Assignment 3

Topic 4: Music Genre Classification

Sid 57853900

1. Background

Music genre classification is a challenging problem in music information retrieval. It has applications in various domains, such as music recommendation systems, digital libraries, and automated music organization. The ability to automatically categorize music into different genres can enhance the user experience by providing personalized music suggestions and efficient navigation through vast music collections.

Over the years, many approaches have been developed for automatic genre classification of music, including rule-based systems, feature-based models, and deep learning-based models. While rule-based systems rely on manually defined rules, feature-based models extract relevant features from the audio signal and classify the music based on these features. On the other hand, deep learning-based models learn hierarchical representations of the audio signal using neural networks and can achieve state-of-the-art performance in genre classification tasks.

This project will use audio files from the GTZAN genre dataset, a widely recognized benchmark for evaluating music genre classification algorithms. This dataset comprises 10 music genres, each represented by 100 audio tracks. The tracks are diverse in instrumentation, style, and origin, reflecting the complexity and richness of different musical traditions. The goal is to develop a deep-learning model that can accurately classify these audio files into their respective genres based on these low-level features. This involves preprocessing the audio data to extract relevant features, designing a neural network architecture that can learn complex patterns from these features, and training the model to minimize classification errors.

2. Method

For this project, a Convolutional Neural Network (CNN) is chosen as the primary algorithm for several reasons:

Feature Learning: CNNs are known for their ability to automatically and adaptively learn features from data. In the context of music genre classification, the network can learn to recognize complex patterns in the audio signal indicative of specific genres without manual feature engineering.

Hierarchical Representation: Music data, like images, has a hierarchical structure where local patterns (e.g., individual notes or chords) combine to form more global structures (e.g., a melody or a chord progression). CNNs are well-suited to capturing such hierarchical patterns

through their multi-layer architecture.

Efficiency: CNNs can process data in a sliding window fashion, making them efficient for handling large audio files. This is particularly beneficial when dealing with high-resolution audio data common in music genre classification tasks.

3. Experiment

3.1 Exploratory Data Analysis

Audio data can be a challenging format to work with due to its disorganized nature and the detailed knowledge of digital signal processing required. To explore audio data, we will employ several methods using librosa as a powerful tool. For our sample file, we will use blues.00000.wav.

```
# plot sample file
audio_fp = '/kaggle/input/gtzan-dataset-music-genre-classification/Data/genres_original/blues/blues.00000.wav'
audio_data, sr = librosa.load(audio_fp)
audio_data, _ = librosa.effects.trim(audio_data)
```

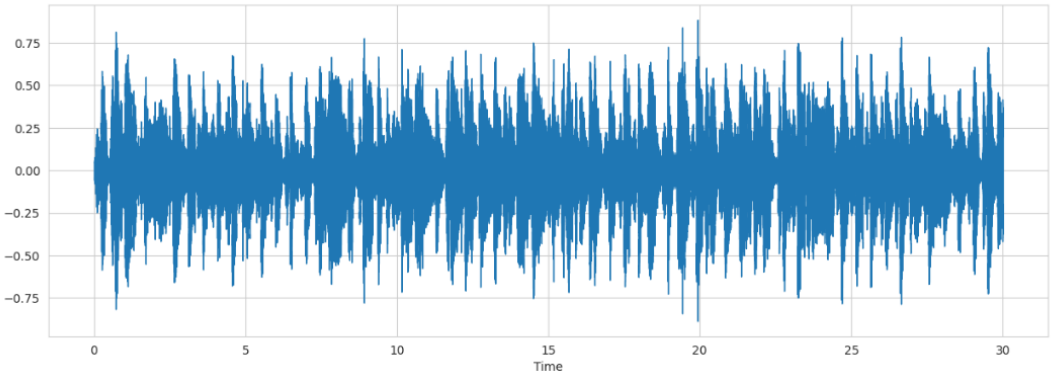
3.1.1 Display Audio

we can listen to the audio data using IPython.display.Audio(audio_data, rate=sr). This allows us to hear the melody directly.

```
IPython.display.Audio(audio_data, rate=sr)
```

3.1.2 Waveform

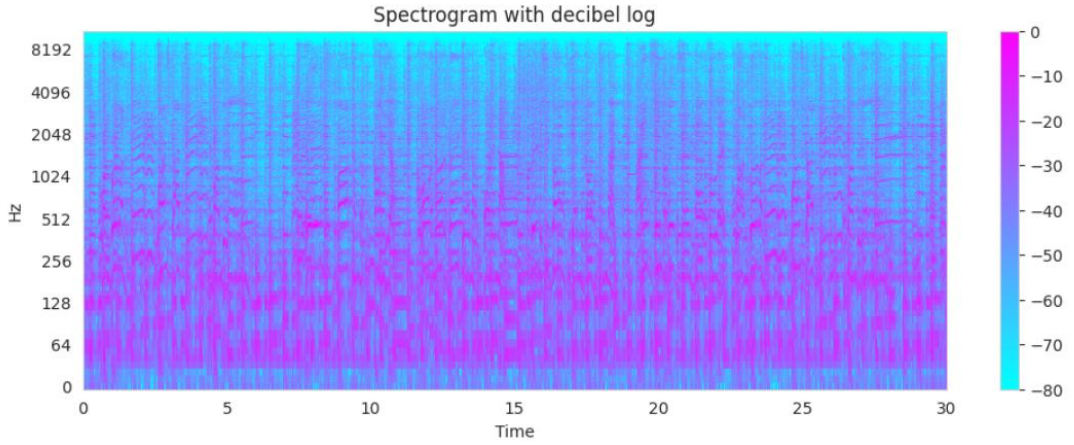
The waveform of the audio data shows the oscillations of pressure amplitude over time. This is effectively the "raw" format of audio data, defined by the sampling frequency and bit depth. The bit depth determines the precision of the amplitude values, which in turn defines the dynamic range of the audio signal.



