Classifying the Genre of a Song with Convolution Neural Networks

Ajay Kesarwani
Universität Passau
Passau, Bavaria, Germany
kesarw01@ads.uni-passau.de

Professor: Ph. D. Tomas Skopal
Universität Passau
Passau, Bavaria, Germany
skopal01@ads.uni-passau.de

ABSTRACT

The main aim of this project is to create a model for classifying the music genre of a song with a CNN convolutional neural network. In this project, first, we get the Mel Frequency Cepstral Coefficients (MFCCs) vector of 1000 songs from the GTZAN genre dataset i.e. unstructured audio multimedia sensory data are gets analyzed and tells about what types of genre of a song it's belonging to. The model was trained based on basically from the 10 common music genres. The prediction was done based on the majority. We found that using the MFCCs feature of audio can produce a better result on our trained model with an accuracy of more than 70%. We also predict the genre of a single song uploaded from the recorded in mobile phone and uploaded in google drive.

INTRODUCTION

A music genre tells about the types of the tradition or style of music. We used to do manually the genre classification of music but based on this project we can do it automatically using the trained model. The model uses the MFCCs feature of the audio and train the model and do the prediction using the machine learning techniques.

RELATED WORK

Genre classification has been done using machine learning techniques. In 2002, G. Tzanetakis and P. Cook [1] both created the GTZAN dataset. Although they have earlier used the Gaussians model and k-nearest neighbour's approach later it was found out that using the CNN can provide a much better result with an accuracy of more than 90%. Therefore, the implementation of this project is done using the CNN model.

