50.039 Deep Learning Project

Group 6

Music Recommender System

Team Members:

1004359	Mihir Chhiber
1004385	Tan Ze Xin Dylan
1004429	Kanashima Hatsumi
1004457	Li Meixuan
1004680	Gargi Pandkar

Github Repository Link:
https://github.com/mihirchhiber/deep_learning_project.git

Table of Contents

1. Introduction

In this project, we are motivated by shazam's music recommendation system. The goal of this project is to output the genre and recommend a list of songs using the input music provided by the users. The team attempted various ways to improve the model, such as experimenting with different model designs and fine tuning hyperparameters such as the optimizers and learning rate.

There are two sorts of recommender systems: content-based and collaborative-based. A content-based system predicts what a user will like based on past preferences, while a collaborative-based system predicts what a particular user likes based on the preferences of other similar users. The hybrid technique is used by most enterprises, such as Netflix, to make recommendations based on a combination of what content a user has previously liked as well as what other similar users have liked.

In this project, we will be building a content-based music recommendation system. We will be creating song embeddings by training the model on genre classification. Using the song embeddings, we will find the similarity between songs and recommend similar songs in the dataset.

2. Dataset

2.1 Dataset Description

The GTZAN dataset is made up of 1000 audio tracks, each of which lasts 30 seconds. It is divided into ten genres, each of which has 100 tracks, as shown in Figure 1. The audio files are all 22050 Hz Mono 16-bit.wav files. From Figure 1, it can be seen that the distribution of each class is balanced.

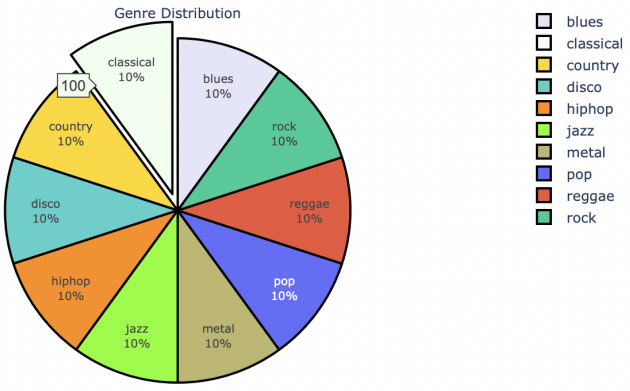
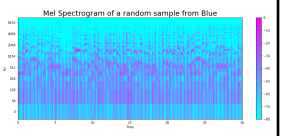
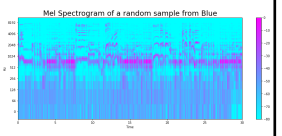
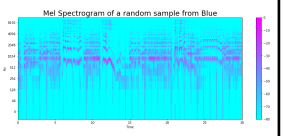
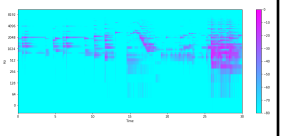
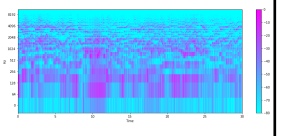
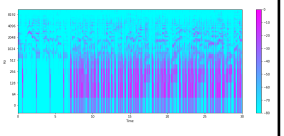
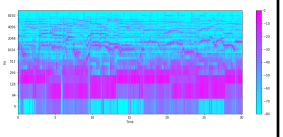
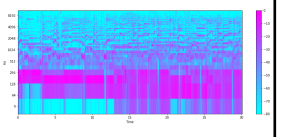
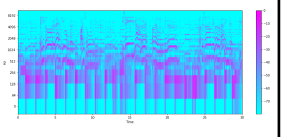
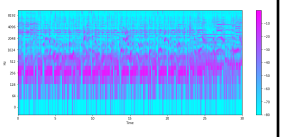
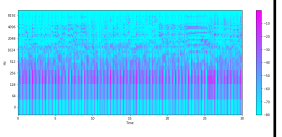
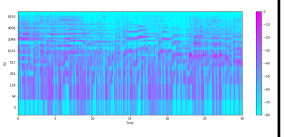
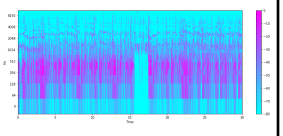
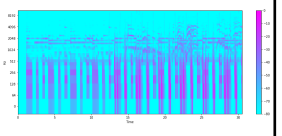
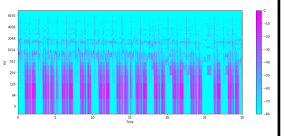
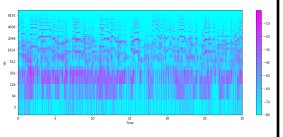
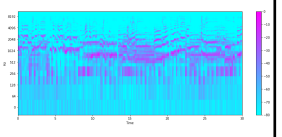
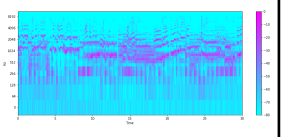
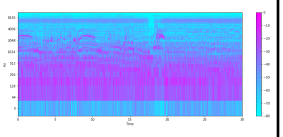
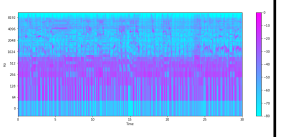
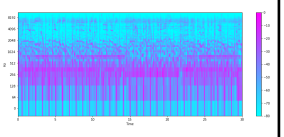
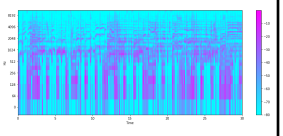
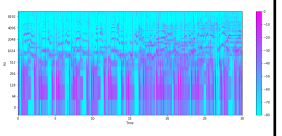
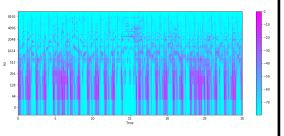
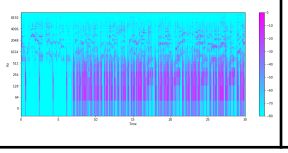
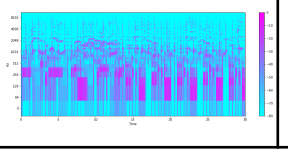
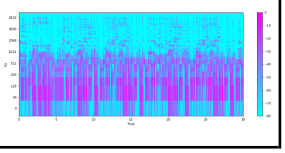


Figure 1: A Pie Chart of Genre Distribution

2.2 Dataset Visualization

A spectrogram is a visual representation of the frequency of sound as it varies over time. People, however, do not perceive frequencies on a linear scale, as depicted by spectrograms. Therefore Mel-spectrograms which translate frequencies to the Mel scale and are known to perform better are employed instead.

A Mel-spectrogram is plotted based on the frequencies of sound against time, and the color-scale on the side reflects the amplitude of the sound in decibels (dB). The Mel-spectrograms for three randomly picked samples from each genre are plotted as reference as shown in Table 1. The Mel-spectrograms show that each genre has a distinct pattern to a certain extent. Metal songs, for example, are typically formed of frequencies ranging from 50Hz to 1000Hz at low amplitudes, with higher aptitudes for the remaining frequency ranges.

Genre/Audio Graph Type	Mel-spectrogram	Mel-spectrogram	Mel-spectrogram
Blues			
Classical			
Country			
Disco			
Hiphop			
Jazz			
Metal			
Pop			
Reggae			

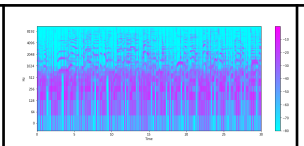
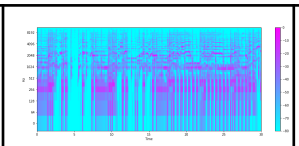
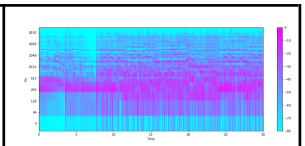
Rock			

Table 1: Mel Spectrogram of 3 samples from each genre

As seen in Figure 2, box plots provide a streamlined approach of summarizing the distribution of groupings of data. For example, the average beat per minute of all audio files in the blues genre is roughly 125bpm. The musical pattern is especially similar for certain genres that are similar, such as blues and classical. Manually accessing and comparing similar music files can also yield this information. It has also been determined that the average beat per minute varies by genre. The beat per minute measurements clearly show that the sound effects for pop and blues music genres differ.

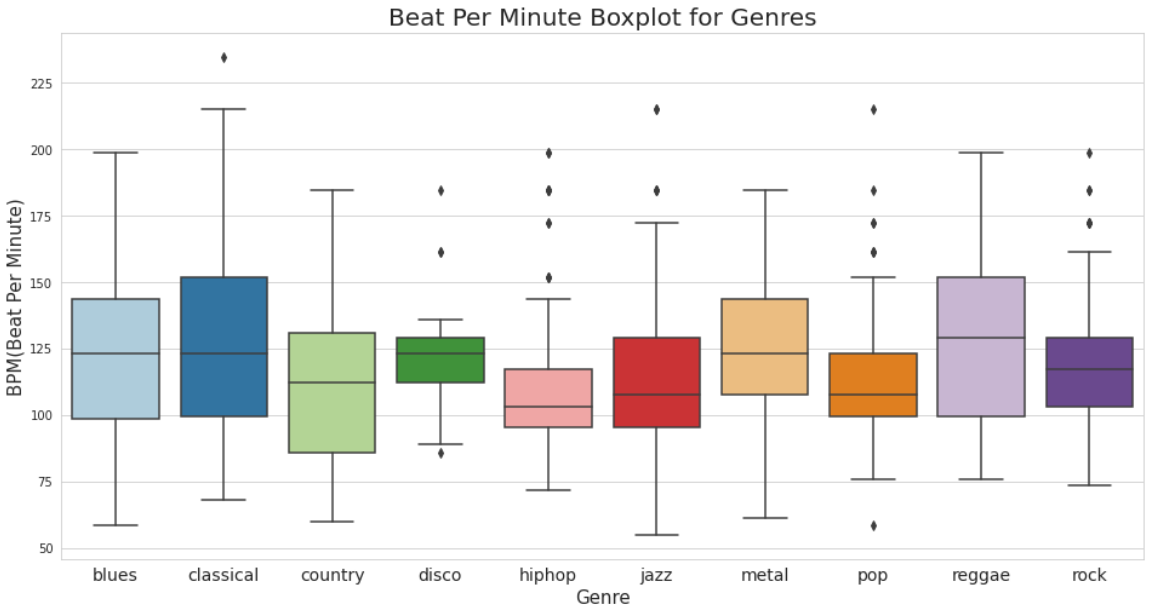


Figure 2: A Box Plot for Genre Distribution

In addition, as shown in Figure 3, we used K-Means Clustering and PCA to categorize music based on similar acoustic attributes. Each genre sample is represented by a separate hue; however, the same genre sample may coexist with others. However, we discovered another intriguing fact: some audio samples from one genre overlap with audio samples from another. One of the examples we can draw from the graph below is the metal and rock genres, which both exist in the same region. It makes sense because their music patterns sound similar. Overall, different genres are scattered throughout the graph, with some genres sharing common qualities.

