INTERNSHIP PROJECT REPORT

1. Project Title

MUSIC GENRE CLASSIFICATION USING CNN

Submitted By:

Miss. Priti Anandrao Shinde.

Miss. Rajashree Vishnu Shinde.

2. Abstract / Project Summary

The "Music Genre Classification using Python" project aims to leverage machine learning techniques for the automatic classification of music genres. The project involves the collection of a diverse dataset comprising audio files from various genres to facilitate robust model training. Key steps include feature extraction using tools like Librosa, preprocessing to handle data irregularities, and the selection of an appropriate machine learning model.

The chosen model undergoes extensive training on the prepared dataset, with a focus on optimizing hyper-parameters for superior performance. Evaluation metrics such as accuracy, precision, recall, and F1-score are employed to assess the model's effectiveness in genre classification. The results, including a confusion matrix, are presented to provide insights into the model's strengths and limitations. To enhance user accessibility, the developed model is deployed in a user-friendly application or web service. Users can input audio files and receive accurate genre predictions. Comprehensive documentation, including code and model details, is provided for transparency and future development.

As part of future improvements, the project suggests exploring transfer learning to further enhance model accuracy and considering user feedback for continuous refinement. The "Music Genre Classification using Python" project contributes to the field of audio processing and machine learning, offering a practical solution for automating music genre categorization.

3. Technologies Used

1. Kaggle – For downloading the dataset
2. Python
3. Libraries Used:

   - ➢ Pandas
   - ➢ Matplotlib
   - ➢ Librosa
   - ➢ Numpy
   - ➢ Os
   - ➢ IPython.display
   - ➢ Tensorflow
   - ➢ Sklearn

   - ➢ Data Preprocessing and Analysis:
     - Numpy & Pandas: For data manipulation and analysis.
     - Matplotlib: For data visualization
     - Librosa: Used for audio feature extraction, providing tools to compute features like Mel-frequency cepstral coefficients (MFCC), chroma, and spectral contrast.
     - Tenserflow: TensorFlow can be used for building and training neural network models.

4. Media Display:

   - ➢ IPython.display : It allows you to embed various types of media, including images, audio, video, and HTML, directly within your interactive Python sessions.

4. Algorithms Used

- Artificial Neural Network:

  Artificial Neural Networks (ANNs), especially deep neural networks, can be used in Music Genre Classification to automatically learn intricate patterns and representations from audio data. Convert raw audio data into relevant features that the neural network can learn from. Common features include Mel-frequency cepstral coefficients (MFCCs), chroma features, and spectral contrast. Normalize the extracted features to ensure consistent input scaling for the neural network. This step is crucial to facilitate the convergence of the model during training.

- Convolutional Neural Network:

  Convolutional Neural Networks (CNNs) are widely used in Music Genre Classification due to their ability to automatically learn hierarchical features from input data, making them effective for tasks involving sequential or spatial patterns. Audio data is converted into a format suitable for CNNs. This typically involves extracting features like Mel-frequency cepstral coefficients (MFCCs) or spectrograms from audio samples. Design a CNN architecture suitable for music genre classification. A common approach includes a stack of convolutional layers followed by pooling layers for hierarchical feature extraction. Flatten the output and connect to one or more fully connected layers for classification. In CNN, the Sequential layer is commonly used to define the architecture of the neural network. The Sequential model allows you to create a linear stack of layers, where each layer is added sequentially.

5. Components/Modules of the Project

- Data Preprocessing

  - Loading and preprocessing audio data. Key Functions: Read audio files and extract relevant features (e.g., MFCCs).Normalize and preprocess the data for model input. Split the dataset into training, validation, and testing sets.

- Model Architecture

  - Specify convolutional layers for feature extraction. Include pooling layers for spatial down-sampling. Flatten the output and add fully connected layers for classification. Compile the model with an appropriate optimizer, loss function, and metrics.

