Deep learning based music genre classification using spectrogram

INTRODUCTION:

The main motivation for this project is to create a music genre classification model,using Deep Learning concepts, that can accurately predict the genre of an input audio with the help of the features obtained from its spectrogram. To implement this model, we have used the famous GTZAN audio data set which has 1000 audio samples that have already been classified among 9 possible genres(blues,classical,country,disco,hip hop,jazz,metal,pop,reggae,rock). As a part of data preprocessing, we identified which of the 1000 .wav files are unreadable and then removed them from the data set - leaving behind 981 audio samples. The features of the dataset were extracted from the spectrogram of the audio file. Since it isn't feasible to extract all the features of the data set, Mel Frequency Cepstral Coefficients(MFCCs) of the spectrogram were computed(with the help of librosa library in python) to find the most dominant features. The data was then split into training and testing data in a 75-25 ratio while training data was further split into a validation set in a 80-20 ratio. As a part of building the model, keras library in tandem with tensorflow were used to build the Convolutional Neural Network which has a great number of advantages in image classification related problems(discussed more deeply in the upcoming sections). The model was built(CNN updates the weights based on validation set error) with 9 output features(to denote the 10 genres) and the model was implemented on the testing data. To compute the efficiency, we used a confusion matrix to find all the false positives and true positives on the training dataset. As an improvement, we use Recurrent Neural Networks to show how this updated model has higher accuracy(discussed more deeply in upcoming sections) with respect to results computed from the confusion matrix.

STEP 1: OBTAINING THE DATA AND PRE-PROCESSING

The dataset was obtained from GTZAN Dataset - Music Genre Classification | Kaggle. Within this dataset there is a collection of audio files of 100 songs for each of the 10 genres. The genres of the dataset are: blues,classical,country,disco,hip hop, jazz, metal, pop , reggae and rock. Each of the audio samples are of 30 seconds and is in a .wav format. Before proceeding, one must check whether certain audio files are corrupted and remove them(which was done using librosa-audio processing library).This left us with 990 audio samples in total.

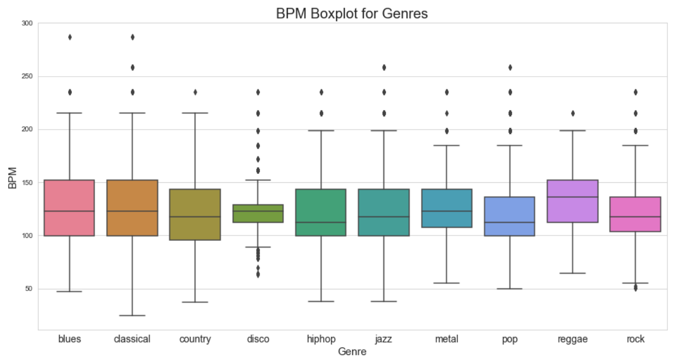


Fig 1: Beats Per Minute for Each Genre

The next step was to visualize how some of the audio signal spectrograms would look like.
A spectrogram is a visual representation of the frequency signatures present in an audio signal over time. This essentially represents the visualized form(an image) of which frequency components contributed to most power in the signal at a particular instance of time in the form of a heatmap structure. The spectrogram can be obtained by taking the Fourier Transform of the audio signal and assigning a color on the heatmap scale to each frequency component. Spectrograms are of high importance because it isn't feasible to use audio in the time domain as an input for the neural network due to the large sampling rate required to process the signal which will mean a larger input vector to the Neural Networks. Instead we can use the spectrogram to be a good data input (since it contains information about all frequency components in the signal) to Neural Networks(specifically Convolutional Neural Networks).
With regards to obtaining the **frequency components, we can use Fast Fourier Transform(FFT) to obtain a dynamic frequency spectrum that gives information on when a certain frequency signature was contributing the most power**. It is generally preferred to take the log amplitude of the peaks of the signatures as we can get more precise information.

