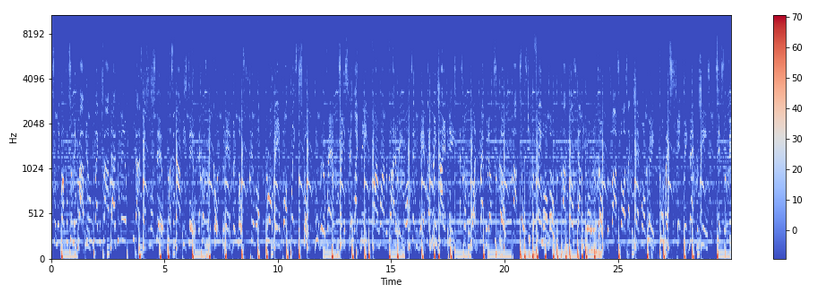
**Final Project**

**Introduction**

My final project is music generation/classification using an LSTM network that has been trained on a dataset of genre classified tracks so that the model can generate and classify a specific genre of music. The plan for the project is to be able to do something like what was seen in lab 4 of the class. What is needed is for the data given, in this case the music audio, to be tokenized and labeled so that it can be inserted into an LSTM model for training. To do this I used a library called Librosa to break down the data from just audio files into features that could be used to train a model. Librosa is a python package for music and audio analysis that creates information about the audio files by measuring its features like frequency, magnitude, decibels, etc. The data that is created from this is used to compute the Short Time Fourier Transom (STFT) which is just several Fourier Transforms or decomposed complex periodic sounds at different points in time. In an easier to understand explanation it’s computation for the audio file’s sound waves over time. This STFT can be displayed by Librosa as well so that we can see the data that it is gathering. This is shown below in a special type of graph called a spectrogram that displays the time, Hz, and decibels  


*Figure 1: spectrogram of sample data*

The most important feature that is extracted from the audio using Librosa is the Mel Frequency Cepstral Coefficients (MFCC) which captures timbral/textural aspects of sound which allows models to tell the difference from what type of sound is produced. For example, this would allow the model to know the difference from a violin sound and piano sound even if they have the same pitch and frequency. This is supposed to approximate the way that humans can tell apart sounds. The process for how Librosa calculates MFCCs is very complex and goes out of the scope of this project but its important to understand why MFCCs are important for training. These MFCCs allow us to make a classifier model that can predict and try to classify music as a certain type of genre but using a model that is only trained on MFCC values can not generate music since the features of an MFCC can not be translated back into music. Because of this I took two approaches to generating music for the classifier to try and classify.

The first approach involved using a very complex model from open AI called Jukebox that involved three different encoding layers that used a quantization-based approach called VQ-VAE. I was not able truly make any changes to this model nor was I able to train it any differently considering the model for Jukebox was already trained on 1.2 million songs and was ridiculously large. I’ll go into further explanation by what I mean by this but as a brief explanation the model is so big just running it a few times takes actual days to produce new music.

The second approach was to create music using a more simplistic method but one that would ultimately create more random samples with no specific genre in mind during training. This approach was to use midi files to train a LSTM model that only produced music using one instrument, in this case a piano. To do this I used a library called Magenta which processes music and image data by using midi files. Midi files are essentially audio files turned into images. It does this by turning notes into pixels and spreading them out using the x axis for time and y axis for pitch. Basically, looking something like figure 2 below.



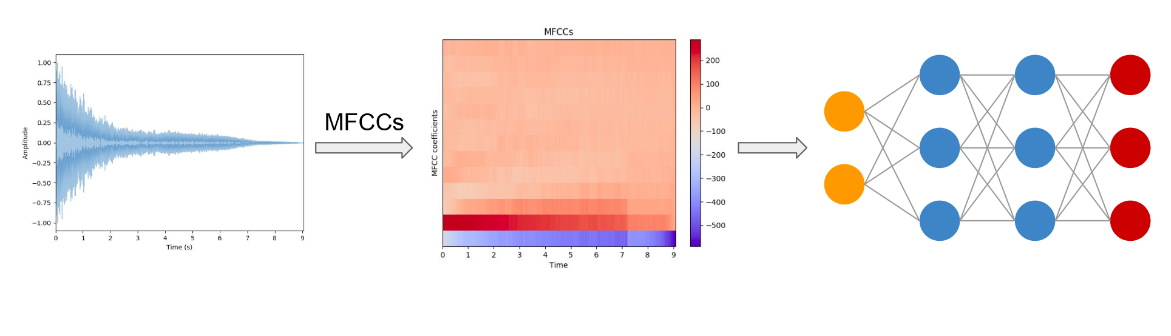
*Figure 2: midi file representation of piano notes from Magenta model*

