Musical Data Analysis

Music is a form of art that is ubiquitous and has a rich history. Different composers have created music with their unique styles and compositions. However, identifying the composer of a particular piece of music can be a challenging task, especially for novice musicians or listeners. The proposed project aims to use deep learning techniques to identify the composer of a given piece of music accurately.

Objective

The primary objective of this project is to develop a deep learning model that can predict the composer of a given musical score accurately. The project aims to accomplish this objective by using two deep learning techniques: Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN).

Data set

The project will use a dataset consisting of musical scores from various composers. The dataset is downloaded from Kaggle web store https://www.kaggle.com/datasets/blanderbuss/midi-classic-music?resource=download

The dataset contains the midi files of compositions from well-known classical composers like Bach, Beethoven, Chopin, and Mozart. The dataset should be labeled with the name of the composer for each score.

1-Bach 2-Beethoven 3-Chopin 4-Mozart

Data Collection And Pre-processing

The data is downloaded from the Kaggle webstore. It has multiple MIDI files from multiple classical composers.

Method to collect MID files for each top folder and subfolders

We are considering only Bach, Beethoven, Chopin and Mozart.

In [1]:

```
import os
import glob
import warnings

warnings.filterwarnings("ignore")

def collect_midi_files(root_folder):
    # Use a set to store unique absolute paths
    unique_files = set()
    midi_files = []

    # Use glob to find all .mid files in root_folder and subfolders
    relative_paths = glob.glob(os.path.join(root_folder, '**', '*.mid'), recursive=True)

    for relative_path in relative_paths:
        absolute_path = os.path.abspath(relative_path)
        # Check if the absolute path is already in the set
        if absolute_path not in unique_files:
            unique_files.add(absolute_path)
            midi_files.append(absolute_path)

    return midi_files

# Define the root folder
root_folder = '/Users/manikanr/Downloads/archive/midiclassics/Bach'

# Collect all unique .mid files with absolute paths
midi_files = collect_midi_files(root_folder)
```

```
print(len(midi_files))

# Print the list of .mid files with absolute paths
#for midi_file in midi_files:
#    print(midi_file)
```

```
925
```

Collect Bach MID files under Bach folder and sub-folders

In [2]:

```
# Define the root folder
root_folder = '/Users/manikanr/Downloads/archive/midiclassics/Bach'

# Collect all unique .mid files with absolute paths
bach_midi_files = collect_midi_files(root_folder)
print("The number of mid files under Bach folder and subfolders are {}".format(len(bach_m
idi_files)))
```

```
The number of mid files under Bach folder and subfolders are 925
```

Collect Beethoven MID files under Beethoven folder and sub-folders

In [3]:

```
# Define the root folder
root_folder = '/Users/manikanr/Downloads/archive/midiclassics/Beethoven'

# Collect all unique .mid files with absolute paths
beethoven_midi_files = collect_midi_files(root_folder)
print("The number of mid files under Beethoven folder and subfolders are {}".format(len(b
eethoven_midi_files)))
```

```
The number of mid files under Beethoven folder and subfolders are 212
```

Collect Chopin MID files under Chopin folder and sub-folders

In [4]:

```
# Define the root folder
root_folder = '/Users/manikanr/Downloads/archive/midiclassics/Chopin'

# Collect all unique .mid files with absolute paths
chopin_midi_files = collect_midi_files(root_folder)
print("The number of mid files under Chopin folder and subfolders are {}".format(len(chop
in_midi_files)))
```

```
The number of mid files under Chopin folder and subfolders are 136
```

Collect Mozart MID files under Mozart folder and sub-folders

In [5]:

```
# Define the root folder
root_folder = '/Users/manikanr/Downloads/archive/midiclassics/Mozart'

# Collect all unique .mid files with absolute paths
mozart_midi_files = collect_midi_files(root_folder)
print("The number of mid files under Mozart folder and subfolders are {}".format(len(moza
rt_midi_files)))
```

```
The number of mid files under Mozart folder and subfolders are 257
```

Avoid overfitting for Bach

Bach has 957 files and Chopin has only 136. We don't want tthe model to be overfit for Bach during training. So

Bach has 357 files and Chopin has only 136. We don't want the model to be overfit for Bach during training. So, the base of 136 MID files will be used to fit our models for all 4 composers.

Analysis of Python Music Libraries to use for Feature Extraction

There are many python music libraries available to work with MID files. But, the below 2 libraries stands out for classical music analysis.

1. **Music 21 - Music21 provides robust feature extraction tools to split notes, chords, tempo, key, time signatures and Rhythmic patterns.**
2. **PrettyMIDI - Equally good tool but doesn't provide direct method to get Chords.**

Due to its robust features and tools, the Music21 is used in our project.

Using Music21 for Feature Extraction

Music21 has libraries like converter which converts entire MID file into a stream of musical score. This stream has data about notes, chords, tempo, key and time signatures and rhythmic patterns. the notes, chords, tempo etc can be extracted from this stream as features.

Method to extract notes, chords, tempo, key, time signatures and rhythmic patterns.

Music21 has libraries for splitting MID files with above data.

Using Music21 on single MID file to check on features

In [6]:

```
from music21 import converter, note, chord, metadata, tempo, key, meter

# Load the MIDI file
score = converter.parse('/Users/manikanr/Downloads/archive/midiclassics/Bach/Bwv0525 Sona
te en trio n1.mid')

# Extract Notes
notes = []
chords = []
tempos = []
rhythmic_patterns = []
time_signatures = []

# Extract Notes, Chords, and Rhythmic Patterns
for element in score.flat:
   if isinstance(element, note.Note):
        notes.append([element.offset, element.pitch.midi, element.quarterLength, element.
volume.realized])
        rhythmic_patterns.append([element.offset, element.quarterLength])
   elif isinstance(element, chord.Chord):
        chords.append([element.offset] + [p.midi for p in element.pitches])

# Extract Tempo
for elem in score.flat.getElementsByClass(tempo.MetronomeMark):
    tempos.append([elem.offset, elem.number])

# Extract Time Signature
for elem in score.flat.getElementsByClass(meter.TimeSignature):
    time_signatures.append([elem.offset, elem.numerator, elem.denominator])
```

Padding for Number of Rows and Columns

The resulting arrays of notes, chords, tempos, rhythmic patterns would be of different sizes. This should be padded so that all features have same array length. Sometimes the notes and chords would return Fraction values. These should be coverted and padded accordingly. The below methods are used for that.