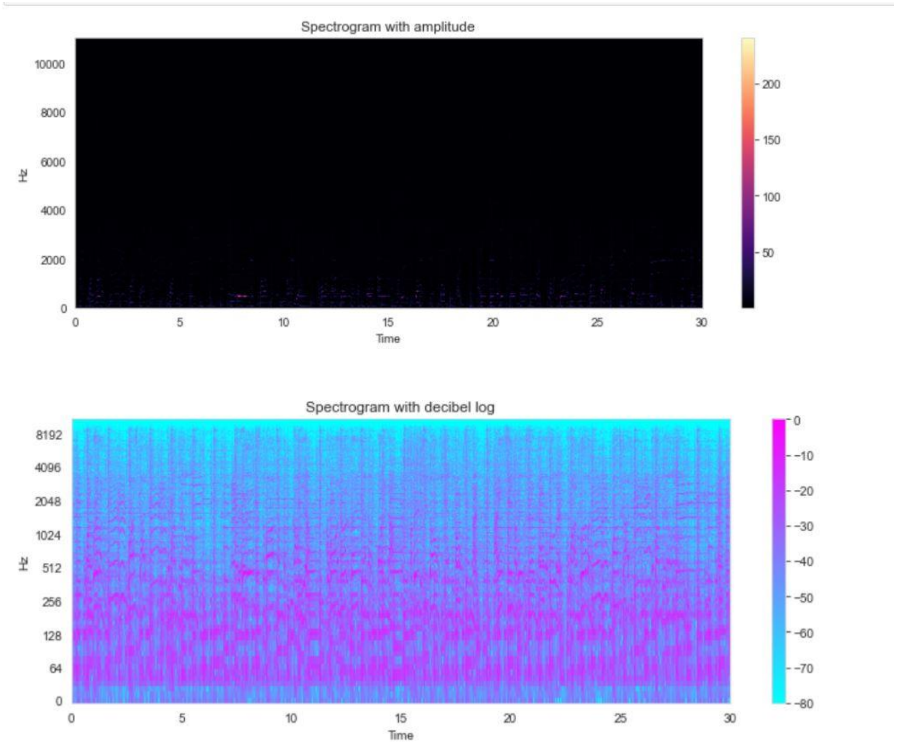


Fig 1a and 1.b representing spectrogram

So now we have spectrogram data which we can feed into the image specific neural network known as CNN by extracting the required features.

However, the main disadvantage of a normal spectrogram is it isn't perceptually relevant - meaning since spectrograms perceive frequencies linearly, it is not adequate to analyze features such as pitch that humans perceive frequency logarithmically. In order to obtain a more perceptually relevant audio spectrogram which can give more information, we will use Mel Spectrograms. Mel Spectrograms are generally built using the following steps: 1) Extract STFT 2) Convert amplitude to DB 3) Convert frequency to Mel Scale (which is done with the help of Mel Banks and Mel filters which ensure frequency is read logarithmically for human perception).

Both the Spectrogram and Mel-Spectrogram images are visualized with the help of Librosa.feature.stft and librosa.feature.mel spectrogram function available in librosa.

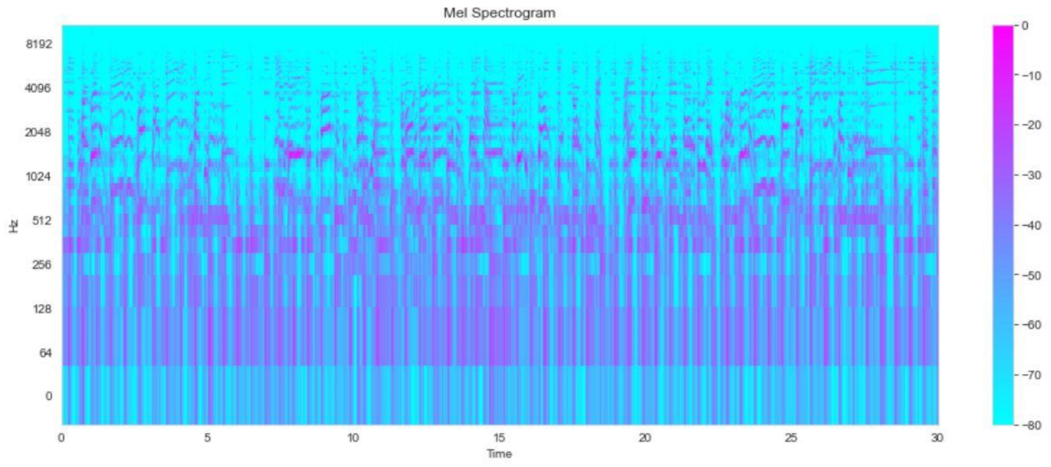


Fig 2. Mel Spectrogram

Now that we have the spectrogram, we will proceed to the important part of Feature Extraction from the spectrogram.