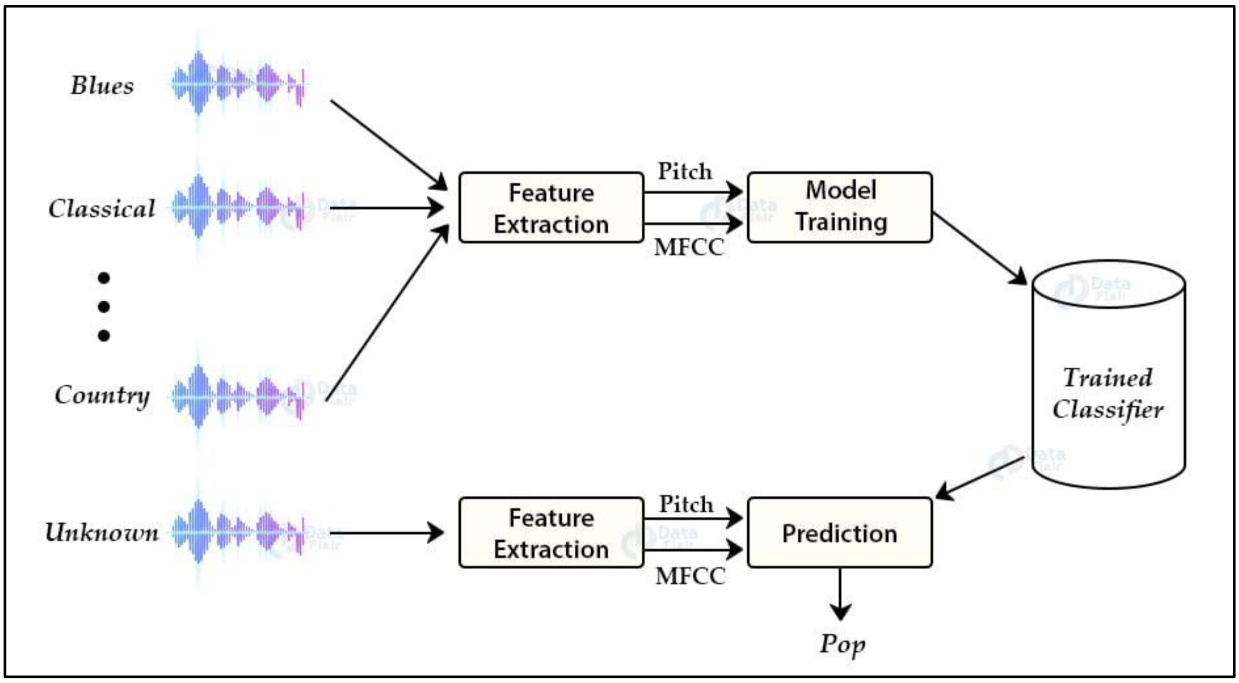
- Training Module

  - Managing the training process of the CNN. Split the dataset into training and validation sets. Train the CNN model with the training data. Adjust hyper-parameters and monitor training progress. Save checkpoints of the trained model.

- Model Evaluation

  - Responsibility: Handling the evaluation of the trained model on test data. Evaluate the model's performance on the testing set. Calculate key metrics such as accuracy, precision, recall, and F1-score.Generate visualizations like confusion matrices. Provide insights into the model's generalization capabilities.

6. Component Diagram



7. Project Algorithm Steps

a) Importing Libraries:

```
import pandas as pd
import numpy as np
import os import librosa
import IPython.display as ipd
import librosa.display
import matplotlib.pyplot as plt
matplotlib inline
```

b) Importing Dataset:

```
audio_dataset_path='/content/drive/MyDrive/ColabNotebooks/a
rchive/Data/genres_original
```

c) Feature Extraction:

```
def features_extractor(file):
audio.sample_rate=librosa.load(file_name,res_type='kaiser_fast
")
mfccs_features = librosa.feature.mfcc(y=audio, sr-sample_rate,
n_mfcc-40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)
return mfccs_scaled_features
```

d) Label Encoding:

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
y=to_categorical(labelencoder.fit_transform(y))
```

e) Splitting The Dataset Into The Training Set And Test Set:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

f) Making Sequential Layers:

```
model=Sequential()
model.add(Dense(1024,input_shape=(40,), activation="relu"))
model.add(Dropout(0.3)).............
###final layer
model.add(Dense(num_labels, activation="softmax"))
```

g) Model Summary:

```
model.summary()
[It gives- Total params: 741674 (2.83 MB)
Trainable params: 741674 (2.83 MB)
Non-trainable params: 0 (0.00 Byte)
```

h) Model Evaluation:

```
model.evaluate(x_test,y_test, verbose=0)
```

i) Model Prediction :