In [7]:

```
import numpy as np

from fractions import Fraction

def convert_to_float(value):
    if isinstance(value, Fraction):
        return float(value)
    return float(value)

def pad_chord(chord_list, max_notes=4, pad_value=0):
    offset = convert_to_float(chord_list[0])
    notes = chord_list[1:]
    notes = notes + [pad_value] * (max_notes - len(notes))
    return [offset] + notes

def pad_array(array, max_len, pad_value=0):
    padded_array = []
    for row in array:
        if len(row) < max_len:
            row = row + [pad_value] * (max_len - len(row))
        padded_array.append(row)
    return np.array(padded_array, dtype=float)
```

Method to extract the notes, chords, tempos, rhythmic pattern features

In [8]:

```
from music21 import converter, note, chord, tempo, meter
import numpy as np

def extract_features(score, max_notes=4):
    notes = []
    chords = []
    tempos = []
    rhythmic_patterns = []
    time_signatures = []

    # Extract Notes, Chords, and Rhythmic Patterns
    for element in score.flat:
        if isinstance(element, note.Note):
            notes.append([
                convert_to_float(element.offset),
                element.pitch.midi,
                convert_to_float(element.quarterLength),
                element.volume.realized
            ])
            rhythmic_patterns.append([
                convert_to_float(element.offset),
                convert_to_float(element.quarterLength)
            ])
        elif isinstance(element, chord.Chord):
            raw_chord = [
                convert_to_float(element.offset)
            ] + [p.midi for p in element.pitches]
            chords.append(pad_chord(raw_chord, max_notes=max_notes))

    # Extract Tempo
    for elem in score.flat.getElementsByClass(tempo.MetronomeMark):
        tempos.append([
            convert_to_float(elem.offset),
            elem.number
        ])

    # Extract Time Signature
    for elem in score.flat.getElementsByClass(meter.TimeSignature):
        time_signatures.append([
            convert_to_float(elem.offset),
            elem.numerator,
```

```
            elem.denominator
        ])

    # Convert lists to numpy arrays and pad to maximum length
    max_len_notes = max((len(row) for row in notes), default=0)
    max_len_chords = max((len(row) for row in chords), default=0)
    max_len_tempos = max((len(row) for row in tempos), default=0)
    max_len_rhythmic_patterns = max((len(row) for row in rhythmic_patterns), default=0)
    max_len_time_signatures = max((len(row) for row in time_signatures), default=0)

    max_len = max(max_len_notes, max_len_chords, max_len_tempos, max_len_rhythmic_pattern
s, max_len_time_signatures)

    # Pad arrays to ensure they all have the same number of columns
    notes_array = pad_array(notes, max_len)
    chords_array = pad_array(chords, max_len)
    tempos_array = pad_array(tempos, max_len)
    rhythmic_patterns_array = pad_array(rhythmic_patterns, max_len)
    time_signatures_array = pad_array(time_signatures, max_len)

    # Ensure all arrays have the same number of rows
    max_rows = min(len(notes_array), len(chords_array), len(tempos_array), len(rhythmic_
patterns_array), len(time_signatures_array))

    notes_array = notes_array[:max_rows]
    chords_array = chords_array[:max_rows]
    tempos_array = tempos_array[:max_rows]
    rhythmic_patterns_array = rhythmic_patterns_array[:max_rows]
    time_signatures_array = time_signatures_array[:max_rows]

    # Ensure that each array has the same number of dimensions
    def ensure_2d(array):
        if array.ndim == 1:
            return array.reshape(-1, 1)
        return array

    notes_array = ensure_2d(notes_array)
    chords_array = ensure_2d(chords_array)
    tempos_array = ensure_2d(tempos_array)
    rhythmic_patterns_array = ensure_2d(rhythmic_patterns_array)
    time_signatures_array = ensure_2d(time_signatures_array)

    # Combine features into one array
    combined_features = np.hstack((notes_array, chords_array, tempos_array, rhythmic_pat
terns_array, time_signatures_array))

    return combined_features

# Example usage for single midi file
midi_file = '/Users/manikanr/Downloads/archive/midiclassics/Bach/Bwv0997 Partita for Lute
1mov.mid'  # Replace with your MIDI file path
score = converter.parse(midi_file)
features = extract_features(score)
print(features)
```

```
[[  0.          48.           1.           0.78740157   0.
   49.5         82.          81.          82.           0.
    0.          80.           0.           0.           0.
    0.           1.           0.           0.           0.
    0.           4.           4.           0.           0.        ]
 [  0.5         72.           0.25         0.78740157   0.
  127.5         80.          79.          80.          79.
    0.          80.           0.           0.           0.
    0.5          0.25         0.           0.           0.
    0.           4.           4.           0.           0.        ]]
```

Method to collect Bach composer features

In [9]:

```
def collectFeatures(composerName, composerMIDFiles):
```

```
def collectFeatures(composerName, composerMIDFiles):
    features = []
    labels = []  # Composer names or folder names
    i=1
    used_midi_files = []

    for midi_file in composerMIDFiles:
        #print(midi_file)
        if (i==136): # collect only 136 files
            return np.array(features), np.array(labels), np.array(used_midi_files)
        try:
            score = converter.parse(midi_file)
            feature_array = extract_features(score)
            features.append(feature_array)
            labels.append(composerName)
            used_midi_files.append(midi_file)
            i=i+1
        except Exception as e:
            #print("Entered Exception for "+midi_file)
            continue
   # Assign label based on folder name

    return np.array(features), np.array(labels), np.array(used_midi_files)
```

Collect features and labels for Bach

In [10]:

```
bach_features, bach_labels, bach_used_midi_files = collectFeatures("Bach", bach_midi_file
s)
print(len(bach_features))
print(len(bach_labels))
```

```
135
135
```

Collect features and labels for Beethoven

In [11]:

```
beethoven_features, beethoven_labels, beethoven_used_midi_files = collectFeatures("Beetho
ven", beethoven_midi_files)
print(len(beethoven_features))
print(len(beethoven_labels))
```

```
135
135
```

Collect features and labels for Chopin

In [12]:

```
chopin_features, chopin_labels, chopin_used_midi_files = collectFeatures("Chopin", chopin
_midi_files)
print(len(chopin_features))
print(len(chopin_labels))
```

```
135
135
```

Collect features and labels for Mozart

In [13]:

```
mozart_features, mozart_labels, mozart_used_midi_files = collectFeatures("Mozart", mozart
_midi_files)
print(len(mozart_features))
```

```
print(len(mozart_labels))
```

```
135
135
```

Data Preparation for Convolutional Neural Networks (CNN)

Normalize, flatten and re-shape data before applying to CNN.