STEP 2: FEATURE EXTRACTION FROM SPECTROGRAMS

From the spectrograms of all the images, we will need to obtain the features and use them for modeling purposes. In the paper being implemented, features such as mel-spectrograms, spectral centroids, spectral rolloff and chromo frequencies have been used- totalling to all of **128 features**. However, this isn't feasible on **local systems due to the computational complexities, so we have performed PCA to extract those features which we consider hold the most information of the dataset.**

Principal Component Analysis and Mel Frequency Cepstral Coefficients:
While there exist 128 features, most of the features have high correlation amongst each other. While it is fine theoretically to build models using all features, computational constraints limit the ability to do so practically. Instead, there is a need to select a subset of all the features which are as relevant as possible - so that we can retain the most amount of information from data while reducing the complexity- and that is why we use **Principle Component Analysis** to select the features.
Mel Spectrograms provide us with the most perceptually relevant features w.r.t pitch which enables us to train and test the models. Therefore, we will use the Mel Spectrogram for our model computations.Mel Spectrograms usually represent 13-14 features of the original spectrogram. **The coefficients that will describe the mel spectrogram are called Mel Frequency Cepstral Coefficients(MFCCs).** MFCC coefficients are a 130X13 size matrix where each of the 13 layers contain the coefficients that correspond to each feature retained by the mel spectrogram. MFCCs have the advantages of describing the large structures of the spectrum while ignoring the finer spectral features. For this reason, it is extensively used in speech and music processing applications.

Librosa.features.mel spectrogram directly return a 13X130 matrix whose transpose denotes the Mel Frequency Cepstral Coefficients. So in our program, we **created a json file that has a dictionary to store all necessary data**: that will have three keys: mapping(to denote which MFCC matrix belongs to which genre), mfcc which stores all the 13X130 coefficients for all the 9990 audio files and the labels which denote the target attributes(i.e the genre) which we are trying to classify to.

Now that we have the features and target attributes, we will move onto the next important step, building the CNN model.

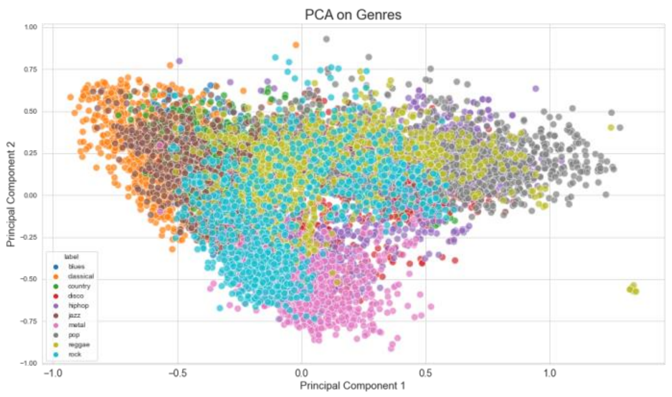


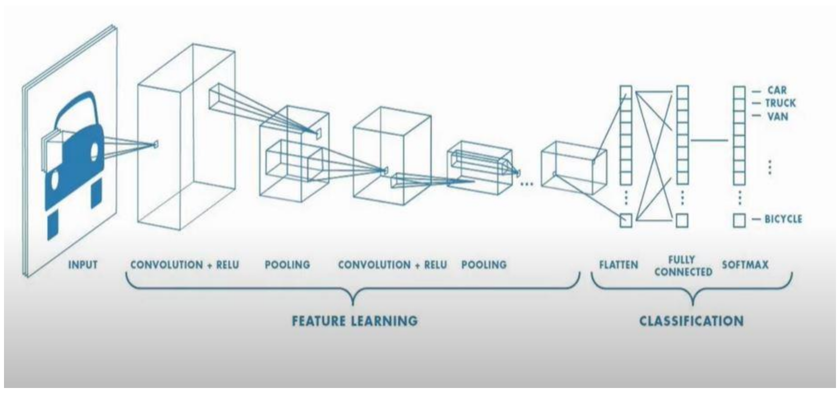
Fig 3. Denotes the first 2 PCAs onto which we can project all points

