Title: Music Genre Classification

Who: Daniel Park, Bill Ma, Prithu Dasgupta (Team “boolean”)

Introduction We are implementing a previously written paper from an advanced Stanford course in Machine Learning and Deep Learning. The main effort of this project is to attempt to classify the musical genre of audio clips using different Deep Learning techniques that we have been taught in CS1470. The main questions of our project lie in two areas. First is data extraction, since computing features for an audio file can be considered fairly complicated. Two approaches that the paper proposes are segmented and sequential, where the distinction is simply featurizing by predetermined periods within the thirty second long audio file or by the entire file itself in successive timesteps. The second main question is once our data is preprocessed which learning techniques will be using and comparing accuracies. Architectures we have considered, at a broad level, are fully-connected networks, convolutional networks, RNNs. The fully-connected network and CNN will take in segmented audio clips, while the RNN will examine the audio features in succession. We chose this paper because we are all interested in how large scale music streaming platforms perform song recommendations, and at its core this corresponds to genre classification.

Methodology We use the GTZAN Genre Collection dataset. GTZAN contains 1000 audio clips in .wav format each 30 seconds in length. These .wav files are considered our inputs and the labels are one of ten predetermined genres. We will use feature extraction to bring these .wav files into a vector space. The paper we reference uses the librosa Python library for feature extraction, which we will use as well.

We tested two methods of feature extraction that are built in to the Librosa library. These are known as Mel-frequency Cepstral Coefficients (MFCC) and Chromograms. Given a .wav file, these create a matrix of dimensions (number of time steps x 20), where there are 20 different audio coefficients for each time step. These coefficients relate to the pitch, bass, and intensity. We experimented with both metrics, but as the plot below displays, the genres are better partitioned by the MFCC, which is what we used for our models.

Once the data was featurized using Librosa and MFCC, we then segmented it based on what model it was being passed onto. For our fully-connected network and CNN, we would take each second audio clip and split it into segments to increase the number of data points. For instance, our data set consists of 1000 thirty second clips. Splitting each clip into half second intervals and calculating MFCC for each of these intervals, gives us 60 data points for each audio clip, or 60,000 data points in total. For each segment, we would take the average of each of the 20 audio coefficients for that segment, resulting is a column vector of size 20. For RNN, however, we were having trouble getting good results with our feature extraction, so we actually added Chromogram analysis from Librosa as another feature in addition to MFCC; we averaged these values over segments of songs before feeding them into our RNN. Finally, we used a random train-test split of 80 percent to 20 percent.

Now that are data was preprocessed for each of the three models, we can list our model architectures.

Our Fully-Connected Network consisted of:

- Dense layer of size 300 with ReLU activation
- Dense layer of size 150 with ReLU activation
- Dense layer of size 10 with softmax activation

Our CNN consisted of:

- Dense layer of size 512 with ReLU activation
- Batch Normalization
- LeakyReLU with alpha of 0.2
- Reshape layer of (4, 4, 32)
- Convolution layer with 64 filters of size 2 by 2 and stride of 1 and “same” padding with ReLU activation
- Convolution layer with 128 filters of size 2 by 2 and stride of 1 and “same” padding with ReLU activation
- Convolution layer with 128 filters of size 2 by 2 and stride of 1 and “same” padding with ReLU activation
- Flatten layer
- Dropout layer of 0.5
- Dense layer of size 150 with ReLU activation
- Dense layer of size 64 with ReLU activation
- Dense layer of size 10 with softmax activation

Our RNN consisted of:

- Dense layer of size 256
- Batch Normalization
- Leaky ReLU with alpha of 0.2
- Reshape layer of size 16 by 16
- LSTM layer of size 128 with dropout of 0.05 and recurrent dropout of 0.35
- LSTM layer of size 32 with dropout of 0.05 and recurrent dropout of 0.35
- Dense layer of size 10 with softmax activation

Challenges The primary issues we ran into had to deal with our preprocessing of the sound files. With Keras, making the networks themselves was very straightforward, but early on we found that the models described in the paper weren’t training very well with the features extracted from the sound files we initially used, especially for our RNN. We initially implemented the benchmark features described in the paper, and because these weren’t training very well we attempted to implement the extended features. However, we weren’t sure what their “covariance features” mentioned in the preprocess portion meant, so we had

to find a way to extract features that would actually train on our RNN. Through research of other papers online, we decided to try appending Chromogram features to our existing features and averaging all features over 12 periods per sound file, which ended up achieving much higher accuracy than the benchmark features.