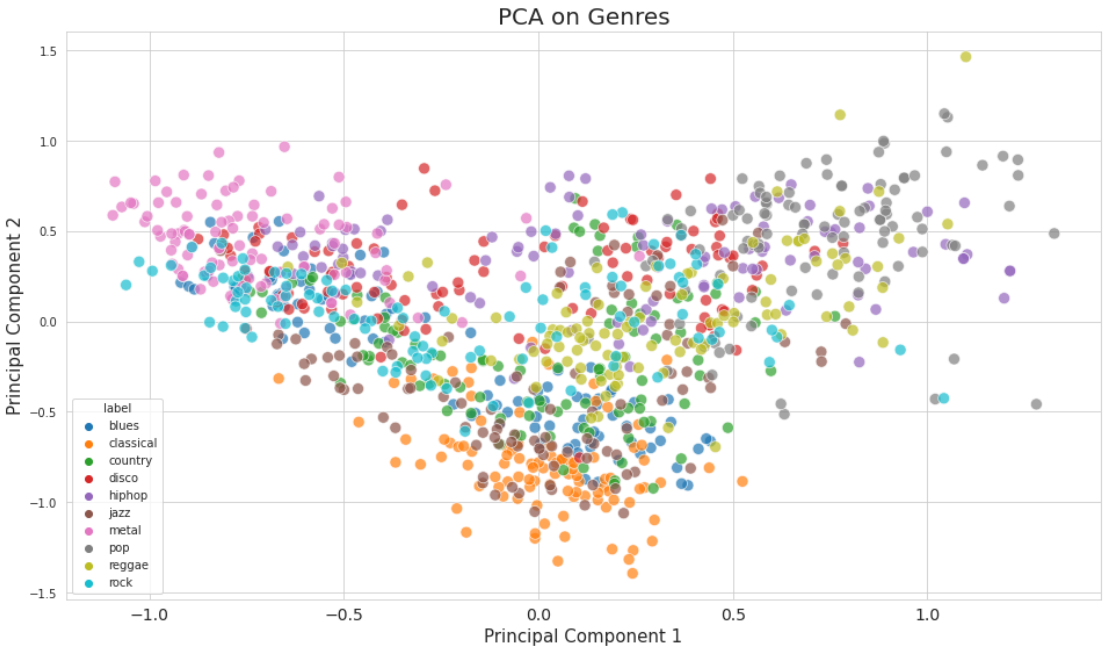


Figure 3: Principal Component Analysis - to visualize possible groups of genres

2.3 Data Preprocessing

Our method will start with songs as audio files as the inputs, convert them to Mel-spectrograms, and feed them into the model. The model's song embeddings will then be retrieved and utilized to discover comparable songs using song similarity scores.

We first constructed a script to convert the songs from audio files to Mel-spectrograms images before training our model. The white borders on the images were removed during the conversion, and the images were then transformed to grayscale images, which are known to perform better. Finally, the image is transformed into Tensors and transposed into a format that the model can use.

3. Experiments

The GTZAN dataset downloaded from Kaggle had one missing file (jazz00054) and hence the total dataset size was 999 audio clips. We use a split ratio of 80:10:10 on the GTZAN dataset to obtain the training, validation and test datasets. The input image shape of the Mel-spectrogram after preprocessing is (219, 338, 1).

The team trained each model with the identical set of hyperparameters (SGD, 25 epochs, early stopping with patience = 5, learning rate = 0.0001) and chose the model with the highest test accuracy.

3.1 Recurrent Neural Network

Songs can be viewed as a series of audio data, which motivates the use of a sequential modeling approach based on recurrent neural networks (RNN). The use of RNNs has several advantages due to their architecture of having shared functions and parameters over

different time steps, such as the reduction in the number of parameters, allowing for better generalization, and a shorter training time.

To apply RNN to songs, we can treat the Mel-spectrograms as sequential data, with each row/column representing a time step. We treated each column as a time step because the vertical patterns on the Mel-spectrograms appear to be more visible than the horizontal margins. As a result, for a 219 x 338 pixels image, there will be 339 time steps consisting of a sequence of length 219.

In the first experiment, we tested Vanilla RNN, LSTM, and GRU over a learning rate of 0.01 with SGD optimizer and the same hidden dimension size and number of layers of 64 and 1, respectively, to find which variety of RNNs are the best suitable. Despite the fact that all models' test accuracies were poor, GRU outperformed Vanilla RNN and LSTM.

Model	Hidden layer dim	Num of layers	Num of parameters	Validation Accuracy	Test Accuracy
Vanilla RNN	64	1	18890	0.100	0.100
LSTM	64	1	73610	0.130	0.100
GRU	64	1	55370	0.290	0.280

Table 2: Table comparing RNN variants

As GRU was significantly better in performance, GRU will be used for the next few experiments. In the second experiment, we sought to see how the depth of RNNs would affect the performance and varied the number of layers. From the second experiment, it was observed that the increase in the number of layers did not improve the performance. Therefore, a GRU with 1 layer will be used for the next few experiments given that it has the best performance.

Model	Hidden layer dim	Num of layers	Num of parameters	Validation Accuracy	Test Accuracy
GRU	64	1	55370	0.290	0.280
GRU	64	2	80330	0.240	0.240
GRU	64	3	105290	0.270	0.230

Table 3: Table comparing number of layers for GRU

In the third experiment, we then sought to see how the size of the hidden layer would affect the performance. From the experiment, it is observed that the increase in the hidden layer’s dimension has improved the performance. Therefore, hidden layers of size 256 will be utilized.

Model	Hidden layer dim	Num of layers	Num of parameters	Validation Accuracy	Test Accuracy
GRU	64	1	55370	0.290	0.280
GRU	128	1	135306	0.320	0.290
GRU	256	1	368906	0.350	0.310

Table 4: Table comparing the size of hidden layer dimension for GRU

Having experimented with variants of RNN, the number of layers, and the size of the hidden layer, we then started with the hyperparameter tuning of the learning rate and type of optimizer using a 1 layer GRU with hidden layer dimension of 256. Based on the experiment, the Adam optimizer with a learning rate of 0.001 seems to result in the best performance.

Learning rate	**0.0001**	**0.0005**	**0.001**
Adam	12.2	12.9	**14.0**
SGD	11.5	10.7	12.5

Table 5: Hyperparameter tuning of learning rate and optimizer for GRU

Finally, we trained a single-layered GRU of 256 hidden layer dimensions with Adam optimizer and learning rate of 0.001, consisting of 368906 parameters. The results are shown in Figures 4 and 5.

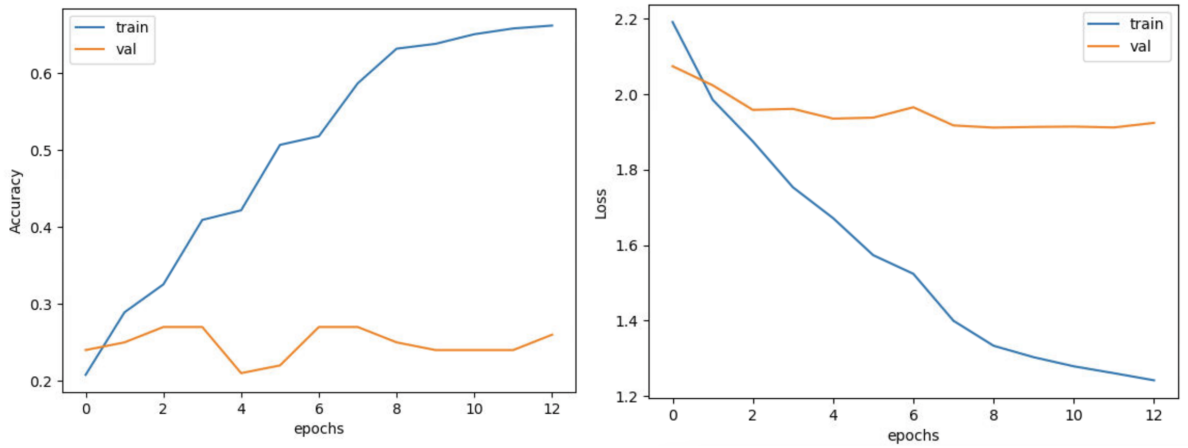
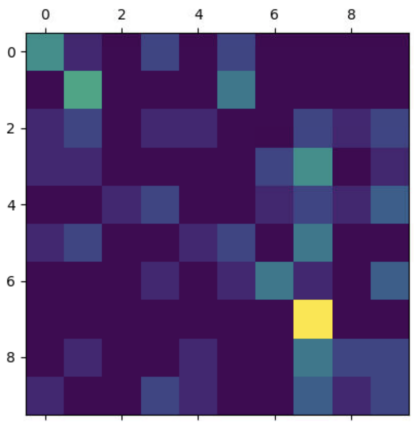


Figure 4. Accuracy and Loss plots for GRU



Classification Report

	precision	recall	f1-score	support
0	0.56	0.50	0.53	10
1	0.46	0.60	0.52	10
2	0.00	0.00	0.00	10
3	0.00	0.00	0.00	10
4	0.00	0.00	0.00	10
5	0.22	0.20	0.21	10
6	0.57	0.40	0.47	10
7	0.32	1.00	0.49	10
8	0.40	0.20	0.27	10
9	0.15	0.20	0.17	10
accuracy			0.31	100
macro avg	0.27	0.31	0.27	100
weighted avg	0.27	0.31	0.27	100

Figure 5. Confusion matrix and classification report for GRU

3.2 Convolutional Neural Network

As RNN had a poor performance for the genre classification task, we decided to employ 2D CNNs with Mel-spectrograms as the inputs since CNNs have been shown to perform well on pictures for feature extraction.

3.2.1 Residual Network

Residual Network, also known as ResNet, is a classic neural network used as a backbone for many types of classification tasks. It is an artificial neural network that aids in the construction of deeper neural networks by employing skip connections or shortcuts to bypass some layers.

