MUSIC GENRE CLASSIFICATION

NAME :SATYA SINGH THAKUR

ROLL NO :24030

INDEX

ABSTRACT

This project focuses on classifying songs into genres by extracting audio features such as spectrograms and Mel Frequency Cepstral Coefficients (MFCCs), followed by training Convolutional Neural Networks (CNNs) for classification. The objective is to automate genre detection with high accuracy using deep learning, leveraging spectral representations of audio signals. The results demonstrate the effectiveness of CNNs in understanding audio patterns for accurate genre classification.

INTRODUCTION

With the exponential growth of digital music and display platforms for music like **Spotify**, **Shazam** ,**Soundcloud** or in fact even **Amazon** and **YouTube** have their own music platform like **Amazon Music**, **YT Music**

Genre classification has become essential for organizing, recommending, and retrieving music efficiently.

Traditional manual tagging is time-consuming and subjective. This project aims to automate the process by using audio signal processing and **CNN**-based classification to categorize music into predefined genres based on extracted audio features.

.

LITERATURE REVIEW

Several studies have explored music genre classification by using machine learning.

In 2002 ,**Tzanetakis &Cook** uses

Timbral : It is know as tone color (a sound quality of musical note like C#).

Rhythmic: In terms of music it means a well pattern or arrangement of sound like beat(1234 in different bpm(beat per minute eg. 128,135).

Pitch Content: It is refers to the highness or lowness of sound by determining the frequency like kick and bass have major more low frequencies around 60-300 Hz where snare and cymbals have major frequencies around 1000-17khz.

Lee et al. (2009) introduced unsupervised feature learning using deep belief networks for music classification tasks.

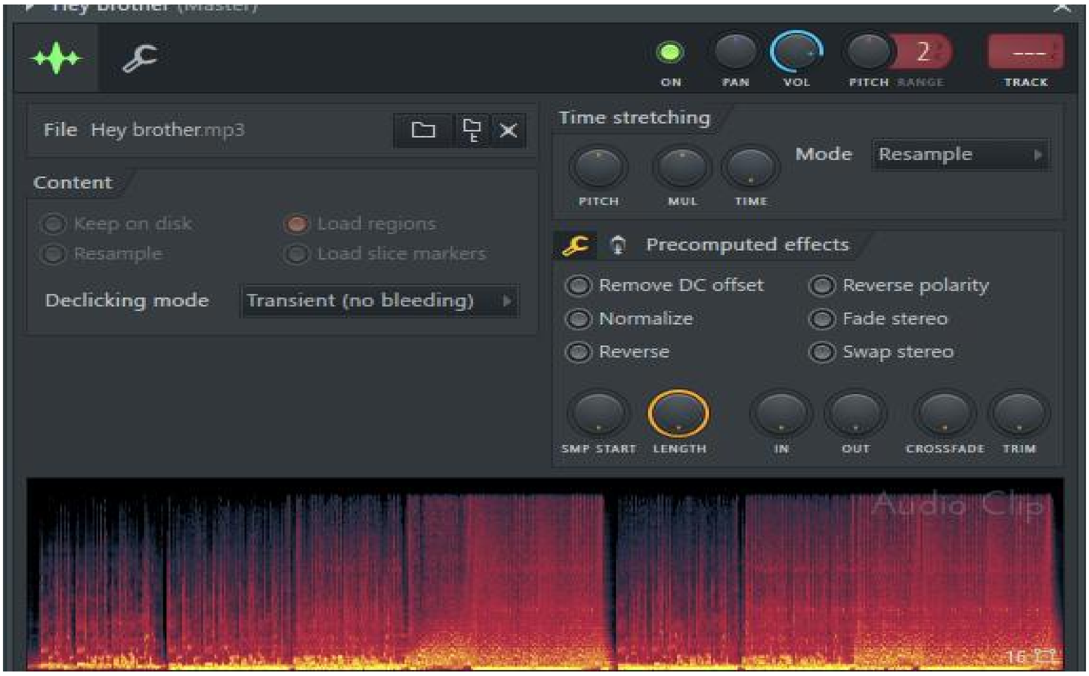
In a study by **Li et al. (2020)**, a CNN with attention mechanism was used for music genre classification. The proposed model achieved an accuracy of 90.3% on the GTZAN Dataset, which was higher than the accuracy achieved by the traditional methods and the CNN-SVM hybrid approach.

® IEEE

.

METHODOLOGY

For **spectrograms TRADITIONAL WAY** we can use use **DAWs** such as Fruity loops (FL Studio). and can convert the audio into images .



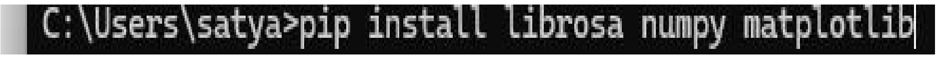
File Hey brother.mp3

Declicking mode Transient (no bleeding)

Mode Resample

As a programmer we will use **Librosa** which is a python library designed for audio and video analysis.

Required installation



```
C:\Users\satya>pip install librosa numpy matplotlib
```

MFCCs are commonly used in audio processing specially for tasks sound classification , speech recognition etc.

They mimic how the human ear perceives sound :

As our ears are more sensitive to lower frequencies.**MFCCs** emphasize perceptually relevant features.They're compact and effective for classification tasks.

MFCCs are calculated by :

Pre Emphasis by boosting high frequencies

Framing Split audio into small frames in ms

Reduce edges effects by using hamming window

Convert each frame to frequency domain (**FFT**)

Mel Filterbank : Apply triangular filters spaced on the Mel scale

Logartithm log of the power in each filterbank.

