

Music Genre Classification with VGG16

Term Project Report

for

CS665 Deep Learning

Group 6

Filip Krutul

Trenton Tidwel

Zhiying Lu


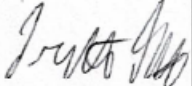
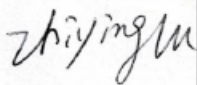
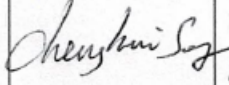
Chenghui Song

April 24, 2021

Declaration

We declare that we have completed this assignment completely and entirely on our own, without any consultation with others. We have read the UAB Academic Honor Code and understand that any breach of the Honor Code may result in severe penalties.

We also declare that the following percentage distribution ***faithfully*** represents individual group members' contributions to the completion of the assignment

Name	Overall Contribution (%)	Major work items completed by me	Signature or initials	Date
Filip Krutul	25	Create/test the model		4/19/21
Trenton Tidwel	25	Create dataset Test the model		4/19/21
Zhiying Lu	25	Fine tune the model Prepare presentation		April 20 2021
Chenghui Song	25	Test/tune the model Draft the final report		4/19/21

Abstract

Music genres are categorical labels created by humans to determine the style of music. When done correctly, the genres provide effective classification that greatly facilitate our comprehension, recognition, and enjoyment of the music we hear. However, the genres may not always be available to users and sometimes may be mislabeled. The current report presents a method to classify music genres using audio spectrograms created from original soundtracks. The spectrograms were used as visual representation for feature extraction by using a deep learning model VGG-16 with the Adam optimizer. Our preliminary results from a total of 5000 samples indicated a promising performance of the model: an accuracy of $> 80\%$ for both training and testing, suggesting that the model is suitable for music genre classification.

1. Introduction

Music genres are important because they provide a fundamental means for understanding and appreciating music as an art form [1]. Publishers use them to organize the data and recommend new songs to users according to their listening habits and the users tend to find new songs according to their categories (genres). However, as the size of music collections continues to grow, especially on online platforms, it becomes impossible to manually keep track of the genres. Due to overlap in subjective taste, some genres are also likely being cross labeled and mislabeled. This has raised a need to automatically organize the songs by genres.

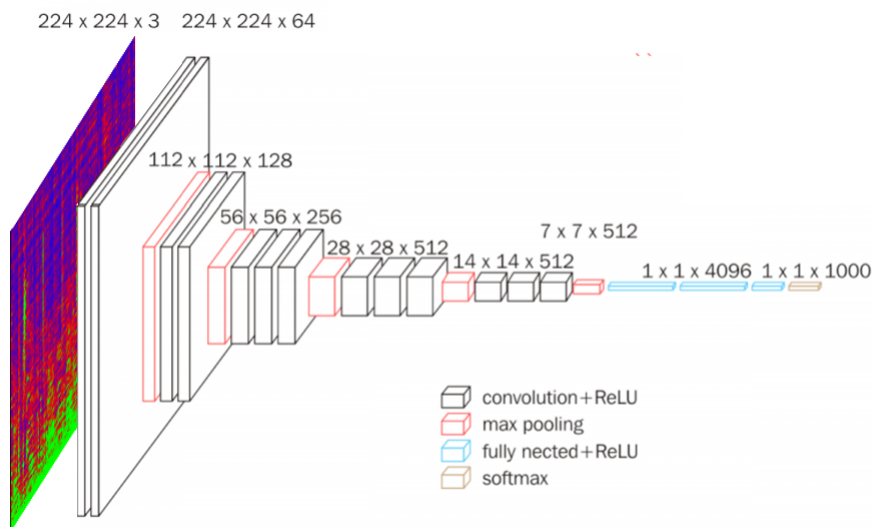


Figure 1: The architecture of VGG16 (adapted from Hassan 2008 [2])

Automatic classification of genres is challenging because of the lack of standardization and vague genre definitions. Human music perception and personal experiences make this agreement even more difficult. Despite the difficulty, automatic genre classification has received increasing attention in the area of music information retrieval [3]. Recently many researchers have used traditional classifiers in machine learning with audio features extracted from online platforms such as Spotify to predict genres, and some have achieved promising results [4-6]. The major drawback is that the audio features are not always conveniently

available for users. Thus directly using sound tracks as input data is preferable. Although there are existing reports that have studied in this direction, the results are not satisfying [e.g., 7].

