Music Artist Identification

Prof. Chandrashekhar Mukherjee, SJSU CMPE 257

Project By:

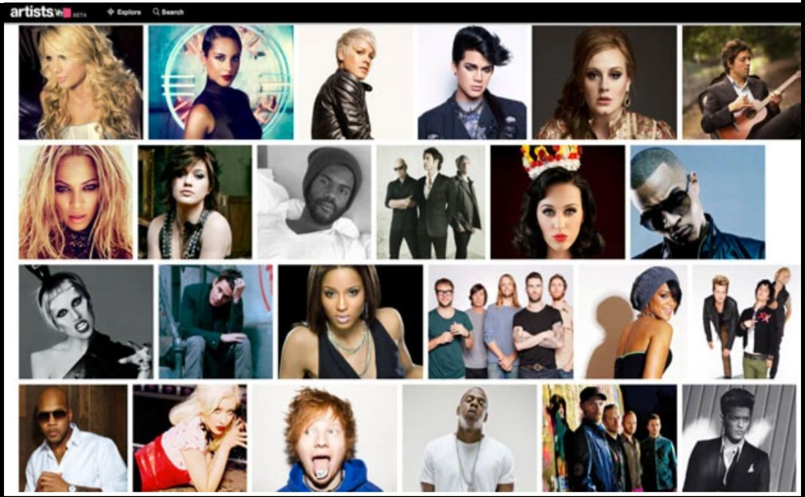
Bhaummi Patel (015270554)
Kai Wei Yang (013798967)
Krupaben Kothadia (013779090)
Vaishak Melarcode Kallampad (015017496)

Objective

In this project we are identifying music artists by uploading a musical audio on a graphical user interface that we have created as a part of this project, which will give us the artist's name for that particular musical audio.

Motivation

A research paper on "Music Artist Classification with Convolutional Recurrent Neural Networks" by Zain Nasrullah and Yue Zhao

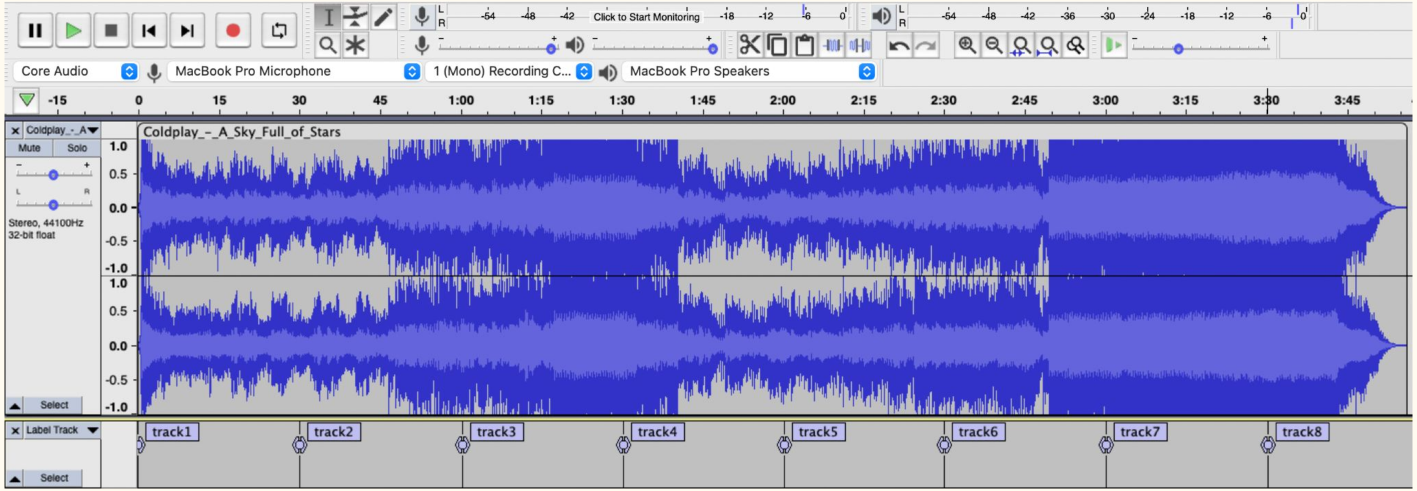


INTRODUCTION

- In this project we are using concepts like:
- Music information retrieval (MIR) which encompasses most audio analysis tasks such as genre classification, song identification, chord recognition, sound event detection, mood detection and feature extraction.
- Implementing deep learning models like ANN, RNN and CNN.
- Implementing user interface

Data Collection

- Manually downloaded songs of 5 artists from the Internet.
- Split a song into segments of 30 seconds each using AUDACITY



Dataset

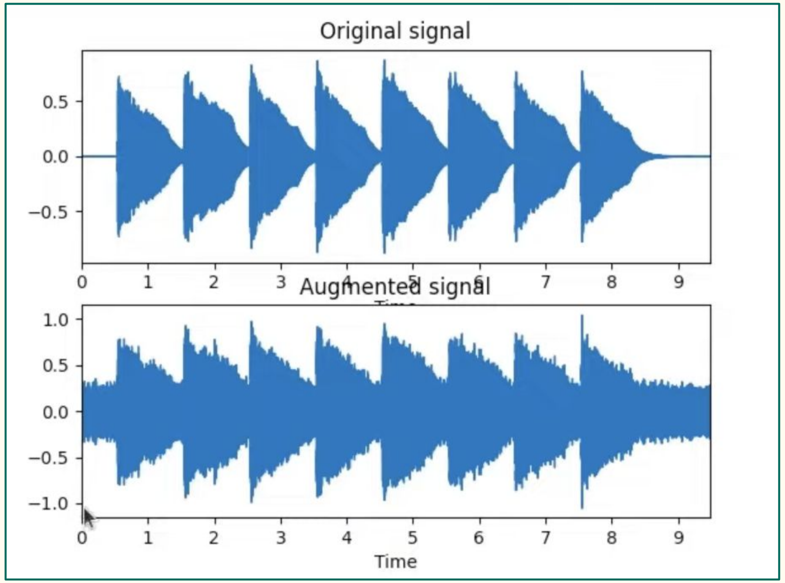
- There are 5 artists in our dataset
- All 5 artists have 100 sound clips each.