ResNet is composed of two main components. The first kind is an identity block, which is used when the input and output activation dimensions are the same, and the second type is a convolution block, which is used when the input and output activation dimensions differ (Gaurav S., 2020)

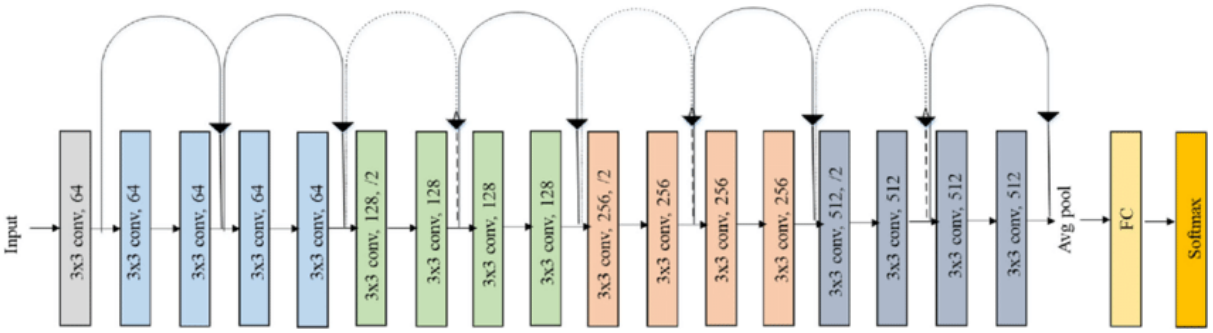


Figure 6: ResNet18 Architecture

The ResNet18 architecture consists of 5 convolutional blocks (Conv2D - Batch Normalization - ReLU - MaxPool2D - 4x Residual Block - AdaptiveAvgPool2D) and one fully connected layer. Each convolutional layer uses 3x3 convolution kernels with stride of 1 and padding of 1.

To understand the significance of the number of parameters in the ResNet architecture and how well the ResNet will perform for genre classification, we first evaluated the original ResNet18 model (4 layers) and then gradually lowered the number of levels to observe how they performed. Please check for appendix 2 for more details.

Model	Number of parameters	Validation Accuracy	Test Accuracy
ResNet18 (Original, 4 layers)	12,852,784	0.72	0.74
ResNet18 (3 layers)	12,688,994	0.79	0.73
ResNet18 (2 layers)	12,607,034	0.81	0.70
ResNet18 (1 layer)	12,566,064	0.80	0.70

Table 6. Test and validation accuracy for the ResNet architechtures

According to the table above, the original ResNet18 with four layers has the highest test accuracy but also the lowest validation accuracy. There was a trend in which the validation accuracy improved but the test accuracy decreased as the number of layers decreased. This could be because higher level features are not being extracted with fewer layers due to which the performance reduces.

Using the most accurate architecture (original ResNet18), the team did a grid search with cross validation to select between Adam and SGD as the optimizer and suitable learning rate.

Learning rate	**0.0001**	**0.0005**	**0.001**
Adam	**36.60**	17.25	15.47
SGD	24.69	29.03	21.79

Table 7. Average test accuracy for 5-fold cross-validation for ResNet18

Following from above results, the original ResNet18 model was trained using Adam with a learning rate of 0.0001 for 25 epochs. Weights are saved for the epoch with best validation accuracy and the last epoch. The results are as shown below.

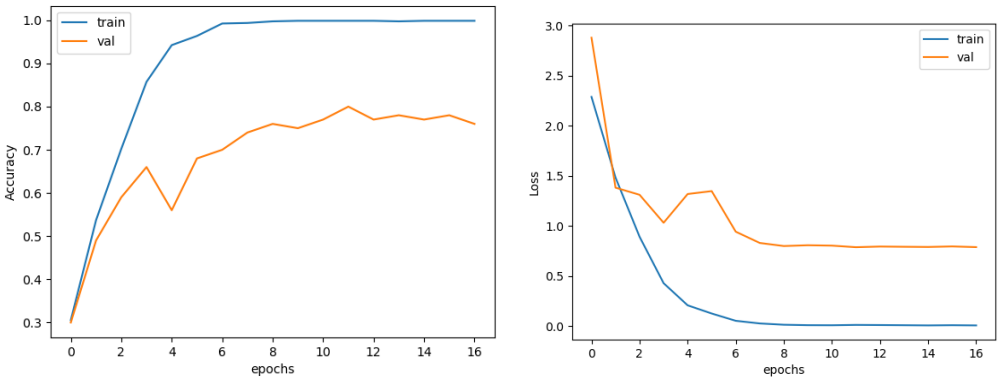
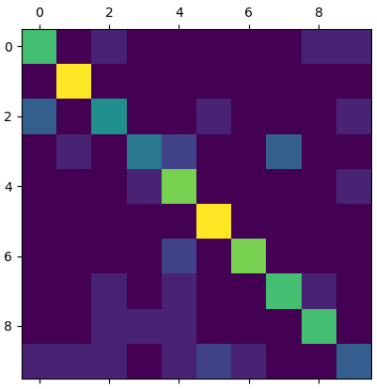


Figure 7. Performance curves for training and validation



Classification Report

	precision	recall	f1-score	support
0	0.64	0.70	0.67	10
1	0.83	1.00	0.91	10
2	0.56	0.50	0.53	10
3	0.67	0.40	0.50	10
4	0.53	0.80	0.64	10
5	0.77	1.00	0.87	10
6	0.89	0.80	0.84	10
7	0.70	0.70	0.70	10
8	0.78	0.70	0.74	10
9	0.50	0.30	0.37	10
accuracy			0.69	100
macro avg	0.69	0.69	0.68	100
weighted avg	0.69	0.69	0.68	100

Figure 8. Confusion matrix and performance for testing

3.2.2 Inception Module

As ResNet is deep spatial model, we wanted to see the performance of a wider model. An inception module is a neural network block developed by Google which approximates an optimal local sparse structure in a CNN by splitting the output and processing each part using a different size of convolution layer. In the inception module shown in Figure 6, we have made use of 1x1 convolutions to reshape the depth of the input, therefore reducing the number of parameters and computations required and still providing good performance.

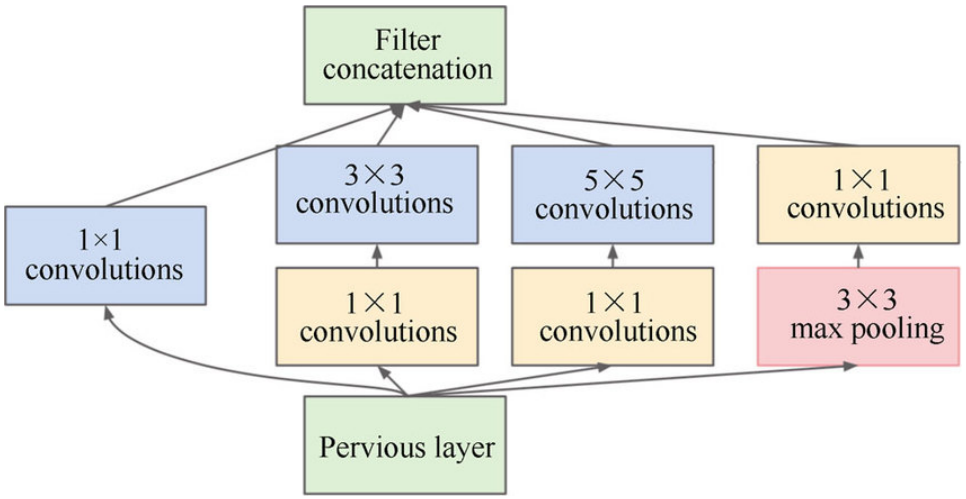


Figure 9: Inception module

In our model, we have implemented an inception module and used it in three different architectures to explore and improve the models performance.

In Single Inception module architecture (shown in Figure 10), we made use of multiple CNNs and then feed the output to the Inception module. We made use of only one inception module as it is heavy in computation. We made use of adaptive average pooling 2-dimensional layer to get the aggregated output from the image processing and then fed the output to linear neural layers to get the output for genre classification.

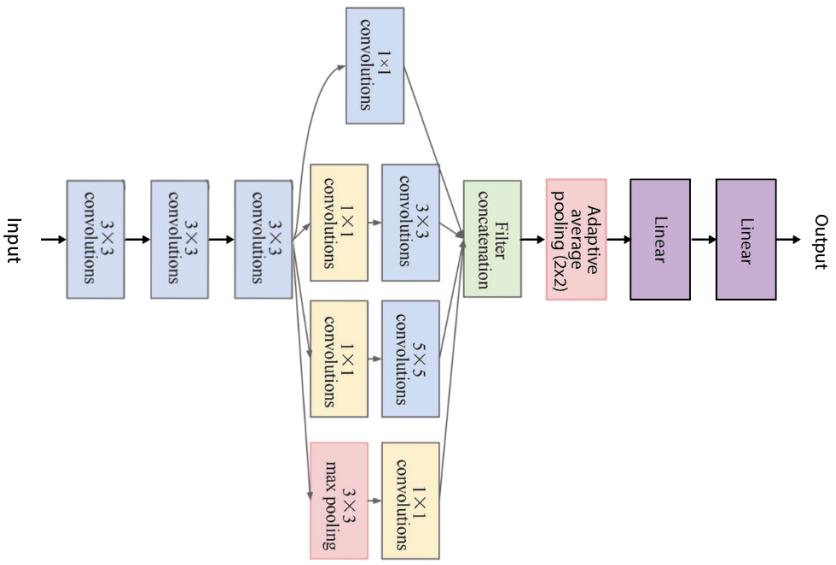


Figure 10: Single Inception module architecture

To make the previous model more complex, we decided to make use of 3 inception modules in sequence with skip connections (shown in Figure 11). Skip connections is a concept used in ResNet architecture which alleviates gradient vanishing problem and also prevents over calculation of extracted features if they are in the ideal processed form as we add the input to the output of the inception module therefore if the features are in the ideal form, the inception module can learn weights such that all the parameters are 0, allowing it to replicate an identity matrix and reproducing the same output as the given input. This model has fewer parameters than the previous model as it eliminates one convolution layer and replaces it with 2 inception layers which have fewer weights but they increase the depth of the network, allowing it to learn more high-level features.