The model that is trained on these midi files is an LSTM model with three LSTM layers and dropout layers in between and a dense layer that connects all of this to the output layer. This model produced good results and could generate music but none of the music really fit a genre so to give it a classification was not really something I could do. I could however run the audio file through my genre classifier to see what it could classify the music as.

**Collecting data and training for classifier**

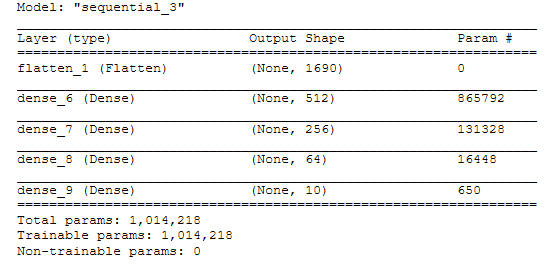
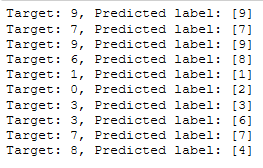
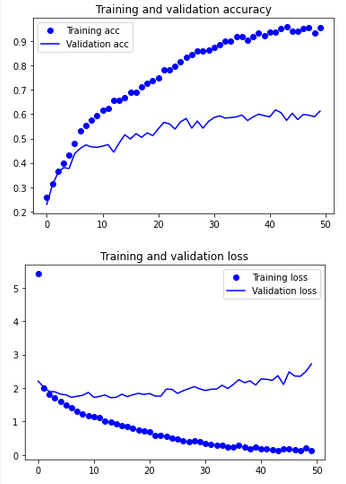
I have collected and preprocessed the Marsyas genre dataset from the following link. [https://www.kaggle.com/andradaolteanu...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbGZseWk4LURLUXhnZmhNWjdSbzZ3dENXT2FOd3xBQ3Jtc0trdFExQndiOXctTm1JdUZPRF9LbnhuLVBwVWJjWTdMTjBFSTZBZ3BKYkFrSjZacktzTmQtRm41T0pnaWROeXd0cDdzQzdqWHNCNWVTS2tKaV8xYWZmVWlBM05RZURZbGZlS1paNWZicUUwaTNDVnk5dw&q=https%3A%2F%2Fwww.kaggle.com%2Fandradaolteanu%2Fgtzan-dataset-music-genre-classification)

Using this dataset, I made a simple music genre classifier by processing the data into a json file that contained the dictionary MFCC values for all the tracks in the dataset. The dataset itself is not very large with only about 100 tracks per genre of music so to make things easier I split up the tacks into smaller tracks which I found to be a common process that many papers on music classification did. After splitting the tracks, I had about 999 music sample for each genre for the classifier which was split into two thirds training and one third validation data.



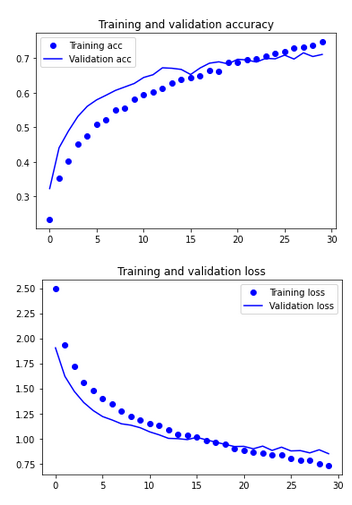
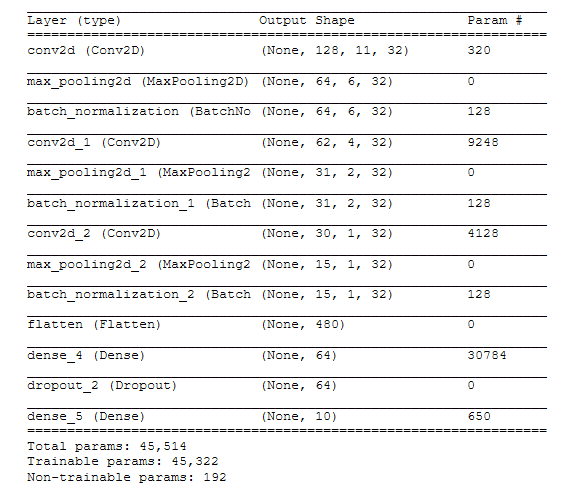
*Figure 3: pipeline representation for how audio data was transferred into MFCC features and fed into the model*