3.1.3 Spectrogram

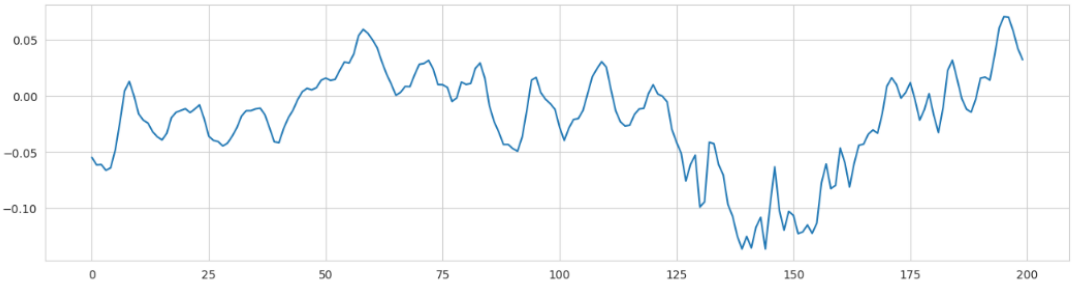
A spectrogram is a visual representation of the intensity of a signal over time, showing

different frequencies at each time step. In audio analysis, a spectrogram provides a detailed view of how frequencies change over time. It has three dimensions: frequency (y-axis), time (x-axis), and amplitude (intensity).



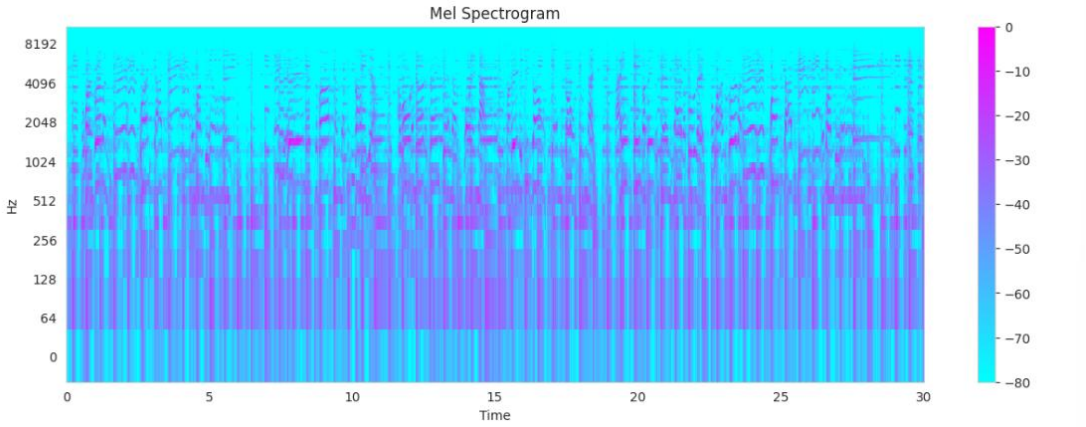
3.1.4 Zoomed Audio Wave

Zooming into the audio waveform allows us to observe the local characteristics of the audio signal in more detail. This can reveal waveform changes within specific time periods, providing insights into the signal's fine-grained structure.



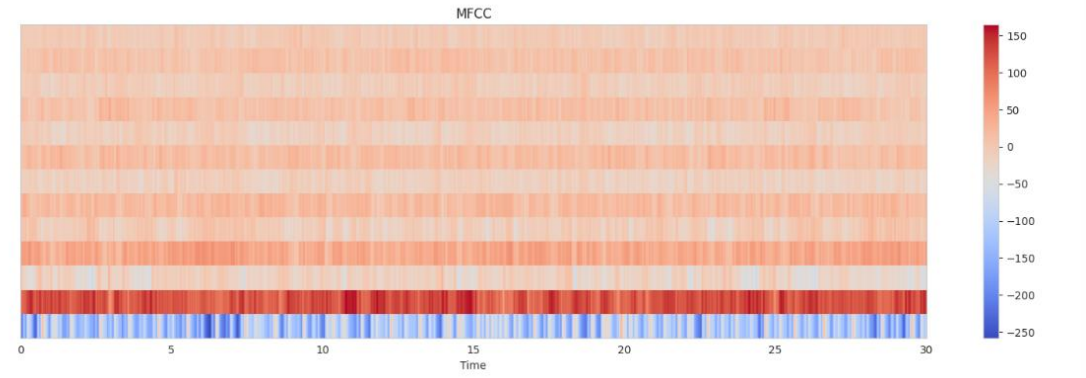
3.1.5 Mel-Spectrogram

The Mel-spectrogram is similar to a spectrogram, but the magnitudes are scaled to a "Mel-scale." This scaling aligns more closely with human perception of sound. The Mel-scale is often summarized using Mel Frequency Cepstral Coefficients (MFCCs).