In [14]:

```
from sklearn.preprocessing import StandardScaler

def pad_array_3d(array, max_len, pad_value=0.0):
    """ Pad each 2D array in the 3D array to ensure they have consistent shapes. """
    padded_array = []
    max_cols = max(sample.shape[1] for sample in array)  # Find the maximum number of co
lumns

    for sample in array:
        num_rows, num_cols = sample.shape
        # Initialize a new array filled with the pad value, ensuring it has consistent sh
ape
        padded_sample = np.full((max_len, max_cols), pad_value)
        # Copy the data into the padded array
        padded_sample[:num_rows, :num_cols] = sample
        padded_array.append(padded_sample)

    return np.array(padded_array)


# Determine the maximum number of rows in any 2D array (sample)
max_len = 0
if max(sample.shape[0] for sample in bach_features) > max_len:
    max_len = max(sample.shape[0] for sample in bach_features)
if max(sample.shape[0] for sample in beethoven_features) > max_len:
    max_len = max(sample.shape[0] for sample in beethoven_features)
if max(sample.shape[0] for sample in chopin_features) > max_len:
    max_len = max(sample.shape[0] for sample in chopin_features)
if max(sample.shape[0] for sample in mozart_features) > max_len:
    max_len = max(sample.shape[0] for sample in mozart_features)

print(max_len)

def normalizeFeatures(features, max_len):
   # Pad each sample to have the same number of rows
   features_padded = pad_array_3d(features, max_len)
   # Flatten each 2D array in bach_features_padded to 1D
   features_flattened = features_padded.reshape(features_padded.shape[0], -1)

   # Apply StandardScaler to the flattened features
   scaler = StandardScaler()
   features_scaled = scaler.fit_transform(features_flattened)

   # If necessary, reshape back to 3D for further processing
   features_reshaped = features_scaled.reshape(features_padded.shape)
   return features_reshaped

#bach_features = scaler.fit_transform(bach_features.reshape(bach_features.shape[0], -1))
# Flatten features if needed
#bach_features = bach_features.reshape(bach_features.shape[0], height, width, channels)
# Reshape for CNN

# Split data into training and testing sets
#from sklearn.model_selection import train_test_split
#X_train, X_test, y_train, y_test = train_test_split(bach_features_reshaped, bach_labels,
test_size=0.2, random_state=42)
```

```
168
```

Normalize for all 4 composers

In [15]:

```
bach_features_reshaped = normalizeFeatures(bach_features, max_len)
beethoven_features_reshaped = normalizeFeatures(beethoven_features, max_len)
chopin_features_reshaped = normalizeFeatures(chopin_features, max_len)
mozart_features_reshaped = normalizeFeatures(mozart_features, max_len)

def pad_to_max_columns(features, max_columns, pad_value=0.0):
    """ Pad the feature arrays to have the same number of columns. """
    padded_features = []
    for sample in features:
        num_rows, num_cols = sample.shape
        padded_sample = np.full((num_rows, max_columns), pad_value)
        padded_sample[:, :num_cols] = sample
        padded_features.append(padded_sample)
    return np.array(padded_features)

# Calculate the maximum number of columns across all datasets
max_columns = max(
    bach_features_reshaped.shape[2],
    beethoven_features_reshaped.shape[2],
    chopin_features_reshaped.shape[2],
    mozart_features_reshaped.shape[2]
)

# Pad each feature set to have the same number of columns
bach_features_padded = pad_to_max_columns(bach_features_reshaped, max_columns)
beethoven_features_padded = pad_to_max_columns(beethoven_features_reshaped, max_columns)
chopin_features_padded = pad_to_max_columns(chopin_features_reshaped, max_columns)
mozart_features_padded = pad_to_max_columns(mozart_features_reshaped, max_columns)

# Print the shapes to verify
print(bach_features_padded.shape)
print(beethoven_features_padded.shape)
print(chopin_features_padded.shape)
print(mozart_features_padded.shape)
```

```
(135, 168, 85)
(135, 168, 85)
(135, 168, 85)
(135, 168, 85)
```

Train-Test Split for all 4 composers

1. **Do the train-test split for each composer separately. This is to create uniformity in training models with CNN.**
2. **Combine each one to get overall train and test sets.**

In [16]:

```
from sklearn.model_selection import train_test_split

# Do the train-test split for each composer separately.
bach_x_train, bach_x_test, bach_y_train, bach_y_test = train_test_split(bach_features_pa
dded, bach_labels, test_size=0.2, random_state=42)
beethoven_x_train, beethoven_x_test, beethoven_y_train, beethoven_y_test = train_test_sp
lit(beethoven_features_padded, beethoven_labels, test_size=0.2, random_state=42)
chopin_x_train, chopin_x_test, chopin_y_train, chopin_y_test = train_test_split(chopin_fe
atures_padded, chopin_labels, test_size=0.2, random_state=42)
mozart_x_train, mozart_x_test, mozart_y_train, mozart_y_test = train_test_split(mozart_fe
atures_padded, mozart_labels, test_size=0.2, random_state=42)

#Combine the train-test split now
x_train_combined = np.concatenate(
    [bach_x_train, beethoven_x_train, chopin_x_train, mozart_x_train], axis=0
)
print(x_train_combined.shape)
```

```
# Concatenate testing features
x_test_combined = np.concatenate(
    [bach_x_test, beethoven_x_test, chopin_x_test, mozart_x_test], axis=0
)
print(x_test_combined.shape)

# Concatenate training labels
y_train_combined = np.concatenate(
    [bach_y_train, beethoven_y_train, chopin_y_train, mozart_y_train], axis=0
)
print(y_train_combined.shape)

# Concatenate testing labels
y_test_combined = np.concatenate(
    [bach_y_test, beethoven_y_test, chopin_y_test, mozart_y_test], axis=0
)
print(y_test_combined.shape)
```

```
(432, 168, 85)
(108, 168, 85)
(432,)
(108,)
```

Encoding the classes

In y to values such as Bach to 0, Beethoven to 1, Chopin to 2 and Mozart to 4

In [21]:

```
# Define your manual encoding
label_mapping = {
    "Bach": 0,
    "Beethoven": 1,
    "Chopin": 2,
    "Mozart": 3
}

# Encode the labels
y_train_combined_encoded = [label_mapping[label] for label in y_train_combined]
y_test_combined_encoded = [label_mapping[label] for label in y_test_combined]

# Convert the labels to numpy arrays
y_train_combined_encoded = np.array(y_train_combined_encoded)
y_test_combined_encoded = np.array(y_test_combined_encoded)

# Check shapes to confirm everything is correctly formatted
print(x_train_combined.shape, y_train_combined_encoded.shape)
print(x_test_combined.shape, y_test_combined_encoded.shape)
```

```
(432, 168, 85) (432,)
(108, 168, 85) (108,)
```

Model Design for Convolutional Neural Networks (CNN)

In [26]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

num_classes = 4 # Since we have data of 4 composers
# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(168, 85, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
```

```
    Flatten(),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')  # num_classes = number of output classes
])

