Image Credit: https://miro.medium.com/max/1200/1*nC8NtGCK-t0t89VrnDj4cA.png

Voice Transfer Analysis

Author: Sudharsan Gopalakrishnan

Acknowledgment: Mentor: Dr. Ganesh Mani, Carnegie Mellon University

Table Of Contents:

Introduction to Voice Transfer

Style transfer usually refers to extracting the style of an entity and transferring it to another object. Voice Transfer, also called Voice cloning, is a type of style transfer that involves voice style. More specifically, it is a process that involves using a recording of a person's voice, extracting his or her style of speaking, and accurately synthesizing a new recording comprising that same style but for something the person did not previously speak. With Voice Transfer, one could incorporate a recreated voice of former President Barack Obama as one's GPS guide or Elon Musk's voice as one's Amazon Alexa voice assistant.

Why Voice Transfer?

During the summer of 2021, one of my parents' best friends tragically passed away due to COVID-19, and we all wanted to console his family. As I kept thinking about ways to support them, I was able to conceive the idea of Voice Transfer to "bring back loved ones". In other words, I want my parents' friend's family to be able to listen to him again, though not in person, should they wish. Many others lost their lives due to COVID-19. Even a small amount of comfort could be vital for those who remain.

Research Question

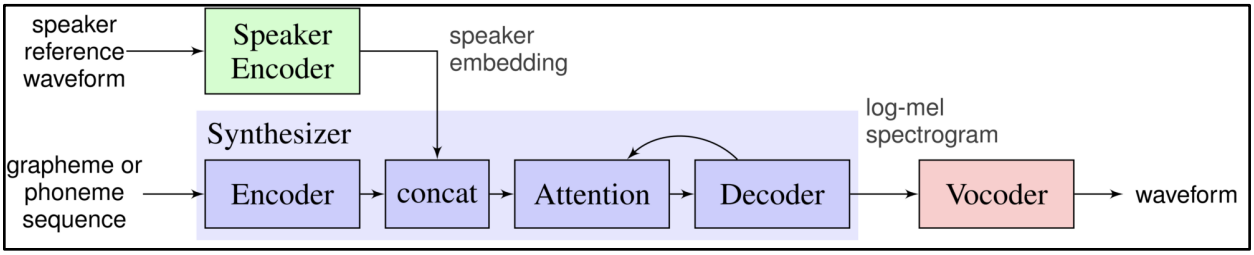
Using Natural Language Processing (NLP), ideally what kind of algorithm would most accurately carry out the process of Voice Transfer, and how can I devise an architecture that would fill in the gaps of the synthesized recording, using existing algorithms and programming libraries?

Hypothesis

An encoder-vocoder dual system can be utilized to generate a waveform from a speech recording that can be applied to any text.

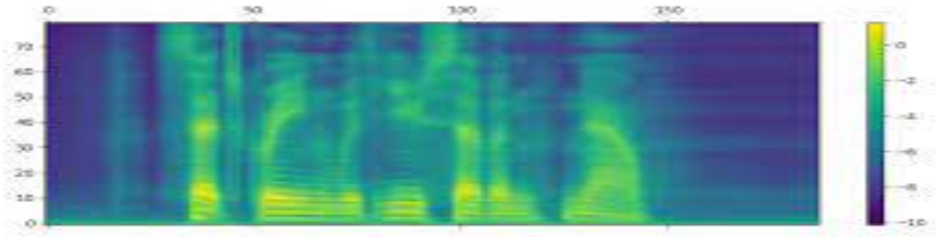
Methods

I chose Python as the programming language for this project because of the rich availability of Machine Learning libraries in that language. To perform Voice Transfer in Python, one needs to use a Neural Network, a Deep Learning algorithm. The algorithm must be able to recognize the style of the recorded sound with reasonably good accuracy that can be improved over time with good training data.