3.1.6 MFCC

MFCCs (Mel-frequency cepstral coefficients) are a small set of features that accurately describe the shape of the spectral envelope. They use the Mel scale to divide the frequency band into sub-bands, and then extract cepstral coefficients using the discrete cosine transform (DCT). MFCCs are widely used in machine learning because they filter the data in a way that closely resembles human hearing, providing a more suitable format for algorithms to learn from.



3.2 Genre classification with original data

We will first use the original audio length as the experiment data and see the performance of different CNN models.

3.2.1 Import libraries

Import all the necessary libraries at the beginning, including CNN model building, visualizing audio, transforming data functions, etc.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
import sklearn.metrics as skm
import sklearn.model_selection as skms
import sklearn.preprocessing as skp
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import random
import librosa, IPython
import librosa.display as lplt
import os, sys, cv2
import numpy as np
import pandas as pd
import IPython.display as ipd
from tqdm import tqdm
import tensorflow as tf
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout, Flatten, LSTM
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Conv1D, MaxPooling1D, MaxPool1D, GaussianNoise, GlobalMaxPooling1D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
from sklearn.model_selection import KFold
```

3.2.2 Extract features

Since this is a Kaggle dataset, we will directly use the Kaggle platform to get the dataset.

We can see there are total ten gernes in this dataset

```
directory_path = '/kaggle/input/Data/genres_original'
print("Files in directory:", os.listdir(directory_path))
```

```
Files in directory: ['disco', 'metal', 'reggae', 'blues', 'rock', 'classi
cal', 'jazz', 'hiphop', 'country', 'pop']
```

Then we use the function get_mfccs which is designed to extract Mel-Frequency Cepstral Coefficients (MFCCs) from audio files stored in a specified directory. It systematically processes each audio file, segmenting them into smaller parts and computing the MFCCs for each segment. This function is particularly useful for preparing audio data for machine learning models, especially in tasks like music genre classification.

The function begins by initializing three empty lists to store the extracted MFCCs, genre names, and genre numbers. It then calculates the number of samples per track based on the specified duration and sampling rate and determines the number of samples per segment by dividing the total samples by the number of segments. The number of MFCC vectors per segment is also calculated to ensure consistency in the feature extraction process.

As the function iterates through each directory and file within the specified path, it skips the parent directory to focus on subdirectories representing different genres. For each audio file, it attempts to load the audio data using the librosa library. The audio is then segmented, and MFCCs are computed for each segment. The MFCCs are transposed to match the expected

shape for further processing.

If the computed MFCCs match the expected shape, they are appended to the list of MFCCs, along with the corresponding genre name and a numerical genre label. This process ensures that only valid MFCCs are included in the final dataset. Any errors encountered during the processing of a file are caught and reported, allowing the function to continue processing the remaining files.

Once all files have been processed, the collected MFCCs, genre names, and genre numbers are converted into NumPy arrays for easier manipulation and use in machine-learning models. The function concludes by returning these arrays, providing a structured dataset ready for training and evaluation.

```
def get_mfccs(directory_path, fs=22050, duration=30, n_fft=2048, hop_length=512, n_mfcc=13, num_segments=10):
    # Initialize lists to store MFCCs, genre names, and genre numbers
    mfccs = []
    genre_names = []
    genre_nums = []

    # Calculate the number of samples per segment and the number of MFCC vectors per segment
    samples_per_track = fs * duration
    samples_per_segment = int(samples_per_track / num_segments)
    mfccs_per_segment = math.ceil(samples_per_segment / hop_length)

    print("MFCC collection started!")
    for i, (path_current, folder_names, file_names) in enumerate(os.walk(directory_path)):
        # Skip the parent directory
        if path_current == directory_path:
            continue

        # Extract genre name from the directory structure
        genre_current = os.path.basename(path_current)

        # Process each file in the genre folder
        for file in file_names:
            file_path = os.path.join(path_current, file)

            try:
                # Load audio data
                audio, fs = librosa.load(file_path, sr=fs)

                # Extract MFCCs for each segment
                for seg in range(num_segments):
                    start_sample = seg * samples_per_segment
                    end_sample = start_sample + samples_per_segment

                    # Compute MFCCs for the current segment
                    mfcc = librosa.feature.mfcc(y=audio[start_sample:end_sample], sr=fs, n_fft=n_fft, hop_length=hop_length, n_mfcc=n_mfcc)
                    mfcc = mfcc.T  # Transpose to match the expected shape

