# Few-shot Voiced Text-to-Speech by Backpropagating Through a Triplet Loss-trained Voice Encoder

James Gabriel Z. Casia

Department of Computer Science, University of the Philippines Cebu, jzcasia@up.edu.ph

A text-to-speech (TTS) system is a system that converts language text into speech. TTS has long been a hot research topic in the field of artificial intelligence, primarily attributed to its relevance in human-computer interfaces. TTS technology has grown from concatenative synthesis to parametric synthesis using deep neural networks and has achieved human-level performance. Today, TTS is used in video narration, voice messages, video streaming services, virtual assistants and more, but these TTS systems often still use a generic, and unnatural sounding voice. In this age of online content creation, there is a shift towards personalization. Thus, studies towards personalizing TTS are a step in the right direction towards improving user experience and adheres with the trend of personalization. This study proposes a novel way of adapting the Tacotron model to generate speech in a target voice by backpropagating through a Convolutional Neural Network based voice encoder loss. This study also uses few-shot learning techniques, which are also used to aid in training, and their influence on the overall model performance is measured. Results show that the choice of hyperparameter values did not significantly affect the model performance, and that the proposed method of backpropagating through the voice loss has negatively affected overall model performance with the resulting model not able to synthesize speech. The best performing model, however, was able to synthesize intelligible and similar speech though it suffers from issues linked to overfitting. Human evaluation results reveal that synthesized speech samples were similar and natural.

**CCS CONCEPTS** • Theory of computation • Theory and algorithms for application domains • Machine learning theory • Semi-supervised learning

**Additional Keywords and Phrases:** text-to-speech, artificial intelligence, tacotron, convolutional neural network, few-shot learning, speech synthesis

## 1 INTRODUCTION

Today, the use of computers and the internet is integral in communication. Hence new technologies such as text-to-speech systems are built to optimize and make communication more efficient. This study is centered around text-to-speech and is centered around finding ways to improve it. A text-to-speech (TTS) system is a system that converts language text into speech [Speech Synthesis, n.d.]. TTS has long been a hot research topic in the field of artificial intelligence, primarily attributed to its relevance in human-computer interfaces. TTS technology has grown from concatenative synthesis to parametric synthesis using deep neural networks and has achieved human-level performance in terms of intelligibility and naturalness, the two main metrics TTS systems are evaluated for [Ren et al., 2019]. Today, TTS is used in video narration, voice messages, video streaming services, virtual assistants, and more, but these TTS systems often still use a generic and unnatural sounding voice. In this age of online content creation, there is a shift towards personalization. Examples of this are tailored searches and personalized advertisements. Thus, studies towards

personalizing TTS are a step in the right direction towards improving user experience and adheres with the trend of personalization.

A study by Jia et al. [2019] explores the challenge of voiced or multi-speaker text-to-speech systems and aims to generate voiced speech (speech in the target voice) with just a voice embedding alone. Their study although showing promising results has achieved a significantly low speaker similarity compared to its naturalness.

This study proposes a novel way of adapting the Tacotron model to generate speech in a target voice by backpropagating through a Convolutional Neural Network based voice encoder loss. This study also uses few-shot learning techniques to aid training and aims to measure the influence of such techniques. Few-shot learning techniques have been widely explored in image related tasks, but the same thing cannot be said about audio related tasks. This study will provide more insights on the effectivity of few-shot learning techniques when applied to audio and speech related tasks. Also, the development of personalized text-to-speech is a step forward in using artificial intelligence to create more personalized, and much more user-friendly experiences.

## 2 RELATED STUDIES

This section details the relevant literature that has influenced this study. The subsections reveal the technologies used and the underlying concepts that lay the groundwork of this study.

### 2.1 Tacotron

Tacotron is an end-to-end TTS model developed by Google. On the contrary to TTS systems wherein they often consist of multiple feature extraction stages such as text analysis, acoustic model, and the audio synthesis module which heavily requires domain expertise, Tacotron directly synthesizes speech from characters hence the term end-to-end. In the study by Wang et al. [2017], Tacotron achieves a 3.82 subjective 5-scale mean opinion score (MOS) on US English and outperforms a production parametric system in speech naturalness. Tacotron was also found out to be substantially faster than sample-level autoregressive based techniques [Wang et al., 2017].

Tacotron2 is the second version of Tacotron. Tacotron2 mainly improves upon Tacotron by making the model more compact. Ablation studies were performed wherein sections of the original Tacotron model that had little to no effect on performance were identified and pruned. Tacotron2 was able to achieve a MOS of 4.53, which was very close to 4.58, the benchmark score for professionally recorded speech. This means that the model was able to produce speech samples that were natural and intelligible, very close to professionally recorded speech [Shen et al., 2018]. The main contribution of this study (Tacotron2) is that it provides a high-performing baseline model that can be easily built upon. Tacotron2 implementations are available in popular deep learning libraries such as *Tensorflow* and *Pytorch*. Because of its high performance, the excellent documentation, availability in multiple libraries, and the wide community support, Tacotron2 becomes the ideal choice for a baseline model. The main gap with the study by Shen et al. [2018] is the fact that it is not tuned to the problem of voiced TTS.

### 2.2 Multi-speaker Text-To-Speech Synthesis

Multi-speaker text-to-speech synthesis is the creation of text-to-speech systems that is able to synthesize speech in different voices, hence the term multi-speaker. A study by Jia et al. [2019] aimed to create a multi-speaker TTS system in a zero-shot fashion. This study (Multi-speaker TTS) aimed to only require a target voice vector to be able to synthesize audio similar to the target voice without retraining. For the voice encoder, the study used recurrent neural networks, specifically 3 stacks of LSTMs with 768 cells per layer and followed by a projection head of 256 dimensions. The study

employed a window size of 800ms with 50% overlap. The model was able to score an average of 17.72 when trained and tested on different datasets. In terms of naturalness, the model was able to achieve an average mean opinion score of 4.145 and 2.95 in terms of speaker similarity [Jia et al., 2019].

The contribution of this study (Multi-speaker TTS) is its novel way of achieving zero-shot multi-speaker TTS synthesis by injecting the voice vector into the synthesizer. The gap in this study (Multi-speaker TTS), however, is that it achieves a relatively low speaker similarity of 2.295 compared to its naturalness score of 4.145. This is attributed to the fact that zero-shot learning is a much more complex task compared to few-shot learning and definitely, many-shot learning.

### 2.3 Speaker Verification

Speaker verification is the process of encoding voice vectors from speech and ensuring voice vectors from different speakers are far apart and vectors from the same speaker are close together. A study by Khan & Hernando [2020] attempted to perform speaker verification by proposing three different encoder systems, Nearest Neighbor Autoencoder, Double-branch Siamese network, and Triple-branch Siamese network. In that study, the Nearest Neighbor Autoencoder system was basically a pre-trained autoencoder that took in a voice vector, and subsequently down-sampled, and up-sampled it into a same width vector. Nearest Neighbors was then performed to identify which voice vectors belonged to the same speaker. The Double-branch, and Triple-branch were both Siamese networks of VGG-inspired Convolutional Neural Networks. In the dataset for the Double-Branch Siamese network system, each line contained a pair of spectrograms and their corresponding label. The label denoted whether the spectrogram pair was from the same speaker or not. The dataset for the Triple-Branch System was an implementation of the triplet loss, wherein triplets of spectrograms were passed. These triplets comprised of the anchor, the negative, and the positive. The anchor is the reference spectrogram. The negative is a different spectrogram from a different speaker and the positive is a different spectrogram but from the same speaker. The models were then trained on the VoxCeleb-2 development set and tested against VoxCeleb-1. With the VoxCeleb-1 test set, the autoencoder system performed the worst with a high equal error rate (EER) of 10.2, while the double-branch and triple-branch networks achieved an EER of about ~6.92. An ensemble of the three systems was tested on external datasets such as VoxSRC-20 test and VoxSRC-20 validation set. The ensemble performed worse on the VoxSRC-20 datasets achieving an average of ~14.5 EER and performed good with an EER of 5.58 on the VoxCeleb-1 test set [Khan & Hernando, 2020].

