
All	AutoVC	2.27	2.12	2.42
All	StarGAN	0.84	0.74	0.93
All	WaveRNN Baseline	4.25	4.10	4.39
All	World Baseline	4.05	3.88	4.20

Table A.4: The data used for figure 7.1

Conversion type	Model	Mean	Lower CI	Upper CI
10 min	AutoVC	1.80	1.54	2.09
10 min	StarGAN	1.89	1.55	2.20
20 min	AutoVC	2.65	2.38	2.94
20 min	StarGAN	1.49	1.22	1.80
30 min	AutoVC	2.89	2.63	3.14
30 min	StarGAN	2.12	1.77	2.49
DF	WaveRNN Baseline	4.35	4.06	4.62
DF	World Baseline	4.18	3.98	4.40
DFF	AutoVC	3.15	2.83	3.48
DFF	StarGAN	1.65	1.37	1.94
DFM	AutoVC	1.63	1.31	1.92
DFM	StarGAN	1.52	1.22	1.88
DM	WaveRNN Baseline	4.22	3.97	4.45
DM	World Baseline	3.92	3.65	4.17
DMF	AutoVC	1.74	1.42	2.08
DMF	StarGAN	1.15	0.91	1.45
DMM	AutoVC	2.89	2.60	3.20
DMM	StarGAN	2.12	1.78	2.49
EF	WaveRNN Baseline	2.62	2.25	2.98
EF	World Baseline	3.78	3.54	4.02
EFF	AutoVC	1.65	1.34	1.92
EFF	StarGAN	1.92	1.57	2.28
EFM	AutoVC	1.71	1.42	2.00
EFM	StarGAN	1.49	1.17	1.88
EM	WaveRNN Baseline	4.37	4.14	4.58
EM	World Baseline	3.43	3.08	3.80
EMF	AutoVC	1.32	1.08	1.60
EMF	StarGAN	1.46	1.20	1.77
EMM	AutoVC	3.03	2.74	3.31
EMM	StarGAN	1.31	0.97	1.71
All	AutoVC	2.14	2.02	2.26
All	StarGAN	1.58	1.46	1.70
All	WaveRNN Baseline	3.89	3.72	4.04
All	World Baseline	3.83	3.68	3.98

Table A.5: The data used for figure 7.2

Conversion type	Model	Answers	People fooled	Percentage fooled	Lower CI	Upper CI
DFF	AutoVC	65	5	0.08	0.03	0.16
DFF	StarGAN	65	4	0.06	0.02	0.14
DFM	AutoVC	65	1	0.02	0.00	0.07
DFM	StarGAN	65	7	0.11	0.05	0.20
DMF	AutoVC	65	3	0.05	0.01	0.12
DMF	StarGAN	65	1	0.02	0.00	0.07
DMM	AutoVC	65	8	0.12	0.06	0.22
DMM	StarGAN	65	2	0.03	0.01	0.10
EFF	AutoVC	65	14	0.22	0.13	0.33
EFF	StarGAN	65	9	0.14	0.07	0.24
EFM	AutoVC	65	4	0.06	0.02	0.14
EFM	StarGAN	65	4	0.06	0.02	0.14
EMF	AutoVC	65	23	0.35	0.25	0.47
EMF	StarGAN	65	10	0.15	0.08	0.26
EMM	AutoVC	65	8	0.12	0.06	0.22
EMM	StarGAN	65	2	0.03	0.01	0.10
All	AutoVC	585	74	0.13	0.10	0.16
All	StarGAN	585	41	0.07	0.05	0.09

Table A.6: The data used for the figure 7.4

Acronyms

AE auto encoder. 32

AI artificial intelligence. 5

ANN artificial neural network. 21, 24–26, 28, 29, 31, 46

BiLSTM bidirectional long short term memory. 26, 36

BN batch normalisation. 28, 29, 36

CEO cheif executive officer. 3

CNN convolutional neural network. 21, 22, 36

COMPAS correctional offender management profiling for alternative Sanctions. 3

DCT discrete cosine transformation. 19

DFT discrete Fourier transformation. 12–14, 18, 19, 49

FT Fourier transformation. 11

GAN generative adversarial network. 2, 37, 38, 40, 41, 67

GDPR general data protection regulation. 3

GE2E generalised end to end. 33, 34

GRU gated recurrent unit. 27, 47, 48

IN instance normalisation. 29

LSTM long short term memory. 21, 25–27, 36, 47

MFCC mel frequency cepstrum coefficients. 19–21, 39, 52

MOS mean opinion score. 58, 60, 61

PCA principal component analysis. 32

RNN recurrent neural network. 21, 24–26, 47

SGD stochastic gradient descend. 27–29

STFT short time Fourier transformation. 12–14, 16–18, 50

VC voice conversion. 68

VCTK Voice Cloning Toolkit. 7, 8, 52, 54, 66, 67

Glossary

backpropagation is a method used to update the weights in a neural network. First a forward pass is done to get the values of each node from the specific input and then a backwards pass is used to go backwards through the network and update each weight via gradient descent, which takes the derivates of the loss function wrt. each parameter to find a local minimum. 38

Deontology The ethical thinking that the morality of an action should be based on whether the action is right or wrong instead of the consequences of said action. 3

false negative A false negative is gained by classifying something positive as negative. 38

false positive A false positive is gained by classifying something negative as positive. 38

L2-norm The L2-norm of a vector $\mathbf{x}$ is also known as the euclidean distance and is simply the square root of the sum of squared elements of $\mathbf{x}$: $||\mathbf{x}||_2 = \sqrt{|x_1|^2 + |x_2|^2 ...}$. 44

softmax function The softmax function takes a vector $\mathbf{x}$ as input and normalises it to follow a probability distribution, meaning the relative difference between the values are kept and the vector sums to 1. The probability that the input belongs to a class c is given by $P(c|\mathbf{x}) = \frac{\exp \mathbf{x}_c}{\sum_C \exp \mathbf{x}_C}$ where x_c is the element of $\mathbf{x}$ corresponding to class c. 41