Music Genre Detection With Long Short Term Memory Neural Networks

Yash Manne

Abstract

Casual listeners and experienced audiophiles alike have long pondered what differentiates one type of music from another and what comprises a song. This project scientifically dissected songs, explored how songs can be rendered visually, and classified songs between 10 genres. Feature generation using a combination of Fourier transforms and other methods were essential to developing robust models. With the generated features, the project effectively classified the genre of music using long short term (LSTM) neural networks and compared it against a series of baseline algorithms such as k-NN, logistic regression, and random forests. Overall, the best performing baseline model was the AdaBoost random forest with an accuracy of 70.4% while the LSTM was 57%.

Introduction

Music genres are conventional categories that designate pieces of music as following certain sets of rules or conventions. The degree of separation between genres is often subjective and varies among genres. There is a clear difference between the dulcet tones of cellos in classical music to the headbanging rhythm of metal and rock music. At the same time, it is much harder to distinguish between metal and rock music — some sources even classify metal as a subcategory of rock!

Most music listeners know intuitively what form of music they prefer and which form a given song falls into. This project hoped to provide a glimpse into how the human brain may subconsciously categorize songs and see whether certain known conventional rules can be accurately applied to delineate between genres.

Through the process, a key goal was to learn more about applying machine learning algorithms to audio data and the different ways to transform audio data into a usable format. The project hoped to explore conventional metrics such as the tempo and frequency of each type of note along with more abstract concepts such as spectral centroids. A key part of the project required applications of Fourier-transforms for feature generation.

While basic classifications have been done using similarity scores of feature values between different genres, these methods don't incorporate the time series element involved in music. This project explores long short-term memory networks (LSTM) as an avenue to incorporate this element and generate more accurate, albeit slower prediction models. To test the performance of LSTM models, the project conducted several baseline models using traditional methods such as k-nearest neighbors (k-NN), multiclass logistic regression, random forests, and boosted random forests. Additionally, to show how the memory of the LSTM is especially vital in bolstering model performance, the project also provides artificial neural networks and convolutional neural networks as a control for the deep connectivity of neural network models.

Task Definition

GTZAN is the most-used public data set for music genre classification. The data set comprises 100, 30-second songs from each of the 10 genres. Specifically, the data set tracks “blues”, “classical”, “country”, “disco”, “hip-hop”, “jazz”, “metal”, “pop”, “reggae”, and “rock”. Each song is stored as 22050Hz Mono 16-bit `.wav` files. All 1000 songs were collected between 2000 and 2001 from a variety of sources such as CD and radio to allow for model robustness. This project aims to take these labeled `.wav` files as input and generate models to classify the labels using supervised learning.

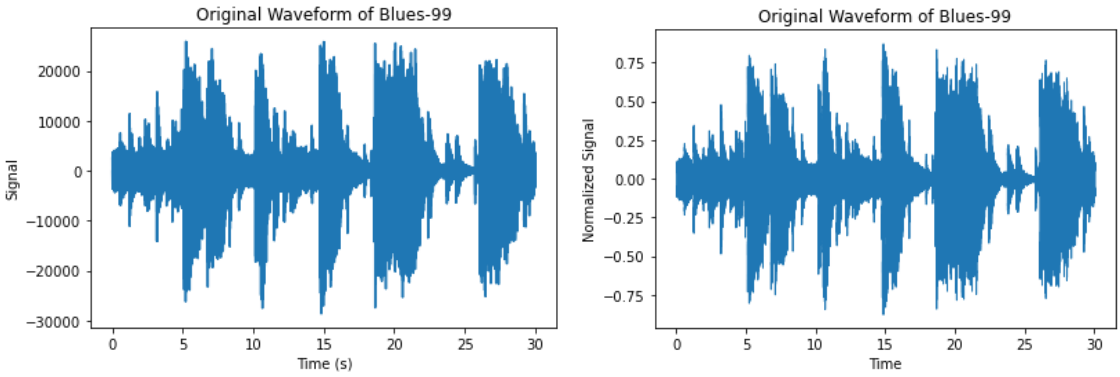


Figure 1: These two plots show how a certain `.wav` file looks for one of the “blues” songs. The data on the left was read in using `scipy.io.wavfile.read` function while the data on the right was read in using `librosa.load` function. Both maintain relative signals but Librosa normalizes the signal to be between -1 and 1 by effectively dividing the raw signals by 2^(16-1) since the data is 16-bit.

Reading in the data set requires the use of specific Python libraries such as `SciPy.io.wavfile` or `Librosa` to read in the `.wav` files as 2-D arrays. The data does not require additional munging before model preprocessing & scaling. As can be seen above, Librosa scales the data between 0-1 while reading in the files and so will be used for this project. Librosa also has multiple built-in functions to generate features from the data. Feature generation is especially important since the raw `.wav` can’t be easily differentiated between one another. The generated features can then be used as inputs for the various supervised learning classification models to generate labels.

Algorithm Definition

This project used multiple supervised classification models ranging from traditional models to complex neural network models. In the set of traditional supervised models, the project will implement k-NN, logistic regression, random forest, and boosted random forest using AdaBoost and gradient boosting.

In the set of neural network models, the project will test feed-forward artificial neural networks (ANNs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), and finally long short-term memory (LSTM) networks, which is a subset of RNNs.

Machine Learning Approach

Feature Generation

As can be seen in **Supplementary Figure 1**, it is very difficult to differentiate between the music of genres when looking solely at the waveforms. As a result, some sets of the features need to be extracted from the data to successfully classify genres.

For traditional models and multi-layer perceptron models, this project investigated 8 features: Mel-frequency cepstral coefficients (MFCC), spectral centroid, spectral bandwidth, zero-crossing rate, spectral roll-off, tempo, chromatogram, and root-mean-square energy value (RMS). All features were generated through the use of the Librosa package.

MFCC is the most commonly used feature extracted from audio data. It performs very well in speech recognition but is also prevalent in applications of genre classification. MFCCs are sets of features that describe the overall shape of the spectral envelope of the music. Calculating it involves 5 main steps:

1. Taking the Fourier transform of (a windowed excerpt of) a signal.
2. Mapping the powers of the spectrum obtained above onto the Mel scale, using triangular overlapping windows or cosine overlapping windows. The Mel scale relates perceived frequency, the pitch, to its actual frequency. Then, the using a Mel filterbank accounts for better differentiation at lower frequencies than higher as is common for human sound perception.
3. Taking the logs of the powers at each of the Mel frequencies.
4. Taking the discrete cosine transform of the list of Mel log powers, as if it were a signal.
5. The MFCCs are the amplitudes of the resulting spectrum.

For this project, 20 MFCCs were extracted for 1293 different frames of the audio files. For traditional models, the means and variances across all time frames were used for each music file. For subsequent models with neural networks, all 1293 x 20 2D arrays were fed as input for each observation.

