Abstract

In this paper we analyzed the classification of music across 10 different genres. The dataset used was the GTZAN dataset which included 1000 30 second clips of music within the 10 different genres. Our best CNN model achieved an accuracy of 67.59% on the MFCC images. The best DNN model achieved an accuracy of 90.78% from the data in the created CSV containing various features. The team learned how to manipulate audio data to develop neural network classifiers that could be used for future recommendation and playlist curation models.

Introduction

The data science team this week was tasked with looking into building a model that could classify a songs music genre. This has a lot of business applications that can be seen in other companies such as Spotify. This could be the start of building automatic genre playlists on our platform and then expand into recommending similar songs and artists based on a user's tastes in a genre. The dataset we used comprised of 1000 30 second clips of songs within 10 different genres: 1. Metal 2. Rock 3. Blues 4. Classical 5. Hip Hop 6. Country 7. Pop 8. Disco 9. Jazz 10. Reggae. We took a couple different approaches in building a model to classify these song structures. We used CNNs to analyze the Mel-Spectrogram of the song to identify the different genres. We also used the CSVs provided by Kaggle to build a dense neural network that contained 57 data points of each clip like temp, pitch, and Mel-frequency cepstrum (MFCC) data. The best network used the data provided by Kaggle that included all of the data points for the songs spit into 3 second clips and was able to achieve an accuracy of 90.77% on the test data set. The best CNN model was able to achieve an accuracy of 67.59% on the MFCC images created from the 30 second clips.

Literature Review

The paper "**C**ombining visual and acoustic features for music genre classification", talks about converting data into Mel spectrogram images and using acoustic features to classify the genre of Music. They were able to use SVM and Adaboost to combine the audio features extracted and the image data from the conversion of the image to a Mel spectrogram. This paper led the team to research the Mel spectrogram approach to convert the audio data to an image that a CNN model could use.

Methods

Review research design and modeling methods

The GTZAN dataset consists of 1000 30 second audio clips in 10 classes, with 100 audio clips per class. The initial setup is 900 audio clips for training and 100 audio clips for testing. We also separated the training data and set aside 100 audio clips for validation, leaving 800 audio clips

for testing. Due to the relatively small amount of data, we also experimented with splitting files into different size chunks, like 3 10 second clips instead of one 30 second clip. This helped increase the amount of data to train our system. All audio files have labels for the images that correspond with 1 of the following 10 classes: 1. Metal 2. Rock 3. Blues 4. Classical 5. Hip Hop 6. Country 7. Pop 8. Disco 9. Jazz 10. Reggae. The 10 classes are diverse and a good challenge to experiment with audio classification.

For modeling solutions to the GTAN classification data problem we experimented with deep dense neural nets and convolutional neural nets. Each of our models were deep, meaning that they contained more than one hidden layer. A convolutional neural network learns features by scanning over the image using a window called a kernel that is then summarized during the pooling step causing the features to be down sample and keeping the most prominent ones. For our models we used max pooling techniques which keeps the largest values of the feature map. Typically, the model is then flattened and passed to a dense layer before being passed to an activation function like softmax for classification. An example architecture for a CNN can be seen in the picture below.