Artists	Training Data	Test Data
Beyonce	75	25
Bruno Mars	75	25
Greenday	75	25
Maroon5	75	25
Migos	75	25

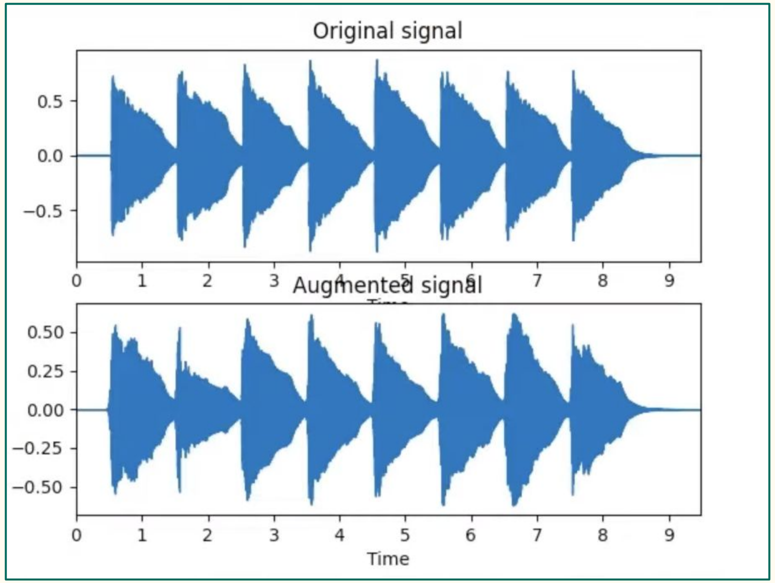
Data Augmentation

Techniques used:

- Adding white noise



- Scaling Pitch

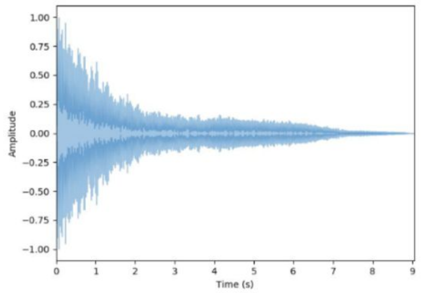


Final Dataset

- After augmentation each artist has 300 sound clips.

Artists	Training Data	Test Data
Beyonce	225	75
Bruno Mars	225	75
Greenday	225	75
Maroon5	225	75
Migos	225	75

Traditional ML pre-processing pipeline for audio data

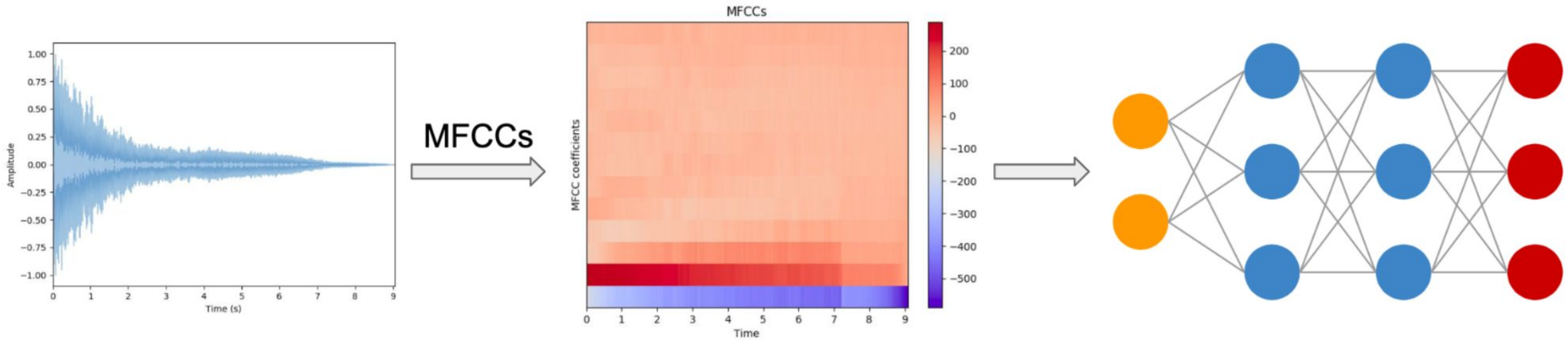


- Amplitude envelope
- Spectral centroid
- 0-crossing rate
- Spectral flux
- ...

ML algorithm

- Feature engineering
- Perform STFT
- Extract time + frequency domain features

Our pipeline for audio data

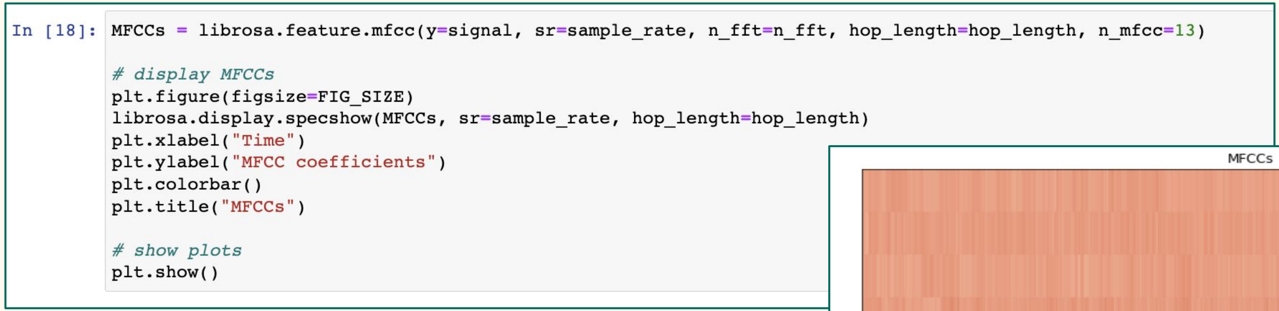


DATA PRE-PROCESSING

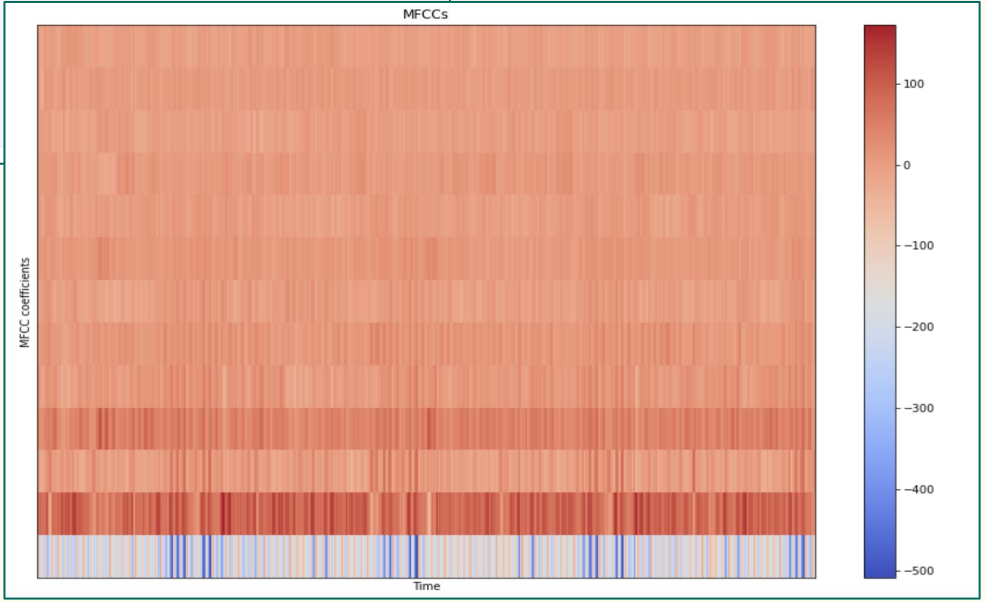
Mel Frequency Cepstral Coefficients (MFCCs)

- The mel frequency cepstral coefficients(MFCCs) of a signal are a small set of features (usually about 10-20) which concisely describe the overall shape of a spectral envelope. In MIR, it is often used to describe timbre.
- The capture the timbral/spectral aspect of sound
- Frequency Domain Feature
- Incorporates all features traditionally used for audio data
- Makes a good approximation of the human auditory system
- Usually 13 to 40 coefficients
- Needs to be calculated for each frame