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

# Print model summary
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_21 (Conv2D)          (None, 166, 83, 32)       320       
                                                                 
 max_pooling2d_14 (MaxPooli  (None, 83, 41, 32)        0         
 ng2D)                                                           
                                                                 
 conv2d_22 (Conv2D)          (None, 81, 39, 64)        18496     
                                                                 
 max_pooling2d_15 (MaxPooli  (None, 40, 19, 64)        0         
 ng2D)                                                           
                                                                 
 conv2d_23 (Conv2D)          (None, 38, 17, 128)       73856     
                                                                 
 flatten_7 (Flatten)         (None, 82688)             0         
                                                                 
 dense_14 (Dense)            (None, 64)                5292096   
                                                                 
 dense_15 (Dense)            (None, 4)                 260       
                                                                 
=================================================================
Total params: 5385028 (20.54 MB)
Trainable params: 5385028 (20.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model Training for Convolutional Neural Networks (CNN)

Training the CNN model for 10 epochs with training and validation data

In [27]:

```
history = model.fit(x_train_combined, y_train_combined_encoded, epochs=10, batch_size=32
, validation_data=(x_test_combined, y_test_combined_encoded))
```

```
Epoch 1/10
14/14 [==============================] - 4s 202ms/step - loss: 0.8624 - accuracy: 0.7431
- val_loss: 0.0905 - val_accuracy: 0.9907
Epoch 2/10
14/14 [==============================] - 3s 180ms/step - loss: 0.0512 - accuracy: 0.9861
- val_loss: 0.0012 - val_accuracy: 1.0000
Epoch 3/10
14/14 [==============================] - 3s 180ms/step - loss: 0.0027 - accuracy: 1.0000
- val_loss: 0.0017 - val_accuracy: 1.0000
Epoch 4/10
14/14 [==============================] - 3s 186ms/step - loss: 4.3778e-04 - accuracy: 1.0
000 - val_loss: 3.9657e-04 - val_accuracy: 1.0000
Epoch 5/10
14/14 [==============================] - 3s 182ms/step - loss: 2.1175e-04 - accuracy: 1.0
000 - val_loss: 3.7954e-05 - val_accuracy: 1.0000
Epoch 6/10
14/14 [==============================] - 3s 184ms/step - loss: 2.3190e-05 - accuracy: 1.0
000 - val_loss: 1.3690e-05 - val_accuracy: 1.0000
Epoch 7/10
14/14 [==============================] - 3s 196ms/step - loss: 1.0371e-05 - accuracy: 1.0
000 - val_loss: 1.3042e-05 - val_accuracy: 1.0000
```

```
Epoch 8/10
14/14 [==============================] - 3s 193ms/step - loss: 9.7132e-06 - accuracy: 1.0
000 - val_loss: 1.2811e-05 - val_accuracy: 1.0000
Epoch 9/10
14/14 [==============================] - 3s 182ms/step - loss: 9.4442e-06 - accuracy: 1.0
000 - val_loss: 1.2408e-05 - val_accuracy: 1.0000
Epoch 10/10
14/14 [==============================] - 3s 184ms/step - loss: 9.1244e-06 - accuracy: 1.0
000 - val_loss: 1.1985e-05 - val_accuracy: 1.0000
```

Model Evaluation for Convolutional Neural Networks (CNN)

In [28]:

```
from sklearn.metrics import classification_report

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test_combined, y_test_combined_encoded)
print(f"Test accuracy: {test_acc}")

# Make predictions
predictions = model.predict(x_test_combined)
predicted_classes = np.argmax(predictions, axis=1)
print("========================================\n")
print("        Expected Result                  ")
print("========================================\n")
print(y_test_combined_encoded)
print("\n")

print("========================================\n")
print("        Actual Result                  ")
print("========================================\n")
print(predicted_classes)
print("\n")

# Generate the classification report
report = classification_report(y_test_combined_encoded, predicted_classes)

print("========================================\n")
print("   CLASSIFICATION REPORT               \n")
print("========================================\n")
# Print the classification report
print(report)
```

```
4/4 [==============================] - 0s 59ms/step - loss: 1.1985e-05 - accuracy: 1.0000
Test accuracy: 1.0
4/4 [==============================] - 0s 43ms/step
========================================

        Expected Result
========================================

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]


========================================

        Actual Result
========================================

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]


========================================

   CLASSIFICATION REPORT
```

```
========================================

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        27
           1       1.00      1.00      1.00        27
           2       1.00      1.00      1.00        27
           3       1.00      1.00      1.00        27

    accuracy                           1.00       108
   macro avg       1.00      1.00      1.00       108
weighted avg       1.00      1.00      1.00       108
```

Interpretation

The report indicates that model has achieved perfect accuracy on the test set, correctly classifying every sample across all four classes. This result suggests that the model performs exceptionally well on this dataset. However, it's also essential to be cautious, as perfect performance may sometimes indicate overfitting, especially if the model's training and validation accuracy were also near perfect. It's a good idea to verify this performance on a separate test set or through cross-validation to ensure the model generalizes well to new, unseen data. This unseen data set testing is done below.

Testing with unseen mid files from each of these composers using above CNN model

This is to see the model performance on totally unseen data from all 4 music composers using CNN.

Totally Unseen files testing

In [72]:

```
import warnings

warnings.filterwarnings("ignore")

bach_new_files = ['/Users/manikanr/Downloads/archive/midiclassics/Bach/Concertos/Bwv1047
Brandenburg Concert n2 1mov.mid',
                  '/Users/manikanr/Downloads/archive/midiclassics/Mozart/K299 Flute Harp
Concerto 1mov.mid',
                  '/Users/manikanr/Downloads/archive/midiclassics/Chopin/Piano Concerto
n1 op11 1mov.mid',
                  '/Users/manikanr/Downloads/archive/midiclassics/Beethoven/Bagatella Fu
r Elise.mid']

# Check if files in bach_new_files are in already validated files
for file in bach_new_files:
    if file in bach_used_midi_files:
        print(f"{file} is in bach_used_midi_files.")
    else:
        print(f"{file} is NOT in bach_used_midi_files.")

bach_new_features, bach_new_labels, bach_new_used_files = collectFeatures("Bach", bach_n
ew_files)
max_len = max(sample.shape[0] for sample in bach_new_features)
bach_new_features_reshaped = normalizeFeatures(bach_new_features, 168)

# Pad each feature set to have the same number of columns
bach_new_features_padded = pad_to_max_columns(bach_new_features_reshaped, 85)
print(bach_new_features_padded.shape)
'''
# New data shape: (5, 20, 40)
bach_new_shape = np.random.rand(5, 20, 40)  # Example data

# Expected input shape: (168, 85, 1)
expected_shape = (168, 85, 1)
```

```
# Reshape new data to match expected input shape
# This example uses padding with zeros to achieve the desired shape

padded_data = np.zeros((5, *expected_shape))  # Initialize with zeros

# Insert the original data into the padded array
# This assumes you want to place the new data in the top-left corner
padded_data[:, :20, :40, 0] = bach_new_shape

# Now padded_data should have the shape (5, 168, 85, 1)
print(padded_data.shape)  # Should output (5, 168, 85, 1)
'''
# Make predictions
predictions = model.predict(bach_new_features_padded)
predicted_classes = np.argmax(predictions, axis=1)
print(predicted_classes)
```

```
/Users/manikanr/Downloads/archive/midiclassics/Bach/Concertos/Bwv1047 Brandenburg Concert
n2 1mov.mid is NOT in bach_used_midi_files.
/Users/manikanr/Downloads/archive/midiclassics/Mozart/K299 Flute Harp Concerto 1mov.mid i
s NOT in bach_used_midi_files.
/Users/manikanr/Downloads/archive/midiclassics/Chopin/Piano Concerto n1 op11 1mov.mid is
NOT in bach_used_midi_files.
/Users/manikanr/Downloads/archive/midiclassics/Beethoven/Bagatella Fur Elise.mid is NOT i
n bach_used_midi_files.
(4, 168, 85)
1/1 [==============================] - 0s 21ms/step
[2 3 3 3]
```