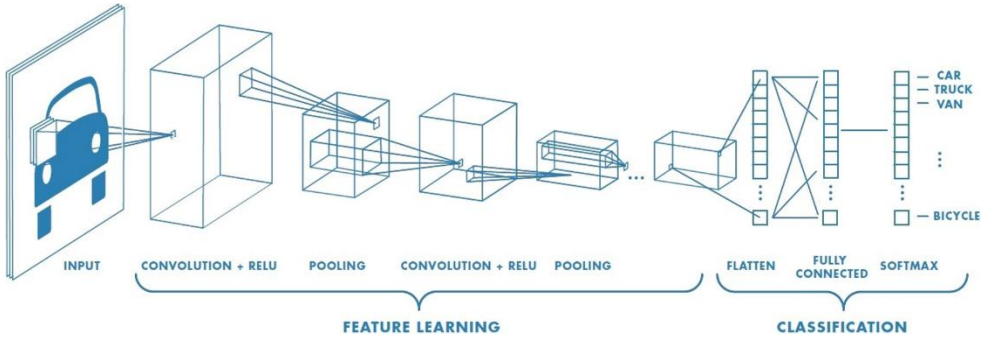


Figure 1: CNN Model Architecture

Implementation and Programming

These models were implemented using the python programming language and several libraries. The main libraries used were pandas for data cleaning and managing, keras tensorflow was used to build the neural networks, sklearn for scaling and confusion matrix, numpy to handle arrays, and matplotlib to build the plots. The sequential class was used to build the neural networks with different activation functions, number of layers, and number of nodes. For regularization techniques we used tensorflow's BatchNormalization and dropout methods to prevent overfitting. The compile method was used with a loss function of SparseCategorialCrossentropy because we did not one hot encode the data. The evaluate method was used to test the accuracy of the model after being fitted on the test data. For dealing with the audio data we also used the package librosa. This package gave use the ability to transform the data into visual images like the Mel-spectrogram and MFCC which allowed us to feed into our CNN model. We also used Ipython to play the audio files inside the jupyter notebooks. The development environment was done in jupyter noteboks and the code can be found attached to this report in pdf format.

We approached the problem by conducting several experiments. For the CNN models we approached it using a Mel-Spectrogram image of the entire 30 second clip and we used MFCC images with segments of 30,3,10,5,and 6 seconds. We then added regularization to the model using the MFCC images split into 6 second intervals, as this was the best performing model.

Data Preparation

One of the first steps in preparing the data for the CNN models was to convert the audio file to a spectrogram image. TO do that we loaded the audio file into librosa and was able to convert it into an image.In exploring the data we looked at several different plots to see which ones would be the most useful. We used a sampling rate of 22000 HZ as this would cover up to 10000 HZ which sufficient for the sounds people hear in music. First, we looked at the wave plot of a blues song. This plot can be seen in the appendix. We also looked at a harmonic and percussive graph of the blues song as can be seen the appendix. These plots were good but did not seem to be enough to classify between genres. So next we created a spectrogram plot as can be seen in the figure below. A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time.

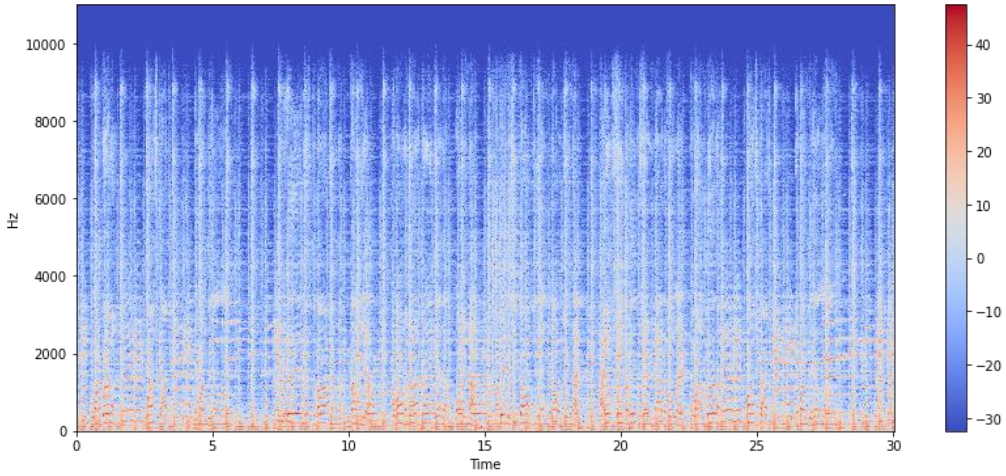


Figure 2: Spectrogram of blues song

From this spectrogram we could see that most of the useful information happened at the lower frequencies towards the bottom of the graph. We then created a Mel-spectrogram plot which is on the log scale to help closer visualize what humans hear. This can be seen in the image below from the same song. Also in the appendix is a Mel-spectrogram for several more songs and genres other than this blues song.

