Final Report

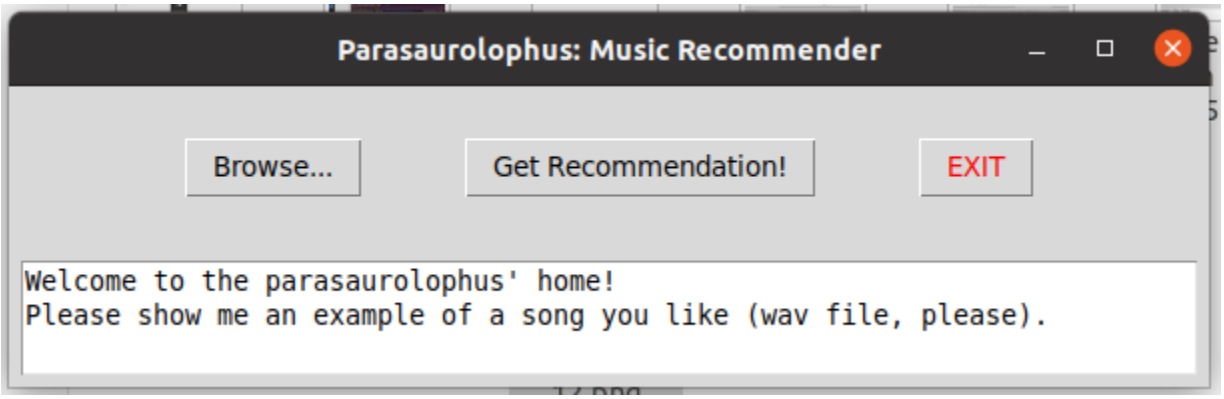
Team Dilophosaurus

Project Summary

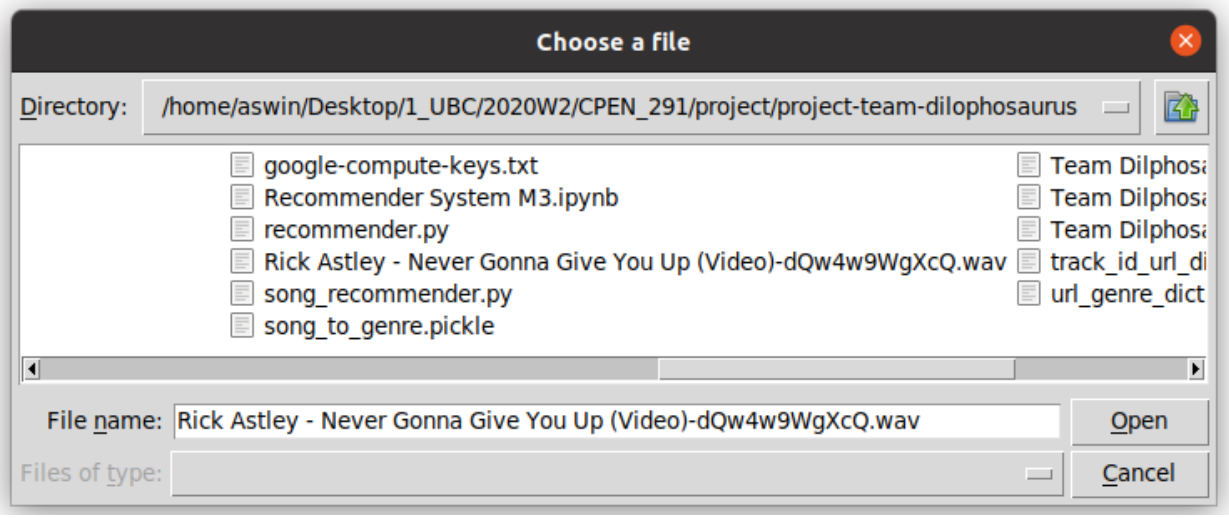
Our project is an application named Parasaurolophus that is designed to give music recommendations to users. The application takes an input wav file and uses a genre classifier's prediction to recommend other music from the same genre. The application recognizes and recommends music from eight different music genres, namely: Blues, Country, Electronic, Folk, Jazz, Latin, Rock, and Rap music. The recommendation is also automatically copied to the user's clipboard for convenience.

