Universiteit Leiden

The Voice Within the Neural Network

Name: Yenebeb Schwarze

Date: 16/08/2021

1st Supervisor: Dr. E.M. Bakker
2nd Supervisor: Prof.dr. M.S.K. Lew

Bachelor Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Cloning voices is a task Neural Networks still struggle with. One of the most difficult parts of cloning voices is creating a meaningful embedding of a voice. There have been attempts that look at voice verification as a tool to create the embeddings. In this thesis we will introduce RTVC+, two models built upon the Real Time Voice Cloning model, that use AutoVC [15] and SpeechSplit [16] to create the embeddings. We research the impact that these methods have on the quality of the generated embedding and compare the results with the original. Because both of our introduced models are dependent on well-tuned bottlenecks in the autoencoders, we also introduce a new method to tune the bottlenecks. This method makes use of the fact that embeddings generated by the speaker encoder should have a division between the male and female speakers. Our research shows that there is a correlation between the embedding distances of the male and female speakers and the quality of the bottleneck settings on the used autoencoders.

Contents

1 Introduction

As early as 1000AD there were talks about Brazen Heads. Brazen Heads were automatons that could speak and answer any questions. Although it is most unlikely that they have ever existed it does tell us that interest in creating human-like voices is not something recent. In 1779 Christian Kratzenstein created a model of the human vocal tract to produce the sound of vowels. Wolfgang von Kempelen improved this machine by adding models of the tongue and lips which enabled the machine to produce consonants as well. The first vocoder was created by Homer Dudley at Bell Labs in the 1930s. This was the first machine that could translate human speech into electrical signals and then back to speech again. This was done in two stages. The encoder part created the signals from the input audio and the decoder part re-created the audio from these signals. By only using the decoder Homer Dudley was able to create a voice-synthesizer called the Voice Demonstrator. The same company, Bell Labs, also produced the first spectrograph [17]. A machine that converts speech into a visible image called a spectrogram. Using the spectrogram Dr Franklin S. Cooper built the Pattern playback. Which was a machine that could convert these pictures back into sound.

In the 1950s the first computer-based speech synthesis systems were created. The first English general computer-based text-to-speech system was made by Noriko Umedat et al in 1968. Research took off with these computer-based speech synthesis methods. But it is only in the past 10 years that neural network-based speech synthesis has taken over the field, achieving better results on voice quality while simultaneously decreasing the amount of human work needed for the pre-processing of the data and feature development [21]. But, even with deep neural networks, there is still a lot of room for improvement. Both on the side of the quality of the voice as well as the efficiency of the synthesizer. [21]

As can be seen by the huge amount of research poured into creating human-like voices, synthesizing voices is difficult. This is mainly because of the dynamic nature of human voices. They change through time, are different for every word and even between the same word they can be pronounced and thus sound different. While they are ever-changing, they do have a similarity. Spoken words, given that they are in the same language, can be understood no matter which person is speaking them. Similarly voices, given that they are from the same person, can be recognized no matter what they are saying.

This has been expressed in two main goals of the speech synthesis model: intelligibility and naturalness [19]. The model should be able to generate an audio output that can be understood, meaning that the interpretation of the text input and the audio output is the same. This is measured with intelligibility. In other words: can the listener understand the audio output generated by the model? Naturalness refers to all the information not included in intelligibility, such as the dialect. For speech cloning, there is a need to have the naturalness of the input voice be the same as the output voice. We will refer to this as the clonability rate. Higher clonability rates will indicate a higher similarity in naturalness. A model that has high intelligibility and clonability rate might be able to perfectly clone the trained data on any text, but still fail at cloning speakers which were not in the training data. Generalization is therefore an important part of speech cloning. The task of speech cloning can be considered a subtask of speech synthesis. Speech cloning inherits the main goals of speech synthesis other than these clonability and generalizability can be

considered the third and fourth goals of speech cloning.

Our goal throughout this thesis is to improve the clonability rate of a state-of-the-art text-to-speech model [10]. Our implementation of this method makes use of both SpeechSplit [14] as AutoVC [13] to improve the embeddings that are generated.

In section 2 we will be touching upon several of the other models that already exist. Following that, section 3 will explain the fundamental terminology used in our project. Section 4 will contain all information about the base model on which everything is built upon. Our problem statement will be discussed in Section 5 and we will introduce our two new models in Section 6. The setup of our different experiments can be seen in Section 7 with the results in section 8. We conclude our findings in Section 9.

2 Related Work

In the past ten years, progress has been made in deep neural networks for text to speech synthesis. We will discuss several of the state-of-the-art models here and their origins.

In 2017 we got introduced to Deep Voice [2] and Tacotron [25]. These were several of the first papers to implement an end-to-end model for text-to-speech synthesis. Baidu's Deep Voice 1 [2] is a neural network implementation of the traditional text-to-speech pipeline. They replaced each aspect of the traditional pipeline with neural networks while keeping the same structure. This resulted in a model that not only needs less manual work in data annotation but also improved the audio synthesizing part using a modified WaveNet [22] model. At the time of publication of Deep Voice, WaveNet was considered state of the art. At the time of writing this paper, Parallel WaveNet [23] is considered state of the art. Parallel WaveNet has a 1000x speedup compared to the original WaveNet [23]. Unless explicitly mentioned WaveNet will refer to the original non-parallelized WaveNet.

Tacotron [25] and later Tacotron 2 [18] were both from Google's team and implemented their version using a sequence-to-sequence model. They further divided their model into multiple modules with each having a separate task. Tacotron 2 uses a modified WaveNet architecture conditioned on spectrograms instead of the Griffin-Lim algorithm [6]. They were able to significantly improve the MOS score of their model, from 4.001 ± 0.087 on Griffin-Lin to 4.526 ± 0.066 on the modified WaveNet [18]. Both Baidu and Google use different models to create a voice able to speak words and sentences fluently. While these networks achieve high scores on naturalness, as can be seen in Tale 1, they are only trainable on one voice at a time.

An improvement on these two networks came in the form of a multi-speaker adaptation of both Baidu's Deep Voice 2 as Tacotron 2 [1] and Baidu's Deep Voice 3 [12]. All three of the networks use an extra speaker embedding as input for the model. These embeddings are important because they enable the model to generalize on the training data. Using the embeddings, the models can generally clone voices that were learned during training. In Table 2 MOS scores are given for the different models. The number of voices that are cloneable, as well as the time it takes to create the voice, differs per model, but because the systems learn a fixed set of speaker embeddings, they do not support synthesis of voices that are unseen during training.

Facebook's Voiceloop [20] was the first network that was able to fit new speakers

Model	Mean Opinion Score (MOS)
Deep Voice 3 (Griffin-Lim)	3.62 ± 0.31
Deep Voice 3 (WORLD)	3.63 ± 0.27
Deep Voice 3 (WaveNet)	3.78 ± 0.30
Tacotron (WaveNet)	3.78 ± 0.34
Deep Voice 2 (WaveNet)	2.74 ± 0.35

Source: Table 2 [12]

Table 1: MOS on Single Speaker dataset

Model	MOS (VCTK)	MOS (LibriSpeech)
Deep Voice 3 (Griffin-Lim)	3.01 ± 0.29	2.37 ± 0.24
Deep Voice 3 (WORLD)	3.44 ± 0.32	2.89 ± 0.38
Deep Voice 2 (WaveNet)	3.69 ± 0.23	-
Tacotron (Griffin-Lim)	2.07 ± 0.31	-
Ground truth	4.69 ± 0.04	4.51 ± 0.18

Source: Table 3 [12]

Table 2: MOS on multi speaker dataset. Tacotron refers to the modified Tacotron model in [1]

after training. Voiceloop implemented a loop that acted as a short-term memory [20]; this not only made the model highly flexible, but it also allowed the use of audio samples with a lot of noise. Because of the high flexibility, Voiceloop can get a high accuracy, even on unseen data by simply retraining the network. The retrained network achieves a MOS score of 3.08 ± 0.95 compared with the MOS score on voices seen during training being 3.57 ± 1.08 [20]. This however still needs more than 10 minutes of data from a single speaker. "While TTS systems typically require several hours of data to model a single speaker, our fitting set contains only 23.65 minutes per speaker on average." (Taigman et al. [20])

Speaker Verification to Text-To-Speech (SV2TTS) [10] introduces a model that can generate voices of speakers which are not seen during training without needing any retraining. Furthermore, it can do this in a zero-shot learning setting. This network is combined out of three parts: an encoder, a synthesizer based on Tacotron, and a vocoder. The separation of the different models, their functions, and their ability to be separately trained made this model ideal to be used as a base model in this project. A similar model is used in [5] which describes the effect of using different speaker embeddings on zero-shot learning. This paper focuses more on the effect of AutoVC [15] and SpeechSplit [16], which were not investigated in [5].