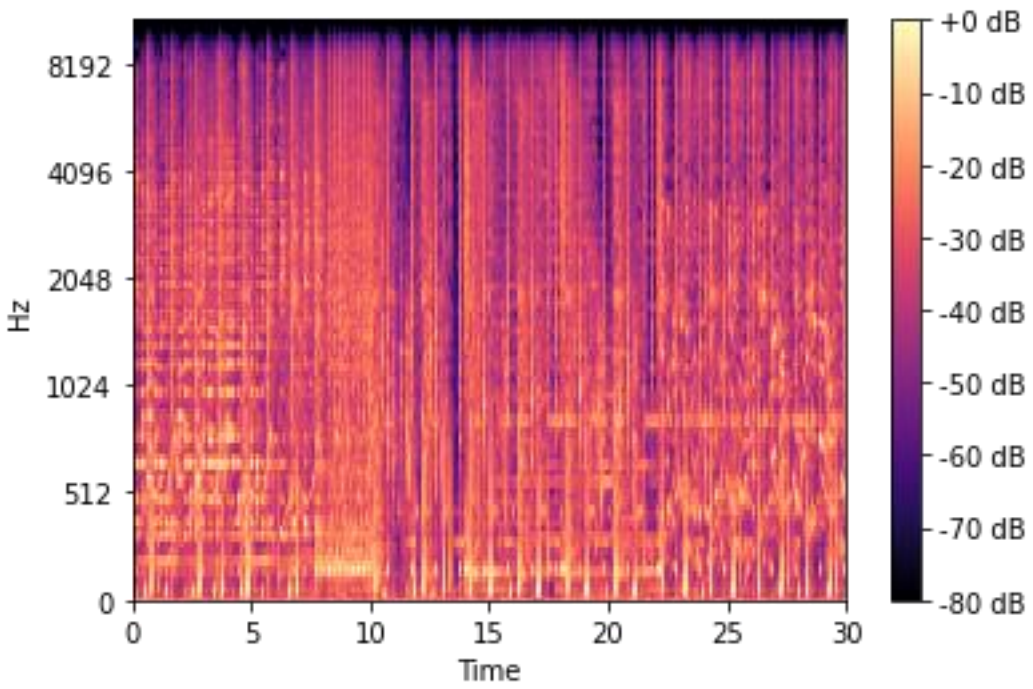


Figure 3: Mel Spectrogram of blues song

To compress this even more we used MFCC. MFCC is computed by taking the Mel value then logging it and taking the discrete transform. This is used a lot in speech recognition. This form was used so we could further split the images into smaller time slices to create more data. This was helpful as our dataset was relatively small for building a neural net, so splitting the audio clips helped increase the amount of data.

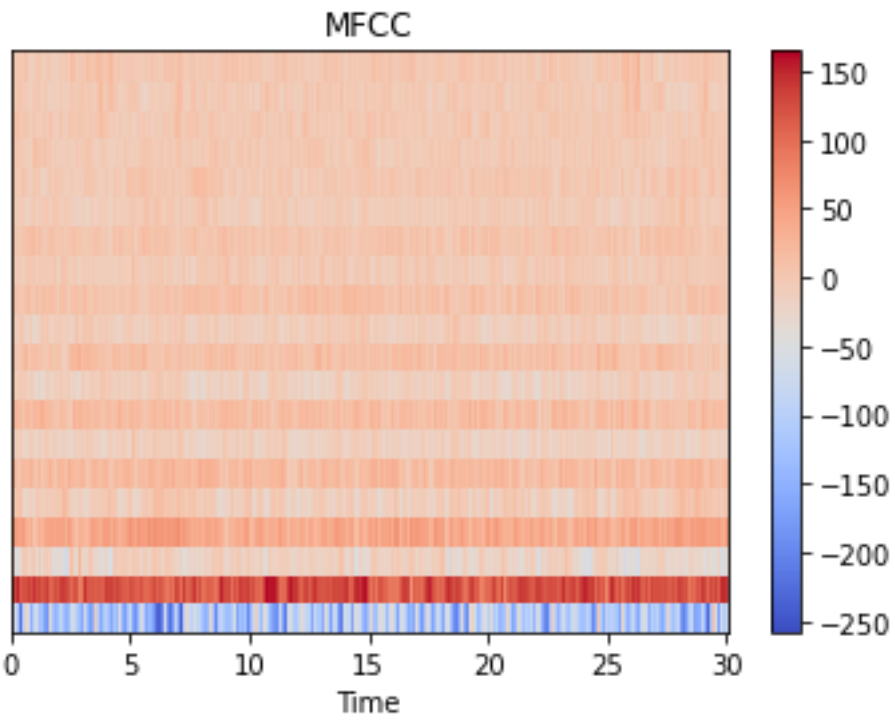


Figure 4: MFCC of blues song

Results

In this section we will go over the results of the neural net models. We first reviewed CNNs, then DNNs, then using regularization like dropout and batch normalization for the best performing CNN model. Some of the constants between models is an optimizer of adam and loss function of SparseCategoricalCrossentropy.

