Dataset Link: https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification/data

Importing Libraries

```
# Specify the path to the GTZAN dataset
from google.colab import drive
drive.mount('/content/drive')

# Define your dataset path (update based on your structure)
dataset_path = '/content/drive/MyDrive/Data 2/genres_original/'  # Update this pa

# List of genres in the dataset
genres = 'blues classical country disco hiphop jazz metal pop reggae rock'.split(
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
import os
import librosa
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPool2D, Flatten, Dense,Drop
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.image import resize
import matplotlib.pyplot as plt
import seaborn as sns
```

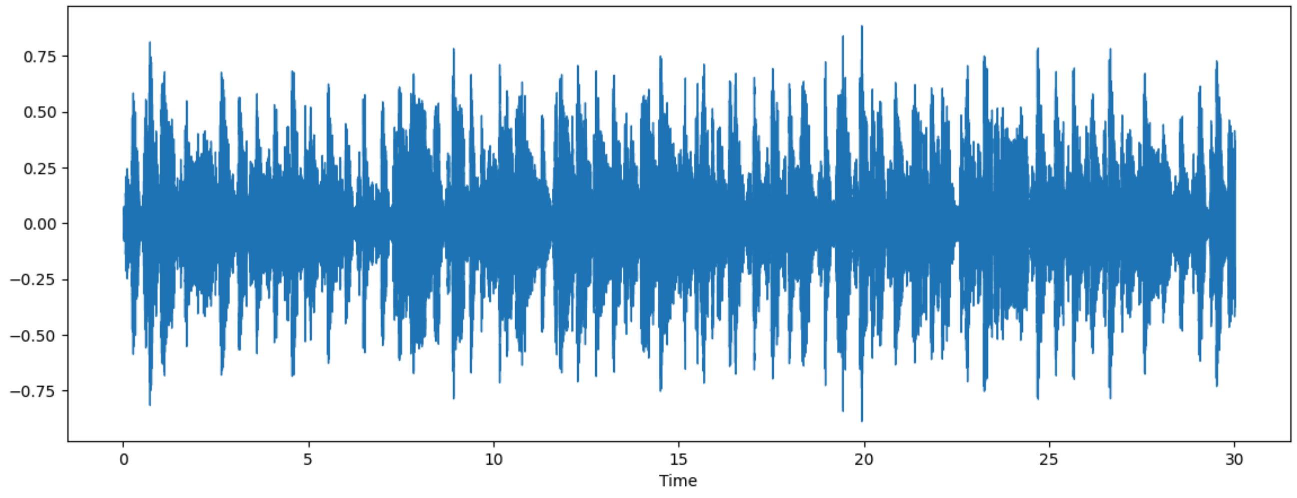
Viewing Dataset & preprocessing stage

Visualizing Single Audio

```
random_file_name = "/content/drive/MyDrive/Data 2/genres_original/blues/blues.000
```

```
x, sr = librosa.load(random_file_name, sr=44100)
plt.figure(figsize=(14, 5))
librosa.display.waveshow(x, sr=sr)
```

```
<librosa.display.AdaptiveWaveplot at 0x7d4646370c40>
```



```
!ls /content/drive/MyDrive/GTZAN\ dataset/Data\ 2/genres_original/blues
```

```
ls: cannot access '/content/drive/MyDrive/GTZAN dataset/Data 2/genres_origina
```

```
import os
print(os.path.exists(random_file_name))  # This should return True if the file ex
```

```
True
```

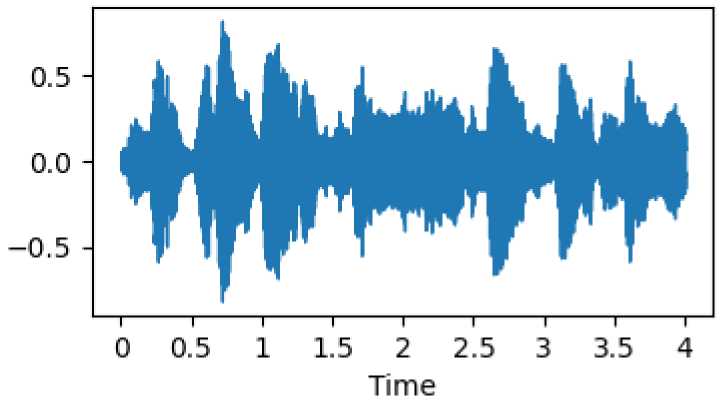
Playing Sound

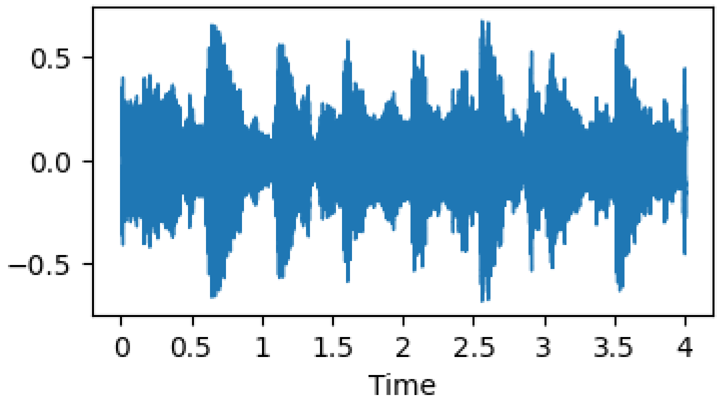
```
from IPython.display import Audio
Audio(data=x, rate=sr)
```

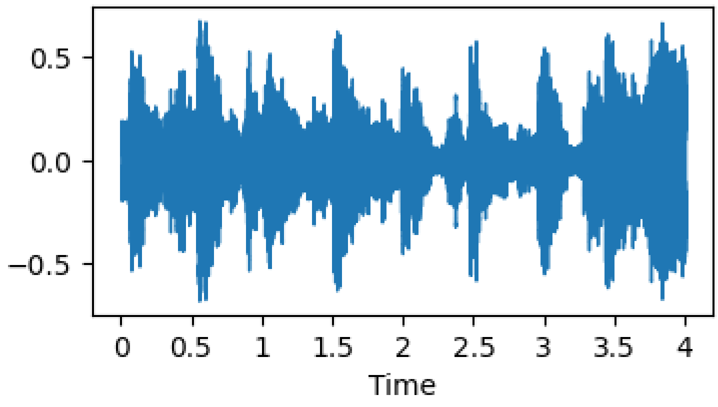
0:00 / 0:30

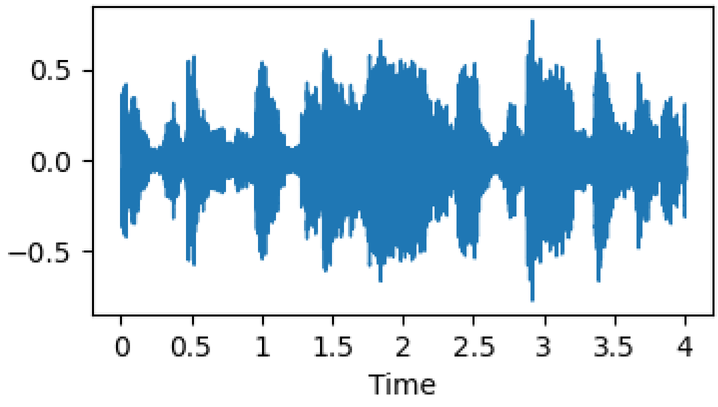
Doing Visualization on chunks of audio

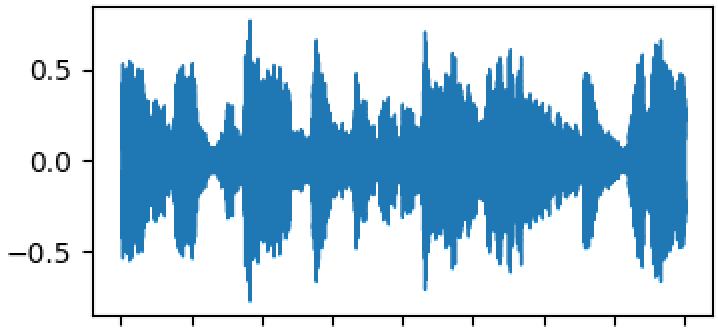
```
audio_path = "/content/drive/MyDrive/Data 2/genres_original/blues/blues.00000.wav
y, sr = librosa.load(audio_path, sr=None)  # sr=None to keep the original samplin

# Define the duration of each chunk and overlap
chunk_duration = 4  # seconds
overlap_duration = 2  # seconds

# Convert durations to samples
chunk_samples = chunk_duration * sr
overlap_samples = overlap_duration * sr

# Calculate the number of chunks
num_chunks = int(np.ceil((len(y) - chunk_samples) / (chunk_samples - overlap_samp

# Iterate over each chunk
for i in range(num_chunks):
    # Calculate start and end indices of the chunk
    start = i * (chunk_samples - overlap_samples)
    end = start + chunk_samples

    # Extract the chunk of audio
    chunk = y[start:end]
    plt.figure(figsize=(4, 2))
    librosa.display.waveshow(chunk, sr=sr)
    plt.show()
```


