Dao Vang
GitHub Repository: Vocal_Replacement_of_Audio_Recordings
github.com/dao-v/Vocal_Replacement_of_Audio_Recordings

Replacement of Vocals of a Speaker in Audio Recordings

Abstract

By taking the data extracted in an audio file, the idea of this project was to replace the voice recording of a person with the voice from a speaker used during the model training process. The goal was to achieve results similar to Auto-Tune, in which the user's voice would be displaced or modified to resemble the voice of another speaker. However, the approach used in this project did not result in any usable audio output.

Introduction

Worldwide, many people in the world were not born with gifted vocals, blocking a path of creative self-expression. Such people could have the talent for lyric writing and music production but would have to rely on others for their vocal talents to complete their product. An example of this could be seen with music produced by many popular DJs, where their music typically features a singer who performs the vocal component of the DJ's song. With this unavoidable problem of artists, this project began with the intention of completely replacing the user's voice with the voice of another person.

This project's approach was to take the audio recording of a person in a database, extract audio information from the recording, and then use the extractions to train a neural network model. After training, the user would record what they wanted to be replaced, pick a speaker with the voice they wanted their recording to be replaced with, and then the scripts would perform the same extractions that were done on the training data but now using the user's recording. The neural network model would output a NumPy array representing audio data, which would then be transformed into a machine-readable audio file.

When an audio file is transformed into a human-readable format in Python, the file is converted from machine-readable to a NumPy array, in which the values of the array depict the summed amplitude values for each audio sample. Audio files infer basic information: the number of audio channels, sample rate, and the length of the audio. When the file is transformed into a NumPy array, the number of audio channels depends on the dimensions of the array. An audio file recorded in mono will have array dimensions (X, 1), while a stereo audio file will have an array of (X, 2). In this project, only mono audio recordings were considered as many songs and audio recordings are record vocals in mono or transform stereo vocal recordings to a mono track. Similar to video recording/display where each second contains a certain number of pictures (frames per second, FPS), a microphone records the summed disturbance in the air in samples per second (sample rate). Knowing the sample rate, the length of an audio file can be determined by simply dividing the total number of samples in an audio file by the sample rate to produce the length of the audio in seconds.

Unfortunately, this project did not perform as designed, which could be due to various reasons. However, when listening to the audio file produced by this project, there was a slight indication that this approach could produce the ideal output with changes to different processes.

Method

The methods used in this project largely relied on the vocal databases downloaded from the "LibriSpeech ASR corpus" (http://www.openslr.org/12/) and the audio data extracted using the pyAudioAnalysis package available on GitHub (https://github.com/tyiannak/pyAudioAnalysis). (Only the vocal datasets with the names containing "clean" were used in this project and not all the speakers were used in this project due to the CSV file size created from pyAudioAnalysis. See database documentation for more information on the datasets.)