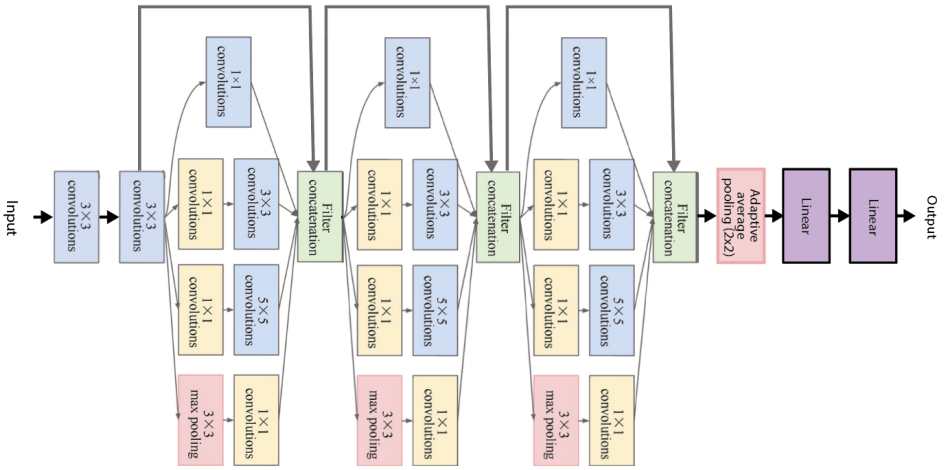


Figure 11: Multiple Inception modules with skip connections

In the third architecture (shown in Figure 12), we went for a different approach where we made use of 3 inception modules which increase the depth of the feature map. This model does not have skip connections.

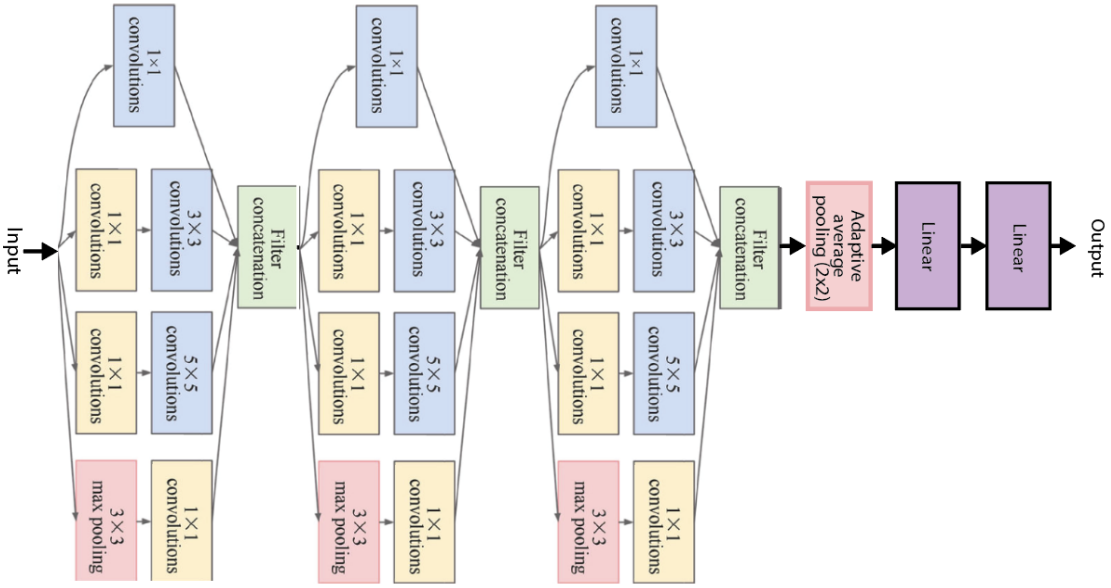


Figure 12: Multiple inception modules with growing filter size architecture

Model	Number of parameters	Validation Accuracy	Test Accuracy
Single Inception module	4,355,138	0.72	0.64
Three Inception modules with skip connections	2,436,034	0.79	0.67
Three Inception modules	2,311,170	0.36	0.28

Table 8: Table comparing different Inception modules

After training the three architectures for 25 epochs using SGD optimizer with learning rate = 0.001 and early stopping with patience = 5, three inception modules with skip connections performed the best. Using this model architecture, we hyperparameter tuned the model using grid search with cross validation to choose between Adam and SGD as the optimizer and appropriate learning rate.

Learning rate	**0.0001**	**0.0005**	**0.001**
Adam	18.46	21.36	12.57
SGD	27.81	29.36	**30.93**

Table 9: Hyperparameter tuning of learning rate and optimizer for three inception modules with skip connections

Following from above results, the model was trained using SGD with a learning rate of 0.001 for 25 epochs. Weights are saved for the epoch with best validation accuracy and the last epoch. The results are as shown in Figure 10 and 11.

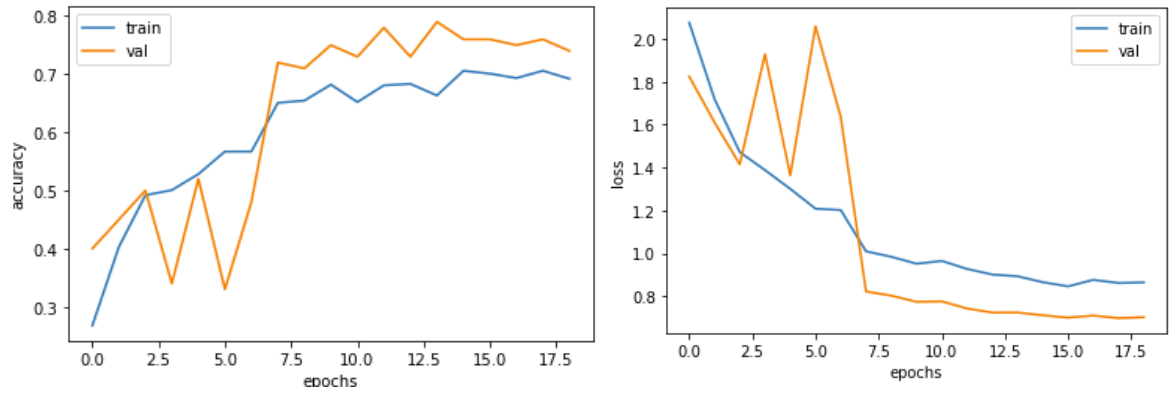
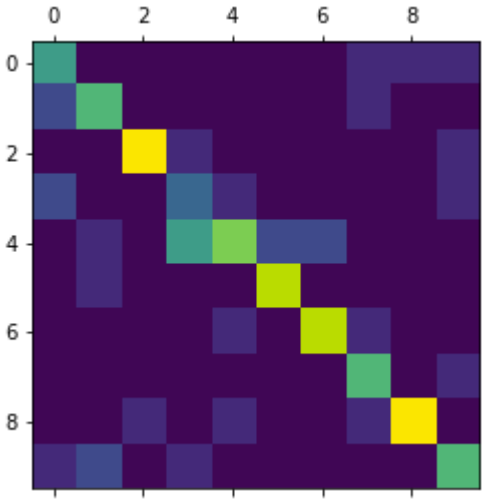


Figure 13. Performance curves for training and validation



Classification Report

	precision	recall	f1-score	support
0	0.50	0.62	0.56	8
1	0.60	0.67	0.63	9
2	0.90	0.82	0.86	11
3	0.30	0.43	0.35	7
4	0.70	0.41	0.52	17
5	0.80	0.89	0.84	9
6	0.80	0.80	0.80	10
7	0.60	0.86	0.71	7
8	0.90	0.75	0.82	12
9	0.60	0.60	0.60	10
accuracy			0.67	100
macro avg	0.67	0.68	0.67	100
weighted avg	0.69	0.67	0.67	100

Figure 14. Confusion matrix and performance for testing

3.2.3 Custom CNN

This CNN model is motivated by the CNN structures in current literature used for music tagging and genre classification. Choi et al. 2017 proposed a Fully Convolutional Network (FCN) with 2D convolution layers for music tagging. Choi et al. 2016 implemented a CNN architecture of five convolutional layers and two fully connected layers for the genre classification task.

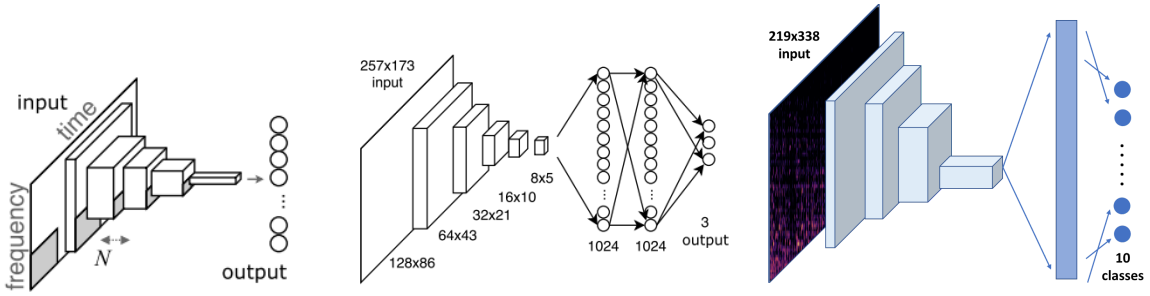


Figure 15. (a) FCN, Choi et al. 2017 (b) CNN, Choi et al. 2016 (c) Our model

Our implementation consists of 4 convolutional blocks (Conv2D - BatchNormalisation - ReLU - MaxPool2D) and one fully-connected layer. Each convolutional layer uses 3x3 convolution kernels with stride of 1 and same zero-padding. After each convolutional layer, batch normalization is applied to provide regularization effect and ReLU activation is used to introduce non-linearity. The feature maps are gradually downsampled at each layer using max-pooling with increasing pooling window sizes and stride (see Table 10). The last convolutional block produces 512 feature maps of 1x2 dimension, which are flattened and passed to a fully connected layer with ReLU activation and dropout of 0.5 before the final output layer.

	Layer 1	**Layer 2**	**Layer 3**	**Layer 4**
Pooling window	3, 3	2, 3	5, 6	6, 6

Pooling stride	2, 2	2, 2	5, 6	5, 4

Table 10. Downsampling using max-pooling

The weights for all Conv2d and Linear layers are initialized using Kaiming initialisation since all activations in the model are ReLU. Initially the number of kernels in the convolutional layers was set to <32, 64, 128, 256> but was reduced to <16, 32, 64, 128> so as to keep the number of trainable parameters small because the dataset is limited. The model has a total of 131,818 trainable parameters. For the loss function, we use categorical cross entropy since genre classification is a multi-class classification task.