Spectral Centroid indicates where the “center of mass” of sound is located and is calculated as the weighted mean of frequencies for a given frame. The mean and the variance across all time frames were used as features for each music file.

Spectral Bandwidth is a measure of the spread of frequencies about a spectral centroid at a given point in time. Specifically, the mean and variance across all time frames were used as features for each music file.

Zero-Crossing Rate shows the number of times that the waveform changes from positive to negative or vice versa in a given time interval. The mean and variance across all time frames were used as features for each music file.

Spectral Roll-off is the frequency below which 85% of the total spectral energy lies. This is calculated for each time frame in the music file. The mean and variance across all time frames were used as features for each music file.

Tempo is the pace of the music in beats per minute. Certain genres like country and reggae tend to be slower than other genres such as disco.

A *Chromatogram* is a representation after the entire spectrum is projected down to the 12 semitones of a musical octave for each frame in the music. These features capture harmony and melody in music without being swayed by instrument type and timbre. The mean and variance across all time frames for each semitone were used as features for each music file.

RMS was calculated from a spectrogram to show a representation of energy over time. This was calculated for each time frame and the mean and variance across all time frames were used as features for each music file.

In addition to these features, a Mel spectrogram was used as the sole input for more complicated neural network models. The spectrogram is essentially a heatmap representing the relative intensities of each frequency across time. Mel spectrograms were chosen over linear-scale spectrograms because the Mel-scale better captures the range of frequencies heard by humans. While the spectrogram was originally scaled to the highest decibel such that values ranged between -80 dB and 0 dB, the data was rescaled to 0 and 1 before inputting into neural networks. As can be seen in **Supplementary Figure 2**, patterns are starting to become apparent in the Mel spectrograms of 5 songs from each genre.

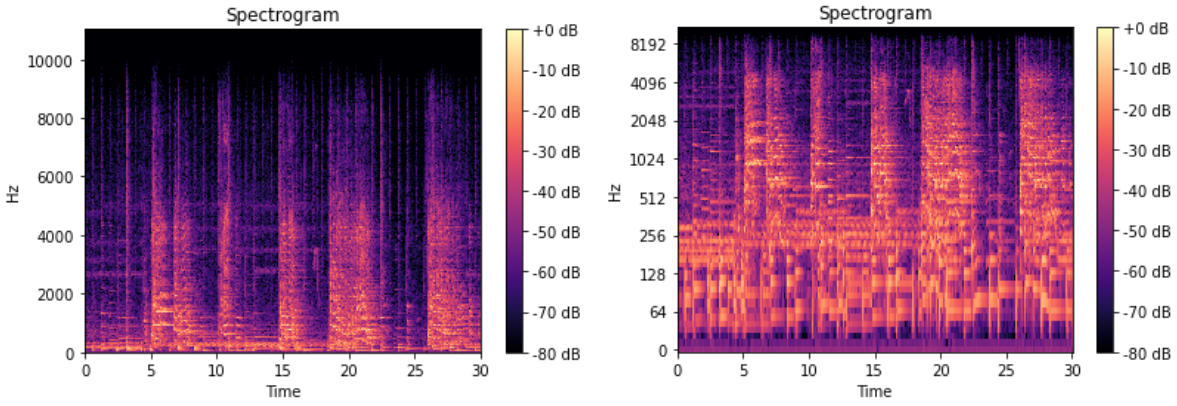


Figure 2: The plots above show the linear and log-scale spectrogram of the normalized audio data for Blues-99. These plots show the separation of the waveform into binned frequencies and indicate which frequencies are present at what time. The separation of frequencies was done using Librosas's `stft` function, which conducts the short-time Fourier transform of specific temporal windows of the audio file and assigns amplitude to a given binned frequency for each of the different windows. Specifically, the parameters were 2048 for the length of the FFT window and 512 audio samples between frames to generate 1025 frequency bins over 1293 window frames. Then, `amplitude_to_db` was used to convert the raw amplitudes to standardize the values between 0 and -80 dB.

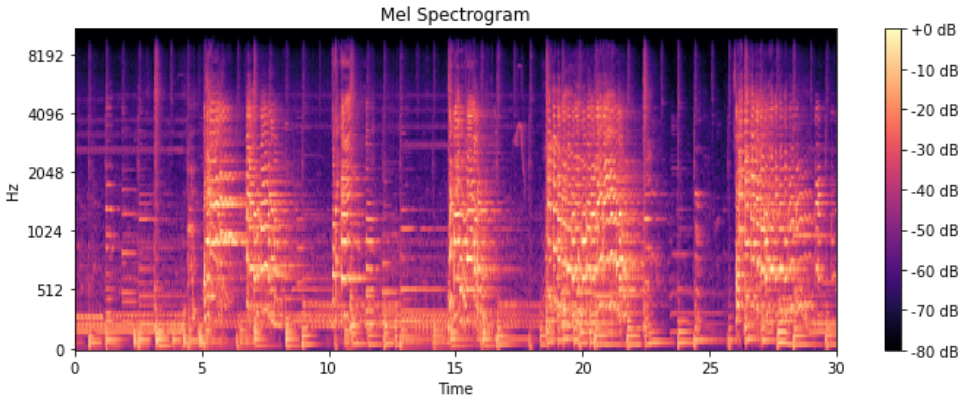


Figure 3: This plot shows the Mel spectrogram of the normalized audio data of Blues-99. It is similar to the spectrograms in Figure 2 except the FFT is done according to the Mel scale, which is a unit of pitch where equal distances sound equally distant to human listeners. As such, only 128 frequency bins are required when using the same FFT parameters as earlier. Mel spectrograms are typically far more useful than linear spectrograms for speech recognition and music genre classification since the linear spectrograms assume equal importance for all frequencies, which is not true for human applications.

Evaluation Strategy

The data set was split into 75% training and 25% testing sets using scikit-learn's `train_test_split` function. The splitting was stratified and shuffled so that equal proportions of each genre were present in each set to avoid model bias. Traditional models were trained on the training set using 5-fold cross-validation while the neural network models were trained on the entirety of the data set. Model performance was solely based on the accuracy of the training set and confusion matrices generated through scikit-learn showed what genres were frequently misclassified.

Hyperparameter Tuning

Hyperparameters for traditional models were trained using scikit-learn's `GridSearchCV` which takes a list of values for hyperparameters and evaluates model performance with 5-fold cross-validation. Hyperparameters for neural network models were changed manually and compared on performance.

Traditional Models

The traditional models of k-NN, logistic regression, random forest, AdaBoost, and gradient boosting were trained using both just the MFCC means and variances as well as the entire feature set listed above. The features were all scaled using scikit-learn's `StandardScaler`, which normalizes each feature by subtracting the mean and dividing the variance of each column. All models were run on interactive Jupyter Notebooks through the High-Performance Computing Cluster (HPCC) with 4 cores and 100 GB of memory. Any training time listed is the total wall time for the `GridSearchCV` to select and train the best model.