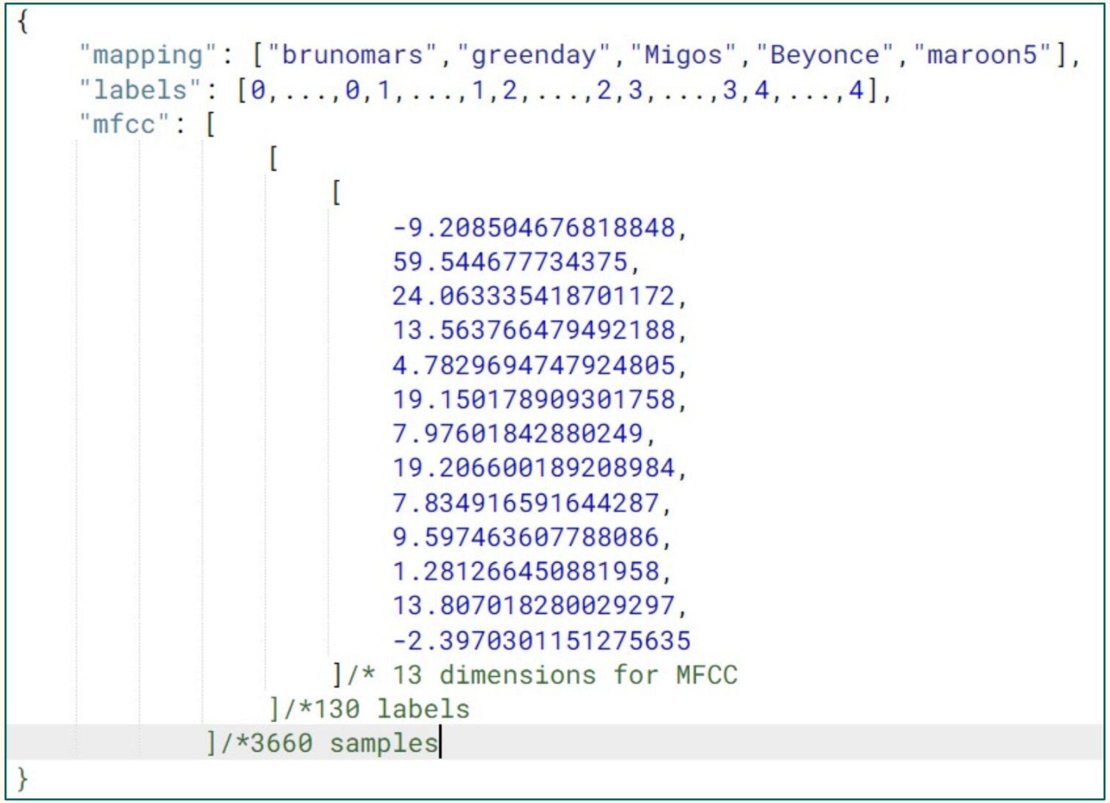
MFCC



```
In [18]: MFCCs = librosa.feature.mfcc(y=signal, sr=sample_rate, n_fft=n_fft, hop_length=hop_length, n_mfcc=13)

         # display MFCCs
         plt.figure(figsize=FIG_SIZE)
         librosa.display.specshow(MFCCs, sr=sample_rate, hop_length=hop_length)
         plt.xlabel("Time")
         plt.ylabel("MFCC coefficients")
         plt.colorbar()
         plt.title("MFCCs")

         # show plots
         plt.show()
```