                    # Append MFCCs and genre information if the shape matches
                    if len(mfcc) == mfccs_per_segment:
                        mfccs.append(mfcc.tolist())
                        genre_names.append(genre_current)
                        genre_nums.append(i - 1)  # Assign a numerical label to each genre

            except Exception as e:
                print(f"Error processing file {file_path}: {e}")
                continue

        print(f"Collected MFCCs for {genre_current.title()}!")

    print("MFCC collection complete!")
    # Convert lists to numpy arrays
    mfccs = np.array(mfccs)
    genre_names = np.array(genre_names)
    genre_nums = np.array(genre_nums)

    return mfccs, genre_names, genre_nums
```

Then we set the corresponding parameters and call the function. Here the parameters set include 13 MFCCs to be extracted, an FFT window size of 2048, a hop length of 512, and the decision to process each audio track as a single segment with a duration of 30 seconds. The

directory path specified points to a dataset containing various music genres.

```
n_mfcc = 13
n_fft = 2048
hop_length = 512
num_segments = 1
track_duration = 30

directory_path = '/kaggle/input/Data/genres_original'

mfccs, genres, genre_nums = get_mfccs(directory_path,
                                      fs=22050,
                                      duration=track_duration,
                                      n_fft=n_fft,
                                      hop_length=hop_length,
                                      n_mfcc=n_mfcc,
                                      num_segments=num_segments)
```

To make the label readable for the CNN we map genre names to numerical labels and convert categorical data into the number format.

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

genre_indices = np.array([label_mapping[label] for label in genres])
```

3.2.3 Split Train, Test Sets

The dataset is split into training, testing, and validation sets with test_size=0.2 and random_state=42. For the data to be used in a CNN, an additional dimension representing the number of channels is added.

```
# Train-validation-test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42, stratify=y_train)

print(f"X training data shape: {X_train.shape}, y training data shape: {y_train.shape}")
print(f"X validation data shape: {X_val.shape}, y validation data shape: {y_val.shape}")
```

```
X training data shape: (638, 1292, 13), y training data shape: (638,)
X validation data shape: (160, 1292, 13), y validation data shape: (160,)
```

```
# Add additional dimension for CNN
X_train_cnn = X_train[..., np.newaxis]
X_val_cnn = X_val[..., np.newaxis]
X_test_cnn = X_test[..., np.newaxis]

input_shape = X_train_cnn.shape[1:4]
X_train_cnn.shape
```

```
(638, 1292, 13, 1)
```

3.2.4 1-block CNN Model

First, we build a 1-block CNN model to do the classification, which consists of a Conv2D layer featuring 32 filters, a kernel size of 3x3, ReLU activation, and input shape matching the input data's shape. This is followed by a MaxPooling2D layer with a pool size of 2x2 and 'same' padding, and a Flatten layer to flatten the output. The model then includes two Dense layers, with the first having 64 units and ReLU activation, and the second with 18 units and softmax activation for multi-class classification.

The model is then compiled with the Adam optimizer at a learning rate of 0.0001, using 'sparse_categorical_crossentropy' as the loss function for multi-class classification, and 'acc' as the metric for accuracy. The model is trained using the fit method on the training data, with validation data provided to monitor performance during training. The batch size is set to 64, and the model is trained for 250 epochs with verbosity level 1 to display training progress.

```
model_cnn1 = Sequential()

model_cnn1.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
model_cnn1.add(MaxPooling2D((2, 2), padding='same'))
model_cnn1.add(Flatten())

model_cnn1.add(Dense(64, activation='relu'))
model_cnn1.add(Dense(10, activation='softmax'))

model_cnn1.summary()
```

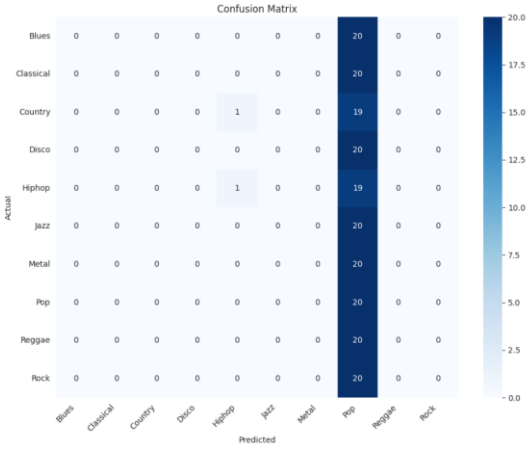
```
model_cnn1.compile(
    optimizer=Adam(learning_rate=0.0001), # can also use 'adam'
    loss='sparse_categorical_crossentropy', # loss for multi-class classification
    metrics=['acc']
)
```

```
hist_cnn1 = model_cnn1.fit(
    X_train_cnn, y_train,
    validation_data=(X_val_cnn, y_val),
    batch_size=64,
    epochs=250,
    verbose=1
)
```

