UE17EC415

SPEECH PROCESSING

MINI PROJECT REPORT

Assignment-2

MUSIC GENRE CLASSIFICATION

Abhilash P
PES1201700366

Gayatri Sreenivasan
PES1201700434

Vittal Srinivasan
PES1201700310

Introduction

Music analysis is a diverse and interesting field. A music session represents a moment for the user from which information can be retrieved. Music genre classification is one of the many branches of Music Information Retrieval. Companies such as Spotify and Soundcloud use music classification to produce recommendations for their customers and determining the genre of the music is the first step in that direction.

Dataset

We use GTZAN genre collection dataset for classification.
http://marsyas.info/downloads/datasets.html

The dataset consists of 10 genres:

1. Classical
2. Country
3. Disco
4. Hiphop
5. Jazz
6. Metal
7. Pop
8. Reggae
9. Rock
10. Blues

Each genre contains 100 songs. Therefore the dataset used contains 1000 songs totally.

Methodology

Machine Learning techniques have proved to be quite successful in extracting trends and patterns from the large pool of data. The same principles are applied in Music Analysis also.

1. **Preprocessing the audio input**

   Sound is represented in the form of an audio signal having various parameters such as frequency, bandwidth, decibel etc. These sounds are available in many formats which allow for the computer to read and analyse them. Here we have used the .wav format. We use the Python library Soundfile for reading the audio files.

2. Feature Extraction

The audio signal consists of many features. We extract the characteristics that are relevant to our classification problem. The process of extracting features and using them for analysis is called feature extraction. We use the librosa and python_speech_features library for feature extraction. Some of the features we extracted for music genre classification include:

1. Zero Crossing Rate:
   The zero crossing rate is the rate of sign-changes along a signal (the rate at which the signal changes from positive to negative or back). It usually has higher values for highly percussive sounds like those in metal and rock.
2. Spectral Centroid:
   It indicates where the ”centre of mass” for a sound is located and is calculated as the weighted mean of the frequencies present in the sound. As compared to a blues genre song whose number of frequencies is spread evenly throughout its length, the metal song has more frequencies towards the end. So the spectral centroid for blues song will lie somewhere near the middle of its spectrum while that for a metal song would be towards its end.
3. Spectral Rolloff:
   It is a measure of the shape of the signal. It represents the frequency below which a specified percentage of the total spectral energy, e.g. 85%, lies.
4. Mel-Frequency Cepstral Coefficients:
   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. It models the characteristics of the human voice. We have taken 20 MFCC’s as mean of all frames of the audio signal.
5. Chroma Frequencies:
   Chroma features are a powerful representation for music audio in which the entire spectrum is projected onto 12 bins representing the 12 distinct semitones (or chroma) of the musical octave.
6. Spectral Flatness:
   Spectral flatness (or tonality coefficient) is a measure to quantify how much noise-like a sound is, as opposed to being tone-like. A high spectral flatness (closer to 1.0) indicates the spectrum is similar to white noise. It is often converted to decibel.

In total, we have extracted 36 features from each audio signal.