Experiment 1

For the first experiment we broke the MFCC images into 6 parts for each song being 5 seconds each. We created a deep convolutional neural network to analyze the image data. The first layer was a conv2d layer of 256 units, 3x3 kernel size, and relu activation function, then a max pooling layer , then another conv2d layer of 512 units, 3x3 kernel size, and relu activation function, then a max pooling layer, then another conv2d layer of 1024 units, 3x3 kernel size, and relu activation function, then a max pooling layer, followed by a flatten layer for input into a 512 unit dense layer, with finally an output layer of 10 units using a softmax activation function.

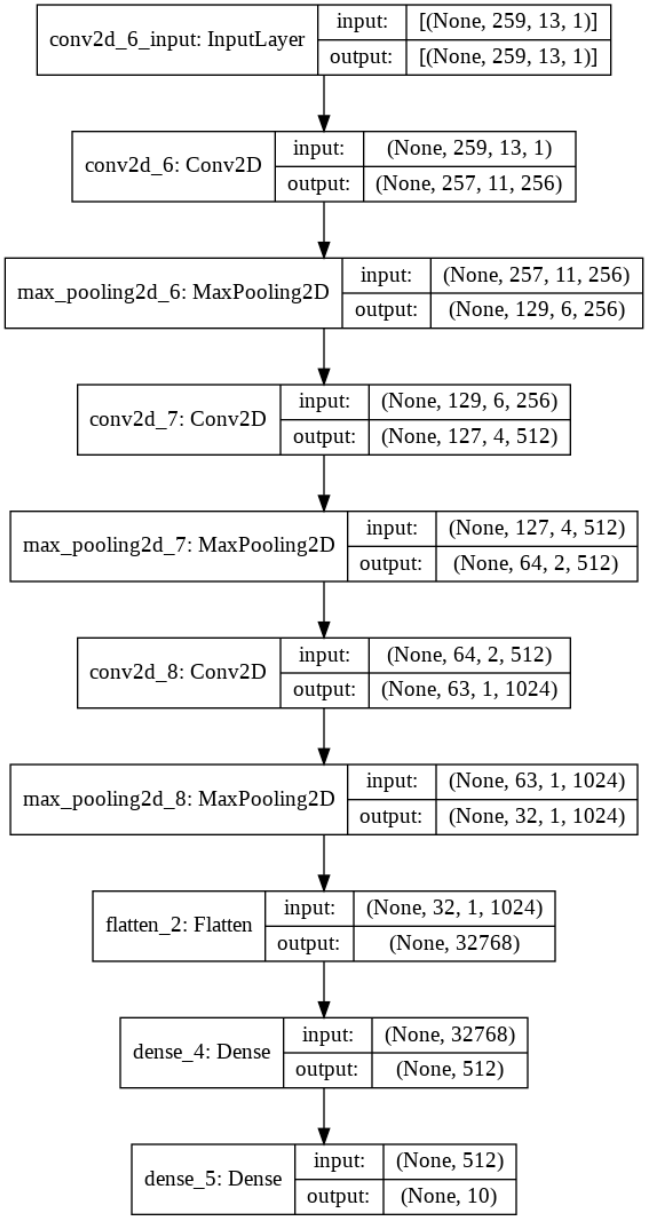


Figure 5: Neural Net Architecture of experiment 1

With this architecture and data preparation the accuracy of the model was 67.22% on the validation data and 65.21% on the test data.

Experiment 2

For the second experiment we kept the exact same neural net architecture as experiment 1. The difference was in the data preparation. This time we did not split the mfcc image into part and kept the whole 30 second image for each song. So this dataset was 1000 images, as compared to experiment 1s 6000 image set. Unexpectedly this model performed worse than experiment 1. With this architecture and data preparation the accuracy of the model was 57.78% on the validation set and 55.02% on the test set.

Experiment 3

This experiment also had the same architecture as experiment 1. However, for this experiment we used the Mel spectrogram image to feed into the CNN model. We converted all 1000 songs into Mel spectrogram images and trained the model. This model achieved an accuracy of 51.26% on the validation set and 50.02% on the test dataset.

Experiment 4

For the fourth experiment we kept the exact same neural net architecture as experiment 1. The difference was in the data preparation This time we did split the MFCC image into 10 parts of 3 seconds, resulting in a dataset of 10000 images. With this architecture and data preparation the accuracy of the model was 62.74% on the validation set and 62.26% on the test set.

