Extending Text To Speech to Arbitrary Voices

Ian McAulay
imcaulay@berkeley

Vinay Parakala
vparakala@berkeley.edu

1 Background

1.1 Problem Statement

Voice conversion (VC) is the task of modifying speech from a first speaker (the source speaker) to sound as if it were spoken by a second speaker (the target speaker). This is a similar idea to artistic style transfer and Deepfake, but applied to speech instead of images. Speech data is composed of linguistic content and speaker identity content. Linguistic content includes the actual words and meaning spoken, while speaker identity is about the attributes of the speaker, such as pitch and style. Therefore voice conversion requires preserving the linguistic content of the source speaker (so the meaning does not change), and the speaker identity content of the target speaker (so the result sounds like the target speaker).

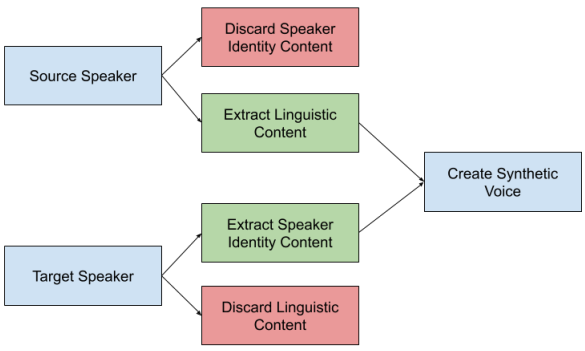

Figure 1: Voice Conversion of Speech Data

1.2 Voice-to-Voice vs Text-to-Speech

Voice conversion can be treated either as a voice-to-voice or text-to-speech (TTS) problem. With voice-to-voice, the input is two audio samples: one is the source speech content, and one is a sample of speech from the target speaker. With TTS, a sample of target speaker speech is still given, but the source input comes from text rather than audio. In both cases, the goal is to output audio that sounds like the source content spoken by the target speaker. A TTS approach is likely more useful and generalizable since in a realistic setting having text would probably be more common than audio. However, a voice-to-voice model could easily be applied to text by first passing the text through a standard TTS system to create source audio. Therefore, we experimented with both voice-to-voice and TTS VC models.

1.3 Evaluation

A voice conversion system can be evaluated on its intelligibility and quality of imitation. If the model effectively preserves the linguistic content of the source while ignoring the linguistic content of the target, it should produce synthetic speech that is clearly understandable and contains the exact same words as the source. Similarly, if the speaker identity of the source is discarded while the speaker identity of the target is preserved, the converted speech should sound exactly as if it were spoken by the target speaker.

1.4 Applications

Applications of voice conversion include creating natural-sounding TTS systems, dubbing movies, and speech to speech machine translation (S2SMT).

2 Data

2.1 Parallel vs. Non-parallel

Voice conversion training data can be categorized as either parallel or non-parallel. Parallel means the data contains different speakers saying exactly the same sentence, while non-parallel means the data contains different speakers saying different things. Training with parallel data generally tends

to give better results, but recent models have been able to match or exceed this performance with non-parallel data. In addition, it is difficult to obtain parallel data, while collecting a large amount of non-parallel data is easy, since the different speakers in the data can be saying anything rather than having to say the same thing. We focused on non-parallel data, since it is easy to acquire and is effective at training the models we tested. We mainly used two datasets in our experiments: VCTK and LJ Speech.

2.2 VCTK

The VCTK (Voice Cloning Toolkit) dataset (https://homepages.inf.ed.ac.uk/jyamagis/page3/page58/page58.html) contains audio samples of 109 speakers reading 400 sentences with corresponding text. All speech is in English, with a variety of accents represented. This dataset was collected specifically for voice conversion research, so the audio quality is high with little background noise. This dataset is frequently used in VC experiments, such as (Fang et al., 2018), (Kameoka et al., 2018), and (Lee et al., 2018).

2.3 LJ Speech

The LJ (Linda Johnson) dataset (https://keithito.com/LJ-Speech-Dataset/) contains 13,100 audio clips from a single speaker reading passages from 7 non-fiction books. All speech is in English, from the same speaker (Linda Johnson). The audio is taken from audiobook recordings, so the quality is very good. This dataset was used as a sanity check in our TTS models, to make sure that the voice embedding networks we added did not adversly affect output voice quality.