The main contribution of this study by Khan & Hernando [2020] is its use of various techniques to perform speaker verification. Although the goal of their study was not to directly compare the different techniques, its results show that its implementation of CNN-based double-branch and triple-branch networks were effective in the speaker verification problem. A problem of this study that was not addressed was its relatively poor performance on external datasets. Since this study only performs speaker verification, additional work must be done in order to create a functioning voiced TTS system.

### 2.4 Backpropagating through Neural Network-based loss functions

Loss functions in general are purely comprised of mathematical functions. New studies, however, have been published that entail using actual neural networks as part of the objective function. One example is a study by Mathiasen & Hvilshøj [2021] that is about backpropagating through the Fréchet Inception Distance (FID). The performance of generative models is usually measured through their FID. The FID is a measure of their ability to generate a batch of images that is close to the embeddings of real images. The FID entails the use of a CNN, in most cases, the Inception

model, and compares the distance between the embeddings of the generated images and real images. The study by Mathiasen & Hvilshøj [2021] had the intention of adapting generative adversarial networks (GANs) to achieve better FIDs by directly backpropagating through the FID. Since the introduction of such metric involved an extra CNN model, this introduced a high time complexity during both forward propagation and backpropagation. To cater to this increased processing complexity, the study by Mathiasen & Hvilshøj [2021] also aimed to optimize the training speed through an algorithm that took advantage of calculating the eigenvalues of small matrices faster. The optimization was successful, and it was claimed to speed up computation at least 25 times [Mathiasen & Hvilshøj, 2021].

The main contribution of the study by Mathiasen & Hvilshøj [2021] is its ingenious strategy of backpropagating through a neural network-based loss function. The problem in the study (Backpropagating through FID) is in the computer vision domain, which is a far cry from the audio domain of this study.

In summary, the main gap found is that TTS systems are not tuned to the problem of voiced TTS. Some concepts that bridge the gap between TTS and voiced TTS have been studied, but they are not without problems. The speaker verification system developed by Khan & Hernando [2020] showed poor performance on external datasets. Zero-shot learning-based voiced TTS proved to have subpar performance on similarity. Backpropagating through CNN-based loss functions have not been applied and studied in the audio space. Hence, an opportunity lies in creating a voiced TTS system that is able to learn from a few speech samples. Instead of zero-shot learning, which achieved poor similarity, few-shot learning techniques is proposed to balance between similarity score and low data proficiency. To aid few-shot learning, a speaker verification-trained CNN voice encoder could be used as an additional voice loss function that would help adapt the synthesizer to the target voice. Experimentation on different and wider embedding sizes is proposed to improve from the work of Khan & Hernando [2020], which had low performance on external datasets.

## 3 METHODOLOGY

This section is divided into four subsections, Analysis, Design, Implementation, and Testing and Evaluation. Each subsection contains the details of the steps taken in this study.

### 3.1 Analysis

Text-to-speech systems, despite their ability to synthesize intelligible and natural sounding speech, were not really tailored to generate high quality speech samples in a target voice with only a few examples. Thus, extra work had to be done to adapt TTS to synthesize speech in specific voices. The adaption of standard TTS models to specific voices, referred to in this study as voiced TTS, was an interesting field of research as it coincided with the trend of improving user experience by potentially replacing generic and unnatural sounding TTS systems with more familiar or personal voices.

Zero-shot multi-speaker systems had also been explored by other studies in performing voiced TTS, but these systems still performed quite poorly in achieving speaker similarity. This was partially due to the nature of zero-shot learning. Zero-shot learning, while robust in low-data scenarios, struggles with performance compared to few-shot and many-shot learning.

### 3.2 Design

There were fields of studies that would potentially bridge the gap between TTS and voiced TTS. One of which was speaker verification. In order to generate speech in a specific voice, TTS systems must be able to identify the unique

characteristics of each voice first and encode that into an embedding vector. This was important since the TTS system would need to ingrain the characteristics of a particular voice in the synthesizer.

The speaker verification system would employ a voice encoder, which would usually be a neural network that would encode a speech sample into a voice vector. LSTMs were used by the study of Jia et al. (2019) on multi-speaker TTS synthesis, but LSTMs were considerably slower than CNNs due to their sequential and autoregressive nature compared to the high parallelizability of CNNs. In the Multi-speaker TTS study (Jia et al, 2019), where the voice encoder was only used once on the generation of the voice vector, performance did not really matter. In this study, where the voice encoder would be used in training for both forward propagation and back propagation, performance would matter a lot. Hence CNNs would be a great candidate for the voice encoder.

Another potentially relevant topic would be the technique of backpropagating through neural network-based loss functions. The optimization objective of voice matching would be difficult because no purely mathematical loss function or algorithm existed that would measure the distance between two voices. Hence, to achieve this objective, neural networks, with their proven robustness and performance, could be used. The idea of backpropagating through neural networks would enable us to create an objective function that would guide the TTS model to synthesize speech in a certain voice. With the use of a voice encoder that would be able to encode the unique characteristics of a voice in a vector, it would be possible to create a voice loss function incorporating such voice encoder that would measure the distance between two embeddings. Backpropagation could then be done through this voice loss function to adapt the synthesizer to synthesize speech in the target voice. This potentially would enhance training and help the model learn even with few examples.

A voiced TTS system would be designed wherein a CNN-based voice loss would be backpropagated through to adapt the model to the target voice. Few-shot learning techniques would also be done to aid model training. Finally, hyperparameter selection would be done to assess the influence of few-shot learning techniques and the voice loss. The following are the specific steps followed in creating the few-shot voiced TTS system.

### *3.2.1 Creation of the Triplet Dataset for the Voice Encoder*

Following the approach of Khan & Hernando [2020] in their speaker verification study, a triplet dataset had to be created. The goal of this triplet dataset would be to teach the voice encoder model to contrast and consolidate differing and similar voices of speech samples. Using the non-identifying speaker labels of the *CommonVoice* dataset, the triplets would be selected in a self-supervised fashion. For the anchor and positive, audio samples of the same speaker would be chosen. For the negative, a random audio sample from a random speaker would be selected. No identification of hard triplets would be done due to intensive labor required to identify such triplets, and it would be expected that the sheer number of triplets would compensate.

### *3.2.2 Voice Encoder Model Variation Experiments*

In order to ensure a better than worst performance while not necessarily spending too much resources on training and identifying the models with the right hyperparameters, two models would be created. These two models would be CNNs inspired from the VGG architecture and would have different embedding sizes and number of parameters. The model with the better performance would be incorporated in the voice loss.

### *3.2.3 Training the Voice Encoders*

The training of the models would have potentially taken a very long time because of the huge dataset and the performance bottleneck of loading audio files from disk into memory. Hence parallel processing would be done in order to train the voice encoder in a sensible time. Due to the sheer duration of training, no early stopping would be performed. Instead, training would be manually stopped to train the model as long as possible. No repetitions would be done as well due to the same reason.

#### *3.2.4 Voice Loss*

The trained models would be compared based on the L1 Loss, ROC Area under the curve (ROC AUC), and equal error rate (EER). The better performing model would be incorporated with the voice loss function. The L2 distance between the target voice embedding and the synthesized speech voice embedding would encompass the voice loss.