Experiment 5

For the fifth experiment we kept the exact same neural net architecture as experiment 1. The difference was in the data preparation. This time we did split the MFCC image into 3 parts of 10 seconds, resulting in a dataset of 3000 images. With this architecture and data preparation the accuracy of the model was 55.19% on the validation set and 55.33% on the test set.

Experiment 6

For the sixth experiment we kept the exact same neural net architecture as experiment 1. The difference was in the data preparation. This time we did split the MFCC image into 5 parts of 6 seconds, resulting in a dataset of 5000 images. This model performed the best out of the previous experiments. With this architecture and data preparation the accuracy of the model was 69.56% on the validation set and 62.17% on the test set.

Experiment 7

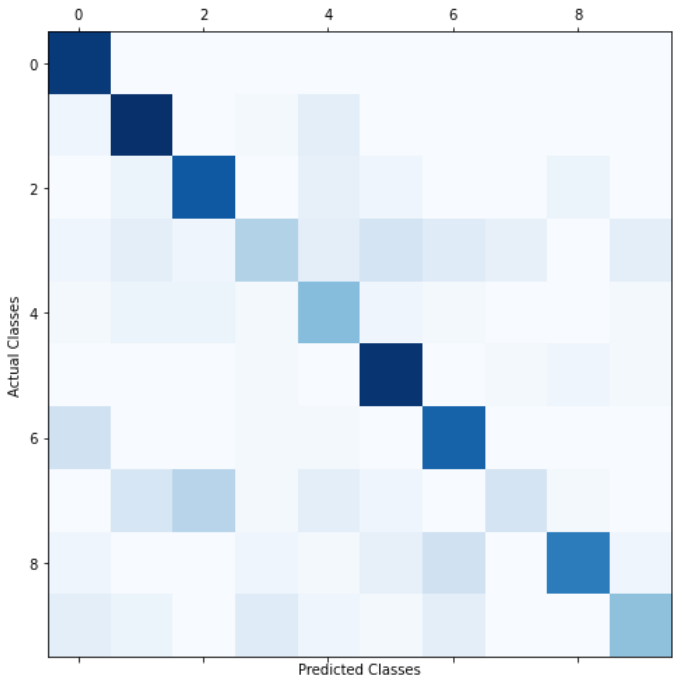
For the seventh experiment we took the same approach as experiment but added regularization. We used dropout and batch normalization as the regularization techniques. A summary of the architecture can be seen in the image below. We used two dropout layers of .3 and a batch normalization layer to add regularization to the model. This model performed best on the test data set out of all the CNN models with an accuracy of 67.59%.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 257, 11, 256)	2560
max_pooling2d (MaxPooling2D)	(None, 129, 6, 256)	0
conv2d_1 (Conv2D)	(None, 127, 4, 512)	1180160
max_pooling2d_1 (MaxPooling2	(None, 64, 2, 512)	0
dropout (Dropout)	(None, 64, 2, 512)	0
conv2d_2 (Conv2D)	(None, 63, 1, 1024)	2098176
batch_normalization (BatchNo	(None, 63, 1, 1024)	4096
max_pooling2d_2 (MaxPooling2	(None, 32, 1, 1024)	0
dropout_1 (Dropout)	(None, 32, 1, 1024)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 512)	16777728
dense_1 (Dense)	(None, 10)	5130

Total params: 20,067,850
Trainable params: 20,065,802
Non-trainable params: 2,048

Figure 6: Model summart of experiment 7

After testing on the test data set we created a confusion matrix from the test dataset. From the confusion matrix as can be seen below, along with the genre key, we can see the genres the classifier was good at classifying and the ones it struggled with. The model was good at classifying blues, classical, and jazz. It was weakest at predicting disco and pop.



{'blues': 0, 'classical': 1, 'country': 2, 'disco': 3, 'hiphop': 4, 'jazz': 5, 'metal': 6, 'pop': 7, 'reggae': 8, 'rock': 9}

Figure 7: Confusion Matirx of Experiment 7 with Key

Experiment 8

After exploring the CNN models we looked to use a DNN model on the CSV data provided by Kaggle. From the CSV we were given a CSV file with 57 columns to help identify the song. Some of the columns were MFCC data, tempo, zero crossing rate, and spectral centroid. For example, using the tempo we were able to build a box plot graph for the different genres as seen below. The comprehensive list of the columns can be seen in the appendix. For this model we used the CSV data produced from 3 second clips of the 30 second songs, resulting in 10000 records. The DNN model performed much better than the CNN data due to the increase and depth of data that wasn't available to the CNN model.