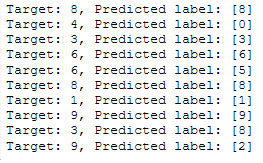
The neural net created for this classification was only a simple NN at first with one flatten input layer, three relu activated hidden layers, and one softmax output layer. For the dataset I was using for testing I only have 10 labels so there where 10 outputs. The results of this simple NN were not amazing because it shows definite signs of overfitting in both its accuracy and loss diagrams.

*Figure 4: Basic NN outputted accuracy graph, summary, and predictions on a few of the test data results, acheives about a 56% accuracy on validation and a loss of up to 3*

After building this first classifier I then tried different models to see if it could be improved. The next model attempted was a CNN that contained 3 conv2D layers and three maxpooling layers, this was tested with different layers amounts and variable changes to get an optimum output. This next model performed better in both accuracy and reducing overfitting.

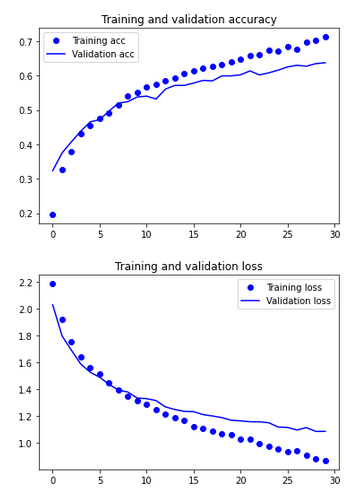
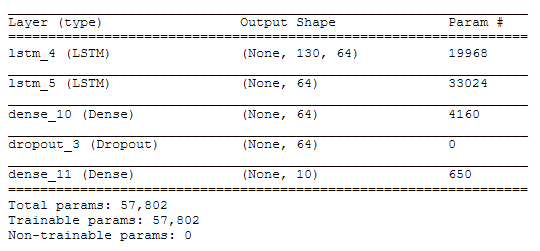
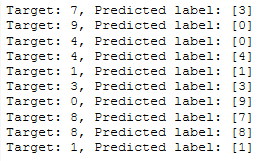




*Figure 5: A CNN model’s outputted accuracy graph, summary, and predictions on a few of the test data results, achieves about 70% validation accuracy and loss below 1*

The results of the CNN where much more promising than the base network with a dramatically smaller amount of overfitting seen in the graphs and even an increase in the validation accuracy for the model. The results of this model make it good at classifying considering there are 10 different genres for the model to be classifying music as.

The last model I tried was a LSTM model that I predicted have higher accuracy with then the other models. This model is what I wanted to use to output generated music similar to how we generated text from an LSTM in class. However, when looking at the results of the model we have a slight decrease in accuracy from the CNN model.

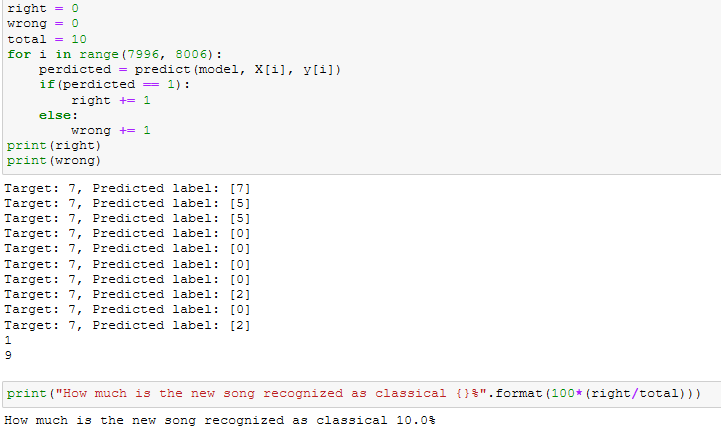
 

*Figure 6: LSTM outputted accuracy graph, summary, and predictions on a few of the test data results, acheives about a 60% accuracy on validation and a loss of up to 1.2*