We do want to mention Glow-TTS here [3]. Just like SV2TTS, Glow-TTS also has zero-shot learning capabilities on unseen speakers. It even claims to be state-of-the-art in

that aspect. Unfortunately, the publish date of the paper was during the experimentation stage of this thesis and the method has not been considered as a candidate.

3 Fundamentals

This section will be used as a referral to the terminology used in the paper.

3.0.1 Grapheme and Phonemes

The original SV2TTS model makes use of phoneme and grapheme sequences as input for the synthesizer. Phonemes are the different sounds letters or combinations of letters can make. Each sound is represented by a different symbol. graphemes are the same sounds written by letters. One phoneme can have multiple graphemes. For instance, the c in car, the graphemes are: k, ck, q, ch while the phoneme is $/c/$.

3.0.2 Spectrograms

A spectrogram is audio in a picture. It is based on the fact that sound can be split into multiple sine waves with different amplitudes and frequencies. These can be plotted in an image, where the vertical axis is the frequency, and the brightness is the amplitude. The horizontal axis is time.

3.0.3 AutoEncoders

AutoEncoders are great for finding different representations of data. They are made of two parts, an encoder, and a decoder. The job of the encoder is to reduce the dimensionality of the input data to some given bottleneck, while the job of the decoder is to recreate the input using the output of the encoder. This in turn forces the encoder to output the most important parts of the input data, since that is the only way that the decoder can recreate the data.

3.0.4 One-Hot Embedding

A one-hot encoding is an embedding with only 1's and 0's. It represents the different categories on the rows and the different data on the columns. When an index is 1 it means that the data represents said category.

3.0.5 Attention Decoding

In most of the TTS models an attention decoder is used. Different from the normal decoders, an attention decoder can select elements from a sequence of encoder inputs which include past outputs of the encoder. This way the decoder learns to align the inputs with the output. This is especially great for text-to-speech, since there is a need for alignment on the words spoken.

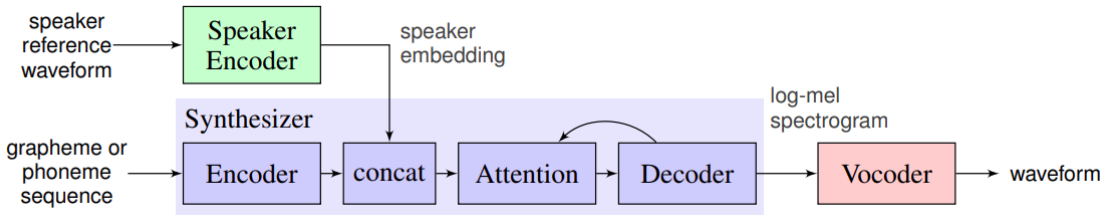
3.0.6 Zero-Shot Learning

The goal of most machine learning models is to classify the training data in certain classes. Once trained, the model can classify data it has not seen in these same classes. The model is therefore limited by the classes it has created during training. Zero-Shot-Learning is a method that enables the classification of unseen categories in the training data during run-time.

3.0.7 MOS

The mean opinion score (MOS) is the most widely used metric in comparing different TTS models. To date, there has been no method to objectively measure speaker quality. The MOS is therefore calculated by using the public to rate created audio fragments on a scale from 1 to 5. 5 being excellent and 1 being poor. On SV2TTS [10] the MOS is divided into speaker quality and speaker similarity.

4 Baseline



Source: SV2TTS Paper

Figure 1: Original model from SV2TTS

The baseline used in this thesis is from [10]. The authors of this paper use Tacotron with some modifications to implement an architecture that can synthesize voices based on a short five second reference voice. It was implemented by J. Corentin in his master thesis [7] using Tensorflow. Later, the repository was updated to run with PyTorch and using Tacotron 1. Because both AutoVC [15], as well as SpeechSplit [16], were implemented with PyTorch and the downsides to using Tacotron 1 were within reason. The following observations were made by the creator of the commit:

"Tacotron 1 learns faster than Tacotron 2. Training time per step is about the same, but the model is usable in fewer steps. Voice quality is similar. Attention is more robust in Taco2." ([9] bluefish, creator of the commit). We have chosen to use the PyTorch implementation as the baseline of our paper.

In the following chapters, we will explain the architecture and how the base model is trained.

4.1 Base Architecture Model

Because the difference in the PyTorch and TensorFlow implementation is mainly in the use of the Tacotron model the overall model as seen in Figure 1 is not impacted.

The base model, as seen in Figure 1, is separated into three main modules:

1. Encoder
2. Synthesizer
3. Vocoder

The encoder has the job of creating an embedding from the audio fragment. It is important that the speaker encoder can capture and identify characteristics of voices even if the referenced waveform is short and noisy. The base model makes use of Generalized End-to-End (GE2E) loss [24] to ensure that these characteristics are different for each speaker while remaining independent of the words spoken.

The original synthesizer from [10] was based on Tacotron 2. As can be seen in Figure 2 the differences between Tacotron 1 and 2 are mainly in the use of the Long Short-Term Memory (LSTM) and Convolutional layers instead of the CHBG [25] blocks and Gated Recurrent Unit (GRU) [4] layers. These simplify the Tacotron 2 model. Another big difference is in the vocoder part but since we use our own vocoder this does not apply to our baseline model.

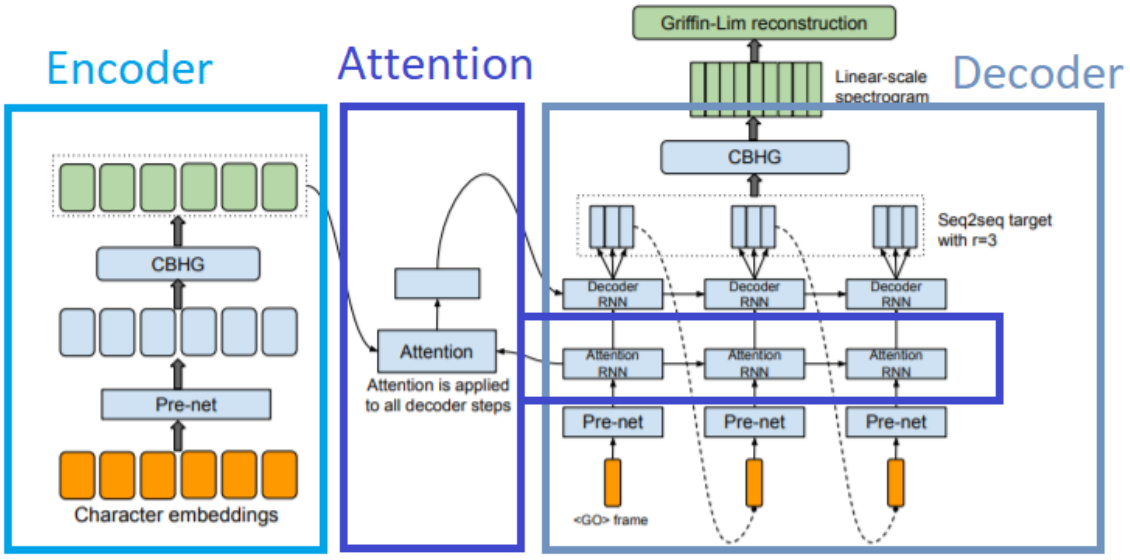
The synthesizer is built on three blocks. Just like Tacotron, it has an encoder, to encode the grapheme or phoneme sequences and an attention-decoder. In the original Tacotron model [25], this would be enough since the voice to be learned remained constant. But the attention-decoder will need a representation of the voice to enable the synthesizer to output different spectrograms depending on the input voice. To fix this our base model has been expanded with an extra concatenate function. The output of the synthesizer encoder, which is the text-encoder, and the output of the speaker encoder are concatenated before going through the decoder.

The vocoder only has one job: turning the spectrogram from the synthesizer into a waveform. WaveRNN [11] is used for the vocoder. WaveRNN is a simplified WaveNet model which uses Recurrent Neural Networks to replace the 60 Convolutions in WaveNet [11]. It is a lot faster than WaveNet achieving $4x$ real-time inference versus $0.3x$ real-time inference on WaveNet [11]. The MOS score achieved by the best WaveRNN is 4.48 ± 0.07 which is comparable with the WaveNet MOS of 4.51 ± 0.08 [11].