App Walkthrough

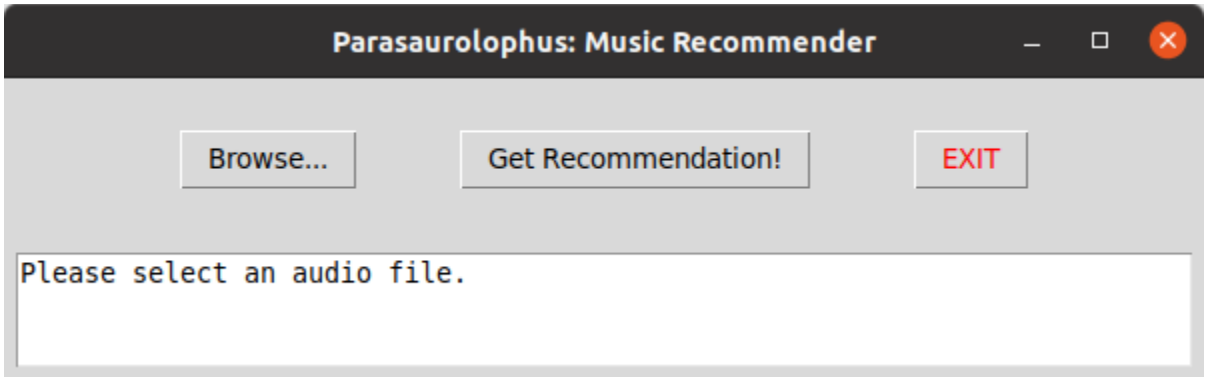
1. The app's welcome screen



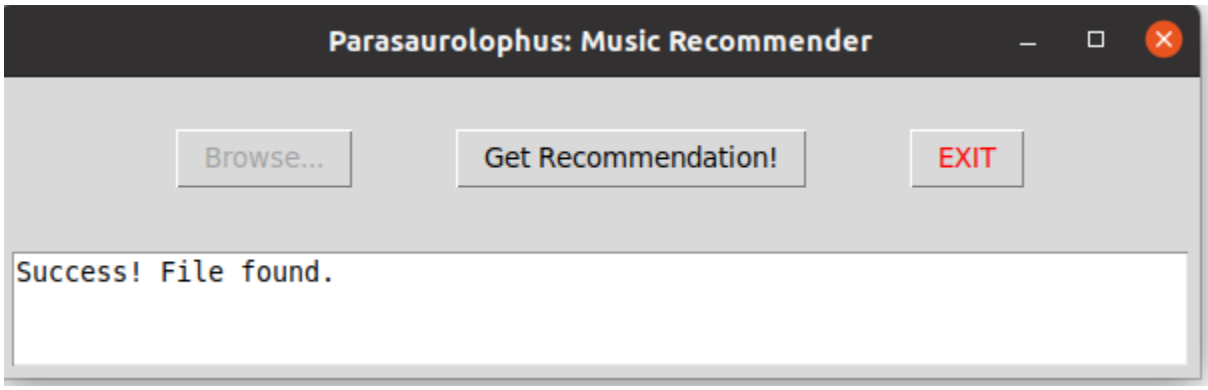
2. The file browser after selecting [Browse…]



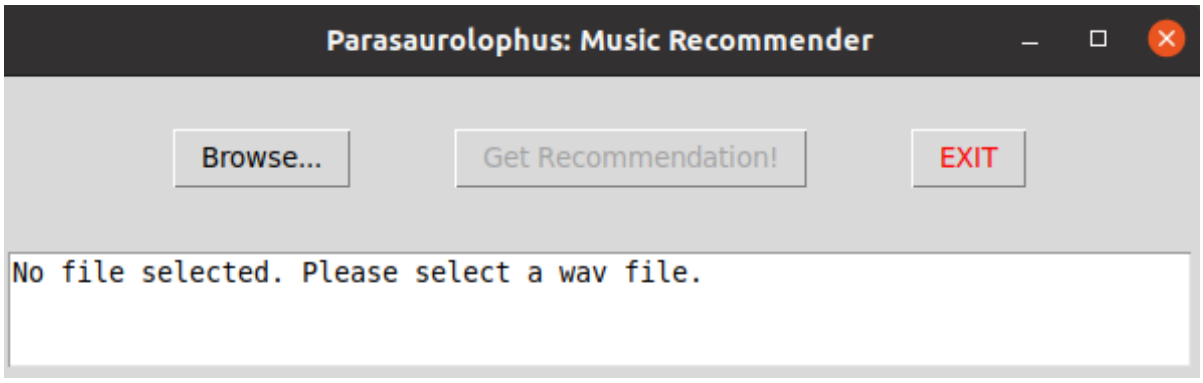
2b. [Get Recommendation] was pressed before inputting a file