In this project, we used a binned fast Fourier transform to create our own spectrograms from original audio tracks in raw ‘wav’ format, and used a deep learning model adapted from VGG-16 (Figure 1) to classify song genres. VGG16 is a convolution neural net (CNN) that has become one of the famous models for large-scale image recognition and has achieved more than 90% top-5 test accuracy in ImageNet [8].

2. Methods

2.1 Create spectrograms by genre

We first implemented an OS script to scrape audio data from Youtube playlists in uncompressed ‘wav’ format, saving the songs in a nested directory by genre. The audio files were then Normalized and passed into a custom Binned-Fast-Fourier-Transform (BFFT) script to create spectrograms of each audio file. The Fourier Transform is an image processing tool which decomposes data from the time (or spatial) domain into the frequency domain [7]. The Fast Fourier Transform (FFT), formerly known as the Cooley-Tukey Algorithm, is the preferred discretization of this transform as we are concerned with only digital music which is already stored as discretized time series information. By binning this application to many small time steps, we can create spectrograms, which show how the audio sample’s frequencies evolve over time (Figure 2).

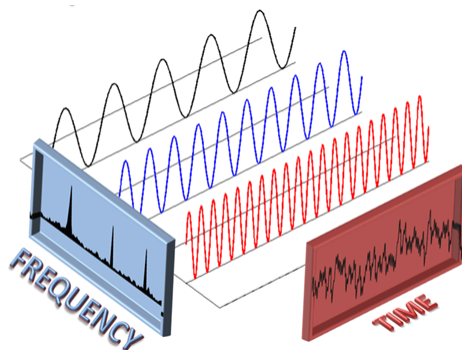


Figure 2: Fourier transformation of audio data (time) into visual data (frequency)[9]

Before applying the BFFT, we chose to cut the songs into 30 second clips to preserve time-series elements, as forcing a 2-minute audio file and a 6-minute audio file into the same sized image could have adverse effects on training accuracy.

Due to the image input size constraints of VGG, the frequency range of the Fourier Transforms was also cut in half as the important information exists only in the lower range of the spectrum; though high frequency information is nice to hear as a listener, it can become harsh if it’s too loud and is generally not visually present in the full range spectrogram. The minimum input size for VGG is 224x224, therefore our BFFT implementation utilizes 224 time bins across the 30 second clips resulting in a bin time of approximately 0.12 seconds. This is an appropriate time length to capture quick artifacts such as snares, hi-hats, and other staccato elements, as well as accurate pitch following of longer phrases. Due to intros and outros of songs generally containing broken down and softer audio information, the first and last 30 seconds of each song were discarded.

VGG accepts color images before converting them to grayscale; the spectrogram color scheme of BRG was chosen due to the Luminance Equation's preference for green over red over blue channels.

$$L = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B$$

The 224x224 BRG .png spectrograms corresponding to 30 second song clips are the final data structure that was passed to the Neural Network (see Figure 3 for spectrogram preparation pipeline and Figure 4 for representative spectrograms of each genre). As the spectrograms were created, the original audio files were deleted out of respect for copyright claims on the audio files.