For the k-NN, hyperparameters for the number of neighbors and the weighting function were tuned. Specifically, neighbors between 4 and 30 and “uniform” and “distance” functions were tested. The best performance with only the MFCC features was 59.6% accuracy on the test set with 6 neighbors and “distance”, which implements the standard euclidean distance function. The best performance with all features was 66.0% accuracy on the test set with the same number of neighbors and weighting function. The models trained in 4.33 s and 8.55 s, respectively.

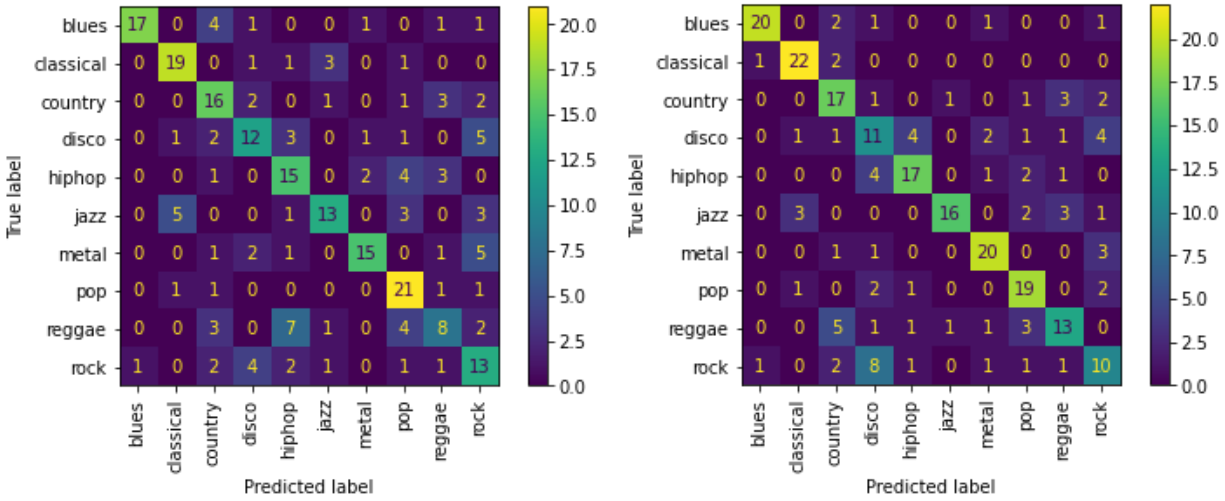


Figure 4: Confusion matrices of test predictions for the best k-NN model trained on only MFCCs (left) and all features (right).

For the logistic regression, hyperparameters for the type of solver and the L2-coefficient were tuned. Specifically, the “saga” and “lbfgs” solvers were tested and L2 regularization parameters were automatically tuned through the `LogisticRegressionCV` function. The best performance with only the MFCC features was 62.8% accuracy on the test set using the “saga” solver. The best performance with all features was 68.0% accuracy on the test set using the same solver. The models were trained in 1 min 32 s and 2 min 21 s, respectively.

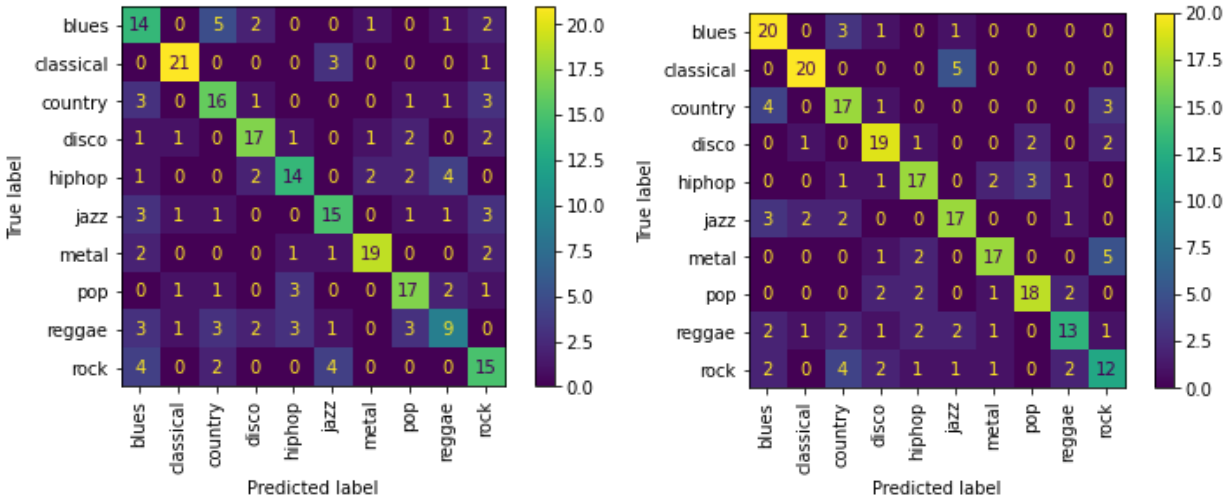


Figure 5: Confusion matrices of test predictions for the best L2-regularized logistic regression model trained on only MFCCs (left) and all features (right).

For the random forest, hyperparameters such as max tree depth, classification criterion, and the number of trees in the forest were tuned. Specifically, depths between 1 and 20 were tested, forest sizes of 100 tree increments between 100 and 500, and Gini impurity and entropy criteria were tested. The best performance with only the MFCC features was 66.8% accuracy on the test set using a max depth of 11, forest size of 400, and Gini impurity. The best performance with all features was 70.0% accuracy on the test set using a maximum depth of 15, 300 trees, and Gini impurity. The models were trained in 6 min 55 s and 10 min 32s, respectively.