STEP 3 : BUILD THE CONVOLUTIONAL NEURAL NETWORK
Convolutional Neural Networks are used for processing images and perform much better than multilayer perceptron algorithms while also having lesser parameters than a dense layer. It is able to process audio signals in this case since we have converted the audio to it's spectrogram whose features are used to learn in CNNs. Since image data is structured, with no change in features with scale variation, it can emulate human vision systems. **CNNs have two major components- Convolution and pooling.** As part of convolution, we essentially apply a kernel(filter) of a grid of weights onto an image. The kernel is a small dimensional grid that will slide over all the pixels of an image to give us a convolutional pixel output. Based on the number of channels present in the image, we will use either a 2D Or 3D kernel to slide over the pixels. Kernels behave as feature detectors.From the adaptive kernel, the network learns the weights of the kernel and tries to minimize the error of misclassification as much as possible. Pooling is a method that helps us downsample the output feature data from the convolutional layer. We don't learn anything from pooling, but it helps us convert whatever features were learnt in the kernel back to the original form so it can be fed to the next layer of the CNN. One has to specify the number of channels of the image data- 3in case of RGB and 1 incase of GreyScale.
Each layer in any neural network must have an activation function. In classification problems, CNNs use the **Rectified Linear Unit(ReLU) activation function** given by:

$$ReLU(h) = \begin{matrix} 0 & if\ h < 0 \\ h & if\ h >= 0 \end{matrix}$$

ReLU has the advantages of better convergence and reduced likelihood for vanishing gradient. This is essential in many neurand neural networks as a vanished gradient can lead to difficulty in training more layers.

Once the CNN layers are completed, we flatten the output to be fed to the output layer where we will use the softmax function as we are doing classification. As we go deeper in the CNN, we see that the features inputted(at low level) are converted to more useful features holding more information(at higher level of information).



The entire implementation of the neural network was done with the help of tensorflow library and tensorflow.keras . The model was created using the keras.Sequential() command and then layers were added to the model with the help of the tensorflow.keras.layers.Conv2D command which adds a CNN layer to the neural network. We need to pool it before sending it to the next layer and this is done with the help of tensorflow.keras.layers.MaxPool2D. We then normalize the output with the help of tensorflow.keras.layers.BatchNormalization. Each of the CNN layers uses 'relu' as the activation function.

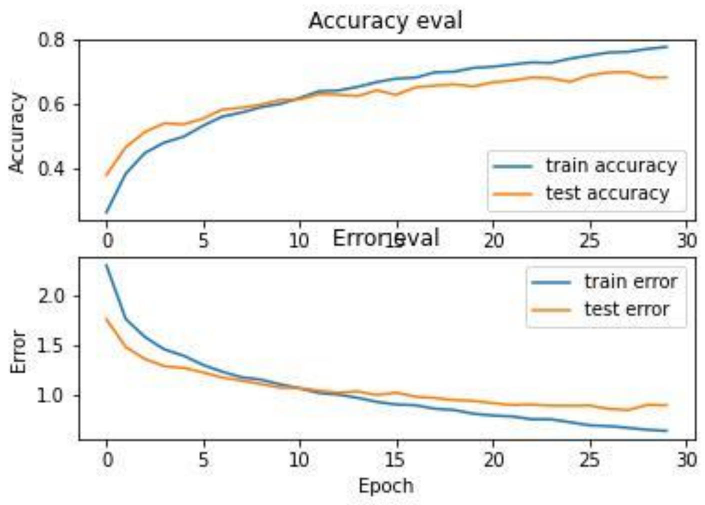
Tackling Overfitting:

There is a possibility that the neural network will overfit the testing data and as a result, give poorer testing error. To solve this problem, we have used tensorflow.keras.layers.Dropout . This essentially ensures that certain neurands at the layer at which it is applied will be dropped off with a certain probability(specified in the function). It has been proved that with Neural networks with many intermediate layers, Dropout of neurands will make the network more robust.