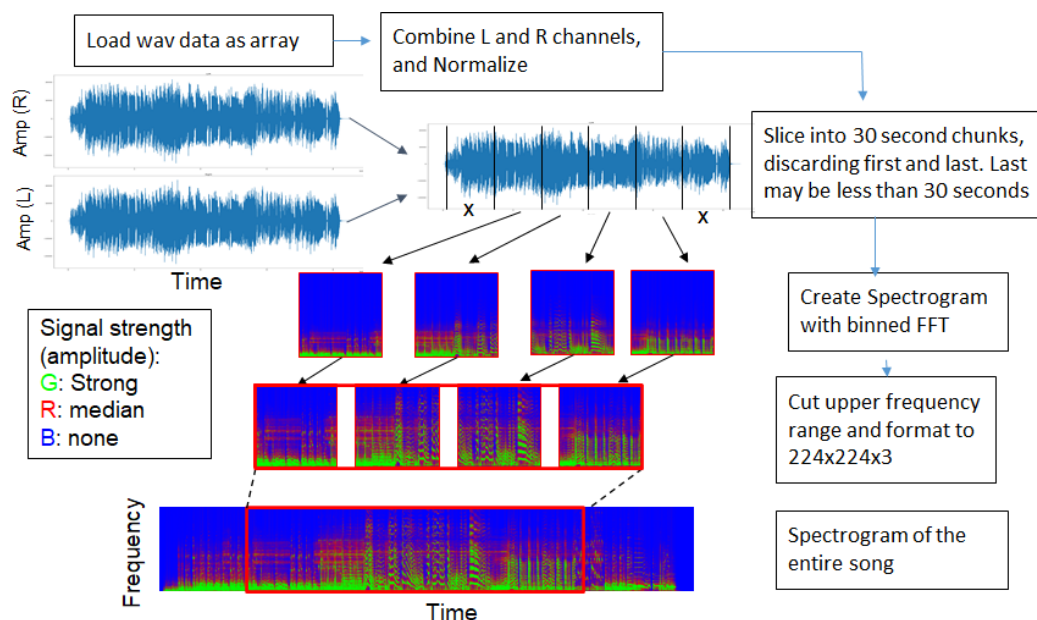


Figure 3: Spectrogram preparation pipeline

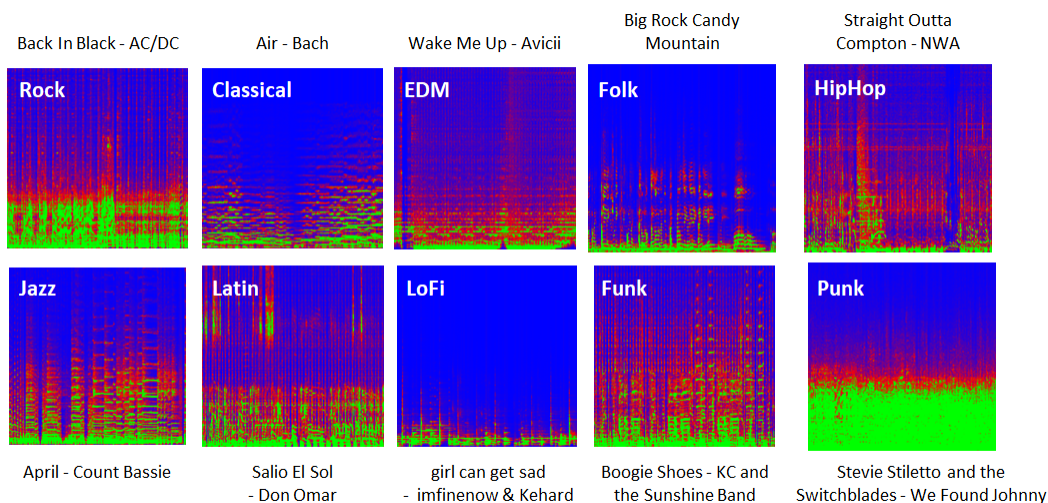


Figure 4: Representative spectrograms

The complete dataset was composed of 5000 spectrogram slices placed in directories corresponding to each genre. Each genre folder contains exactly 500 images. Before training the model, the master folder of spectrogram data was split into training, testing, and validation datasets with a specified ratio (0.7, 0.2, and 0.1 respectively). In order to further increase the size of the dataset and add noise into the training process to reduce overfitting, we augmented the data by using vertical and horizontal shifts, as well as zooming and rotations within the image data generator for the model.