2.4 Additional Data Sources

It would probably be beneficial to train on noisier data as well to help make the model more robust and useful on real-world audio. Audiobooks, speeches (such as TED talks), and speech from other languages could be good additional data sources.

3 Related Work

In the past there have been non-neural approaches to this problem (Tomoki Toda and Tokuda, 2007), but recently neural architectures have given significant improvements (Chorowski et al., 2017). These include GAN variations which can effectively synthethize audio in general, such as WaveGAN (Chris Donahue, 2019), and GANs specifically designed for voice conversion, such as CycleGANs (Fang et al., 2018) and StarGANs (Kameoka et al., 2018). Voice conversion can also be treated as a TTS problem, such as with Facebook's VoiceLoop (Yaniv Taigman and Nachmani, 2018). The Tacotron TTS system can be modified to perform VC by adding a speaker embedder network, to create synthetic voices to mimic arbitrary target speakers unseen during training time (Lee et al., 2018) and (Ye Jia, 2019). Closely related problems and models include prosody transfer (instead of style transfer) using Tacotron (RJ Skerry-Ryan, 2018), and polyglot TTS to transfer speaker style across different languages (Eliya Nachmani, 2019).

4 Approach

4.1 Voice-to-Voice Baseline Model

The main baseline model we used for the voice-to-voice approach is StarGAN (Kameoka et al., 2018). This is a variation of a GAN which uses cycle-consistency loss, like a CycleGAN (Fang et al., 2018), with a classifier model in addition to the normal generator and discriminator models. The purpose of the classifier is to introduce additional loss terms for the generator to help the generator learn to imitate multiple different speakers. In particular, the domain classification loss of the generator is defined as
$L_{cls}^{G}(G) = -E_{x\sim p(x),c\sim p(c)}[\log p_C(c|G(x,c))]$
where G is the generator, x is the source audio, c is the target speaker, and $p_C(c|G(x,c))$ is the probability the classifier assigns to class c when given the output of G as input. This loss will be lowest when the classifier network assigns a high probability of speaker c to the output audio of the generator network when the generator is trying to imitate speaker c. In other words, the generator is incentivized to produce audio which will be classified as belonging to the target speaker. We trained this architecture on 10 speakers from VCTK to establish baseline audio results.

We ran two open source implementions of StarGANs for VC (Kameoka et al., 2018):
https://github.com/hujinsen/StarGAN-Voice-Conversion
https://github.com/liusongxiang/StarGAN-Voice-Conversion

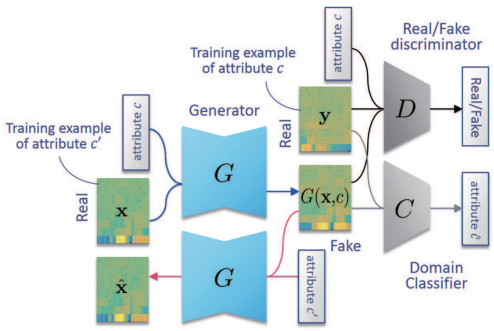

Figure 2: Baseline StarGAN Architecture (Kameoka et al., 2018)

These give us baseline audio samples to compare our results to, and models to build off of and modify.

4.2 Text-to-Speech Baseline Model

In 2017, Google published a paper describing the Tacotron model, where they present a text-to-speech neural network based model that directly learns to synthesize speech from text, capturing linguistic and prosody features (Wang et al., 2017). Given transcribed speech data, this model could generate natural sounding speech. Tacotron converts character sequences to mel-spectrograms which are converted into speech using Griffin-Lim reconstruction. It does using an encoder-decoder model.

The encoder part of the network consists of a CBHG (Convolutional Bank + Highway Network + Bidirectional GRU) module which encodes character embeddings into an internal feature representation used by the decoder.

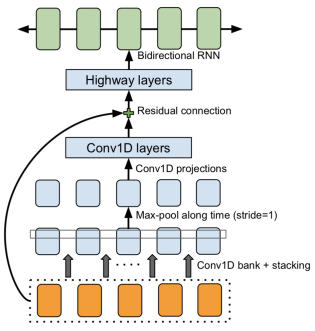

Figure 3: CBHG Module of Tacotron Model

The decoder part of the network converts this internal feature representation to a mel-spectrogram using a content-based attention decoder. A recurrent attention layer produces the attention query at each decoder time step, from the output frames of the decoder (These frames are frames from the output mel-spectrogram). The output of the encoder is passed through an attention layer, which uses the query from the attention RNN in conjugation with the output of the encoder to generate a context vector. This context vector is concatenated with the query from the attention RNN and passed through another recurrent network. The outputs of this are passed through another CBHG module to form a frames of the output mel-spectrogram.