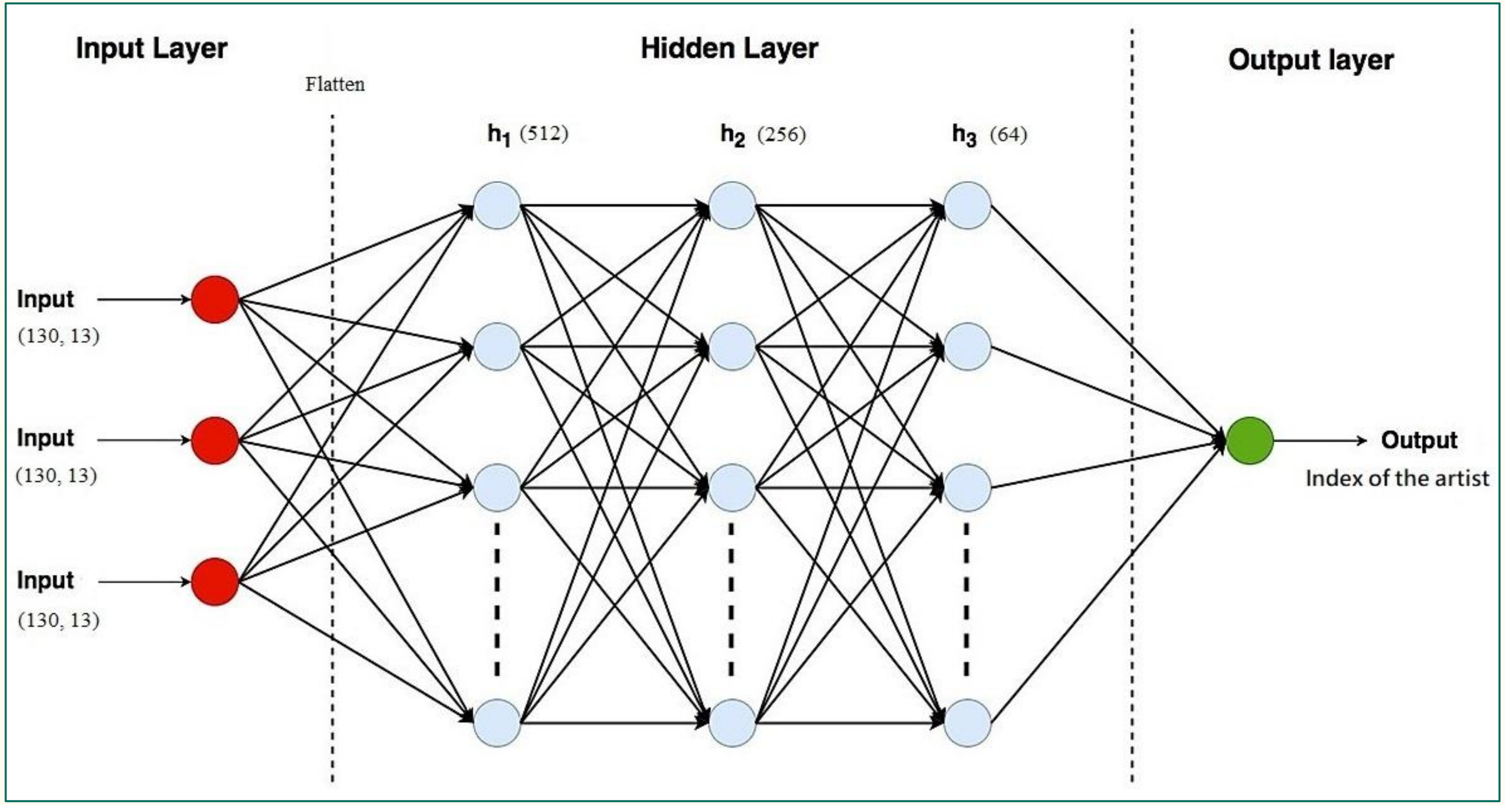
JSON



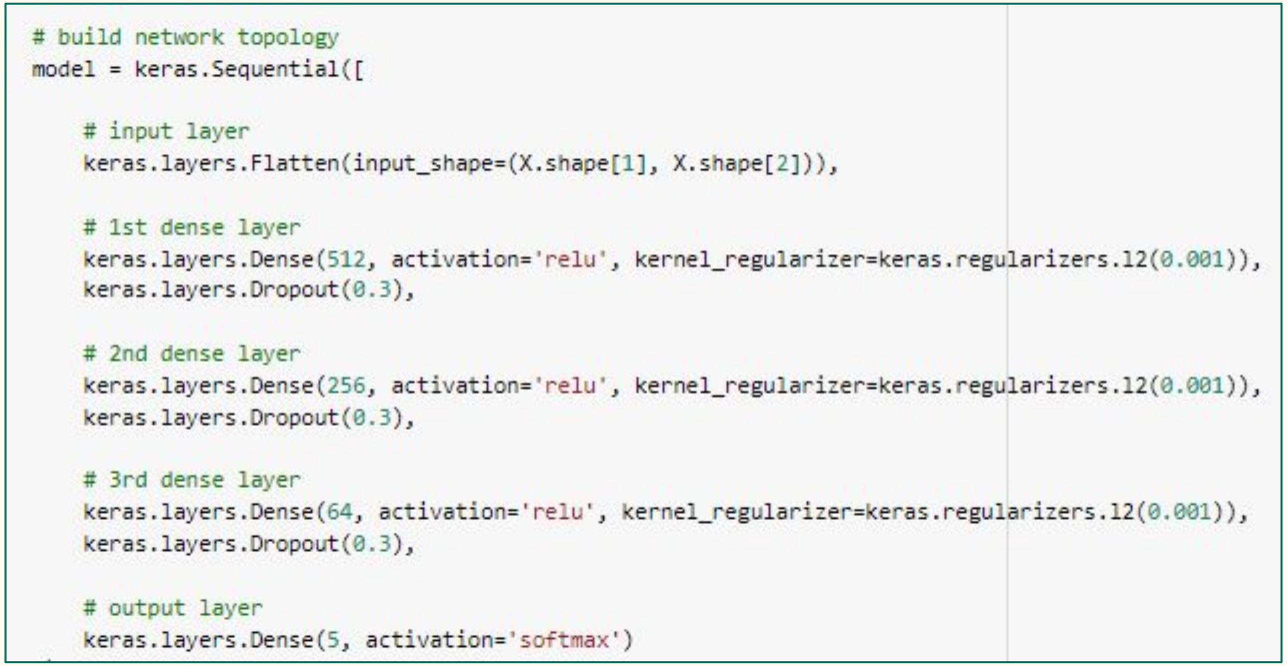
```
{
    "mapping": ["brunomars","greenday","Migos","Beyonce","maroon5"],
    "labels": [0,...,0,1,...,1,2,...,2,3,...,3,4,...,4],
    "mfcc": [
                [
                    [
                        -9.208504676818848,
                        59.544677734375,
                        24.063335418701172,
                        13.563766479492188,
                        4.782969474792480 5,
                        19.150178909301758,
                        7.97601842880249,
                        19.206600189208984,
                        7.834916591644287,
                        9.59746360778808 6,
                        1.281266450881958,
                        13.807018280029297,
                        -2.397030115127563 5
                    ]/* 13 dimensions for MFCC
                ]/*130 labels
            ]/*3660 samples
}
```

Model Selection

Artificial Neural Network



Artificial Neural Network

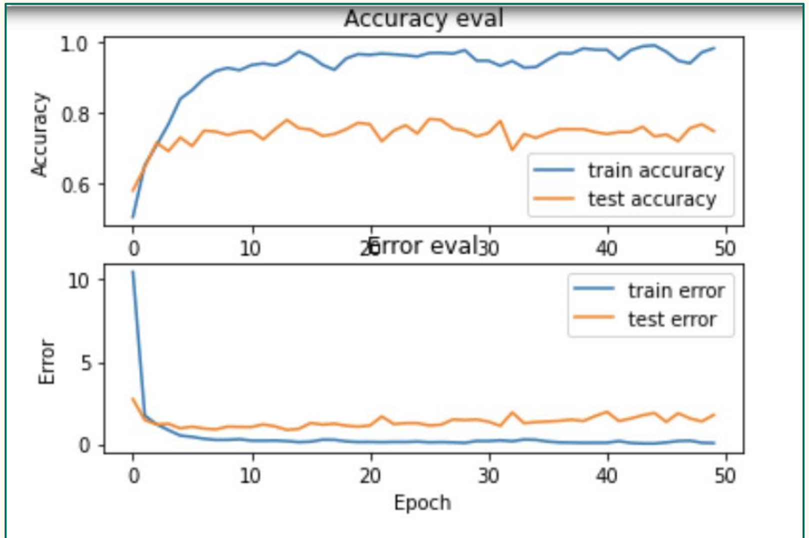


```
# build network topology
model = keras.Sequential([

    # input layer
    keras.layers.Flatten(input_shape=(X.shape[1], X.shape[2])),

    # 1st dense layer
    keras.layers.Dense(512, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),

    # 2nd dense layer
    keras.layers.Dense(256, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),

    # 3rd dense layer
    keras.layers.Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),

    # output layer
    keras.layers.Dense(5, activation='softmax')
```

1. Create ANN model
2. Create an input layer to flatten the input data
3. Create 1st layer with size 512
4. Create 2nd layer with size 256
5. Create 3rd layer with size 64
6. Add the regularizer to the layer to prevent overfitting
7. Input the data from 3rd layer to output layer and output the index prediction of the artist

ANN Model Accuracy

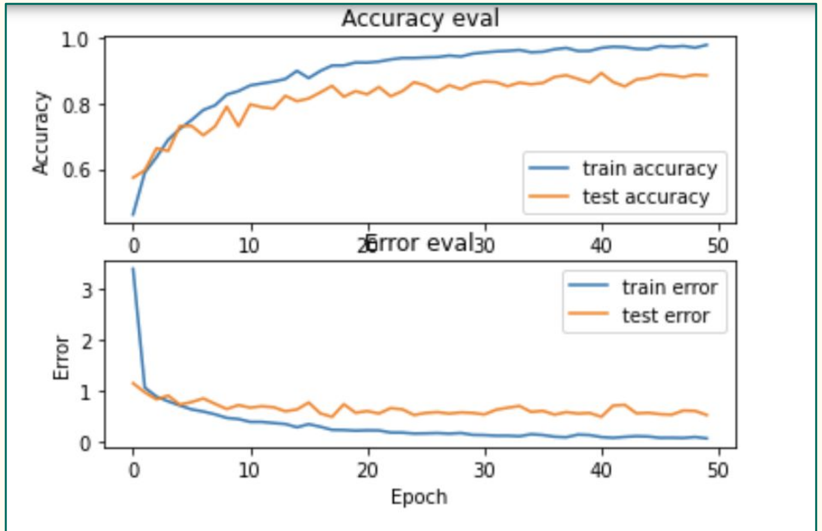
Without Augment Data



```
39/39 - 0s - loss: 1.6127 - accuracy: 0.7779

Test accuracy: 0.7778688669204712
```