We attempted hyperparameter tuning to select the optimiser and learning rate. Grid search with k-fold cross validation was implemented using the dataset reserved for training and validation. In the stratified 5-fold cross validation, the model was trained only for 10 epochs and without dropout applied to the FC layer. The average test accuracy was calculated (see table below) and used as an estimate to compare the performance of the model with different combinations of optimiser and learning rate.

Learning rate	**0.0001**	**0.0005**	**0.001**
Adam	22.02	24.47	**27.02**
SGD	7.78	10.89	13.68

Table 11. Average test accuracy for 5-fold cross-validation

Following from above results, the model (with dropout of 0.5 applied to FC layer) was trained using Adam with a learning rate of 0.001 for 25 epochs. Weights are saved for the epoch with best validation accuracy and the last epoch. The results are as shown below.

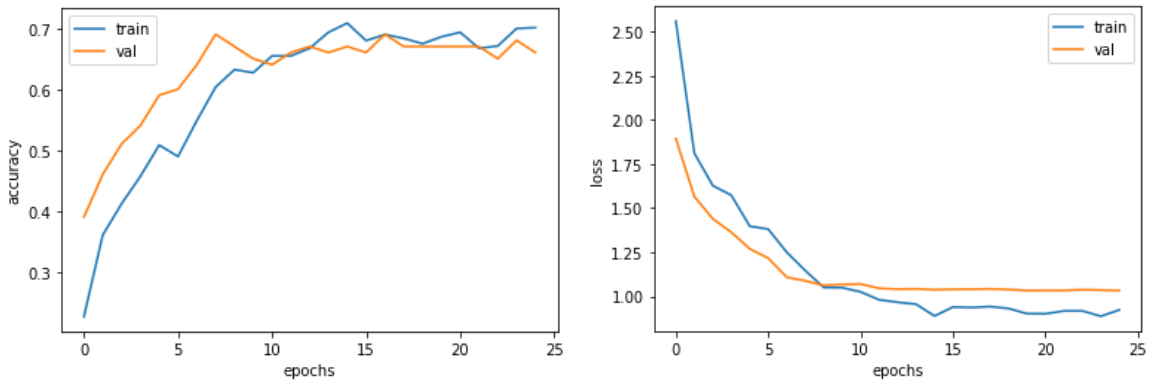
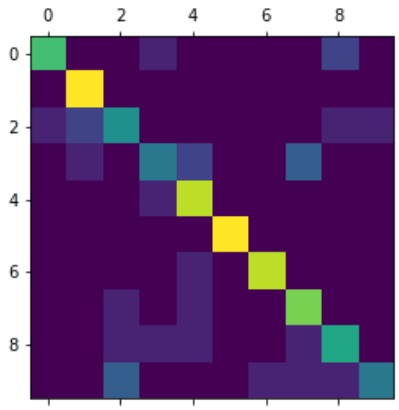


Figure 16. Performance curves for training and validation



Classification Report

	precision	recall	f1-score	support
blues	0.88	0.70	0.78	10
classical	0.77	1.00	0.87	10
country	0.50	0.50	0.50	10
disco	0.57	0.40	0.47	10
hiphop	0.64	0.90	0.75	10
jazz	1.00	1.00	1.00	10
metal	0.90	0.90	0.90	10
pop	0.62	0.80	0.70	10
reggae	0.60	0.60	0.60	10
rock	0.80	0.40	0.53	10
accuracy			0.72	100
macro avg	0.73	0.72	0.71	100
weighted avg	0.73	0.72	0.71	100

Figure 17. Confusion matrix and performance for testing

4. Model Comparison

No.	Approach	Validation Accuracy	Testing Accuracy	Number of parameters
1	Single-layered GRU with 256 hidden dim	0.24	0.31	368,906
2	ResNet18 with 4 layers	0.72	0.74	12,852,784
3	Multiple Inception modules with skip connections	0.79	0.67	2,436,034
1	Custom CNN	0.69	0.72	131,818

Table 12. A Table for model comparison

Going through our top four models, we can observe that Resnet18 with 4 layers performs the best with high test and validation accuracy which is very comparable with Custom CNN which has slightly worse validation and testing accuracy and almost hundred times less parameters, making it computationally efficient with a great performance. The other two models are performing worse than Custom CNN with more parameters than Custom CNN making it computationally less efficient. Therefore, we selected Custom CNN for genre classification.

The Mel-spectrogram represents audio clips as frequency on the y-axis and time on the x-axis. A 2D CNN model allows time and frequency invariances in different scales by gradual 2D sub-samplings (Choi et al. 2017). It will be able to capture both temporal (song progression over time) and spatial (across frequency range) features from the Mel-spectrogram. It is possible to use a 1D CNN structure by moving along the time axis, however this would result in information of the frequency range being compressed. At least to the human eye, there seems to be distinguishing information along the frequency axis for samples in the dataset particularly those samples which have repeating patterns along the time axis (see Figure 18). It would have been interesting to examine the convolution kernels to understand which musical properties are being learned, however we were not able to do so for this project.

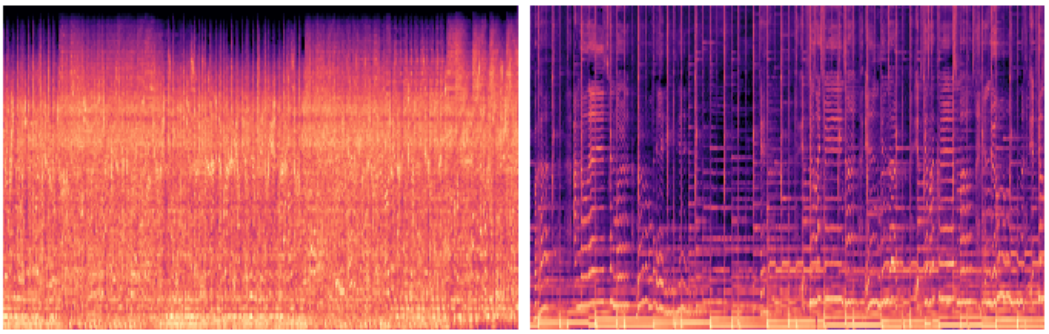


Figure 18. Mel-spectrograms of (a) metal00018 (b) pop00013

5. Song Recommendation

Our initial idea was to predict the genre of an input song and then recommend songs by randomly selecting *k* songs from the list of songs in the predicted genre, based on the assumption that a listener is likely to enjoy songs belonging to the same genre. However, using a model trained on the genre classification task as an encoder will provide better recommendation for songs that lie on the boundaries of the genre space. This is because the model should have learned to recognise musical characteristics that are shared amongst songs although they may be classified as different genres, especially those songs which could belong to more than one genre.

We selected the CustomCNN model[1] and trained the model on the genre classification task. After the training was finished, we removed the last linear layer from the model and took the output of the model which has a dimension of 1 x 128 as the embedding for the input song. This allows us to create a fingerprint of each song with the help of our classification task which acts as the fake task. After extracting the song embedding for all the songs in our dataset, we normalize each embedding and store them in a matrix format where each row is the embedding of a song. We store the song embeddings and the song names in pickle format to use it later.

We will be using the song embeddings to find song similarity scores between a new song and the songs in the dataset and list out the top-k songs. To calculate the song similarity between a query song and the songs from our dataset, we performed matrix multiplication using numpy, making it computationally efficient to get similarity scores. The song similarity score of a pair of songs is the dot product of two normalized vectors i.e., it is the cosine similarity of the song embeddings. From the cosine similarity scores between the query and each song in our dataset, we get the top-k song recommendations by our solution.

6. Discussion

Embedding evaluation

To evaluate the embeddings, we calculated the top 5 recommended songs for each song in the dataset. We checked whether the top 5 songs (excluding the song itself) are from the same genre and gave it a score by using the formula number of same genre songs / total number of songs. We took an average of the score for each genre and an overall average score. The overall average is 0.78 which means approximately 4 out of 5 songs suggested were from the same genre.

[1] We chose to use this model despite it not having the best performance, because (i) it did not suffer greatly from overfitting (ii) it had the smallest number of trainable parameters, hence less computation resources. Also, it had the simplest structure for deconvolution of filters although we were unable to implement deconvolution for the project.

```
blues 0.774
classical 0.960
country 0.616
disco 0.558
hiphop 0.736
jazz 0.867
metal 0.884
pop 0.732
reggae 0.668
rock 0.436
average score : 0.722
```

Figure 19. Evaluation of song encodings

Embedding visualization

We can observe clusters in the t-SNE plot (see Figure 21) corresponding to most genres of the 10 available genres. We also observe that genres for which points are spread across the space have lower evaluation scores - particularly rock, country and disco.