0.5
0.0
−0.5
0 0.5 1 1.5 2 2.5 3 3.5 4
Time

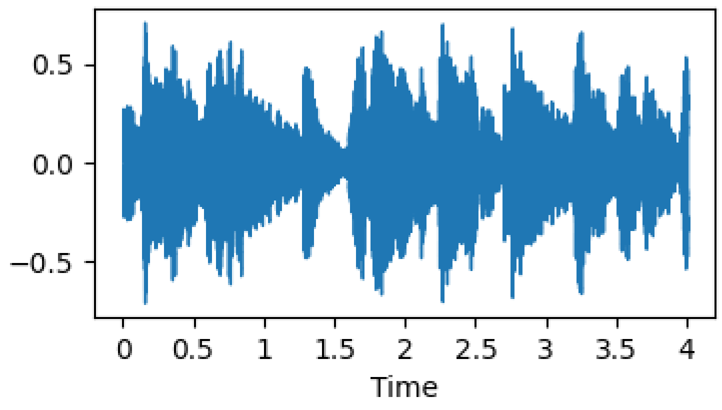

0.5
0.0
−0.5
0 0.5 1 1.5 2 2.5 3 3.5 4
Time

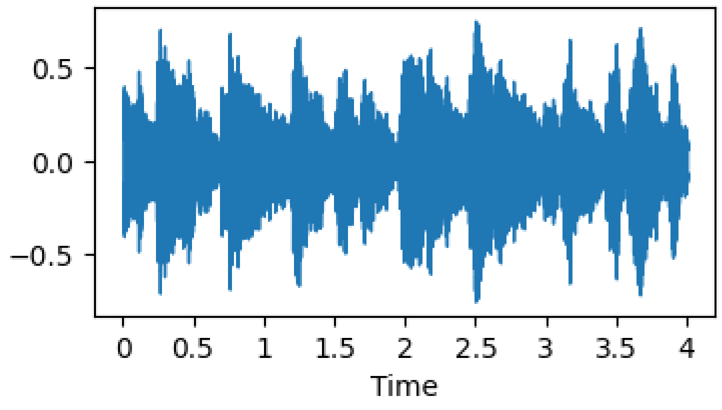

0.5
0.0
−0.5
0 0.5 1 1.5 2 2.5 3 3.5 4
Time

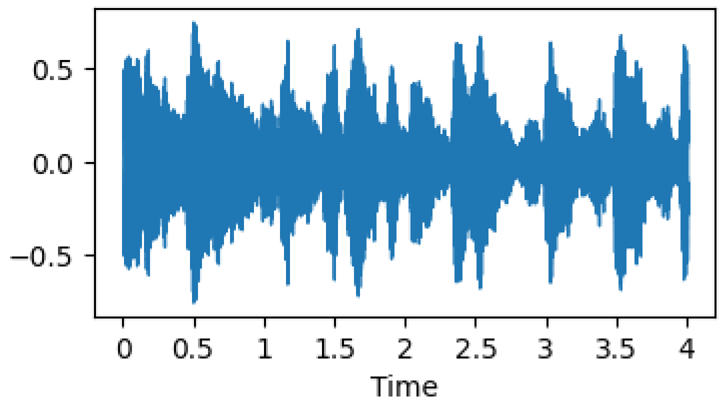

0.5
0.0
−0.5
0 0.5 1 1.5 2 2.5 3 3.5 4
Time

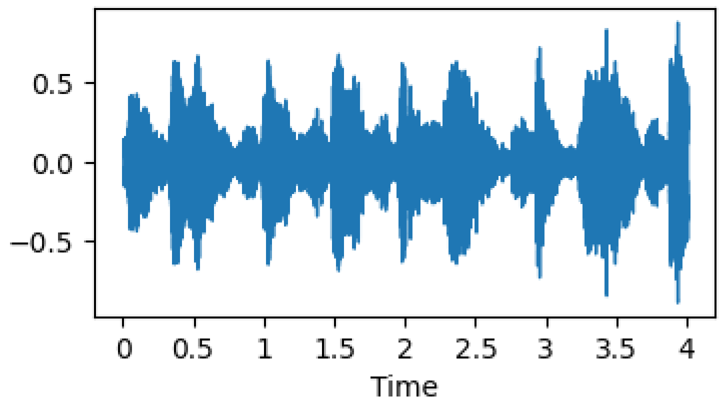

0.5
0.0
−0.5

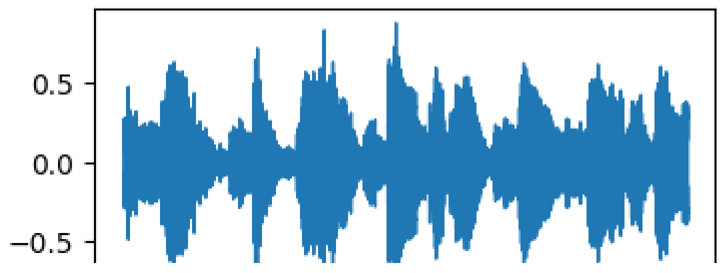
0
0.5
1
1.5
2
2.5
3
3.5
4
Time

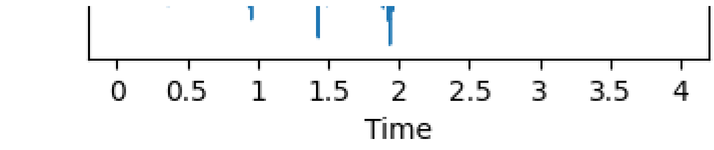

0.5
0.0
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time

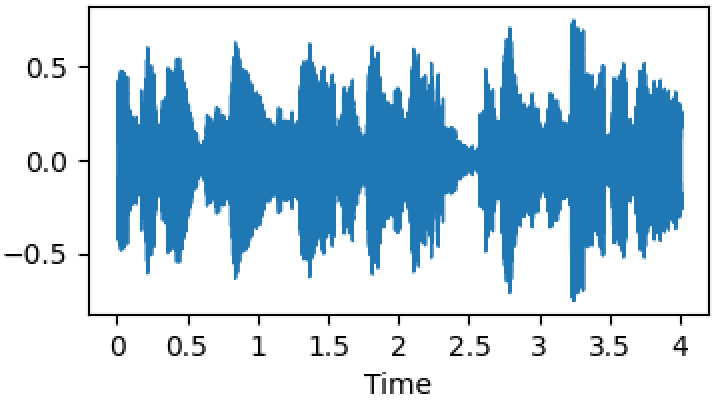

0.5
0.0
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time

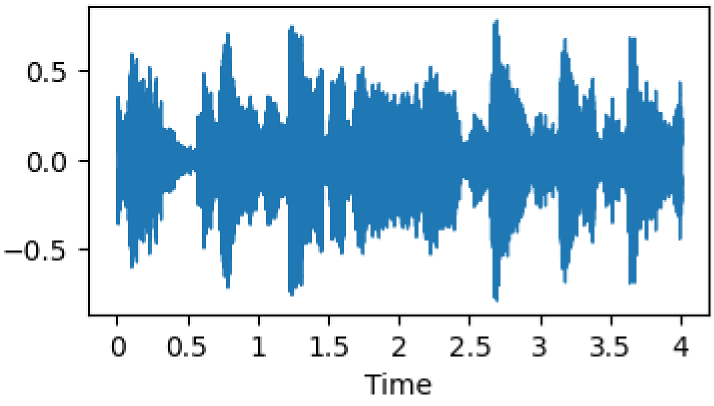

0.5
0.0
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time

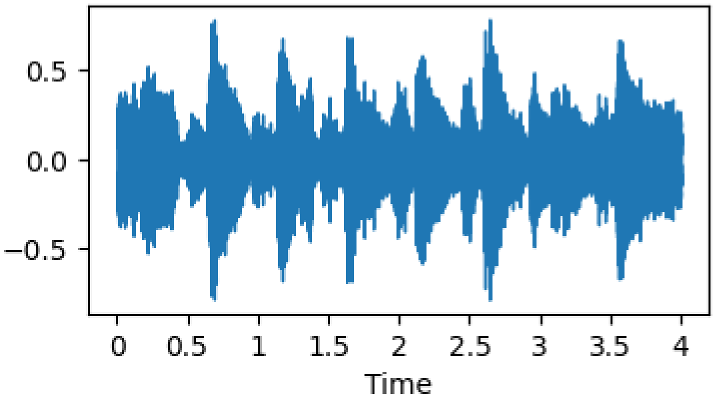

0.5
0.0
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time

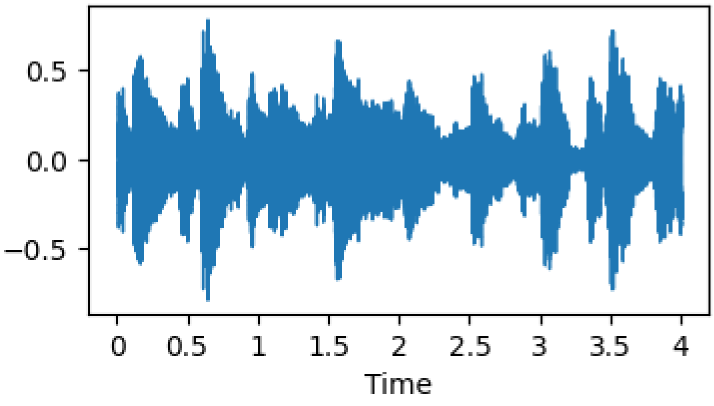

0.5
0.0
−0.5

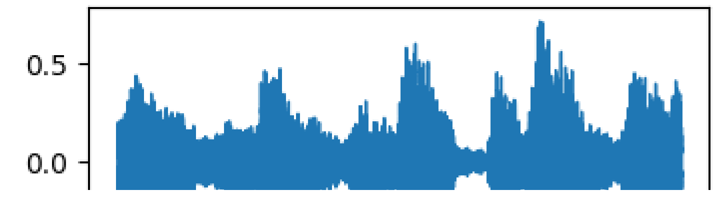

0
0.5
1
1.5
2
2.5
3
3.5
4
Time

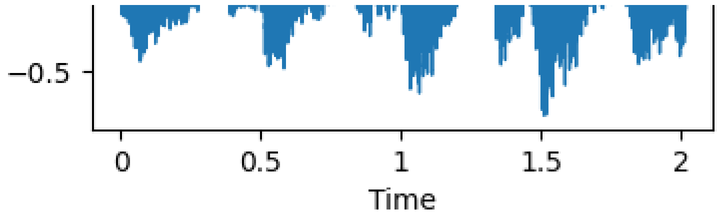

0.5
0.0
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time


0.5
0.0
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time


0.5
0.0
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time


0.5
0.0
−0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
Time


0.5
0.0

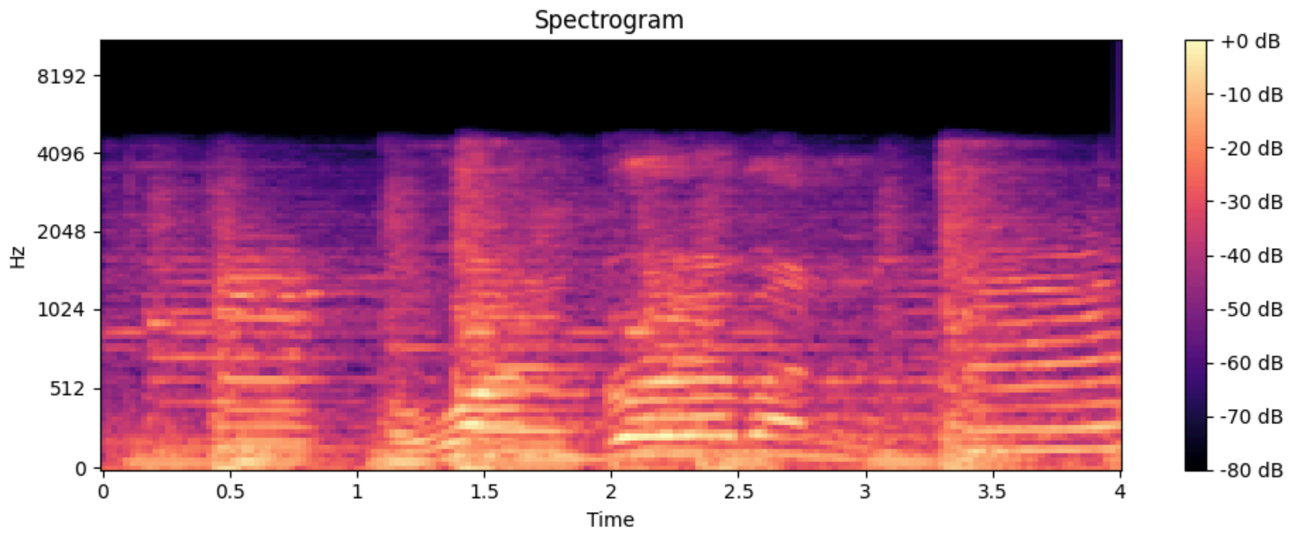


Melspectrogram Visualization

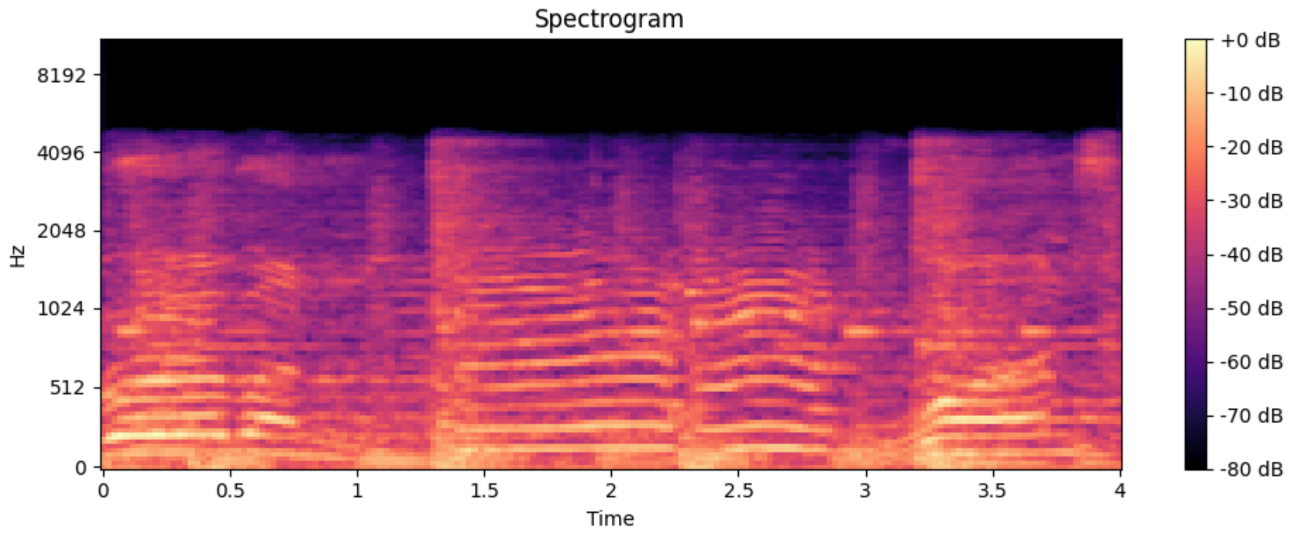
```
#Plotting Melspectrogram of Entire audio
def plot_melspectrogram(y,sr):
    # Compute the spectrogram
    spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)
    # Convert to decibels (log scale)
    spectrogram_db = librosa.power_to_db(spectrogram, ref=np.max)
    # Visualize the spectrogram
    plt.figure(figsize=(10, 4))
    librosa.display.specshow(spectrogram_db, sr=sr, x_axis='time', y_axis='mel')
    plt.colorbar(format='%+2.0f dB')
    plt.title('Spectrogram')
    plt.tight_layout()
    plt.show()
```

```
def plot_melspectrogram_chunks(y,sr):
    # Define the duration of each chunk and overlap
    chunk_duration = 4  # seconds
    overlap_duration = 2  # seconds

    # Convert durations to samples
    chunk_samples = chunk_duration * sr
    overlap_samples = overlap_duration * sr

    # Calculate the number of chunks
    num_chunks = int(np.ceil((len(y) - chunk_samples) / (chunk_samples - overlap_

    # Iterate over each chunk
    for i in range(num_chunks):
        # Calculate start and end indices of the chunk
        start = i * (chunk_samples - overlap_samples)
        end = start + chunk_samples

        # Extract the chunk of audio
        chunk = y[start:end]

        # Compute the Mel spectrogram for the chunk
        mel_spectrogram = librosa.feature.melspectrogram(y=chunk, sr=sr)
        print(mel_spectrogram.shape)
        spectrogram_db = librosa.power_to_db(mel_spectrogram, ref=np.max)
        # Visualize the spectrogram
        plt.figure(figsize=(10, 4))
        librosa.display.specshow(spectrogram_db, sr=sr, x_axis='time', y_axis='me
        plt.colorbar(format='%+2.0f dB')
        plt.title('Spectrogram')
        plt.tight_layout()
        plt.show()


#Spectrogram of Audio chunks
plot_melspectrogram_chunks(y=x,sr=sr)
```