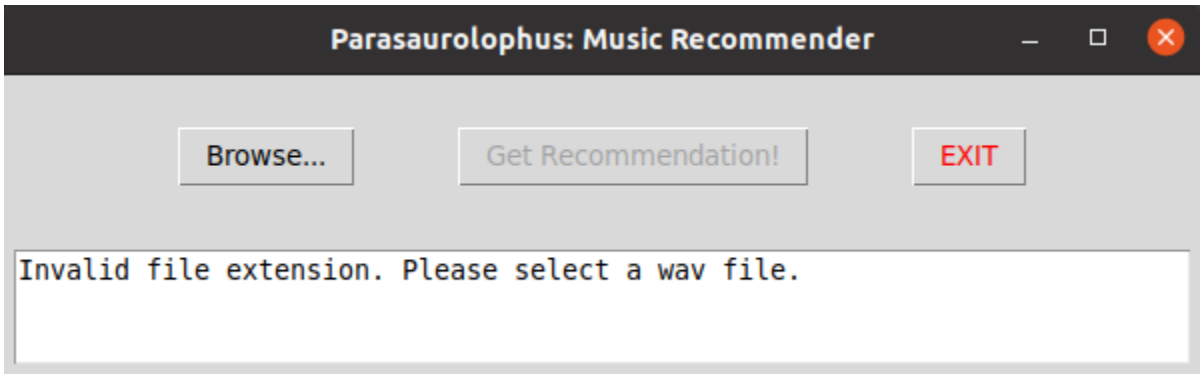
3. A valid input file has been selected



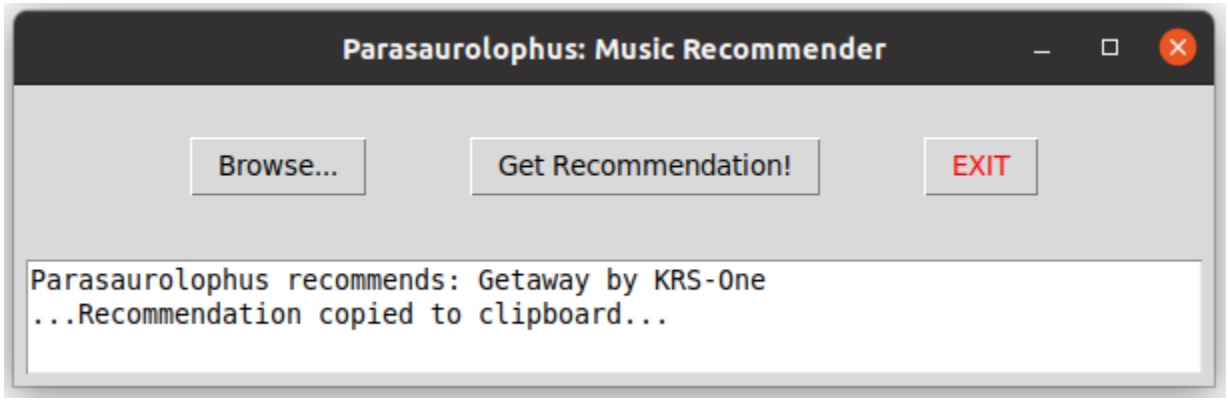
3b. A file was not selected after clicking [Browse…]