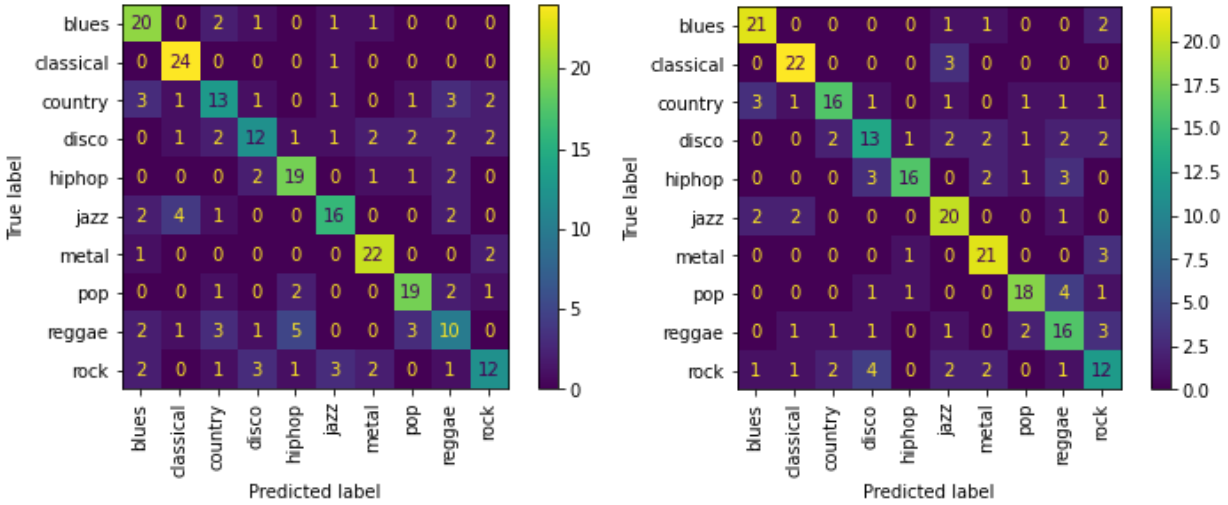


Figure 6: Confusion matrices of test predictions for the best random forest model trained on only MFCCs (left) and all features (right).

For the AdaBoost classifier, performance was very low using base_estimators of single decision trees. As such, a base estimator of a random forest classifier was used. Hyperparameters such as learning rate and the maximum number of random forest models in the forest were tuned. Specifically, learning rates between 0.1 and 2 at increments of 0.1 and the number of models

between 50 and 300 at increments of 10 were tested. The best performance with only the MFCC features was 66.4% accuracy on the test set using a learning rate of 0.9 and 160 as the maximum number of models. The best performance with all features was 70.4% accuracy on the test set using a learning rate of 1.8 and a maximum number of models of 190. The models were trained in 2 min 43 s and 4 min 21s, respectively.

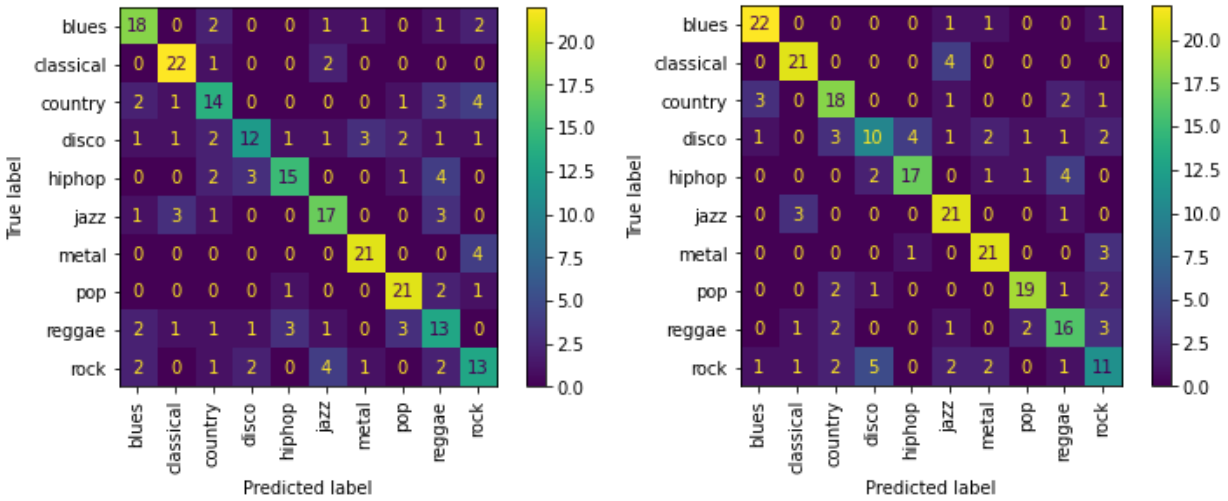


Figure 7: Confusion matrices of test predictions for the best AdaBoost model trained on only MFCCs (left) and all features (right).

For the gradient boost classifier, hyperparameters such as learning rate, and the maximum number of random forest models in the forest were tuned. Specifically, learning rates between 0.1 and 2 at increments of 0.1 and the number of models between 50 and 300 at increments of 10 were tested. The best performance with only the MFCC features was 62.4% accuracy on the test set using a learning rate of 0.5 and 140 as the maximum number of models. The best performance with all features was 69.6% accuracy on the test set using a learning rate of 0.4 and a maximum number of models of 240. The models were trained in 35 min 18 s and 1 hr 8 min, respectively.

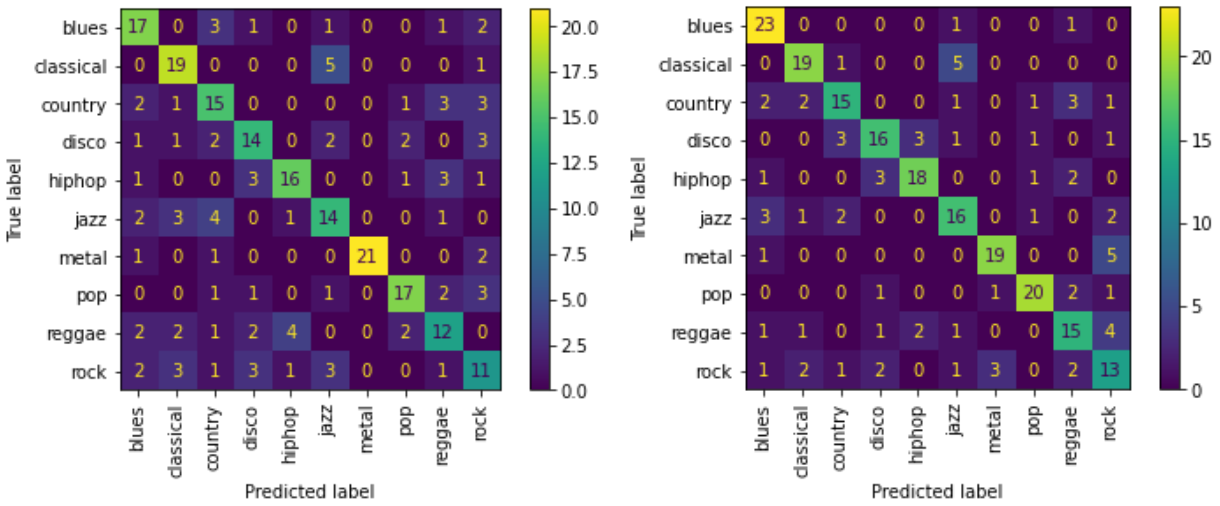


Figure 8: Confusion matrices of test predictions for the best gradient boost model trained on only MFCCs (left) and all features (right).

Overall, it's pretty clear that the best traditional model is the AdaBoost random forest model with all features, having an accuracy of 70.4% and training in 4 minutes. From the confusion matrix, it can be seen that the reggae-hip-hop, jazz-classical, disco-rock, and disco-hip-hop pairs occasionally get misclassified.