→ Jemine's SV2TTS diagram[1]

For the purposes of this research, I used Resemble AI Machine Learning engineer Corentin Jemine's open-source GitHub called Real-Time-Voice-Cloning, which is an implementation of a Voice Transfer algorithm known as SV2TTS. It comprises a speaker encoder, a synthesizer, and a vocoder. A speaker encoder is a network that takes in speech as input for it to be processed and then outputs an embedding to capture the sound of the speaker but does not take into account what the speaker actually says. The speaker encoder uses factors such as the pitch, accent, and the tone of the sound coming from the speaker to create a speaker embedding, which represents the speaker's identity. The synthesizer takes text as input, which is mapped to phonemes, the smallest unit of human sound. Using a recurrent network, it then creates mel spectrograms, a type of data visualization that displays a sound's frequency versus its time duration. In this case, Jemine used the *Tacotron 2*[2] framework in order to create these spectrograms.

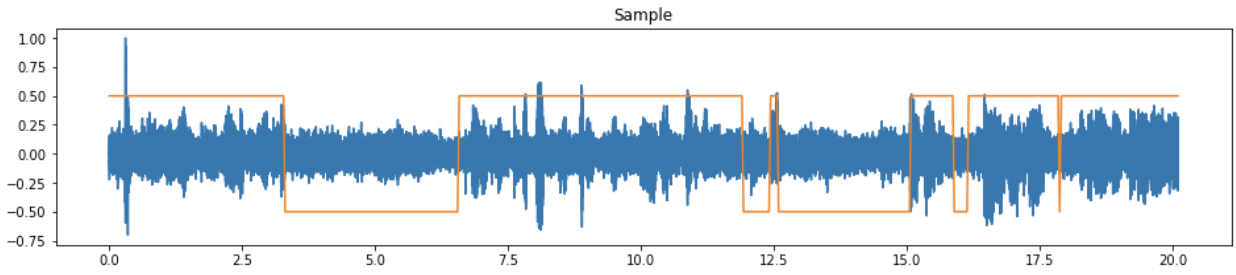


[1] Jemine' s paper: https://matheo.uliege.be/bitstream/2268.2/6801/5/s123578Jemine2019.pdf

[2] Tacotron 2 Documentation: https://ai.googleblog.com/2017/12/tacotron-2-generating-human-like-speech.html

→ Example of a mel spectrogram generated using Matplotlib[3]

A synthesizer must be trained in order to perform as well as possible, so once the speaker embedding is created, the decoder of the synthesizer generates another mel spectrogram again recurrently. The generated spectrogram is compared to the original target to generate a loss which is then optimized for the model to perform more accurately. Lastly, the vocoder turns the generated mel spectrogram into raw audio that can be listened to as spectrograms do not allow the sound to be heard. Jemine's implementation of his vocoder is based on that of *WaveNet*[4], a modern Deep Learning model that generates raw audio waveforms. Another thing that should be completed for voice cloning is called Voice Activity Detection or the process of disregarding the unvoiced or silent parts of all the input recordings as they can have an influence on the partially sampled utterances of the original recordings. To do this in Python, Jemine used the webrtcvad package[5] which would use a binary flag to detect the voiced and unvoiced parts of the recordings.



[3] https://www2.informatik.uni-hamburg.de/wtm/publications/2019/QWW19/LipSound-%20Neural%20Mel-spectrogram%20Reconstruction%20for%20Lip%20Reading-for-WTM-web.pdf

[4] WaveNet documentation: https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

[5] webrtcvad package: https://github.com/wiseman/py-webrtcvad

→ Example of a binary flag on a waveform graph[6]