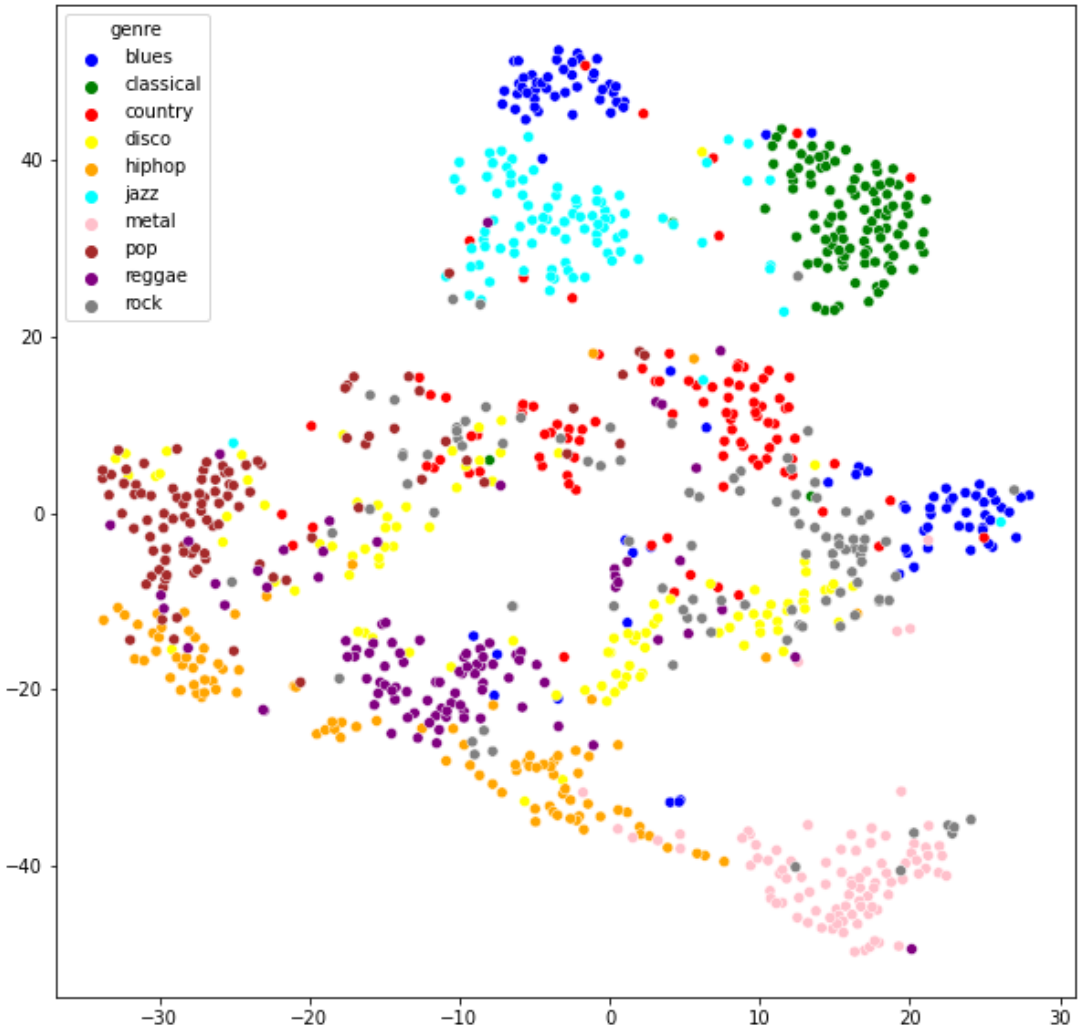


Figure 20. Visualization of song encodings using t-SNE

To further understand why some of the songs are so distant from their genre, we investigated a few examples with the lowest score. “Rock00002” has 5 recommended songs from the genre “disco”. After listening to the clips to understand, “disco00049” had the

highest similarity score and the beat of the song was very similar to “rock00002”. In another example, “rock00003” is rather slow paced for a rock song and “country00084” has the highest similarity score which is rather fast paced for a country song, therefore the pace of both the songs is similar.

7. Conclusion

In summary, we have tried RNN, CNN integrated with Inception Module, Custom CNN and ResNet18. After a holistic model comparison, we think custom CNN is able to output a relatively better and more stable result with an efficient computation time. The selected custom CNN model will be using extracted song embeddings to produce similarity scores between the selected song and the rest in the datasets. The performance of our embedding is satisfying as majority of the songs are from the same genre and the rest rest share a similar music patterns.

8. References

K. Choi, G. Fazekas, M. Sandler and K. Cho, "Convolutional recurrent neural networks for music classification," 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017, pp. 2392-2396, doi: 10.1109/ICASSP.2017.7952585.

Choi, K., Fazekas, G., & Sandler, M.B. (2016). Explaining Deep Convolutional Neural Networks on Music Classification. *ArXiv, abs/1607.02444*.

Pierre O. (2019, March). Github. How to initialize deep neural networks? Xavier and Kaiming initialization. Retrieved from: https://pouannes.github.io/blog/initialization/#xavier-and-kaiming-initialization

Gaurav S. (2020, May).Pluralsight.com.Transfer Learning with ResNet in PyTorch. Retrieved from: https://www.pluralsight.com/guides/introduction-to-resnet

9. Appendix

Appendix 1: Additional Data Visualizations

This appendix includes extra data visualizations we have done for knowing about the dataset. It serves as an extra reference when you refer to 2.2 Data visualization.

Let's have an overall correlation graph for feature mean values:

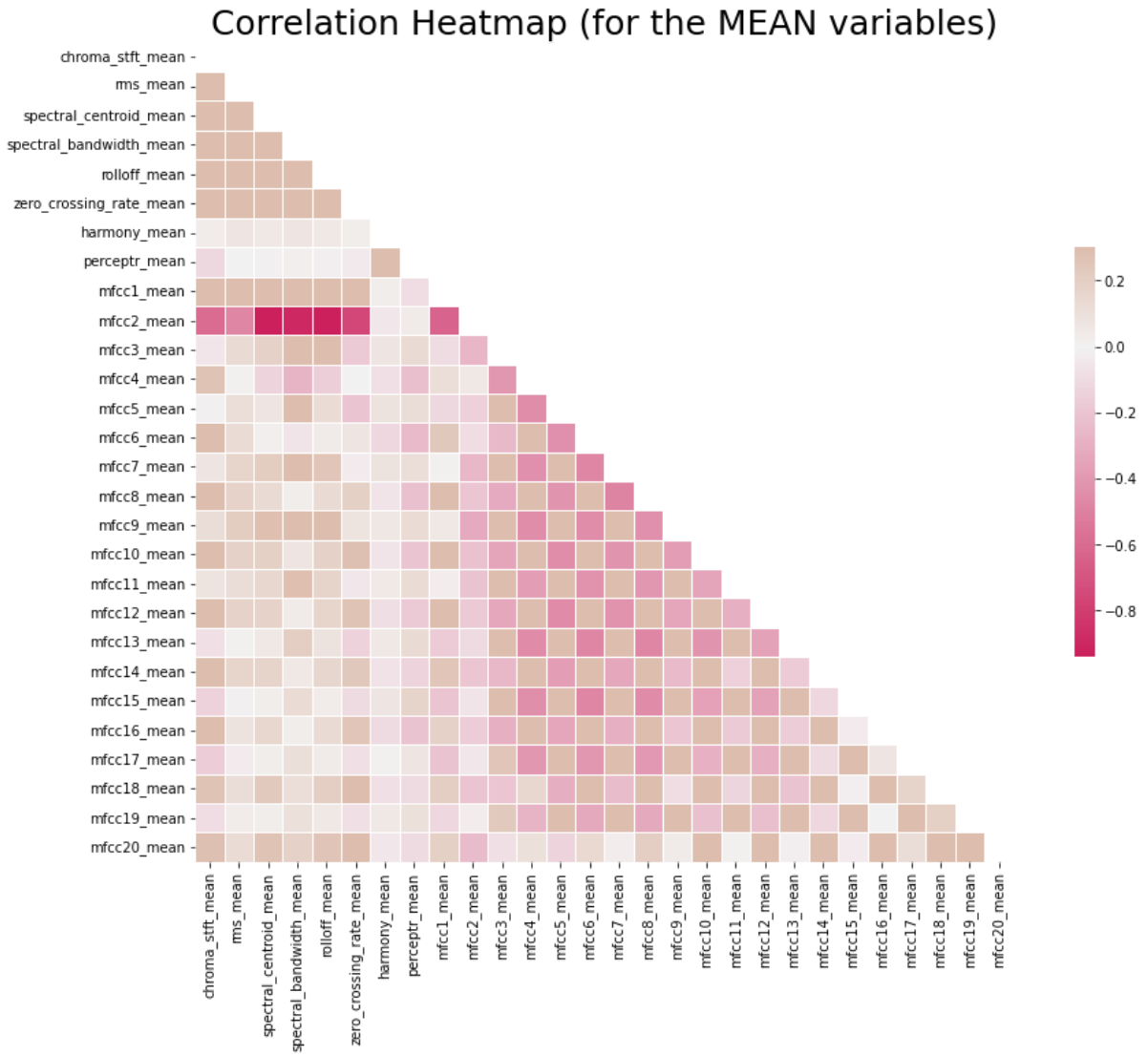
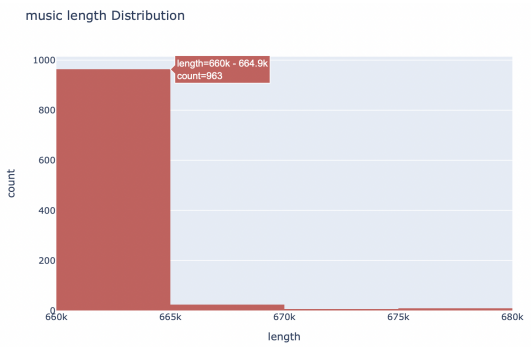
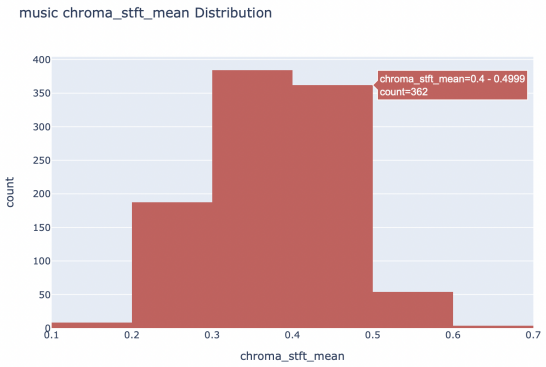
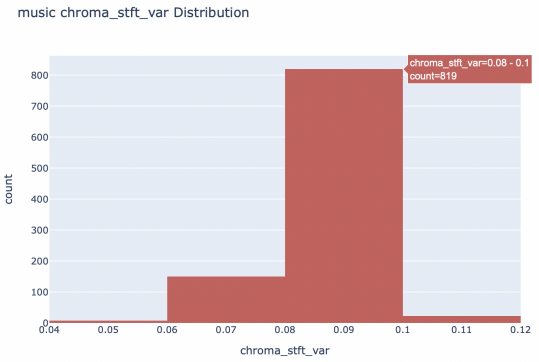
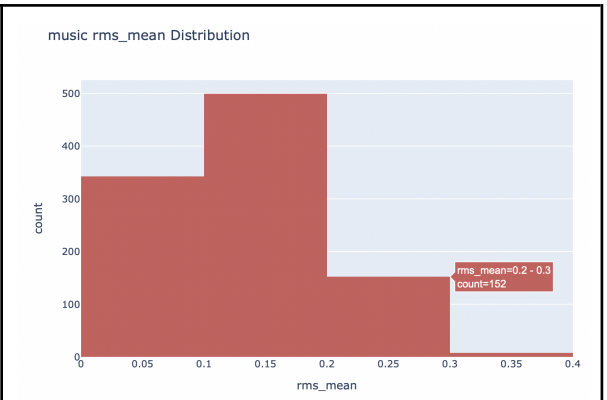
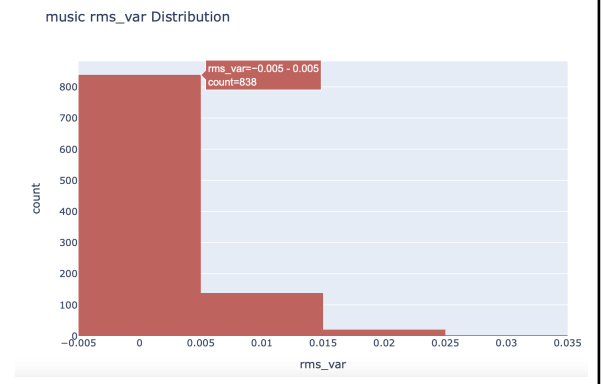
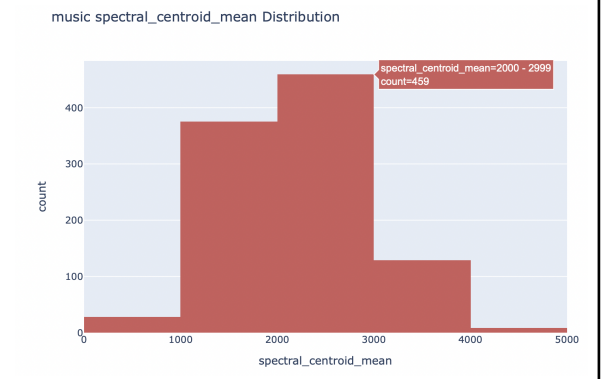
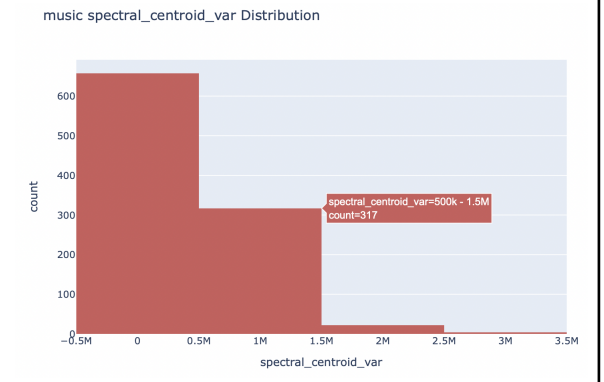