```
model.predict(x_test).argmax(axis=1)
```

j) Classification on new data:

```
filename="/content/drive/MyDrive/ColabNotebooks/archive/Data/genres_original/classical/classical.00093.wav"
audio.sample_rate=librosa.load(filename,res_type='kaiser_fast)
```

```
mfccs_features=librosa.feature.mfcc(y=audio, sr=sample_rate,
n_mfcc=40)
mfccs_scaled_features=np.mean(mfccs_features.T, axis=0)
print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_probabilities =model.predict(mfccs_scaled_features)
predicted_label = np.argmax(predicted_probabilities, axis=1)
print(predicted_label)
prediction_class=labelencoder.inverse_transform(predicted_la
bel)
prediction_class
```

8. Comparison of All the Algorithms Tested

Convolution Neural Network and K-nearest Neighbour classification are two distinct algorithms with different approaches which I have tested for the project and implemented Convolution Neural Network on the basis of it's advantages in the project. Due to their ability to automatically learn hierarchical features from audio data.

1. Automatic Feature Learning:
   - CNN: It can automatically learn hierarchical features directly from raw input data, such as audio spectrograms. This ability to learn relevant representations can be crucial in capturing complex patterns and dependencies in audio data.

   - KNN: It relies on predefined features and may not adapt well to high-dimensional and complex data like audio spectrograms without extensive manual feature engineering.

2. Parameter Tuning and Training:
   - CNN: It require tuning hyperparameters, but they can be trained end-to-end with backpropagation. The learning process involves adjusting internal parameters during training, allowing the model to adapt to the characteristics of the data.