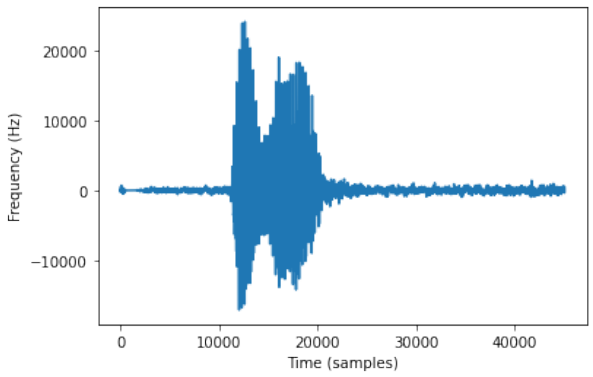
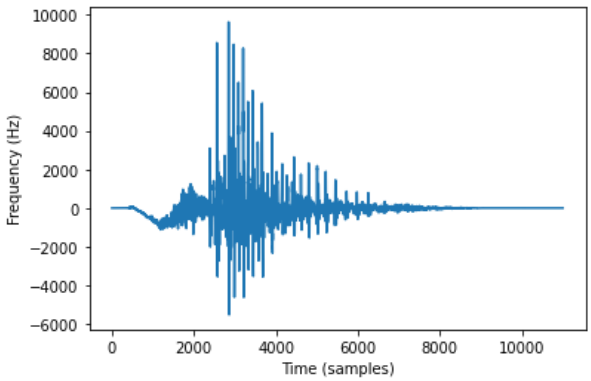
Data

To see how accurate the SV2TTS encoder-vocoder system is, I tested the model with my own recordings, which I used to synthesize new voices. I created waveforms of both my original recordings as well as the synthesized recordings and grouped them in sets of two for a side-by-side comparison. In terms of the content of the recordings, I limited them to one-word recordings as it is much easier to compare them.

Notes: The **x-axis of each graph below represents the time, in samples**, and the **y-axis of each graph below represents the frequency of the recorded and synthesized recordings, in Hertz**.

[6] Image from the github to the right: https://github.com/nicklashansen/voice-activity-detection

Word → Hello

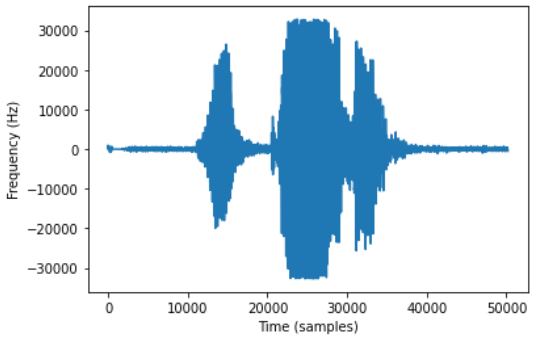
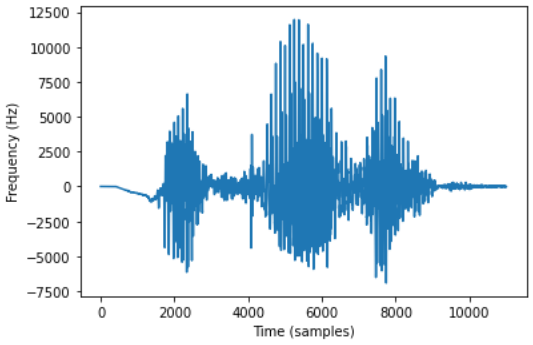
Original Recording:	Synthesized Recording:
	

As seen above, the waveforms look very similar in shape. However, the duration of the recordings is very different. The original recording's sample rate is 44100 per sec, the default sample rate. The synthesized recording's sample rate is 16100 per sec, much lower than that of the original recording. The reason for this is the synthesized recording waveform has "gaps" in it that the synthesizer was not able to capture the style of the original recording entirely. Thus, the vocoder was only able to generate based on almost all of the voiced[7] parts of the original recording.