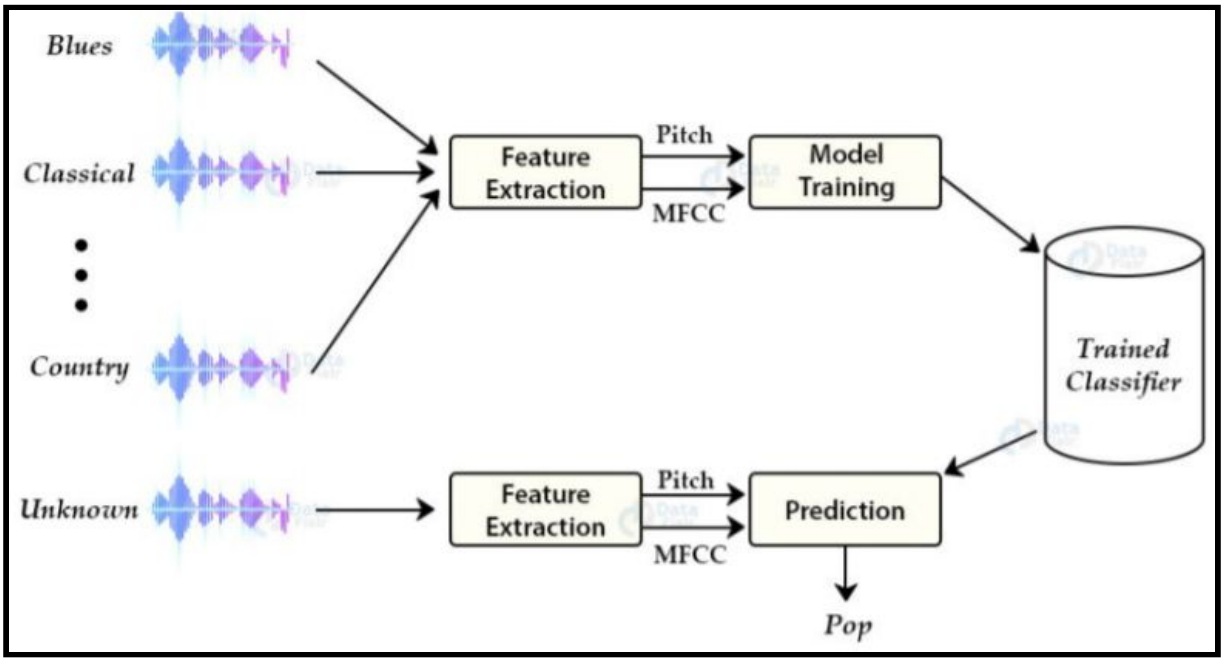
3. **Classifier**

   We can use existing classification algorithms to classify the songs into different genres. We can either use the spectrogram images directly for classification or can extract the features and use the classification models on them. Here we have extracted the features directly for classification. We use a 1D Convolutional Neural Network architecture to do the same. It consists of Average Pooling, Batch Normalization, Dropout and Dense layers

4. **Output Prediction**

   Given the input, the output predicted is one out of 10 classes of genre being - disco, jazz, blues, country, hiphop, pop, rock, classical, metal, reggae.

Block Diagram



Algorithm

We follow the following steps:

1. Preprocess the audio signal.
2. Extract the 36 features.
3. Use 1D CNN architecture in order to construct the classifier.
4. Pass the input to the classifier for training the model.
5. Input test data for prediction.
6. Print the predicted output.

Code

```
import math
import Signal_Analysis
from Signal_Analysis import features
from Signal_Analysis.features import signal
import numpy as np
import scipy.io.wavfile as wavfile
import librosa
import soundfile as sf
import warnings
warnings.filterwarnings("ignore")
import pickle
import os,sys
import sklearn.mixture
from sklearn import preprocessing
import python_speech_features as mfcc
import scipy
from sklearn.model_selection import train_test_split
from scipy.fftpack import dct

#Code to preprocess the input audio files and extract the 36 features
from each audio.

def extract_mfcc(audio,sr):
    mfcc_feature = preprocessing.normalize(mfcc.mfcc(audio,rate, 0.025,
0.01,20,nfft = 1200, appendEnergy = True))
    mfcc_feature=np.mean(mfcc_feature,axis=0)
    return mfcc_feature

def extract_spec_cent(audio,sr):
  a=np.mean(librosa.feature.spectral_centroid(audio,sr))
  return np.array([a*2/sr])

def extract_flatness(audio,sr):       #returns value in [0,1] per frame
    return
np.array([np.mean(librosa.feature.spectral_flatness(audio))])

def extract_srolloff(audio,sr):   #returns per frame in Hz
    return
np.array([np.mean(librosa.feature.spectral_rolloff(audio,sr))])
```

```
def extract_zcr(audio,sr):
  return np.array([np.mean(librosa.feature.zero_crossing_rate(audio))])

def extract_chroma(audio,sr):
  chroma
=np.mean(np.transpose(preprocessing.normalize(librosa.feature.chroma_st
ft(audio,sr))),axis = 0)
  return chroma
path = 'Data/genres_original'
os.listdir(path)
genres = os.listdir(path)
labels = []
features = []
count = 0
a = {}
for i in genres:
  g = np.asarray(count)
  a[count] = i

  for j in os.listdir(path+'/'+i):
    if (j!='.ipynb_checkpoints' and j!='jazz.00054.wav'):
      audio,sr = sf.read(path+'/'+i+'/'+j)
      aud=librosa.util.normalize(audio)
      l1=[]
      l1=extract_mfcc(aud,sr).tolist()
      l1.extend(extract_spec_cent(aud,sr).tolist())
      l1.extend(extract_srolloff(aud,sr).tolist())
      l1.extend(extract_chroma(aud,sr).tolist())
      l1.extend(extract_flatness(aud,sr).tolist())
      l1.extend(extract_zcr(aud,sr).tolist())
      features.append(l1)
      labels.append(g)
  count+=1
features = np.array(features)
labels = np.array(labels)
np.save('drive/My Drive/genre_features', x)
np.save('drive/My Drive/genre_labels', y)

#Code for the CNN classifier
import tensorflow as tf
import numpy as np
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,
Activation,Flatten,Conv2D,MaxPooling2D
import pickle
from sklearn.model_selection import train_test_split
import keras
import os
import matplotlib.pyplot as plt
from matplotlib import pyplot
from keras.preprocessing import image
from PIL import Image
from sklearn.model_selection import train_test_split
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Dropout, Flatten
from tensorflow.python.keras.layers import Conv2D, MaxPooling1D, Input,
concatenate, BatchNormalization, Conv1D, AveragePooling1D
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

X = np.load('genre_features.npy')
Y = np.load('genre_labels.npy')
X=np.expand_dims(np.array(X),axis=2)
print(X.shape)
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size =
0.2)

#CNN model definition
model = Sequential()
model.add(Conv1D(256, 5,activation='relu', input_shape=(36,1)))
model.add(BatchNormalization())
model.add(AveragePooling1D(pool_size=2))

model.add(Conv1D(128, 3, activation='relu'))
model.add(BatchNormalization())
model.add(AveragePooling1D(pool_size=2))

model.add(Conv1D(64, 3, activation='relu'))
model.add(AveragePooling1D(pool_size=2))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
```

```
model.add(Dropout(0.2))
model.add(Dense(256,activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(128,activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

model.summary()
model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['accuracy'])

history =model.fit(x_train, y_train, epochs=500,
verbose=1,validation_data=(x_test, y_test) )
model.save('speech_genre.model')

#plotting the accuracies
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,5.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
#testing the model
model = tf.keras.models.load_model("speech_genre.model")
```

```
loss, accuracy = model.evaluate(x_test,y_test)
print('Test accuracy :', accuracy)

#Retrieve a batch of features from the test set
predictions = model.predict(x_test)
# Apply a sigmoid since our model returns logits
#print(predictions)
predictions =np.argmax(predictions, axis=1)
labels=np.argmax(y_test, axis=1)

print('Predictions:\n', predictions)
print('Labels:\n', labels)

cm=confusion_matrix(labels,predictions)
print(cm)

f=f1_score(labels,predictions,average=None)
print('f1 score:',f)
```

