Music Genre Classification Using Convolutional and Recurrent Neural Networks

Edison Liang, Matthew Kelleher, and Jared Cole

Introduction

Music genre classification is an interesting problem in the field of audio processing, and has proven to be a fundamental challenge that furthers the development and experience of users, streaming services, and music production. As a group, we worked to train convolutional and recurrent neural networks to classify songs into one of 10 genres. We went through different phases of the project, initially using Spotify's API with numerical data and then through training data solely from .wav files. In all, we wanted to predict, with reasonable room for error, the genre of songs based on 30 second snippets of music.

Dataset Retrieval & Preprocessing

Initial Approach

Initially, we attempted to classify 6 genres based solely on numerical data from Spotify's API. We used a dataset of 30,000 songs, each with numerical data on the song's danceability, energy, key, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, and duration, with genre as the label. The preprocessing entailed turning the csv data into pandas dataframes, then numpy arrays. As numpy arrays, we were able to utilize sklearn preprocessing to fit and transform features into standard scale and genre categories into one hot encoding. Standard scaling proved to be crucial as our accuracy was only around 20% before it was implemented. Overall, the preprocessing effectively transformed the data to be highly usable, but the data itself was limited due to the nature of high variability even among genres, so it could only achieve a best testing accuracy of 53% with the recurrent network.

Refined Approach Through Use of .wav Song Files

Using a dataset found online of 10 genres, each with 100 30 second song snippets, we were able to collect a lot of data by using a high sample rate. We utilized a sample rate of 22050 samples per second, and with 30 second tracks this made for 661,500 samples per track. This preprocessing helped us turn 100 songs per genre into 66.15 million different data points per genre, which greatly helped our neural networks with lots of information.

Within each track, we use the python library librosa to process the .wav files into usable numerical data. We load the track into a signal, then split the signal into 5 segments, and generate Mel-Frequency Cepstral Coefficients (MFCCs) via short-time fourier transform. The MFCCs used collectively make the Mel-Frequency Cepstral (MFC) which as a whole is the short-term power spectrum of a sound. After collecting the

MFCCs, we dump them into a JSON file along with the genre labels, and split the data into a 75:25 train to test ratio. Most importantly, the MFCCs acted as a way to turn raw waveform data into highly usable and available numerical information.

Implementation & Architectures

With both approaches we decided to go with recurrent neural networks and convolutional neural networks. We selected these neural networks because within the nature of song classification, there aren't many long-range dependencies to look out for. Certainly there are repeating song structures and harmonies, but that is generally song-specific and not specific to a genre. For this reason, the reliable convolutional and recurrent neural networks looked to be the best option.

Initial Approach

For the convolutional neural network in our original numerical data training, we used tensorflow to layer 1D convolutional layers with max pooling layers, then flattened down into non-convolutional dense layers. Even with tinkering around with all sorts of layer formation, we were only able to achieve a best accuracy of 49%.

Similarly, for the recurrent neural network, we used tensorflow to sequentially layer. We tried layering with LSTM, GRUs, bidirectional and not, and again we found ourselves with underwhelming results of 53% at its best.

Refined Approach Through Use of .wav Song Files

For the convolutional neural network, we have three convolutional layers that flatten into a dense neural layer. Each of the three 2D convolutional layers are composed of 32 filters, a 3x3 kernel, and the rectified linear unit (relu) for the activation function. After each convolutional layer, we utilize a maxpooling layer to downsize the convolutional output. The maxpooling layer strides in a 2 by 2 fashion so 2 at a time across the x and y axes. Finally, we have a batch normalization layer to ensure faster convergence and a more stable network through recentering and rescaling. These three layers are then repeated 3 times, and sent to more layers to flatten and then process through a dense neural layer. Finally, we decide on our output through the softmax function with 10 options for the 10 genres in another dense layer. In all, there are 13 total layers to this neural network

For the recurrent neural network, it was composed of LSTM layers and dense layers. We used two LSTM layers of 64 units, then a dense neural layer of 64 with the relu activation function. Lastly, we used a dropout layer that drops out 30% of the units to prevent overfitting before putting it into the decision layer which again is a softmax function with 10 options.

Training & Development Process

Initial Approach

For both the convolutional and recurrent networks training overall was a lot of trial and error for the initial approach. There were a lot of aspects not initially accounted for, such as the standard scaling, amount of layers, amount of units per layer, and the specific details of implementing the networks. However, through online research to figure out where we were going wrong, we were able to troubleshoot the issues. Figuring out how to best compile the network was also an area for development, but we eventually landed on using the Adam optimizer and the categorical crossentropy for the loss function. Finally, test accuracy results started as low as 20% and as we continued to work on it, we achieved a high of 49% for the convolutional and 53% for the recurrent, each with a total of only 8 epochs necessary. With epochs higher than 8, the training would not only take longer, but would vastly overfit to the training data, and made the actual testing accuracy lower than with less epochs. In all, it was clear the initial approach just wasn't good enough if it could only achieve about 50% accuracy with only six genre options.