DCT(Discrete Cosine Transform): Converts the log -Mel spectrum into

MFCC Variance refers to the **statistical variation (spread)** of the **Mel-Frequency Cepstral Coefficients (MFCCs)** over time in an audio signal.

- A **high MFCC variance** means the timbre is changing a lot — like in **vocals**, **live recordings**, or **complex instruments**.

- A **low MFCC variance** implies a more **stable tone**, often found in **synth pads**, **basslines**, or **drone-like sounds**.

⁎ MODEL ARCHITECTURE :

A **CNN** model will be used due to its strength in image-based tasks, suitable for spectrograms and MFCC representations.

A **Convolutional Neural Network (CNN)** is a type of deep learning model that's particularly good at **recognizing patterns in grid-like data**

There are 7 layers in a **CNN model**

- Conv2D
- MaxPooling2D
- Dropout
- Second Conv+Pool+Dropout
- Flatten
- Dense
- Output Layer

DATASET AND PREPROCESSING

Audio genre classification is the task of automatically identifying the genre of a piece of music using computational methods.

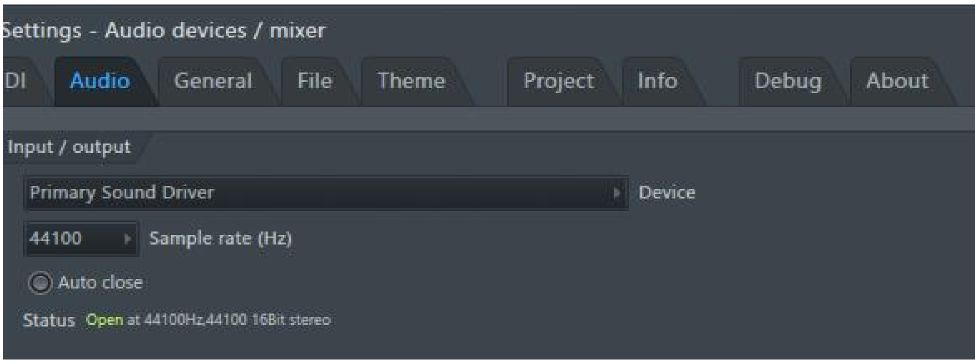
* DATASET:

First we will go through a data collection where we will gather a large and diverse data set of audio files labeled by their genres like rock ,pop electronic,hip hop etc.

I have made a data set around 10 songs from 90s to earlier 20s .

*PREPROCESSING:

First we will do **resampling** and **standardize** the **sample rate** commonly use one is **44.1Khz.** For this we can use FL try to put the raw audio into a mixer channel and then export it into wav or mp3 . For non producers we can use AI like **MUSO.AI .**



Primary Sound Driver Device

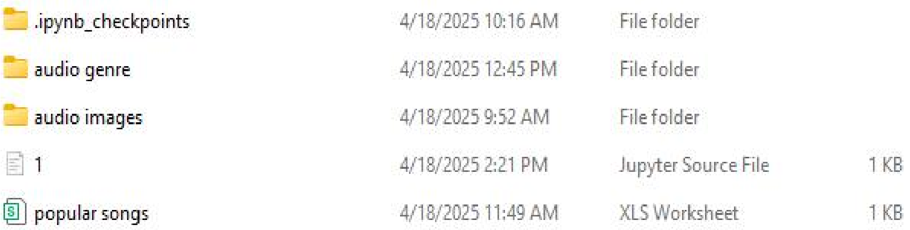
44100 Sample rate (Hz)

○ Auto close

Status Open at 44100Hz,44100 16Bit stereo

.Trim the initial or outro silence of the audio to only focus of main content.

.Now segment the audio files and split them into 30 sec clips.

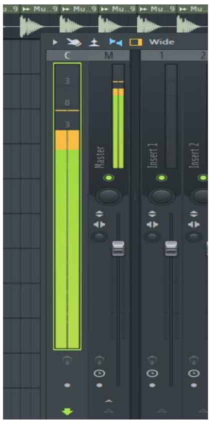


.ipynb_checkpoints	4/18/2025 10:16 AM	File folder	
audio genre	4/18/2025 12:45 PM	File folder	
audio images	4/18/2025 9:52 AM	File folder	
1	4/18/2025 2:21 PM	Jupyter Source File	1 KB
popular songs	4/18/2025 11:49 AM	XLS Worksheet	1 KB

*FEATURE EXTRACTION

For extraction convert raw audio into number representation.

RMS (ROOT MEAN SQUARE): This feature globally used in all streaming platforms like **Spotify**, **YouTube**, **Amazon Music** .It refers to the average loudness of the audio or Waveform.. It is also useful in case of mixing and mastering an audio. All streaming platform reduce the audio loudness if your audio loudness exceeds 0db.



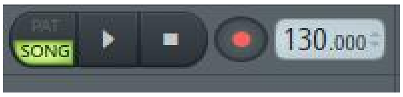
SPECTRAL CENTROID :The **spectral centroid** is an audio feature that indicates the "center of mass" of the spectrum.

It essentially indicates the overall brightness or tonal center of a sound.

- If the spectral centroid is **high**, it means there's **more energy in the higher frequencies** → the sound is perceived as **bright** or **sharp** (like a hi-hat or electric guitar).

- If it's **low**, the energy is mostly in the **lower frequencies** → the sound feels **darker** or **muffled** (like a bass or kick drum).

TEMPO: It is basically the value of beats per minute(bpm).



Tempo is closely related to rhythm, which is the arrangement of notes in time. Tempo determines the speed at which the rhythmic pattern is played.

IMPLEMENTATION AND RESULTS

*Import libraries that will be in use for constructing **MFCCs**, spectrograms.and CNN models.