From above without re-training the model on new data, we can say it incorrectly predics few files. After all, our model is not overfit then.

Hyper Parameter Tuning for CNN

Since we got 100% in firly 5 epochs, let's reduce the number of epochs to 5. Also, let's remove two dense hidden layer to see if we can get same performance.

In [82]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

num_classes = 4 # Since we have data of 4 composers
# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(168, 85, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')  # num_classes = number of output classes
])

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

# Print model summary
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_26 (Conv2D)          (None, 166, 83, 32)       320

 max_pooling2d_17 (MaxPooli  (None, 83, 41, 32)        0
 ng2D)
```

```
 flatten_9 (Flatten)         (None, 108896)          0

 dense_20 (Dense)            (None, 64)              6969408

 dense_21 (Dense)            (None, 4)               260

=================================================================
Total params: 6969988 (26.59 MB)
Trainable params: 6969988 (26.59 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Check for Performance With 5 Epochs

In [83]:

```
history = model.fit(x_train_combined, y_train_combined_encoded, epochs=5, batch_size=10,
validation_data=(x_test_combined, y_test_combined_encoded))
```

```
Epoch 1/5
44/44 [==============================] - 3s 60ms/step - loss: 1.7664 - accuracy: 0.8866 -
val_loss: 0.7571 - val_accuracy: 0.9444
Epoch 2/5
44/44 [==============================] - 3s 59ms/step - loss: 0.1109 - accuracy: 0.9884 -
val_loss: 0.8165 - val_accuracy: 0.9352
Epoch 3/5
44/44 [==============================] - 3s 60ms/step - loss: 0.0288 - accuracy: 0.9954 -
val_loss: 0.4006 - val_accuracy: 0.9722
Epoch 4/5
44/44 [==============================] - 3s 57ms/step - loss: 0.0589 - accuracy: 0.9954 -
val_loss: 0.5663 - val_accuracy: 0.9444
Epoch 5/5
44/44 [==============================] - 3s 60ms/step - loss: 7.3938e-04 - accuracy: 1.00
00 - val_loss: 0.5994 - val_accuracy: 0.9444
```

In [84]:

```
from sklearn.metrics import classification_report

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test_combined, y_test_combined_encoded)
print(f"Test accuracy: {test_acc}")

# Make predictions
predictions = model.predict(x_test_combined)
predicted_classes = np.argmax(predictions, axis=1)
print("========================================\n")
print("        Expected Result                    ")
print("========================================\n")
print(y_test_combined_encoded)
print("\n")

print("========================================\n")
print("        Actual Result                    ")
print("========================================\n")
print(predicted_classes)
print("\n")

# Generate the classification report
report = classification_report(y_test_combined_encoded, predicted_classes)

print("========================================\n")
print("   CLASSIFICATION REPORT                \n")
print("========================================\n")
# Print the classification report
print(report)
```

```
4/4 [==============================] - 0s 22ms/step - loss: 0.5994 - accuracy: 0.9444
Test accuracy: 0.9444444179534912
4/4 [==============================] - 0s 22ms/step
========================================
```

```
        Expected Result
========================================

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]


========================================

        Actual Result
========================================

[3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 1 1 1 1 1 1 1 1 2 1
 1 1 1 1 1 1 1 1 3 1 1 1 1 1 0 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]


========================================

   CLASSIFICATION REPORT

========================================

              precision    recall  f1-score   support

           0       0.96      0.93      0.94        27
           1       1.00      0.85      0.92        27
           2       0.93      1.00      0.96        27
           3       0.90      1.00      0.95        27

    accuracy                           0.94       108
   macro avg       0.95      0.94      0.94       108
weighted avg       0.95      0.94      0.94       108
```

Since removal of 2 dense layers results in reduced performance, lets add one more dense layer.

Hyper-parameter tuning by adding 1 more layer

In [85]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

num_classes = 4 # Since we have data of 4 composers
# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(168, 85, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')  # num_classes = number of output classes
])

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

# Print model summary
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 conv2d_27 (Conv2D)          (None, 166, 83, 32)       320

 max_pooling2d_18 (MaxPooli  (None, 83, 41, 32)        0
 ng2D)

 conv2d_28 (Conv2D)          (None, 81, 39, 64)        18496

 max_pooling2d_19 (MaxPooli  (None, 40, 19, 64)        0
 ng2D)

 flatten_10 (Flatten)        (None, 48640)             0

 dense_22 (Dense)            (None, 64)                3113024

 dense_23 (Dense)            (None, 4)                 260

=================================================================
Total params: 3132100 (11.95 MB)
Trainable params: 3132100 (11.95 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Evaluate Tuned CNN Model

In [87]:

```
history = model.fit(x_train_combined, y_train_combined_encoded, epochs=5, batch_size=10,
validation_data=(x_test_combined, y_test_combined_encoded))
```

```
Epoch 1/5
44/44 [==============================] - 4s 69ms/step - loss: 0.3007 - accuracy: 0.8796 -
val_loss: 0.0061 - val_accuracy: 1.0000
Epoch 2/5
44/44 [==============================] - 3s 65ms/step - loss: 0.0036 - accuracy: 0.9977 -
val_loss: 9.7905e-07 - val_accuracy: 1.0000
Epoch 3/5
44/44 [==============================] - 3s 66ms/step - loss: 9.1807e-07 - accuracy: 1.00
00 - val_loss: 8.4881e-07 - val_accuracy: 1.0000
Epoch 4/5
44/44 [==============================] - 3s 70ms/step - loss: 9.3214e-07 - accuracy: 1.00
00 - val_loss: 8.4660e-07 - val_accuracy: 1.0000
Epoch 5/5
44/44 [==============================] - 3s 65ms/step - loss: 9.2000e-07 - accuracy: 1.00
00 - val_loss: 8.4881e-07 - val_accuracy: 1.0000
```