So the classifier is able to somewhat accurately classify audio files into genres of music but as stated earlier it does by using MFCC values of audio files and these MFCC values although can be predicted can’t be used to generate new audio. In the class prior we were able to generate text using data from text files that where being passed into a model, but in this case the values that we can output from the model can’t be used to generate music. Because of this, instead of using this classifying model to generate music I decided to make a different model that could generate music and see if the classifying model could classify the genre of music that was generated.

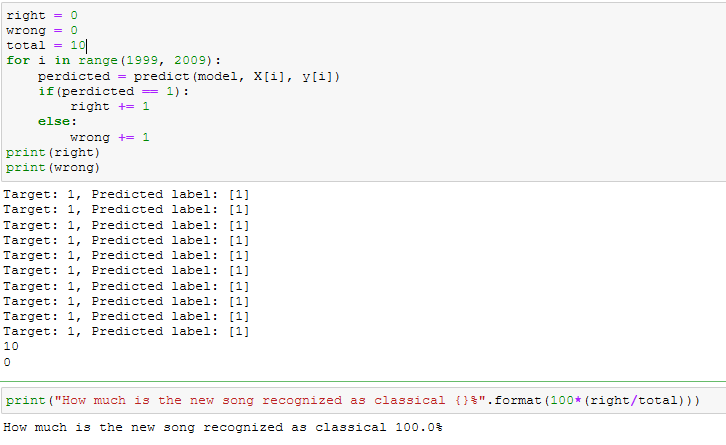
**Open AI Jukebox**

Using Jukebox to generate music was interesting and allowed me to specify the genre of music that I wanted the model to create. However, Jukebox is a cumbersome model trained on a dataset way larger and more diverse than anything I could feasibly get myself so making any meaningful contributions to Open AI software was almost impossible for me to do. The best I could really do is test the model on my own data to see how it could perform so my initial idea was to gather samples from my test data, one of each genre I was classifying and see what it could make from those samples. I quickly ran into two separate problems with this experimentation. The first was in how Jukebox compressed their data passed into the model. Because of the multiple decoding layers in the model each time the audio was decoded and reconstructed for the next layer, more noise would be added to the file and to the prediction of the model. This led to noisy files that had a lot of background sound that ultimately could not be recognized as music of any specific genre. The first attempt at this was using a pop song sample from the test data that was roughly only 30 seconds long. I passed this audio file into Jukebox and had it primed on only first ten seconds of audio in the file. This meant that Jukebox would listen to the first 10 seconds of the file with out trying to change or add any audio before taking over and generating its own music. This first attempt will be attached in the directory called pop. When listening to the samples generated by Jukebox for this song its immediately apparent that the background sound is making it almost impossible to hear the music. I also found that this background sound was consistent in other tests using different audio which would ultimately make each outputted music sample extremely similar and impossible to classify for a model looking for features in the audio files MFCC values. This is further shown in the figure 7 below which shows the predicted values of what classification model thought the music sample was at different points in the audio



*Figure 7: Misclassifications of the audio generated by jukebox that show that it could only recognize the first few seconds of the song sample as the target output*

You can see from the figure that the target output should be 7 which is the classification for pop music but the only time the classification model even recognizes the music as being pop music is at the very beginning of the music sample which we already stated included 10 seconds of unchanged music from the original pop song the sample was primed from. This means that none of the generated parts of the sample where actually recognized as pop music and I believe this to be the fault of the noise generated in the background from the decompression. To test this theory, I had to find a way to remove the noise from the outputted Jukebox files. I was able to do this by using some tools made by open AI and documented by a GitHub user Zags who created a google colab file that allowed me to up sample the music files outputted by Jukebox. This process was able to get rid of the noise that was generated by Jukebox when creating music samples and even made it possible for my genre classifier to correctly classify the entirety of the generated sample as the correct genre of music. This can be seen in my next attempt using Jukebox where I used a classical music sample from my data to prime the generated music for Jukebox. Jukebox created three different samples based off of this classical music and each one of them were classified as classical throughout the entirety of the generated part of the files. These samples can be found in the attached directory called classical and the results of one the samples being predicted by the classifier can be seen in Figure 8 below.