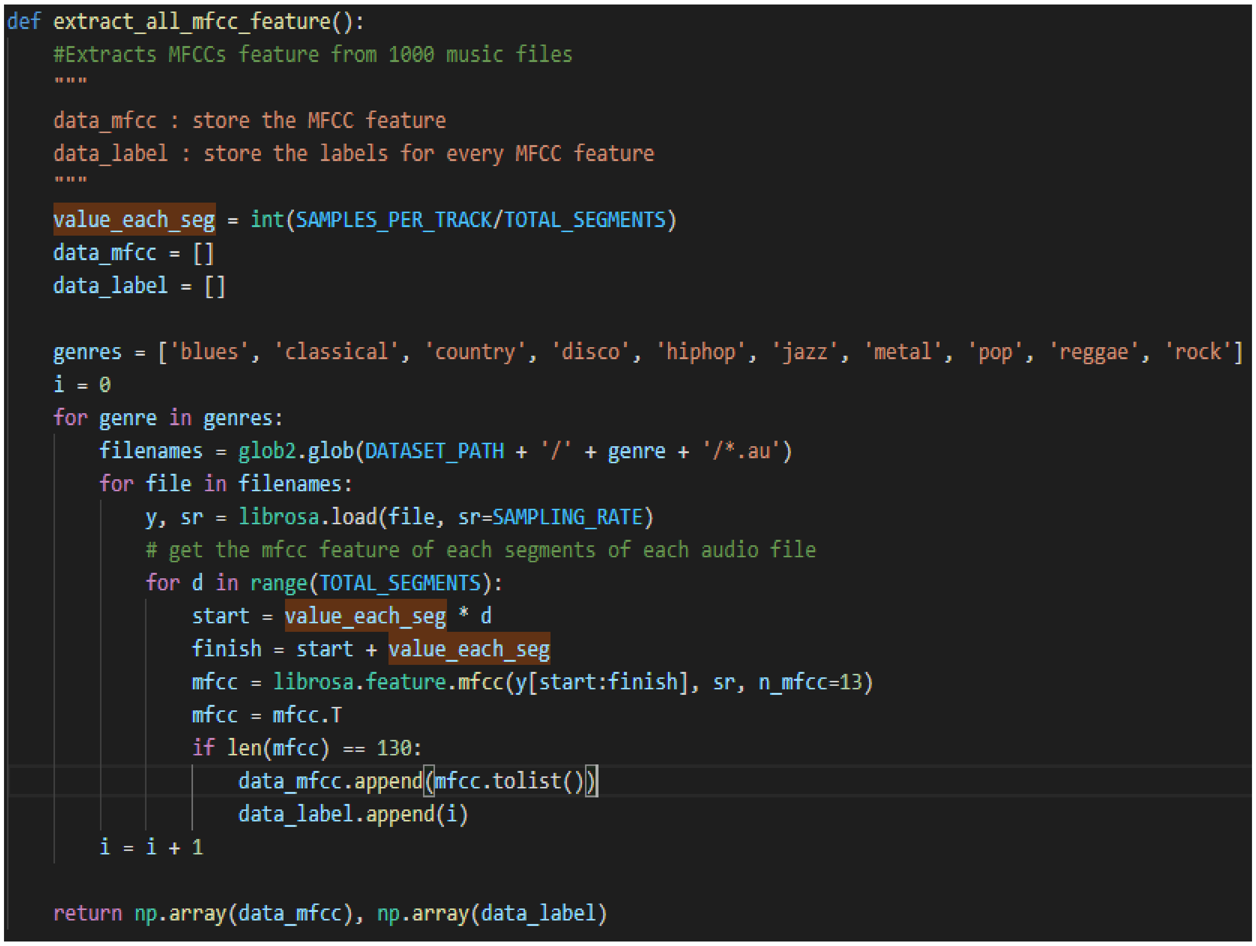
DATASET

We get the GTZAN dataset from the MARYSAS website [2]. The dataset has 1000 different songs which include 10 different genres each having 30-second clips, 22050Hz sampling rate and in .au format. The genres are blues, classical, country, disco, hippo, jazz, metal, pop, reggae, rock. We took 70% for training, 10% for validating and 20% for testing.

PREPROCESSING

We get the MFCCs from the raw audio, which is used for speech recognition. We took only 13 (Discrete Cosine Transform) DCT coefficients for evaluating the performance of the feature vector because the lower order coefficients contain most of the information about the overall spectral shape of the source-filter transfer function. We used the 'Librosa library' [7] with a sampling rate of 22050Hz. To feed the model with a large number of data we subdivided each audio file into 10 segments. Therefore, for each file we have 10 different segments, basically, after splitting into 10 segments we are now having the 1000 data for each genre which means that we have 10,000 data to train our model. While looping through each file, the MFCC value of each segment is stored in the array according to the value of each segment which is defined based on the samples per track over the total segments (=10) where sample per track is a multiplication of sampling rate (=22050) and duration (= the 30s) of an audio file. We also store the labels of each segment respective of their genre where 0 is for blues and 1 for classical and so on which means 9 for the last genre i.e. rock.

Below is the code snippet attached for reference.



```python
def extract_all_mfcc_feature():
    #Extracts MFCCs feature from 1000 music files
    """
    data_mfcc : store the MFCC feature
    data_label : store the labels for every MFCC feature
    """
    value_each_seg = int(SAMPLES_PER_TRACK/TOTAL_SEGMENTS)
    data_mfcc = []
    data_label = []

    genres = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock']
    i = 0
    for genre in genres:
        filenames = glob2.glob(DATASET_PATH + '/' + genre + '/*.au')
        for file in filenames:
            y, sr = librosa.load(file, sr=SAMPLING_RATE)
            # get the mfcc feature of each segments of each audio file
            for d in range(TOTAL_SEGMENTS):
                start = value_each_seg * d
                finish = start + value_each_seg
                mfcc = librosa.feature.mfcc(y[start:finish], sr, n_mfcc=13)
                mfcc = mfcc.T
                if len(mfcc) == 130:
                    data_mfcc.append(mfcc.tolist())
                    data_label.append(i)
        i = i + 1

    return np.array(data_mfcc), np.array(data_label)
```

Figure 1

Mel Frequency Cepstral Coefficients (MFCCs)

Figures 2.a to 2.j are MFCCs graphs of each genre i.e. blues, classical, country, disco, hippo, jazz, metal, pop, reggae, rock. The graph from the bottom to the top has 13 main sectors which tell DCT coefficients as we can see from the graph that most of the energy is at the bottom of the graph.