We can tell from the test accuracy and classification report that this 1-block CNN with original data input's performance is not good at all. With the same dataset, we consider increasing the CNN's complexity to improve the performance.

```
loss_cnn1, acc_cnn1 = model_cnn1.evaluate(X_test_cnn, y_test)
print(f"Test Loss: {loss_cnn1}")
print(f"Test Accuracy: {acc_cnn1}")
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 0.0868 - loss: 2.2993
Test Loss: 2.2998571395874023
Test Accuracy: 0.10499999672174454
```



Classification Report:

	precision	recall	f1-score	support
Blues	0.00	0.00	0.00	20
Classical	0.00	0.00	0.00	20
Country	0.00	0.00	0.00	20
Disco	0.00	0.00	0.00	20
Hiphop	0.50	0.05	0.09	20
Jazz	0.00	0.00	0.00	20
Metal	0.00	0.00	0.00	20
Pop	0.10	1.00	0.18	20
Reggae	0.00	0.00	0.00	20
Rock	0.00	0.00	0.00	20
accuracy			0.10	200
macro avg	0.06	0.11	0.03	200
weighted avg	0.06	0.10	0.03	200

3.2.5 3-block CNN Model

We build the second model which is a 3-block CNN. The CNN architecture consists of three Conv2D layers with ReLU activation functions. The first layer has 32 filters, a kernel size of 3x3, and takes the input shape of the data. It is followed by a MaxPooling2D layer with a pool size of 3x3 and 'same' padding to reduce the spatial dimensions of the feature maps. The second Conv2D layer has 64 filters, also with a 3x3 kernel size, and is followed by another MaxPooling2D layer with a pool size of 3x3 and 'same' padding. The third Conv2D layer has 64 filters, and a 2x2 kernel size, and is followed by a MaxPooling2D layer with a pool size of 2x2 and 'same' padding. After the convolutional and pooling layers, a Flatten layer is added to transform the three-dimensional output to a one-dimensional array. This is followed by two Dense layers; the first Dense layer has 64 units with ReLU activation, and the second Dense layer has 10 units with softmax activation to output probabilities for 10 different classes.

The model is then compiled with the same parameters; we can tell from the result that this time, the test accuracy is 0.5 which is significantly improved compared to the first model but still needs some improvement. Also, the confusion matrix and the classification can provide a more intuitive result of this model's s preface

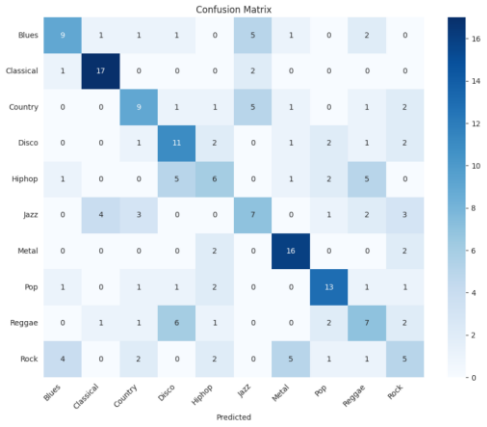
```
model_cnn2 = Sequential()

model_cnn2.add(Conv2D(32, 3, activation='relu', input_shape=input_shape))
model_cnn2.add(MaxPooling2D(3, strides=(2,2), padding='same'))


model_cnn2.add(Conv2D(64, 3, activation='relu'))
model_cnn2.add(MaxPooling2D(3, strides=(2,2), padding='same'))

model_cnn2.add(Conv2D(64, 2, activation='relu'))
model_cnn2.add(MaxPooling2D(2, strides=(2,2), padding='same'))

model_cnn2.add(Flatten())
model_cnn2.add(Dense(64, activation='relu'))

# output to 10 classes for predictions
model_cnn2.add(Dense(10, activation='softmax'))
model_cnn1.summary()
```

```
model_cnn2.compile(
    optimizer=Adam(learning_rate=0.0001), # can also use 'adam'
    loss='sparse_categorical_crossentropy', # loss for multi-class classification
    metrics=['acc']
)
```

```
hist_cnn2 = model_cnn2.fit(
    X_train_cnn, y_train,
    validation_data=(X_val_cnn, y_val),
    batch_size=64,
    epochs=250,
    verbose=1
)
```

```
loss_cnn2, acc_cnn2 = model_cnn2.evaluate(X_test_cnn, y_test)
print(f"Test Loss: {loss_cnn2}")
print(f"Test Accuracy: {acc_cnn2}")
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 126ms/step - acc: 0.4628 - loss: 2.7453
Test Loss: 2.5696022510528564
Test Accuracy: 0.5
```



Classification Report:

	precision	recall	f1-score	support
Blues	0.56	0.45	0.50	20
Classical	0.74	0.85	0.79	20
Country	0.50	0.45	0.47	20
Disco	0.44	0.55	0.49	20
Hiphop	0.38	0.30	0.33	20
Jazz	0.37	0.35	0.36	20
Metal	0.64	0.80	0.71	20
Pop	0.62	0.65	0.63	20
Reggae	0.35	0.35	0.35	20
Rock	0.29	0.25	0.27	20
accuracy			0.50	200
macro avg	0.49	0.50	0.49	200
weighted avg	0.49	0.50	0.49	200