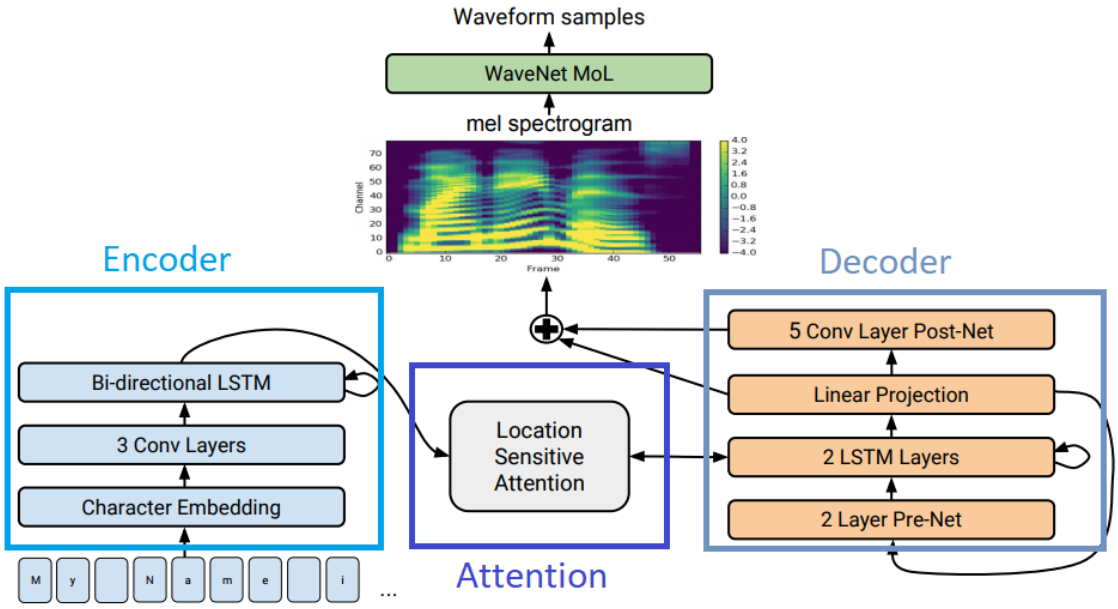
4.2 Training

The three components in SV2TTS can be, (and are) trained separately from each other. As we mentioned the speaker encoder uses the GE2E loss model, which enables it to train on its own with only a dataset with audio of different speakers.

Once the speaker encoder is trained the synthesizer can be trained. Because the synthesizer needs the output of the speaker encoder, this cannot be done simultaneously. While training the synthesizer the parameters and weights of the speaker encoder are frozen, the output of the speaker encoder is then used to condition the synthesizer. As



(a) Tacotron 1

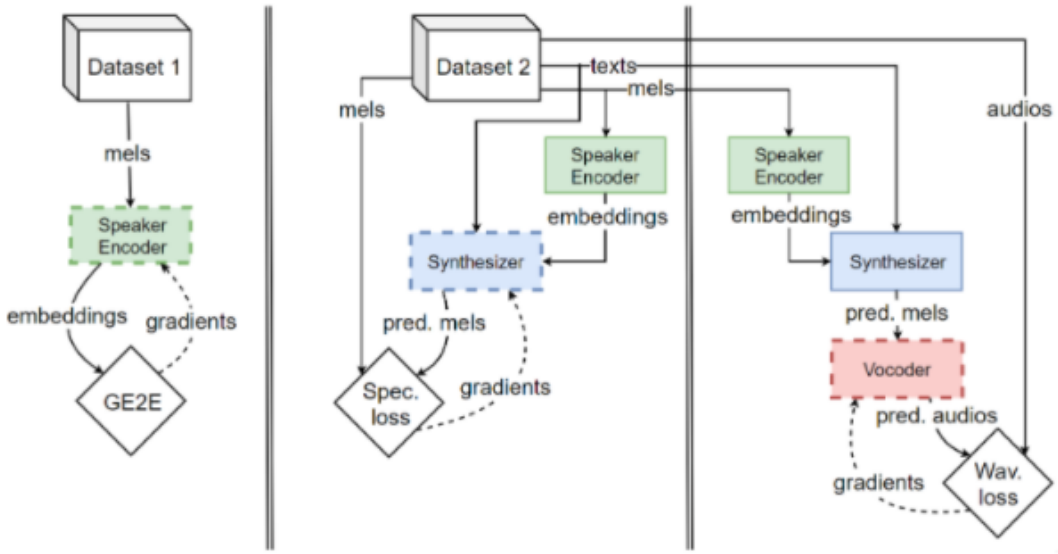


(b) Tacotron 2

Figure 2: Differences Tacotron 1 and Tacotron 2

can be seen in Figure 3, the gradients obtained from the loss are only applied on the synthesizer.

The vocoder is trained on the mel-spectrograms obtained by the synthesiser. Just like the speaker encoder, the vocoder can also be trained without the other modules. Although training it with the direct output of the synthesiser should decrease the noise



Source: [7]

Figure 3: Training of components SV2TTS

generated. Each of the blocks in our base model can thus be trained separately.

5 Problem Statement

The SV2TTS model has the advantages of being able to function in a zero-shot learning environment [10] and is especially useful due to the separated speaker encoder, synthesizer, and vocoder. It does however have its demerits: "The proposed model does not attain human-level naturalness, despite the use of a WaveNet vocoder" (Jia et al. [10])

SE Training Set	Speakers	Embedding Dim	Naturalness	Similarity	SV-EER
LS-Clean	1.2K	64	3.73 ± 0.06	2.23 ± 0.08	16.60%
LS-Other	1.2K	64	3.60 ± 0.06	2.27 ± 0.09	15.32%
LS-Other + VC	2.4K	256	3.83 ± 0.06	2.43 ± 0.09	11.95%
LS-Other + VC + VC2	8.4K	256	3.82 ± 0.06	2.54 ± 0.09	10.14%
Internal	18K	256	4.12 ± 0.05	3.03 ± 0.09	5.08%

Source: Table 5, [10]

Figure 4: Results of SV2TTS model with speaker encoder trained on different datasets

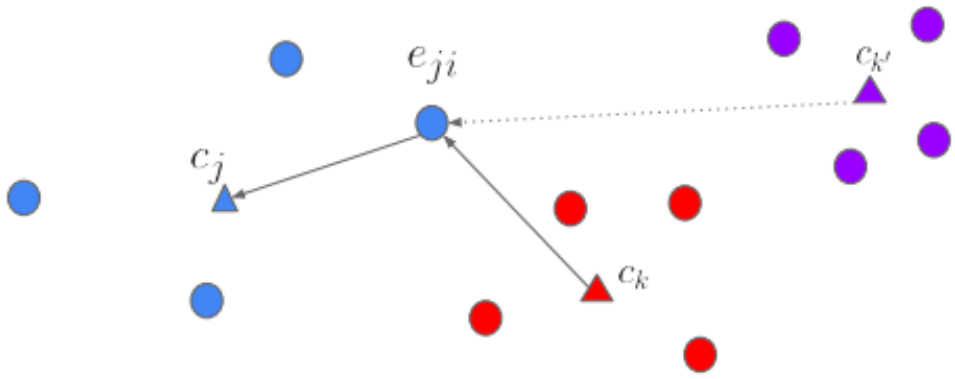
In Figure 4 the relation between the number of speakers used for the Speaker Encoder and the performance of the overall data can be seen. "It is likely that the ability of the proposed model to generalize well across a wide variety of speakers is based on the quality of the representation learned by the speaker encoder." ([10])

If generalization is dependent on the representation learned by the speaker encoder there is a need to investigate the speaker encoder and consider the use of other encoders. Our problem statement thus becomes: "is there a way to improve the results of the base model by using a different embedding".

5.1 GE2E

There seems to be a high correlation between the quality of the embeddings generated by the vocoder and the quality of the voice output by the synthesizer.

Training the vocoder on more voices creates better embeddings, but there should be a limit. The way the embedding is made, using the GE2E loss, might also limit the quality of the embedding.



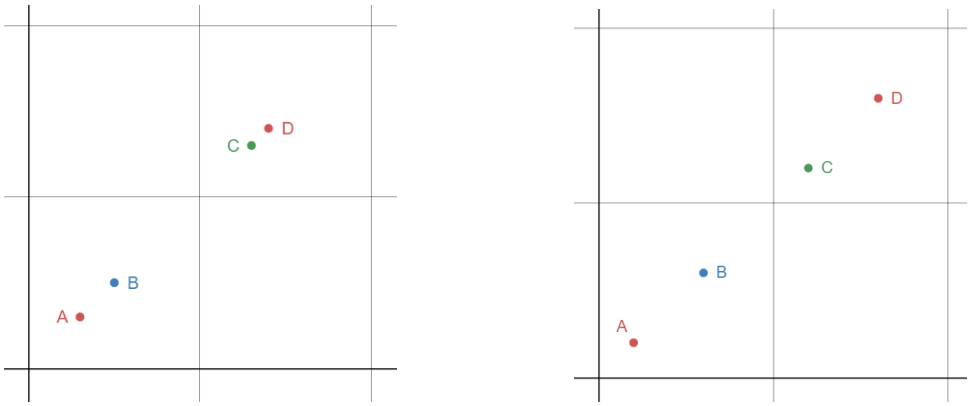
Source: Generalized E2E loss Paper [24]

Figure 5: Embeddings GE2E Loss

The encoder, based on the GE2E model, learns to push embeddings from different voices away from each other while pushing embeddings from the same speaker to each other, which we can see clearly in Figure 5. This means that the embeddings are optimized to represent a speaker. This is great for speaker verification but might not be an optimal solution for creating a representation that transmits information for speech cloning. In the most optimal case, every audio fragment from the same speaker will generate the same embedding.