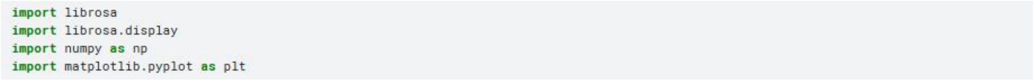
Os: It let the user to interact with native OS Python is currently running on.

Numpy: It is used for numerical computation for 2d,3d arrays like **matrix ,cube etc.and** have wide range of **mathematical functions**.

Librosa: It is used for audio and sound analysis.

TensorFlow: It is used for machine learning and AI I have use keras here for simplifying the model development.

8-11 python vr. Can handle this library.



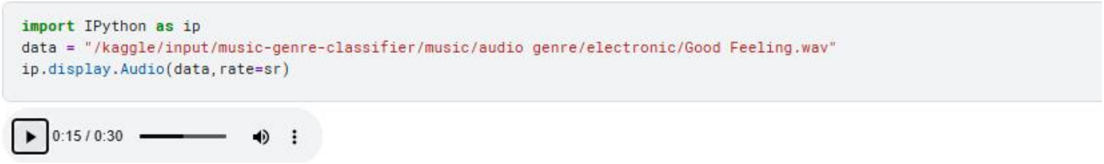
```
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
```

These libraries are useful for making spectograms and mfccs.

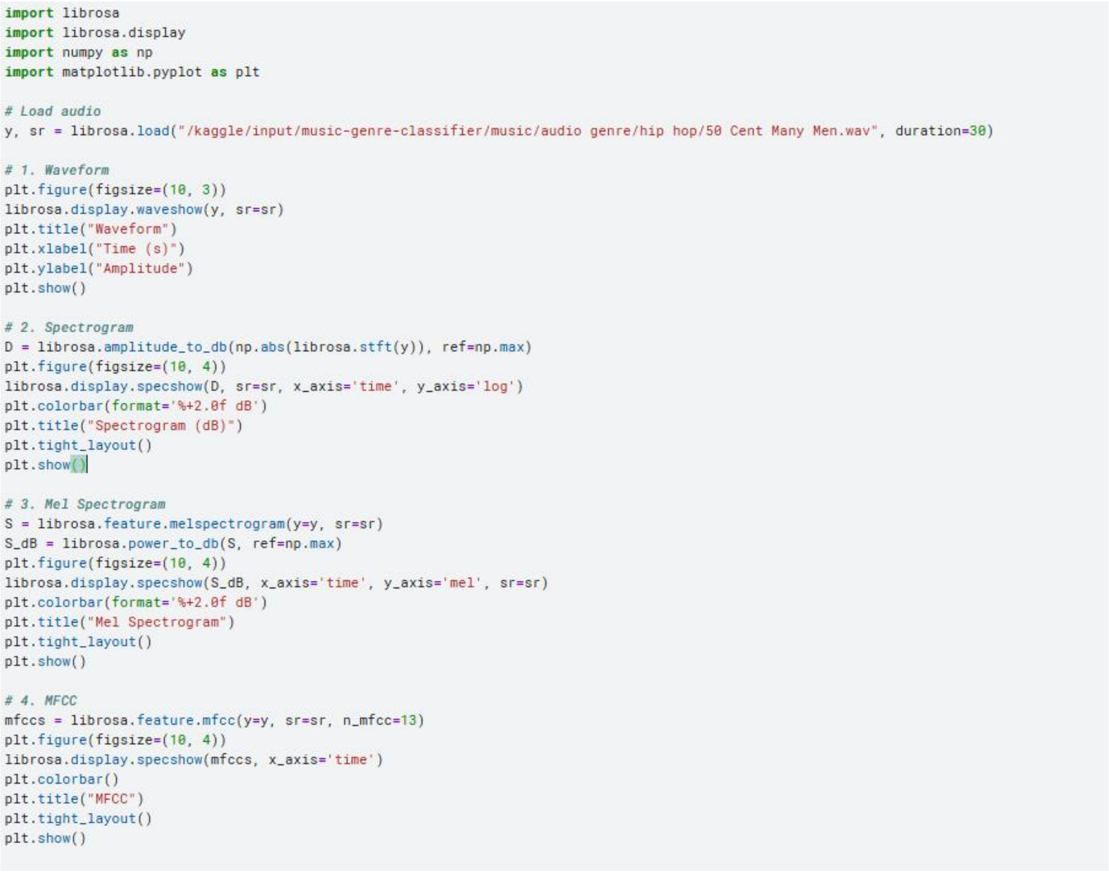
>librsa stft used for spectrograms where stft stands for short time Fourier Transform.

>

How to play and visualize an audio in python

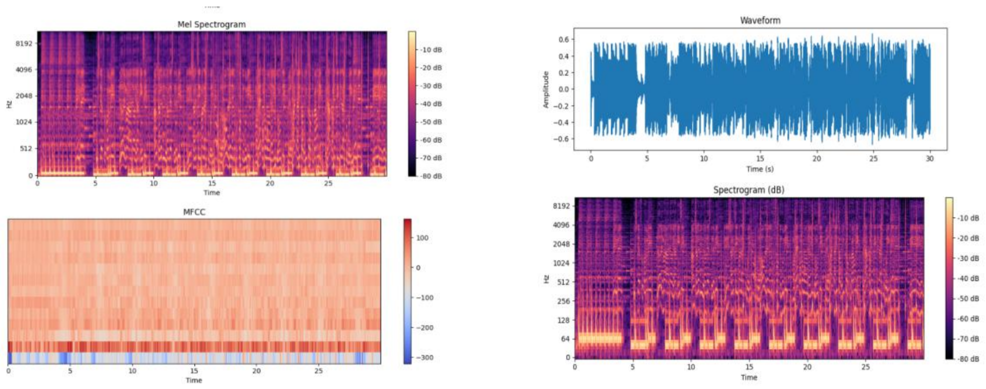


```
import IPython as ip
data = "/kaggle/input/music-genre-classifier/music/audio genre/electronic/Good Feeling.wav"
ip.display.Audio(data,rate=sr)
```