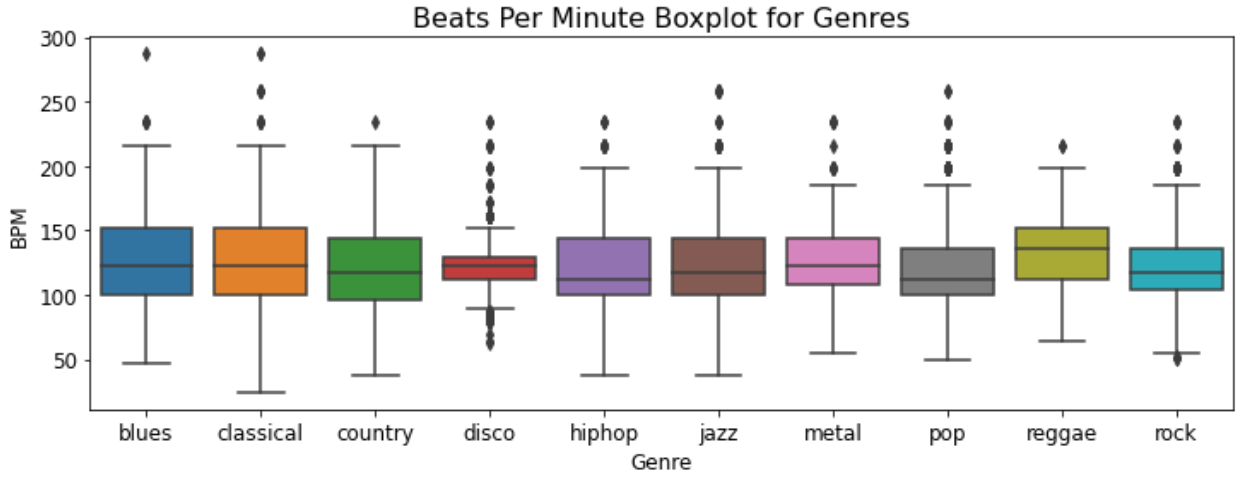


Figure 8: BPM Boxplot for audio data

The DNN architecture build was four layers, with an input layer containing 256 units, then a dense layer 128 nodes and relu, followed by another dense layer of 64 nodes and relu, with finally a 10 node output layer with a softmax activation layer. With this architecture we achieved and accuracy of 90.34% on the validation set and 90.78% on the test set.

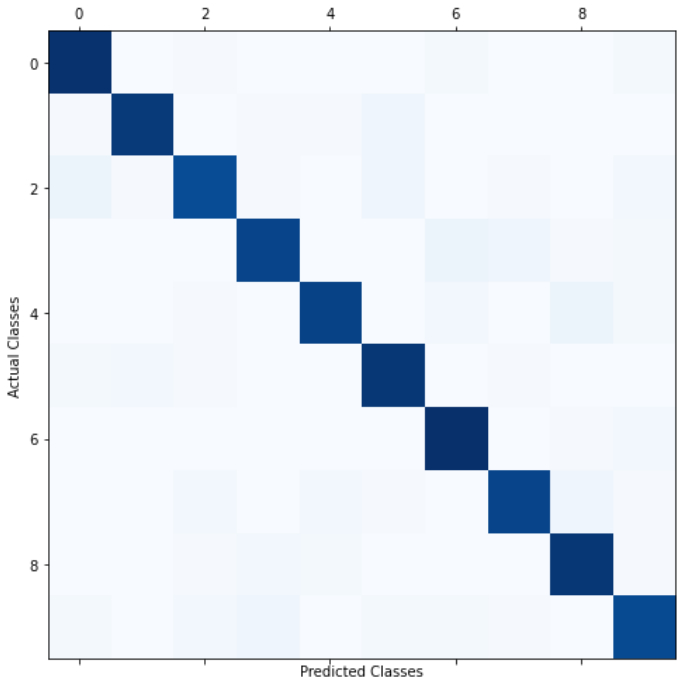


Figure 9: Confusion Matrix for DNN of Experiment 8

Complete Set of Results

Experiment	Training Accuracy %	Validation Accuracy %	Test Accuracy %	Model Type
1	91.98%	67.22	65.21	CNN
2	94.96	57.78	55.02	CNN
3	98.25	51.26	50.02	CNN
4	87.94	62.74	62.26	CNN
5	71.45	55.19	55.33	CNN
6	91.58	69.56	62.17	CNN
7	96.78	66.89	67.59	CNN
8	99.70	90.34	90.78	DNN
9	99.8	70.02	73.53	DNN