#### *3.2.5 Creation of the few-shot dataset*

Few-shot learning techniques would be explored to potentially help the model train faster. An experiment using different hyperparameter values for the few-shot learning techniques would be done. Gradient clipping would be a technique that would help against exploding gradients by truncating the norms of the gradients to a certain value [Goodfellow et al., n.d.]. Values of 0.5 and 1.0 would be used. Layer freezing would be a technique in which layers of the model would be frozen to prevent re-training and speed up training [Brock et al., 2017]. Freezing the encoder, decoder and postnet layers would be experimented with. Elastic weight consolidation (EWC) would be performed to deal with catastrophic forgetting due to finetuning on a slightly different task of voiced TTS. EWC would constrain important parameters to retain their values thus maintaining representations on the previous task while also fitting to the new task [Kemker et al., 2017]. For EWC, a lambda of 0.0 (no EWC), 2.0, and 4.0 would be tested. The incorporation of the voice loss would be experimented with to visualize how the models would perform with or without it.

#### *3.2.6 Training the Synthesizer*

Finally, the synthesizer would be trained incorporating the configurations detailed in the hyperparameter space. A pre-trained Tacotron model would be tuned to the new speech samples and the incorporated voice loss. For clarity, the models would be trained with no randomization in the data across models and would be trained in parallel. Just like with the voice encoder, due to the lengthy duration of training, no early stopping would be performed. Instead, training would be manually stopped to train the model as long as possible. No repetitions would be done as well due to the same reason.

#### *3.2.7 Model comparison and selection*

The trained models would be compared through their resulting losses. The best model would then be selected and continued training. The best model would be then used for human evaluation.

#### *3.2.8 Overview*

Figure 3.1 shows an overview of the path of the forward propagation and backpropagation (error) signal. The black arrows represent the forward propagation signal, and the red arrows represent the backpropagation signal. The backpropagation signal flows from the two loss functions, voice loss and Tacotron loss to the synthesizer (Tacotron). The synthesizer then would update its weights based on the gradients to minimize both losses. Note that the vocoder is not included for simplicity. Figure 3.2 shows the flow of data. A user would simply add input text, then the synthesizer would output the voiced speech.

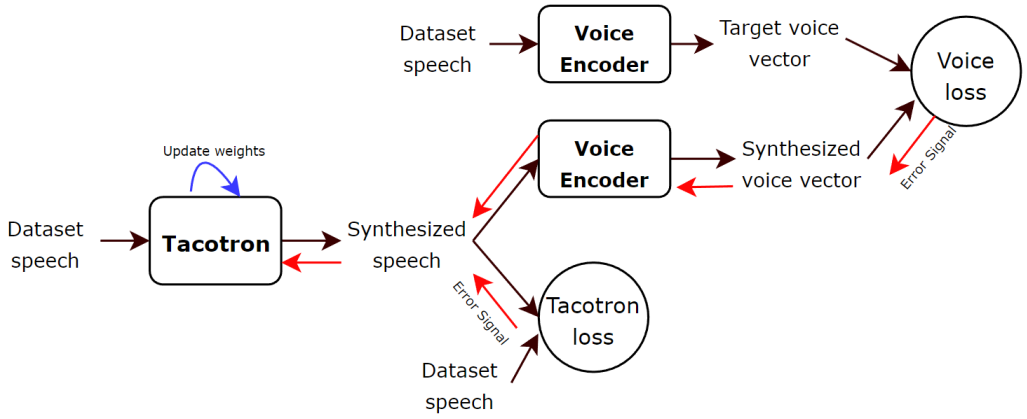


**Figure 3.1.** Forward propagation and backpropagation signal path.


**Figure 3.2.** Data flow diagram of voiced TTS.

### 3.3 Implementation

This subsection details how this study was achieved based on the research design. This part is further subdivided into sections detailing the steps.

#### *3.3.1 Creation of the Triplet Dataset for the Voice Encoder*

*CommonVoice* Corpus 1.0 was the version used for this project, which had a total of 21 GB, 803 total hours, and 33,541 unique voices. In the dataset, 21% claimed to have US English accent, and 41% claimed to be male and 10% claimed to be female [Mozilla Common Voice, n.d.]. Only a portion of the dataset was used. A maximum of 30 recordings were selected randomly per speaker. For the anchor and positive, each recording was paired with another recording of the same speaker forming a maximum of 435 triplets per speaker. For the negative, a random audio from a random speaker was used. This resulted into a triplet dataset of 1.2 million examples. *Pandas*, and *PandaSQL* was used to preprocess and create the dataset.

### *3.3.2 Voice Encoder Model Variation Experiments*

Two separate models were created for experimentation. Both models were inspired from the VGG architecture and varied number of parameters and embedding dimension. The first model, nicknamed VGG2 had 1.2 million parameters and had a 512-embedding dimension. The second model, nicknamed VGG3 had 250,000 parameters and had a 256-embedding dimension. Both models contained fully connected layers in their output layers. The neural network library of *Pytorch* was used to design the models.

### *3.3.3 Training the Voice Encoders*

Training took very long due to the performance bottleneck caused by reading audio files from the storage drive into memory, and the sheer amount of audio files itself. Parallelism techniques was necessary to get decent results in an attainable time. Multithreading was done using the threading library of Python, and a separate thread was spun to load each audio file. Batching was performed, and a maximum batch size of 512 was identified. A gradient step was the processing of a batch of triplets. Standard *adam* optimizers from the *Pytorch* library with a learning rate of 1e-3, and the L1 Loss (Mean absolute Error) was used. *Wandb*, a machine learning graphing software was used to chart the model loss, ROC AUC, and EER. The models were trained for roughly 4,300 steps.

### *3.3.4 Voice Loss*

Models were compared based on the L1 Loss, the test ROC-AUC, and EER on two datasets, *CommonVoice* and *LibriSpeech* dataset. The best performing model was selected and incorporated with the TTS synthesizer for the voice loss.

### *3.3.5 Finetuning Tacotron.*

There were 36 models in total with different hyperparameter configurations. The configurations are detailed in Table 3.1. In creating the few-shot dataset, self-recording was done with a mobile phone and random sentences read from *RandomSentenceGenerator*. Sentences were short and consisted of 5 words on average. The recordings were short with roughly 3 seconds each amounting to 90 seconds in total and roughly 90 unique words. A total of 30 audio samples were recorded. Half of the recordings were used for training and the remaining half for testing. The target voice embedding was then calculated through getting the average voice vector of the training samples.

The Tacotron model chosen was a *Pytorch* implementation by McCarthy [2022]. This model was pre-trained on 180,000 steps on the *LJSpeech* dataset. It was found out that the machine could handle 36 models at a time. Hence all the 36 models were trained in parallel. The *Pytorch adam* optimizer was used with 1e-4 learning rate. The models were trained for 7 hours with 38,000 steps. The models were then compared through their training and testing losses. The identified best model was then trained for an additional 190,000 steps for 33 hours. Loss curves and metrics of the training were logged using *Wandb*.