Results

1. **The CNN model architecture is as follows:**

- Three Conv1D layers were used with nodes varying from 64 to 256 and activation function relu.
- Each layer is filled by an average pool size of 2 and Batch Normalisation.
- Dropout of 0.2 is added followed by three dense layers varying with 512 to 128 nodes.
- Final dense layer has 10 nodes with a softmax activation function to predict the probability of each input.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 32, 256)           1536
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 256)           1024
_________________________________________________________________
average_pooling1d (AveragePo (None, 16, 256)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 14, 128)           98432
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 128)           512
_________________________________________________________________
average_pooling1d_1 (Average (None, 7, 128)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 5, 64)             24640
_________________________________________________________________
average_pooling1d_2 (Average (None, 2, 64)             0
_________________________________________________________________
dropout (Dropout)            (None, 2, 64)             0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 512)               66048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 357,706
Trainable params: 356,938
Non-trainable params: 768
_________________________________________________________________
```

Fig. CNN model Architecture

2. **Training:**

The model is executed for 500 epochs leading to a training accuracy of 98.12% and validation accuracy of 51.50%.

```
Epoch 499/500
25/25 [==============================] - 1s 24ms/step - loss: 0.0609 - accuracy: 0.9787 - val_loss: 3.6459 - val_accuracy: 0.4950
Epoch 500/500
25/25 [==============================] - 1s 25ms/step - loss: 0.0517 - accuracy: 0.9812 - val_loss: 3.5358 - val_accuracy: 0.5150
```

Fig. Epochs for the CNN model

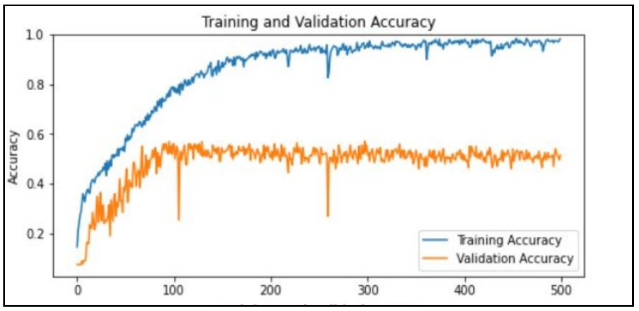


Fig. Plot for training and validation accuracy

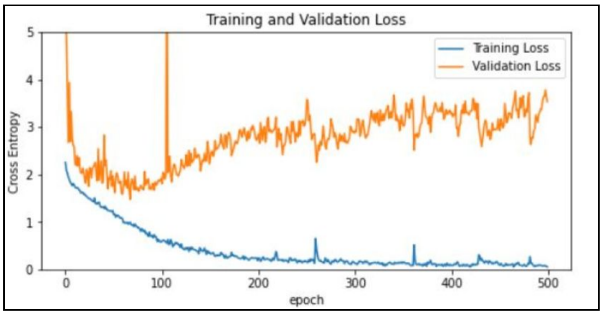


Fig. Plot for training and validation loss

3. **Testing:**

The model was tested with 200 samples which gave an accuracy around 50%.