   - KNN: It does not have a traditional training phase, but selecting an appropriate value for k (number of neighbors) and choosing a suitable distance metric is crucial. It may not adapt to the data as seamlessly as CNNs during the learning process.

3. Spatial Relationships in Spectrograms:
   - CNN: Designed for image analysis, CNNs excel at capturing spatial hierarchies. In music genre classification, where audio features often have spatial dependencies in spectrograms, CNNs are better suited to recognize intricate patterns.

   - KNN: It treats features independently, potentially missing spatial structures present in spectrogram data.

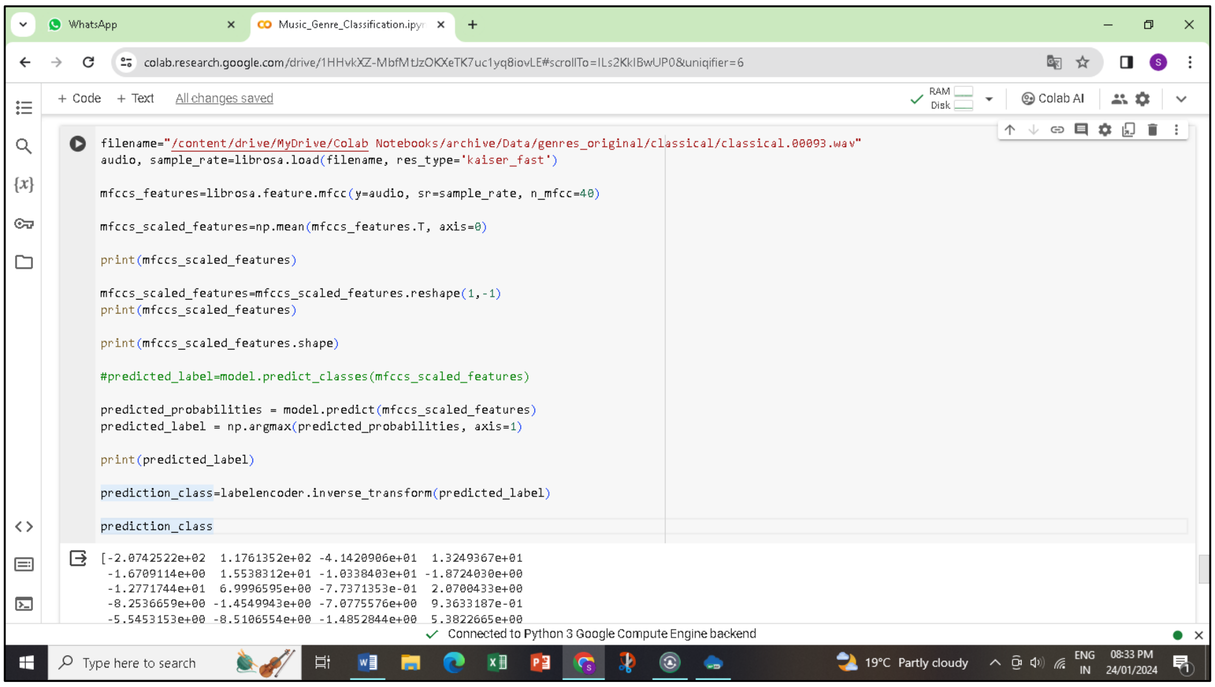
4. Generalization to Unseen Data:
   - CNN: Deep CNNs have the capacity to generalize well to unseen data. They can capture both low-level and high-level features, enabling recognition of patterns even in complex audio data not seen during training.

   - KNN: KNN may struggle with generalization, particularly in high-dimensional spaces. It can be sensitive to irrelevant features and may not perform optimally when presented with new and unseen audio data.

5. Computational Efficiency:
   - CNN: While training deep CNNs can be computationally intensive, once trained, they can make predictions efficiently, especially for large datasets. Optimized libraries and hardware acceleration (e.g., GPUs) can enhance efficiency.

   - KNN: In this predictions can be computationally expensive, especially as the dataset size increases. Calculating distances to all training samples during prediction may result in longer prediction times.

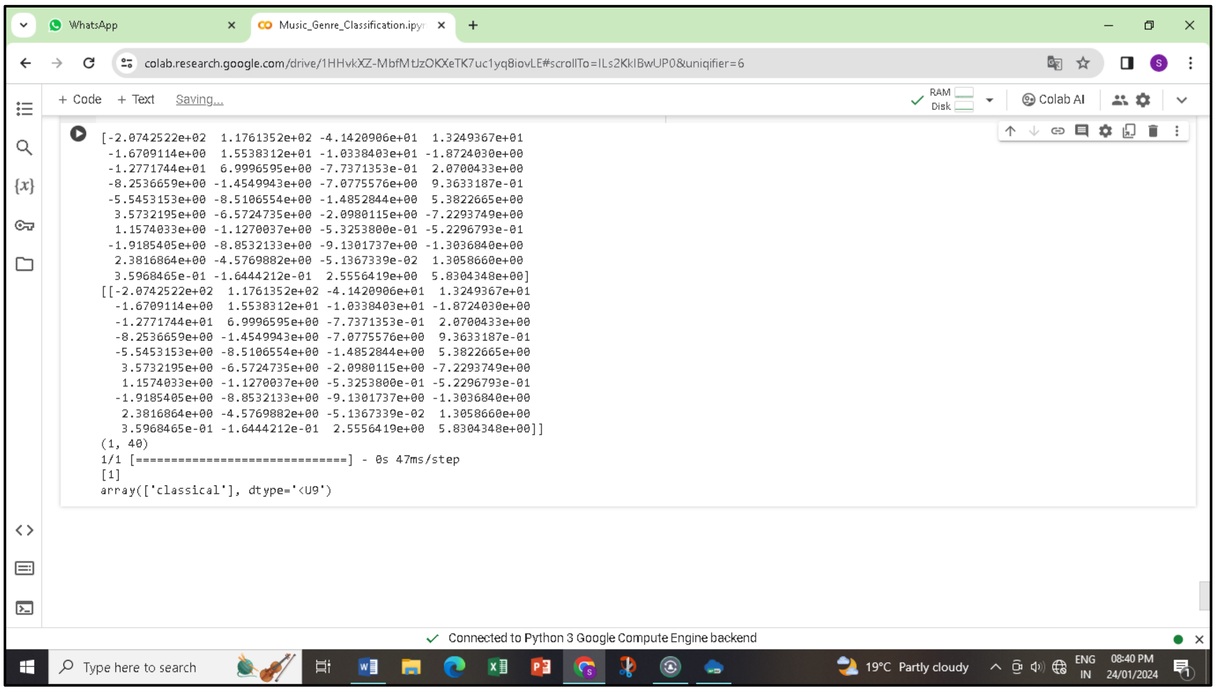
9. Output Screenshot