2.2 Choice of model and customization: VGG16

The VGG16 model was chosen due to its high level of performance on ImageNet. While other models have achieved a slightly higher accuracy on ImageNet, the purpose of this project was to determine whether VGG16 can be properly adapted to learn and predict spectrogram image labels. To instantiate the basis for the classifier, VGG16 was loaded with pretrained ImageNet weights as a base for fine tuning. During instantiation, the fully connected layers at the top of the model were excluded in order to turn the bottom portion of the model into a feature extractor. Instead of appending custom fully connected layers at the top of the model, a global max pooling layer coupled with dropout was used instead. There are several advantages to this alternative architecture. The primary reason for its use was a reduction in the overfitting potential of fully connected layers. Custom fully connected layers can be prone to overfitting because they come with their own set of trainable parameters, which increase the overall complexity of the network. The global max pooling layer, on the other hand, has no trainable parameters, and does not increase network complexity. In this approach, the convolutional and pooling layers in the base model create feature maps representing components of the input spectrogram. The global max pooling layer takes the maximum values from the feature maps of the last convolutional layer and outputs a feature map vector. Coupled with global max pooling, dropout is utilized for further regularization by setting input units to 0 with a frequency of 0.3. A GAP (global average pooling) architecture with no fully connected layers was also tested with and without dropout, but the best results were achieved with a combination of global max pooling and dropout. The trainability of all the layers was maintained as well, in order to fine tune all of the weights in the model. Experimenting with freezing some of the layers in the base of the model resulted in much lower accuracy, although computational time also decreased as a result. See Figure 5 for the model summary, Table 1 and Figure 6 for detailed summary and model plot.

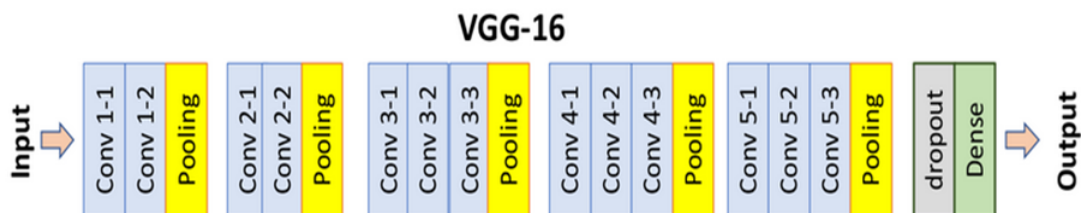


Figure 5: Simplified VGG16 model summary

Table 1: model summary

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_max_pooling2d (Global	(None, 512)	0
dropout (Dropout)	(None, 512)	0
dense (Dense)	(None, 10)	5130
Total params: 14,719,818		
Trainable params: 14,719,818		
Non-trainable params: 0		