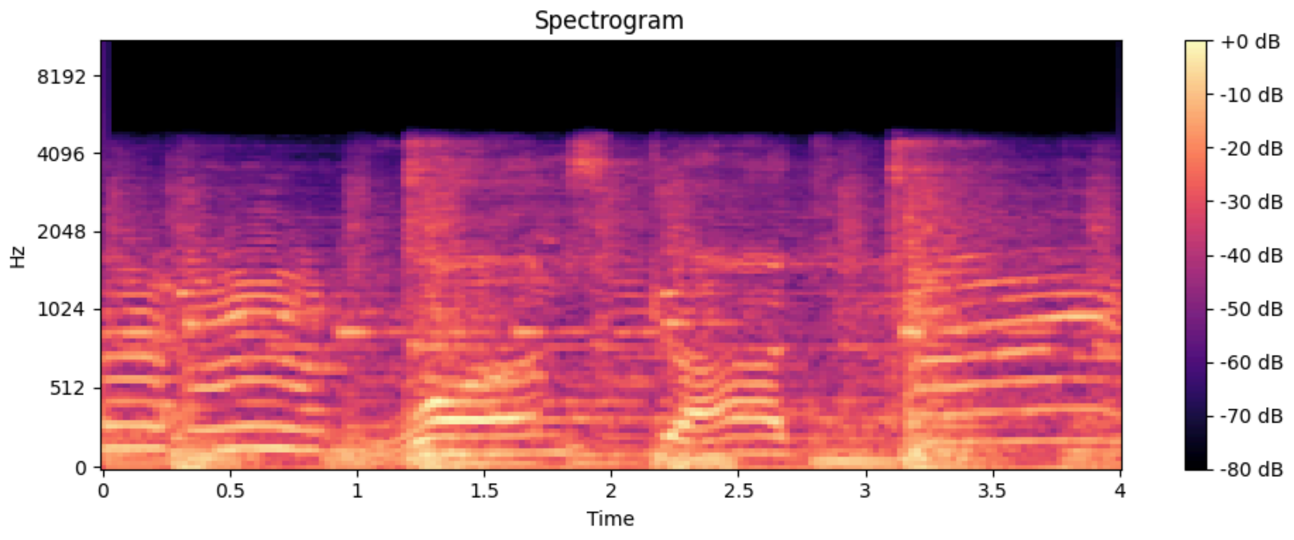
```
(128, 173)
```



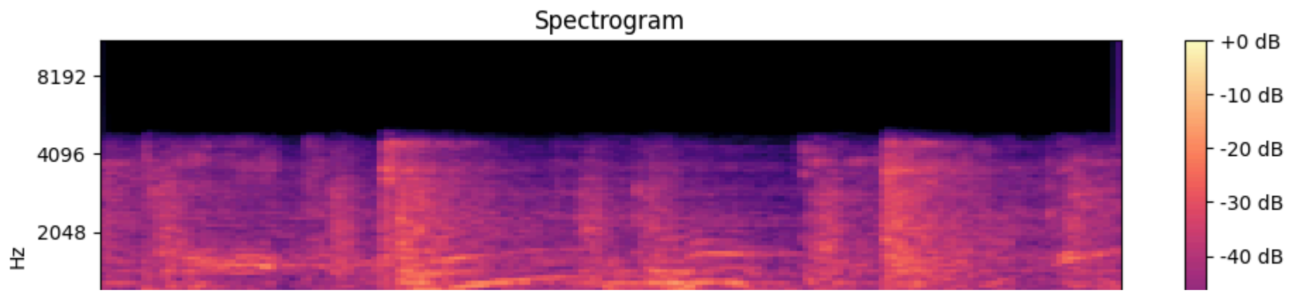
```
(128, 173)
```

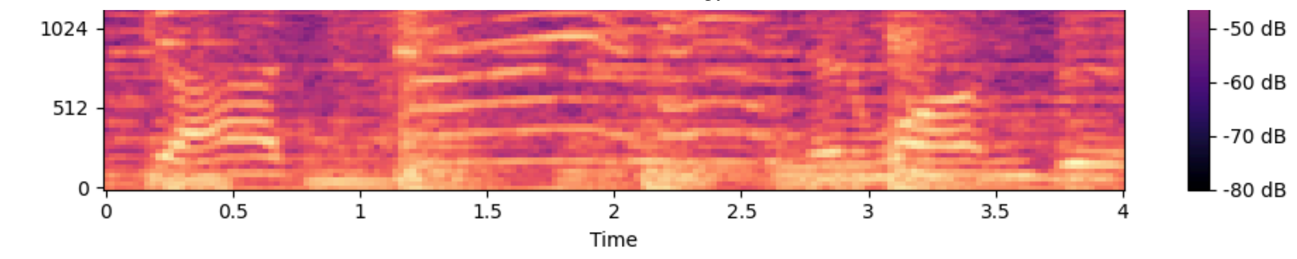


```
(128, 173)
```

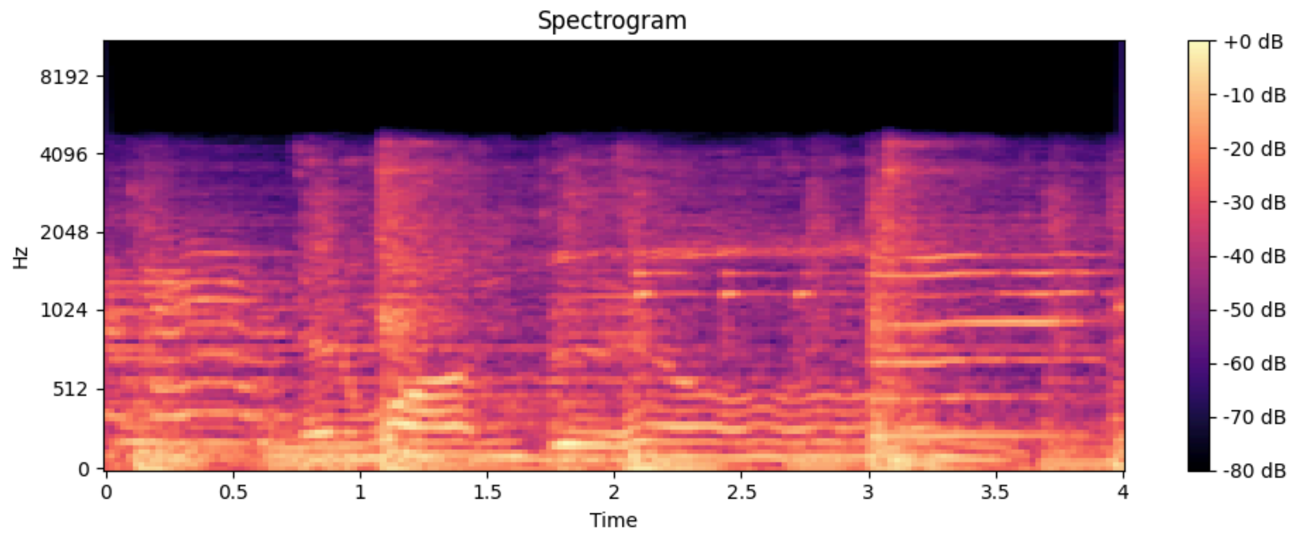


```
(128, 173)
```

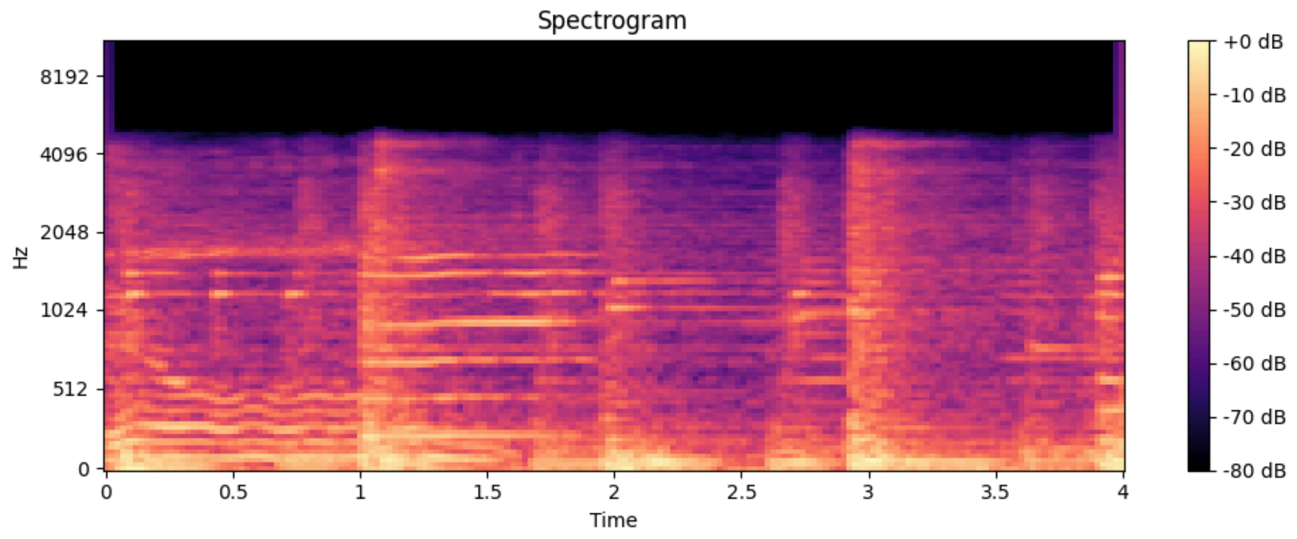




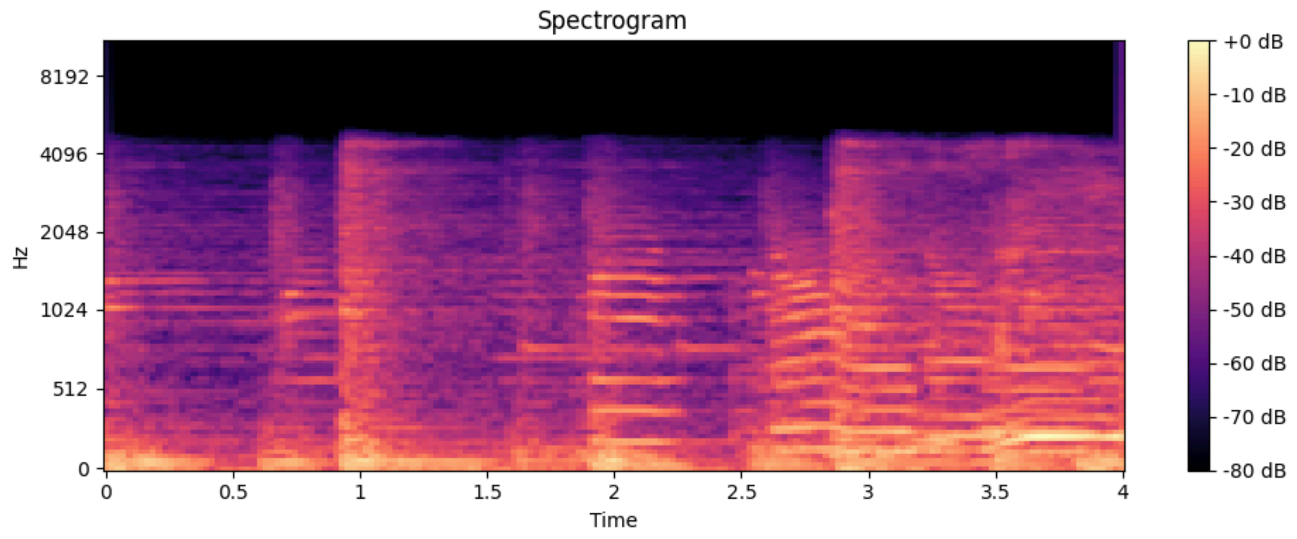
```
(128, 173)
```



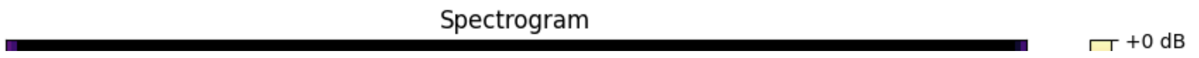
```
(128, 173)
```

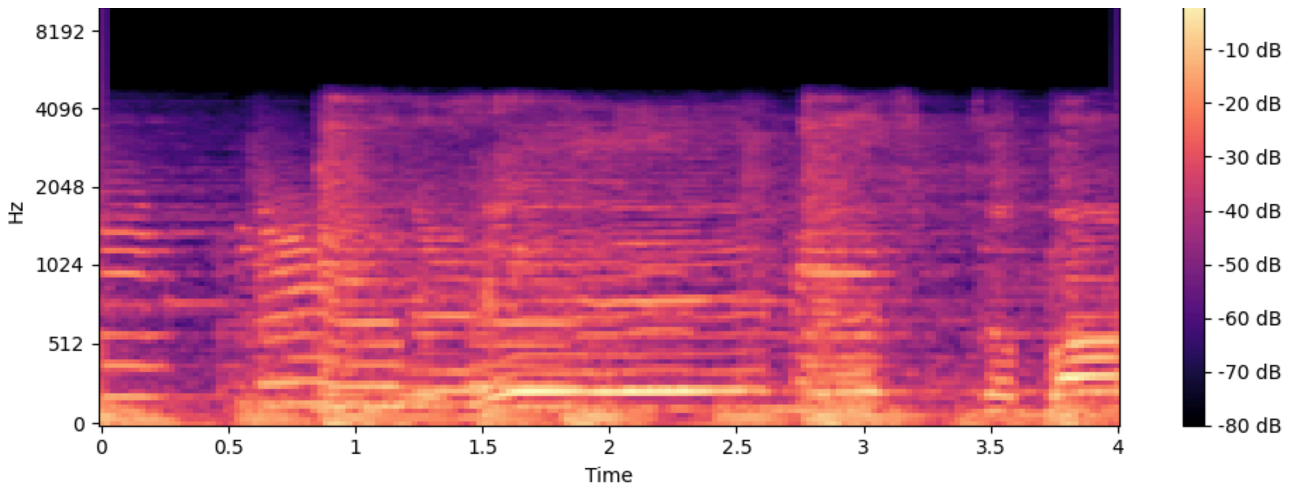


```
(128, 173)
```

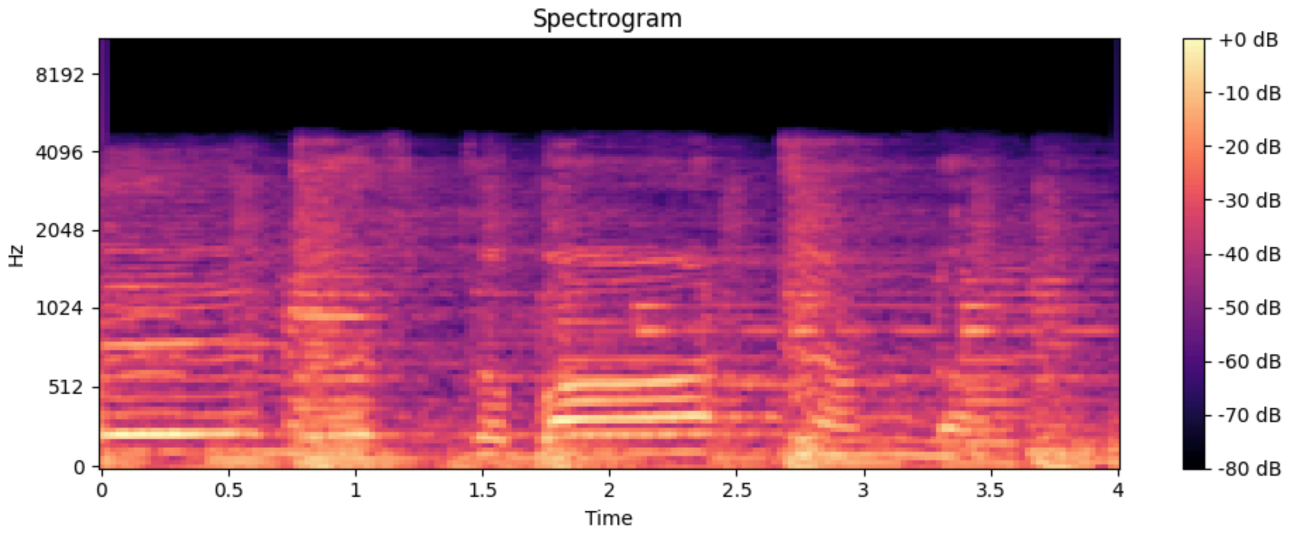


```
(128, 173)
```

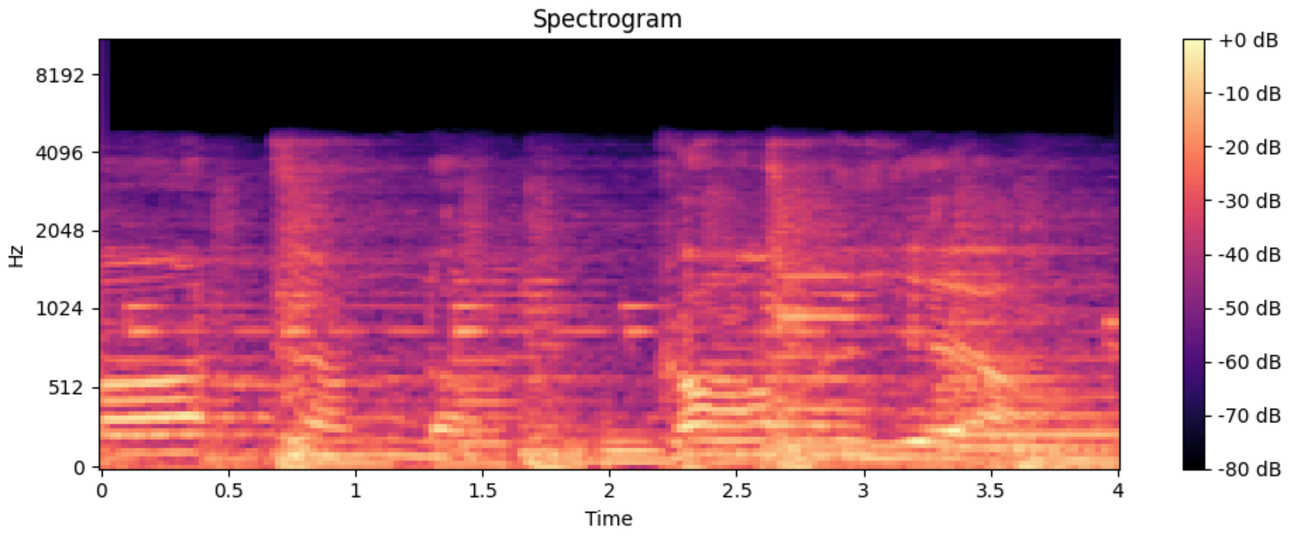