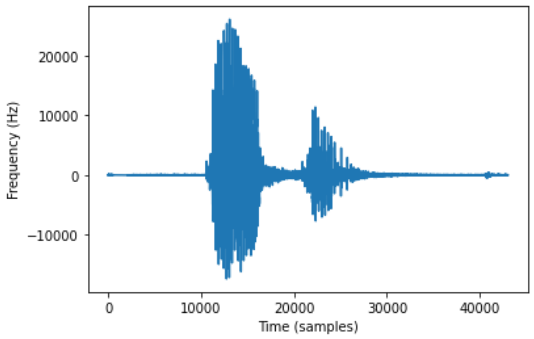
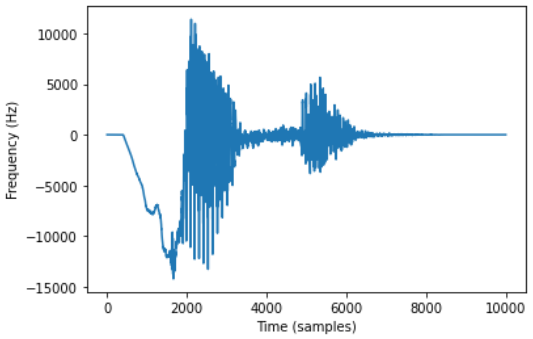
In the following pages are some more examples of recorded and synthesized words.

[7] Voiced parts of a recording are areas that comprise non-zero amplitudes of sound frequency, or vibrations.

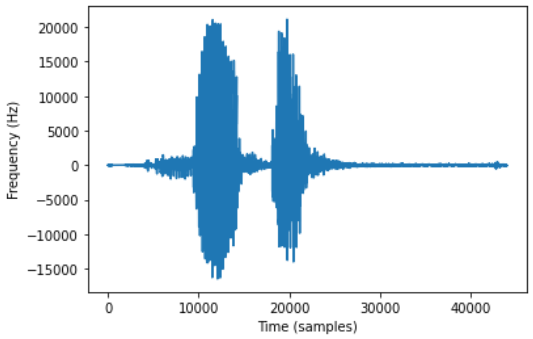
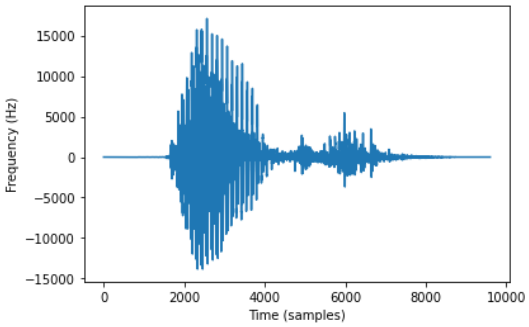
Word → Recorder

Original Recording:	Synthesized Recording:
	

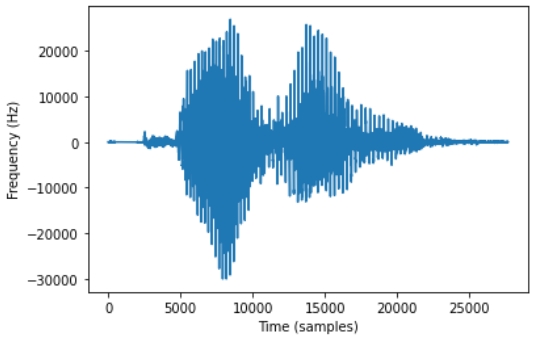
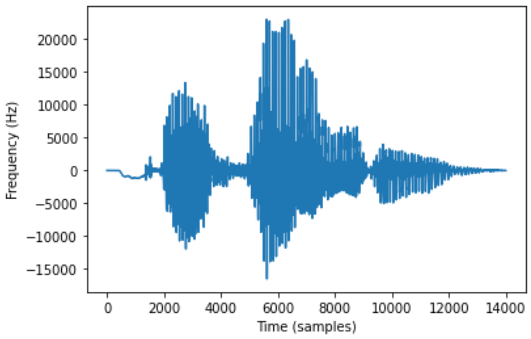
Word → Python

Original Recording:	Synthesized Recording:
	