Neural Network Models

Once again, all models were trained using interactive Jupyter notebooks on the HPCC used TensorFlow-gpu. The notebooks required 200 GB with 4 cores and 4 GPUs to train ANNs and CNNs and used 300 GB and 8 cores and 4 GPUs to train RNNs and LSTMs. It should be noted that GPUs were successfully used for ANNs and CNNs but dependencies for allocating resources between GPUs failed while training RNNs and CNNs so model fitting was done solely through the 8 CPUs.

All neural networks were trained with either the full MFCC 2D arrays or the full Mel spectrogram. Additionally, the feed-forward ANN was trained with the full feature set. In all neural networks, the Adam optimization algorithm was used for training and all training used categorical cross-entropy from logits as the loss function to optimize since the output of the neural network was the logit odds of each class and not a probability value. All models also had a learning rate of 0.001 as this is what was recommended by TensorFlow.

For all networks overfitting was a consistent issue as the training error was drastically lower than the validation error as the number of epochs increased. As such, dropout layers and batch normalization layers were used to reduce overfitting.

All ANN models were trained with ReLU and SELU activation functions with a batch size of 32 and 100 epochs. Additionally, all dense layers had L2 regularization with lambda = 0.001.

The best model trained with only MFCCs had a testing accuracy of 50.4%. This model had a flattening input layer, 3 dense layers of 512 nodes, 1 dense layer of 128 nodes, and a final output layer of 10 nodes for the 10 genres. Each dense layer had ReLU activation. Additionally, there was a dropout layer with probabilities of 0.01, 0.03, 0.01, and 0.01, respectively, after each dense layer. This model took 39 s to train.

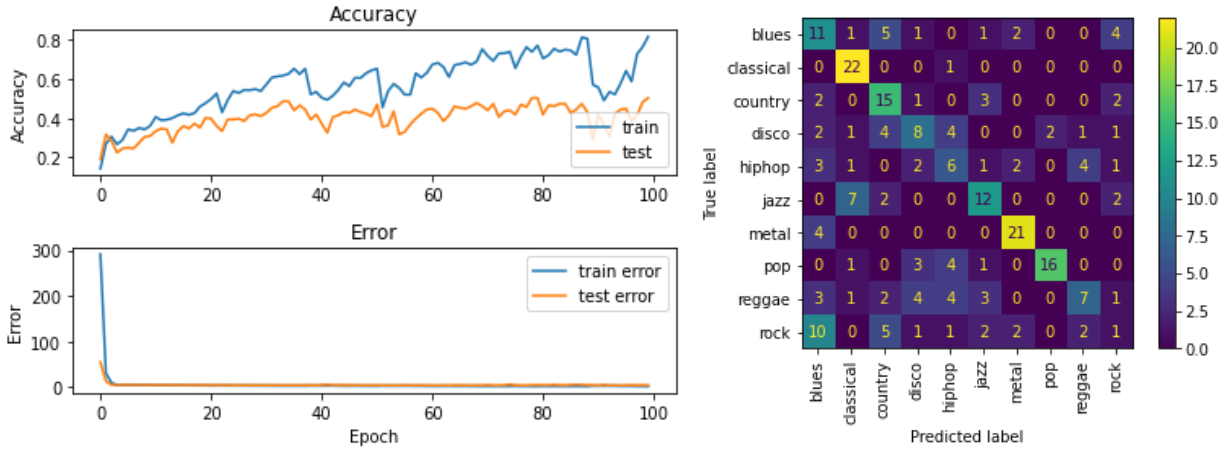


Figure 9: Best ANN model with MFCC input.

The best ANN model with the Mel spectrogram input was 44.1% accurate. This model had a flattening input layer, 4 dense layers of 512, 256, 128, and 64 nodes, and a final output layer of 10 nodes for the 10 genres. Each dense layer had ReLU activation. Additionally, there was a dropout layer with probabilities of 0.01, 0.02, 0.01, and 0.01, respectively, after each dense layer. This model took 8 min 2s to train.

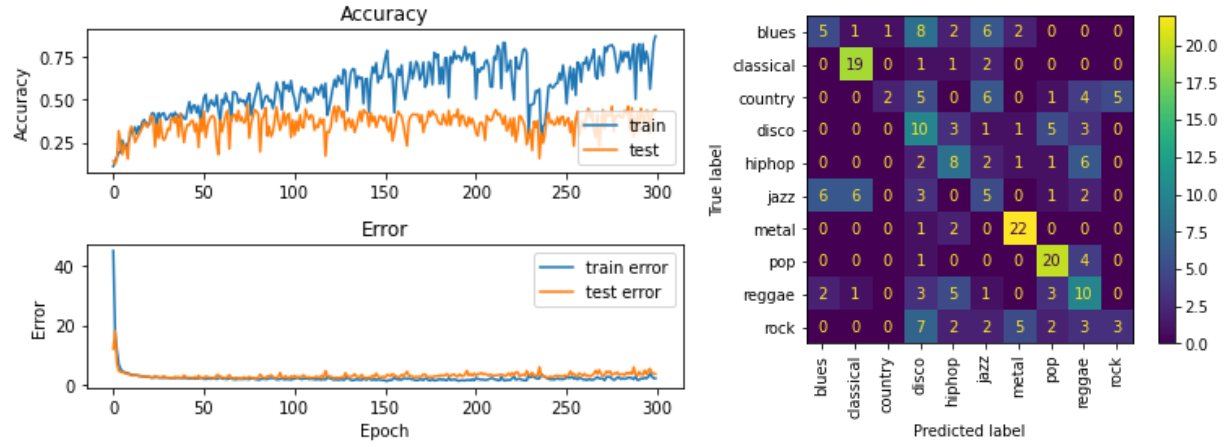


Figure 10: Best ANN model with Mel spectrogram input.

The best ANN model with all features had an accuracy of 74.4%. This model had a flattening input layer, 3 dense layers of 512 nodes, 1 dense layer of 128 nodes, and a final output layer of 10 nodes for the 10 genres. Each dense layer had ReLU activation. Additionally, there was a dropout layer with probabilities of 0.01 after each dense layer. This model took 16.6 s to train.