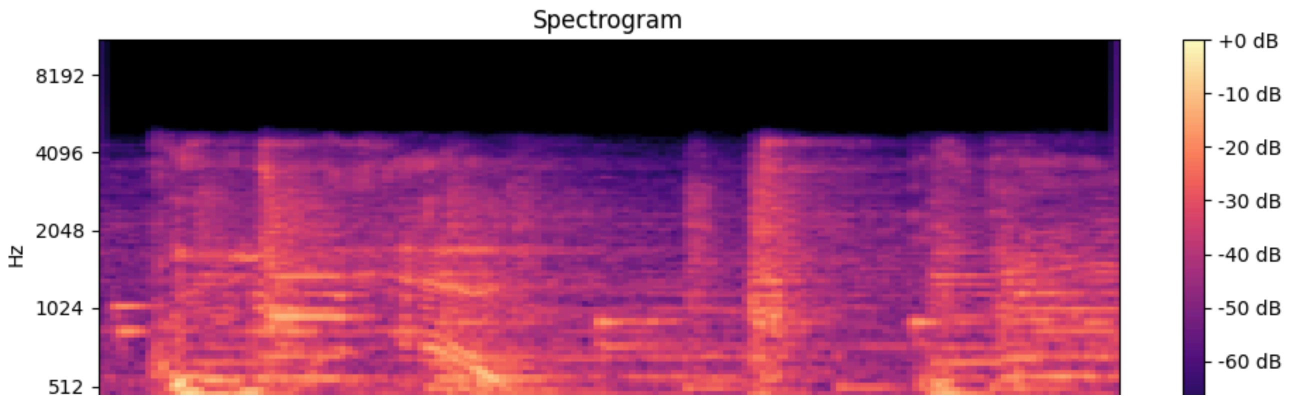
```
(128, 173)
```

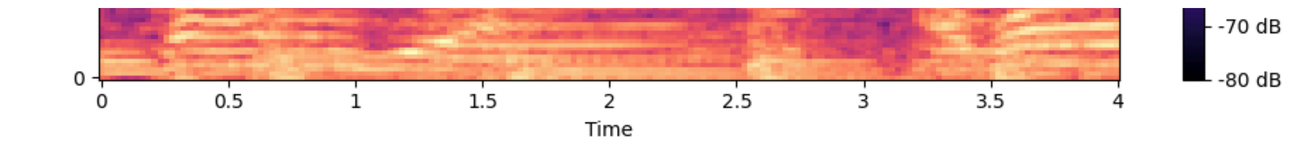


```
(128, 173)
```

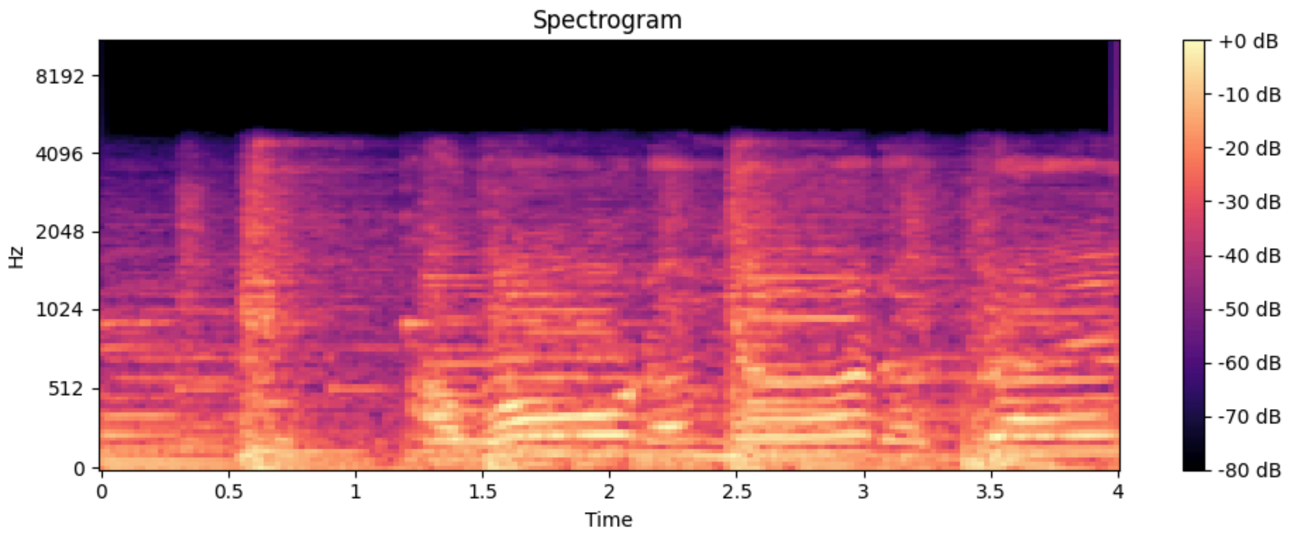


```
(128, 173)
```

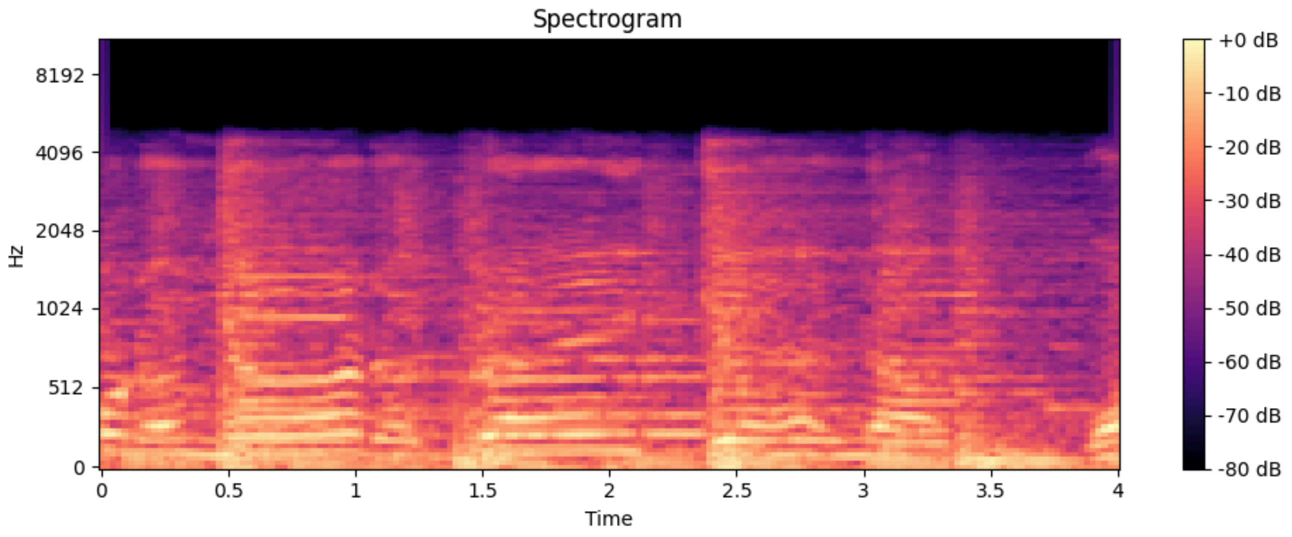




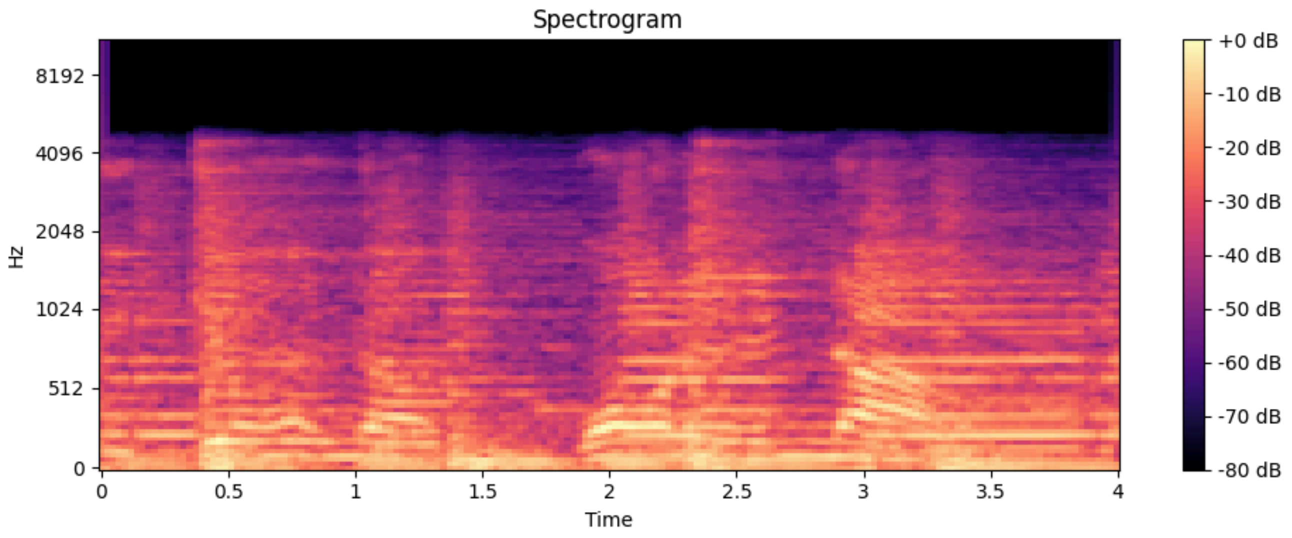
```
(128, 173)
```



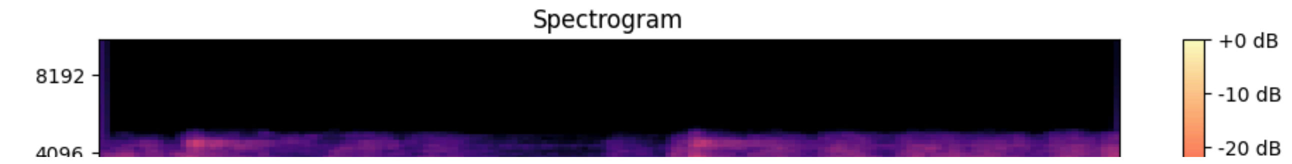
```
(128, 173)
```

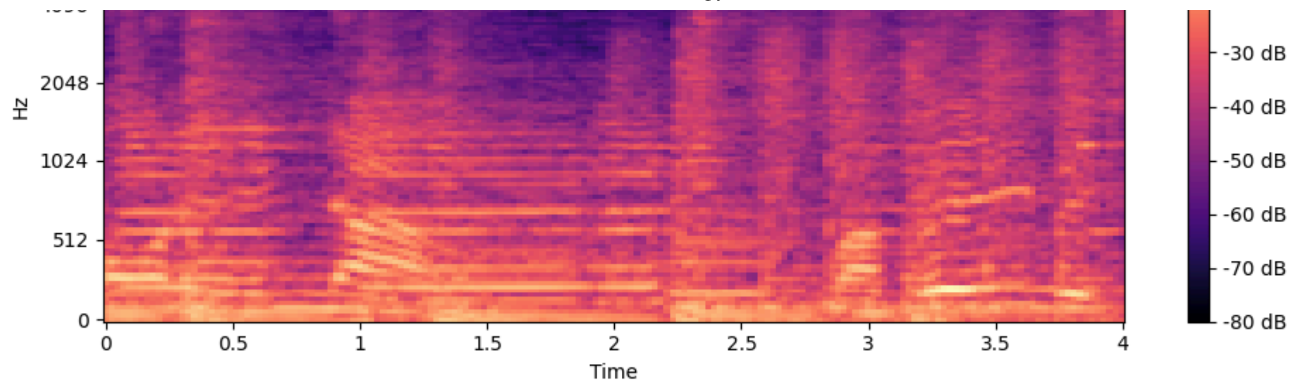


```
(128, 173)
```

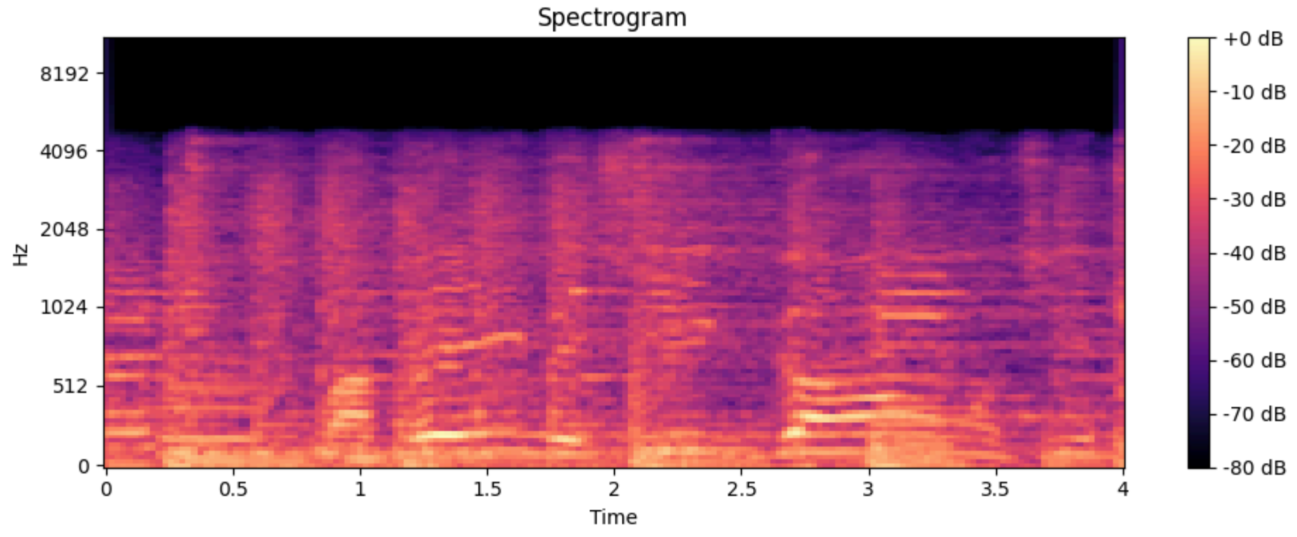


```
(128, 173)
```

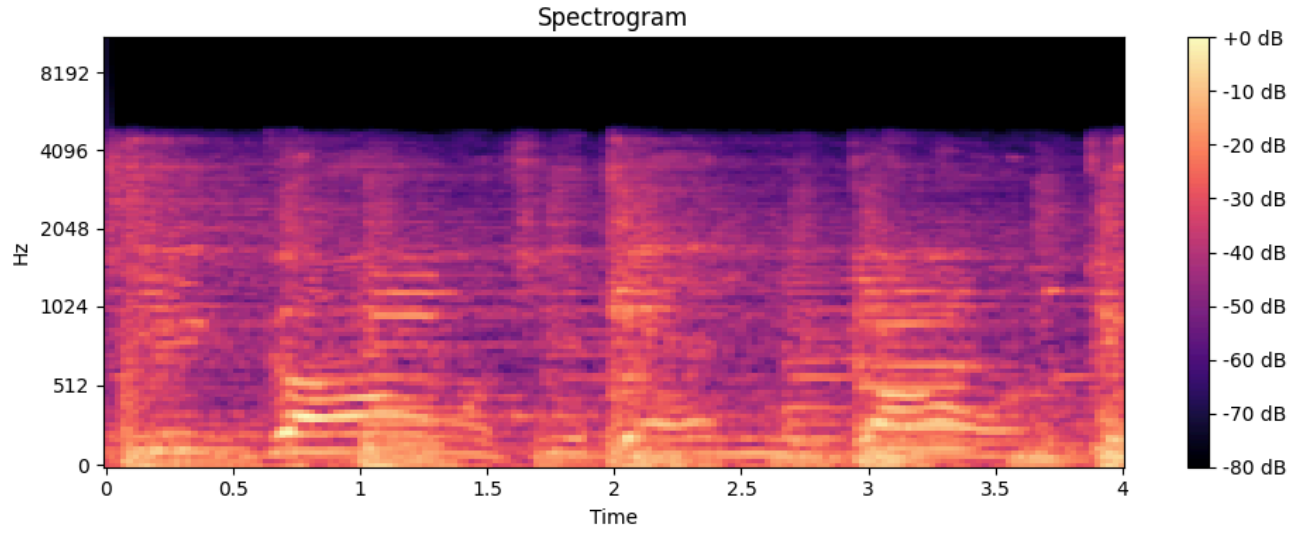




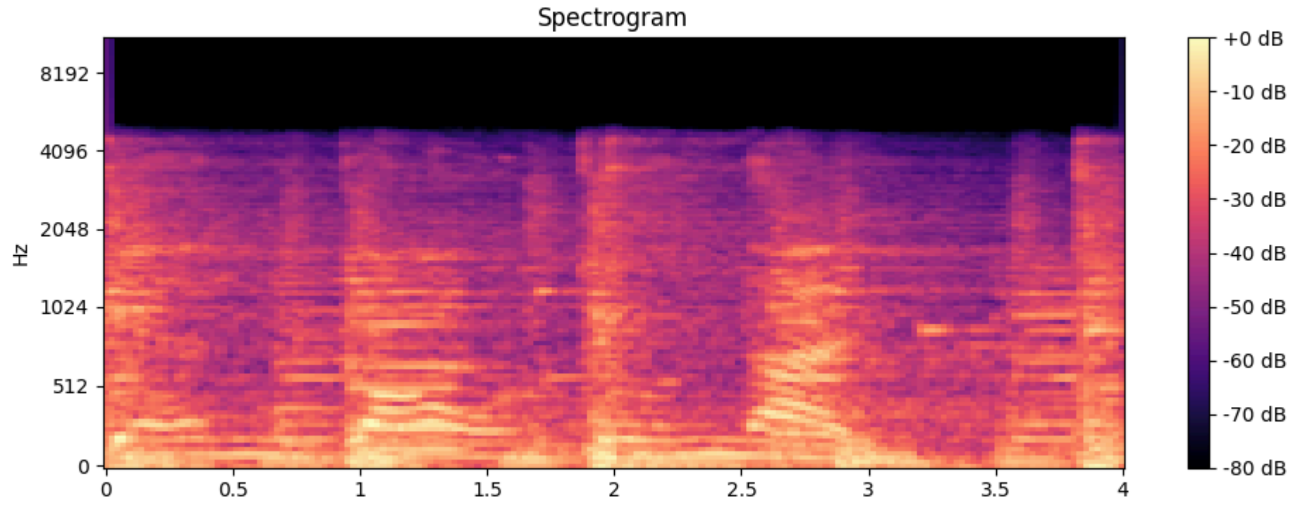
```
(128, 173)
```



```
(128, 173)
```