```
7/7 [==============================] - 0s 5ms/step - loss: 3.5358 - accuracy: 0.5150
Test accuracy : 0.5149999856948853
```

Fig. Testing accuracy for Model

The predictions along with labels for each sample is given below:

```
Predictions:
 [8 4 8 3 8 3 8 4 0 8 2 3 6 7 7 0 3 5 0 8 8 2 0 4 4 2 8 4 7 6 2 3 8 4 8 7 8
 5 1 8 7 4 0 8 9 0 8 6 1 0 3 4 4 8 9 1 6 8 2 7 0 1 3 4 0 3 8 3 2 4 6 5 5 8
 1 1 7 3 0 8 3 9 4 2 5 2 7 2 8 0 9 3 6 1 5 0 2 9 0 7 0 4 7 2 9 0 0 8 3 2 7
 9 4 8 6 3 9 9 1 3 0 1 3 5 0 1 0 6 2 2 0 1 8 7 2 4 8 7 6 4 3 3 4 0 5 3 2 5
 4 3 5 2 5 1 5 2 5 6 2 0 9 4 1 3 0 6 3 3 6 4 7 0 7 8 3 7 9 2 6 6 1 1 9 2 5
 5 0 8 0 7 8 4 4 6 1 3 2 8 1 2]
Labels:
 [9 4 8 3 8 3 5 4 0 9 8 3 6 7 7 0 3 5 0 6 7 5 4 9 4 9 8 4 7 7 9 3 0 6 0 7 2
 6 1 8 2 4 2 6 9 3 8 6 2 6 2 4 1 6 5 7 6 6 2 7 8 1 3 0 1 3 9 3 2 8 6 5 9 8
 0 9 7 3 5 9 3 0 4 1 9 2 7 2 1 0 5 1 1 1 5 4 2 5 8 7 0 4 7 3 9 6 0 8 3 2 2
 9 9 1 7 3 9 9 7 3 0 5 3 0 0 1 1 1 2 2 8 7 4 9 1 1 1 7 6 4 3 5 8 3 9 3 2 5
 6 3 6 2 5 7 5 2 5 6 2 0 1 4 3 3 0 9 0 1 5 4 7 6 7 8 7 7 2 0 6 1 0 5 3 2 5
 5 2 6 6 9 0 4 6 6 9 3 2 8 6 3]
```

Fig. Predicted labels and given labels for the test inputs

A confusion matrix is a table that is used to describe the performance of a classification model on a set of test data for which the true values are known.
In statistical analysis of classification, the F-score or F-measure is a measure of a test's accuracy.
The confusion matrix and F1 scores for each classifier are mentioned below:

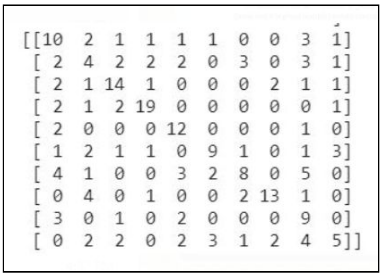


Fig.Confusion matrix of predicted output

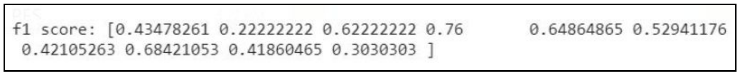


Fig. F1 score of each class of predicted output

4. **Improvements on the Accuracy:**
   Accuracy can be improved by experimenting with the model used. Using a CNN model on the spectrogram images itself might give a better accuracy.
   RNN's could also be used for the model as they give better results when dealing with time series data like audio signals.

Conclusion

Thus, music genre classification has been successfully performed on a dataset containing 1000 songs. Hence the genre the input has been predicted with a test accuracy of around 50 percent.