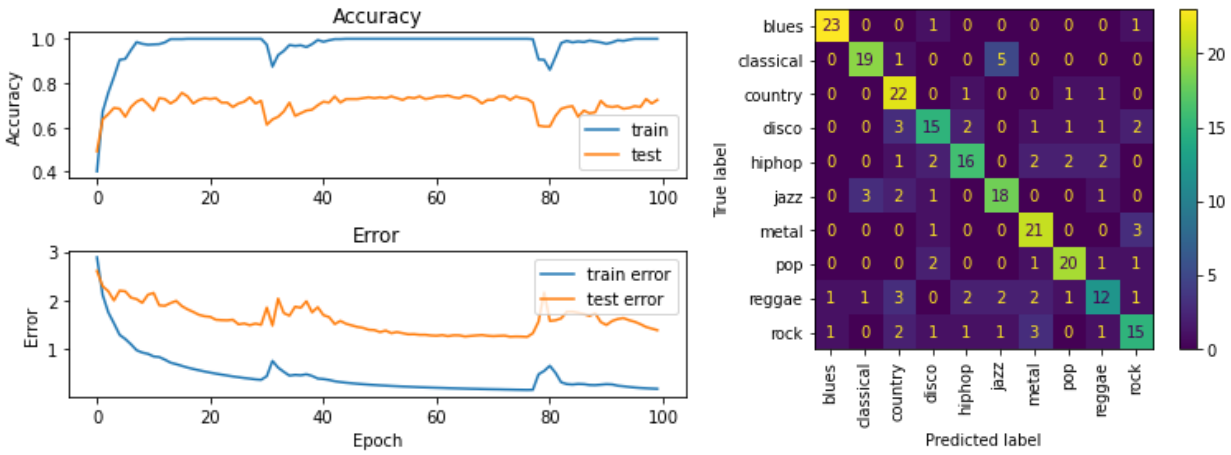


Figure 11: Best ANN model with the entire feature set.

For the CNN models, additional hyperparameters such as stride length, padding, kernel size, and pool size needed to be tuned. Max pooling was used to reduce overfitting, increase robustness to noise, and decrease the complexity of models. To decrease training time, early stopping with patience of 10 was implemented such that training ends if the loss doesn't decrease for 10 consecutive epochs up to a maximum of 100 epochs. Since CNN required 3D input, `np.newaxis` was used to add an additional dimension to the 2D data.

The best-performing CNN model with the MFCC input had an accuracy of 54.24%. This model had 4 2D convolutional layers with 64, 32, 32, and 16 filters, respectively. Each layer had ReLU activation and had kernel sizes of 3, 3, 2, and 1, respectively. Each layer had a stride of 1 and zero-padding. Each layer was followed by a 2D max-pooling layer with pool sizes of 2, 2,2, and 1, respectively, and zero-padding and a default batch normalization layer. Following the convolutional layers, output was flattened and fed into a 64-node, ReLU-activation, dense layer with L2-regularization and a dropout of 0.3 before the final output layer of 10 nodes. This model took 1 min 11 s to train.

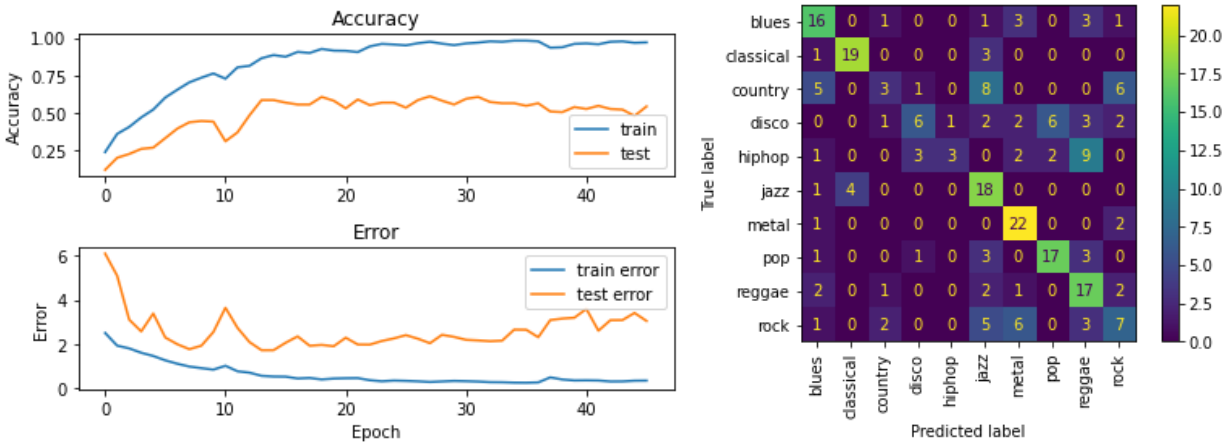


Figure 12: Best CNN model with MFCC input.

The best-performing CNN model with the Mel spectrogram input had an accuracy of 57.20%. This model had 4 2D convolutional layers with 64, 32, 32, and 16 filters, respectively. Each layer had SELU activation and had kernel sizes of 3, 3, 2, and 1, respectively. Each layer had a stride of 1 and zero-padding. Each layer was followed by a 2D max-pooling layer with pool sizes of 2, 2,2, and 1, respectively, and zero-padding and a default batch normalization layer. Following the convolutional layers, output was flattened and fed into a 64-node, SELU-activation, dense layer with L2-regularization and a dropout of 0.3 before the final output layer of 10 nodes. This model took 16 min 3 s to train and used a batch size of 5.

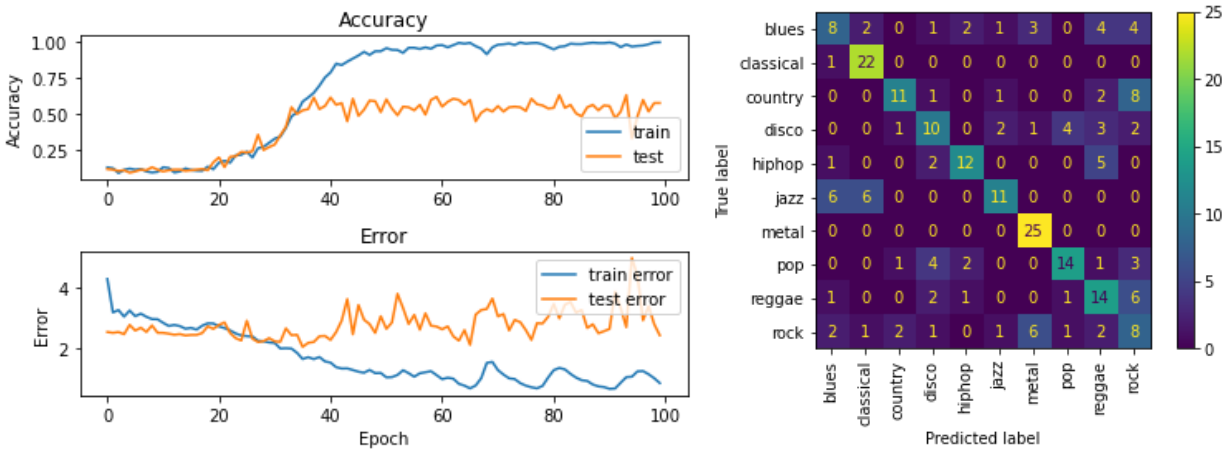


Figure 13: Best CNN model with Mel spectrogram input.

For RNN and LSTM models, the number of units in each layer, the number of layers, and the dropout probabilities were varied.