Implementation of the CNN and results:

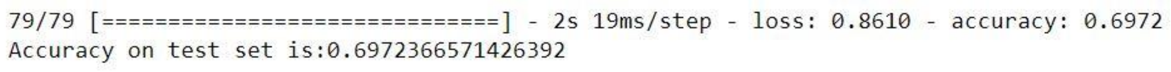
As specified in the paper, we have implemented a 3 layers CNN (excluding the output layer). The features and target attributes are extracted and stored in numpy arrays.The channel of the spectral features is in greyscale-hence we use a 1 channel CNN. As mentioned earlier, the MFCC coefficients for 13 features has the input shape of 130x13x1 (where 130 represent the sampling rate/hop length), 13 denotes the number of features and 1 denotes the number channels(greyscale).The kernel size has been taken as 2X2 and after each convolutional block, the dimension of the matrix is reduced. Finally, the matrix will be flattened into straight vector and passed into a fully connected layer for classification. At the end of the output layer, use the softmax function which gives the output in the form of probabilities. For running the model, we will use the Adam optimizer and the loss function used was sparse_categorical_crossentropy. The **model was run with a batch size of 32 and number of epochs totalling to 30.**

<u>Results:</u>



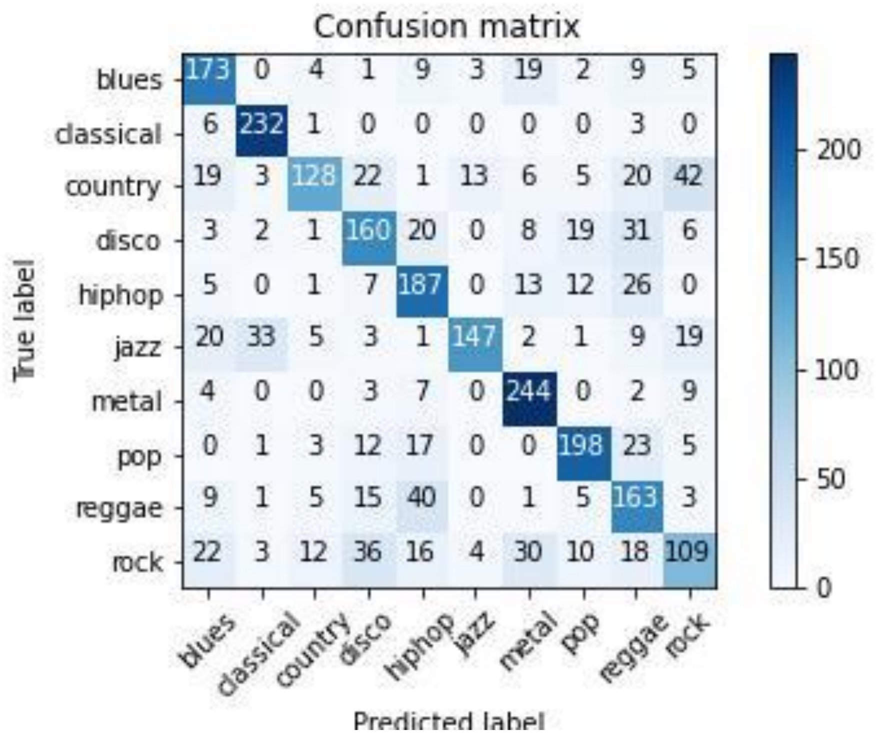
We see an accuracy of **69.67% was achieved on the testing data**. Overfitting hasn't taken place and the difference in error between testing and training data didn't become too large. However, a lower accuracy was achieved for essentially 2 reasons:

1) The number of training data points fed into the model isn't sufficient. However, due to constraints in computational limits, we cannot use more data points (an store them in a json file).
2) The number of features extracted from the dataset is 13 (only MFCC features). It doesn't seem to have captured the entire essence of the data but again due to computing complexity, it is not possible to use more features.
3) **CNNs have the fatal flaws of translational invariance** which could be offset by using data augmentation(adding more points). However,due to aforementioned constraints, to improve efficiency we will use Recurrent Neural Networks.



```
79/79 [==============================] - 2s 19ms/step - loss: 0.8610 - accuracy: 0.6972
Accuracy on test set is:0.6972366571426392
```