Refined Approach Through Use of .wav Song Files

When we started to redo the development and training in the refined approach, we took a lot from what we learned in our initial approach and came into it much more comfortably. We tried various learning rates to try and tune it in, and found that there were only minor differences between tested learning rates like 0001, 0.0005, 0.01, and 0.001, but 0.001 worked best. Batch size choice was also an area of little relative importance as sizes 32, 64, 128, 256 seemed to not change accuracy much, but larger sizes did decrease training time. As for our number of epochs, unlike the initial approach, 50-60 epochs was the sweet spot for highest accuracy without overfitting. To combat overfitting, we made sure to graph test accuracy and validation accuracy to keep overfitting in check when developing the network architecture. As for training time and accuracy, we found the convolution neural network to only take about 100 seconds to train and test for 60 epochs, with a validation accuracy of 75%. The recurrent neural network took substantially longer to run at about 12 and a half minutes, but had an outstanding validation accuracy of 79%. Overall, we took what we learned in training the initial network and applied our learnings and techniques to the .wav files to achieve much higher accuracy with even more genres (10 genres compared to 6 in the initial).

Challenges/Obstacles & Solutions

Some challenges we faced during this project and their solutions are as follows:

Limited Information in Initial Data:

The initial dataset sourced from Spotify's API provided only numerical features, limiting the depth of information available for training. Spotify's numerical data on different genres still had high variability even within the same genre, making it challenging to establish clear categories to fall into.

The solution was drastic, but transitioning to a dataset of .wav files allowed us to capture more intricate details of music through high-resolution audio samples, and high sample rates allowed for much more effective information to be obtained. In the end our validation accuracy increased a lot with even more genres so it was very effective.

Optimizing Neural Network Architectures:

Fine-tuning the architecture of both the convolutional and recurrent neural networks was a complex task, especially in the initial approach where a satisfactory accuracy seemed impossible.

Playing around with different parameters and introducing multiple layers allowed us to reach much higher accuracies. For example, in the convolutional network, the 3 sets of convolutional layers with maxpooling and batch normalization improved stability, training time, and efficiency.

Overfitting and Model Generalization:

Balancing overfitting and model generalization was a persistent challenge, particularly evident in the initial training stages.

Implementing techniques such as dropout layers and adjusting the number of epochs during the refined approach allowed us to strike a better balance, preventing overfitting while achieving higher validation accuracies. Also, graphing the test accuracy against the validation accuracy helped us see where disparities between the two were in the development stage, showing significant overfitting.

Transition to MFCCs and Feature Extraction:

As a group, we were very unfamiliar with MFCCs and audio processing in general. We knew this would be the step to take to really make an effective classifier, but learning all about signal processing was certainly a challenge.

Using the librosa library turned out to be an exceptionally effective way to generate the necessary MFCCs. It was able to process the signals for us, and do the necessary calculations for the short-time fourier transform, all with easy to read documentation.

Results, Observations, & Conclusions

Overall, we are very pleased with our results. 79% accuracy on a very complex and rigorous problem such as genre classification is better than we were hoping for, and we are proud that this was able to work in general.

It's obvious raw waveform data of music is certainly enough to classify music to a high degree, but there was definitely more we could have done with the project given much more time and resources. Ideally, if we had a way to combine our MFCC data in conjunction with the Spotify API numerical data, they could work hand in hand to classify genres more accurately, and possibly even discern sub-genres. Additionally, if we were able to retrieve waveform data for an even larger dataset, i.e. 1000 songs per genre instead of 100, we could have potentially had an even better classifier.

Audio classification and processing plays an essential role in various areas such as music streaming, speech recognition, and environmental monitoring. It enhances user experiences on streaming platforms, enables accurate speech interpretation, and aids in identifying specific sounds. While of course our model is not perfect, it is certainly a major pillar in the foundation for advances in all areas of audio processing.

In conclusion, our journey from the initial exploration of numerical features to the refined approach with .wav files and MFCCs resulted in a significant advancement in music genre classification accuracy. As a team, we all learned a lot more on how neural networks are applied, and we especially recognize all of the nuances in the audio processing world. Learning about neural networks doesn't compare to building one and seeing it improve as we optimize it. This project acted as a great way for us to learn and tackle a new problem, but also helped us learn the foundation for further advancements in the ever-changing audio-processing world.