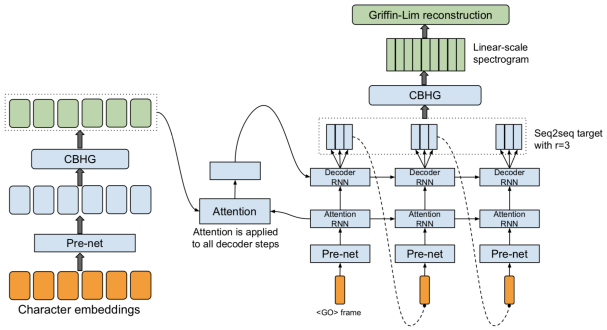

Figure 4: Tacotron Model

The original Tacotron architecture was trained on a single-speaker dataset and therefore could only generate single-speaker output. We can modify this architecture to account for multiple speakers by creating an embedding table which maps speaker IDs to embeddings and concatenates these speaker embeddings to the inputs of the Attention RNN and Decoder RNN parts of the decoder section of the network (Kameoka et al., 2018). We made this modification to the original Tacotron architecture and used it as our baseline, training it on all 109 speakers in VCTK.

Our baseline run was done here: `https://github.com/vparakala/tacotron`. The source code used in the original paper was not open-source, but this is a fork of a good attempt at recreating it. In this fork we added provisions for multiple speakers by adding a speaker embedding table (used as our baseline), and a voice embedding network (used as our experimental group)

4.3 Voice-to-Voice

Voice conversion models are often restricted to only imitating voices they were trained on. For example, the StarGANs model is many-to-many, meaning a single model can transfer the style from multiple voices, but it still relies on a one-hot at-

tribute vector which is limited to imitating target speakers the model was trained on. One significant advantage of the Tacotron + speaker embedder model is its ability to imitate arbitrary voices outside of the training data (i.e. any-to-any). It accomplishes this through a speaker embedder network to create a vectorized speaker representation containing information on the identity of the target speaker. We can apply this idea to StarGANs to replace the one-hot attribute vector with a speaker embedding which can represent arbitrary speakers to perform any-to-any voice conversion.

The generator network in the original StarGANs model contains a downsampling section to convert source audio to a lower dimensional representation. The downsampled result is then concatted with a one hot vector representing the target speaker, passed through bottleneck layers, and upsampled to produce output audio of the same length as the input source audio (Kameoka et al., 2018). We modified this original architecture by passing in target audio to a second downsampling network, and concatting the downsampled target audio representation with the downsampled source audio representation. The result is passed through bottleneck and upsample layers as before to produce output audio.

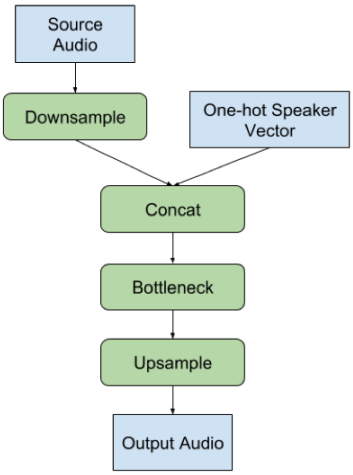

Figure 5: Original StarGAN Generator (many-to-many)

Note that although the source and target downsamplers have very similar structures, they don't share weights, since we want the source downsampler to extract linguistic content information, while the target downsampler should extract speaker style information. For our experiments,

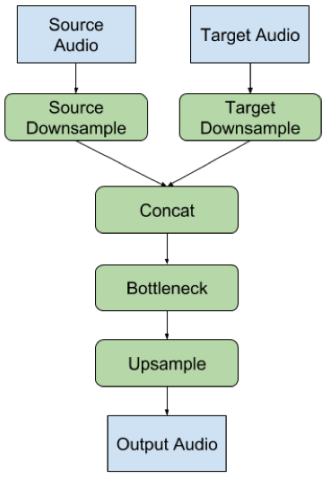

Figure 6: Our Modified StarGAN Generator (any-to-any)

we kept the target downsampler architecture very similar to the source downsampler, though in principal they could be completely different.