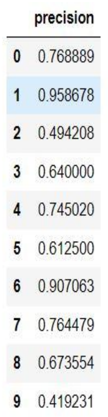
Fig. denotes the overall accuracy



True label \ Predicted label	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	173	0	4	1	9	3	19	2	9	5
classical	6	232	1	0	0	0	0	0	3	0
country	19	3	128	22	1	13	6	5	20	42
disco	3	2	1	160	20	0	8	19	31	6
hiphop	5	0	1	7	187	0	13	12	26	0
jazz	20	33	5	3	1	147	2	1	9	19
metal	4	0	0	3	7	0	244	0	2	9
pop	0	1	3	12	17	0	0	198	23	5
reggae	9	1	5	15	40	0	1	5	163	3
rock	22	3	12	36	16	4	30	10	18	109

The above figure represents the Confusion Matrix

The above image denotes the confusion matrix over the testing data.
The confusion matrix was implemented with the sklearn libraries. We see that the accuracy is highest for metal and classical music**. Rock music seems to have been the most erroneously classified data.**



	precision
0	0.768889
1	0.958678
2	0.494208
3	0.640000
4	0.745020
5	0.612500
6	0.907063
7	0.764479
8	0.673554
9	0.419231

<u>**Improvement - RNN -LSTM Neural Networks**</u>

Recurrent Neural Networks are a type of neural network where the outputs from the previous step are fed as input to the current step. RNN have a **"memory"** which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

Why is RNN an improvement over CNNs?

Since the order of the data (in audio spectrogram for example) is very important, it will make sense to use a neural network that will make predictions on a current input vector based on information learnt in previous iteration. Mel spectrograms have melodies that would only make sense to learn on if they are to be arranged in a particular order(based on time steps). **We want the network to be able to process data sequentially regardless of the number of data points-which is something that RNNs can do while CNNs cant.** This means that the CNN model we developed is not able to predict as well since time series data such as audio has some information based on past input that we ignore completely and don't learn anything from. This will negatively impact the efficiency of the model.

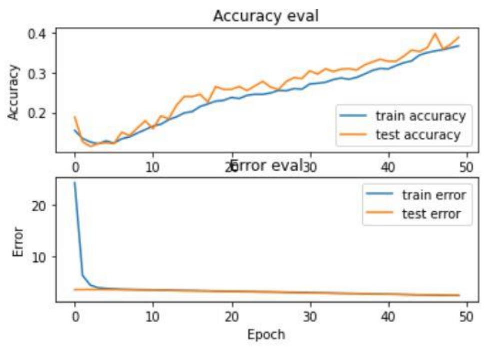
However, RNNs don't have the ability to keep track of long term memory - networks can't use long term memory from the distant past. Also,**training RNNs are difficult as they have the effects of vanishing and exploding gradients** with more layers. As a result, RNNs can't learn patterns with long dependencies(which is always seen in audio data). In order to tackle this problem, various types of RNN networks have been developed - with the most successful model to detect long dependencies being **RNN-Long Short Term Memory Networks(LSTM).**

LSTMs can learn long patterns and therefore improve the accuracy of models. The improved learning of the LSTM allows the user to train models using sequences with several hundreds of time steps, something the RNN struggles to do.

Implementation of the RNNs and Results:

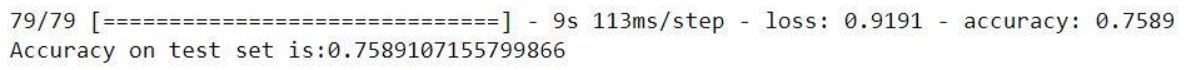
The RNN-LSTM was implemented with the help of tensorflow.keras in python (just like in CNN) using the command tensorflow.keras.layers.LSTM . In the implementation, we have used two RNN-LSTM layers, one dense layer with activation function as 'relu' to normalize predictions from previous layers and account for overfitting and a Dense softmax output layer for the classification. The rest of the implementation is the same as CNN- we used the Adam optimizer to compile the model and the loss function used was sparse categorical cross entropy.