Training Accuracy : 98.65%

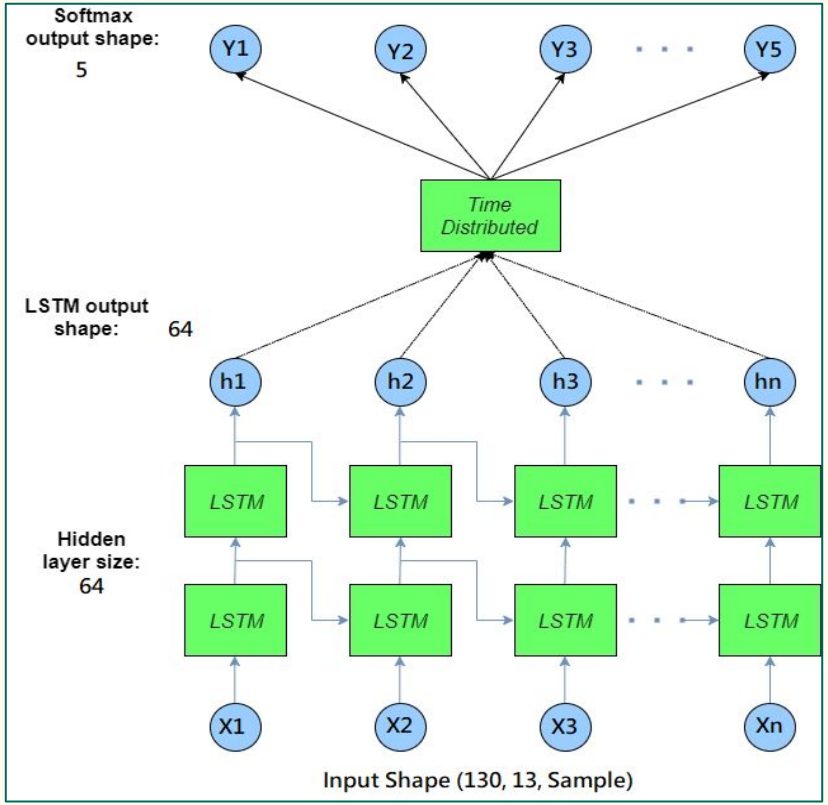
With Augment Data



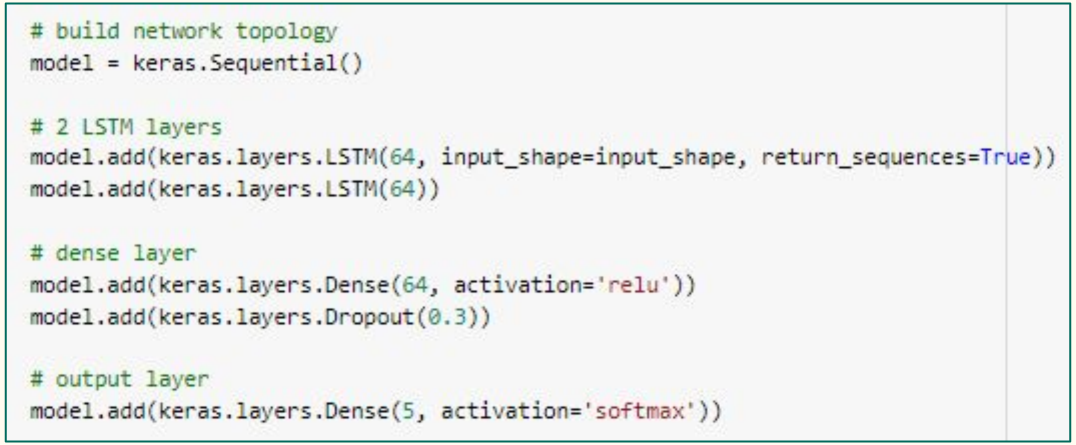
```
115/115 - 0s - loss: 0.5644 - accuracy: 0.8836

Test accuracy: 0.8836065530776978
```

Training Accuracy : 98.01%

Recurrent Neural Network



Recurrent Neural Network

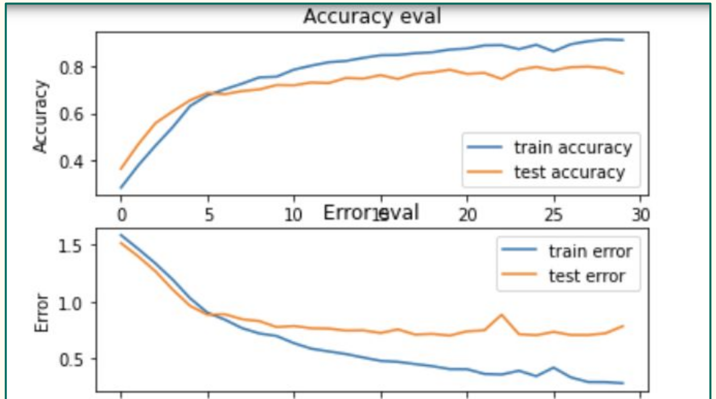


```
# build network topology
model = keras.Sequential()

# 2 LSTM layers
model.add(keras.layers.LSTM(64, input_shape=input_shape, return_sequences=True))
model.add(keras.layers.LSTM(64))

# dense layer
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.3))

# output layer
model.add(keras.layers.Dense(5, activation='softmax'))
```

1. Create RNN model
2. Create the LSTM and set the layer size to 64
3. The output of the LSTM layer would input to the dense layer
4. The output layer would generate the index prediction of the artist

RNN Model Accuracy

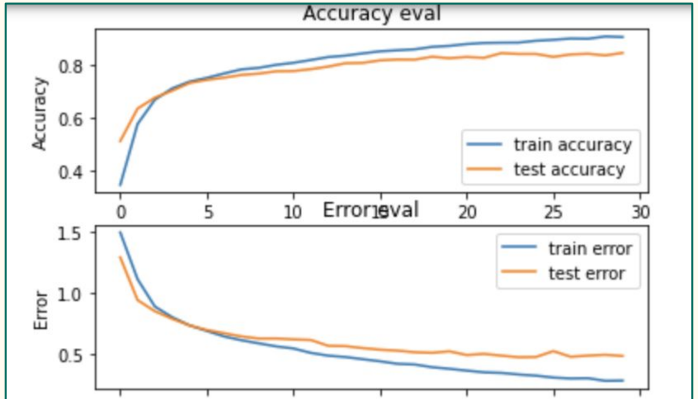
Without Augment Data



```
39/39 - 1s - loss: 0.7153 - accuracy: 0.7844

Test accuracy: 0.784426212310791
```