3.2.6 3-block CNN Model with Regulation

The model_cnn3 is an enhanced version of the model_cnn2, featuring a more complex architecture that includes an additional convolutional block, which consists of a Conv2D layer with 64 filters and a kernel size of 2x2, followed by Batch Normalization, MaxPooling2D, and Dropout layers. This extra block allows model_cnn3 to capture finer details in the input data. Moreover, model_cnn3 includes more Dropout layers—two at a rate of 0.2 and one at 0.5—which helps to mitigate overfitting by randomly disabling a fraction of neurons during training. This model also has a denser first dense layer with 128 units compared to the 64 units in model_cnn2, potentially enabling it to learn more complex patterns. The final dense layer in model_cnn3 is designed to output predictions for 10 classes, similar to model_cnn2, but the overall architecture is deeper and wider, what's more, it offers more regularization through additional Batch Normalization and Dropout layers. These enhancements are

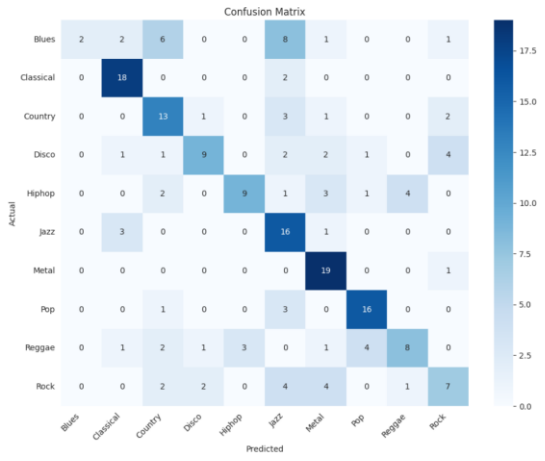
expected to improve the model's ability to generalize new data and reduce overfitting, ultimately leading to better performance in multi-class classification tasks.

We can see from the result that now the test accuracy is around 0.59 which has improved compared to the second model.

```
model_cnn3 = Sequential()

# Create a convolution block
model_cnn3.add(Conv2D(32, 3, activation='relu', input_shape=input_shape)) # first hidden conv layer
model_cnn3.add(BatchNormalization())
model_cnn3.add(MaxPooling2D(3, strides=(2,2), padding='same')) # MaxPool the results
model_cnn3.add(Dropout(0.2))

# Add another conv block
model_cnn3.add(Conv2D(64, 3, activation='relu'))
model_cnn3.add(BatchNormalization())
model_cnn3.add(MaxPooling2D(3, strides=(2,2), padding='same'))
model_cnn3.add(Dropout(0.1))

# Add another conv block
model_cnn3.add(Conv2D(64, 2, activation='relu'))
model_cnn3.add(BatchNormalization())
model_cnn3.add(MaxPooling2D(2, strides=(2,2), padding='same'))
model_cnn3.add(Dropout(0.1))

# Flatten output to send through dense layers
model_cnn3.add(Flatten())
model_cnn3.add(Dense(128, activation='relu'))
model_cnn3.add(Dropout(0.5))

# output to 10 classes for predictions
model_cnn3.add(Dense(10, activation='softmax')) # Softmax activation for multi-class classification
model_cnn3.summary()
```

```
loss_cnn3, acc_cnn3 = model_cnn3.evaluate(X_test_cnn, y_test)
print(f"Test Loss: {loss_cnn3}")
print(f"Test Accuracy: {acc_cnn3}")
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 121ms/step - acc: 0.5651 - loss: 2.8579
Test Loss: 2.7483718395233154
Test Accuracy: 0.5849999785423279
```



Classification Report:

	precision	recall	f1-score	support
Blues	1.00	0.10	0.18	20
Classical	0.72	0.90	0.80	20
Country	0.48	0.65	0.55	20
Disco	0.69	0.45	0.55	20
Hiphop	0.75	0.45	0.56	20
Jazz	0.41	0.80	0.54	20
Metal	0.59	0.95	0.73	20
Pop	0.73	0.80	0.76	20
Reggae	0.62	0.40	0.48	20
Rock	0.47	0.35	0.40	20
accuracy			0.58	200
macro avg	0.65	0.58	0.56	200
weighted avg	0.65	0.58	0.56	200

3.3 Genre classification with split data

3.3.1 Import libraries

First, import all the necessary libraries, including those for building CNN models, visualizing audio, transforming data functions, etc. Just like the experiment using the original data.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
import sklearn.metrics as skm
import sklearn.model_selection as skms
import sklearn.preprocessing as skp
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import random
import librosa, IPython
import librosa.display as lplt
import os, sys, cv2
import numpy as np
import pandas as pd
import IPython.display as ipd
from tqdm import tqdm
import tensorflow as tf
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout, Flatten, LSTM
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Conv1D, MaxPooling1D, MaxPool1D, GaussianNoise, GlobalMaxPooling1D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
from sklearn.model_selection import KFold
```

3.3.2 Extract features

Then we use the same function get_mfccs which is designed to extract Mel-Frequency Cepstral Coefficients (MFCCs) from audio files stored in a specified directory. But this time we split the original 30-second audio file into 3-second segments as the neural network training requires a large number of samples and this segmentation effectively increases the number of samples we can input into the network. Thus we may have a higher accuracy.

We reset the parameters set including 13 MFCCs to be extracted, an FFT window size of 2048, a hop length of 512, and the decision that each audio track will be divided into 10 segments for analysis. The directory path specified points to a dataset containing various music genres. The results show that the extracted MFCCs have a shape of (9986, 130, 13). This indicates that there are 9986 samples in total, which is much more than the size of the original samples.