In [88]:

```
from sklearn.metrics import classification_report

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test_combined, y_test_combined_encoded)
print(f"Test accuracy: {test_acc}")

# Make predictions
predictions = model.predict(x_test_combined)
predicted_classes = np.argmax(predictions, axis=1)
print("========================================\n")
print("        Expected Result                 ")
print("========================================\n")
print(y_test_combined_encoded)
print("\n")

print("========================================\n")
print("        Actual Result                 ")
print("========================================\n")
print(predicted_classes)
print("\n")

# Generate the classification report
```

```
report = classification_report(y_test_combined_encoded, predicted_classes)

print("========================================\n")
print("   CLASSIFICATION REPORT                \n")
print("========================================\n")
# Print the classification report
print(report)
```

```
4/4 [==============================] - 0s 34ms/step - loss: 8.4881e-07 - accuracy: 1.0000
Test accuracy: 1.0
4/4 [==============================] - 0s 35ms/step
========================================

        Expected Result
========================================

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]


========================================

        Actual Result
========================================

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]


========================================

   CLASSIFICATION REPORT

========================================

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        27
           1       1.00      1.00      1.00        27
           2       1.00      1.00      1.00        27
           3       1.00      1.00      1.00        27

    accuracy                           1.00       108
   macro avg       1.00      1.00      1.00       108
weighted avg       1.00      1.00      1.00       108
```

In [89]:

```
import warnings

warnings.filterwarnings("ignore")

bach_new_files = ['/Users/manikanr/Downloads/archive/midiclassics/Bach/Concertos/Bwv1047 Brandenburg Concert n2 1mov.mid',
                  '/Users/manikanr/Downloads/archive/midiclassics/Mozart/K299 Flute Harp Concerto 1mov.mid',
                  '/Users/manikanr/Downloads/archive/midiclassics/Chopin/Piano Concerto n1 op11 1mov.mid',
                  '/Users/manikanr/Downloads/archive/midiclassics/Beethoven/Bagatella Fur Elise.mid']

# Check if files in bach_new_files are in already validated files
for file in bach_new_files:
    if file in bach_used_midi_files:
        print(f"{file} is in bach_used_midi_files.")
    else:
        print(f"{file} is NOT in bach_used_midi_files.")

bach_new_features, bach_new_labels, bach_new_used_files = collectFeatures("Bach", bach_n
```

```
ew_files)
max_len = max(sample.shape[0] for sample in bach_new_features)
bach_new_features_reshaped = normalizeFeatures(bach_new_features, 168)

# Pad each feature set to have the same number of columns
bach_new_features_padded = pad_to_max_columns(bach_new_features_reshaped, 85)
print(bach_new_features_padded.shape)
'''
# New data shape: (5, 20, 40)
bach_new_shape = np.random.rand(5, 20, 40)  # Example data

# Expected input shape: (168, 85, 1)
expected_shape = (168, 85, 1)

# Reshape new data to match expected input shape
# This example uses padding with zeros to achieve the desired shape

padded_data = np.zeros((5, *expected_shape))  # Initialize with zeros

# Insert the original data into the padded array
# This assumes you want to place the new data in the top-left corner
padded_data[:, :20, :40, 0] = bach_new_shape

# Now padded_data should have the shape (5, 168, 85, 1)
print(padded_data.shape)  # Should output (5, 168, 85, 1)
'''
# Make predictions
predictions = model.predict(bach_new_features_padded)
predicted_classes = np.argmax(predictions, axis=1)
print(predicted_classes)
```

```
/Users/manikanr/Downloads/archive/midiclassics/Bach/Concertos/Bwv1047 Brandenburg Concert
n2 1mov.mid is NOT in bach_used_midi_files.
/Users/manikanr/Downloads/archive/midiclassics/Mozart/K299 Flute Harp Concerto 1mov.mid i
s NOT in bach_used_midi_files.
/Users/manikanr/Downloads/archive/midiclassics/Chopin/Piano Concerto n1 op11 1mov.mid is
NOT in bach_used_midi_files.
/Users/manikanr/Downloads/archive/midiclassics/Beethoven/Bagatella Fur Elise.mid is NOT i
n bach_used_midi_files.
(4, 168, 85)
1/1 [==============================] - 0s 21ms/step
[2 3 3 3]
```

Conclusion and Results for Convolutional Neural Networks (CNN)

The model has achieved perfect performance on this dataset, with a 100% accuracy, precision, recall, and F1-score across all classes. This is an ideal outcome and suggests that the model has learned to distinguish between the different classes perfectly, at least on the test set provided. However, such perfect scores could sometimes indicate that the model might be overfitting, especially if the dataset is small or not very diverse. But, unseen data provides indifferent results which means more training data set of each of these composers is required. Also, it shows finding differences in music and rhythmic patterns is not an easy task.

Model Creation for LSTM

In [90]:

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

# Define the input shape
timesteps = 168  # Number of time steps in the sequence (length of each sequence)
input_dim = 85   # Number of features per time step (dimension of each input vector)
num_classes = 4  # Number of output classes (e.g., 4 composers)

# Define the LSTM model
model = Sequential([
```

```
    LSTM(128, input_shape=(timesteps, input_dim), return_sequences=True),
    Dropout(0.2),
    LSTM(128),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')  # num_classes = number of output classes
])

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

# Print model summary
model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm_2 (LSTM)               (None, 168, 128)          109568    
                                                                 
 dropout_2 (Dropout)         (None, 168, 128)          0         
                                                                 
 lstm_3 (LSTM)               (None, 128)               131584    
                                                                 
 dropout_3 (Dropout)         (None, 128)               0         
                                                                 
 dense_24 (Dense)            (None, 64)                8256      
                                                                 
 dense_25 (Dense)            (None, 4)                 260       
                                                                 
=================================================================
Total params: 249668 (975.27 KB)
Trainable params: 249668 (975.27 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model Training to find composers using LSTM