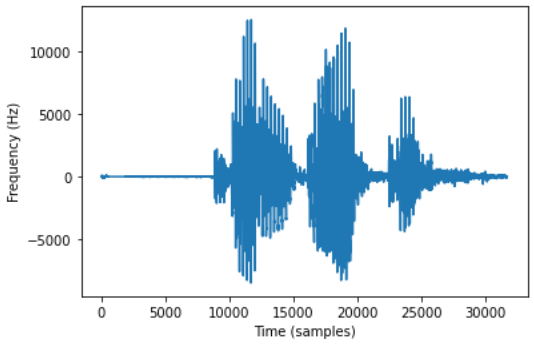
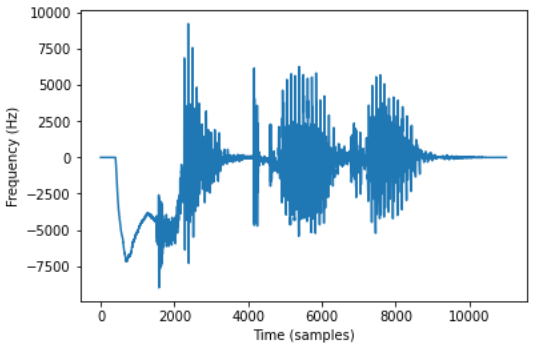
Word → Data

Original Recording:	Synthesized Recording:
	

Word → Programming

Original Recording:	Synthesized Recording:
	

Word → Computer

Original Recording:	Synthesized Recording:
	

Along with the graphs above, I also surveyed a few of my family members and others and had them listen to the pairs of original and synthesized recordings above. My family knows exactly how I sound and acknowledged that the synthesized recordings sounded nearly accurate. Though anecdotal, this provides additional reinforcement to the synthesized recordings' level of accuracy.

Discussion and Conclusion

After researching different algorithms and implementations of Voice Transfer, I eventually came across Jemine's implementation of it, which can be viewed publicly at no cost. While other implementations are available, this one, in particular, is easily accessible from GitHub and is reasonably well-known. Jemine used an RNN (Recurrent Neural Network), and

from my research, I learned that this type of neural network is more capable of Voice Transfer compared with some of the others. RNNs comprise many connections between nodes and layers, and they are able to recognize sequential characteristics via patterns from the input data–this is widely used in domains involving prediction. Hence, RNNs outperform other neural networks in this context. Another way to think about this is how these networks in particular incorporate recurrence or the idea of continuously going around and around in a network. Computationally, these networks start with the input, generate weights, and optimize (as much as possible) for the best model of the input, output based on this, and finally loop back to the beginning to create better weights and thus better models (or optimize for higher accuracy). This can be analogous to the engineering design process, which involves brainstorming ideas to solve a problem, choosing the best one and then testing it, evaluation, and finally back to the start–then further implementations and improving the idea. RNNs are essentially cyclic networks.

An encoder-vocoder system is actually very accurate considering the overall shapes of the different pairs of graphs. However, a closer look with a magnifying glass also reveals large differences between each graph of each pair such as length of samples, amplitude, gaps in the recordings, frequency of the sound, etc. The reason is that vocoders, at least as of now, really only have the capability to take in a portion of all the incoming high-frequency components from recordings/microphones (or any other sources of sound). Thus, they are only able to produce a time series of signals with some accuracy only. Because Jemine used an RNN, some ways to improve this would be to increase the number of neurons and hidden layers of the network as well as using more data and epochs.

What would the next steps be?

As seen in the graphs, the synthesized recordings are not too accurate though they maintain a relatively similar shape to the original recordings. Up to this point, I have mainly experimented in terms of recording and synthesizing. I have also tried to analyze the distances between the peaks of recordings as well as sinusoidal regression. Some ways I can take this further are to record with different people, of both genders; possibly make this multi-linguistic; survey more people who know me and create a proper accuracy algorithm that would perhaps make a model of the original recording and see how well the synthesized recording fits with the model. One of the main obstacles that are present when it comes to synthesizing new voices is the gaps in the synthesized recordings. Another goal should be to decrease the number of gaps overall. This would mean that the vocoder would need to have the capacity to take in additional high-frequency components from the recordings. But this would also mean high computational power. One of the primary goals of computer scientists is to minimize the use of computational power to perform a given task. Think about the movie Jurassic Park (1996, Steven Spielberg). Geneticists were able to recreate dinosaurs by extracting their DNA from very old mosquitos caught in amber, or at least what was left of the DNA. There were gaps in the DNA sequences. To resolve this, the geneticists used frog DNA to “fill in the gaps and complete the codes”. This can also apply here. In other words, instead of using a lot more computational power, it is possible to fill in those gaps with a relatively simple pattern recognition algorithm (simple compared to vocoder enhancement). Using a vocoder is vital for voice cloning, but it need not necessarily be improved for better results. Having these kinds of algorithms to “fill the gaps” are

more than adequate for improvement.