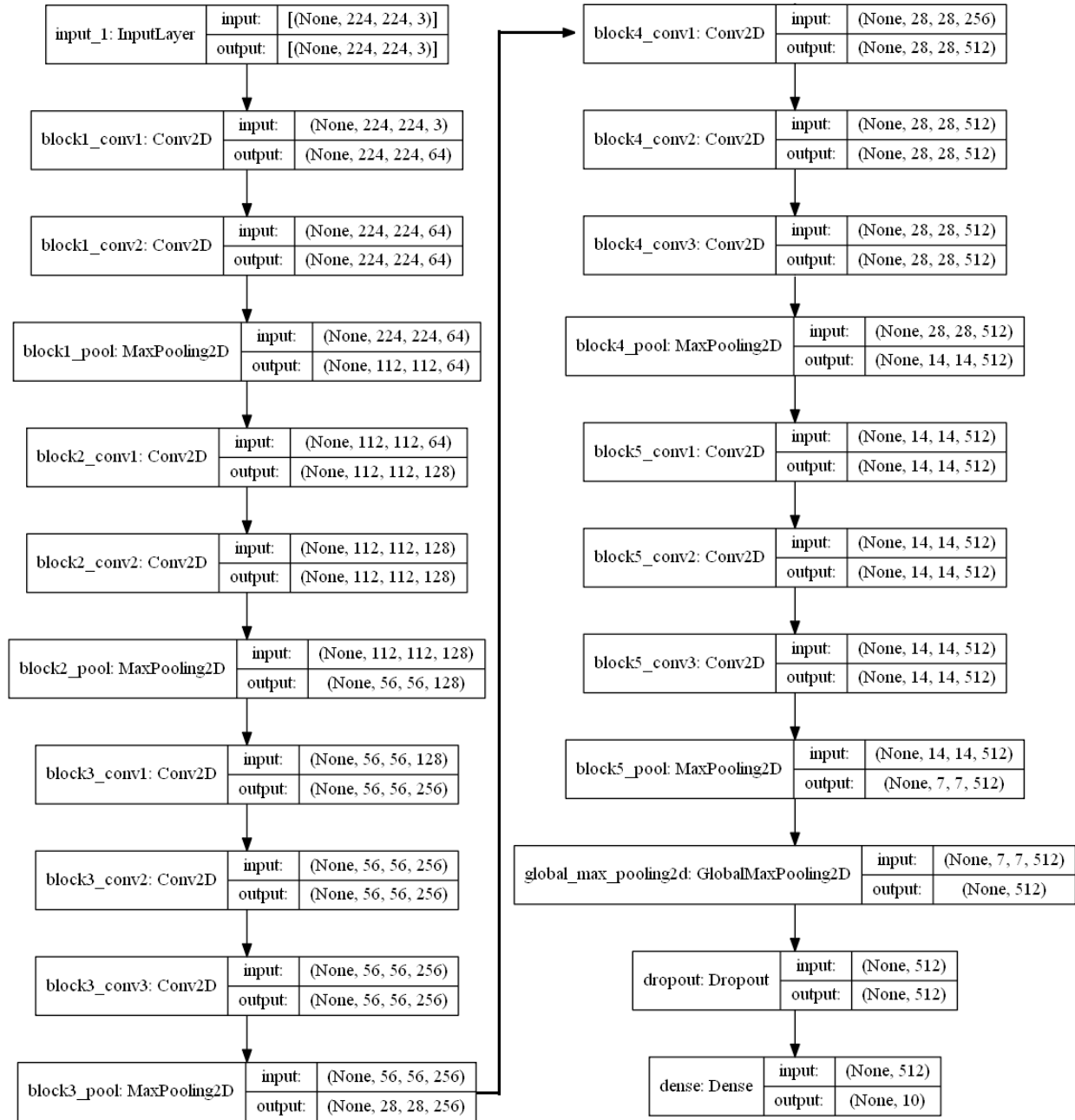


Figure 6: VGG16 model plot

Thus, our model is a sequence of convolution layers, interspersed with activation functions. The initial 224x224 input images were convolved repeatedly with different filters and the class scores (C) with volume size of 1x1x10 were computed. Each of the 10 numbers corresponds to a class score among the 10 genres. In this way, the model transforms the original pixel values to the final class scores, which is a possibility score calculated by the Softmax function at the end of the model:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

where C is one of the 10 classes (genres), and s are the vector scores calculated from the classes. Finally, a report of a probability over the C classes for each image can be obtained by using Cross-Entropy loss for multi-class classification:

$$CE = - \sum_i^C t_i \log(f(s)_i)$$

where t_i and s_i are the groundtruth and CNN score for each class i in C . See Figure 7 for full classification pipeline.

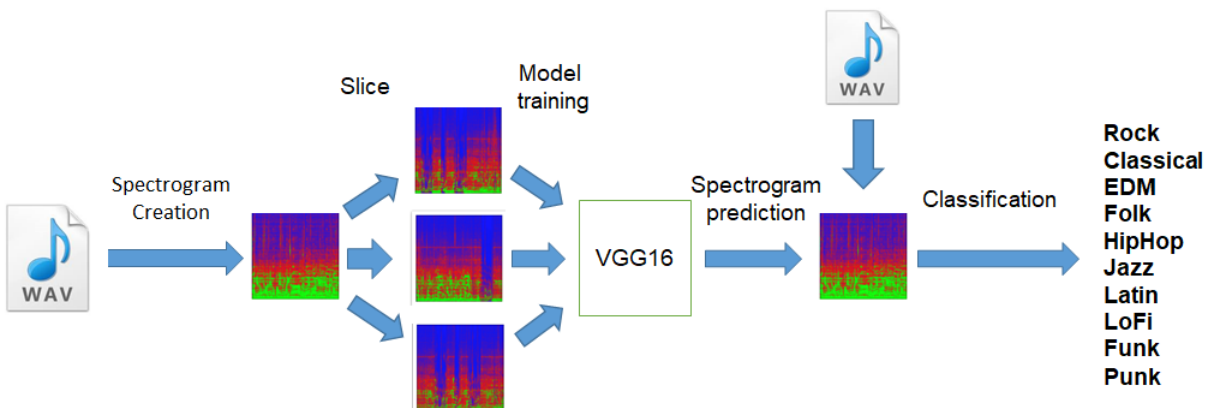


Figure 7: Full classification pipeline

3. Results

Our initial training accuracy was about 75% and testing accuracy was about 65% when the learning rate was 0.0005 and with 10 epochs. The gap between training and testing scores suggests that the model was overfitted. We then fine tuned the model by decreasing learning rate to 0.0001, increasing epochs, and adding a dropout layer. The final performance was improved to $> 80\%$ for both training and testing with much smaller overfitting (see Figure 8). With more epochs, the final accuracy was 83% (see Figure 9).