```
n_mfcc = 13
n_fft = 2048
hop_length = 512
num_segments = 10
track_duration = 30

directory_path = '/kaggle/input/gtzan-dataset-music-genre-classification/Data/genres_original'

mfccs, genres, genre_nums = get_mfccs(directory_path,
                                      fs=22050,
                                      duration=track_duration,
                                      n_fft=n_fft,
                                      hop_length=hop_length,
                                      n_mfcc=n_mfcc,
                                      num_segments=num_segments)

# Print the shapes of the extracted data
print(f"MFCCs shape: {mfccs.shape}")
print(f"Genres shape: {genres.shape}")
print(f"Genre numbers shape: {genre_nums.shape}")
```

```
MFCCs shape: (9986, 130, 13)
Genres shape: (9986,)
Genre numbers shape: (9986,)
```

3.3.3 Split Train, Test Sets

Just like the former experiment, we split the train test sat and added additional dimension for CNN using the same parameter to make sure that the dataset size is the only change.

```
# Train-validation-test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42, stratify=y_train)

print(f"X training data shape: {X_train.shape}, y training data shape: {y_train.shape}")
print(f"X validation data shape: {X_val.shape}, y validation data shape: {y_val.shape}")
```

```
X training data shape: (6390, 130, 13), y training data shape: (6390,)
X validation data shape: (1598, 130, 13), y validation data shape: (1598,)
```

```
# Add additional dimension for CNN
X_train_cnn = X_train[..., np.newaxis]
X_val_cnn = X_val[..., np.newaxis]
X_test_cnn = X_test[..., np.newaxis]

input_shape = X_train_cnn.shape[1:4]
X_train_cnn.shape
```

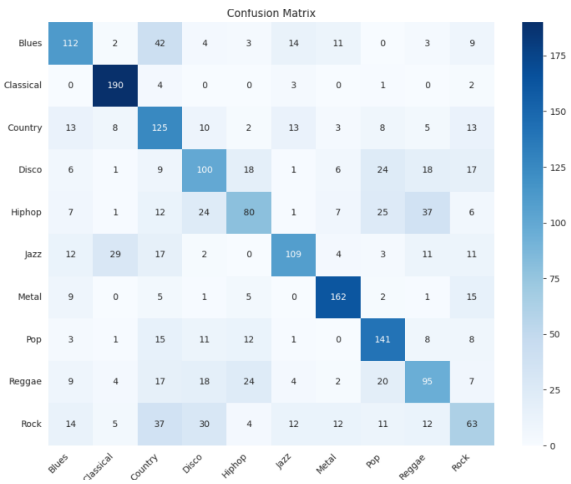
```
(6390, 130, 13, 1)
```

3.3.4 1-block CNN Model

We used the same 1-block CNN model and found that this time the test accuracy was 0.589 which is much better than the result using unsliced data.

```
loss_cnn1, acc_cnn1 = model_cnn1.evaluate(X_test_cnn, y_test)
print(f"Test Loss: {loss_cnn1}")
print(f"Test Accuracy: {acc_cnn1}")
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - acc: 0.5912 - loss: 3.5778
Test Loss: 3.6451141834259033
Test Accuracy: 0.5890890955924988
```




```
Classification Report:

              precision    recall  f1-score   support

       Blues       0.61      0.56      0.58       200
   Classical       0.79      0.95      0.86       200
     Country       0.44      0.62      0.52       200
       Disco       0.50      0.50      0.50       200
      Hiphop       0.54      0.40      0.46       200
        Jazz       0.69      0.55      0.61       198
       Metal       0.78      0.81      0.80       200
         Pop       0.60      0.70      0.65       200
      Reggae       0.50      0.47      0.49       200
        Rock       0.42      0.32      0.36       200

    accuracy                           0.59      1998
   macro avg       0.59      0.59      0.58      1998
weighted avg       0.59      0.59      0.58      1998
```

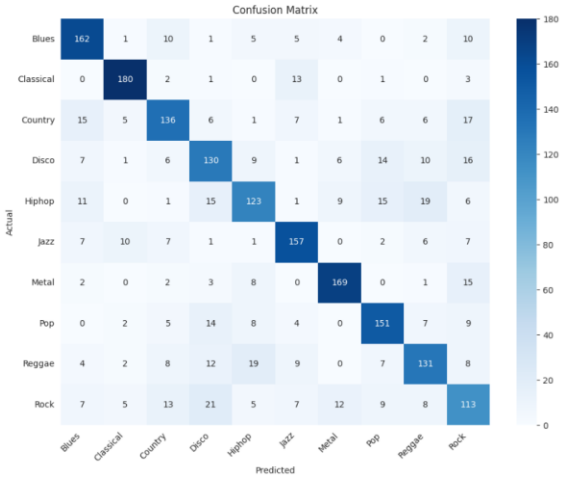
3.3.5 3-block CNN Model

Then we used the second 3-block CNN model to continue and found that this time with a

more complex CNN model the test accuracy reached 0.726 which is the highest accuracy so far.