The encoder learns to divide speakers based on some characteristic in the audio. We don't know what that characteristic is, and there are likely multiple characteristics in the real case. Let us imagine this being the frequency of the audio. If the main characteristic on which the embeddings are divided is frequency, then, because of the GE2E loss, embeddings from different speakers will be pushed away from each other meanwhile embeddings from the same speakers will be pushed towards each other. This means that, since we recognize our speakers through their frequency, the embedding of

different speakers with similar frequencies will be pushed away from each other, while embeddings of different speakers with highly different frequencies will stay the same.



(a) Real Distance (b) Embedding Distance

Figure 6

For two characteristics x and y the real distance might be highly different from the generated distance by GE2E. Take 4 voices: A, B, C and D. A & B, and C & D might have characteristics x and y which start out relatively close to each other like in Figure 6a. But because of the way GE2E works, they will be pushed away from each other. So, while voices A and B might have a lot more similarities compared to voices B and C, the distances in the embedding between A & B and B & C is quite the same.

This in turn means that, while the embedding generated by the encoder contains some characteristic of the audio, there is no clear relation between the distances of the embeddings and the similarity of the audio. The embedding is more like the name of the speaker, an ID. This ID is then projected to the synthesizer. Since the embeddings are similar on audio fragments of the same speaker, even as the text embedding changes, the decoder learns to output a spectrogram that resembles the original spectrogram of the speaker based on the embedding. As we've learned there is no clear information to decode from the speaker embedding and thus the decoder is likelier to "hardcode" different embeddings on different ways of making the spectrogram.

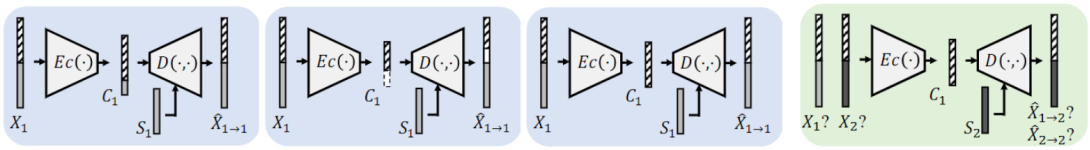
6 Real Time Voice Cloning+

From section 5 we found that there is a lack of information in the embedding. The question then remains: "which information is lacking and how can we add this information in the embedding?". For our solution, we look at speech conversion and with it we introduce RTVC+AutoVC and RTVC+SpeechSplit two models that integrate implementations of AutoVC [13] and SpeechSplit [14] into the speaker encoder.

6.1 AutoVC & SpeechSplit

Speech conversion aims to convert voices into other voices, which is closely related to speech cloning. Both need an embedding that can represent a voice. By changing specific

parts of said embedding, speech conversion can then change the voice. Finding which part to change -or in other words, finding the relation between the voice and the embedding- is the challenge.



(a) Bottleneck too wide (b) Bottleneck too narrow (c) Bottleneck just right (d) Conversion

Source: AutoVC Paper [13]

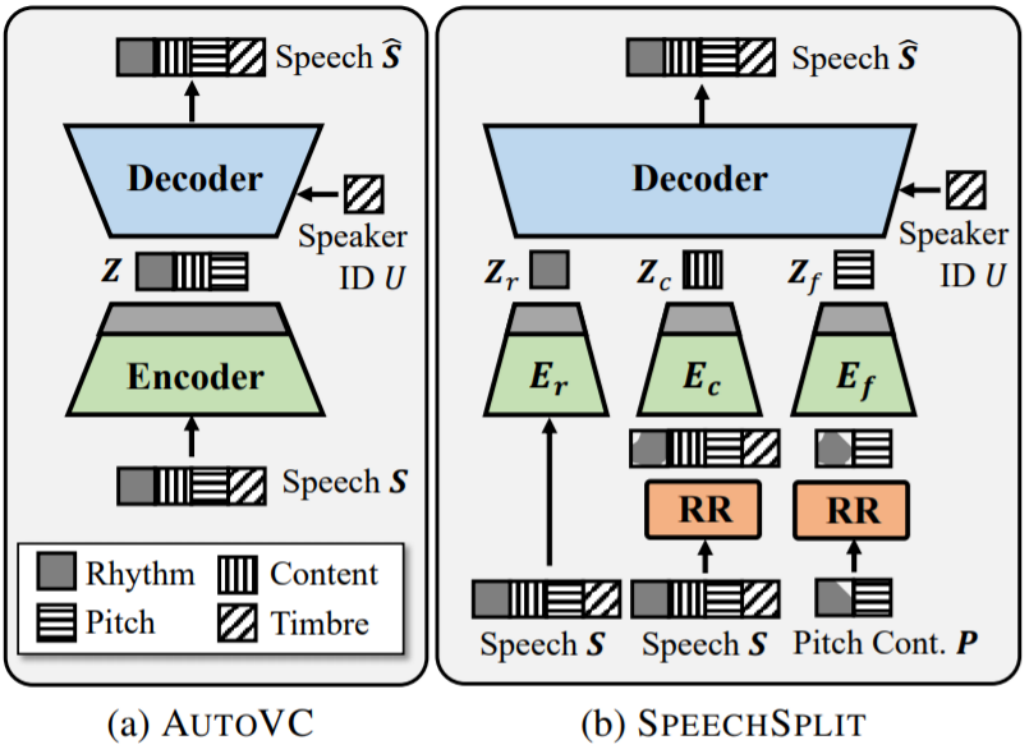
Figure 7: AutoVC Bottleneck

AutoVC [15] and after that SpeechSplit [16] try to solve this using an autoencoder. The idea behind both models is that by inserting the identity information in the decoder the encoder is trained to only find the content-encoding. The original spectrogram contains all information about the speaker, where embedding created from that in the GE2E model represents some information in this spectrogram, in particular the identity information. In Figure 7 we see this part as $S1$ and the grey part of $X1$. By feeding this information to the decoder, the decoder will not need this information from the encoder. The encoder thus learns to output the remaining information to recreate the spectrogram. If the bottleneck, which is the output dimension of the encoder, is too wide the encoder will output the remaining information plus some information that is already in the $S1$ embedding. If it is too narrow, the encoder will not have enough space to output all the information. By finding the right bottleneck we can get the encoder to output the remainder of the information needed to recreate the spectrogram. AutoVC [15] does indeed have the missing information, but it also includes information about what is being said, the text, in the output of its encoder. This is great for speech conversion, where we do not want to change what is being said, but problematic for speech cloning where we want to separate the speaker information and the text information.

This is where we can use SpeechSplit [16], which takes it a step further, as can be seen in Figure 8. In the SpeechSplit [16] paper, the authors refer to the identity information as the Timbre. Other than the Timbre there are three other parts to be found inside the spectrogram: Rhythm, Pitch and Content. The output of the AutoVC [15] encoder has all three of these remaining parts while SpeechSplit [16], as its name suggests, splits these three parts. They do this by using three encoders instead of the one in AutoVC [15]. Each of these encoders is conditioned to only output one of the three parts of the spectrogram.

6.2 Implementation

The implementation of the RTVC+ models required modifications to the original base model. These changes can be summed up into the Speaker Encoder changes and the Synthesizer changes. The overview of the new models can be seen in Figure 9.



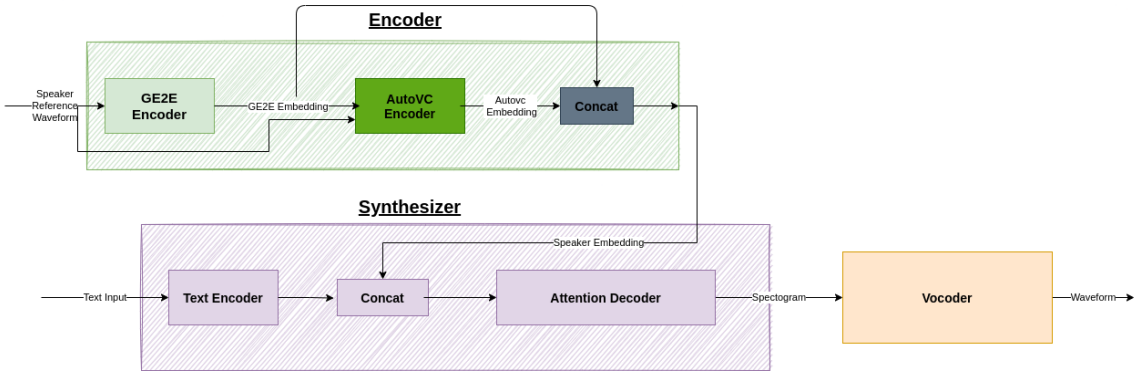
Source: SpeechSplit Paper [14]