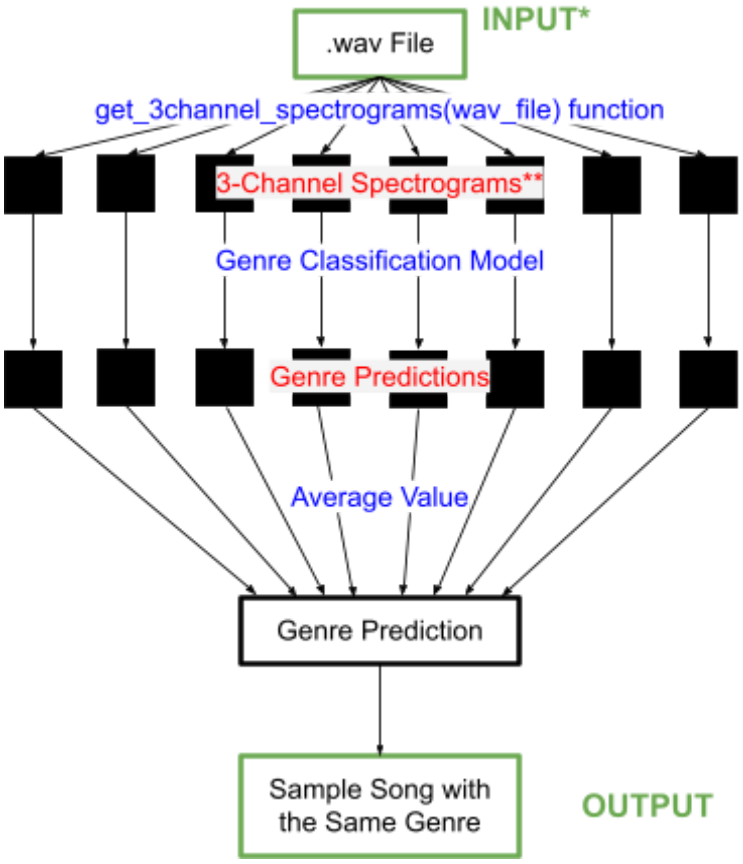
3c. File selected was not a .wav file



4. The Parasaurolophus gives a recommendation!



Data Flow Diagram



* The input is required to be a wav file with duration of at least 90 seconds.
** “3-channel spectrogram” = A colour image (of size 224 x 224) which has a spectrogram representing 30 seconds of music for each of its three colour channels.

Modules

Classifier Creation: *genreClassifier M4.ipynb*

Sample URL Finder

The sample URL finder was used to create a dictionary of songs to Youtube links. The finder uses the .h5 files collected for each genre from the MillionSongDataset to get the information of the songs such as song title, artist name and genre. Using that information as an input of the function *search_and_grab_url(search_term),* we get the youtube link of each song. In total, we created 3 different dictionaries: URL to genre, track ID to genre and finally song to genre. Each dictionary was saved using Pickle so that we do not need to create dictionaries everytime we run the classifier.

Sample URL Filtering and Audio Extraction

With the dictionary of songs mentioned above, we iterated through it using a loop to identify and remove duplicate links. After duplicate links were removed, we downloaded the audio of each video in .wav format. We then ran our spectrogram extraction function, *get_3channel_spectrograms(song_file)* on these wav files and extracted spectrograms to add to our dataset.

Dataset

Our dataset was originally planned to be created exclusively from the Million Song Dataset. Due to storage constraints for the 300GB dataset, we instead used samples from the Million Song Subset and scaped Youtube with the modules above for additional samples. Our final dataset has 547 samples for each genre (total 4376). Our dataset is stored as eight separate directories, one for each genre. (ie. the 'Rock' directory contains all samples for the rock genre). Then, for the samples that our dataset class *genreClassificationDatasetSpectrogram* outputs, it gives a label corresponding to the directory in which the sample was found.

Classifier Training

The training and testing pipeline involve the typical steps:

- Instantiating a dataset.
- Splitting the dataset into training and testing sets.
- Feeding the datasets into a data loader.
- Setting an optimizer and scheduler.
- Finally, running some epochs of model training and testing until convergence.
    - We found that the genre classifier usually converges in about 10 epochs.
    - With 547 samples per genre in the full dataset, this process takes about 20 minutes (given GPU support).

Genre Classifier: *VGG-19*

The genre classifier is a model created by applying transfer learning on a pre-trained VGG model. The model takes in three channels of spectrogram inputs. To produce these spectrograms, we have created a spectrogram extractor that takes in wav files and extracts triads of spectrograms each 30 seconds long. The model then uses these spectrograms to classify inputs as one of eight predefined genres. The genre classifier was trained on a custom dataset composed of samples from the Million Song Subset as well as videos from Youtube. Our final version of the model was trained on a dataset containing 4376 samples (training and testing combined).