```
loss_cnn2, acc_cnn2 = model_cnn2.evaluate(X_test_cnn, y_test)
print(f"Test Loss: {loss_cnn2}")
print(f"Test Accuracy: {acc_cnn2}")
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - acc: 0.7232 - loss: 1.8641
Test Loss: 1.8517080545425415
Test Accuracy: 0.7267267107963562
```



Classification Report:

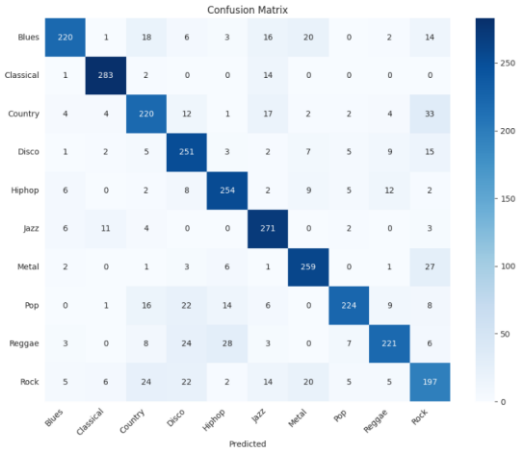
	precision	recall	f1-score	support
Blues	0.75	0.81	0.78	200
Classical	0.87	0.90	0.89	200
Country	0.72	0.68	0.70	200
Disco	0.64	0.65	0.64	200
Hiphop	0.69	0.61	0.65	200
Jazz	0.77	0.79	0.78	198
Metal	0.84	0.84	0.84	200
Pop	0.74	0.76	0.75	200
Reggae	0.69	0.66	0.67	200
Rock	0.55	0.56	0.56	200
accuracy			0.73	1998
macro avg	0.73	0.73	0.73	1998
weighted avg	0.73	0.73	0.73	1998

3.3.6 3-block CNN Model with Regulation

Lastly, we use the 3-block CNN Model with Regulation, with the usage of Batch Normalization and Dropout, the stability and generalization of the model are improved and the test accuracy is 0.805, which is a satisfying result, showing that relatively complex CNN model can perform well with the properly processed data.

```
loss_cnn3, acc_cnn3 = model_cnn3.evaluate(X_test_cnn, y_test)
print(f"Test Loss: {loss_cnn3}")
print(f"Test Accuracy: {acc_cnn3}")
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - acc: 0.8010 - loss: 0.8573
Test Loss: 0.8287850618362427
Test Accuracy: 0.8053053021430969
```



Classification Report:

	precision	recall	f1-score	support
Blues	0.89	0.73	0.80	300
Classical	0.92	0.94	0.93	300
Country	0.73	0.74	0.73	299
Disco	0.72	0.84	0.77	300
Hiphop	0.82	0.85	0.83	300
Jazz	0.78	0.91	0.84	297
Metal	0.82	0.86	0.84	300
Pop	0.90	0.75	0.81	300
Reggae	0.84	0.74	0.79	300
Rock	0.65	0.66	0.65	300
accuracy			0.80	2996
macro avg	0.81	0.80	0.80	2996
weighted avg	0.81	0.80	0.80	2996

3.4 Summary

The table below summarizes the performance of different models with various data inputs. This experiment demonstrates that increasing the complexity of the CNN model and using split data (3-second segments) significantly improves the accuracy of music genre classification. The most effective model was the 3-block CNN with regularization techniques applied to split data, achieving a test accuracy of 0.805. This suggests that a combination of advanced model architecture and effective data preprocessing can lead to better performance in music genre classification tasks.

Model Type	Data Input	Test Accuracy
1-block CNN Model	Original	Not Satisfactory
3-block CNN Model	Original	0.5
3-block CNN Model with Regulation	Original	0.59
1-block CNN Model	Split	0.589
3-block CNN Model	Split	0.726
3-block CNN Model with Regulation	Split	0.805

4. Conclusion

In conclusion, the project successfully developed a deep-learning model for music genre classification using the GTZAN dataset. The study found that more complex CNN models, combined with split data input, yield better classification accuracy. The 3-block CNN model with regularization techniques applied to split data performed the best, indicating that such an approach can effectively capture the nuances of different music genres. Future work could explore even more sophisticated models or additional feature extraction methods to further enhance classification performance.

5. Environment

The notebook runs directly on Kaggle platform.