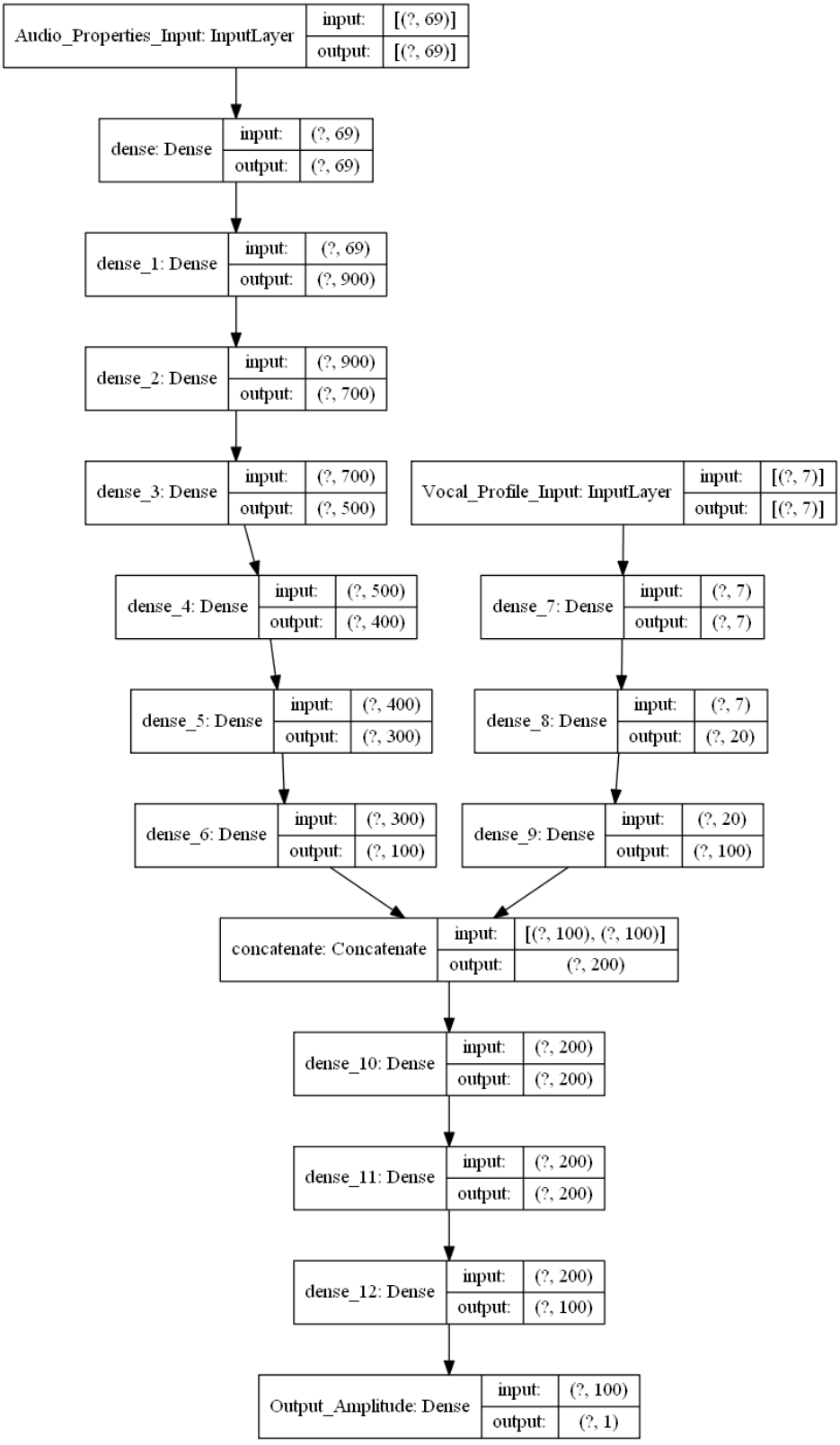
To begin, the vocal recordings of some of the speakers from the vocal databases had the audio properties extracted to produce audio features seen in Table 1. (The table is a direct copy from the pyAudioAnalysis wiki website, https://github.com/tyiannak/pyAudioAnalysis/wiki/3.-Feature-Extraction) The audio features extracted were for each sample of the audio files with a "window" size of 512. Once these audio properties were extracted, the speakers' vocal profile was created by accumulating all the amplitude values from the recordings for a speaker to determine the minimum, maximum, mean, and median amplitude values, and the standard deviation. Next, the neural network model was defined according to the model outlined in Figure 1. The idea when designing the neural network was to give enough nodes to each layer to allow the model to best learn the complex relationships between the many audio features extracted and the resulting amplitude for that audio sample. The main Keras activation function used was the parametric rectified linear unit (PReLU) to account for negative initial inputs and the output layer's activation function was a simple linear function to be able to output a large numeric range in both positive and negative directions. The loss function was mean squared error (MSE) and the optimizer was Adam. All parameters for the activation functions, loss function, and optimizer were kept default. Training the neural network involved inputting two separate datasets (the audio properties and the vocal profile) and the label (amplitude values extracted from the audio file).

Once training was completed, the user's vocal recording was run through the same process as the speakers used in the training to extract the same audio features. A speaker from the speakers that were used during the model training needed to be picked as the voice to be used as the replacement. The audio features and the chosen speaker's vocal profile would then be used in the trained model to predict a NumPy array, which would then be transformed into an audio file.

Table 1: Audio Features Extracted Using the feature_extraction() Function from pyAudioAnalysis

Feature ID	Feature Name	Description
1	Zero Crossing Rate	The rate of sign-changes of the signal during the duration of a particular frame.
2	Energy	The sum of squares of the signal values, normalized by the respective frame length.
3	Entropy of Energy	The entropy of sub-frames' normalized energies. It can be interpreted as a measure of abrupt changes.
4	Spectral Centroid	The center of gravity of the spectrum.
5	Spectral Spread	The second central moment of the spectrum.
6	Spectral Entropy	Entropy of the normalized spectral energies for a set of sub-frames.
7	Spectral Flux	The squared difference between the normalized magnitudes of the spectra of the two successive frames.
8	Spectral Rolloff	The frequency below which 90% of the magnitude distribution of the spectrum is concentrated.
9-21	MFCCs	Mel Frequency Cepstral Coefficients form a cepstral representation where the frequency bands are not linear but distributed according to the mel-scale.
22-33	Chroma Vector	A 12-element representation of the spectral energy where the bins represent the 12 equal-tempered pitch classes of western-type music (semitone spacing).
34	Chroma Deviation	The standard deviation of the 12 chroma coefficients.