Training Accuracy : 91.51%

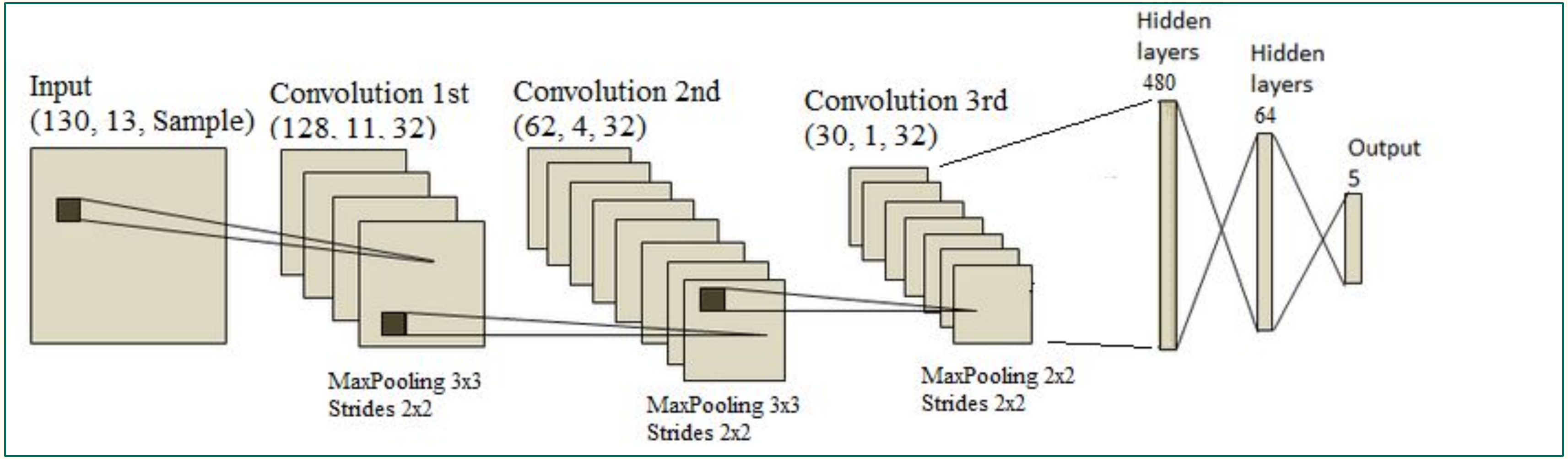
With Augment Data



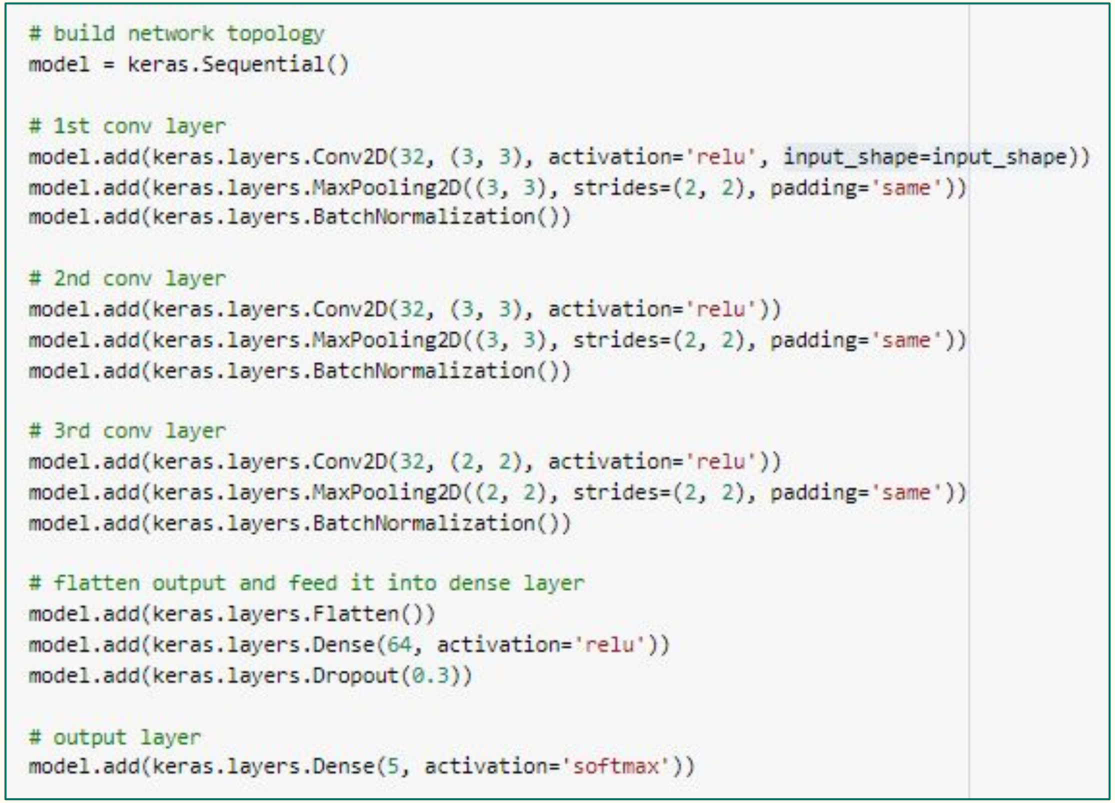
```
115/115 - 3s - loss: 0.4831 - accuracy: 0.8418

Test accuracy: 0.841803252696991
```

Training Accuracy : 90.69%

Convolutional Neural Network



Convolutional Neural Network

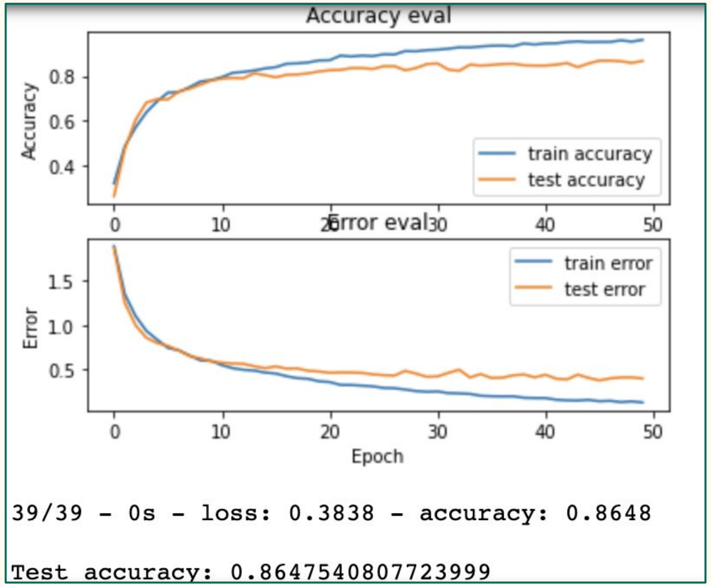


```
# build network topology
model = keras.Sequential()

# 1st conv layer
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# 2nd conv layer
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# 3rd conv layer
model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
model.add(keras.layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# flatten output and feed it into dense layer
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.3))

# output layer
model.add(keras.layers.Dense(5, activation='softmax'))
```

1. Define CNN model
2. Create 1st CNN model layer and set MaxPooling to 3x3 and strides to 2x2
3. Create 2nd CNN model layer and set MaxPooling to 3x3 and strides to 2x2
4. Create 3rd CNN model layer and set MaxPooling to 2x2 and strides to 2x2
5. The output would be flatten and input to the dense layer
6. The output layer would output the index prediction of the artist