Recommender: *class Recommender from recommender.py*

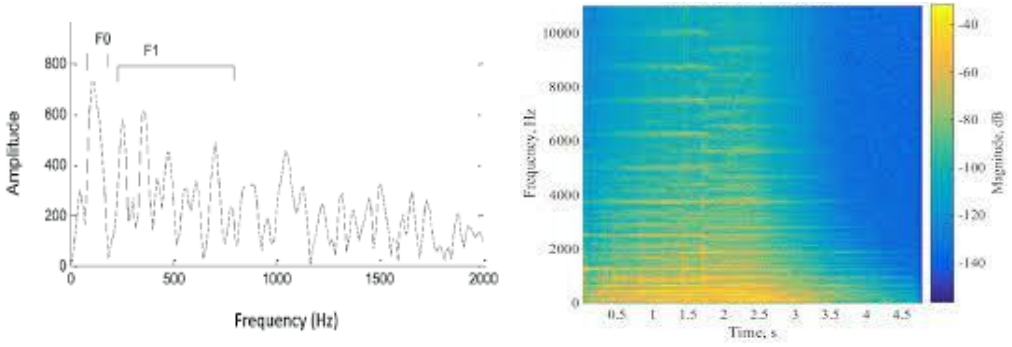
This class is a wrapper for the genre classifier. It loads the genre classifier, generates spectrograms from an input .wav file, feeds them into our model, interprets the model's outputs to identify the genre of the input file, and recommends another song in the same genre.

Spectrogram Extractor: *get_3channel_spectrograms(self, song_file)*

The method *get_3channel_spectrograms* from class *Recommender* takes in a .wav file and returns a list of tensors of shape 3 x 224 x 224. That is, triads of spectrograms, each of shape 224 x 224 and representing 30 seconds of the input .wav file. The method segments each full 90 seconds of the .wav file into 30-second chunks. Librosa, a popular audio analysis package, is used to calculate the short-time Fourier transform of the 30-second chunks. The result of the transform is then converted into spectrograms by converting the complex-valued output of the short-time Fourier transform into real numbers. Finally, these spectrograms are consecutively grouped into threes and packed together as tensors of shape 3 x 224 x 224.

Short-time Fourier Transform?

A short-time Fourier transform is essentially like a Fourier transform applied on sections of the audio at a time, rather than on the entire length of audio all at once. So, instead of producing a single histogram of frequencies that make up the audio, it produces one for each subsection of audio. This difference is shown in the images that follow, with the output of a Fourier transform on the left, and that of a short-time Fourier transform on the right; each vertical strip of the graph on the right is a histogram similar to that in the left hand image, produced by the regular Fourier transform.



We use the short-time Fourier transform to extract spectrograms from audio, as opposed to the regular Fourier transform, because the former gives more time dependent information about audio, such as rhythm and melody.

Shaping the Output of the Short Time Fourier Transform

The shape of the short-time Fourier transform's output is given by $frequencyBins \times numFrames$, where $frequencyBins$ and $numFrames$ are defined by the equations

- $frequencyBins = (FRAMESIZE / 2) - 1$
- $numFrames = [(samples - FRAMESIZE) / HOPSIZE] - 1$

For the short-time Fourier transform function from Librosa, the shape of its output was made to be 224 x 224 as desired, by setting "FRAMESIZE" and "HOPSIZE" in "librosa.stft(song_clips[i][j], n_fft=FRAMESIZE, hop_length=HOPSIZE)" appropriately.

The variable $samples$ is equal to the sampling rate (22050 Hz in this spectrogram extractor) times the duration of the audio on which the short-time Fourier transform is being applied (30 seconds in this spectrogram extractor). Thus, $samples$ is equal to 22050 x 30 = 661500. With that, finding appropriate values for FRAMESIZE and HOPSIZE is a matter of solving for them in the following system of equations:

- $(FRAMESIZE / 2) - 1 = 224$
- $[(661500 - FRAMESIZE) / HOPSIZE] - 1 = 224$

With Librosa's default sampling rate of 22050 samples per second of audio, FRAMESIZE := 446, and HOPSIZE := 2965 ensure an output shape of 224 x 224 for the short-time Fourier transform.