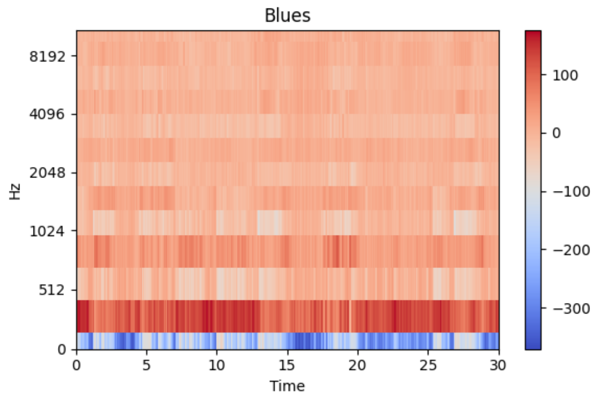


Figure 2.a

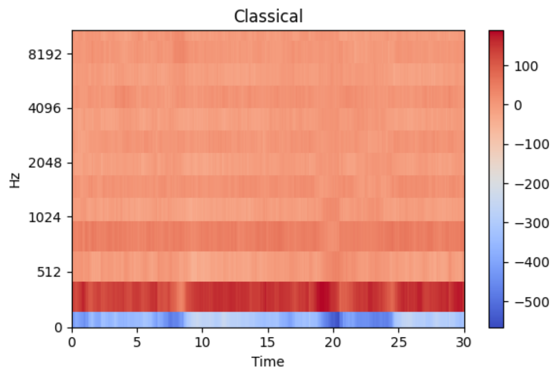


Figure 2.b

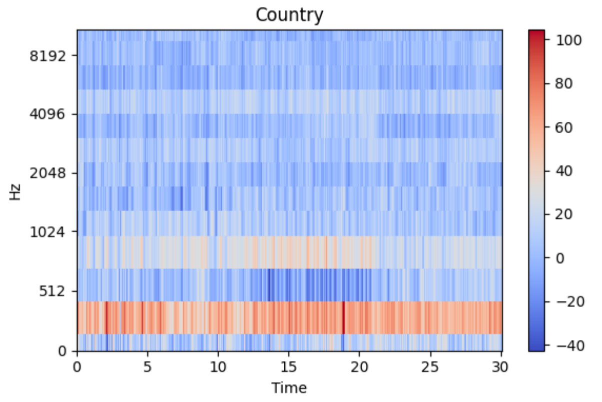


Figure 2.c

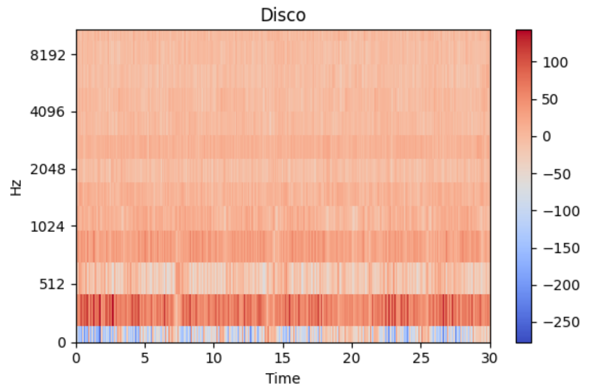


Figure 2.d

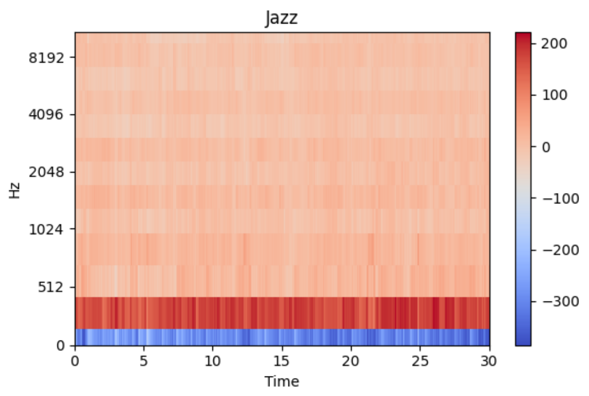


Figure 2.e

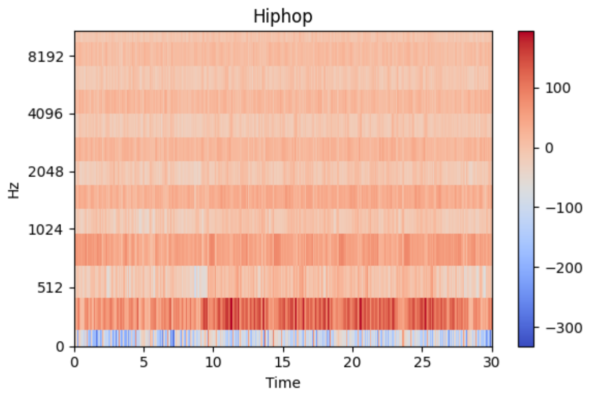


Figure 2.f

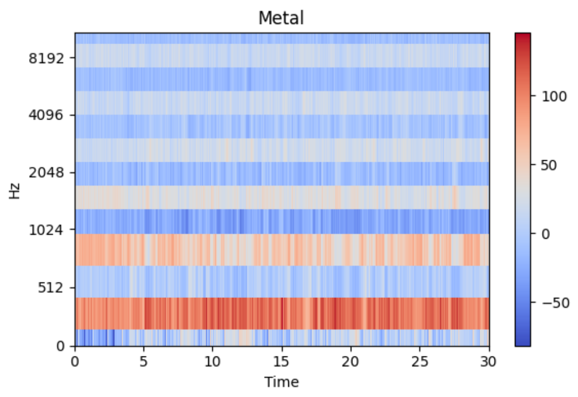


Figure 2.g

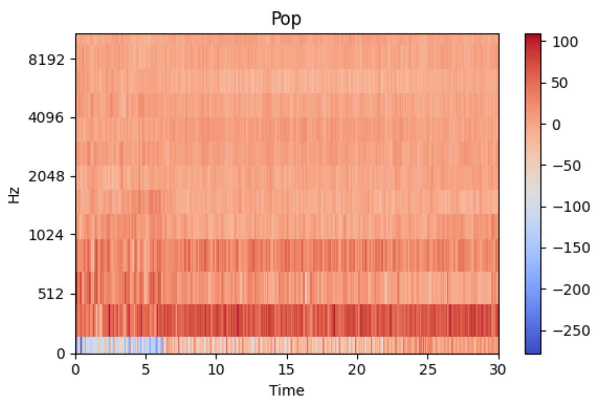


Figure 2.h

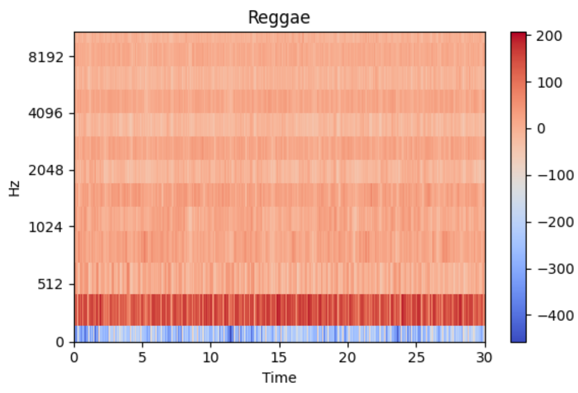


Figure 2.i

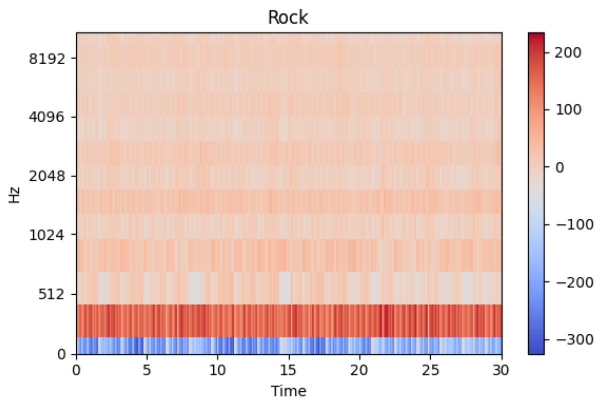


Figure 2.j

CLASSIFICATION

Model Creation

After obtaining the feature vectors, we create our model using the CNN convention. We implement the model using TensorFlow [5] and Keras [6]. We used the 3 convolution layers with 'relu' activation, 'softmax' output, and categorical cross-entropy loss. The 2d convolution layer is used with 32 neurons with kernel size 3 x 3 or 2 x 2 which read the input data and output the sum of the elements within the window, then the max-pooling operation is used with kernel 3 x 3 and strides of 2 x 2 with the same padding to selects the maximum element from another window. Afterwards, we used the Batch Normalization to converge a faster and more reliable model. To increase the robustness and avoid overfitting we randomly drop 25% of neurons while training.

To overcome the sparse stochastic gradient descent problems while using the audio time-series in 2 dimensions we used the Adam optimization. We keep the learning rate very low (=0.0001) so that model gives the optimal output which increases the accuracy, although by doing so model learns very slowly. We used the standard batch size of 32 also keep the epoch value to 30

After training the model we can see the difference in the accuracy of unseen data is not much different in the training model, which is almost less than between 5 to 10 percentage.