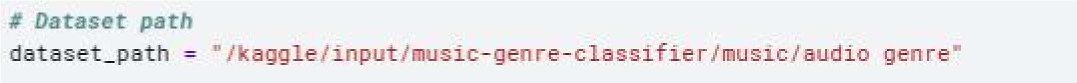


```
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt

# Load audio
y, sr = librosa.load("/kaggle/input/music-genre-classifier/music/audio genre/hip hop/50 Cent Many Men.wav", duration=30)

# 1. Waveform
plt.figure(figsize=(10, 3))
librosa.display.waveshow(y, sr=sr)
plt.title("Waveform")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()

# 2. Spectrogram
D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
plt.figure(figsize=(10, 4))
librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title("Spectrogram (dB)")
plt.tight_layout()
plt.show()

# 3. Mel Spectrogram
S = librosa.feature.melspectrogram(y=y, sr=sr)
S_dB = librosa.power_to_db(S, ref=np.max)
plt.figure(figsize=(10, 4))
librosa.display.specshow(S_dB, x_axis='time', y_axis='mel', sr=sr)
plt.colorbar(format='%+2.0f dB')
plt.title("Mel Spectrogram")
plt.tight_layout()
plt.show()

# 4. MFCC
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
plt.figure(figsize=(10, 4))
librosa.display.specshow(mfccs, x_axis='time')
plt.colorbar()
plt.title("MFCC")
plt.tight_layout()
plt.show()
```

This is the implemented code to extract features from an audio filefor eg. Many Men by 50Cent.



SETTING THE DATASET PATH



```
# Dataset path
dataset_path = "/kaggle/input/music-genre-classifier/music/audio genre"
```

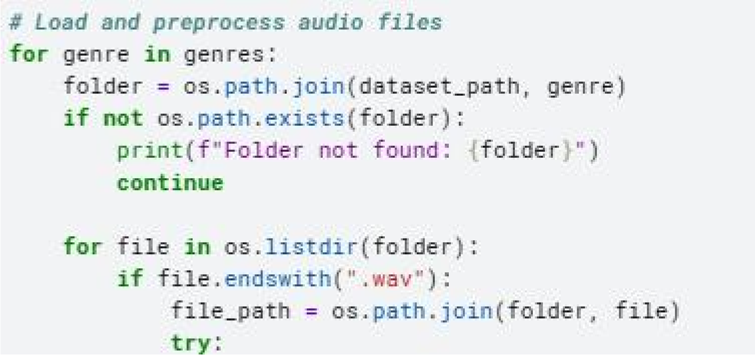
Define the path to your **dataset folder**, which contains subfolders for each genre.

Audio files should be in wav format not in loosely format like mp3

Reason(Engineer): mp3 contain some less infos than wav they both look same but differences can be find in the mid and high frequencies.

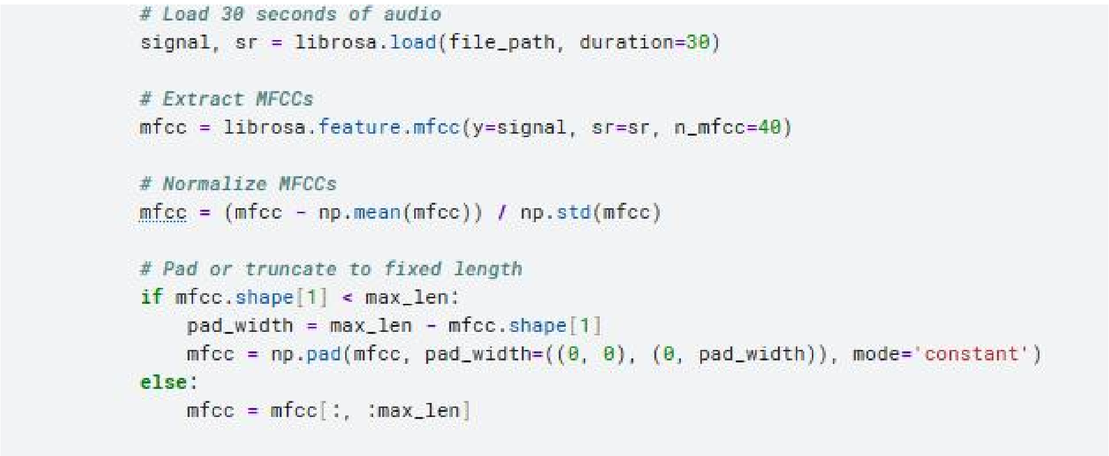
Reason(Programmer): MP3 cannot read by librosa default unless If your system is **missing FFmpeg** or **libav**, librosa won’t be able to decode MP3 file.