Figure 8: SpeechSplit vs AutoVC

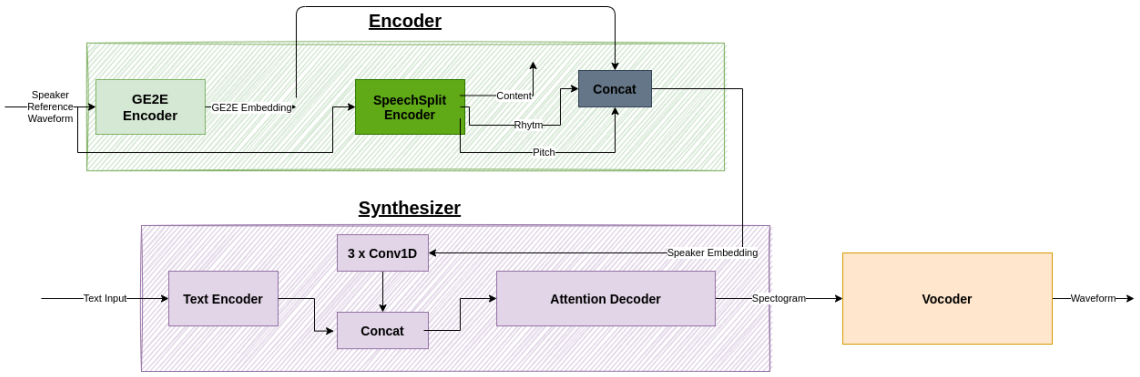
6.2.1 Speaker Encoder

The original AutoVC [15] model was implemented using a GE2E encoder. Integrating AutoVC [15] into the base model only needed the AutoVC [15] model to be retrained on the new GE2E embeddings. As can be seen in Figure 9 ,AutoVC [15] -just like SpeechSplit [16]- needed an extra concatenate function. The main reason for this is that we wanted to keep the original information from GE2E inside the embedding. This meant that other than retraining the model on the GE2E encodings in our base model AutoVC [15].

SpeechSplit [16] was implemented with a one-hot embedding. One-hot embeddings are not usable in zero-shot conversions, so we needed to change that into the GE2E embedding. We overhauled the audio processing part of the code to enable this. Because the output of the encoders was fed directly into the SpeechSplit [16] decoder, we had to add a way to get these outputs when not training the model. Since SpeechSplit is highly dependent on its dimension bottleneck [16], a wrong dimension for the different encoders can lead to completely different results. Finding the right dimensionality bottlenecks was therefore an important part of integrating SpeechSplit [16] with RTVC. The Content encoder in SpeechSplit is mainly responsible for encoding what is being said in the audio fragments [16]. Because this part should not be needed, we have chosen to discard the output of the content encoder as seen in Figure 9b.



(a) SV2TTS+AutoVC



(b) SV2TTS+SpeechSplit

Figure 9: SV2TTS+ with newly added speaker encoders

6.2.2 Synthesizer

To implement the RTVC+ models there was also a need to modify the synthesizer from the base model. Both AutoVC [15] and SpeechSplit [16] used a form of pre-processing for the audio. To guarantee the same quality of embedding there was a need to implement the same pre-processing in the synthesizer. For the RTVC+SpeechSplit model, we added three Conv1D layers to reduce the dimensionality. The main reason being that the flattened embedding generated by SpeechSplit [16] had a dimensionality of 10.000, which resulted in major memory usage and slow training. On the other hand, the RTVC+AutoVC model needed an extra processing step on the audio. This was needed because the size of the output embedding changed depending on the length of the audio while the base RTVC synthesizer was made for a fixed-length audio embedding. We clipped the audio length and padded extra 0's on cases where the audio length was not long enough.

7 Experiments

All our experiments were conducted on an Ubuntu 20.04 machine using an NVIDIA 3070, 8GB GPU. Training of the different models was done on the same machine. Training of both the speaker encoder and the synthesizer was done using the LibriSpeech-train-clean-100 dataset containing 100 hours of English speech and the corresponding text data.

The experiments themselves can be divided into two segments: experiments on the separate speaker encoders and experiments on the complete models. The experiments on the separate speaker encoders used UMAP, a dimensionality reduction tool, to reduce the dimensionality to 2. For our result, we generated 20 audio fragments for each of the RTVC+ models as well as a newly trained base model. Because of our limited computer power, we decided to cap the training of the synthesizer to 100k iterations.

7.1 Speaker Encoder

During the training of AutoVC [15] and SpeechSplit [16], there was a need to know if the embeddings generated made any sense. To get an insight into this we used UMAP to reduce the embeddings to a dimensionality of 2. Once the embeddings were reduced, we plotted them into a 2D graph. A good embedding should have a separation between male and female voices. Our test data was therefore built using 10 male and 10 female voices each with a couple of audio fragments. By plotting male vs females in different colors, we could then get an insight on if and how well the embedding was working. To get further insights we also calculated the average distance between the male and female embeddings after the dimensionality reduction. SpeechSplit [16] generates three embeddings and as such, we calculated the distance for each of the encodings to find their relation.

Both AutoVC [15] and SpeechSplit [16] use bottlenecks to generate the right encoding. Finding the right bottleneck is difficult, so we experimented on the influence of the bottlenecks on the embedding distance to find a relation between the two.

7.2 Complete Model

For our experiments on the complete RTVC+ models, we wanted to find the effects of the RTVC+ models on the four goals of speech cloning: Intelligibility, Naturalness, Clonability and Generalizability.

Although most of the measurements performed use MOS scores, we have tried to create an objective way to measure the four goals. We did this on 20 random audio fragments from speakers which were seen during the training and 20 random audio fragments from speakers unseen during training. The only restriction we set on the audio fragments is that the duration needed to be at least 10 seconds.

Our objective measurements use pre-trained models to generate scores. First, to measure the intelligibility we made use of Google's speech-to-text API using the Python Speech Recognition library. We did this by calculating the cosine similarity between the output of the Google speech API and the real sentence.

For the clonability rate, we used a similar method, using the Python Resemblyzer package [8]. GE2E is great for speech recognition. By using a pre-trained model of GE2E we should therefore be able to get an estimate on the clonability rate. There is a problem with using GE2E however, as we have discussed in section 5.1 distance cannot be used as a scale of how similar different voices are. A high distance between the real and cloned voice does not necessarily mean a low similarity. It means that the GE2E encoder recognizes the cloned and real voice fragments as two different speakers. So instead Resemblyzer [8] uses a threshold on the cosine-similarity of the embeddings to label voices as "real" or "fake". Here, "real" means that the cloned voice is above the threshold and gets recognized as the same speaker. We call the opposite "fake". Using this we can count the amount of cloned audio fragments which are labelled "real". A higher number of "real" labels will mean better performance.

As an additional test, we have calculated our own MOS, using Google Form and one randomly chosen speaker in each of the trained and untrained speaker sets. To measure each of the four goals, the public was asked to rate the truth of the following statements:

- "I can understand what is being said"
- "The audio is generated by a computer"
- "The voices in the audio clips sound similar"

A rating of 1 represented the complete falsity of the statement and a rating of 5 represented complete truth. This was done for each of the three models: RTVC+AutoVC, RTVC+SpeechSplit, the base model as well as a ground truth measurement using a random clip of the same speaker. For the last question: "the voices in the audio clips sound similar" we used another random clip of the same speaker. This same clip was used for comparison on all four cases, three plus the ground truth.

8 Results & Discussions

8.1 AutoVC

In Figure 10 the distance between Male and Female voices are plotted over the training epochs. The first thing that stands out is that the distance seems to start high. This does not seem to make sense, as we would expect the distance to start low and get higher during the training. The reason for this is that the AutoVC [15] encoder takes the spectrogram and the GE2E embedding as input. This means that without training the AutoVC [15] encoder will output the GE2E embedding. But during training, it replaces the embedding to the Content embedding. The effects of this can be seen in Figure 11.