CNN Model Accuracy

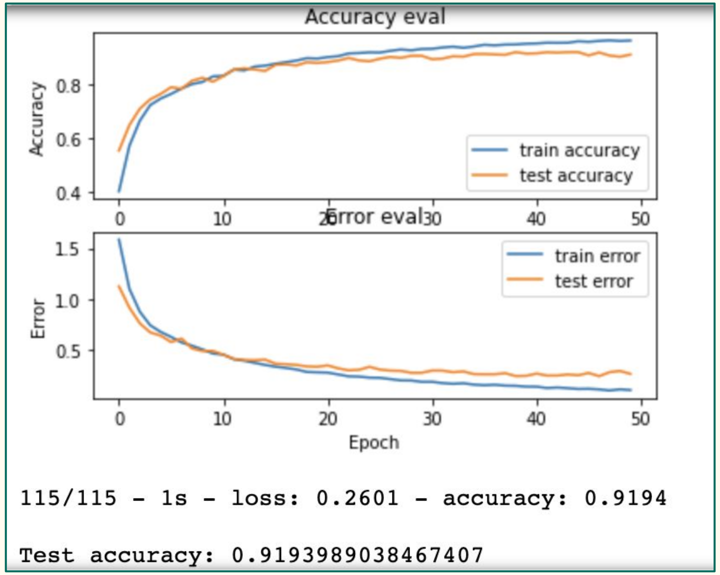
Without Augment Data



```
39/39 - 0s - loss: 0.3838 - accuracy: 0.8648

Test accuracy: 0.8647540807723999
```

Training Accuracy : 96.23%

With Augment Data

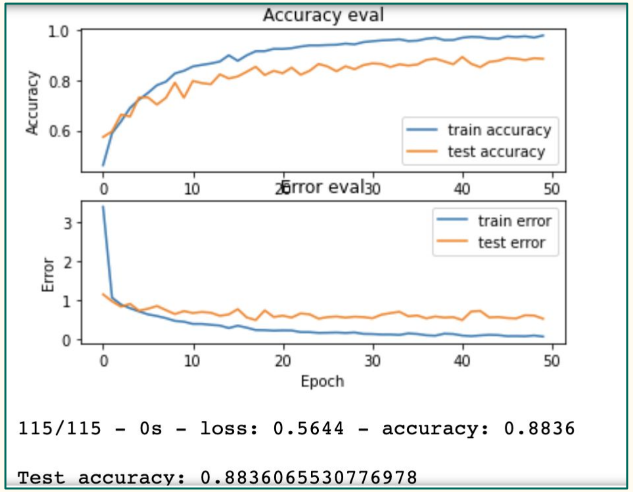


```
115/115 - 1s - loss: 0.2601 - accuracy: 0.9194

Test accuracy: 0.9193989038467407
```

Training Accuracy : 96.51%

Model Accuracy Comparison

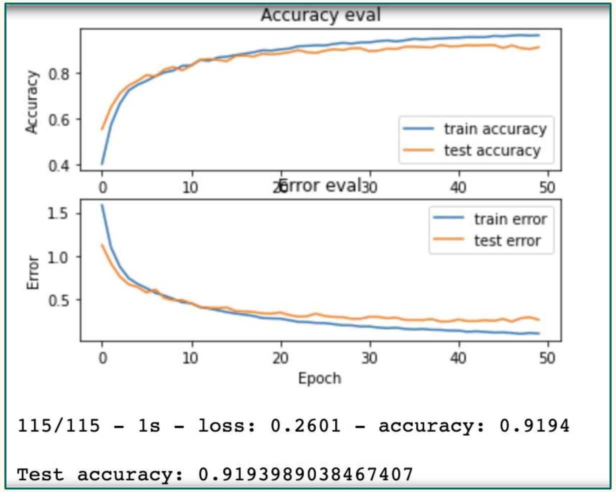
ANN



```
115/115 - 0s - loss: 0.5644 - accuracy: 0.8836

Test accuracy: 0.8836065530776978
```

Accuracy: 0.883

CNN

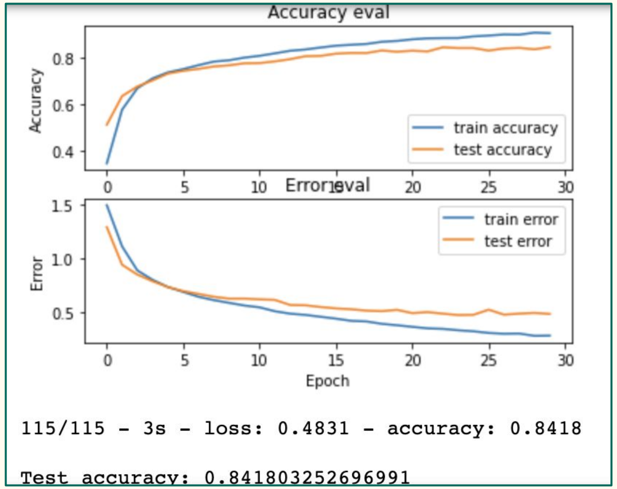


```
115/115 - 1s - loss: 0.2601 - accuracy: 0.9194

Test accuracy: 0.9193989038467407
```

Accuracy: 0.919

Highest Accuracy!

RNN



```
115/115 - 3s - loss: 0.4831 - accuracy: 0.8418

Test accuracy: 0.841803252696991
```