**Table 3.1.** Hyperparameter configuration of the models.

| Name | gradient_clip | voice_loss | layer_freeze | ewc_lambda |
|---|---|---|---|---|
| ewc5_model_0 | 1.0 | False | 0 | 0.0 |
| ewc5_model_1 | 1.0 | False | 1 | 0.0 |
| ewc5_model_2 | 1.0 | False | 2 | 0.0 |
| ewc5_model_3 | 1.0 | False | 0 | 4.0 |
| ewc5_model_4 | 1.0 | False | 1 | 4.0 |
| ewc5_model_5 | 1.0 | False | 2 | 4.0 |
| ewc5_model_6 | 1.0 | False | 0 | 2.0 |
| ewc5_model_7 | 1.0 | False | 1 | 2.0 |
| ewc5_model_8 | 1.0 | False | 2 | 2.0 |
| ewc5_model_9 | 1.0 | True | 0 | 0.0 |
| ewc5_model_10 | 1.0 | True | 1 | 0.0 |
| ewc5_model_11 | 1.0 | True | 2 | 0.0 |
| ewc5_model_12 | 1.0 | True | 0 | 4.0 |
| ewc5_model_13 | 1.0 | True | 1 | 4.0 |
| ewc5_model_14 | 1.0 | True | 2 | 4.0 |
| ewc5_model_15 | 1.0 | True | 0 | 2.0 |
| ewc5_model_16 | 1.0 | True | 1 | 2.0 |
| ewc5_model_17 | 1.0 | True | 2 | 2.0 |
| ewc5_model_18 | 0.5 | False | 0 | 0.0 |
| ewc5_model_19 | 0.5 | False | 1 | 0.0 |
| ewc5_model_20 | 0.5 | False | 2 | 0.0 |
| ewc5_model_21 | 0.5 | False | 0 | 4.0 |
| ewc5_model_22 | 0.5 | False | 1 | 4.0 |
| ewc5_model_23 | 0.5 | False | 2 | 4.0 |
| ewc5_model_24 | 0.5 | False | 0 | 2.0 |
| ewc5_model_25 | 0.5 | False | 1 | 2.0 |
| ewc5_model_26 | 0.5 | False | 2 | 2.0 |
| ewc5_model_27 | 0.5 | True | 0 | 0.0 |
| ewc5_model_28 | 0.5 | True | 1 | 0.0 |
| ewc5_model_29 | 0.5 | True | 2 | 0.0 |
| ewc5_model_30 | 0.5 | True | 0 | 4.0 |
| ewc5_model_31 | 0.5 | True | 1 | 4.0 |
| ewc5_model_32 | 0.5 | True | 2 | 4.0 |
| ewc5_model_33 | 0.5 | True | 0 | 2.0 |
| ewc5_model_34 | 0.5 | True | 1 | 2.0 |
| ewc5_model_35 | 0.5 | True | 2 | 2.0 |

### 3.4 Testing and Evaluation

The voice encoder was evaluated and compared using their L1 Loss, Receiver operating Characteristic Curve Area under the curve (ROC AUC), and equal error rate (EER) on the *CommonVoice* and the *LibriSpeech* dataset. The ROC

AUC is a non-parametric measure that would quantify the true positivity and false positivity rate of a classifier. It has a range of 0.0 to 1.0 with a higher score signifying better classification accuracy [Classification, n.d.]. The EER is simply an extension of the ROC curve. It is the point of the ROC curve that touches the descending diagonal line. The EER ranged from 0.0% to 100%, and a lower EER meant lesser classification error [Shen, 2020]. *Numpy* and *Sklearn* metrics was used in calculating the metrics. Since the voice encoder experiment was simplistic and had no repetitions, no statistical analysis was performed, and only simple comparison was done to select the voice encoder. The chosen model was then employed in the voice loss.

The 36 synthesizer models were evaluated based on their final training and test loss. The best performing model was identified through comparison and then trained. The naturalness and similarity of the best model was then evaluated through a human opinion survey and measured by the Mean Opinion Score (MOS). MOS is a metric commonly used to measure quality based on human opinion. The MOS ranges from 0 to 5, with an increasing score signifying better or higher quality. The survey was done on Google Forms. The survey initially asked whether the respondent wore headphones with little background noise, headphones with some background noise, and speakers. Respondents were then asked to rate the voice similarity of 7 query samples (TTS generated) versus the single reference sample (real voice). Respondents were also asked to rate the naturalness of these samples. Responses were rated in a 0 - 5 scale. Sample query on voice similarity and naturalness can be seen in Appendix B. The full survey link is accessible in Appendix D. The survey garnered 36 total responses, and the results are detailed in the results section.

## 4 RESULTS

This section presents how the dataset was preprocessed and the results of the training. The evaluation results are shown here, and how they addressed the gaps identified in the second chapter were also discussed.

### 4.1 Voice Encoder

Model details and training results are shown in Table 4.1. VGG2 was trained for 4,400 steps and VGG3 for 4,200. VGG2 achieved a ROC-AUC score of 0.958, while VGG3 achieved a ROC-AUC score of 0.966 on the *CommonVoice* test set. VGG2 achieved 0.932, and VGG3 achieved a 0.962 ROC-AUC on the *LibriSpeech* test set. The performance of VGG2 and VGG3 on the same dataset with the training set was roughly on par with each other. VGG3, however, has a higher performance on the external test set. No statistical analysis could be performed due to the absence of repetitions, but even if the difference between VGG2 and VGG3 was not significant, VGG3 was still superior as it was able to achieve roughly the same performance even with 1/5th the number of parameters. Thus, VGG3 had been selected to incorporate with voice loss for training the synthesizer.

**Table 4.1.** Voice Encoder Model Details and Training Steps.

| | VGG2 | VGG3 |
|---|---|---|
| Vector Size | 512 | 256 |
| Num. of Parameters | 1,191,488 | 248,992 |
| Training Steps | 4,400 | 4,200 |

**Table 4.2**. Results for Voice Encoder.

| | VGG2 | VGG3 |
|---|---|---|
| ROC-AUC(CommonVoice) | 0.958 | 0.966 |
| EER(CommonVoice) | 0.114 | 0.097 |
| ROC-AUC(LibriSpeech) | 0.932 | 0.962 |