Another challenge we had was the building of a model that achieved the accuracy desired. This relied a lot on the hyperparameters. In initial tests, the accuracy was nowhere near the target. To counteract this, we tried changing a lot of hyperparameter values before changing the model. This often helped a lot more than expected, but took a fair amount of trial and error (outlined in the results section). By changing these parameters in ways that allowed the model to train for longer or better, we often were able to address the challenge. A few in particular included the batch size, the number of epochs, and the number of segments for each audio file.

Lastly, we had the challenge of defining the models' architectures themselves. This relied less on testing and more on intuition and prior experience with these models. For instance, it made a lot of sense to add dense layers to each of the models, since these would help the model learn better. This improved the results significantly, compared to using only, say, a Conv2D or LSTM. If our initial intuitions were not quite successful, even after changing hyperparameters, we addressed this challenge by reevaluating and restructuring our model using trial and error with ideas we may have to improve how the model learns to classify these audio clips.

Results We have attached runs with different hyperparameters for each model below as well as the accuracy per genre for our CNN. However, our optimal runs for each model in terms of test accuracy were 0.6258 for fully-connected, 0.779 for CNN, and 0.8379 for RNN. In terms of accuracy per genre for CNN, seemingly Classical and Metal performed the best, while Rock and Country performed the worst.

Fully-Connected

Batch Size	Learning Rate	First Dense Size	Second Dense Size	Third Dense Size	Epochs	Accuracy
25	0.01	30	10	N/A	1	0.292
25	0.01	150	10	N/A	1	0.2972
25	0.01	300	10	N/A	1	0.3058
25	0.01	300	150	10	1	0.33
25	0.01	300	150	10	1	0.345
100	0.01	150	64	10	10	0.5545
100	0.01	300	150	10	10	0.6258

CNN

Batch Size	Learning Rate	First Conv2D Size	Second Conv2D Siz	Third Conv2D Size	Dropout	First Dense Size	Second Dense S	Third Dense Siz	Epochs	Accuracy
25	0.01	64	64	64	0.35	150	10	N/A	1	0.31
25	0.01	64	64	128	0.5	150	64	10	1	0.271
100	0.01	64	128	128	0.5	150	64	10	10	0.5545
200	0.01	64	128	128	0.5	150	64	10	100	0.779

RNN

Batch Size	Learning Rate	First Dense Size	Dropout	LSTM Units	Second LSTM S	Dropout	Epochs	Accuracy
25	0.01	N/A	N/A	64	10	0.35	1	0.4208
25	0.01	N/A	N/A	64	10	0.5	1	0.3792
100	0.01	N/A	N/A	64	16	0.5	10	0.6971
100	0.01	256	0.2	96	32	0.5	100	0.7945
128	0.01	256	0.2	128	32	0.5	100	0.8379

Accuracy per Genre (CNN)

Genre	Accuracy
Rock	0.5142857143
Reggae	0.7673267327
Pop	0.8505154639
Metal	0.9157894737
Jazz	0.7258883249
HipHop	0.6470588235
Disco	0.7512437811
Country	0.585492228
Classical	0.9275362319
Blues	0.7178217822

There are some observations after finding these results that are noteworthy. For one, regarding the accuracy changes, we noted several differentiators. In many tests, there was a large jump in accuracy after an increase in the number of epochs. This is natural due to greater training opportunity, but is still surprising because it highlights how much of a difference this change makes. Another factor that made a difference was the strength of

dense layers in this type of audio classification. For instance, in RNN adding a dense layer to the model helped significantly (more so that an epoch change could help). Lastly, there was an observation on the types of music that were well classified. Namely, the two highest accuracies came in Classical and Metal, as mentioned previously. This makes a certain amount of intuitive sense, since these forms often might have instruments or tones that are very distinct, particularly since we are not dealing with data points of long length due to preprocessing.

Reflection We found this project to be very insightful and were actually very surprised at how strong the results we received were. We found the biggest challenge of our project to be preprocessing. Understanding how to vectorize the .wav files for each audio clip using Librosa was a large portion of our project, since we had to weigh factors such as size of audio clips, number of data points, and segmentation of clips. We had to decide and evaluate the different options for preprocessing, and developed a rationale as to which methods would give the best accuracies. As seen in the results section, our accuracies for all models were greater than our benchmark of sixty percent, fairly significantly. We all thought that the preprocessing portion of our project was the most interesting, since the model architectures were primarily review from the assignments in CS1470. Hyperparameter trials were also fairly time consuming. The main lessons of this project were how to process audio files and how to efficiently test hyperparameters. Finally, we all intend to continue working on this project in the near future. We plan to dive deeper into how audio coefficients are computed and possibly implement a Transformer model. Another possible next step would be to explore options with different preprocessing methods that give different data to train on for these music genre classification models.

Link to Github https://github.com/danielparkdp/MusicGenreClassification