Simple RNN models performed terribly on the Mel spectrogram input and were well below the accuracy of a baseline single-layer perceptron model (47.5%). LSTM models also only achieved a maximum accuracy equal to the single-layer perceptron model and as such, will not be shown here.

Likewise, simple RNN models with MFCC input had an accuracy well below the accuracy of a baseline single-layer perceptron model (45.7%) and they will not be shown here.

The best-performing LSTM model on the MFCC input had an accuracy of 55.08%. This model had two LSTM layers with 64 units followed by a ReLU-activated dense layer with 64 nodes and a final output layer of 10 nodes. The model was very overfitted and took close to 20 minutes to train and a total CPU time of 1 hour.

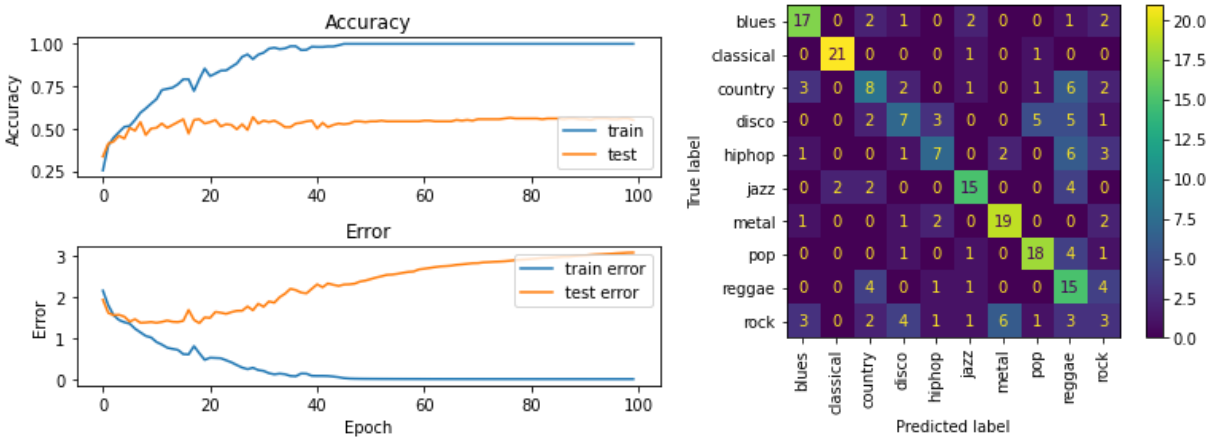


Figure 14: Best LSTM model with MFCC input.

Of all the neural network models, the LSTM model had the best performance (55.08% accuracy) with the MFCC input while the CNN model had the best performance with the Mel spectrogram input (57.20%). However, the ANN model with all features had the best overall performance with an accuracy of 74.4%! From the confusion matrix, it can be seen that the jazz-classical and metal-rock pairs occasionally get misclassified.

Summary, Conclusions, and Outlook

Overall, the two keys goal of this project were to learn how to extract features from audio data for use in modeling and to was to effectively classify musical genres. The first goal was a definite success as the project detailed the use of over 8 different features that were used in subsequent models. The second objective was a less definite success. The project was able to successfully classify genres with 70% accuracy but this isn't much better than a simple k-NN model, which classified at a 66% level. The original hope was to show that LSTM could drastically improve classification accuracy. However, it can be seen that the best overall model was an ANN with all features and this model performed 4% better than a traditional random forest model using the same feature set. Additionally, the traditional random forest method was able to get an accuracy of 66.8% with just the MFCC data compared to the best accuracy of 55.1% from the LSTM. This doesn't support the original hypothesis. To be fair, LSTM performed better with MFCCs than any other neural network model.

A likely reason for the poor performance of neural network methods may be attributed to the relatively low amount of data. A future workaround would be to split each audio file into smaller increments and increase the total number of data points. Additionally, it might be worthwhile to explore different learning rates and optimizers for each model training. It could also be useful to combine whole MFCC and spectrogram arrays as a 3D input for future models. Finally, it would be interesting to experiment with different Fourier-transform parameters while creating the MFCCs before model training.

On the whole, the project was informative and provided much-needed experience in working with TensorFlow to implement deep neural network models.

Bibliography

Data:

- G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," in IEEE Transactions on Speech and Audio Processing, vol. 10, no. 5, pp. 293-302, July 2002, doi: 10.1109/TSA.2002.800560.

Background Information:

- Aurélien Géron (2019). H*ands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems* (2nd ed.). O'Reilly.

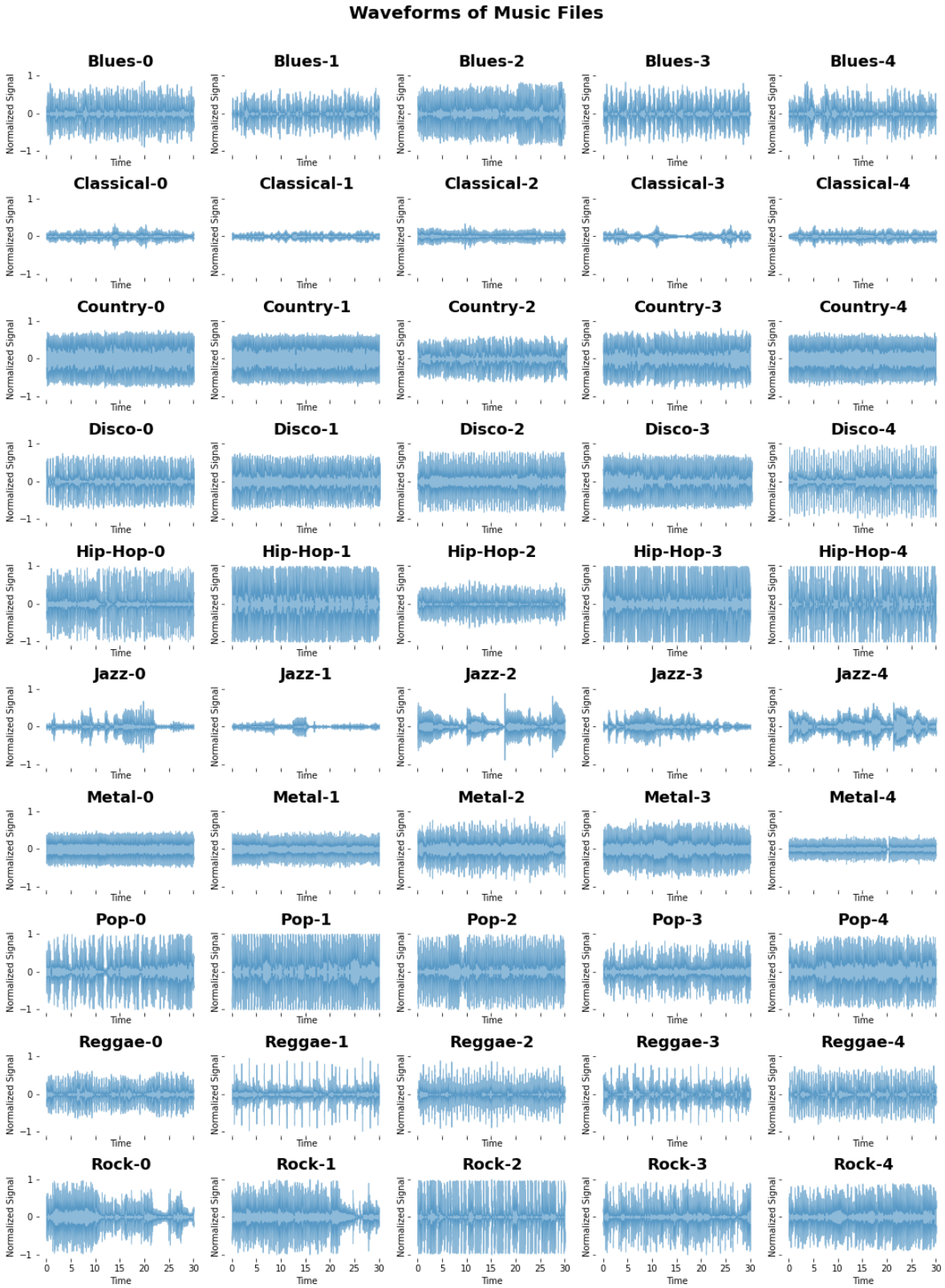
Software:

- Brian McFee, Alexandros Metsai, Matt McVicar, Stefan Balke, Carl Thomé, Colin Raffel, Frank Zalkow, Ayoub Malek, Dana, Kyungyun Lee, Oriol Nieto, Dan Ellis, Jack Mason, Eric Battenberg, Scott Seyfarth, Ryuichi Yamamoto, Viktor Andreevich Morozov, Keunwoo Choi, Josh Moore, … Thassilo. (2022). librosa/librosa: 0.9.1 (0.9.1). Zenodo. https://doi.org/10.5281/zenodo.6097378
- Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
- TensorFlow Developers. (2022). TensorFlow (v2.8.0). Zenodo. https://doi.org/10.5281/zenodo.5949125

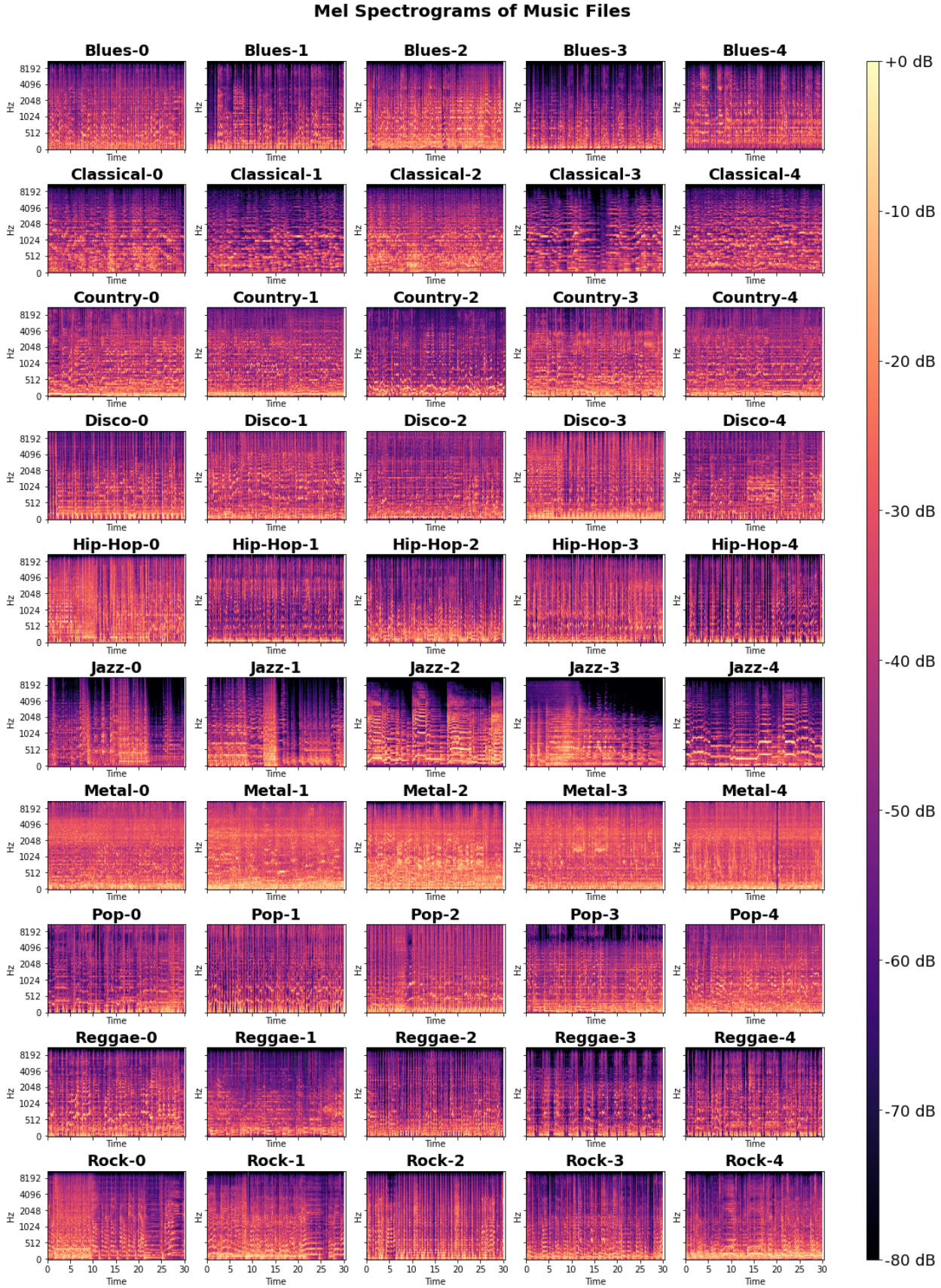
Additional Resources:

- https://towardsdatascience.com/how-i-understood-what-features-to-consider-while-training-audio-files-eedfb6e9002b
- https://towardsdatascience.com/visualizing-audio-data-and-performing-feature-extraction-e1a489046000
- https://towardsdatascience.com/extract-features-of-music-75a3f9bc265d
- https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8
- https://www.analyticsvidhya.com/blog/2021/06/mfcc-technique-for-speech-recognition/
- https://analyticsindiamag.com/a-tutorial-on-spectral-feature-extraction-for-audio-analytics/
- https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd
- https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53
- https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Supplementary Figures



Supplementary Figure 1: Waveforms of 5 samples of each genre.



Supplementary Figure 2: Mel Spectrograms of 5 samples of each genre.