**Table 4.3.** Final loss values for the models.

| Name | voice loss | t_loss | ewc_loss | loss | test_loss | avg_loss |
|---|---|---|---|---|---|---|
| ewc5_model_0 | 0.9251816273 | 0.3894145191 | 0.0 | 0.3894145191 | 0.389226377 | 0.38932044805 |
| ewc5_model_1 | 0.9240260124 | 0.3965981603 | 0.0 | 0.3965981603 | 0.4066807032 | 0.40163943175 |
| ewc5_model_2 | 0.9342589378 | 0.7036734819 | 0.0 | 0.7036734819 | 0.7822099924 | 0.74294173715 |
| ewc5_model_3 | 0.927788794 | 0.3899408281 | 0.0004811706021 | 0.3904219866 | 0.401121825 | 0.3957719058 |
| ewc5_model_4 | 0.9271577597 | 0.4007889032 | 0.0004876966123 | 0.4012765884 | 0.4015895128 | 0.4014330506 |
| ewc5_model_5 | 0.9350728989 | 0.6928675175 | 0.0005041319528 | 0.6933716536 | 0.781719923 | 0.7375457883000001 |
| ewc5_model_6 | 0.9258553386 | 0.3874357343 | 0.0002459207026 | 0.387681663 | 0.3935800791 | 0.39063087105 |
| ewc5_model_7 | 0.9318072796 | 0.3960293829 | 0.0002503926517 | 0.3962797821 | 0.4003247619 | 0.398302272 |
| ewc5_model_8 | 0.933024168 | 0.7024765015 | 0.0002523389703 | 0.7027288675 | 0.7769488096 | 0.73983883855 |
| ewc5_model_9 | 0.4325576723 | 0.6486860514 | 0.0 | 1.081243753 | 1.081083775 | 1.081163764 |
| ewc5_model_10 | 0.4557353854 | 0.6349618435 | 0.0 | 1.090697289 | 1.083834171 | 1.08726573 |
| ewc5_model_11 | 0.7048860788 | 0.8205808401 | 0.0 | 1.525466919 | 1.623530626 | 1.5744987725000001 |
| ewc5_model_12 | 0.4337844253 | 0.6580614448 | 0.0005102124996 | 1.092356086 | 1.082912922 | 1.087634504 |
| ewc5_model_13 | 0.4216244221 | 0.6527506709 | 0.000534763094 | 1.074909925 | 1.088676453 | 1.0817931889999999 |
| ewc5_model_14 | 0.7034144402 | 0.8089872003 | 0.000509920239 | 1.512911558 | 1.620074272 | 1.566492915 |
| ewc5_model_15 | 0.4396421909 | 0.6510578394 | 0.0002597050625 | 1.090959787 | 1.082683444 | 1.0868216155 |
| ewc5_model_16 | 0.4277828634 | 0.649336338 | 0.0002673994459 | 1.077386618 | 1.103803635 | 1.0905951265 |
| ewc5_model_17 | 0.7056950927 | 0.8266615868 | 0.0002552412043 | 1.532611966 | 1.621532083 | 1.5770720245 |
| ewc5_model_18 | 0.9233779907 | 0.3881410062 | 0.0 | 0.3881410062 | 0.401597172 | 0.3948690891 |
| ewc5_model_19 | 0.9275838733 | 0.398971498 | 0.0 | 0.398971498 | 0.3959047198 | 0.3974381089 |
| ewc5_model_20 | 0.943467319 | 0.7239243984 | 0.0 | 0.7239243984 | 0.7760611176 | 0.749992758 |
| ewc5_model_21 | 0.9286417961 | 0.397675693 | 0.0004808585509 | 0.3981565535 | 0.3903647065 | 0.39426063 |
| ewc5_model_22 | 0.9268534184 | 0.4002259374 | 0.0004887943505 | 0.4007147253 | 0.4035470188 | 0.40213087205 |
| ewc5_model_23 | 0.9415733814 | 0.7005708218 | 0.0005036513321 | 0.701074481 | 0.7713308334 | 0.7362026572 |
| ewc5_model_24 | 0.9295777082 | 0.3877193928 | 0.0002466285077 | 0.387966007 | 0.3895674646 | 0.3887667358 |
| ewc5_model_25 | 0.9258705974 | 0.397264719 | 0.0002502857824 | 0.3975149989 | 0.4024662971 | 0.399990648 |
| ewc5_model_26 | 0.9202346206 | 0.7011007667 | 0.0002523681032 | 0.7013531327 | 0.7754003406 | 0.73837673665 |
| ewc5_model_27 | 0.4621998668 | 0.6411823034 | 0.0 | 1.103382111 | 1.076452136 | 1.0899171234999998 |
| ewc5_model_28 | 0.4438171089 | 0.6448031664 | 0.0 | 1.088620305 | 1.088903189 | 1.088761747 |
| ewc5_model_29 | 0.7013915181 | 0.8133755922 | 0.0 | 1.51476717 | 1.61430788 | 1.564537525 |
| ewc5_model_30 | 0.4476340711 | 0.5835078359 | 0.0005128887715 | 1.031654716 | 1.075819135 | 1.0537369255 |
| ewc5_model_31 | 0.4568097889 | 0.5801846981 | 0.0005215330166 | 1.037515998 | 1.085932136 | 1.061724067 |
| ewc5_model_32 | 0.7198534012 | 0.869899869 | 0.0005095771048 | 1.59026289 | 1.620098948 | 1.605180919 |
| ewc5_model_33 | 0.4309724569 | 0.6005493402 | 0.000260460045 | 1.031782269 | 1.101672888 | 1.0667275785 |
| ewc5_model_34 | 0.4414139688 | 0.6014453173 | 0.0002629851806 | 1.043122292 | 1.096844077 | 1.0699831845 |
| ewc5_model_35 | 0.7102069855 | 0.8701492548 | 0.0002556196123 | 1.580611825 | 1.620160818 | 1.6003863215 |

### 4.2 Finetuning Tacotron

Table 4.3 shows the resulting losses after training the models for 38,000 steps. The average loss (*avg_loss*) is the average between the training loss and the test loss. The results were arranged based on the technique used. In Figures

4.1-4.4, models were grouped (same color) based on the variable hyperparameter. Dots or lines of the same color meant that there was a single variable in question and all else were constant. For 3 values in a search space, like in Freeze Layers, and EWC lambda, there would be 36/3, 12 different lines. For 2 values like in Gradient Clipping, Voice Loss there would be 36/2, 18 different colored lines.

### *4.2.1 Gradient Clipping*

Gradient Clipping using threshold values of 0.5, and 1.0 bore no significant effect on the loss as shown by the flat line in Figure 4.1. This suggested that the gradients were smaller than the threshold values. A cause for this was that the pre-trained model had already been trained for 180k steps in which its gradients had become small since gradient descent decreases gradients over time. In hindsight, smaller threshold values should have been used.

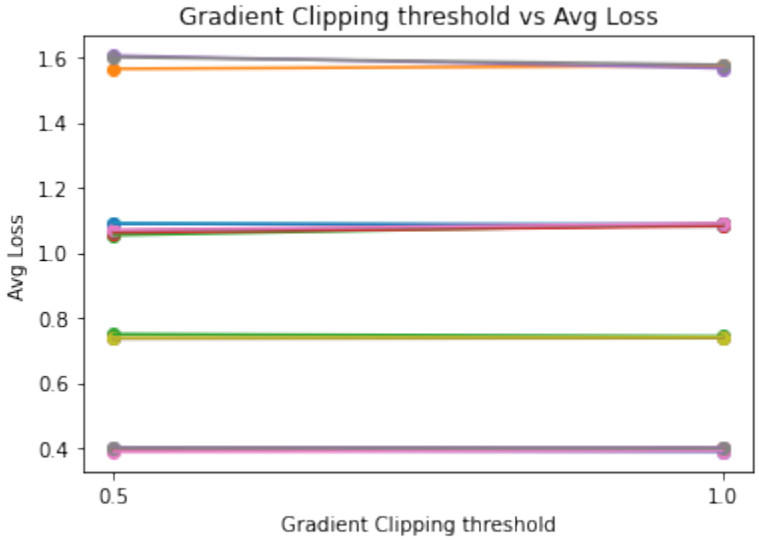


**Figure 4.1.** Results of Gradient Clipping thresholds on Average Loss.

### *4.2.2 Layer Freezing*

In Figure 4.2, it can be observed that there is no significant difference between No Layers Frozen and having only the encoder layer frozen. This could suggest that the weights of the encoder had converged. This occurrence also supported the observations of Brock et al. [2017] in which earlier layers would converge to simple configurations. It also can be observed that freezing the decoder layer has hugely increased loss. This suggested that the weights of the decoder should be changed to adapt to the new data. The huge loss was attributed to the model being unable to fit the new data.

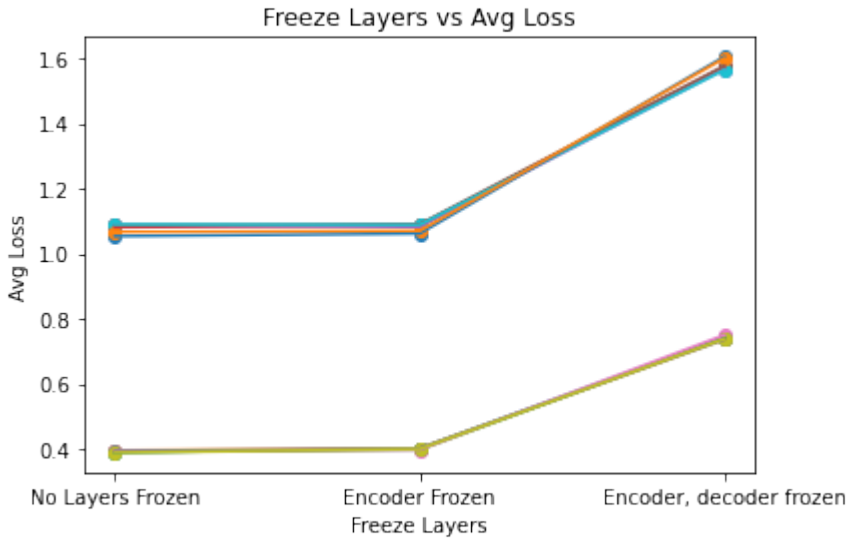


**Figure 4.2.** Results of Layer Freezing on Average Loss.

### *4.2.3 Elastic Weight Conditioning*

In Figure 4.3, it can be observed that the choices of 0.0, 2.0, and 4.0 for lambda did not affect the model loss. This suggested that the choice of hyperparameter values was not suitable for the already pre-trained model and the new task.