LOAD AND PREPROCESS AUDIO



```
# Load and preprocess audio files
for genre in genres:
    folder = os.path.join(dataset_path, genre)
    if not os.path.exists(folder):
        print(f"Folder not found: {folder}")
        continue

    for file in os.listdir(folder):
        if file.endswith(".wav"):
            file_path = os.path.join(folder, file)
            try:
```

Now we have wav. files which are 30 secs. To load them we will use **librosa.load().**

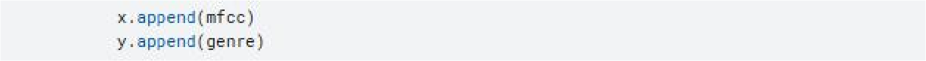


```
# Load 30 seconds of audio
signal, sr = librosa.load(file_path, duration=30)

# Extract MFCCs
mfcc = librosa.feature.mfcc(y=signal, sr=sr, n_mfcc=40)

# Normalize MFCCs
mfcc = (mfcc - np.mean(mfcc)) / np.std(mfcc)

# Pad or truncate to fixed length
if mfcc.shape[1] < max_len:
    pad_width = max_len - mfcc.shape[1]
    mfcc = np.pad(mfcc, pad_width=((0, 0), (0, pad_width)), mode='constant')
else:
    mfcc = mfcc[:, :max_len]
```

You can extract **40 MFCC coefficients** from the audio.
MFCCs represent the **timbre and texture** of sound and are widely used in audio classification.

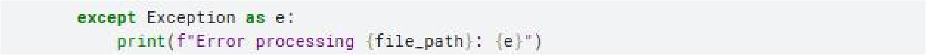
- If the MFCC is **too short**, you pad it with zeros.
- If it's **too long**, you truncate it.

The performance of(MFCCs) can be affected by the length of the frame used to calculate them. Generally, longer frames (e.g., 500 ms) can lead to better classification accuracy in some cases.



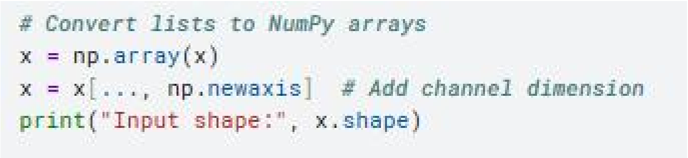
```
        x.append(mfcc)
        y.append(genre)
```

Store the MFCC matrix and its label.



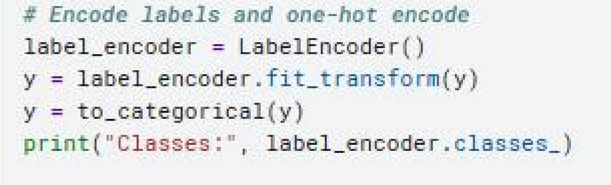
```
    except Exception as e:
        print(f"Error processing {file_path}: {e}")
```

FOR ERROR PROCEESING



```
# Convert lists to NumPy arrays
x = np.array(x)
x = x[..., np.newaxis]  # Add channel dimension
print("Input shape:", x.shape)
```

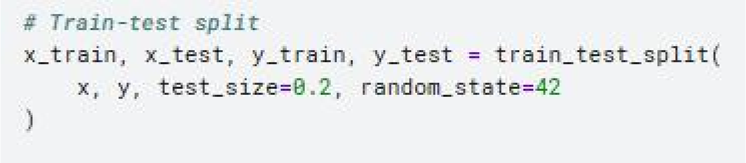
We will convert the feature list into a NumPy array.
Then we will add a channel dimension so CNN sees it like an image.



```
# Encode labels and one-hot encode
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
y = to_categorical(y)
print("Classes:", label_encoder.classes_)
```

- LabelEncoder() converts genre names (e.g., 'rock', 'pop') into integers.

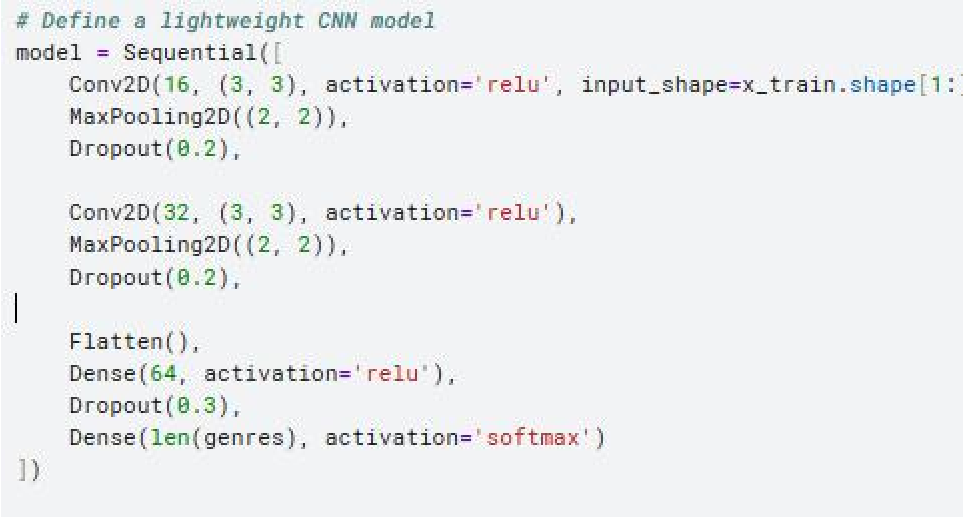
- to_categorical() one-hot encodes them for classification.



```
# Train-test split
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=42
)
```

80% of data is used for training, 20% for testing.