One issue we ran into was matching the length of the source audio input, target audio input, and converted audio output. In the original architecture, the generator takes source audio and a one hot target speaker identity vector as input. The generator is divided into three subnetworks: source downsampling, bottleneck, and upsampling. The source downsampling subnetwork of the generator only has convolutions, batch normalization, and gated linear unit layers. Since there are no layers with fixed input size (such as fully connected or RNN layers), the network can take in arbitrary sized source audio input. The bottleneck layers don't change the dimensionality, and the upsampling subnetwork contains deconvolutions that exactly reserve the convolutions from downsampling. Therefore the generator can take in source audio input of any length and output audio of the same length, which is important for voice conversion where the converted audio should match the source. This approach works when the target speaker input is a one hot vector, because the vector can be repeated to match the size of the source audio and concattenated along the channel dimension (i.e. the third dimension). For example, if the source audio input has dimension $H \times W \times 1$, after downsampling the dimension may be $H' \times W' \times C'$. If there are S speakers in the training data, the target speaker one hot vector will be length S. Equivalently, the dimen-

sion of the one hot vector can be thought of as $1 \times 1 \times S$. This vector gets repeated to produce a matrix with dimension $H' \times W' \times S$, so that it can be concattenated with the downsampler output (which has dimension $H' \times W' \times C'$) to create an $H' \times W' \times (C' + S)$ matrix to pass in to the bottleneck layers.

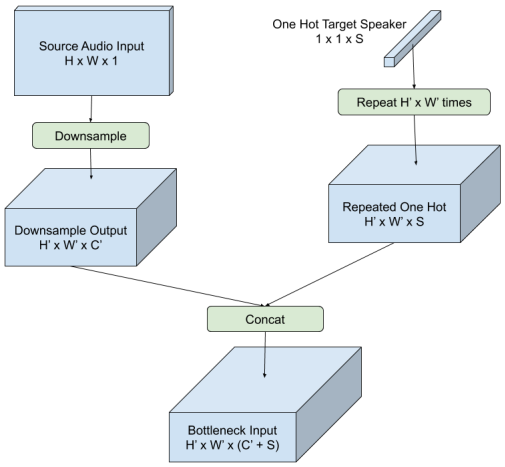

Figure 7: Original StarGAN Source-Target Concatenation

However, this approach needs to be our altered for our modified generator where the one hot target speaker vector is replaced by the target downsampling network. Instead of simply repeating a one hot vector to match the source audio dimensions, we downsample the target audio until it has a width of 1, and then repeat the vector until we get a matrix matching the source data dimensions. Since we determine the height dimension by the number of MFCC coefficients we choose, we can constrain the source audio and target audio height to always match. For example, let the input source audio have dimensions $H \times W_s \times 1$ before downsampling and $H' \times W'_s \times C'_s$ after downsampling. The target audio will always have height H, but may have different width (since the source and target audio samples can be different lengths), so let its dimensions be $H \times W_t \times 1$. We downsample the target audio to get an $H' \times 1 \times C'_t$ matrix, and repeat this matrix W'_s times along the second axis to get a new matrix of dimension $H' \times W'_s \times C'_t$. At this point we can concatenate the source (with dimensions $H' \times W'_s \times C'_s$) with the target (with dimensions $H' \times W'_s \times C'_t$) along the third axis to get an $H' \times W'_s \times (C'_s + C'_t)$ matrix to pass into the botteneck layers.

Intuitively, we downsample the target audio until its width is 1 because this dimension represents time and the length of the audio. By repeating the target speaker across this dimension, we are encoding the fact that the target style is invariant across time. In other words, at each timestep of the source audio, we should imitate the target speaker using the same embedding.