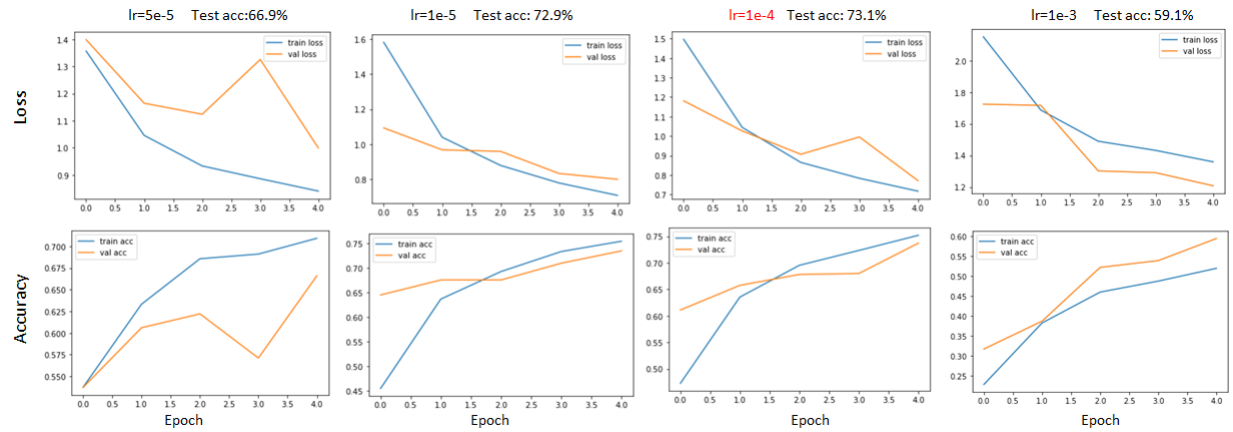


Figure 8: Tuning the model. Best accuracy achieved when learning rate was 1e-4

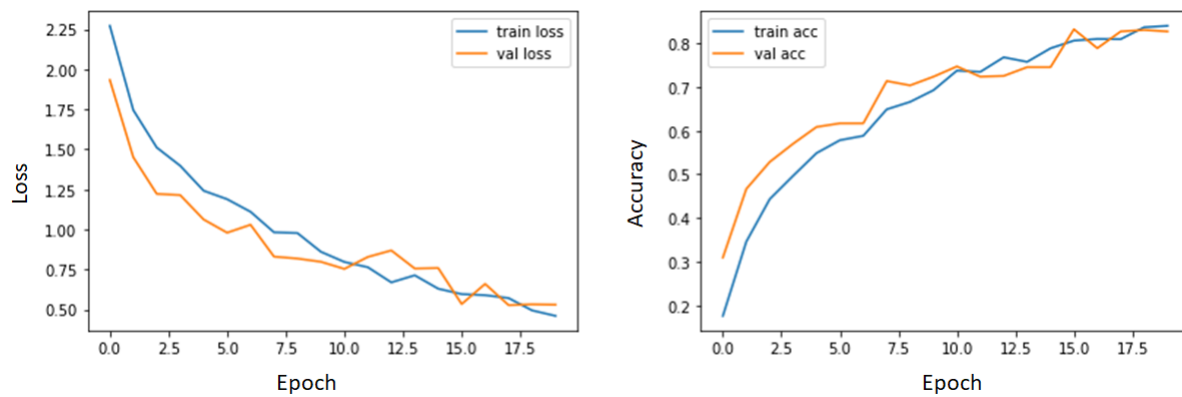


Figure 9: Loss and accuracy history in final run

Figure 10 shows the performance of our model on the test data. It suggests that half of the genres were about or more than 90% correctly classified. The overall accuracy was about 85.8%, and the overall misclassification rate was 14.2%. Using this confusion matrix, we made a chord graph (Figure 10 right) which nicely shows the proportion of misclassification. For example, part of the EDM and Latin were misclassified as each other. This is because that Latin uses some of the EDM elements so there's some overlap between them.