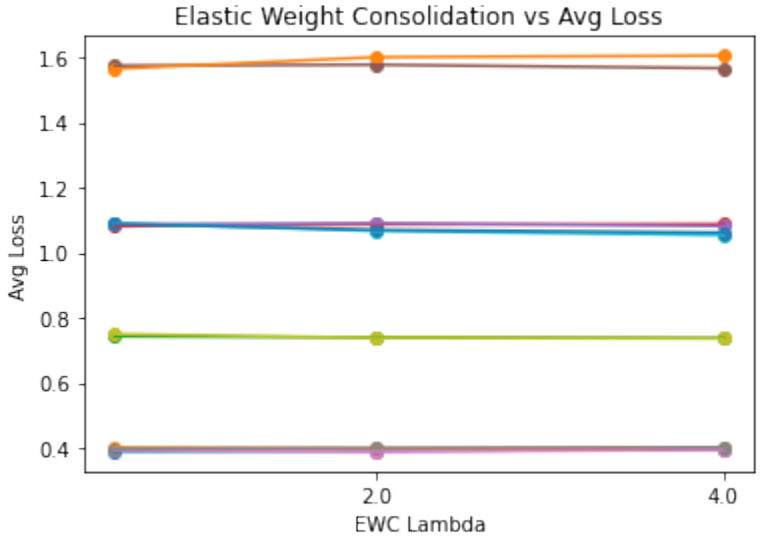


**Figure 4.3.** Results of EWC lambda vs Avg loss.

### *4.2.4 Voice Loss*

In Figure 4.4, it can be seen that the use of voice loss strongly affected the synthesizer (Tacotron) loss, though the voice loss was not directly added to the synthesizer loss. In Figure 4.5, it can be observed that when not incorporating voice loss, the synthesizer loss (*t_loss*) was able to reach a relatively low loss of roughly 0.4. In Figure 4.6, it can be observed that the model was optimizing for voice loss evident to the perfectly decreasing voice loss curve, but the synthesizer loss suffered as it seemed to have increased variance and seemed to have flattened at 0.6, way above the 0.4 baseline.

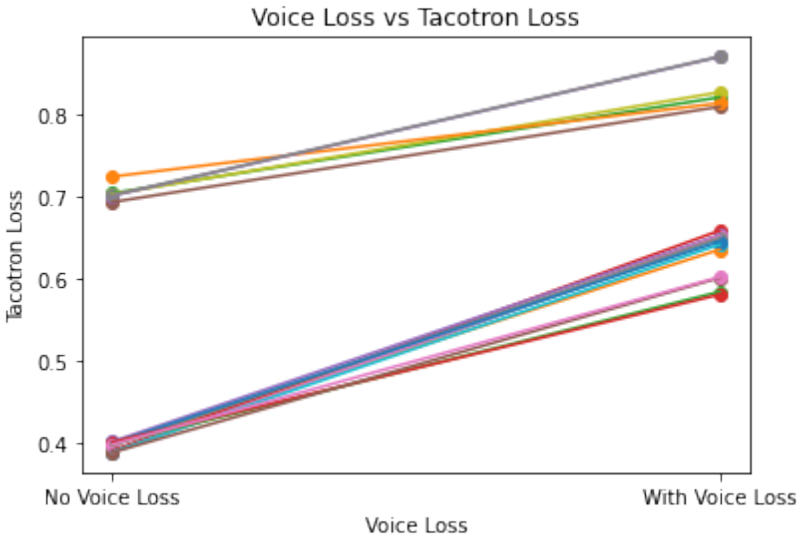


**Figure 4.4.** Results of Voice Loss vs Tacotron Loss.


**Figure 4.5.** Loss curves for baseline model.

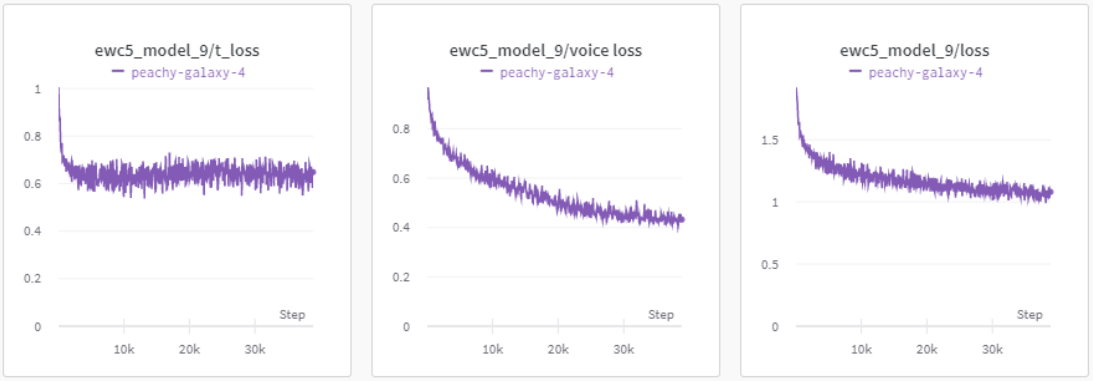


**Figure 4.6.** Loss curves for model incorporating Voice Loss.

### *4.2.5 Inspection of Loss curves*

The models could be subdivided into four groups based on their incorporation of voice loss and frozen layers. This grouping was also affected by loss, evident in Figures 4.1 and 4.3. Looking at Figures 4.7 - 4.10, a couple things can be observed. First is that the freezing encoder and decoder induced high variance in the loss. And that in models that did not optimize for voice loss, they still managed to somehow bring it down to some level, though still with high variance.