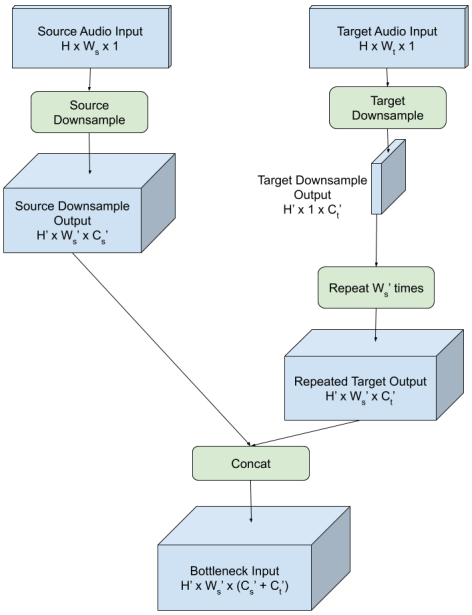

Figure 8: Our Modified StarGAN Source-Target Concatenation

In our case, we set $H = 36, H' = 9, C'_s = 256, W_t = 8192$, and $C'_t = 256$. W_s and W'_s are variable and depend on the length of the input source audio. We set $W_t = 8192$ because this corresponds to about 40 seconds of speech, which seems like a reasonable sample to use to imitate the target speaker. We set $C'_s = C'_t$ for convenience, but these could be different values. It seems likely that the source audio contains more relevant information than the target audio, and we had issues with the model overfitting to speakers from the training set, so we think lowering C'_t to reduce the target speaker vector dimensionality could improve performance.

4.4 Text-to-Speech

Our goal was to extend the multi-speaker Tacotron model we made to one that could imitate any speaker. We did this by replacing the speaker embedding table in the network with a voice embedder that takes the log-mel spectrogram as input and returns a speaker embedding. During training the log mel-spectrogram passed in as input was the

log of the target mel-spectrogram, and in testing it was some arbitrary voice sample. The loss was the same as the original tacotron:

$$Loss = ||Y_{mel} - \hat{Y}_{mel}||_2 + ||Y_{linear} - \hat{Y}_{linear}||_2$$

The linear loss is the $l2 - norm$ of the difference between the output linear scale spectogram and the input linear scale spectogram and the mel loss is the $l2 - norm$ of the difference between the output spectogram rescaled to use the mel scale and the input mel scale spectogram. Both of these losses are weighted equally. The embedding network is trained along with the rest of the model. One approach used here: (Lee et al., 2018) is to pass the log-mel spectogram through a series of convolutional layers. Then max-pool over time and frequency and pass the resulting vector through a dense layer. We added this voice embedder to

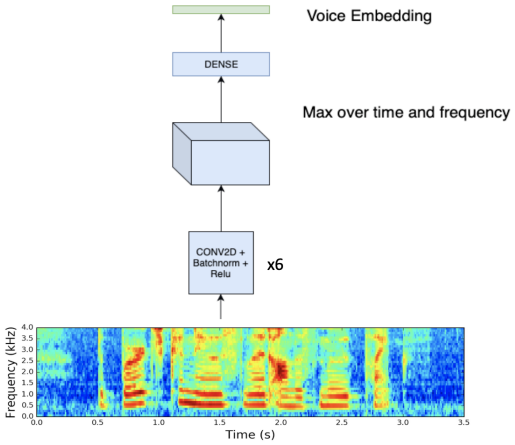

Figure 9: CNN Voice Embedding

the network and passed in the voice embeddings to each step of the recurrent units of the decoder of Tacotron: Each convolutional layer had 2 filters with stride length 2 over the time axis and 1

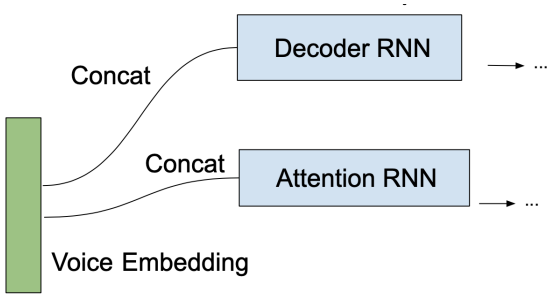

Figure 10: How the voice embeddings are incorporated in the network

over the frequency axis. In our model, the frequency axis was split into 80 bins and the time was capped at 17 seconds (The longest recording in VCTK is 15 seconds, longest recording in LJ Speech is 11 seconds). The size of the voice embedding was 256. Intuitively we thought this approach would be effective at generating voice embeddings because the architecture was unconnected to the length of the recording which we thought should be completely irrelevant in generating a voice embedding. However, when we trained this model on the single-speaker LJ dataset we found output generated speech to be muffled and generally unintelligable.