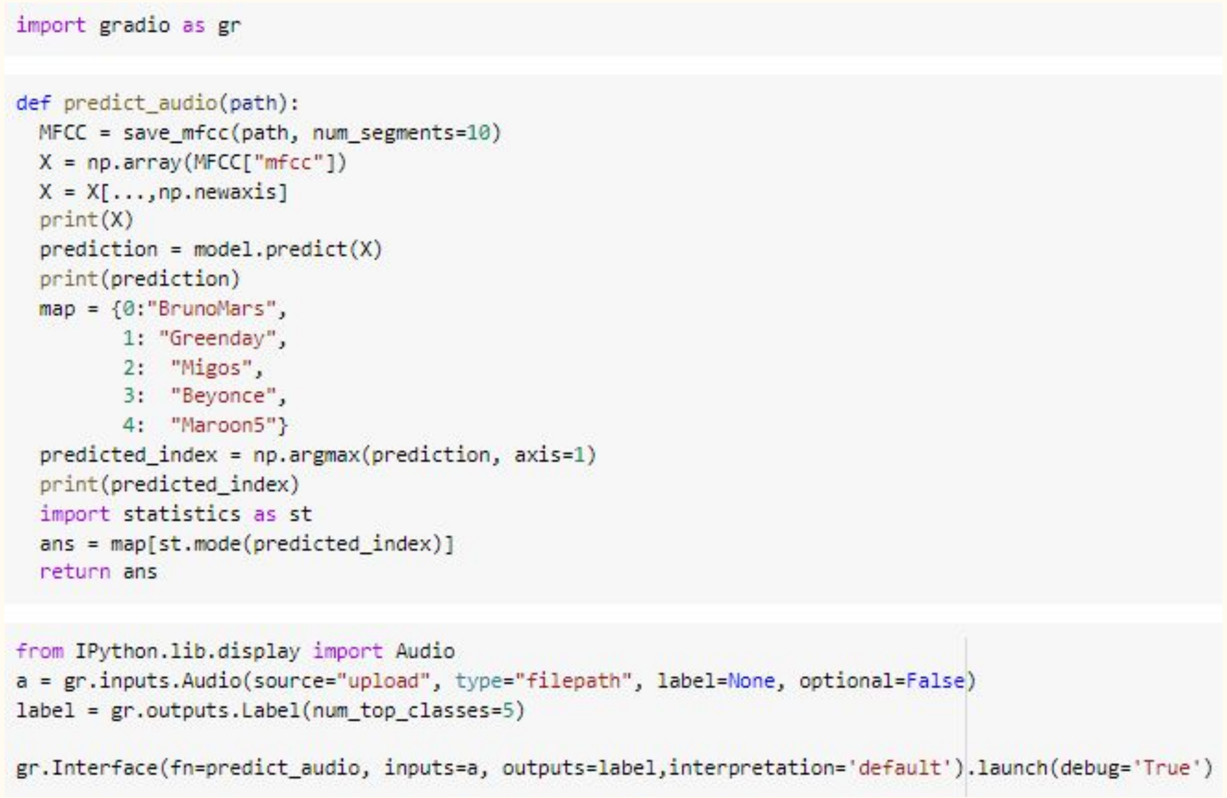
Accuracy: 0.842

Our Application

User Interface for Music Identification

- The gradio model would generate a URL of a web app for the user to achieve the music identification application
- We use gradio model to develop user interface for user to upload arbitrary length music file
- The uploaded music data would pass into the Neural Network model to make predictions
- The accurate predictions of the music data would output on the result box in the gradio model

User Interface (Gradio) Code



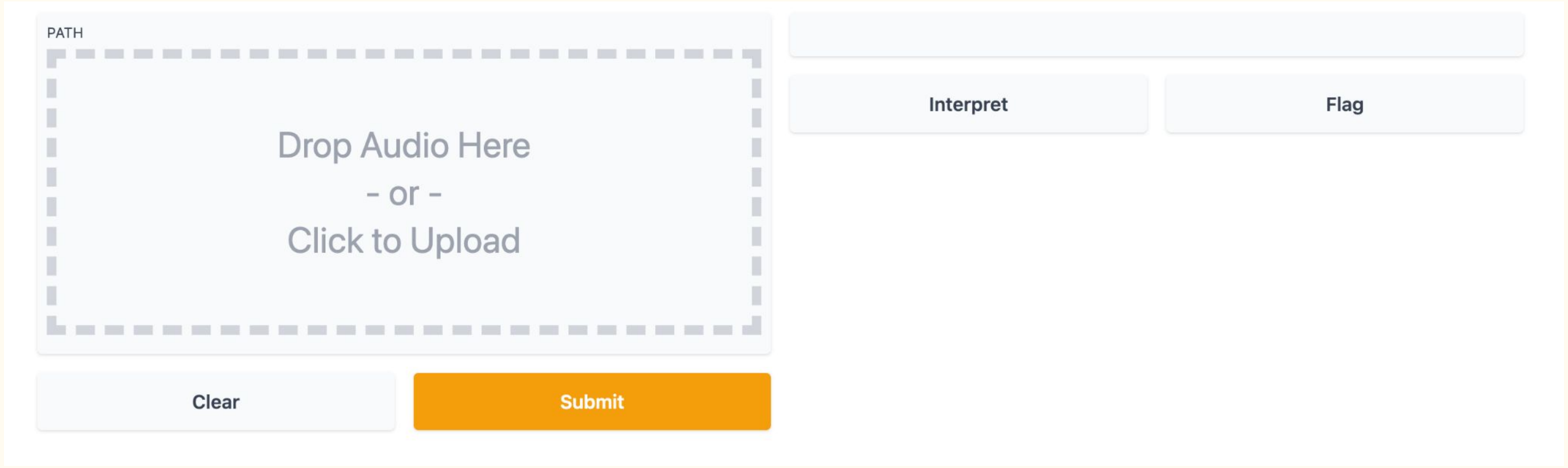
```python
import gradio as gr
```

```python
def predict_audio(path):
  MFCC = save_mfcc(path, num_segments=10)
  X = np.array(MFCC["mfcc"])
  X = X[...,np.newaxis]
  print(X)
  prediction = model.predict(X)
  print(prediction)
  map = {0:"BrunoMars",
         1: "Greenday",
         2:  "Migos",
         3:  "Beyonce",
         4:  "Maroon5"}
  predicted_index = np.argmax(prediction, axis=1)
  print(predicted_index)
  import statistics as st
  ans = map[st.mode(predicted_index)]
  return ans
```

```python
from IPython.lib.display import Audio
a = gr.inputs.Audio(source="upload", type="filepath", label=None, optional=False)
label = gr.outputs.Label(num_top_classes=5)

gr.Interface(fn=predict_audio, inputs=a, outputs=label,interpretation='default').launch(debug='True')
```

1. Import **gradio**
2. Define **prediction model**, mapping and algorithm for the music data
3. Create **user interface** with Gradio with the prediction model to output the accurate result

Application Sample Output



GUI Demonstration

Web application: `https://20030.gradio.app/`

Applications of our Model

- Integrating this can add an feature of sound search engine to the application like spotify, youtube song finder etc.

+ Music artist identification =

- It can be used by currently available sound search engines which are build on the older approaches were the feature selection is limited to few and as general as amplitude, noise and all.

+ Music artist identification =

- It can be upgraded to web sound search engine by implementing the web database.

Music Artist Identification + Web_based Database =

Insights

- We learnt that cnn is capable of giving the most accuracy in both the scenario, that is basic data and the augmented data.
- The data scaled to higher number of dimensions gave the accurate result.
  - According to earlier approaches the data was scaled to few features -> model was more complex -> less accurate
  - Then it was scaled to 13 dimensions of MFCC -> simpler model -> more accurate prediction
- Adding the noisy data to the training set improves the accuracy.
- Songs in different bases and beats can be taken care of by scaling the data into noise.

Challenges

- Artist classification is additionally difficult because each artist has a limited number of training examples, which vary artistically from song to song and over time as an artist's style evolves.
- To deal with the different variations of the songs.
    - Firstly if the song is played with different base or with some variations in beats it was not able to recognize the artist.
    - If some kind of noise was there in the audio clip it use to fail to recognize the artist correctly.
    - This was resolved by adding upgraded audio to the data
- Taking the input audio in desired format.
    - The input to model required is json
    - While the one we was getting was either mp3, wav
    - So had to do the data pre processing.
- Had to make the dataset manually as there is no such dataset available online.
    - Downloaded the songs and breaked the audio into smaller clips manually.
    - As the dataset was made by us we could not cover the large number of artist considering the time constraints.

Future Upgradation

- We are planning to implement the hybrid model that is CRNN.
    - Right now the accuracy is as good as 91.9%
    - And by implementing this model the complexity will increase
    - While we have no idea by how much the complexity is going to change.
- Right now we have used the MFCC to represent the audio features, we can try implement it using audios as a spectrogram, as it allows convolutional layers to learn global structures
- A label for fallback recognition can be added to the model.
    - One way to do that is we add an entire set of data with random artists other than the desired ones and then building model based on this training data.
    - The other way to do that is to modify the model that can take care of the fallback intents while predicting the artist.

Reference

- [1]https://medium.com/@kalyaniavhale7/tutorial-on-gradio-library-ecb8055923a1
- [2]http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/
- [3]https://adventuresinmachinelearning.com/keras-lstm-tutorial/
- [4]https://github.com/musikalkemist/DeepLearningForAudioWithPython

THANK YOU :)