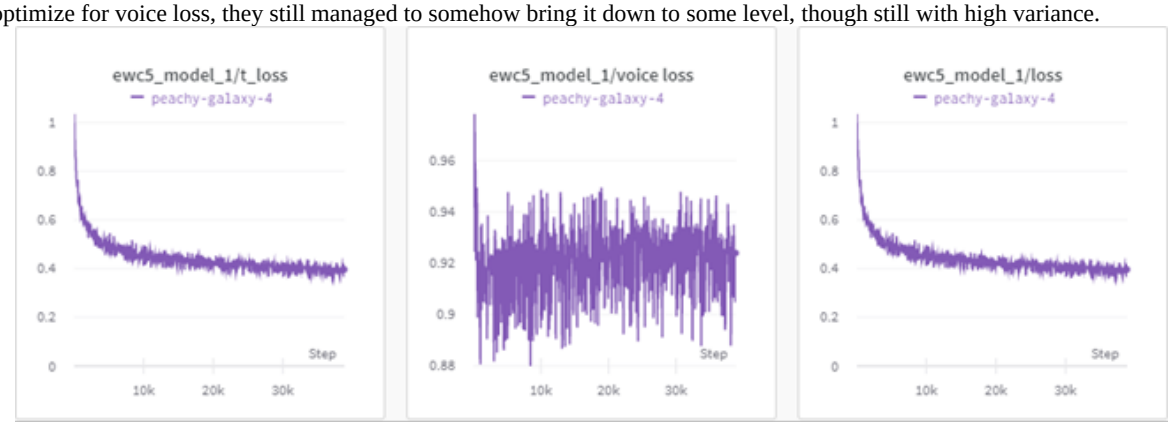


**Figure 4.7.** Model 1 Loss curves.

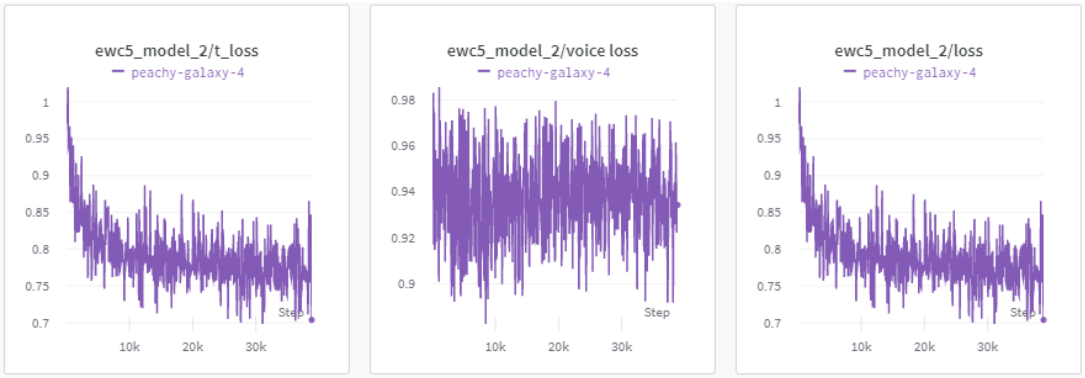


**Figure 4.8.** Model 2 Loss curves.

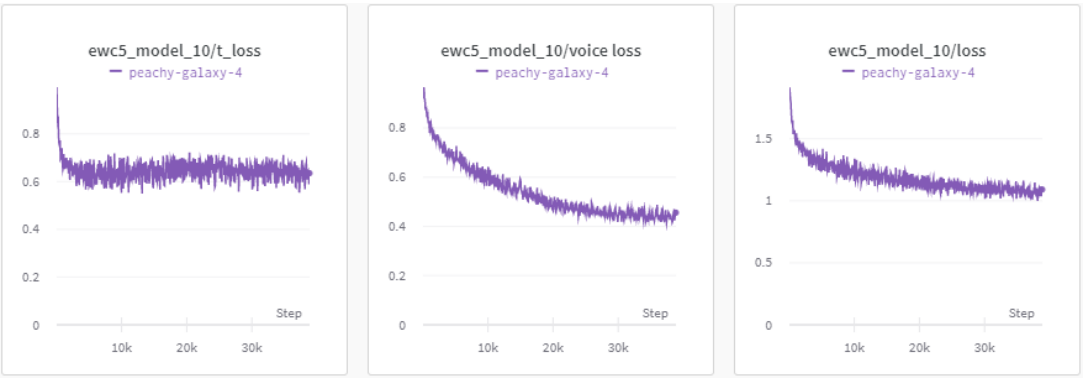


**Figure 4.9.** Model 10 Loss curves.

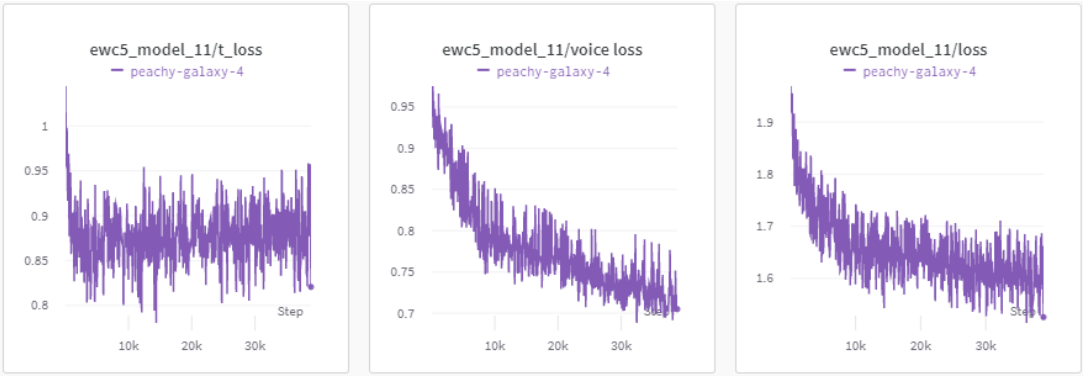


**Figure 4.10.** Model 11 Loss curves.

### *4.2.6 Selection and evaluation of the best model*

The model with the lowest average loss after 30,000 steps was found to be model 24. This model was then trained for another 190,000 steps. This model was evaluated in terms of voice similarity and speech naturalness through human evaluation. Figure 4.11 shows the distribution of the respondents in terms of whether they wore headphones for the

survey. Figures 4.12 - 4.13 shows the results for the first questions in the survey for each evaluation metric. Overall, the model attained a mean opinion score of 3.64 ± 0.475 on speaker similarity, and 3.36 ± 0.379 on naturalness as detailed in Table 4.4.