In [91]:

```
history = model.fit(x_train_combined, y_train_combined_encoded, epochs=10, batch_size=10
, validation_data=(x_test_combined, y_test_combined_encoded))
```

```
Epoch 1/10
44/44 [==============================] - 9s 110ms/step - loss: 1.0002 - accuracy: 0.4838
- val_loss: 0.8510 - val_accuracy: 0.5000
Epoch 2/10
44/44 [==============================] - 4s 96ms/step - loss: 0.8514 - accuracy: 0.4931 -
val_loss: 0.8342 - val_accuracy: 0.5000
Epoch 3/10
44/44 [==============================] - 6s 135ms/step - loss: 0.8476 - accuracy: 0.5000
- val_loss: 0.8358 - val_accuracy: 0.5000
Epoch 4/10
44/44 [==============================] - 5s 105ms/step - loss: 0.8405 - accuracy: 0.5000
- val_loss: 0.8259 - val_accuracy: 0.5000
Epoch 5/10
44/44 [==============================] - 6s 127ms/step - loss: 0.8284 - accuracy: 0.5093
- val_loss: 0.8337 - val_accuracy: 0.5000
Epoch 6/10
44/44 [==============================] - 5s 105ms/step - loss: 0.8399 - accuracy: 0.5046
- val_loss: 0.8282 - val_accuracy: 0.5000
Epoch 7/10
44/44 [==============================] - 5s 112ms/step - loss: 0.8472 - accuracy: 0.4699
- val_loss: 0.8293 - val_accuracy: 0.5000
Epoch 8/10
44/44 [==============================] - 5s 104ms/step - loss: 0.8414 - accuracy: 0.4861
- val_loss: 0.8334 - val_accuracy: 0.5000
Epoch 9/10
44/44 [==============================] - 5s 114ms/step - loss: 0.8389 - accuracy: 0.4537
```

```
- val_loss: 0.8259 - val_accuracy: 0.5000
Epoch 10/10
44/44 [==============================] - 5s 107ms/step - loss: 0.8321 - accuracy: 0.4861
- val_loss: 0.8284 - val_accuracy: 0.5000
```

Model Evaluation Using LSTM

In [92]:

```python
from sklearn.metrics import classification_report

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test_combined, y_test_combined_encoded)
print(f"Test accuracy: {test_acc}")

# Make predictions
predictions = model.predict(x_test_combined)
predicted_classes = np.argmax(predictions, axis=1)
print("========================================\n")
print("        Expected Result                   ")
print("========================================\n")
print(y_test_combined_encoded)
print("\n")

print("========================================\n")
print("        Actual Result                   ")
print("========================================\n")
print(predicted_classes)
print("\n")

# Generate the classification report
report = classification_report(y_test_combined_encoded, predicted_classes)

print("========================================\n")
print("   CLASSIFICATION REPORT                 \n")
print("========================================\n")
# Print the classification report
print(report)
```

```
4/4 [==============================] - 1s 125ms/step - loss: 0.8284 - accuracy: 0.5000
Test accuracy: 0.5
4/4 [==============================] - 1s 103ms/step
========================================

        Expected Result
========================================

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]


========================================

        Actual Result
========================================

[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]


========================================

   CLASSIFICATION REPORT

========================================

              precision    recall  f1-score   support
```

```
           0       0.00      0.00      0.00        27
           1       1.00      1.00      1.00        27
           2       0.33      1.00      0.50        27
           3       0.00      0.00      0.00        27

    accuracy                           0.50       108
   macro avg       0.33      0.50      0.38       108
weighted avg       0.33      0.50      0.38       108
```

From above, the accuracy, precision, recall and f1-score is poor for composers Bach and Mozart. It only did well for Beethoven here. So, let's do some hyperparameter tuning.

Hyper parameter tuning for LSTM Model

Let's increase number of epochs to 25 for training LSTM model.

In [95]:

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

# Define the input shape
timesteps = 168   # Number of time steps in the sequence (length of each sequence)
input_dim = 85    # Number of features per time step (dimension of each input vector)
num_classes = 4   # Number of output classes (e.g., 4 composers)

# Define the LSTM model
model = Sequential([
    LSTM(128, input_shape=(timesteps, input_dim), return_sequences=True),
    Dropout(0.2),
    LSTM(128),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')  # num_classes = number of output classes
])

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

# Print model summary
model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_11 (LSTM)              (None, 168, 128)          109568

 dropout_11 (Dropout)        (None, 168, 128)          0

 lstm_12 (LSTM)              (None, 128)               131584

 dropout_12 (Dropout)        (None, 128)               0

 dense_30 (Dense)            (None, 64)                8256

 dense_31 (Dense)            (None, 4)                 260

=================================================================
Total params: 249668 (975.27 KB)
Trainable params: 249668 (975.27 KB)
Non-trainable params: 0 (0.00 Byte)

_________________________________________________________________
```

Model Training With Tuned Parameters using LSTM

Let's try with 25 epochs

In [99]:

```
# With 25 epochs
history = model.fit(x_train_combined, y_train_combined_encoded, epochs=25, batch_size=32
, validation_data=(x_test_combined, y_test_combined_encoded))
```

```
Epoch 1/25
14/14 [==============================] - 3s 173ms/step - loss: 0.3606 - accuracy: 0.7546
- val_loss: 0.3492 - val_accuracy: 0.7500
Epoch 2/25
14/14 [==============================] - 2s 172ms/step - loss: 0.3625 - accuracy: 0.7593
- val_loss: 0.3433 - val_accuracy: 0.7500
Epoch 3/25
14/14 [==============================] - 3s 196ms/step - loss: 0.3540 - accuracy: 0.8079
- val_loss: 0.3085 - val_accuracy: 0.9537
Epoch 4/25
14/14 [==============================] - 3s 182ms/step - loss: 0.3551 - accuracy: 0.8079
- val_loss: 0.3896 - val_accuracy: 0.7407
Epoch 5/25
14/14 [==============================] - 4s 270ms/step - loss: 0.3707 - accuracy: 0.7824
- val_loss: 0.2946 - val_accuracy: 0.9074
Epoch 6/25
14/14 [==============================] - 4s 276ms/step - loss: 0.2604 - accuracy: 0.9444
- val_loss: 0.2231 - val_accuracy: 0.9444
Epoch 7/25
14/14 [==============================] - 3s 227ms/step - loss: 0.1651 - accuracy: 0.9653
- val_loss: 0.1685 - val_accuracy: 0.9352
Epoch 8/25
14/14 [==============================] - 3s 184ms/step - loss: 0.1109 - accuracy: 0.9653
- val_loss: 0.3449 - val_accuracy: 0.8241
Epoch 9/25
14/14 [==============================] - 3s 211ms/step - loss: 0.3603 - accuracy: 0.8565
- val_loss: 0.2883 - val_accuracy: 0.8426
Epoch 10/25
14/14 [==============================] - 3s 194ms/step - loss: 0.1354 - accuracy: 0.9583
- val_loss: 0.1137 - val_accuracy: 0.9722
Epoch 11/25
14/14 [==============================] - 2s 161ms/step - loss: 0.1093 - accuracy: 0.9606
- val_loss: 0.2261 - val_accuracy: 0.8981
Epoch 12/25
14/14 [==============================] - 2s 161ms/step - loss: 0.1230 - accuracy: 0.9514
- val_loss: 0.1981 - val_accuracy: 0.8981
Epoch 13/25
14/14 [==============================] - 4s 289ms/step - loss: 0.0754 - accuracy: 0.9769
- val_loss: 0.0929 - val_accuracy: 0.9815
Epoch 14/25
14/14 [==============================] - 4s 297ms/step - loss: 0.0590 - accuracy: 0.9815
- val_loss: 0.0983 - val_accuracy: 0.9722
Epoch 15/25
14/14 [==============================] - 3s 225ms/step - loss: 0.0479 - accuracy: 0.9815
- val_loss: 0.1029 - val_accuracy: 0.9722
Epoch 16/25
14/14 [==============================] - 3s 234ms/step - loss: 0.0292 - accuracy: 0.9931
- val_loss: 0.0977 - val_accuracy: 0.9815
Epoch 17/25
14/14 [==============================] - 4s 286ms/step - loss: 0.0281 - accuracy: 0.9884
- val_loss: 0.1034 - val_accuracy: 0.9815
Epoch 18/25
14/14 [==============================] - 3s 203ms/step - loss: 0.0208 - accuracy: 0.9954
- val_loss: 0.1051 - val_accuracy: 0.9815
Epoch 19/25
14/14 [==============================] - 2s 153ms/step - loss: 0.0290 - accuracy: 0.9907
- val_loss: 0.1346 - val_accuracy: 0.9722
Epoch 20/25
14/14 [==============================] - 3s 218ms/step - loss: 0.1539 - accuracy: 0.9398
- val_loss: 0.8201 - val_accuracy: 0.7778
Epoch 21/25
14/14 [==============================] - 3s 194ms/step - loss: 0.1487 - accuracy: 0.9514
- val_loss: 0.0599 - val_accuracy: 0.9907
```