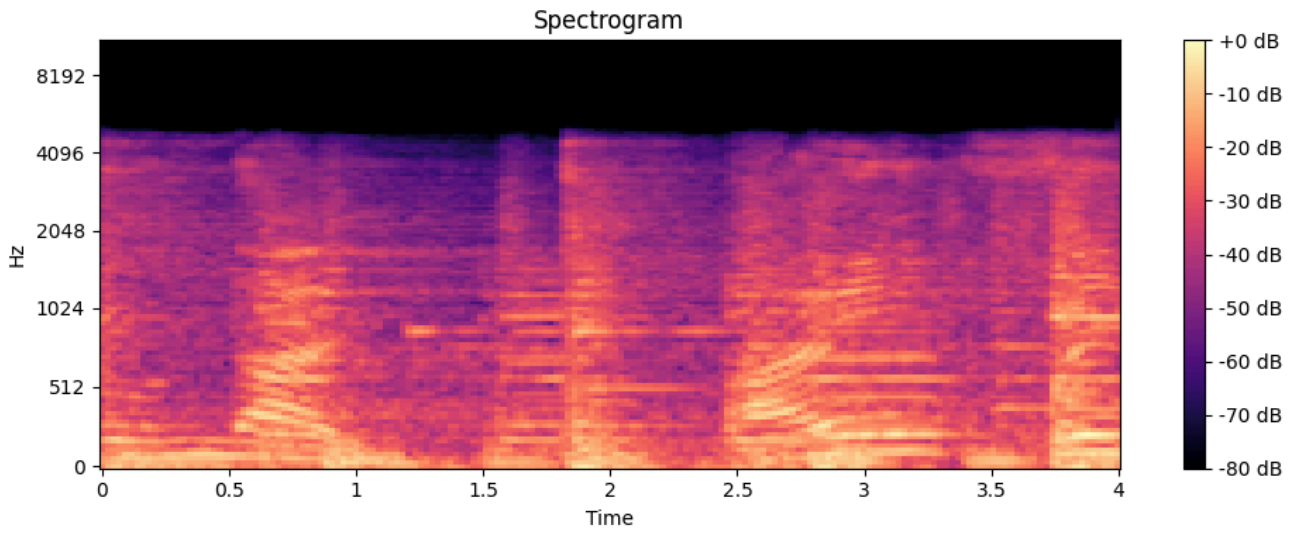


```
(128, 173)
```

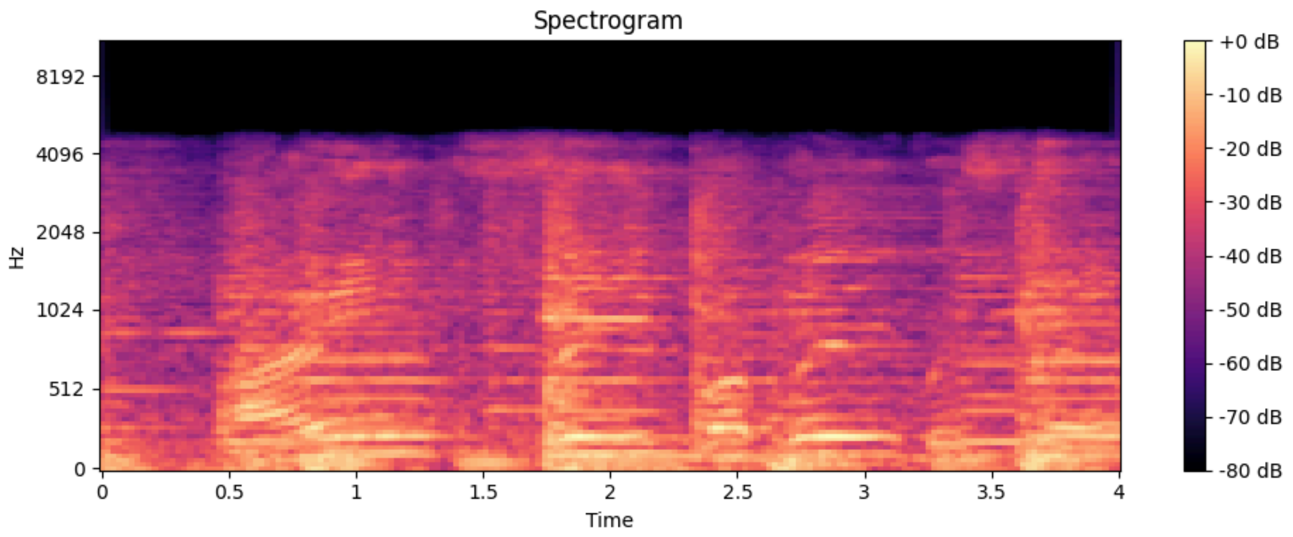


Time

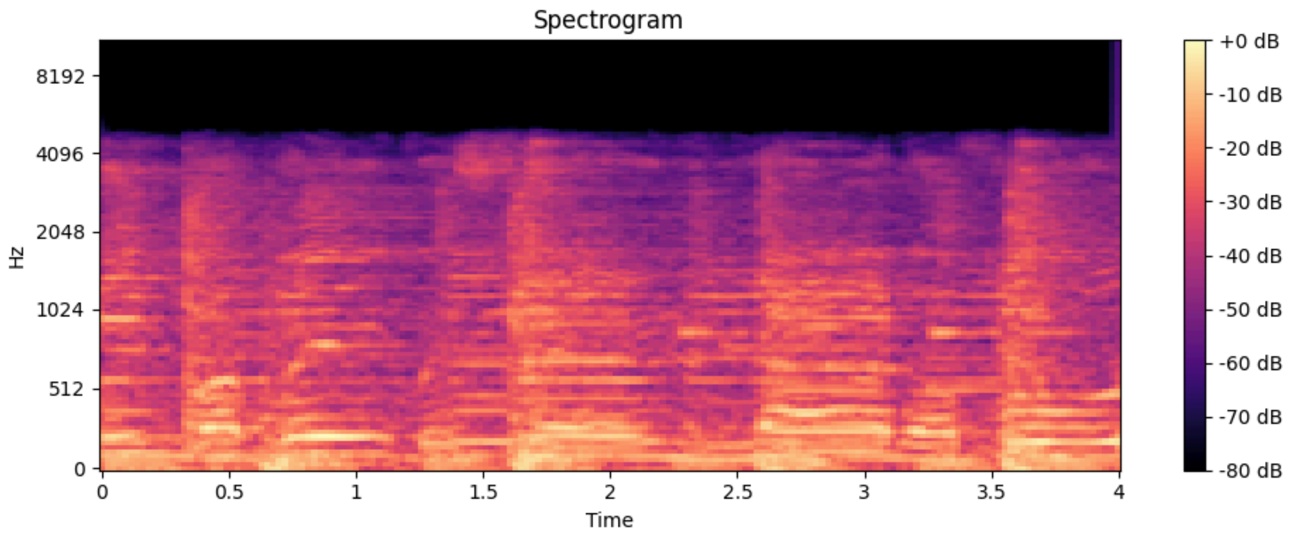
```
(128, 173)
```



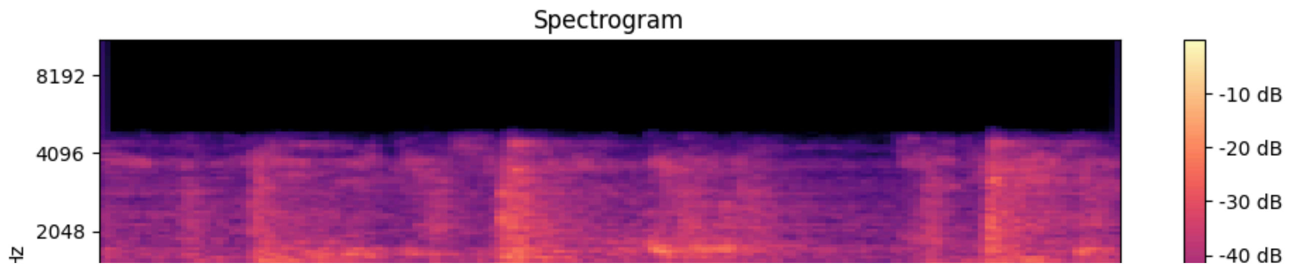
```
(128, 173)
```

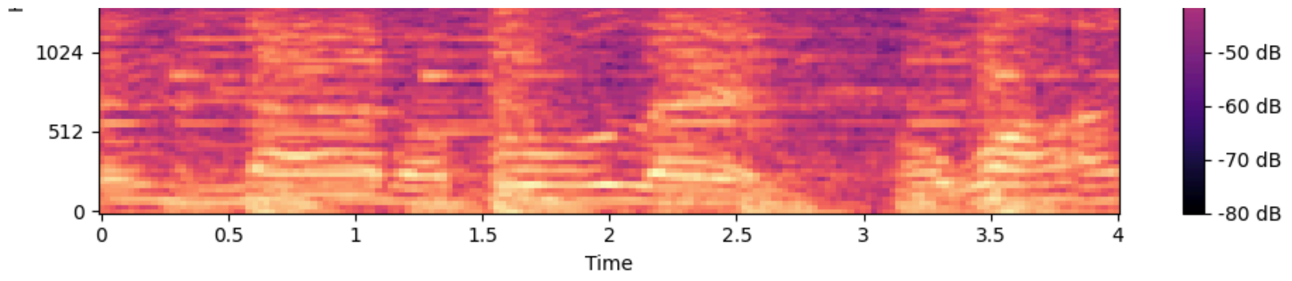


```
(128, 173)
```

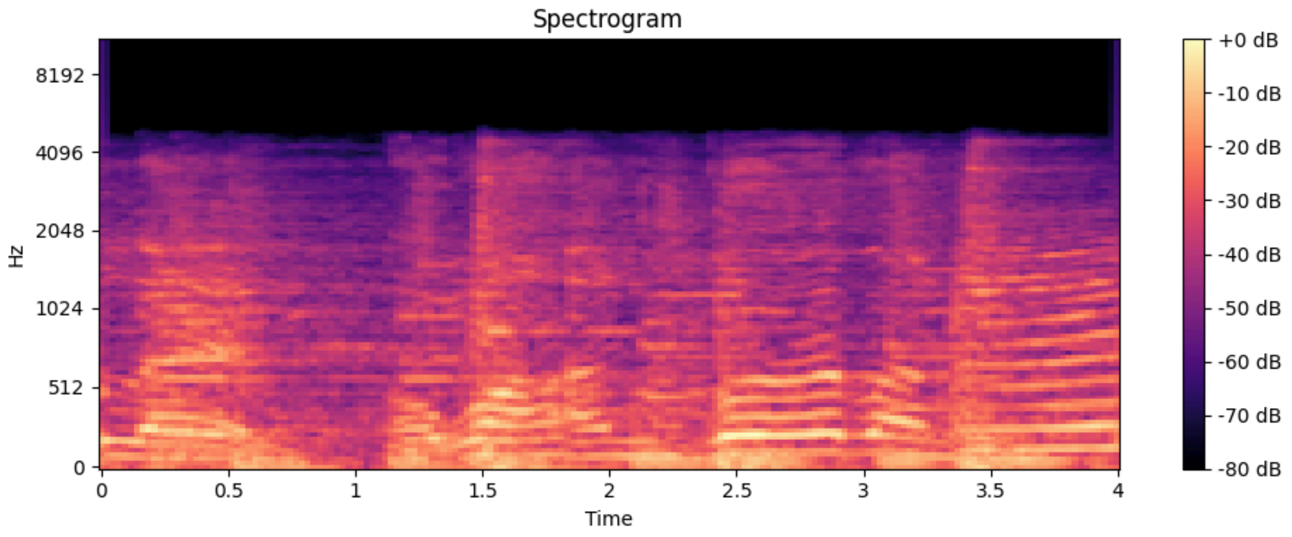


```
(128, 173)
```

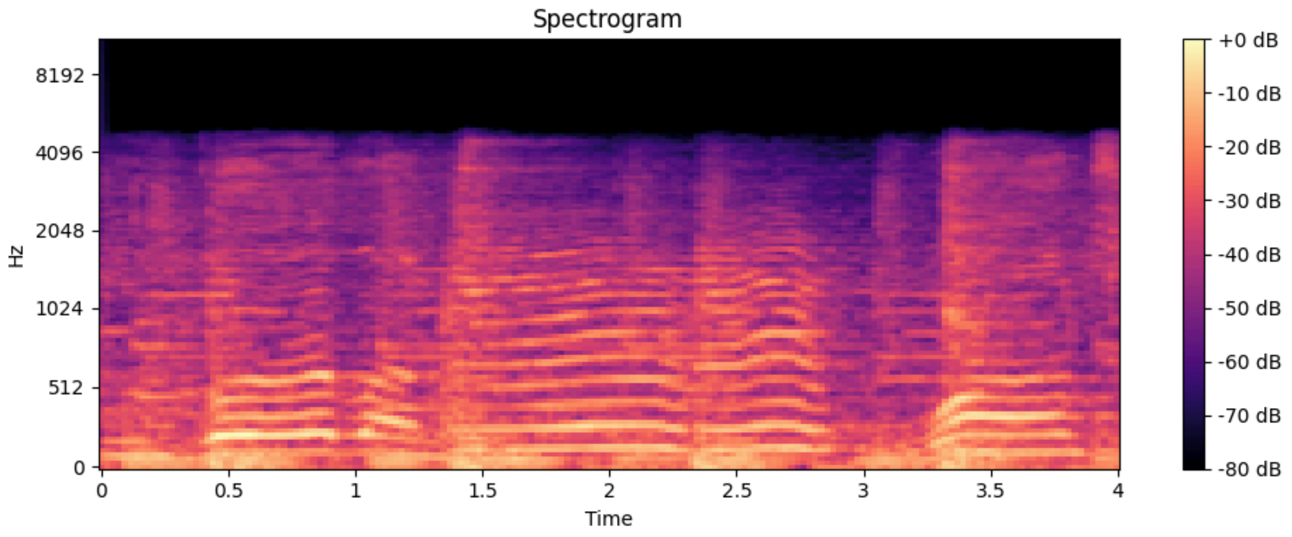




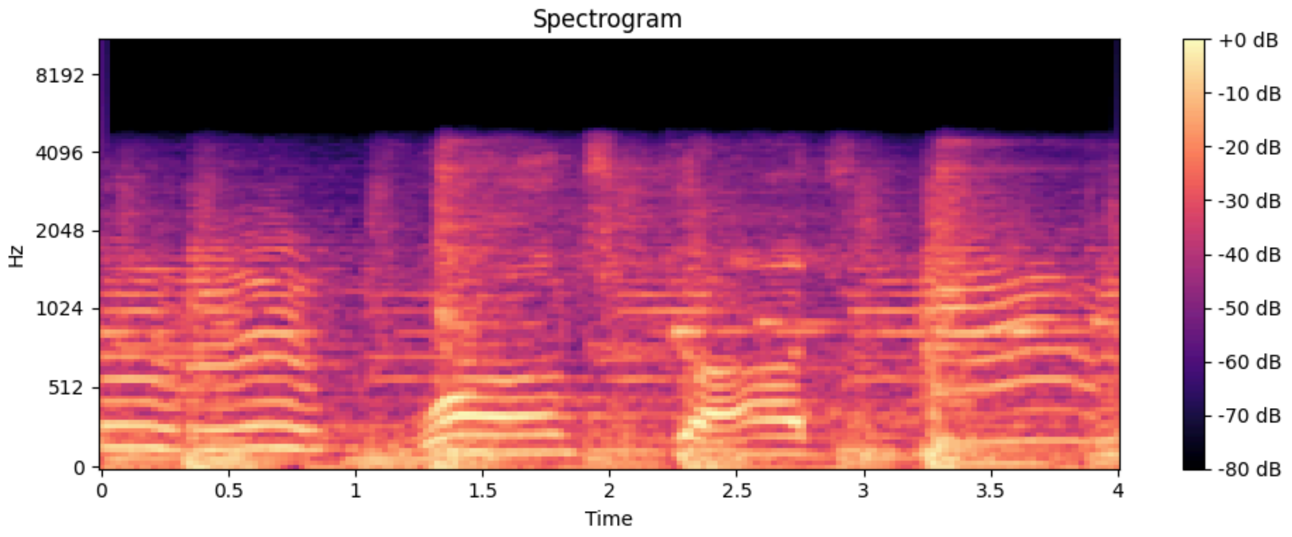
```
(128, 173)
```



```
(128, 173)
```