Figure 1: Neural Network Model

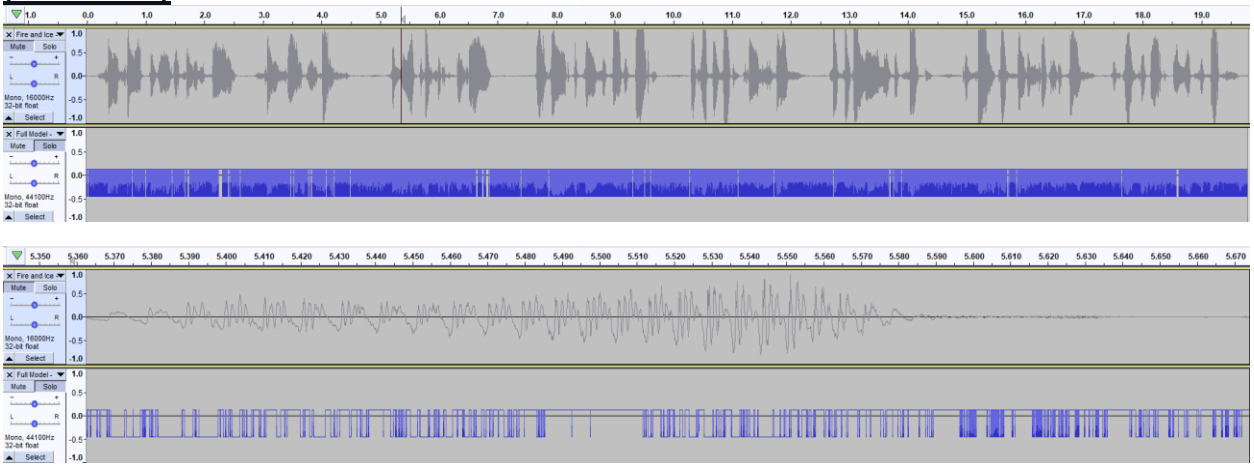


Results

The resulting output NumPy array from the model (as well as various model variants) did not return any vocally distinguishable audio files. Oddly, many model variants failed to produce varying values in the NumPy array elements, meaning that in all elements the values were all the same.

There was one model that returned an audio file with static audio noise with slight hints of a voice. However, when examining the model produced audio file in Audacity (a widely used free-sourced audio tool) and comparing it with the original recording, it was clear that the model was not able to correctly predict the amplitudes as silence in the original recording (amplitude values close to zero) resulted in predictions of high amplitude values. In contrast, where there were high amplitude values in the original recording, there were no correlating amplitude values in the predicted audio file. Figure 2 shows a snippet of the two audio files with the top being the original and the bottom being the predicted audio file after normalization. The original audio file (Original Recording.WAV) and the predicted audio file (Predicted Transformation.WAV) can be listened to and are stored in the GitHub repository. (The recording is a reading of Fire and Ice by Robert Frost.)

Figure 2: Audio Analysis of Original Recording and Predicted Audio File in Audacity (Waveform)



Conclusion & Final Thoughts

Although the results of the project did not produce any usable products, this project did provide an insight into what steps can be taken in the future to improve or create anew a model that would be closer to the ideal outcome. The issues with this project seemed to lie within four factors, limited domain knowledge, the audio properties extracted, the limitations of neural networks, and the performance of Keras/TensorFlow.

The first issue that could have produced the unexpected results could have been the limited domain knowledge of the programmer. However, after examining the available public information on audio data in relation to Python (or other languages), there does not seem to be many useful resources. Also, audio engineering, in terms of industry work, does not require the same in-depth knowledge needed in this project as their tools are typically used to directly

modify audio (adjusting audio properties by trail-and-error) instead of audio analysis (extracting and examining audio properties). This leads to a lot of irrelevant information about audio properties and the understanding of how sound works (computationally) becomes hard to grasp without spending a lot of time studying or developing the field.

The audio properties extracted did not seem to correlate with the amplitude information it was extracted from as the connection between the two did not seem to be established in the model—meaning that reverse engineering of the amplitude was not established by the neural network weights. Different audio properties should be extracted in future versions and changing the output of the neural network model to predict a larger range of samples (such as a second of samples or a sample rate) than one sample could help with the prediction performance as the scope of the input information would be larger.

The regression output values of neural networks for many activation functions are very limited, with many of the minimum return values of the activation functions being around (-3 to 0) and the maximum values being anyone's guess (unless using tanh or sigmoid). Even after rescaling the amplitude labels to a linear scale, confining the labels to a scale between -1 to 1, the models with an output activation function of tanh was returning a constant prediction value for every input. After examining the under-performance of the provided activation functions for labels with large ranges (-35000 to 35000), rescaled or not, it would be better to create a custom activation functions that can accommodate the complexity of the input data and the numeric range of the output values.

Due to the inner workings of Keras/TensorFlow, there are a lot of coding exceptions that needed to be made when the neural networks are not simple models—where models with any custom or "advanced" modifications needed special coding additions to the scripts, regardless if the modifications are pre-defined in Keras. This leads to the possibility that many of the issues faced for many of the model variants were due to Keras rather than the model definitions, such as predicting the same values were prevalent in many model variants regardless of small or big changes to the models or the features fed into the models. Another example was that GPU sync crashes were common as well, and in a few cases before a crash, the training of a model was running with no oddities. But, after a GPU crash, the training of the same model or a new model with the same definition would return non-changing loss values and metrics. This type of bugs consisted even after manually flushing the GPU memory and restarting the computer. With Keras/TensorFlow potentially being a factor in the performance of model variants, it was unclear on the approaches to fix the issues as directly modifying Keras/TensorFlow was a skillset on its own.

Should this project be approached again in a future date, many different strategies could be implemented to simplify the modeling and data preprocessing. However, this project at its current stage would be useful on how to model future projects that attempt to solve the same problem.