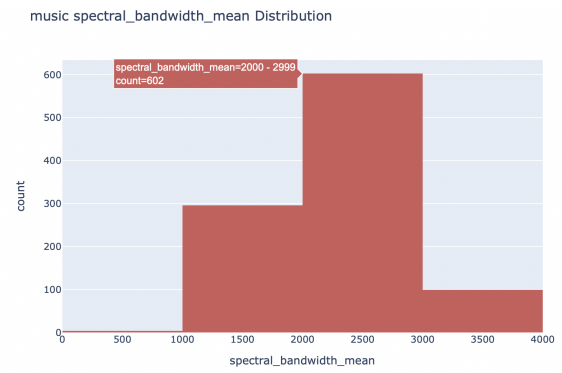
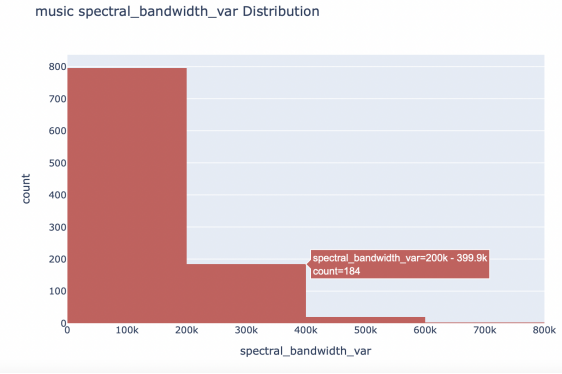
Fig: Correlation Heatmap for the mean variables of audio attributes

Therefore, we know that perceptr_mean column is the most important feature among all mean variables. Since the logic is the same, we may not do the similar plotting for the variance variables to save some report space. To note, the other correlation map reflects that perceptr_var column has the most feature importance among all var variables.

We also had an understanding of how many audio files are residing in a specific range of corresponding values after visualizing the audio attributes as a whole. The attributes we have achieved after librosa processing include ‘length’, ‘chroma_stft_mean’, ‘chroma_stft_var’, ‘rms_mean’, ‘rms_var’, ‘spectral_centroid_mean’, ‘spectral_centroid_var’, ‘spectral_bandwidth_mean’, ‘spectral_bandwidth_var’.

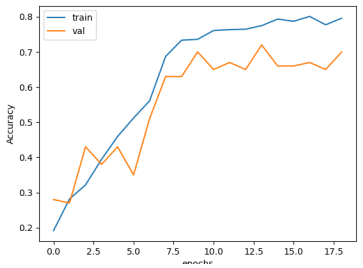
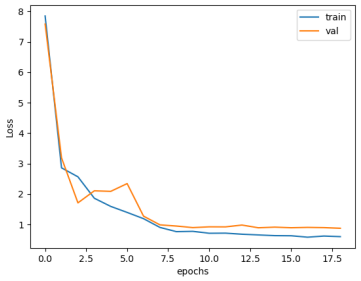
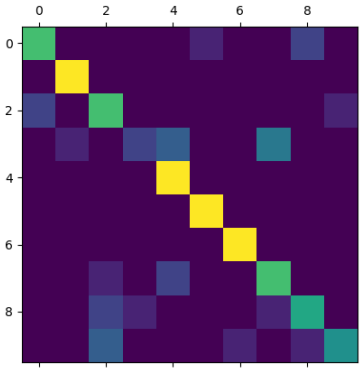
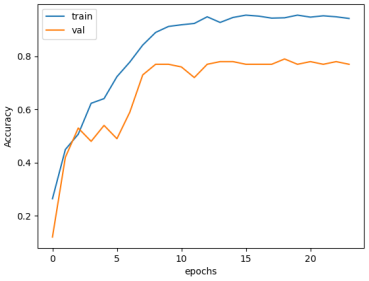
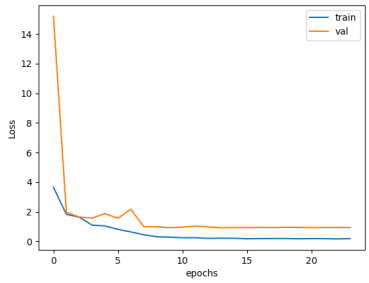
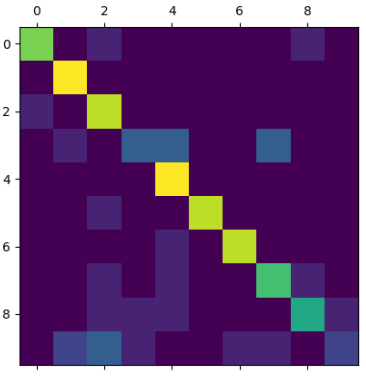
Distribution Graph Type	
Music Length Distribution	
Music Chroma_stft_mean Distribution	
Music Chroma_stft_var Distribution	

Music rms_mean Distribution	
Music rms_var Distribution	
Music spectral_centroid_mean Distribution	
Music spectral_centroid_var Distribution	

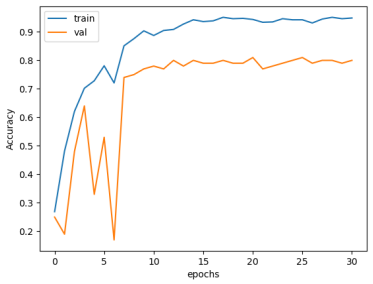
Music spectral_bandwidth_mean Distribution	
Music spectral_bandwidth_var Distribution	

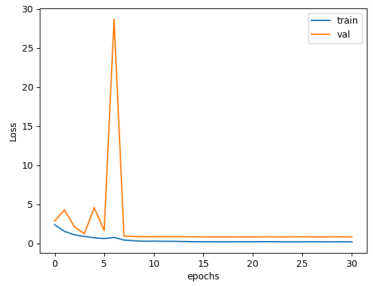
Appendix 2: Summary Performance Table of ResNet

This appendix includes all performance graphs we have tried for Resnet18. It serves as an extra reference when you refer to 3.3.2 Residual Network.

Model	Validation	Test
ResNet18 – Original	 	Classification Report precision recall f1-score support 0 0.78 0.70 0.74 10 1 0.91 1.00 0.95 10 2 0.54 0.70 0.61 10 3 0.67 0.20 0.31 10 4 0.67 1.00 0.80 10 5 0.91 1.00 0.95 10 6 0.91 1.00 0.95 10 7 0.58 0.70 0.64 10 8 0.67 0.60 0.63 10 9 0.83 0.50 0.62 10 accuracy 0.74 100 macro avg 0.75 0.74 0.72 100 weighted avg 0.75 0.74 0.72 100 
ResNet18 – Removed 1 layer	 	Classification Report precision recall f1-score support 0 0.89 0.80 0.84 10 1 0.77 1.00 0.87 10 2 0.56 0.90 0.69 10 3 0.60 0.30 0.40 10 4 0.62 1.00 0.77 10 5 1.00 0.90 0.95 10 6 0.90 0.90 0.90 10 7 0.64 0.70 0.67 10 8 0.75 0.60 0.67 10 9 0.67 0.20 0.31 10 accuracy 0.73 100 macro avg 0.74 0.73 0.71 100 weighted avg 0.74 0.73 0.71 100 

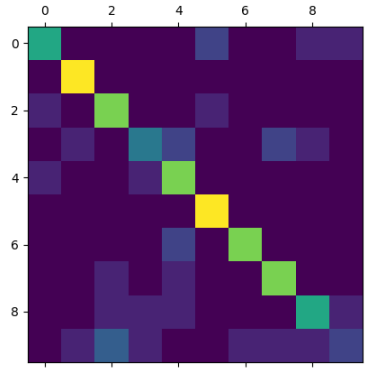
ResNet18 – Removed 2 layers



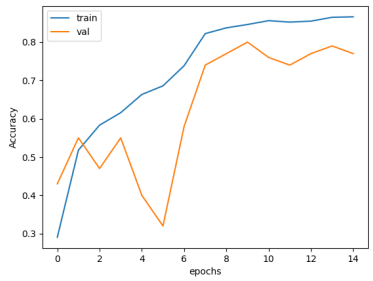


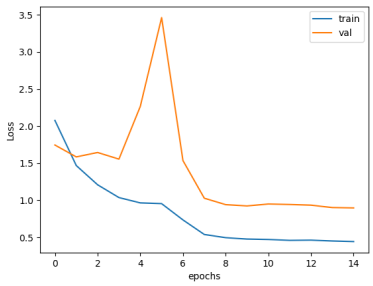
Classification Report

	precision	recall	f1-score	support
0	0.75	0.60	0.67	10
1	0.83	1.00	0.91	10
2	0.62	0.80	0.70	10
3	0.57	0.40	0.47	10
4	0.57	0.80	0.67	10
5	0.77	1.00	0.87	10
6	0.89	0.80	0.84	10
7	0.73	0.80	0.76	10
8	0.67	0.60	0.63	10
9	0.50	0.20	0.29	10
accuracy			0.70	100
macro avg	0.69	0.70	0.68	100
weighted avg	0.69	0.70	0.68	100