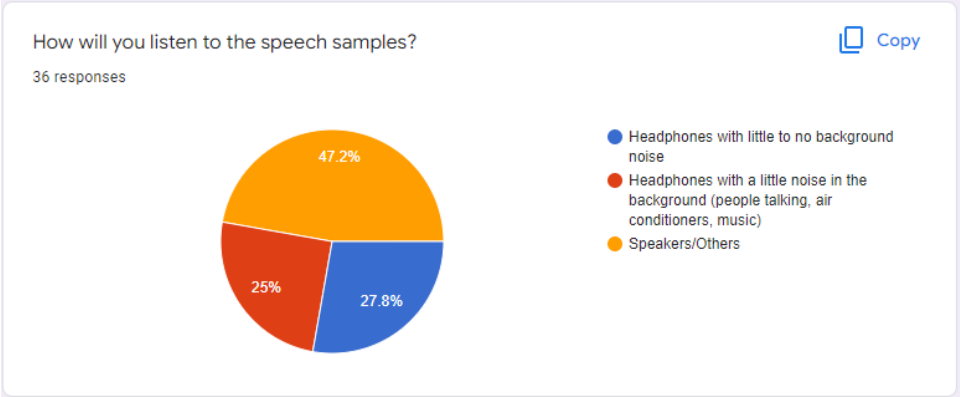


**Figure 4.11.** Survey result for how respondents listened.

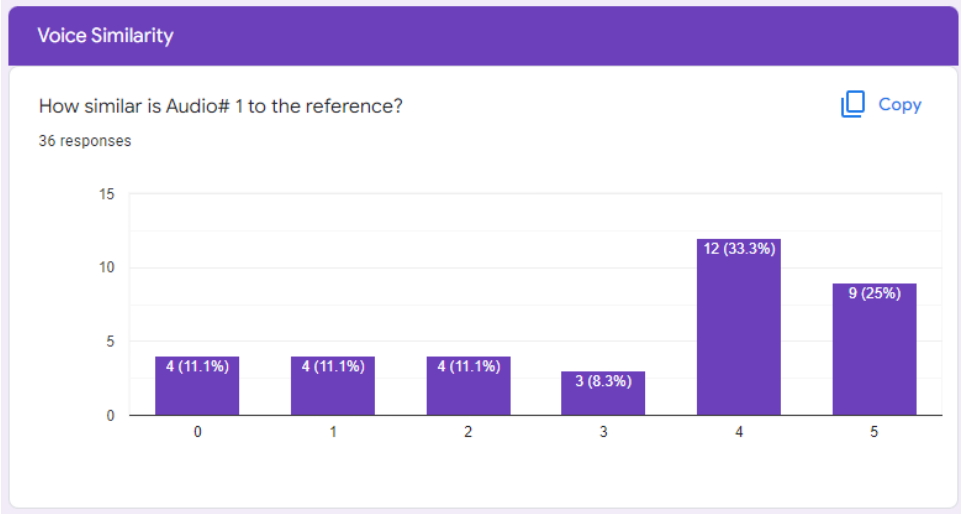


**Figure 4.12.** Survey sample results for Voice Similarity.

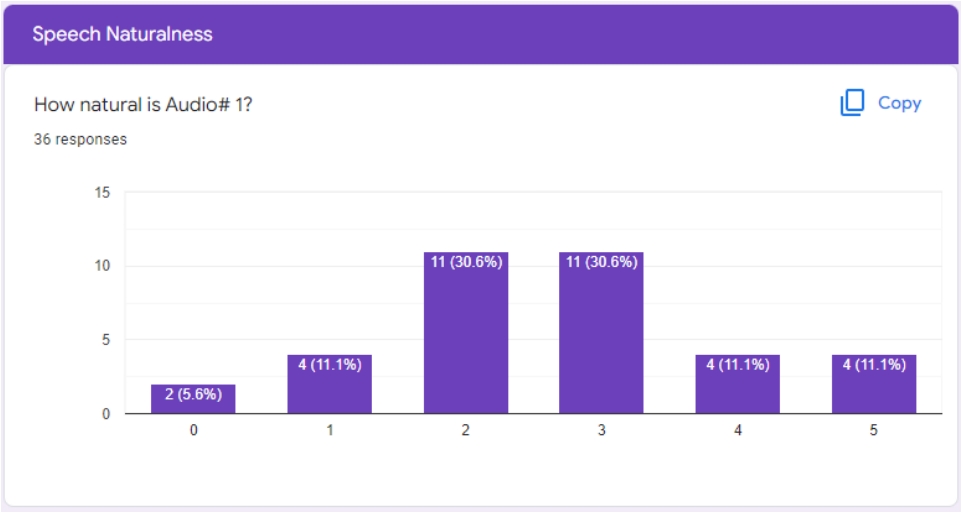


**Figure 4.13.** Survey sample results for Speech Naturalness.

## 5 DISCUSSIONS

In this study, the effectiveness of few-shot learning techniques and the incorporation of a voice loss was evaluated. The results revealed that the hyperparameter value choices for the few-shot learning techniques gradient clipping and elastic weight consolidation was not ideal as they had been shown to not have affected the performance. The voice loss ended up worsening the performance and rendering the model unable to fit and minimize the Tacotron loss, though the voice loss was minimized to some extent. The model with voice loss incorporated ended up not being able to synthesize speech. Further investigation must be performed to find the reason why the voice loss has worsened the performance. A few possible reasons were pinpointed. First was that the selection of particularly hard triplets for the triplet dataset had not been done, though it was stressed by Schroff et al. [2015] in their study. Triplet selection was not done because of the amount of labor selecting triplets from millions of audio files would require, and that it was hoped that the sheer number of triplets used would already include relatively harder triplets. Another possible reason was that encoding an any-length speech recording to a fixed-length voice vector entailed some level of lossy compression. This lossy compression made it more difficult to find a mapping between recordings to voice embeddings. Multiple reductions in the creation of the voice vector could also be a reason as it would have made backpropagation more difficult. The reductions performed were averaging across window slices and speech samples. This made the problem harder to fit and thus made backpropagation more challenging as gradients would need to travel through these reductions.

The best models in terms of average loss were models 24, 0, and 6 with the common denominator being no Voice Loss and no layers frozen. Model 24 was chosen as the representative and was trained indefinitely but terminated at about 190,000 steps. Sample outputs were generated, and the URLs could be found in the Appendix D. The speech samples exhibited a noisy background and a sound of a spacebar at the end. These were signs of overfitting because the original Tacotron had no background noise, and somehow the model learned to synthesize such noise imitating the data. The sound of a spacebar was generated because all the data examples had a sound of a spacebar at the end, marking the end of the recording, and since no noise reduction or post-processing was done after recording the target speech samples. The best model was evaluated and attained a mean opinion score of 3.64 ± 0.475 on speaker similarity, and 3.36 ± 0.379 on naturalness as detailed in Table 4.3.

## 6 CONCLUSION

In this study, a novel method of voiced TTS was attempted through few-shot learning techniques and backpropagating through a voice loss. The results of the study showed that the hyperparameter value choices for the few-shot learning techniques gradient clipping and elastic weight consolidation was not ideal as they had been shown to not have affected the performance. The results also showed that the freezing of only the encoder layers of Tacotron did not affect performance, but freezing both the encoder and decoder layers hugely affected performance. This observation of freezing early layers having little effect on performance supported the observations of Brock et al [2017]. The novel method of backpropagating through the voice loss had led to negatively affecting the synthesizer, and thus overall model causing it to have higher loss and unable to optimize any further. Initial suspected factors were identified, but further investigation must be done for confirmation. Finally, the best model was able to generate natural and similar voiced speech with a similarity score of $3.64 \pm 0.475$ and a naturalness score of $3.36 \pm 0.379$. The generated speech samples, however, still had some pronunciation errors and artifacts such as background noise and a spacebar sound at the end.

### 6.1 Recommendations

Future work would entail further investigating the voice loss and the selection of more appropriate hyperparameter values.

## REFERENCES


[1] Brock, A., Lim, T., Ritchie, J. M., & Weston, N. (2017). FreezeOut: Accelerate Training by Progressively Freezing Layers (arXiv:1706.04983). arXiv. http://arxiv.org/abs/1706.04983

[2] Classification: ROC Curve and AUC | Machine Learning Crash Course. (n.d.). Google Developers. Retrieved June 21, 2022, from https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

[3] Goodfellow, I., Bengio, Y., & Courville, A. (n.d.). Deep Learning Book. MIT Press. Retrieved June 21, 2022, from https://www.deeplearningbook.org/contents/rnn.html

[4] Jia, Y., Zhang, Y., Weiss, R. J., Wang, Q., Shen, J., Ren, F., Chen, Z., Nguyen, P., Pang, R., Moreno, I. L., & Wu, Y. (2019). Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis (arXiv:1806.04558). arXiv. http://arxiv.org/abs/1806.04558

[5] Kemker, R., McClure, M., Abitino, A., Hayes, T., & Kanan, C. (2017). Measuring Catastrophic Forgetting in Neural Networks (arXiv:1708.02072). arXiv. http://arxiv.org/abs/1708.02072

[6] Khan, U., & Hernando, J. (2020). The UPC Speaker Verification System Submitted to VoxCeleb Speaker Recognition Challenge 2020 (VoxSRC-20) (arXiv:2010.10937). arXiv. http://arxiv.org/abs/2010.10937

[7] Mathiasen, A., & Hvilshøj, F. (2021). Backpropagating through Fr\'echet Inception Distance (arXiv:2009.14075). arXiv. http://arxiv.org/abs/2009.14075

[8] McCarthy, O. (2022). WaveRNN [Python]. https://github.com/fatchord/WaveRNN (Original work published 2018)

[9] Mozilla Common Voice. (n.d.). Retrieved June 21, 2022, from https://commonvoice.mozilla.org/

[10] Ren, Y., Ruan, Y., Tan, X., Qin, T., Zhao, S., Zhao, Z., & Liu, T.-Y. (2019). FastSpeech: Fast, Robust and Controllable Text to Speech (arXiv:1905.09263). arXiv. https://doi.org/10.48550/arXiv.1905.09263

[11] Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 815–823. https://doi.org/10.1109/CVPR.2015.7298682

[12] Shen, J. (2020, September 28). ROC(Receiver operating characteristic) and EER (Equal Error Rate). Medium. https://jimmy-shen.medium.com/roc-receiver-operating-characteristic-and-eer-equal-error-rate-ac5a576fae38

[13] Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., Skerry-Ryan, R. J., Saurous, R. A., Agiomyrgiannakis, Y., & Wu, Y. (2018). Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions (arXiv:1712.05884). arXiv. http://arxiv.org/abs/1712.05884

[14] Wang, Y., Skerry-Ryan, R. J., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., Le, Q., Agiomyrgiannakis, Y., Clark, R., & Saurous, R. A. (2017). Tacotron: Towards End-to-End Speech Synthesis (arXiv:1703.10135). arXiv. http://arxiv.org/abs/1703.10135