8.1.1 Bottleneck

When the right dimensionality bottleneck is selected the encoder will eventually learn the right way to output the Content embedding. Looking at Figure 10, we see two different dimensionality bottlenecks in blue and red. We expect the overall distance to increase

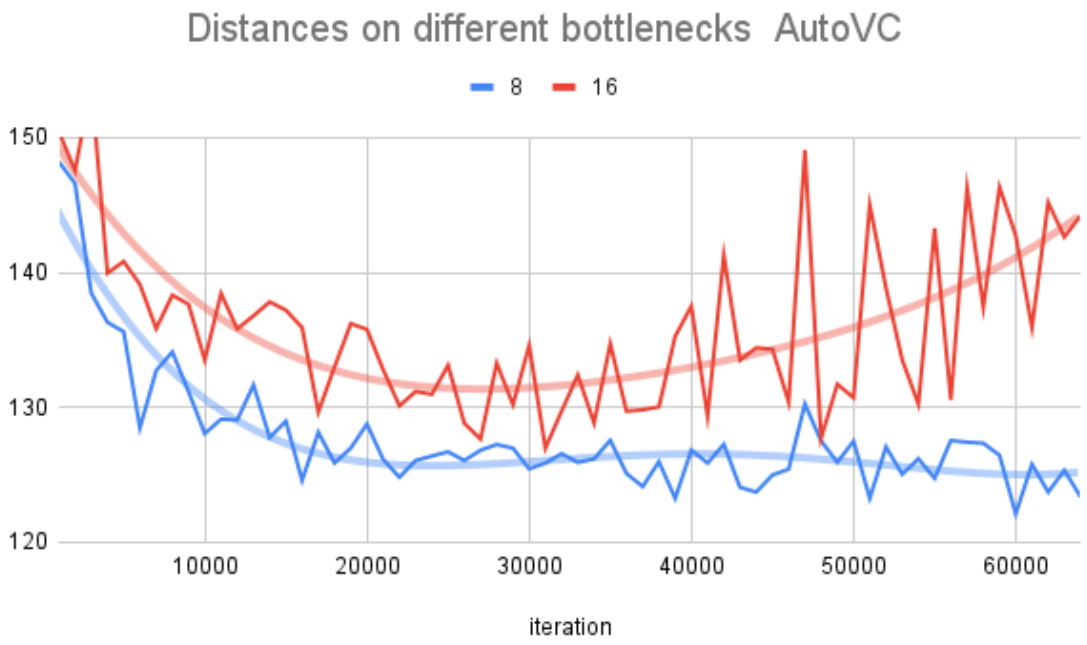
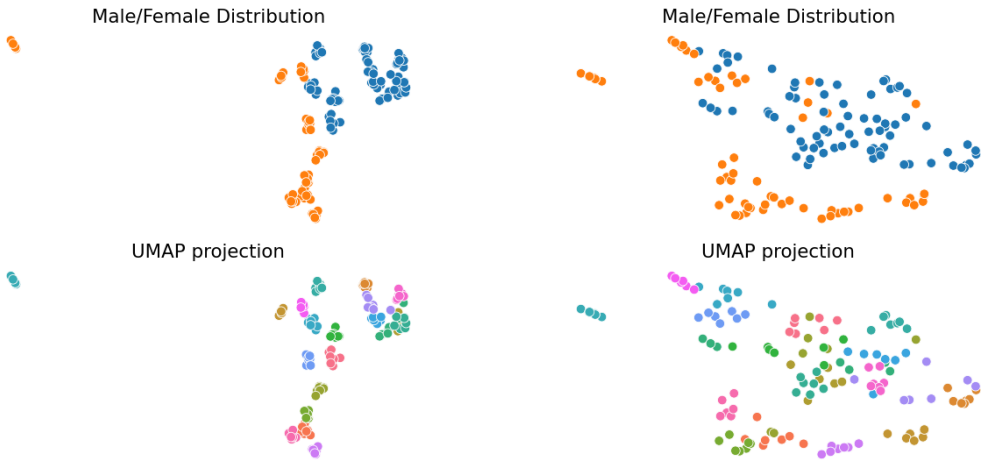


Figure 10: Overal distances between Male and Female Speakers over Epochs

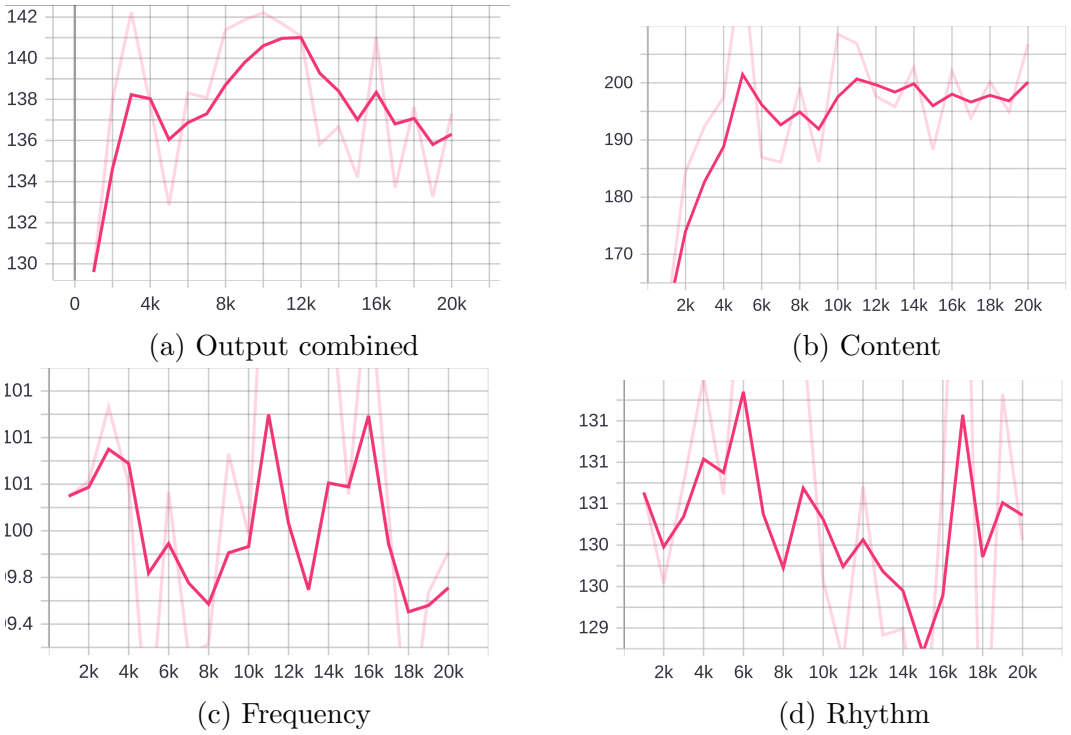


(a) Second Epoch (b) Fifth epoch

Figure 11: Second and Fifth epoch embeddings AutoVC.
Top: Female vs Male distribution. Bottom: Distribution per speaker

for the right embedding and it seems that there is a correlation between the two. When selecting the wrong dimension bottleneck, the distance flattens out while with the right bottleneck, the dimension increases again. Finding the right bottleneck therefore might become easier by comparing the embedding distances.

8.2 SpeechSplit



(a) Output combined

(b) Content

(c) Frequency

(d) Rhythm

Figure 12: Distance Male vs Female embedding on the different encodings from SpeechSplit

As discussed before in section 6.1 SpeechSplit [16], unlike AutoVC [15], enables us to get multiple embeddings, each representing an aspect of speech. This means that we can also plot the different embeddings separate from each other, the result of which gives us a great insight into which part of the embedding is speaker dependent. In Figure 12 we have plotted the distances of each embedding during training. Rhythm seems to have no change in distance during training, which is something that was expected. Not only is it the embedding with the lowest dimension, but the Rhythm of any audio should not be dependent on the gender of the speaker. The high degree of separation in the content encoder was unexpected, however. Like Rhythm, the Content embedding should not be gender specific. There are two possible explanations for this: first, it might be because of our limited data. Our training set only has around 250 speakers and the audio for each speaker is taken from different audiobooks. This means that there is a chance that the contents spoken by each speaker differ enough to create the distance seen between male and female speakers. Another reason might be that the content dimensionality is too big. If the bottleneck is too wide on the content encoding, the resulting embedding will have a lot of the speaker embedding. However, in Figure 13 we find that lowering the bottleneck increases the distance even more. This leads us to believe that the cause is most likely due to the limited training data.

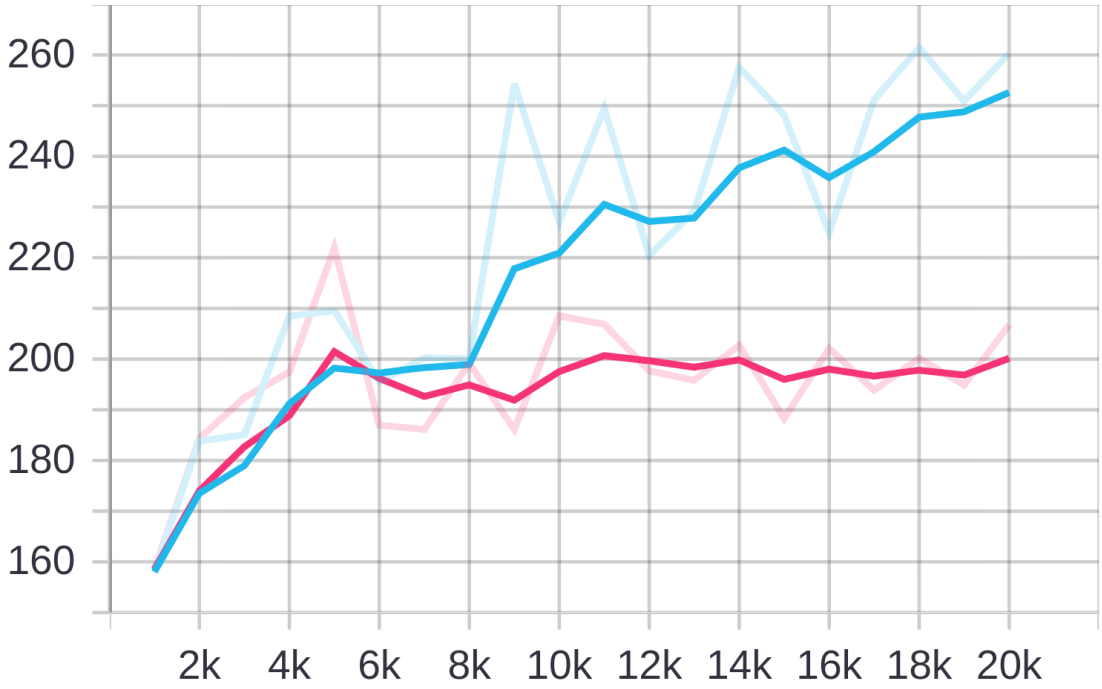


Figure 13: Effect of content bottleneck size on embedding distance between Male and Female voices. Bottleneck: 48 (Blue, Top), 64 (Red, Bottom)

8.2.1 Bottleneck

We have plotted the effects of different bottlenecks in Figure 14. Using the distance to find the right bottleneck dimension does not seem to be quite as impactful as with AutoVC [15]. Other than the Rhythm section of the embedding the other bottlenecks do not seem to have much effect on the distances.