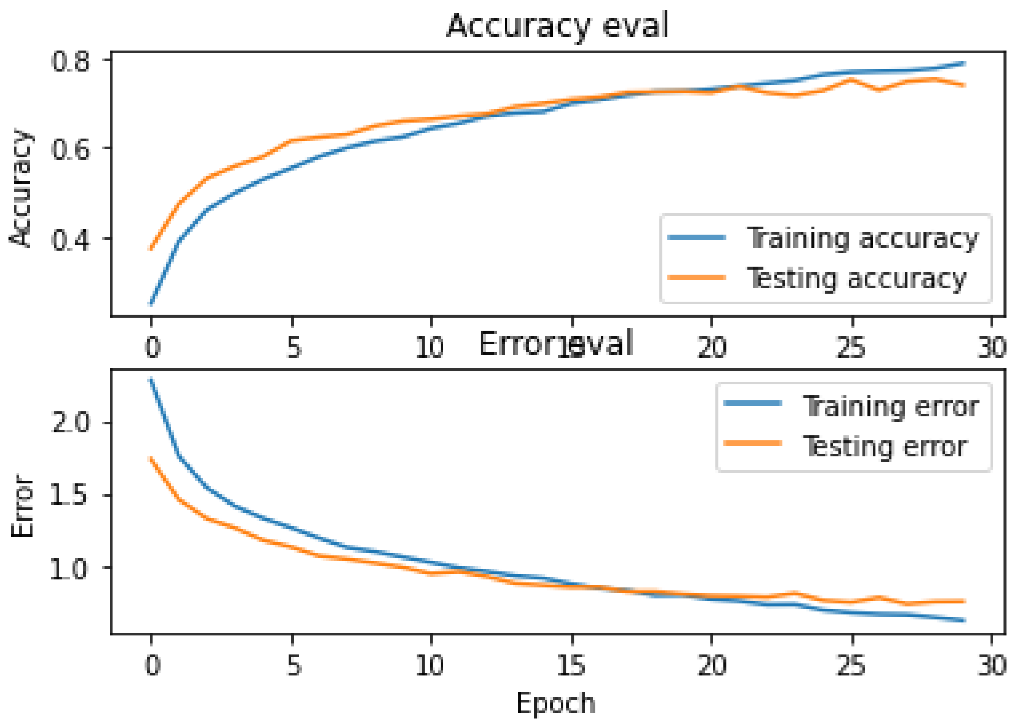


Figure 3

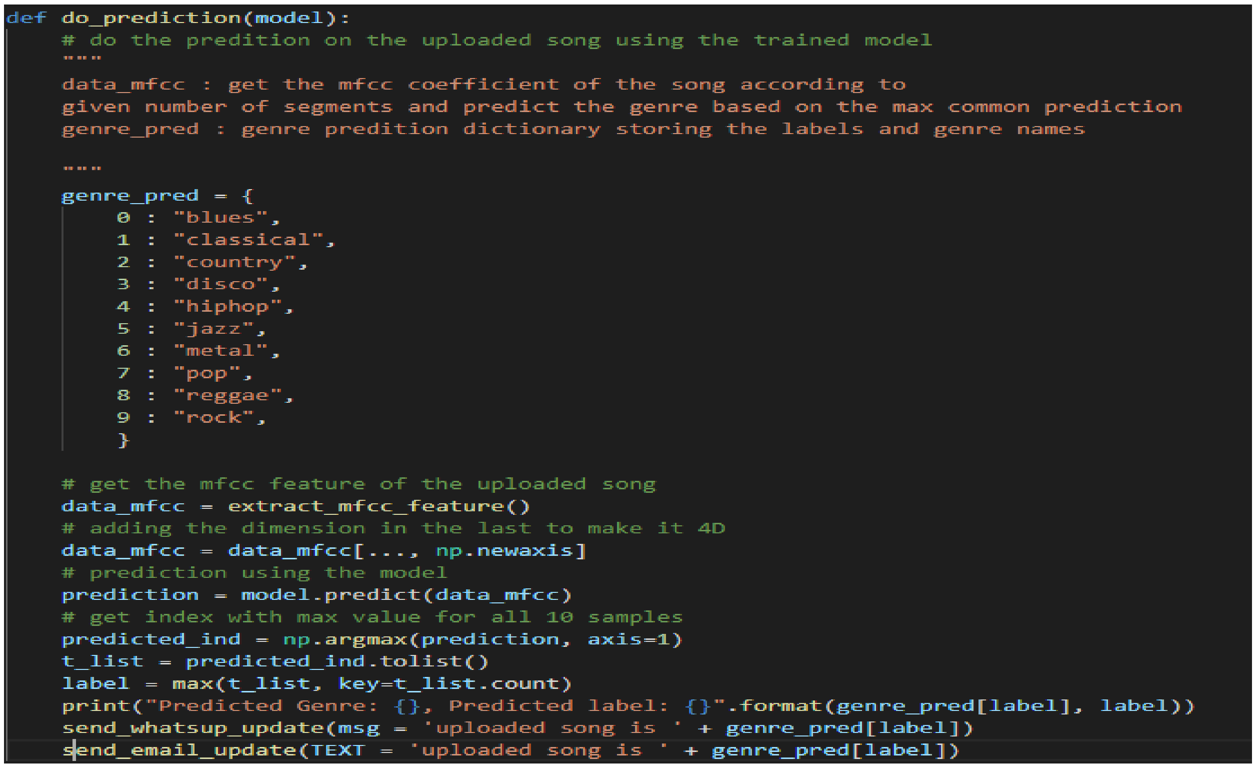
The first half of the plot from Figure 3, tells about the Accuracy for training data and testing data whereas the bottom second part tells about the error rate or loss of the graph. Here, the model is learning slowly with the training data the accuracy keeps increasing whereas for the testing data it was similar to training but after 20 epoch it shows a little decrease. Similar behaviour is seen in the error rate or loss function as well, for the training it keeps on decreasing while for testing it shows little increase after 20 epoch. The graph does not show any overfitting because it is smooth in almost most of the places.

Single Song Testing

We have also used this model to get the genre of a particular song recorded from the radio station or some anonymous places in the smart mobile. As part of this project, an interface is also developed where users can upload the song in their google drive in some folder and then give the path in the model so that it can predict the genre. To do so, we again read the song and convert it to the audio file i.e. .wav format so that the librosa library can read the MFCCs feature from it. The sampling rate 22050Hz is used. In this case, since we are taking 10 segments of each song. We may have that model predict the 10 different genres but since our model is more than 70% accurate so we may have to pick only 2 to 3 different genres from 10. We select those values where most of the segments predict the max common genre.