Model Interface: *get_genre_predictions(self, imgs)*

The method *get_genre_predictions* feeds the loaded genre classifier 3-channel spectrograms from the list of 3-channel spectrograms created by the **spectrogram extractor**.

Again, 3-channel spectrograms are triads of spectrograms of size 224 x 224 merged into a tensor of shape 3 x 224 x 224. This tensor shape is the one accepted by our genre classifier.

The method then takes the classifiers genre predictions for all the 3-channel spectrograms and returns their average.

Recommendation Generator: *get_recommendation(self, genre_label)*

The method *get_recommendation* takes the output from our genre classifier as an input, and uses it to produce a recommendation from the same genre as the input song. However, rather than implementing a recommender system, we opted to randomly select the recommendation from the corresponding genre folder in our dataset. The implementation for the recommender is relatively simple compared to our other modules. Taking the prediction from the genre classifier, we iterate through songs of the desired genre in the dictionary, *song_to_genre,* that maps a string containing song title and artist, to the song's genre. This continues until a randomly generated index up to the number of samples for that genre is reached. The key in *song_to_genre* at which the iteration stopped is returned.

GUI: *song_recommender.py*

The GUI was implemented with the TKinter library and imports the genre classifier, spectrogram extractor, and recommender. The GUI features an exit button to close the window, a browse button which uses the tkfiledialog module, and a button that runs the classifier, extractor, and recommender. A text box created using tk.Text is used to show results. To prevent the user from editing the text box meant for the output, we used button states to keep the text box disabled whenever the program is not writing to it. Should the GUI be used improperly (ie. the recommender is run without an input or the input is not a wav file), an error message is written into the text box describing the issue. After a recommendation is made, it is automatically copied into the user's clipboard.

Contributions

Individual Contributions

- Carroll, Quinn
  - Collected h5 files of *Rap and Country* songs from the Million Song Dataset (M1)
  - Performed testing on the model to find the best values for gamma and momentum (M2)
  - Created class genreClassificationDatasetSpectrogram in paired programming session with Cassiel (M3)
  - Collected spectrograms from youtube audio to increase number of samples in *Blues* subset (M4)
  - Created data flow diagram of our genreClassifier process (M4)
  - Created outline for video presentation (M4)
- Jung, Cassiel
  - Collected .h5 file of *Folk and Blues* from the Million Song Dataset (M1)
  - Tested the model by differing the step size while keeping the other values constant to get the best accuracy (M2)
  - Wrote dataset class that matches spectrograms and genre with Quinn (M3)
  - Collected spectrograms to increase number of samples used for training model (M4)
    - Folk, Blues, Latin and Country

    - Made last GUI feedback to check if everything works fine (M4)
- Poon, Matthew
    - Created a script to extract and filter h5 song samples from the Million Song Dataset (M1)
    - Collected h5 files for Metal, Reggae, Classical, Latin, and Electronic music. (Metal, Reggae, and Classical discarded due to lack of samples) (M1)
    - Created the structure used for storing the dataset (individual genre folders) (M2)
    - Performed tests to determine the best learning rate for our model (M2)
    - Created a script to extract information and save dictionaries for URL:Genre, TrackID:Genre, and Song:Genre (M3)
    - Debugged GUI functions with Aswin (M4)
    - Collected spectrograms for additional samples for the genres listed below: (M4)
        - Rap, Latin, Jazz, Electronic
    - Recorded and edited the video for submission
- Sai Subramanian, Aswin
    - Collected .h5 files for Rock and Jazz from the Million Song Dataset (M1).
    - Performed tests to determine an effective batch-size to speed up training time (M2).
    - Developed spectrogram extraction process (M3).
    - Developed process to download audio samples from YouTube, and store spectrograms generated from them into our dataset (M3).
    - Merged spectrograms collected by team members into full dataset (M4).
    - Debugged GUI functions with Matthew (M4).
    - Experimented with model training parameters such as number of genres, number of samples per genre to find trends in training results. This helped make decisions about how to improve the genre classifier's accuracy (M3/M4).

Shared Contributions (Done in Paired Programming)

- Created a custom dataset class that:
    - Extracts features found in h5 files in the Million Song Dataset (M2)
    - Takes these features and moulds them into the correct shape for the VGG model (M2)
- Created a GUI combining the different modules of the project (M4)
- Created the Google Slides presentation and recorded audio for the video