8.3 Complete Model

Our implementations of RTVC+SpeechSplit and RTVC+AutoVC seem to perform significantly worse than our base model in the clonability aspect. SpeechSplit [16] does seem to do better at generalizing data as seen in the similar performance in both seen and unseen data in Table 3 and 4. Both AutoVC [15] and SpeechSplit [16] have a similar MOS on unseen data compared with the base model as seen in Figure 6. This gives us an insight into the ability of the models to generalize. The use of GE2E as our measurement might have contributed to the high scores of AutoVC [15] and the base model in Table 3 and 4. The base model explicitly uses GE2E as means of creating the speaker while AutoVC [15] uses GE2E in its learning process. SpeechSplit [16] however does not use GE2E in its learning process and thus highly different embeddings from SpeechSplit [16] can be expected. The characteristics on which GE2E creates the embeddings might therefore be similar on AutoVC [15] and our base model. This might also explain the difference in score on SpeechSplit [16] as seen in 15. In Figure 15 a comparison between our objective function using GE2E and the MOS on similarity can be seen. The results on our base model and AutoVC [15] are strikingly similar. Although these results give credibility to the

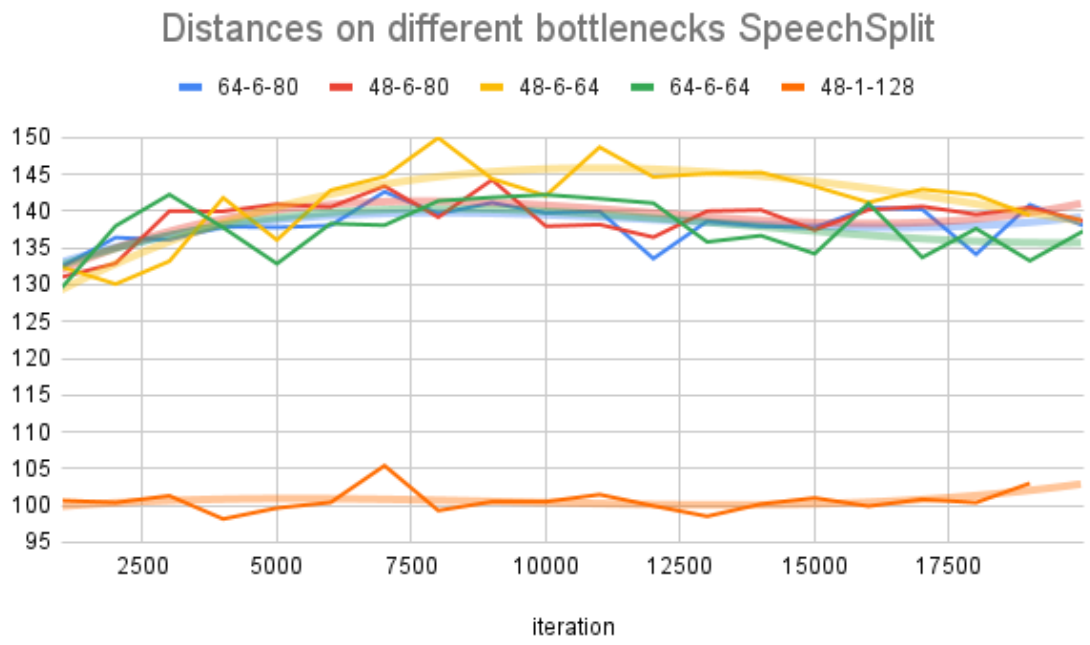


Figure 14: Effect of different bottleneck sizes on distance development. Dimensions given by: Content-Rhythm-Frequency

GE2E measurement, the differences in the SpeechSplit [16] scores underline our concerns on the characteristics front.

The intelligibility scored by using Google's Speech API is more evenly spread. The rather low scores overall in Table 5 are explained by the low amount of iteration on which each model has been trained. Because the text said in both seen and the unseen data is the same, there is no real difference in the scores. Figure 16 has the comparison between the MOS ratings and the ratings obtained using Google's text-to-speech API. The ratings seem to be highly different. A possible explanation is that the two methods measure different aspects of intelligibility. For the MOS score on intelligibility, the public is asked to rate the sentence: "I can understand what is being said". The user does not get the sentence that is being said, and neither is the user asked to exactly compare the original sentence with what is being said, while this is exactly what is being done using the Google API method. The results are therefore influenced by the difference in what is being measured. In Section A all of the tables containing the results of the complete model can be found.

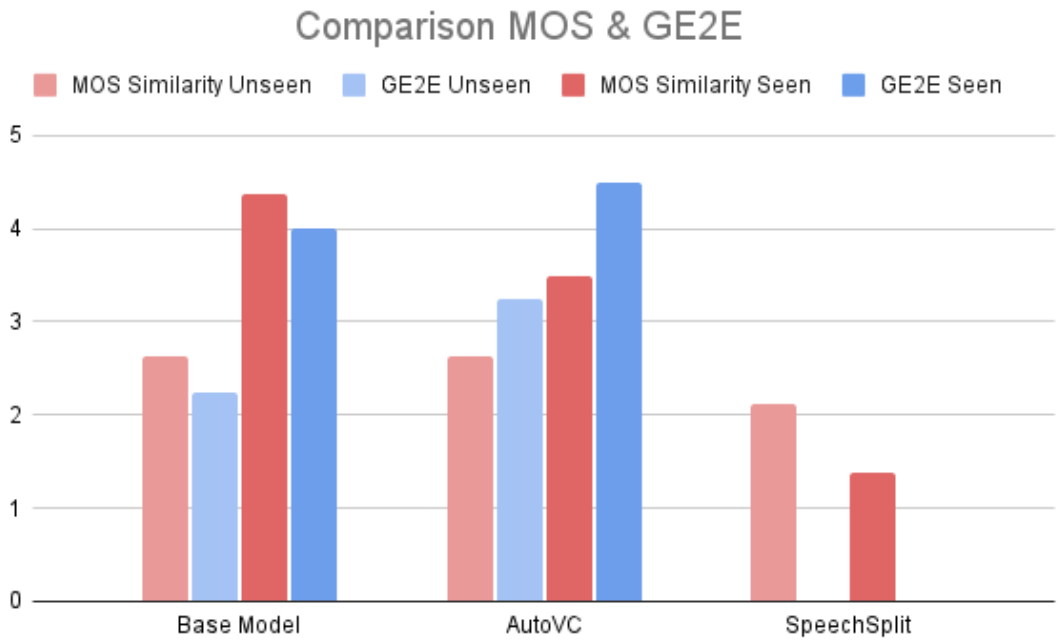


Figure 15: Comparison MOS and GE2E on similarity score

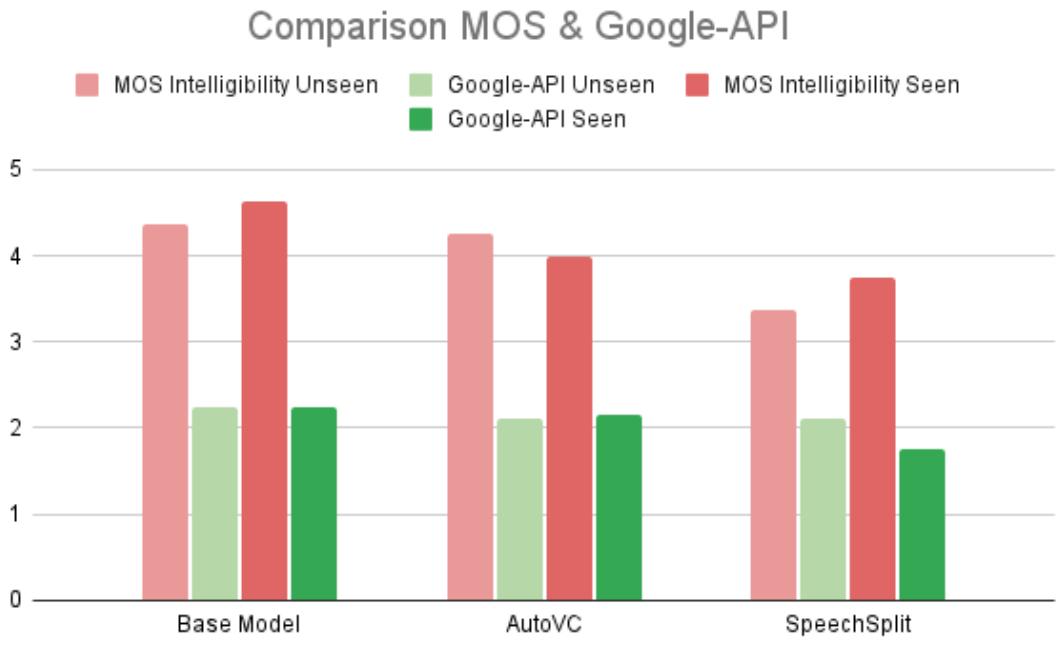


Figure 16: Comparison MOS and Google's text-to-speech API on Intelligibility score