We also implemented the email and whatsup [4] update so that the user could know the genre in their mobile directly.

Snippet of the code is attached below:



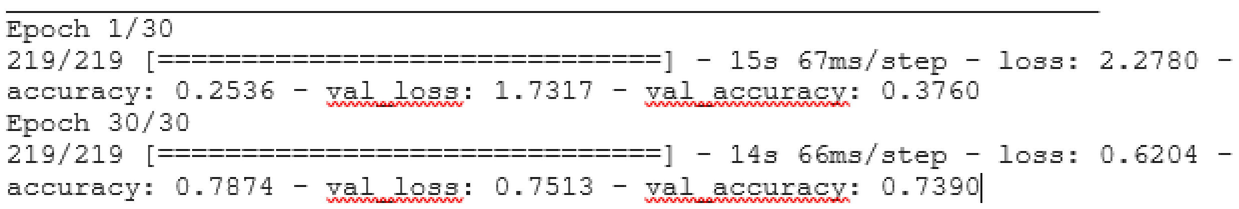
```
def do_prediction(model):
    # do the predition on the uploaded song using the trained model
    """
    data_mfcc : get the mfcc coefficient of the song according to
    given number of segments and predict the genre based on the max common prediction
    genre_pred : genre predition dictionary storing the labels and genre names

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

    # get the mfcc feature of the uploaded song
    data_mfcc = extract_mfcc_feature()
    # adding the dimension in the last to make it 4D
    data_mfcc = data_mfcc[..., np.newaxis]
    # prediction using the model
    prediction = model.predict(data_mfcc)
    # get index with max value for all 10 samples
    predicted_ind = np.argmax(prediction, axis=1)
    t_list = predicted_ind.tolist()
    label = max(t_list, key=t_list.count)
    print("Predicted Genre: {}, Predicted label: {}".format(genre_pred[label], label))
    send_whatsup_update(msg = 'uploaded song is ' + genre_pred[label])
    send_email_update(TEXT = 'uploaded song is ' + genre_pred[label])
```

Figure 4

To run the model we have used the google colab [3] which give GPU access so we can train our model a little faster than a normal system.

DISCUSSION

When the model starts getting trained we have seen that the accuracy of the training data and test data keep increasing starting from the first epoch till the last epoch. The accuracy of the training data increased from 25-30% to 75-95% and on the testing data, it increased from 22-25% till 70-90%. Below is the snippet from the processed model. The accuracy difference in the training data and test data is below 5% which says that our model is not overfitted. The loss is the error while predicting the correct genre of song. Which keeps on decreasing from 2.27 to 0.78 in the training data and 1.73 to 0.73 in the testing data.



```
Epoch 1/30
219/219 [==============================] - 15s 67ms/step - loss: 2.2780 -
accuracy: 0.2536 - val_loss: 1.7317 - val_accuracy: 0.3760
Epoch 30/30
219/219 [==============================] - 14s 66ms/step - loss: 0.6204 -
accuracy: 0.7874 - val_loss: 0.7513 - val_accuracy: 0.7390
```

CONCLUSION & FUTURE WORK

Training the model with the MFCCs feature vector of the song give better accuracy similar to a normal human being. The CNN model produces an accuracy between 70-90% which is way better than any other model, although, computation time to train the model is a little higher in the normal system.

In the future, we may increase the accuracy while predicting the unseen audio file. This can be done by training the model with a large amount of data. As we have used only 2 hidden layers we may increase that too to get a more advanced model. We may also use the RNN technique which may give us better results. Finally, like training, the model in the local computer takes a huge time so we can use the GPU or some way to train the model quickly.

REFERENCES

[1] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. IEEE Transactions on Speech and Audio Processing, 10(5):293–302, 2002.

[2] http://marsyas.info/
[3] https://colab.research.google.com/
[4] https://web.whatsapp.com/
[5] https://www.tensorflow.org/
[6] https://keras.io/
[7] https://librosa.org/doc/latest/index.html