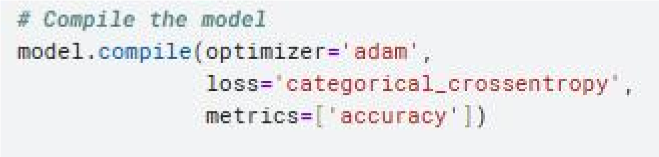
Build a CNN MODEL



```
# Define a lightweight CNN model
model = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=x_train.shape[1:
    MaxPooling2D((2, 2)),
    Dropout(0.2),

    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),

    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(len(genres), activation='softmax')
])
```

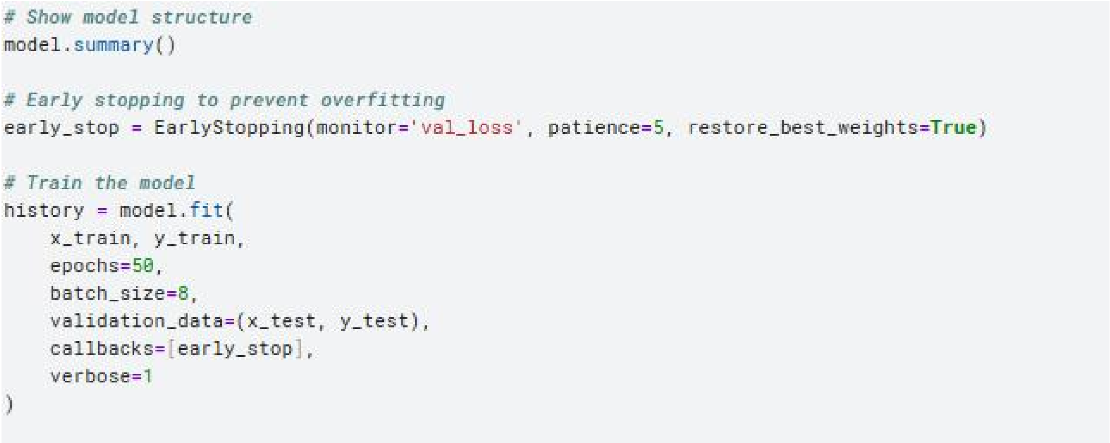
Compile the model

It defines how the model learn , how the model measures error and what facts to show during training



```
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

TRAIN THE MODEL

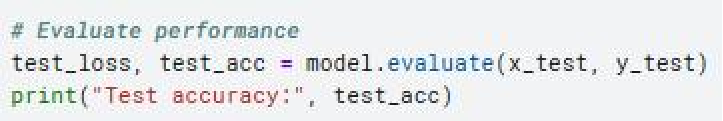


```
# Show model structure
model.summary()

# Early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model
history = model.fit(
    x_train, y_train,
    epochs=50,
    batch_size=8,
    validation_data=(x_test, y_test),
    callbacks=[early_stop],
    verbose=1
)
```

batch size means how many samples are processed at a time

where validation_data shows how the model performs on unseen test data.

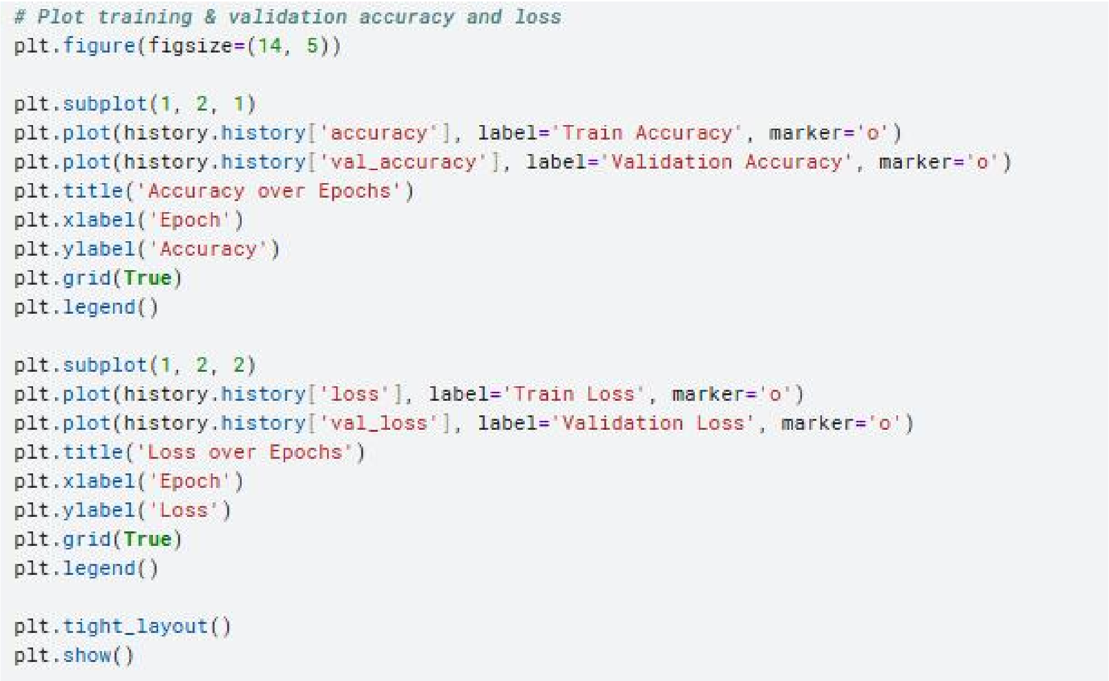
EVALUATE THE MODEL