Another paper (Ye Jia, 2019) suggested using a stack of 3 LSTM layers with hidden unit size 256. The number of recurrent units would be adjusted based on the size of the input. Each frame of

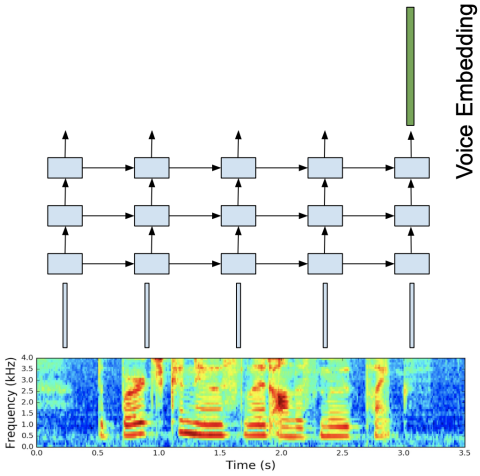

Figure 11: LSTM voice embedding

the log mel-spectrogram passed in as input represents 50 ms of recording. For the longest input, the voice embedding network would have 310 recurrent units. As shown in the figure above, the voice embedding is generated from the output of the last unit of the top layer of the LSTM network. We trained out network using this embedding on the LJ Dataset and found output voices to be as crisp as the original single-speaker tacotron model.

We then used the LSTM voice embedding network in conjugation with Tacotron and trained it on VCTK, a multi-speaker dataset. The network was able to mimic the speaker embedding table in terms of quality, but did not generalize well to speakers that were not in the training set. Perhaps a training dataset with a larger and more varied speaker set could help mitigate this problem. Al-

ternatively, passing in a noisy sample instead of the log of the target mel-spectogram could help regularlize the speaker embedding network, which could make it more robust to speakers that are not in the training data set.

The voices generated from the VCTK dataset in general were not as high quality as those from the LJ Dataset. This is likely because the VCTK dataset often has pauses at the beginnings of its recordings. Some sort of data cleaning could have likely improved performance.

5 Results

5.1 Voice-to-Voice

After implementing our modifications to the original StarGAN architecture to replace the one hot speaker vector with a target speaker downsampling network, we evaluated the new model. While the baseline StarGAN model gives fairly good results, it is restricted to imitating speakers from the training set. As with the baseline, we trained our new model with 10 speakers from VCTK. For target speakers in the training set, the modified model produced good quality audio, comparable to the original StarGAN architecture. However, for target speakers outside the training data, the results were significantly worse. This suggests our modification doesn't prevent the generator from learning to imitate voices, but it does require more work to generalize to arbitrary speakers.

We can visually observe the results with spectrograms of the source, target, and converted audio. Bands of blue represent silence, while red and orange represent high energy. We can see both the source and converted audio have the same patterns of noise for the first 5.5 seconds, with silence at the end. This suggests the model is correctly matching the linguistic content and words of the source audio. Likewise, we can see how the source has many dense regions of high energy, but both the target and converted audio have significantly less. This is what we expect since the style of the converted should be the same as the style of the target, so it seems like in general the model is capturing at least some of the target style.

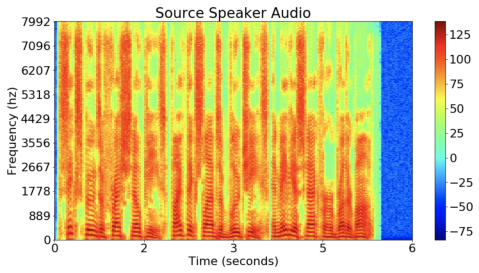

Figure 12: Source Audio Spectrogram

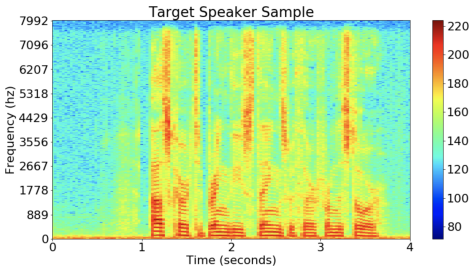

Figure 13: Target Audio Spectrogram

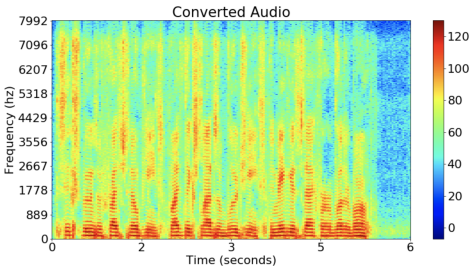

Figure 14: Converted Audio Spectrogram

5.2 Text-to-Speech

Measuring quality of audio recordings is a difficult task. For the StarGANs approach, we showed results by visually comparing spectograms from target and generated audio samples. For the TTS approach, we wanted to compare output audio samples between the multi-speaker Tacotron which used a speaker embedding table to generate voice embeddings, and our voice embedding network model, which uses a 3 layer LSTM network to generate voice embeddings from sample audio. To compare speech quality, we used an automatic speech to text generator (https://speech-to-text-demo.ng.bluemix.net), to transcribe output audio and compared the accuracy of the transcriptions between the multi-speaker Tacotron and the voice embedding network Tacotron. We ran the 100

generated sentences from each model through the speech to text generator, and compared transcription accuracy. Results are shown below:

TTS Transciption Results	
Mode Type	Transcription Accuracy
Embedding Table	91.3%
Embedding Network	86.5%

The embedding table accuracy is better, but voice embeddings from the LSTM embedder generate similar quality output. This measure of accuracy only concerns the clarity of the output, and not how well it stylistically matches the sample voice. Empirically, both methods do a similarly good job of doing this. If we had more time, we would have asked people in a survey what they thought of the clarity and style of our generated voice samples. This is what papers we referenced did.

6 Tools

For voice-to-voice experiments with StarGANs, we used pytorch and worked off an open source implementation of the original paper: https://github.com/liusongxiang/StarGAN-Voice-Conversion

Since voice-to-voice VC requires audio files, we used Google Cloud Text-to-Speech to create speech samples of arbitrary text. Using a generic TTS system like Google Cloud's allows us to create audio files so we can still use a voice-to-voice system on text input.

For text-to-speech, we used Tensorflow and worked off a widely cited attempt at mimicking the architecture described in the original Tacotron paper https://github.com/liusongxiang/StarGAN-Voice-Conversion. Our fork (https://github.com/vparakala/tacotron) uses this as a starting point. In our fork we add functionality for multi-speaker and "any-speaker" TTS, in which any-speaker uses our voice embedding network to generate voice embeddings from arbitrary speakers.

For audio processing, we used sox, ffmpeg, and librosa.

7 Lessons Learned

Almost all of the major roadblocks we faced during this project had to do with set up. Finding a server that could run our models was difficult, especially when we wanted to use GPUs to improve performance because the repositories we forked from had very specific requirements, and it was difficult to match these requirements on the servers. We expected GPU usage to improve performance considerably, but it only improved by about 1.5x. In future, we will spend more time planning and researching what servers and what computers we want to run our models on, and consider this when we investigate what libraries and versions of tensorflow/pytorch we use in our code.

8 Team Contributions

Ian McAulay - 50%
Vinay Parakala - 50%

Both worked on some aspects like data collection. Ian focused on the voice-to-voice approach and Vinay focused on the text-to-speech approach.

References

Jan Chorowski, Ron J. Weiss, Rif A. Saurous, and Samy Bengio. 2017. On using backpropagation for speech texture generation and voice conversion.

Miller Puckette Chris Donahue, Julian McAuley. 2019. Adversarial audio synthesis.

Lior Wolf Eliya Nachmani. 2019. Unsupervised polyglot text-to-speech.

Fuming Fang, Junichi Yamagishi, Isao Echizen, and Jaime Lorenzo-Trueba. 2018. High-quality nonparallel voice conversion based on cycle-consistent adversarial network.

Hirokazu Kameoka, Takuhiro Kaneko, Kou Tanaka, and Nobukatsu Hojo. 2018. Stargan-vc: Non-parallel many-to-many voice conversion with star generative adversarial networks.

Younggun Lee, Taesu Kim, and Soo-Young Lee. 2018. Voice imitating text-to-speech neural networks.

Ying Xiao Yuxuan Wang Daisy Stanton Joel Shor Ron J. Weiss Rob Clark Rif A.Saurous RJ Skerry-Ryan, Eric Battenberg. 2018. Towards end-to-end prosody transfer for expressive speech synthesis with tacotron.

Alan W. Black Tomoki Toda and Keiichi Tokuda. 2007. *IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, 15(8):2222–2235.

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. 2017. Tacotron: Towards end-to-end speech synthesis.

Adam Polyak Yaniv Taigman, Lior Wolf and Eliya Nachmani. 2018. Voiceloop: Voice fitting and synthesis via a phonological loop.

Ron J. Weiss Quan Wang Jonathan Shen Fei Ren Zhifeng Chen Patrick Nguyen Ruoming Pang Ignacio Lopez Moreno Yonghui Wu Ye Jia, Yu Zhang. 2019. Transfer learning from speaker verification to multispeaker text-to-speech synthesis.