Table 1: Complete Experiment Results

Conclusion

Our best CNN model achieved an accuracy of 67.59% on the MFCC images. The best DNN model achieved an accuracy of 90.78% from the data in the created CSV containing various features. From this we see that if we can capture all of the features from songs that are within the dataset that using a DNN is the best approach. If we don't have the ability or time to compute these features, then transforming the audio clip to a Mel-Spectrogram then MFCC is useful to train the model. For both the DNN and the CNN we saw that when the song clips were broken into more data points that the accuracy of the models increased as it had more data to learn from. In conclusion we learned some valuable practices for working with audio data and how to

build a model that can accurately classify music. This will be a good move for the company to continue to work on, as we look to build playlist generators and music recommendation engines.

Appendix

Literature Review Paper:
https://www.sciencedirect.com/science/article/pii/S0957417415006326?casa_token=RyOBpqlHthAAAAAA:Fc4GVd2ablWNbyXKRmdbe95RxnqOvcqygJr4I8sLlQjTHayvylv4i2PNEEkhvqMolBt_xEC86EI

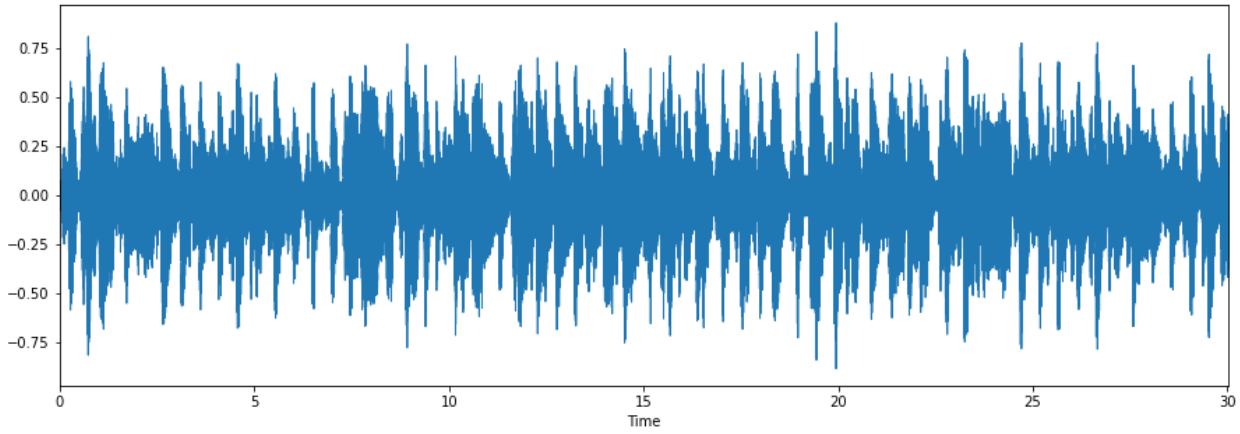


Figure 10: Waveplot for blues song