Appendix:

Code:

In order to access Jemine's GitHub as well as import all the necessary libraries, I coded the following.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
from playsound import playsound
from io import BytesIO

import sys
sys.path.append('Real-Time-Voice-Cloning')

from IPython.display import display, Audio, clear_output
from IPython.utils import io
import ipywidgets as widgets
import numpy as np

from synthesizer.inference import Synthesizer
from encoder import inference as encoder
from vocoder import inference as vocoder
from pathlib import Path

encoder.load_model('Real-Time-Voice-Cloning' / Path("encoder/saved_models/
pretrained.pt"))
synthesizer = Synthesizer('Real-Time-Voice-Cloning' / Path("synthesizer/
saved_models/pretrained/pretrained.pt"))
vocoder.load_model('Real-Time-Voice-Cloning' / Path("vocoder/saved_models/
pretrained/pretrained.pt"))
```

To record sounds, instead of manually recording with a recorder app, I created my own program that would use Exception Handling to do so. Essentially, the program immediately starts when it runs and instantly stops when there is a keyboard interruption.

```
import pyaudio
import wave

audio = pyaudio.PyAudio()
stream = audio.open(format=pyaudio.paInt16, channels=1, rate=44100,
input=True, frames_per_buffer=1024)
frames = []

try:
    while True:
        data = stream.read(1024)
        frames.append(data)
except KeyboardInterrupt:
    pass

stream.stop_stream()
stream.close()
audio.terminate()

sound_file = wave.open("recording.wav", "wb")
sound_file.setnchannels(1)
sound_file.setsampwidth(audio.get_sample_size(pyaudio.paInt16))
sound_file.setframerate(44100)
sound_file.writeframes(b''.join(frames))
```

→ The code above records sound with the pyaudio library and stores the recording into a file with the wave library.

To use Jemine's synthesizer, I coded the following.

```
sound = encoder.embed_utterance(encoder.preprocess_wav('recording.wav',
44100))
text = "Programming"
print("Synthesizing new audio...")
#with io.capture_output() as captured:
specs = synthesizer.synthesize_spectrograms([text], [sound])
generated_wav = vocoder.infer_waveform(specs[0])
generated_wav = np.pad(generated_wav, (0, synthesizer.sample_rate),
mode="constant")
clear_output()
new_sound = display(Audio(generated_wav, rate=synthesizer.sample_rate,
autoplay=True))
new_sound
```

References:

Github. (n.d.). *RealTimeVoiceCloning*. Google Colab. https://towardsdatascience.com/you-can-now-speak-using-someone-elses-voice-with-deep-learning-8be24368fa2b

Mwiti, D. (2019, August 28). *A 2019 Guide to Speech Synthesis with Deep Learning*. Heartbeat. https://heartbeat.fritz.ai/a-2019-guide-to-speech-synthesis-with-deep-learning-630afcafb9dd

Resemble AI. (n.d.). *Real-Time-Voice-Cloning*. Github. https://github.com/CorentinJ/Real-Time-Voice-Cloning

Resemble AI. (2019). Master thesis: Automatic Multispeaker Voice Cloning. *Liége Université Library*, 10-29. https://matheo.uliege.be/bitstream/2268.2/6801/5/s123578Jemine2019.pdf

Seif, G. (2019, July 2). *You can now speak using someone else's voice with Deep Learning*. towards data science. https://towardsdatascience.com/you-can-now-speak-using-someone-elses-voice-with-deep-learning-8be24368fa2b