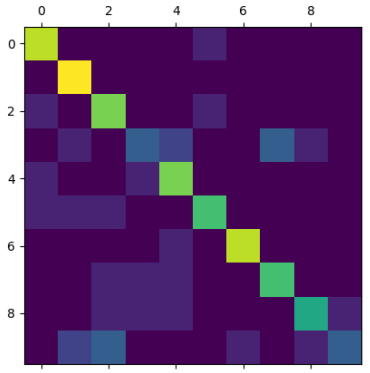
ResNet18 – Removed 3 layers





Classification Report

	precision	recall	f1-score	support
0	0.75	0.90	0.82	10
1	0.71	1.00	0.83	10
2	0.57	0.80	0.67	10
3	0.50	0.30	0.37	10
4	0.62	0.80	0.70	10
5	0.78	0.70	0.74	10
6	0.90	0.90	0.90	10
7	0.70	0.70	0.70	10
8	0.75	0.60	0.67	10
9	0.75	0.30	0.43	10
accuracy			0.70	100
macro avg	0.70	0.70	0.68	100
weighted avg	0.70	0.70	0.68	100



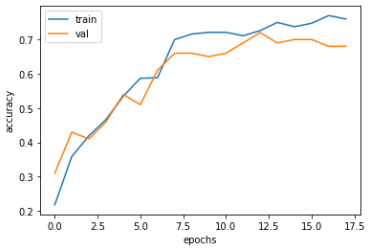
Appendix 3: Summary Performance Table Of ConvNet

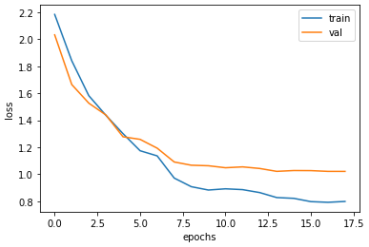
This appendix includes more performance graphs we have tried for CNN. It contains different combinations of the number of CNN layers and the number of fully connected layers. Also we have combined the CNN with the inception modules. It serves as an extra reference when you refer to 3.2 Convolutional Neural Network.

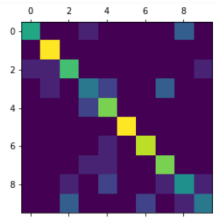
<table>
<tr><th>Model</th><th>Validation</th><th>Test</th></tr>
<tr>
<td>

4 Conv2d + 2 FC layers

- Larger number of kernels per layer <32, 64, 128, 256>
- No weight initialization
- No dropout
- Using SGD optimizer with 0.001 learning rate and 0.9 momentum
- 785,354 trainable parameters

</td>
<td>




</td>
<td>


Classification Report

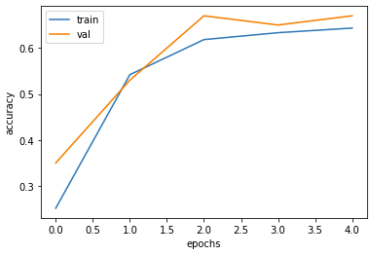
	precision	recall	f1-score	support
blues	0.86	0.60	0.71	10
classical	0.83	1.00	0.91	10
country	0.64	0.70	0.67	10
disco	0.50	0.40	0.44	10
hiphop	0.57	0.80	0.67	10
jazz	1.00	1.00	1.00	10
metal	0.82	0.90	0.86	10
pop	0.67	0.80	0.73	10
reggae	0.56	0.50	0.53	10
rock	0.67	0.40	0.50	10
accuracy			0.71	100
macro avg	0.71	0.71	0.70	100
weighted avg	0.71	0.71	0.70	100

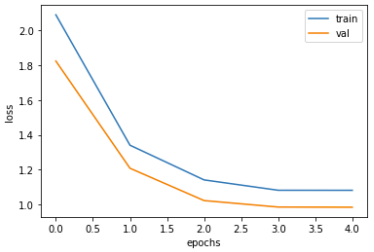
</td>
</tr>
<tr>
<td>

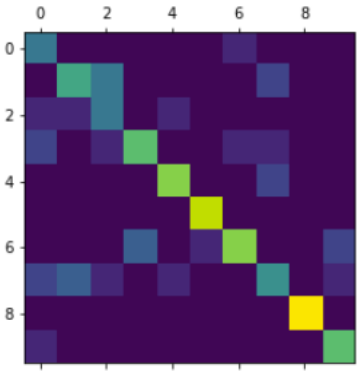
3 CNN + Inception module

- Multiple CNN
- 4.3 mil parameters

```
Epoch 23/24
----------
train Loss: 1.0710 Acc: 0.6733
val Loss: 0.9902 Acc: 0.6700

Epoch 24/24
----------
train Loss: 1.0822 Acc: 0.6533
val Loss: 0.9992 Acc: 0.6600

Training complete in 18m 8s
Best val Acc: 0.710000
```

</td>
<td>




</td>
<td>


Classification Report

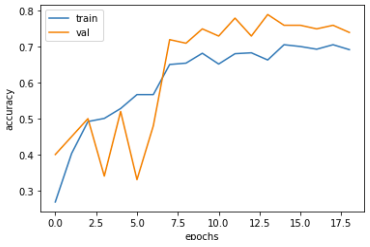
	precision	recall	f1-score	support
0	0.40	0.80	0.49	5
1	0.60	0.50	0.55	12
2	0.40	0.57	0.47	7
3	0.70	0.58	0.61	12
4	0.80	0.80	0.80	10
5	0.90	1.00	0.95	9
6	0.80	0.57	0.64	14
7	0.50	0.38	0.41	13
8	1.00	1.00	1.00	10
9	0.70	0.88	0.78	8
accuracy			0.64	100
macro avg	0.67	0.66	0.66	100
weighted avg	0.70	0.61	0.65	100

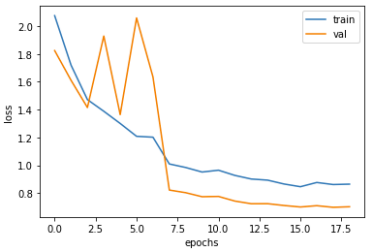
</td>
</tr>
</table>

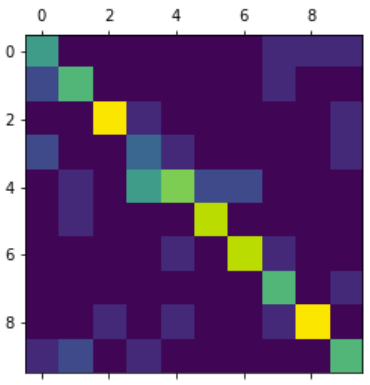
2 CNN + 3 Inception module with skip connections

- Multiple CNN
- 4.3 mil parameters

```
Epoch 22/24
----------
train Loss: 1.0788 Acc: 0.6508
val Loss: 1.1106 Acc: 0.6100

Epoch 23/24
----------
train Loss: 1.0930 Acc: 0.6333
val Loss: 1.1082 Acc: 0.6400

Epoch 24/24
----------
train Loss: 1.0935 Acc: 0.6395
val Loss: 1.1100 Acc: 0.6200

Training complete in 18m 10s
Best val Acc: 0.640000
```





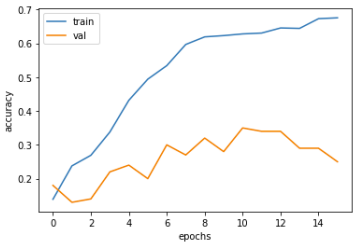


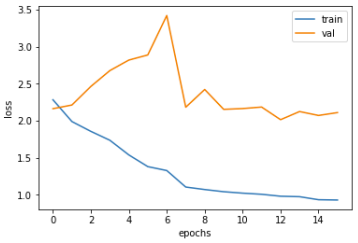
Classification Report

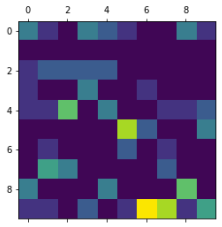
	precision	recall	f1-score	support
0	0.50	0.62	0.56	8
1	0.60	0.67	0.63	9
2	0.90	0.82	0.86	11
3	0.30	0.43	0.35	7
4	0.70	0.41	0.52	17
5	0.80	0.89	0.84	9
6	0.80	0.80	0.80	10
7	0.60	0.86	0.71	7
8	0.90	0.75	0.82	12
9	0.60	0.60	0.60	10
accuracy			0.67	100
macro avg	0.67	0.68	0.67	100
weighted avg	0.69	0.67	0.67	100

3 Inception modules in series

- Progressive increase in filter number for inception module
- 2.3 mil parameters







Classification Report

	precision	recall	f1-score	support
0	0.30	0.21	0.25	14
1	0.00	0.00	0.00	0
2	0.20	0.22	0.21	9
3	0.30	0.60	0.40	5
4	0.30	0.21	0.25	14
5	0.60	0.55	0.57	11
6	0.00	0.00	0.00	4
7	0.20	0.22	0.21	9
8	0.50	0.45	0.48	11
9	0.40	0.17	0.24	23
accuracy			0.28	100
macro avg	0.28	0.26	0.26	100
weighted avg	0.35	0.28	0.30	100

Appendix 4: Interesting Finding–A Recommender System in terms of Similarity Score (with no Deep Learning Method)

This appendix includes an initial recommender system we have built using sk-learn library solely. It is presented here just in case for your interests. Please feel free to skip otherwise

One of our proposed approaches was to use cosine similarity scores to build a recommender system. Such a system will calculate the similarity scores between one selected song with the rests. The output will display three songs and the highest value will be regarded as the most similar and thus, the recommended choice of the system.

```
#please feel free to choose any
find_similar_songs('blues.00000.wav')
Audio('blues.00000.wav')
```

```
The recommended song based on your habits of listening blues.00000.wav :
filename
disco.00088.wav      0.765158
rock.00074.wav       0.737537
country.00055.wav    0.729619
Name: blues.00000.wav, dtype: float64
```


```
# display audio files
Audio('disco.00088.wav')
```