```
filename="/content/drive/MyDrive/Colab Notebooks/archive/Data/genres_original/classical/classical.00093.wav"
audio, sample_rate=librosa.load(filename, res_type='kaiser_fast')

mfccs_features=librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)

mfccs_scaled_features=np.mean(mfccs_features.T, axis=0)

print(mfccs_scaled_features)

mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)

print(mfccs_scaled_features.shape)

#predicted_label=model.predict_classes(mfccs_scaled_features)

predicted_probabilities = model.predict(mfccs_scaled_features)
predicted_label = np.argmax(predicted_probabilities, axis=1)

print(predicted_label)

prediction_class=labelencoder.inverse_transform(predicted_label)

prediction_class
```

```
[-2.0742522e+02  1.1761352e+02 -4.1420906e+01  1.3249367e+01
 -1.6709114e+00  1.5538312e+01 -1.0338403e+01 -1.8724030e+00
 -1.2771744e+01  6.9996595e+00 -7.7371353e-01  2.0700433e+00
 -8.2536659e+00 -1.4549943e+00 -7.0775576e+00  9.3633187e-01
 -5.5453153e+00 -8.5106554e+00 -1.4852844e+00  5.3822665e+00
```

Connected to Python 3 Google Compute Engine backend



```
[-2.0742522e+02  1.1761352e+02 -4.1420906e+01  1.3249367e+01
 -1.6709114e+00  1.5538312e+01 -1.0338403e+01 -1.8724030e+00
 -1.2771744e+01  6.9996595e+00 -7.7371353e-01  2.0700433e+00
 -8.2536659e+00 -1.4549943e+00 -7.0775576e+00  9.3633187e-01
 -5.5453153e+00 -8.5106554e+00 -1.4852844e+00  5.3822665e+00
  3.5732195e+00 -6.5724735e+00 -2.0980115e+00 -7.2293749e+00
  1.1574033e+00 -1.1270037e+00 -5.3253800e-01 -5.2296793e-01
 -1.9185405e+00 -8.8532133e+00 -9.1301737e+00 -1.3036840e+00
  2.3816864e+00 -4.5769882e+00 -5.1367339e-02  1.3058660e+00
  3.5968465e-01 -1.6444212e-01  2.5556419e+00  5.8304348e+00]
[[-2.0742522e+02  1.1761352e+02 -4.1420906e+01  1.3249367e+01
  -1.6709114e+00  1.5538312e+01 -1.0338403e+01 -1.8724030e+00
  -1.2771744e+01  6.9996595e+00 -7.7371353e-01  2.0700433e+00
  -8.2536659e+00 -1.4549943e+00 -7.0775576e+00  9.3633187e-01
  -5.5453153e+00 -8.5106554e+00 -1.4852844e+00  5.3822665e+00
   3.5732195e+00 -6.5724735e+00 -2.0980115e+00 -7.2293749e+00
   1.1574033e+00 -1.1270037e+00 -5.3253800e-01 -5.2296793e-01
  -1.9185405e+00 -8.8532133e+00 -9.1301737e+00 -1.3036840e+00
   2.3816864e+00 -4.5769882e+00 -5.1367339e-02  1.3058660e+00
   3.5968465e-01 -1.6444212e-01  2.5556419e+00  5.8304348e+00]]
(1, 40)
1/1 [==============================] - 0s 47ms/step
[1]
array(['classical'], dtype='<U9')
```

Connected to Python 3 Google Compute Engine backend

10. GitHub Link of the Project

https://github.com/Priti-Anandrao-Shinde/WisdomSprouts_Internship2024/blob/main/Music_genre.ipynb

11. Future Scope

The Music Genre Classification project using Python has several avenues for future improvements and expansions. Some potential areas of expansion and improvement include:

1. Transfer Learning: Explore the use of pre-trained neural network models, such as those trained on large audio datasets. Fine-tune these models for music genre classification to benefit from their learned features.

2. Multimodal Classification: Explore incorporating additional data modalities, such as album cover images or lyrics, to perform multimodal music genre classification for a more comprehensive analysis.

3. Real-time Classification: Extend the project to handle real-time music genre classification. This could involve building a system that classifies music as it is being played or streamed.

4. Web Application or Mobile App: Develop a user-friendly web application or mobile app for music genre classification. This allows users to interact with the model easily and explore its capabilities.

5. Continuous Model Optimization: Implement mechanisms for continuous model optimization using techniques like online learning, ensuring that the model adapts to changes in musical trends and patterns over time.