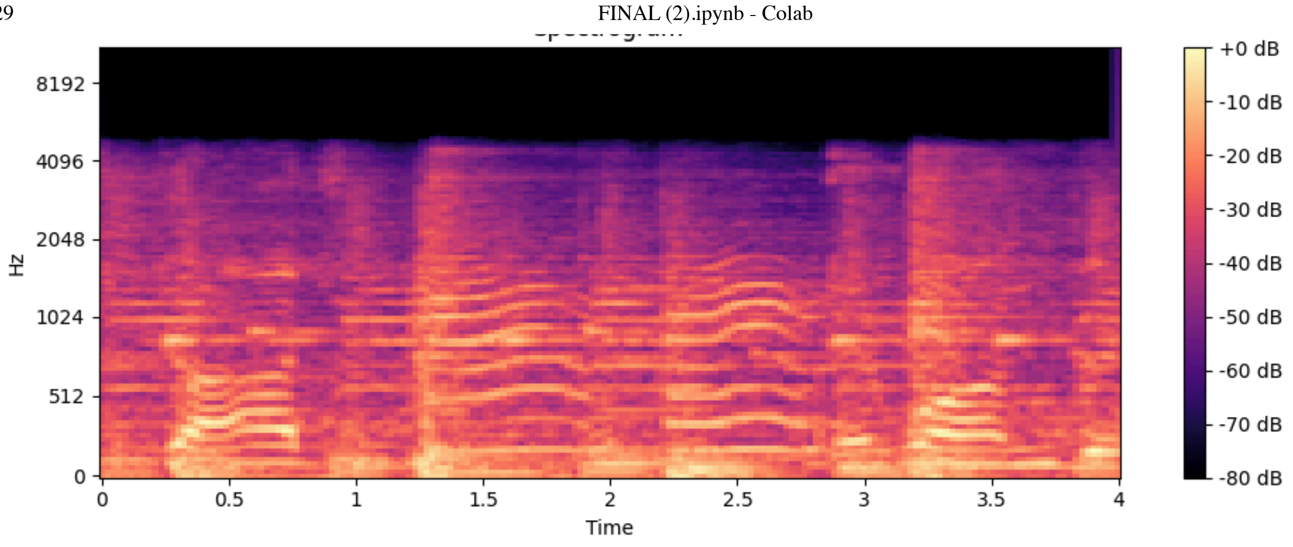


```
(128, 173)
```

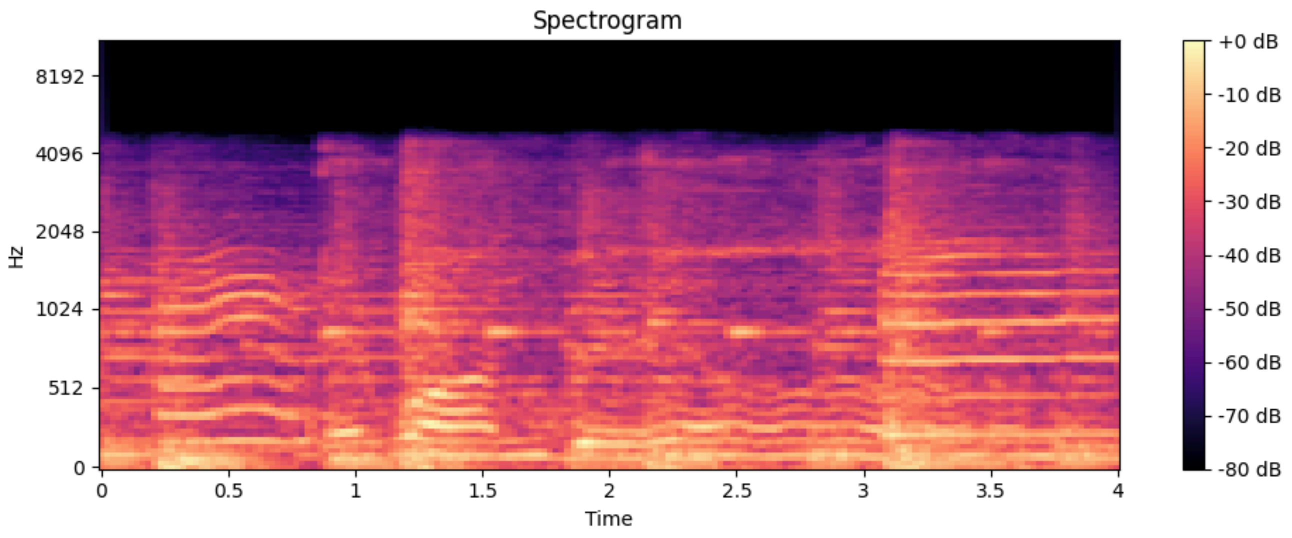


```
(128, 173)
```

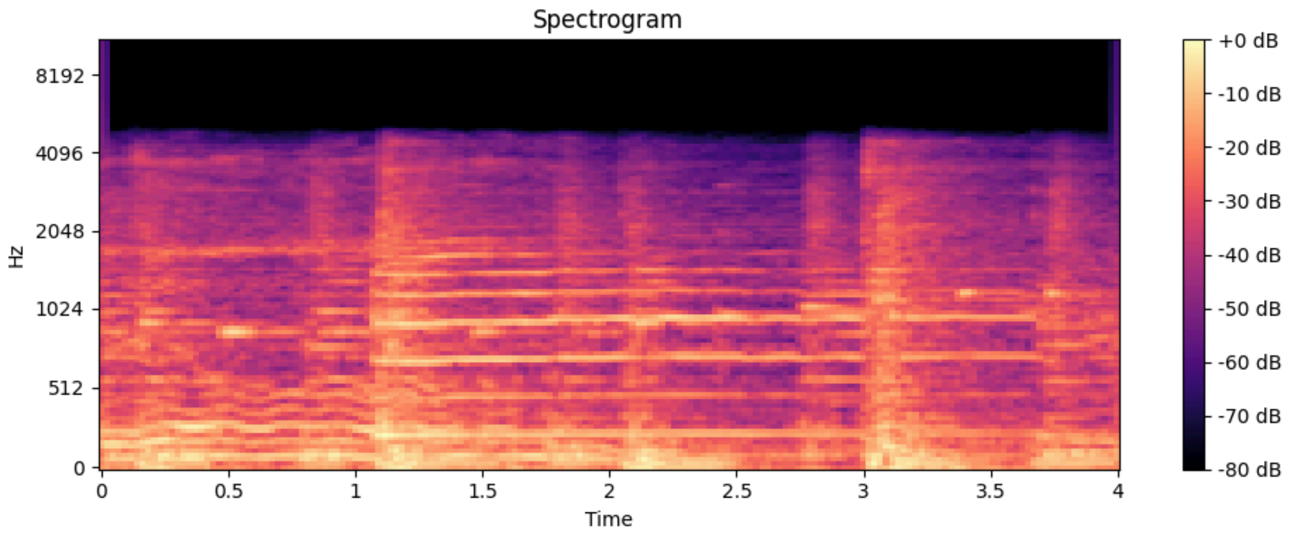
Spectrogram



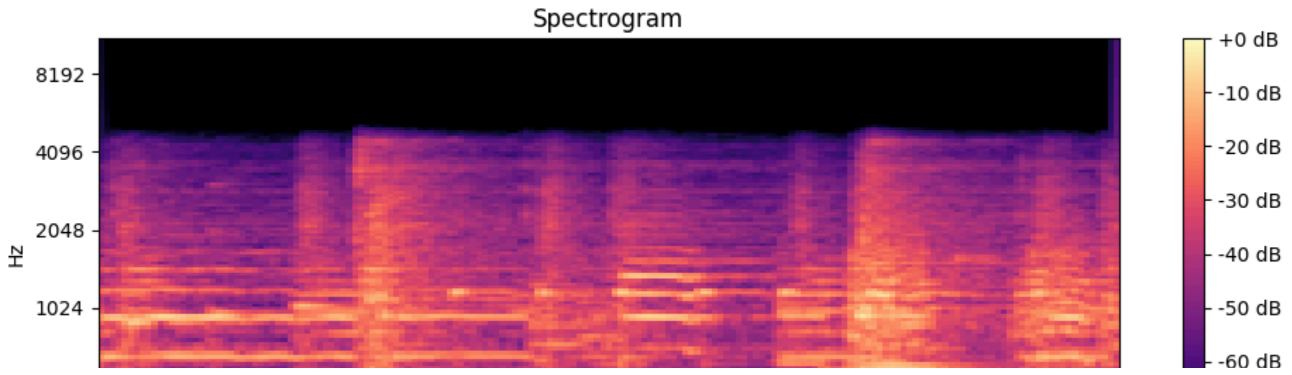
```
(128, 173)
```

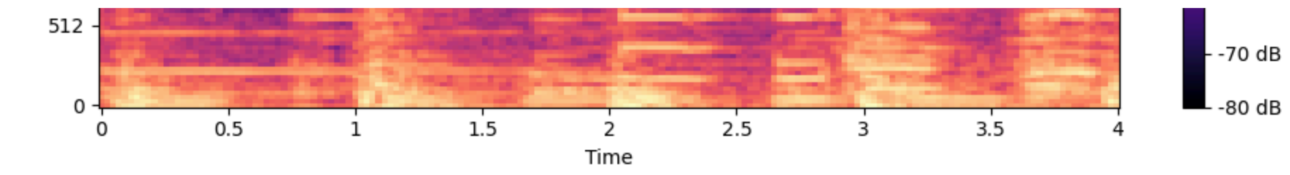


```
(128, 173)
```

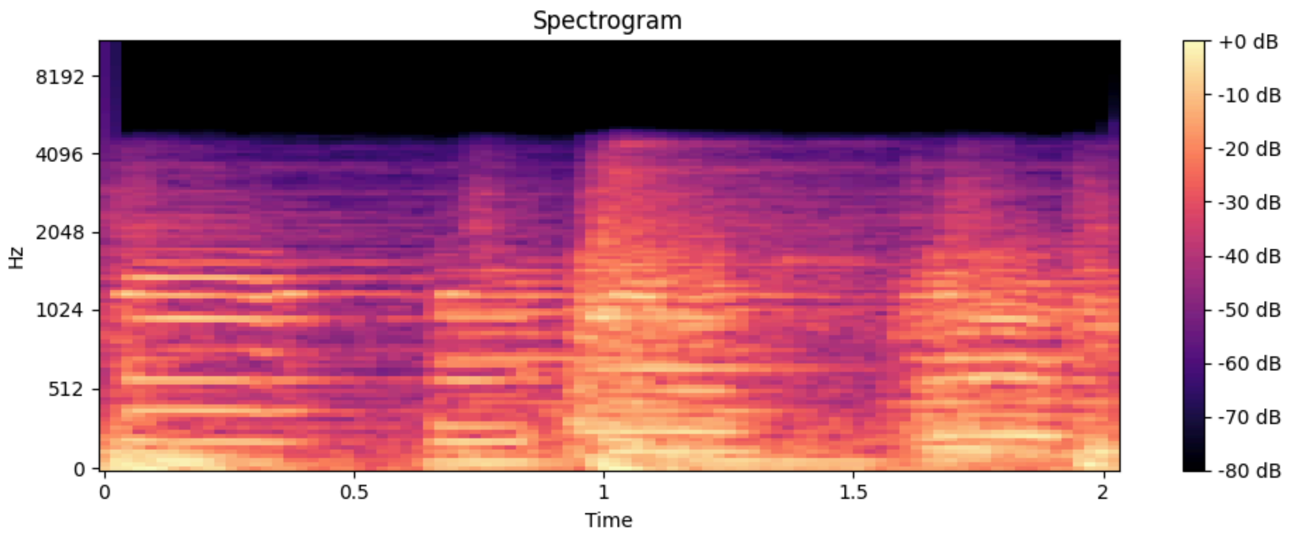


```
(128, 173)
```

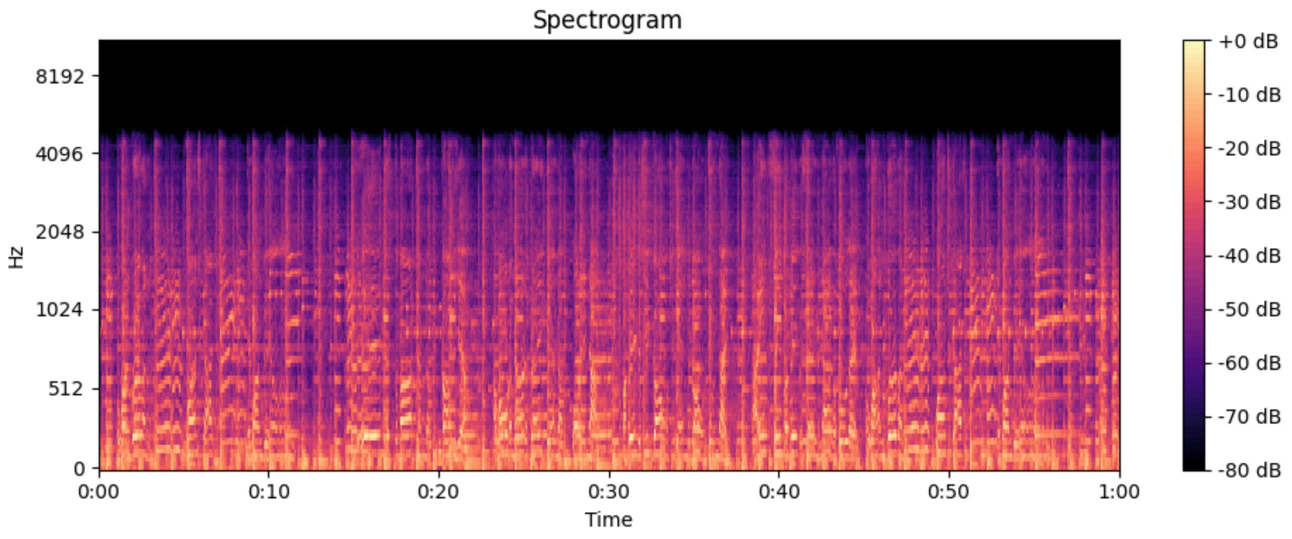



512
0
0
0.5
1
1.5
2
2.5
3
3.5
4
Time
-70 dB
-80 dB

```
(128, 88)
```


Spectrogram
8192
4096
2048
1024
512
0
Hz
0
0.5
1
1.5
2
Time
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB

```
#Spectrogram of Entire audio
plot_melspectrogram(y=x,sr=sr)
```



Data Preprocessing

```
# Define your folder structure
data_dir = '/content/drive/MyDrive/Data 2/genres_original/'
classes = ['blues', 'classical','country','disco','hiphop','jazz','metal','pop','
```

```
# Load and preprocess audio data
def load_and_preprocess_data(data_dir, classes, target_shape=(150, 150)):
    data = []
    labels = []

    for i_class, class_name in enumerate(classes):
        class_dir = os.path.join(data_dir, class_name)
        print("Processing--",class_name)
        for filename in os.listdir(class_dir):
            if filename.endswith('.wav'):
                file_path = os.path.join(class_dir, filename)
                audio_data, sample_rate = librosa.load(file_path, sr=None)
                # Perform preprocessing (e.g., convert to Mel spectrogram and res
                # Define the duration of each chunk and overlap
                chunk_duration = 4  # seconds
                overlap_duration = 2  # seconds

                # Convert durations to samples
                chunk_samples = chunk_duration * sample_rate
                overlap_samples = overlap_duration * sample_rate

                # Calculate the number of chunks
                num_chunks = int(np.ceil((len(audio_data) - chunk_samples) / (chu

                # Iterate over each chunk
                for i in range(num_chunks):
                    # Calculate start and end indices of the chunk
                    start = i * (chunk_samples - overlap_samples)
                    end = start + chunk_samples

                    # Extract the chunk of audio
                    chunk = audio_data[start:end]

                    # Compute the Mel spectrogram for the chunk
                    mel_spectrogram = librosa.feature.melspectrogram(y=chunk, sr=

                #mel_spectrogram = librosa.feature.melspectrogram(y=audio_data, s
                    mel_spectrogram = resize(np.expand_dims(mel_spectrogram, axis
                    data.append(mel_spectrogram)
                    labels.append(i_class)

    return np.array(data), np.array(labels)
```

Start coding or generate with AI.