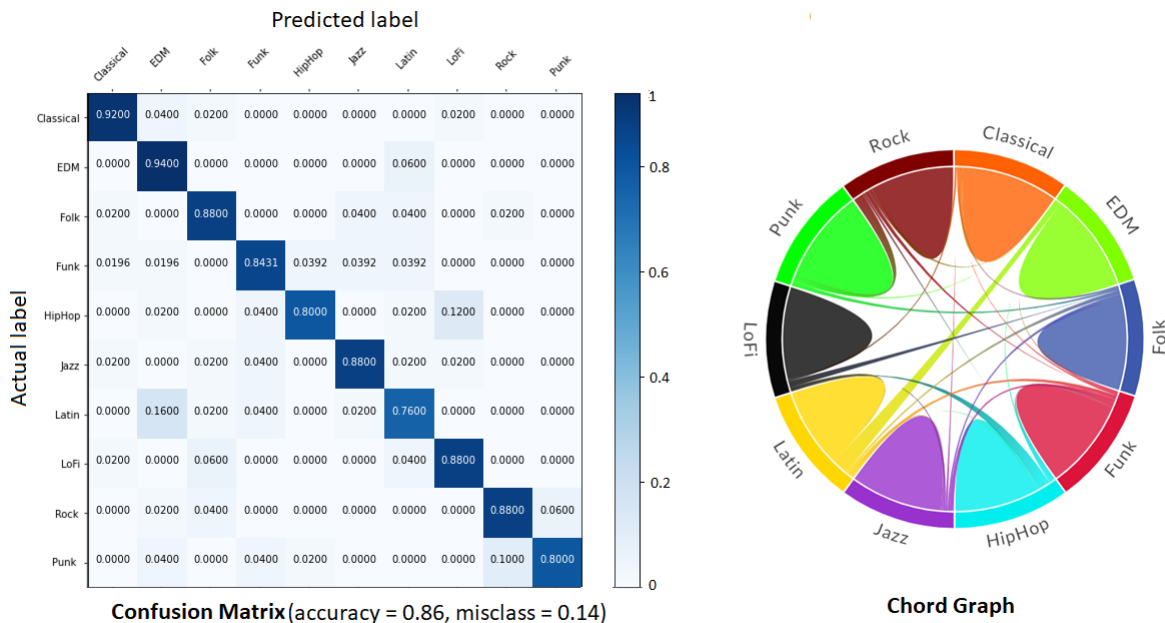


Figure 10: Confusion Matrix and Chord Graph

Following the training of a model on slices of songs instead of full audio clips, it proves to be interesting how the model can perform on an entire song. We implemented this using the same overall 30 second Binned FFT structure, this time filling one folder with the spectrograms of a single song. The individual images, passed into the `model.predict()` function, serve as a basis for the most frequently predicted genre to be applied to the overall track. An example of this functionality is shown in Figure 11, where the overall genre of the *Foo Fighters* track: *Best of You* is correctly predicted as ‘Rock’ with one of the six slices misclassifying as ‘Punk’. This is a good case study as it showcases a relative difference in the punk and rock classifications; the misclassified slice has a relatively lower presence of high frequency information and a greater presence of mid-low frequency information. Listening to the song, this section corresponds to the 2:00 - 2:30 section, which is the high emotion instrumental breakdown where the guitar takes up more space and the drums transition from heavy cymbal usage to heavy tom usage, which is more in line with traditional ‘Punk’ stylism. We feel this is less of a misclassification and more of a showcase of cross-genre elements; while Foo Fighters are not ‘Punk’, they do absorb elements from this genre.