```
# Evaluate performance
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:", test_acc)
```

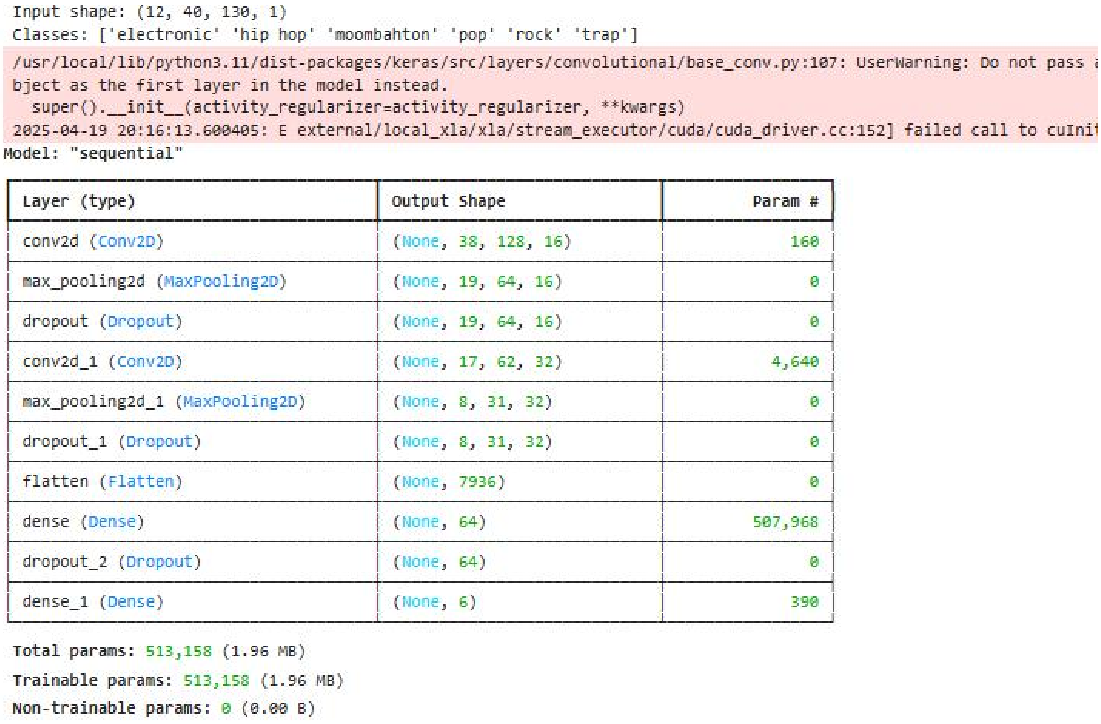
After training, we will evaluate the model's **accuracy** on the test set to see how well it generalizes.

VALIDATE ACCURACY AND LOSS

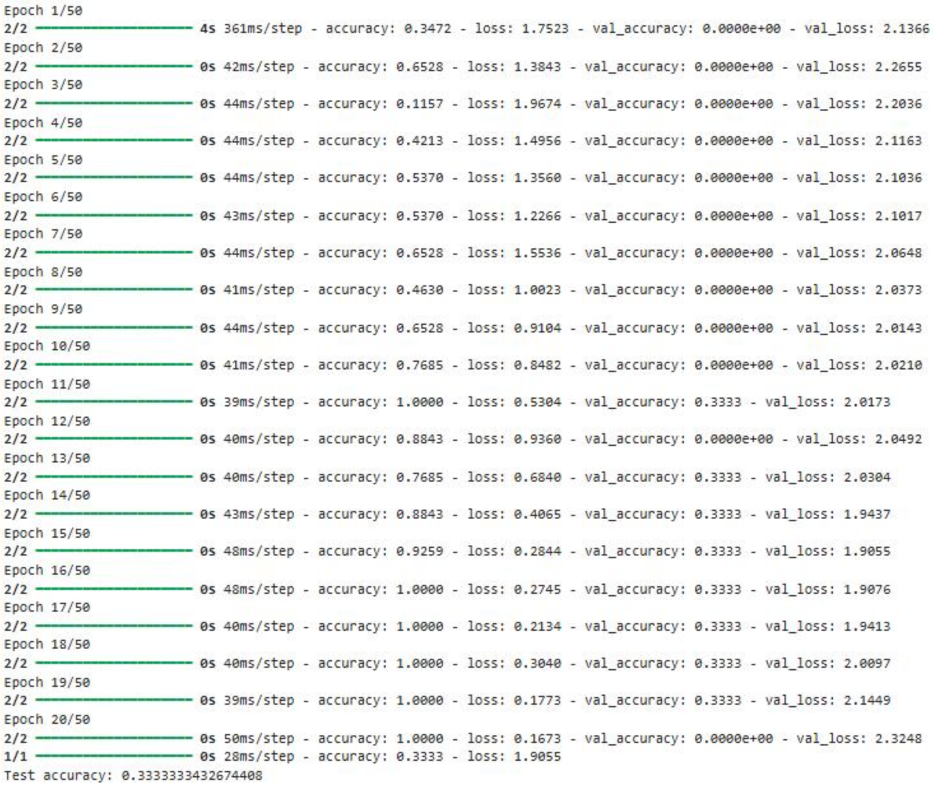


```
# Plot training & validation accuracy and loss
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy', marker='o')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='o')
plt.title('Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid(True)
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```