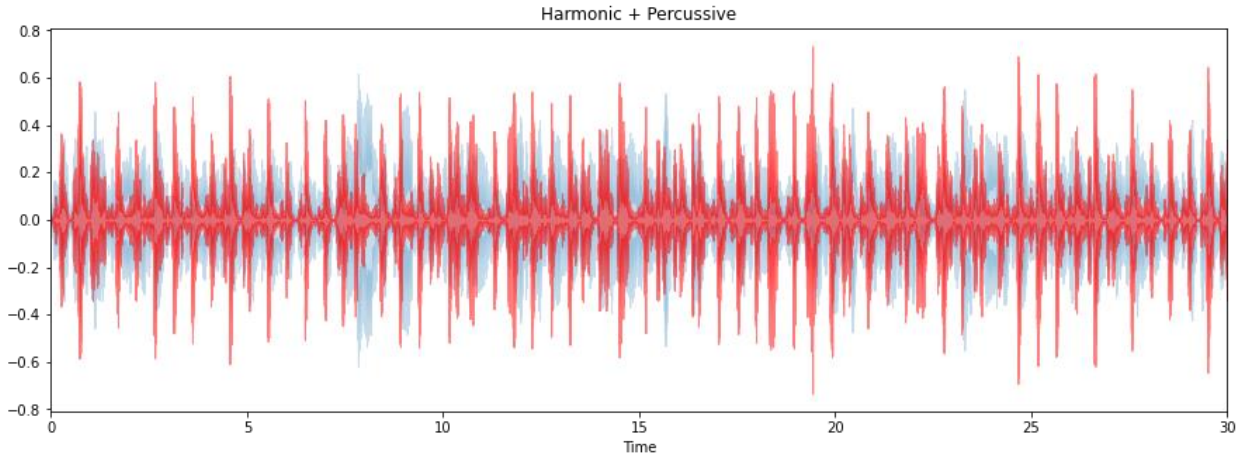


Figure 11: Harmonic and Percussive graph for blues song

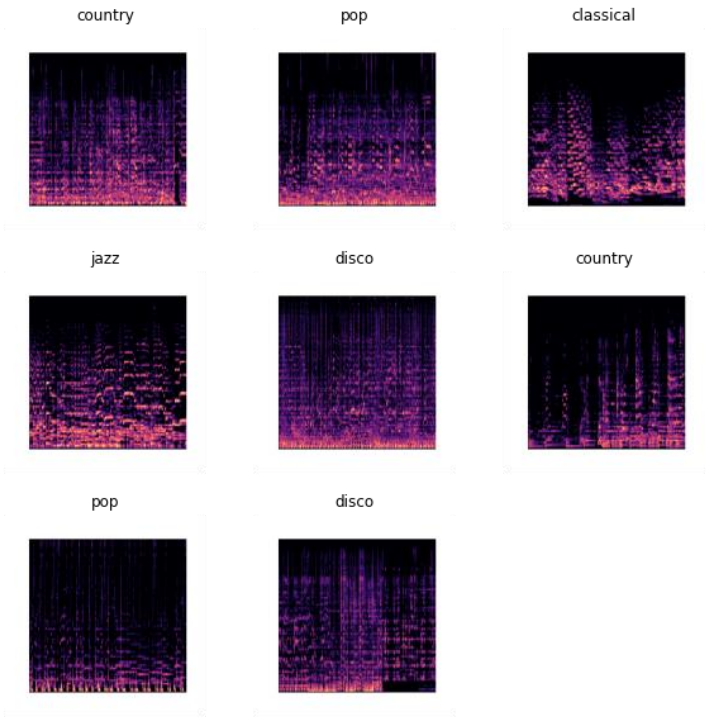


Figure 12: Mel Spectrogram images for 8 images

```
Index(['filename', 'length', 'chroma_stft_mean', 'chroma_stft_var', 'rms_mean',
       'rms_var', 'spectral_centroid_mean', 'spectral_centroid_var',
       'spectral_bandwidth_mean', 'spectral_bandwidth_var', 'rolloff_mean',
       'rolloff_var', 'zero_crossing_rate_mean', 'zero_crossing_rate_var',
       'harmony_mean', 'harmony_var', 'perceptr_mean', 'perceptr_var', 'tempo',
       'mfcc1_mean', 'mfcc1_var', 'mfcc2_mean', 'mfcc2_var', 'mfcc3_mean',
       'mfcc3_var', 'mfcc4_mean', 'mfcc4_var', 'mfcc5_mean', 'mfcc5_var',
       'mfcc6_mean', 'mfcc6_var', 'mfcc7_mean', 'mfcc7_var', 'mfcc8_mean',
       'mfcc8_var', 'mfcc9_mean', 'mfcc9_var', 'mfcc10_mean', 'mfcc10_var',
       'mfcc11_mean', 'mfcc11_var', 'mfcc12_mean', 'mfcc12_var', 'mfcc13_mean',
       'mfcc13_var', 'mfcc14_mean', 'mfcc14_var', 'mfcc15_mean', 'mfcc15_var',
       'mfcc16_mean', 'mfcc16_var', 'mfcc17_mean', 'mfcc17_var', 'mfcc18_mean',
       'mfcc18_var', 'mfcc19_mean', 'mfcc19_var', 'mfcc20_mean', 'mfcc20_var',
       'label'],
      dtype='object')
```

Figure 13: Features from CSV that were capture from audio data