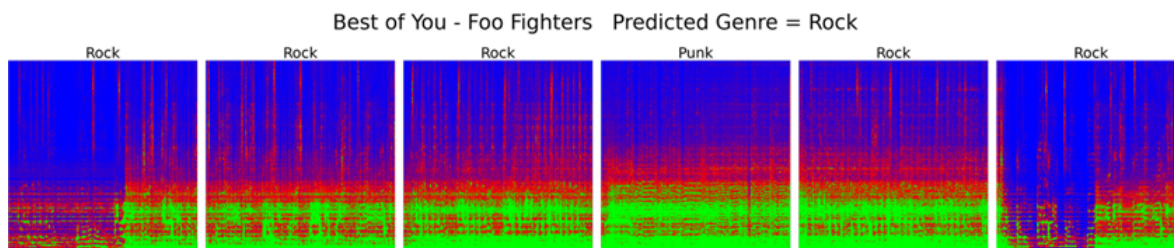


Figure 11: Full Song Classification Demonstration

4. Discussion and future direction

In the current project, we used VGG-16 as a deep learning model and audio spectrograms as input data to classify song's genres. The final accuracy was $> 80\%$, which is comparable with other existing reports that used other models with spectrograms, which ranged between $60\% - 90\%$ [6, 10-14]. For a fair comparison, we'll need to test our model on the same datasets that other researchers used.

We believe it would also be beneficial to run a version of our model in parallel with a Recurrent Neural Network which could help identify event features of the songs and improve accuracy. We felt the Convolutional approach would be substantially more accurate than a stand alone RNN, but this does not discount the potential use of an RNN in the framework of a best performing genre recognition algorithm.

The current project only tested 10 major genres. Whereas in reality, there are substantially more genres and subgenres, which makes the classification task difficult. Thus, in order to make our model useful, we will need to test it with more genres and larger data sets. Our long-term goal is to use the model for real-time genre recognition, so that the program directly recognizes the genre when a song is being played.

References

1. Preston Cram. "Are Music Genres Important? 3 Ways They Improve Our Experience With Music". www.prestoncram.com/post/are-music-genres-important-3-ways-they-improve-our-listening-experience.
2. Hasson, M.L. (2018) "VGG16 – Convolutional Network for Classification and Detection". <https://neurohive.io/en/popular-networks/vgg16/>
3. Fu, Z., Lu, G., Ting, K.M., Zhang, D. (2011). A survey of audio-based music classification and annotation. *IEEE Transactions on Multimedia*, 13(2), 303–319.
4. Bajwa et al. "Song Genre Prediction Using NLP and Audio Features". www.github.com/etarakci/music-genre-prediction#readme
5. Yash Dua (2019) "Music Genre Prediction with Spotify's Audio Features". <https://towardsdatascience.com/music-genre-prediction-with-spotifys-audio-features-8a2c81f1a22e>
6. Sianipar (2019) "Predicting a Song's Genre Using Natural Language Processing". <https://betterprogramming.pub/predicting-a-songs-genre-using-natural-language-processing-7b354ed5bd80>
7. Costa, Y. M. G ; Oliveira, L. S ; Koerich, A. L ; Gouyon, F. (2011). Music genre recognition using spectrograms. 18th International Conference on Systems, Signals and Image Processing, 2011-06, p.1-4 IEEE
8. Smolyan, K., and Zisserman A. (2015) "Very Deep Convolutional Networks for Large-Scale Image Recognition". arXiv:1409.1556 [cs.CV]
9. James, M. (2012) "A Faster Fourier Transform". <https://www.i-programmer.info/news/181-algorithms/3644-a-faster-fourier-transform.html>
10. Wim van Drongelen. (2018) "Continuous, Discrete, and Fast Fourier Transform" in *Signal Processing for Neuroscientists (Second Edition)*, Academic Press, 103-118.
11. Defferrard, Michael et al. "Learning to Recognize Musical Genre from Audio". arXiv:1803.05337

[cs.SD]

12. Bhat, Umesh. (2016) "Finding the genre of a song with Deep Learning". <https://hackernoon.com/finding-the-genre-of-a-song-with-deep-learning-da8f59a61194>
13. N. M R and S. Mohan B S, (2020) "Music Genre Classification using Spectrograms," 2020 International Conference on Power, Instrumentation, Control and Computing (PICC), Thrissur, India, 2020, pp. 1-5, doi: 10.1109/PICC51425.2020.9362364.
14. Yandre Costa, L.S. Oliveira, and A.L. Koerich. (2011) "Music Genre Recognition Using Spectrograms". IEEE Xplore