<u>Results</u>:

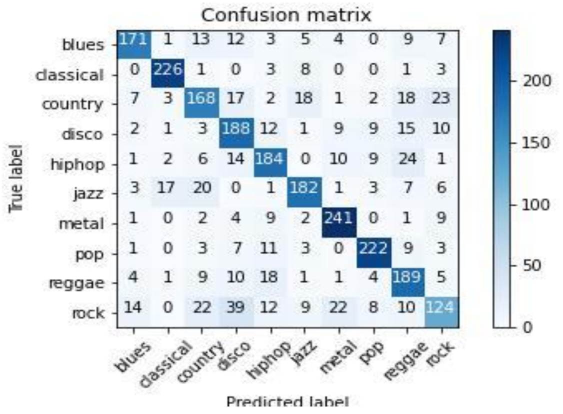


We see an accuracy of **75.89% was achieved on the testing data.** Overfitting hasn't taken place and the difference in error between testing and training data didn't become too large. We are able to see that the RNN-LSTM has obtained a higher accuracy than the CNN model as it has taken the past data also into account and learnt from it. Hence, the intuition for the improvement model has been satisfied.

To obtain a higher accuracy, data augmentation can be done -but local systems have computational constraints hence we did not consider it.



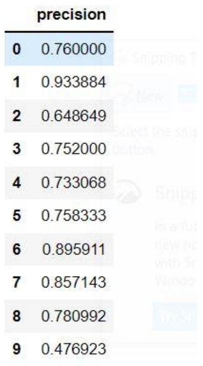
```
79/79 [==============================] - 9s 113ms/step - loss: 0.9191 - accuracy: 0.7589
Accuracy on test set is:0.7589107155799866
```



Confusion matrix

True label \ Predicted label	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	171	1	13	12	3	5	4	0	9	7
classical	0	226	1	0	3	8	0	0	1	3
country	7	3	168	17	2	18	1	2	18	23
disco	2	1	3	188	12	1	9	9	15	10
hiphop	1	2	6	14	184	0	10	9	24	1
jazz	3	17	20	0	1	182	1	3	7	6
metal	1	0	2	4	9	2	241	0	1	9
pop	1	0	3	7	11	3	0	222	9	3
reggae	4	1	9	10	18	1	1	4	189	5
rock	14	0	22	39	12	9	22	8	10	124

The above image denotes the confusion matrix over the testing data.
The confusion matrix was implemented with the sklearn libraries. **RNN model has been able to get a better performance overall in getting true positives compared to the CNN model**, We see that the accuracy is highest for metal and classical music. Rock music seems to have been the most erroneously classified data but it has a higher percentage of true positives here than it had in CNNs.



	precision
0	0.760000
1	0.933884
2	0.648649
3	0.752000
4	0.733068
5	0.758333
6	0.895911
7	0.857143
8	0.780992
9	0.476923

HENCE, WE HAVE BEEN ABLE TO PROVE THAT USING A RECURRENT NEURAL NETWORK WITH LSTM GIVES USE BETTER RESULTS THAN THE CONVENTIONAL CNN NETWORK.

CONCLUSION:

We have been able to implement our algorithm that helps us classify audio samples(obtained from GTZAN dataset) based on the music genre (among 10 class labels). Using the concepts of Principle Component Analysis, we extracted the features of the audio spectrogram using MFCCs which we proved retains as much information of the dataset as possible-we cannot extract all features and train model due to computational constraints. We then implemented the Convolutional Neural Network as described in the paper, and we obtained an accuracy of 69.67% .Higher accuracy wasn't achieved due to computational constraints with a local system(did not extract all 128 features since we don't have resources to do so) and also since CNNs didn't exploit sequential data. We then explained why CNNs are not able to capture information of previous time steps as it has no memory components and why RNN-LSTM has memory elements which help make the model more accurate. We implemented the RNN model and were able to get an overall accuracy of 75.89%- RNN-LSTM was proven to outperform CNN. We finally showed the confusion matrix obtained from both models and showed how many of the training samples are falsely classified.