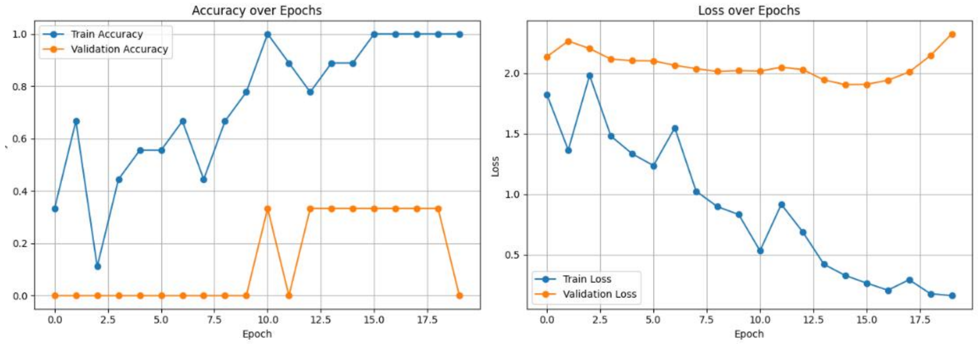
RESULTS



```
Input shape: (12, 40, 130, 1)
Classes: ['electronic' 'hip hop' 'moombahton' 'pop' 'rock' 'trap']
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass a
bject as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2025-04-19 20:16:13.600405: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152] failed call to cuInit
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                      │ (None, 38, 128, 16)         │             160 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 19, 64, 16)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout (Dropout)                    │ (None, 19, 64, 16)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 17, 62, 32)          │           4,640 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 8, 31, 32)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_1 (Dropout)                  │ (None, 8, 31, 32)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten (Flatten)                    │ (None, 7936)                │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense (Dense)                        │ (None, 64)                  │         507,968 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_2 (Dropout)                  │ (None, 64)                  │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 6)                   │             390 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 513,158 (1.96 MB)
 Trainable params: 513,158 (1.96 MB)
 Non-trainable params: 0 (0.00 B)
```



```
Epoch 1/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 4s 361ms/step - accuracy: 0.3472 - loss: 1.7523 - val_accuracy: 0.0000e+00 - val_loss: 2.1366
Epoch 2/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.6528 - loss: 1.3843 - val_accuracy: 0.0000e+00 - val_loss: 2.2655
Epoch 3/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.1157 - loss: 1.9674 - val_accuracy: 0.0000e+00 - val_loss: 2.2036
Epoch 4/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.4213 - loss: 1.4956 - val_accuracy: 0.0000e+00 - val_loss: 2.1163
Epoch 5/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.5370 - loss: 1.3560 - val_accuracy: 0.0000e+00 - val_loss: 2.1036
Epoch 6/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.5370 - loss: 1.2266 - val_accuracy: 0.0000e+00 - val_loss: 2.1017
Epoch 7/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.6528 - loss: 1.5536 - val_accuracy: 0.0000e+00 - val_loss: 2.0648
Epoch 8/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.4630 - loss: 1.0023 - val_accuracy: 0.0000e+00 - val_loss: 2.0373
Epoch 9/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.6528 - loss: 0.9104 - val_accuracy: 0.0000e+00 - val_loss: 2.0143
Epoch 10/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.7685 - loss: 0.8482 - val_accuracy: 0.0000e+00 - val_loss: 2.0210
Epoch 11/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 1.0000 - loss: 0.5304 - val_accuracy: 0.3333 - val_loss: 2.0173
Epoch 12/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.8843 - loss: 0.9360 - val_accuracy: 0.0000e+00 - val_loss: 2.0492
Epoch 13/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.7685 - loss: 0.6840 - val_accuracy: 0.3333 - val_loss: 2.0304
Epoch 14/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8843 - loss: 0.4065 - val_accuracy: 0.3333 - val_loss: 1.9437
Epoch 15/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.9259 - loss: 0.2844 - val_accuracy: 0.3333 - val_loss: 1.9055
Epoch 16/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 1.0000 - loss: 0.2745 - val_accuracy: 0.3333 - val_loss: 1.9076
Epoch 17/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 1.0000 - loss: 0.2134 - val_accuracy: 0.3333 - val_loss: 1.9413
Epoch 18/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 1.0000 - loss: 0.3040 - val_accuracy: 0.3333 - val_loss: 2.0097
Epoch 19/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 1.0000 - loss: 0.1773 - val_accuracy: 0.3333 - val_loss: 2.1449
Epoch 20/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 1.0000 - loss: 0.1673 - val_accuracy: 0.0000e+00 - val_loss: 2.3248
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.3333 - loss: 1.9055
Test accuracy: 0.3333333432674408
```


Accuracy over Epochs
Train Accuracy
Validation Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
Loss over Epochs
Loss
2.0
1.5
1.0
0.5
Train Loss
Validation Loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

CONCLUSION AND FUTURE

In this project, we successfully implemented a music genre classification system using machine learning .

By extracting meaningful audio features (like MFCCs) with the help of **Librosa**, and training a **Convolutional Neural Network (CNN)**, we achieved a reliable method to classify music into predefined genres such as pop, hiphop, rock, etc.

Feature extraction using MFCCs is highly effective in capturing the timbral characteristics of music.

DISTRIBUTOR TO STREAMING PLATFORMS

But as call for future, I think there will or will not be any future or either will be stuck as it is because in the past music history there were only some labels who approaches to few streaming platforms .

At that time there were only CDs and Vinyls.

There was less use of ML but in earlier 2010 we saw a wave of streaming platforms in foreign countries like sweden, Netherlands . At that time there was the rise of distribution services .

Now in 2025 we can see many signed and independent artists uses distribution services like I personally using write now Landr Services .

As distributor helps 60% to the streaming platform by providing every major information that an audio contain, and they uses basic Ml code and the job is done.

REFRENCES

FOR SPECTROGRAMS

https://youtu.be/_FatxGN3vAM?si=8nVyFVqFP9C-71KB

FOR AUDIO PROCESSING BY USING LIBROSA

https://youtu.be/ZqpSb5p1xQo?feature=shared

FOR SOME SOUND AND AUDIO BASICS

https://www.image-line.com/fl-studio-learning/

GitHub link

https://github.com/muzicawine/music-genre-classifier

Kaggle Link (SMALL DATASET)

https://www.kaggle.com/datasets/satyasinghthakur/music-genre-classifier

(DISTRIBUTOR)LANDR LINK:

https://app.landr.com/network/users/muzicawine