Splitting Dataset into Training and Test set

```
# Split data into training and testing sets
data, labels = load_and_preprocess_data(data_dir, classes)
#print("\nData:",data,"\nlabel",labels)
```

```
Processing-- blues
Processing-- classical
Processing-- country
Processing-- disco
Processing-- hiphop
Processing-- jazz
Processing-- metal
Processing-- pop
Processing-- reggae
Processing-- rock
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,
```

Building CNN Model

```
model = tf.keras.models.Sequential()
X_train[0].shape

model.add(Conv2D(filters=32,kernel_size=3,padding='same',activation='relu',input_
model.add(Conv2D(filters=32,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))

model.add(Conv2D(filters=64,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=64,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))

model.add(Conv2D(filters=128,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=128,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))

model.add(tf.keras.layers.Dropout(0.3))

model.add(Conv2D(filters=256,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=256,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))

model.add(Conv2D(filters=512,kernel_size=3,padding='same',activation='relu'))
model.add(Conv2D(filters=512,kernel_size=3,activation='relu'))
model.add(MaxPool2D(pool_size=2,strides=2))

model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(units=1200,activation='relu'))
model.add(Dropout(0.45))

#Output Layer
model.add(Dense(units=len(classes),activation='softmax'))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-c12e942e708b> in <cell line: 1>()
----> 1 model = tf.keras.models.Sequential()
      2 X_train[0].shape
      3 
      4 
model.add(Conv2D(filters=32,kernel_size=3,padding='same',activation='relu',in
      5 model.add(Conv2D(filters=32,kernel_size=3,activation='relu'))

NameError: name 'tf' is not defined
```

Next steps: Explain error

```
model.summary()
```

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━
┃ Layer (type)                         ┃ Output Shape                ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━
│ conv2d (Conv2D)                      │ (None, 150, 150, 32)        │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ conv2d_1 (Conv2D)                    │ (None, 148, 148, 32)        │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ max_pooling2d (MaxPooling2D)         │ (None, 74, 74, 32)          │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ conv2d_2 (Conv2D)                    │ (None, 74, 74, 64)          │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ conv2d_3 (Conv2D)                    │ (None, 72, 72, 64)          │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 36, 36, 64)          │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ conv2d_4 (Conv2D)                    │ (None, 36, 36, 128)         │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ conv2d_5 (Conv2D)                    │ (None, 34, 34, 128)         │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 17, 17, 128)         │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ dropout (Dropout)                    │ (None, 17, 17, 128)         │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ conv2d_6 (Conv2D)                    │ (None, 17, 17, 256)         │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ conv2d_7 (Conv2D)                    │ (None, 15, 15, 256)         │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ max_pooling2d_3 (MaxPooling2D)       │ (None, 7, 7, 256)           │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ conv2d_8 (Conv2D)                    │ (None, 7, 7, 512)           │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ conv2d_9 (Conv2D)                    │ (None, 5, 5, 512)           │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ max_pooling2d_4 (MaxPooling2D)       │ (None, 2, 2, 512)           │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ dropout_1 (Dropout)                  │ (None, 2, 2, 512)           │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ flatten (Flatten)                    │ (None, 2048)                │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ dense (Dense)                        │ (None, 1200)                │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ dropout_2 (Dropout)                  │ (None, 1200)                │
├──────────────────────────────────────┼─────────────────────────────┼───────
│ dense_1 (Dense)                      │ (None, 10)                  │
└──────────────────────────────────────┴─────────────────────────────┴───────
 Total params: 7,182,458 (27.40 MB)
 Trainable params: 7,182,458 (27.40 MB)
 Non-trainable params: 0 (0.00 B)
```

```
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentrop
```

```
X_train.shape,y_train.shape
```

```
((11980, 150, 150, 1), (11980, 10))
```

```
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weigh
training_history = model.fit(X_train, y_train, epochs=20, batch_size=32, validati
```

```
Epoch 1/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 61s 117ms/step - accuracy: 0.1964 - loss: 2.1728
Epoch 2/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 52s 75ms/step - accuracy: 0.4576 - loss: 1.5465 ·
Epoch 3/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 42s 78ms/step - accuracy: 0.5801 - loss: 1.2009 ·
Epoch 4/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 78ms/step - accuracy: 0.6615 - loss: 0.9811 ·
Epoch 5/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 78ms/step - accuracy: 0.7211 - loss: 0.8072 ·
Epoch 6/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 78ms/step - accuracy: 0.7531 - loss: 0.7125 ·
Epoch 7/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 29s 76ms/step - accuracy: 0.7858 - loss: 0.6025 ·
Epoch 8/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 28s 75ms/step - accuracy: 0.8252 - loss: 0.5087 ·
Epoch 9/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 75ms/step - accuracy: 0.8575 - loss: 0.4197 ·
Epoch 10/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 42s 78ms/step - accuracy: 0.8750 - loss: 0.3588 ·
Epoch 11/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 40s 76ms/step - accuracy: 0.8969 - loss: 0.3046 ·
Epoch 12/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 42s 78ms/step - accuracy: 0.9036 - loss: 0.2848 ·
Epoch 13/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 40s 75ms/step - accuracy: 0.9335 - loss: 0.2031 ·
Epoch 14/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 28s 75ms/step - accuracy: 0.9288 - loss: 0.2198 ·
Epoch 15/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 28s 75ms/step - accuracy: 0.9377 - loss: 0.1874 ·
```

```
model.save("Trained_model.keras") #Mac
model.save("Trained_model.h5") #Windows
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
```

```
training_history.history
```

```
  0.49223706126213074,
  0.6019198894500732,
  0.668948233127594,
  0.7197829484939575,
  0.7627713084220886,
  0.7911519408226013,
  0.8305509090423584,
  0.8544240593910217,
  0.8780467510223389,
  0.900166928768158,
  0.9069282412528992,
  0.9298831224441528,
 'loss': [1.996436595916748,
  1.4395904541015625,
  1.1397587060928345,
  0.9565372467041016,
  0.808724582195282,
  0.6869816184043884,
  0.5974817872047424,
  0.49103501439094543,
  0.4210062325000763,
  0.35975968837738037,
  0.29699549078941345,
  0.275332510471344,
  0.21439309418201447,
  0.21280182898044586,
  0.1751575917005539],
 'val_accuracy': [0.4240400791168213,
  0.5485809445381165,
  0.6570951342582703,
  0.6951586008071899,
  0.7121869921684265,
  0.7689482569694519,
  0.800000011920929,
  0.8203672766685486,
  0.8353922963142395,
  0.8507512807846069,
  0.844073474407196,
  0.8727879524230957,
  0.8808013200759888,
  0.8727879524230957,
  0.8791319131851196],
 'val_loss': [1.6337497234344482,
  1.272579550743103,
  0.9638379216194153,
  0.8703981041908264,
  0.8051190376281738,
  0.7010636925697327,
  0.586625874042511,
  0.5436527729034424,
  0.5005928874015808,
  0.4755260944366455,
  0.47711509466171265,
  0.3926008641719818,
  0.42279279232025146,
  0.40897613763809204,
  0.4208763837814331]}
```

```
#Recording History in json
import json
with open('training_hist.json','w') as f:
  json.dump(training_history.history,f)
```

Model Evaluation

```
##Model Evaluation on Training set
train_accuracy=model.evaluate(X_train,y_train,verbose=0)
print(train_accuracy[1])
```

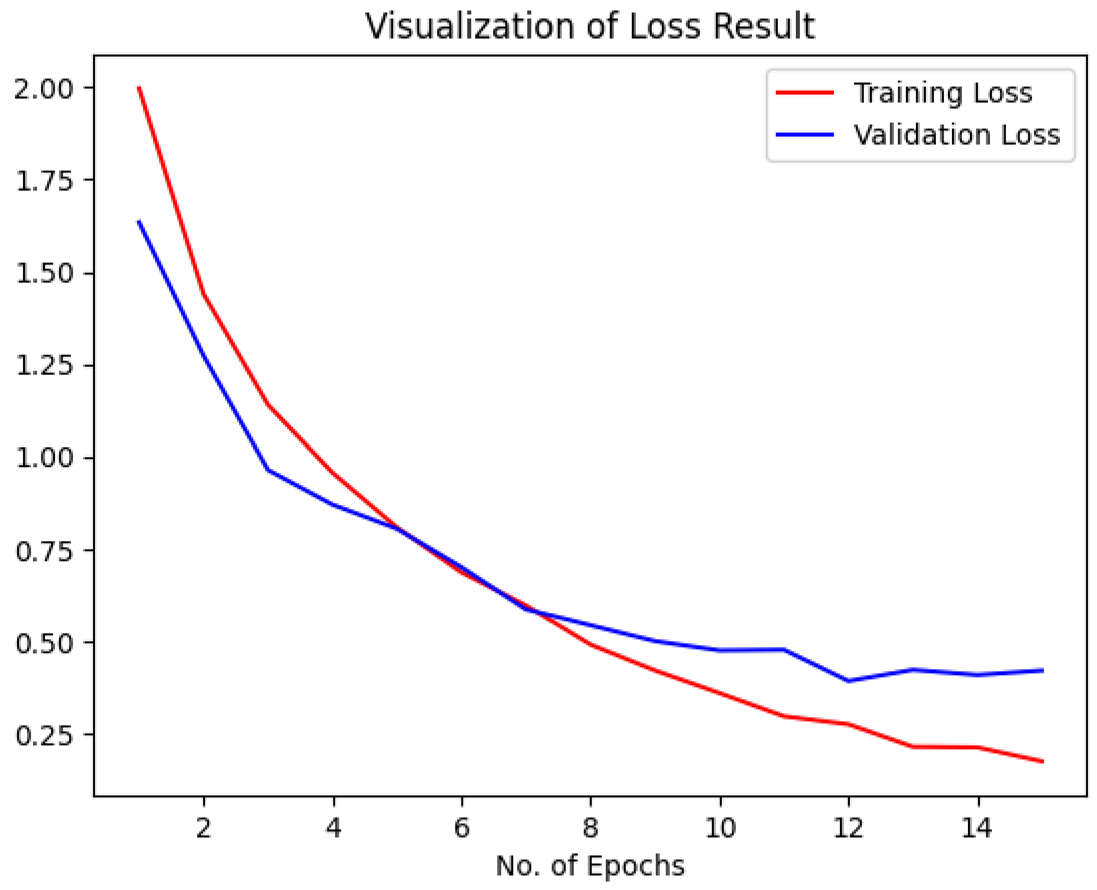
```
0.9609348773956299
```

```
##Model Evaluation on Test set
test_accuracy=model.evaluate(X_test,y_test,verbose=0)
print(test_accuracy[1])
```

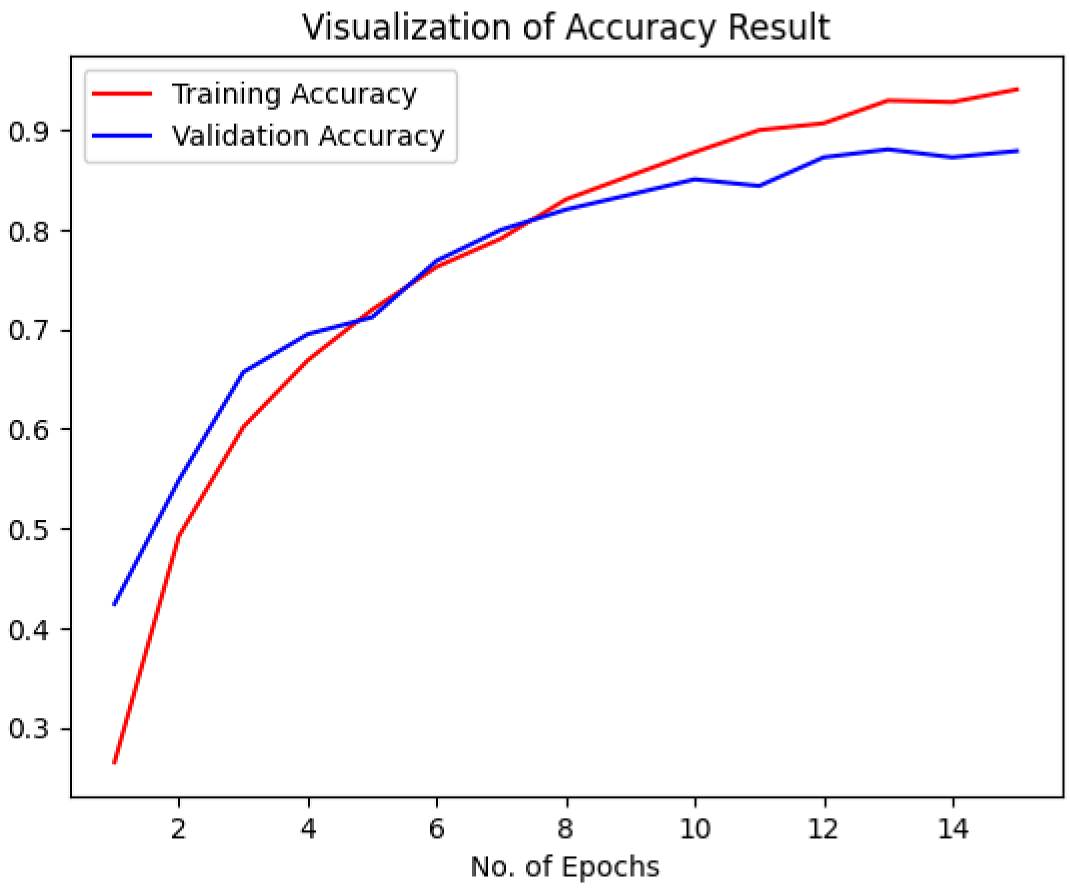
```
0.8727879524230957
```

Accuracy and Loss Visualization

```
epochs = range(1, len(training_history.history['loss']) + 1)

# Now plot
plt.plot(epochs, training_history.history['loss'], color='red', label='Training L
plt.plot(epochs, training_history.history['val_loss'], color='blue', label='Valid
plt.xlabel('No. of Epochs')
plt.title('Visualization of Loss Result')
plt.legend()
plt.show()
```



```
#Accuracy Visualization
plt.plot(epochs,training_history.history['accuracy'],color='red',label='Training
plt.plot(epochs,training_history.history['val_accuracy'],color='blue',label='Vali
plt.xlabel('No. of Epochs')
plt.title('Visualization of Accuracy Result')
plt.legend()
plt.show()
```



Precision, Recall, Confusion Metrics calculation

```
y_pred = model.predict(X_test)
y_pred
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step
array([[4.6429559e-06, 9.3179132e-08, 1.1684841e-03, ..., 9.9086154e-01,
        5.7058569e-05, 8.2353043e-04],
       [3.6382064e-06, 3.9823650e-14, 5.4341536e-13, ..., 6.8793430e-12,
        1.9366317e-09, 7.7653596e-07],
       [7.9156433e-09, 3.6882221e-16, 5.1614447e-14, ..., 2.7000200e-15,
        6.0725725e-13, 5.7835500e-08],
       ...,
       [9.9999988e-01, 1.2722124e-14, 5.8557340e-08, ..., 1.4839574e-11,
        2.5682259e-13, 7.9211304e-08],
       [3.2432501e-05, 9.9782795e-01, 9.3496463e-05, ..., 2.2222071e-04,
        3.0902054e-04, 1.8013288e-04],
       [8.9910207e-10, 9.3031980e-14, 3.2044012e-12, ..., 3.7272416e-06,
        1.3736822e-06, 5.6004790e-08]], dtype=float32)
```

```
y_pred.shape
```

```
(2995, 10)
```

```
y_test.shape
```

```
(2995, 10)
```

```
predicted_categories = np.argmax(y_pred, axis=1)
predicted_categories
```

```
array([7, 6, 6, ..., 0, 1, 4])
```

```
y_test
```

```
array([[0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

```
true_categories = np.argmax(y_test, axis=1)
true_categories
```

```
array([7, 6, 6, ..., 0, 1, 4])
```

```
classes
```

```
['blues',
 'classical',
 'country',
 'disco',
 'hiphop',
 'jazz',
 'metal',
 'pop',
 'reggae',
 'rock']
```