*Figure 8: the classifier model classifying a Jukebox generated sample as classical which is what the sample file was based off of and should be*

However, there was one major problem that still arouse from this process that made testing almost impossible beyond the one generated classical sample that I had. Jukebox is a very large model that takes a long time to generate music even just by itself taking hours with decent GPU usage to generate samples like three sample I just mentioned earlier. The up-sampling process developed by Open AI is even worse at generating files in a timely matter. In figure 9 below you can see how it basically took me a whole day to get the up sampled audio I needed for classification.

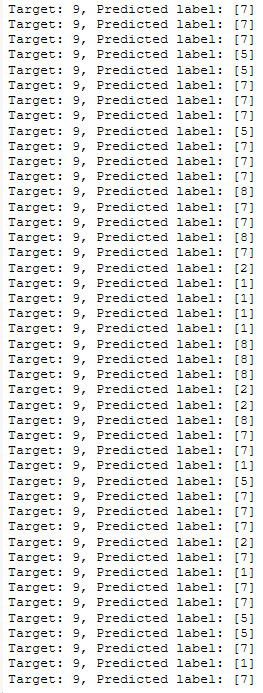


*Figure 9: timestamp of google colab 9 hours and 50 minutes into running the up-sampling process*

Because of this I had to look for a different way for me to generate music that could be tested on the classifier which led to the next approach in music generation.

**Generation using midi files and Magenta library**

My first attempts at using midi files were made using the Magenta library which has a few pretrained LSTM models on it there where already capable of generating their own midi files of generated music. Using the structure of one of these models called the lookback model which specialized in looking back at previous notes generated to create repeated notes that could be used to make a structured chorus I developed a new model that involved three LSTM layers and dropout layers in between and a dense layer that connected all of this to the output layer. From this model I was able to train it on midi files from old Nintendo consoles that was able to get from this site <https://www.vgmusic.com/>. Then I was able to output the midi files using the note generation apart of Magenta’s library with different types of synthetization for notes. These files are attached to directory called midi and can be seen being passed into the classifier below for figure 10.



*Figure 10: Predictions of the midi files created from model trained on video game music, the predictions are primarily pop and jazz, 7 and 5, then reggae and blues, 8 and 1, with a little bit of classical, 2. The target output can be ignored since it is just a place holder for how the prediction method was set up, the midi files don’t have a genre because they were trained off video game music.*

Looking at the classifier the music that was generated by model using midi files from video games was primarily classified as pop and jazz. These are the values 7 and 5 respectively. For the target value I just set it to 9 which is rock because that was how I had the function for predictions set up.

**Conclusion**

In the end I was not able to fully recognize my project as one neural network that could classify music and generate music of similar genre classifications but this, I believe is because of the complex nature of the generation process for music. Since I was trying to use audio files of songs which have multiple instilments, lyrics, and ultimately way more features and data then just one instrument, generating comprehensible music from these features is more changing. I believe if I were to continue this project, I would continue to further explore the process of using midi files to train a model that can generate music since I believe it is possible to still have different genres of music played on only one instrument like a piano. By keeping the data in this simple format, I can make a model that can not only recognize patterns form the midi file such as tempo and pitch based off the note placements in the midi image, but in theory it should be able to recognize genres of music played on this instrument based off the notes and how they are played. Despite this however, the music generated from such a model will only still be able to be played on one instrument and based on what I have seen from LSTMs trained on midi files, could still be random when generating music, even with thousands of epochs on some pretrained models the music was kind of just throwing notes out like crazy. Because of this fact, Jukebox may actually be the proper way to generate music especially if it can be further trained or fine tuned to not produce as much noise, not take as long, and not have random moments in generation where lyrics seem to distract the model from generating actual music.