9 Conclusion

There are several aspects to cloning voices. During the research of this thesis, it not only became clearer how far we have already gotten but also how many challenges still lay ahead. GE2E, the method used to create embeddings in our base model, has its limitations as discussed in section 5. Our problem statement thus became: "Is there a way to improve the results of the base model by using a different embedding". In section 6 we introduced our two solutions: RTVC+AutoVC and RTVC+SpeechSplit. Both methods originated from speech conversion and made use of autoencoders to force the encoder part to create the right embeddings. Tuning the bottleneck in autoencoders is difficult, sometimes it is only possible to know how well a certain bottleneck performs after completely training the model. By using a dimensionality reduction method and modelling the distances between male and female speakers we found a way to get insight into the quality of the embeddings during training, which makes early stopping possible in case the bottleneck seems to be wrong. In the results of the experiments, which can be found in section 8, we find that our own models seem to be underperforming compared with the base model. These methods should not be excluded in future work, however. Better results might be found in plenty of ways: by tuning the dimensionality bottlenecks further, by training on more speakers or even by training the encoders longer. Encoders based on speaker verification tasks like the GE2E encoder used in our base model are great, but they are limited. More research should be done in embeddings like SpeechSplit [16] and AutoVC [15] because progress in audio embeddings will be key in solving the many challenges in speech cloning. Not only will improvements in speaker embeddings help models in creating better voices but, if improved enough, it can be used as an objective measurement of the speech cloning criteria.

A
Measurements Complete Model

	AutoVC	SpeechSplit	Base Model
Seen Data	80%	0%	90%
Unseen Data	45%	0%	65%

Table 3: Percentage of generated data that reached threshold of 80% of similarity using GE2E

	AutoVC	SpeechSplit	Base Model
Seen Data	83%	64%	86%
Unseen Data	79%	65%	82%

Table 4: Average similarity on generated audio files using the GE2E embeddings

	AutoVC	SpeechSplit	Base Model
Seen Data	43%	35%	45%
Unseen Data	42%	42%	45%

Table 5: Average similarity score on Google's text-to-speech API using generated audio files

	AutoVC	SpeechSplit	Base Model	Ground Truth
Seen Data	1,75	1,25	2,375	4,875
Unseen Data	2,125	1,75	2	4,25

Table 6: MOS Naturalness

	AutoVC	SpeechSplit	Base Model	Ground Truth
Seen Data	3,5	1,375	4,375	4,75
Unseen Data	2,625	2,125	2,625	4,75

Table 7: MOS Similarity

	AutoVC	SpeechSplit	Base Model	Ground Truth
Seen Data	4	3,75	4,625	5
Unseen Data	4,25	3,375	4,375	5

Table 8: MOS Intelligibility

References

[1] Sercan Arik, Gregory Diamos, Andrew Gibiansky, John Miller, Kainan Peng, Wei Ping, Jonathan Raiman, and Yanqi Zhou. Deep voice 2: Multi-speaker neural text-to-speech. *Conference on Neural Information Processing Systems*, 2017.

[2] Sercan O. Arik, Mike Chrzanowski, Adam Coates, Gregory Diamos, Andrew Gibiansky, Yongguo Kang, Xian Li, John Miller, Andrew Ng, Jonathan Raiman, Shubho Sengupta, and Mohammad Shoeybi. Deep voice: Real-time neural text-to-speech. *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[3] Edresson Casanova, Christopher Shulby, Eren Gölge, Nicolas Michael Müller, Frederico Santos de Oliveira, Arnaldo Candido Junior, Anderson da Silva Soares, Sandra Maria Aluisio, and Moacir Antonelli Ponti. Sc-glowtts: an efficient zero-shot multi-speaker text-to-speech model. *Interspeech*, 2021.

[4] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *ArXiv*, 2014.

[5] Erica Cooper, Cheng-I Lai, Yusuke Yasuda, Fuming Fang, Xin Wang, Nanxin Chen, and Junichi Yamagishi. Zero-shot multi-speaker text-to-speech with state-of-the-art neural speaker embeddings. *IEEE Trans. Acoustics, Speech and Sig. Proc*, 2020.

[6] Daniel W. Griffin, Jae, S. Lim, and Senior Member. Signal estimation from modified short-time fourier transform. *IEEE Trans. Acoustics, Speech and Sig. Proc (ICASSP)*, pages 236–243, 1984.

[7] Corentin Jemine. Real-time voice cloning. Master's thesis, 2019. URL `https://matheo.uliege.be/handle/2268.2/6801`. Last Accessed: 20-08-2021.

[8] Corentin Jemine. Resemblyzer, 2020. URL `https://github.com/resemble-ai/Resemblyzer`. Last Accessed: 20-08-2021.

[9] Corentin Jemine. Implementation real time voice cloning, 2020. URL `https://github.com/CorentinJ/Real-Time-Voice-Cloning`. Last Accessed: 20-08-2021.

[10] Ye Jia, Yu Zhang, Ron J. Weiss, Quan Wang, Jonathan Shen, Fei Ren, Zhifeng Chen, Patrick Nguyen, Ruoming Pang, Ignacio Lopez Moreno, and Yonghui Wu. Transfer learning from speaker verification to multispeaker text-to-speech synthesis. *Advances in Neural Information Processing Systems 31*, 2018.

[11] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. *arXiv*, 2018.

[12] Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan O. Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. Deep voice 3: Scaling text-to-speech

with convolutional sequence learning. *International Conference on Learning Representations*, 2018.

[13] Kaizhi Qian. Autovc, 2020. URL `https://github.com/auspicious3000/autovc`. Last Accessed: 20-08-2021.

[14] Kaizhi Qian. Speechsplit, 2020. URL `https://github.com/auspicious3000/SpeechSplit`.

[15] Kaizhi Qian, Yang Zhang, Shiyu Chang, Xuesong Yang, and Mark Hasegawa-Johnson. Autovc: Zero-shot voice style transfer with only autoencoder loss. *arXiv v2*, 2019.

[16] Kaizhi Qian, Yang Zhang, Shiyu Chang, David Cox, and Mark Hasegawa-Johnson. Unsupervised speech decomposition via triple information bottleneck. *arXiv v5*, 2020.

[17] Donald Shankweiler and Carol A. Fowler. Seeking a reading machine for the blind and discovering the speech code. 2014. URL `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm`. Last Accessed: 14-08-2021.

[18] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, RJ Skerry-Ryan, Rif A. Saurous, Yannis Agiomyrgiannakis, and Yonghui Wu. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. *IEEE Trans. Acoustics, Speech and Sig. Proc (ICASSP)*, 2018.

[19] Jose M. R. Sotelo, Soroush Mehri, Kundan Kumar, J. F. Santos, Kyle Kastner, Aaron C. Courville, and Yoshua Bengio. Char2wav: End-to-end speech synthesis. *International Conference on Learning Representations (ICLR)*, 2017.

[20] Yaniv Taigman, Lior Wolf, Adam Polyak, and Eliya Nachmani. Voiceloop: Voice fitting and synthesis via a phonological loop. *arXiv: Learning*, 2018.

[21] Xu Tan, Tao Qin, Frank Soong, and Tie-Yan Liu. A survey on neural speech synthesis. -, 2021.

[22] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *9th ISCA Speech Synthesis Workshop*, 2016.

[23] Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C. Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel wavenet: Fast high-fidelity speech synthesis. *International Conference on Machine Learning*, 2018.

[24] Li Wan, Quan Wang, Alan Papir, and Ignacio Lopez Moreno. Generalized end-to-end loss for speaker verification. *IEEE Trans. Acoustics, Speech and Sig. Proc (ICASSP)*, 2017.

[25] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. Tacotron: Towards end-to-end speech synthesis. *Interspeech*, 2017.