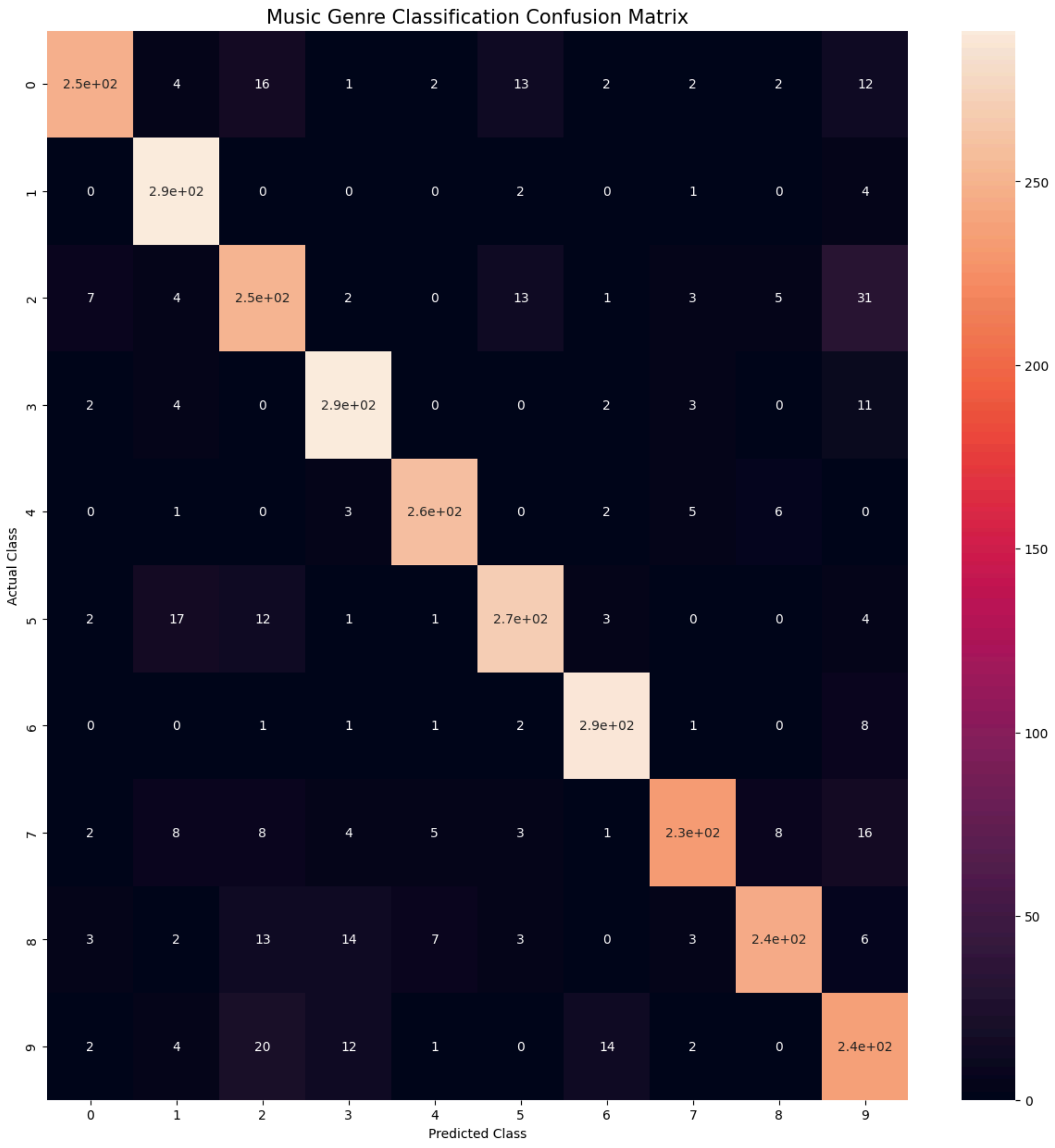
```
from sklearn.metrics import confusion_matrix,classification_report
cm = confusion_matrix(true_categories,predicted_categories)
# Precision Recall F1score
print(classification_report(true_categories,predicted_categories,target_names=cla
```

```
              precision    recall  f1-score   support

       blues       0.93      0.82      0.87       302
   classical       0.87      0.98      0.92       298
     country       0.78      0.79      0.79       317
       disco       0.88      0.93      0.91       312
      hiphop       0.94      0.94      0.94       277
```

```
        jazz       0.88      0.87      0.88       311
       metal       0.92      0.95      0.94       302
         pop       0.92      0.81      0.86       289
      reggae       0.92      0.83      0.87       296
        rock       0.72      0.81      0.76       291

    accuracy                           0.87      2995
   macro avg       0.88      0.87      0.87      2995
weighted avg       0.88      0.87      0.87      2995
```

Confusion Matrix Visualization

```
plt.figure(figsize=(15, 15))
sns.heatmap(cm,annot=True,annot_kws={"size": 10})

plt.xlabel('Predicted Class',fontsize = 10)
plt.ylabel('Actual Class',fontsize = 10)
plt.title('Music Genre Classification Confusion Matrix',fontsize = 15)
plt.show()
```



	0	1	2	3	4	5	6	7	8	9
0	2.5e+02	4	16	1	2	13	2	2	2	12
1	0	2.9e+02	0	0	0	2	0	1	0	4
2	7	4	2.5e+02	2	0	13	1	3	5	31
3	2	4	0	2.9e+02	0	0	2	3	0	11
4	0	1	0	3	2.6e+02	0	2	5	6	0
5	2	17	12	1	1	2.7e+02	3	0	0	4
6	0	0	1	1	1	2	2.9e+02	1	0	8
7	2	8	8	4	5	3	1	2.3e+02	8	16
8	3	2	13	14	7	3	0	3	2.4e+02	6
9	2	4	20	12	1	0	14	2	0	2.4e+02

Start coding or generate with AI.

RNN Model

```
# Mount Google Drive in Colab
from google.colab import drive
drive.mount('/content/drive')

# Define the dataset path
dataset_path = '/content/drive/MyDrive/Data 2/genres_original/'

# Import necessary libraries
import os
import librosa
import numpy as np

# List of genres in the dataset
genres = 'blues classical country disco hiphop jazz metal pop reggae rock'.split(

# Create lists to store file paths and labels
file_paths = []
labels = []

# Loop through each genre directory to load the files and their corresponding lab
for genre in genres:
    genre_folder = os.path.join(dataset_path, genre)
    for filename in os.listdir(genre_folder):
        if filename.endswith('.wav'):
            file_paths.append(os.path.join(genre_folder, filename))
            labels.append(genres.index(genre))  # Label is the index of the genre
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
# Function to extract MFCC features
def extract_mfcc(file_path, n_mfcc=40, max_pad_len=174):
    y, sr = librosa.load(file_path, sr=None)
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)

    # Pad or truncate the MFCC to ensure consistent length
    pad_width = max_pad_len - mfcc.shape[1]
    if pad_width > 0:
        mfcc = np.pad(mfcc, pad_width=((0, 0), (0, pad_width)), mode='constant')
    else:
        mfcc = mfcc[:, :max_pad_len]

    return mfcc

# Extract MFCC features for the entire dataset
mfcc_features = [extract_mfcc(file) for file in file_paths]

# Convert lists to numpy arrays for further processing
X = np.array(mfcc_features)
y = np.array(labels)
```

```
from sklearn.model_selection import train_test_split

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

# Reshape data for LSTM (samples, timesteps, features)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2])
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Define the LSTM model
def create_lstm_model(input_shape):
    model = Sequential()

    # First LSTM layer
    model.add(LSTM(128, return_sequences=True, input_shape=input_shape))
    model.add(Dropout(0.3))

    # Second LSTM layer
    model.add(LSTM(128, return_sequences=False))
    model.add(Dropout(0.3))

    # Fully connected layer
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.3))

    # Output layer with softmax for multi-class classification (10 genres)
    model.add(Dense(10, activation='softmax'))

    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metri
    return model

# Define the input shape for the model (timesteps, features)
input_shape = (X_train.shape[1], X_train.shape[2])

# Create and compile the model
lstm_model = create_lstm_model(input_shape)

# Train the model
history = lstm_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_d

# Save the trained model
lstm_model.save('lstm_model.h5')

# Evaluate the model
test_loss, test_accuracy = lstm_model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Epoch 1/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 5s 24ms/step - accuracy: 0.0860 - loss: 2.3275 -
Epoch 2/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.2436 - loss: 2.1689 -
Epoch 3/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.3693 - loss: 1.9533 -
Epoch 4/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.4582 - loss: 1.6640 -
Epoch 5/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.5461 - loss: 1.4330 -
Epoch 6/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.6662 - loss: 1.0192 -
Epoch 7/50
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7454 - loss: 0.7877 -
Epoch 8/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8490 - loss: 0.5434 -
Epoch 9/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8712 - loss: 0.4178 -
Epoch 10/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8782 - loss: 0.3728 - v
Epoch 11/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9100 - loss: 0.3135 -
Epoch 12/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9260 - loss: 0.2678 -
Epoch 13/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9537 - loss: 0.1848 -
Epoch 14/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9343 - loss: 0.2414 - v
Epoch 15/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9590 - loss: 0.1532 - v
Epoch 16/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9627 - loss: 0.1298 -
Epoch 17/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9739 - loss: 0.1115 - v
Epoch 18/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9384 - loss: 0.1940 -
Epoch 19/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9494 - loss: 0.1786 -
Epoch 20/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9421 - loss: 0.1979 - v
Epoch 21/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9552 - loss: 0.1627 - v
Epoch 22/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9669 - loss: 0.1405 - v
Epoch 23/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9595 - loss: 0.1383 -
Epoch 24/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9586 - loss: 0.1134 - v
Epoch 25/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9849 - loss: 0.0747 - v
Epoch 26/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9885 - loss: 0.0655 -
Epoch 27/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9729 - loss: 0.0980 -
Epoch 28/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9735 - loss: 0.0849 -
Epoch 29/50
```

Accuracy is high during training, but low during testing

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping

# Define the LSTM model
def create_lstm_model(input_shape):
    model = Sequential()

    # First LSTM layer with L2 regularization and Dropout
    model.add(LSTM(128, return_sequences=True, input_shape=input_shape,
                   kernel_regularizer=l2(0.001)))
    model.add(Dropout(0.3))

    # Second LSTM layer with L2 regularization and Dropout
    model.add(LSTM(128, return_sequences=False,
                   kernel_regularizer=l2(0.001)))
    model.add(Dropout(0.3))

    # Fully connected layer with L2 regularization and Dropout
    model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.001)))
    model.add(Dropout(0.3))

    # Output layer with softmax for multi-class classification (10 genres)
    model.add(Dense(10, activation='softmax'))

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Define the input shape for the model (timesteps, features)
input_shape = (X_train.shape[1], X_train.shape[2])

# Create and compile the model
lstm_model = create_lstm_model(input_shape)

# Add EarlyStopping callback to stop training when validation accuracy doesn't im
early_stopping = EarlyStopping(monitor='val_accuracy', patience=5, restore_best_w

# Train the model with EarlyStopping
history = lstm_model.fit(X_train, y_train, epochs=50, batch_size=32,
                         validation_data=(X_test, y_test),
                         callbacks=[early_stopping])

# Save the trained model
lstm_model.save('lstm_model.h5')

# Evaluate the model
test_loss, test_accuracy = lstm_model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Epoch 1/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 34ms/step - accuracy: 0.0957 - loss: 2.8799 - va
Epoch 2/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.2547 - loss: 2.7108 - va
Epoch 3/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.3254 - loss: 2.5005 - va
Epoch 4/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.4676 - loss: 2.1542 - va
Epoch 5/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6039 - loss: 1.7597 - va
Epoch 6/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.6929 - loss: 1.4354 - va
Epoch 7/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7822 - loss: 1.2689 - va
Epoch 8/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8677 - loss: 0.9327 - va
Epoch 9/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8850 - loss: 0.8732 - va
Epoch 10/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9134 - loss: 0.8236 - va
Epoch 11/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9444 - loss: 0.7437 - va
Epoch 12/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9306 - loss: 0.7152 - va
Epoch 13/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9569 - loss: 0.6500 - va
Epoch 14/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9652 - loss: 0.6474 - va
Epoch 15/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9672 - loss: 0.5845 - va
Epoch 16/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9687 - loss: 0.6162 - va
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.2845 - loss: 3.7574
Test Accuracy: 28.00%
```

Hybrid model

```python
# Convert to numpy arrays
X_mfcc = np.array(mfcc_features)

# Check the shape before reshaping
print(f"Before reshaping, X_mfcc shape: {X_mfcc.shape}")
```

```
Before reshaping, X_mfcc shape: (999, 40, 174)
```

```python
# Ensure MFCCs have the correct shape for LSTM (samples, timesteps, features)
X_mfcc = np.array(mfcc_features)

# If the current shape is (num_samples, n_mfcc, timesteps), we need to reshape it
if len(X_mfcc.shape) == 3:  # Ensure it's already a 3D array
    X_mfcc = X_mfcc.transpose((0, 2, 1))  # Transpose to (samples, timesteps, fea

# Check the new shape
print(f"After reshaping, X_mfcc shape: {X_mfcc.shape}")
```

```
After reshaping, X_mfcc shape: (999, 174, 40)
```

```
import os
import librosa
import numpy as np
from tensorflow.image import resize
from tensorflow.keras.utils import to_categorical


# Function to extract Mel-spectrogram
def extract_mel_spectrogram(file_path, target_shape=(128, 128)):
    audio_data, sample_rate = librosa.load(file_path, sr=None)
    mel_spectrogram = librosa.feature.melspectrogram(y=audio_data, sr=sample_rate
    mel_spectrogram = librosa.power_to_db(mel_spectrogram, ref=np.max)
    mel_spectrogram = resize(np.expand_dims(mel_spectrogram, axis=-1), target_sha
    return mel_spectrogram

# Function to extract MFCCs
def extract_mfcc(file_path, n_mfcc=40, max_pad_len=174):
    audio_data, sample_rate = librosa.load(file_path, sr=None)
    mfcc = librosa.feature.mfcc(y=audio_data, sr=sample_rate, n_mfcc=n_mfcc)

    # Padding or truncating MFCCs to fixed length
    pad_width = max_pad_len - mfcc.shape[1]
    if pad_width > 0:
        mfcc = np.pad(mfcc, pad_width=((0, 0), (0, pad_width)), mode='constant')
    else:
        mfcc = mfcc[:, :max_pad_len]

    return mfcc

# Initialize empty lists for features and labels
mel_spectrograms = []
mfcc_features = []
labels = []

# Define the genres and labels
genre_list = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal'
label_map = {genre: index for index, genre in enumerate(genre_list)}

# Define dataset path (update this path based on your Google Drive)
dataset_path = '/content/drive/MyDrive/Data 2/genres_original/'

# Loop through each genre folder and process audio files
for genre in genre_list:
    genre_folder = os.path.join(dataset_path, genre)
    for filename in os.listdir(genre_folder):
        if filename.endswith(".wav"):
            file_path = os.path.join(genre_folder, filename)

            # Extract Mel-spectrogram and MFCC
            mel_spectrogram = extract_mel_spectrogram(file_path)
            mfcc = extract_mfcc(file_path)

            # Check for valid feature extraction
            if mel_spectrogram is not None and mfcc is not None:
                mel_spectrograms.append(mel_spectrogram)
                mfcc_features.append(mfcc)
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, LSTM, Der
from tensorflow.keras.models import Model

# Convert labels to one-hot encoding (Make sure `y` is a numpy array)
y = np.array(labels)
y = to_categorical(y, num_classes=len(genre_list))

# Split the data into training and test sets using numpy arrays, not tensors
X_mel_train, X_mel_test, X_mfcc_train, X_mfcc_test, y_train, y_test = train_test_sp

# Define the CNN input (spectrogram)
cnn_input = Input(shape=(128, 128, 1))  # Example shape for Mel-spectrograms
x = Conv2D(32, kernel_size=(3, 3), activation='relu')(cnn_input)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(64, kernel_size=(3, 3), activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Flatten()(x)

# Define the LSTM input (MFCCs)
lstm_input = Input(shape=(X_mfcc_train.shape[1], X_mfcc_train.shape[2]))  # (timest
y = LSTM(128, return_sequences=True)(lstm_input)
y = LSTM(128)(y)

# Combine the CNN and LSTM branches
combined = concatenate([x, y])
```