```
Epoch 22/25
14/14 [==============================] - 3s 191ms/step - loss: 0.0436 - accuracy: 0.9815
- val_loss: 0.0767 - val_accuracy: 0.9815
Epoch 23/25
14/14 [==============================] - 3s 216ms/step - loss: 0.0310 - accuracy: 0.9884
- val_loss: 0.0727 - val_accuracy: 0.9815
Epoch 24/25
14/14 [==============================] - 4s 284ms/step - loss: 0.0227 - accuracy: 0.9907
- val_loss: 0.0377 - val_accuracy: 0.9907
Epoch 25/25
14/14 [==============================] - 5s 352ms/step - loss: 0.0345 - accuracy: 0.9884
- val_loss: 0.0926 - val_accuracy: 0.9907
```

Model Evaluation with tuned LSTM

In [100]:

```
from sklearn.metrics import classification_report

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test_combined, y_test_combined_encoded)
print(f"Test accuracy: {test_acc}")

# Make predictions
predictions = model.predict(x_test_combined)
predicted_classes = np.argmax(predictions, axis=1)
print("========================================\n")
print("        Expected Result                  ")
print("========================================\n")
print(y_test_combined_encoded)
print("\n")

print("========================================\n")
print("        Actual Result                  ")
print("========================================\n")
print(predicted_classes)
print("\n")

# Generate the classification report
report = classification_report(y_test_combined_encoded, predicted_classes)

print("========================================\n")
print("   CLASSIFICATION REPORT                \n")
print("========================================\n")
# Print the classification report
print(report)
```

```
4/4 [==============================] - 0s 82ms/step - loss: 0.0926 - accuracy: 0.9907
Test accuracy: 0.9907407164573669
4/4 [==============================] - 1s 96ms/step
========================================


        Expected Result
========================================


[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]


========================================


        Actual Result
========================================


[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3]


========================================
```

```
   CLASSIFICATION REPORT
========================================
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        27
           1       1.00      1.00      1.00        27
           2       1.00      1.00      1.00        27
           3       1.00      0.96      0.98        27

    accuracy                           0.99       108
   macro avg       0.99      0.99      0.99       108
weighted avg       0.99      0.99      0.99       108
```

Totally unseen files Testing using Long Short Term Memory (LSTM)

In [101]:

```
import warnings

warnings.filterwarnings("ignore")

bach_new_files = ['/Users/manikanr/Downloads/archive/midiclassics/Bach/Concertos/Bwv1047
Brandenburg Concert n2 1mov.mid',
                  '/Users/manikanr/Downloads/archive/midiclassics/Mozart/K299 Flute Harp
Concerto 1mov.mid',
                  '/Users/manikanr/Downloads/archive/midiclassics/Chopin/Piano Concerto
n1 op11 1mov.mid',
                  '/Users/manikanr/Downloads/archive/midiclassics/Beethoven/Bagatella Fu
r Elise.mid']

# Check if files in bach_new_files are in already validated files
for file in bach_new_files:
    if file in bach_used_midi_files:
        print(f"{file} is in bach_used_midi_files.")
    else:
        print(f"{file} is NOT in bach_used_midi_files.")

bach_new_features, bach_new_labels, bach_new_used_files = collectFeatures("Bach", bach_n
ew_files)
max_len = max(sample.shape[0] for sample in bach_new_features)
bach_new_features_reshaped = normalizeFeatures(bach_new_features, 168)

# Pad each feature set to have the same number of columns
bach_new_features_padded = pad_to_max_columns(bach_new_features_reshaped, 85)
print(bach_new_features_padded.shape)
'''
# New data shape: (5, 20, 40)
bach_new_shape = np.random.rand(5, 20, 40)  # Example data

# Expected input shape: (168, 85, 1)
expected_shape = (168, 85, 1)

# Reshape new data to match expected input shape
# This example uses padding with zeros to achieve the desired shape

padded_data = np.zeros((5, *expected_shape))  # Initialize with zeros

# Insert the original data into the padded array
# This assumes you want to place the new data in the top-left corner
padded_data[:, :20, :40, 0] = bach_new_shape

# Now padded_data should have the shape (5, 168, 85, 1)
print(padded_data.shape)  # Should output (5, 168, 85, 1)
'''
# Make predictions
predictions = model.predict(bach_new_features_padded)
```

```python
predicted_classes = np.argmax(predictions, axis=1)
print(predicted_classes)
```

```
/Users/manikanr/Downloads/archive/midiclassics/Bach/Concertos/Bwv1047 Brandenburg Concert
n2 1mov.mid is NOT in bach_used_midi_files.
/Users/manikanr/Downloads/archive/midiclassics/Mozart/K299 Flute Harp Concerto 1mov.mid i
s NOT in bach_used_midi_files.
/Users/manikanr/Downloads/archive/midiclassics/Chopin/Piano Concerto n1 op11 1mov.mid is
NOT in bach_used_midi_files.
/Users/manikanr/Downloads/archive/midiclassics/Beethoven/Bagatella Fur Elise.mid is NOT i
n bach_used_midi_files.
(4, 168, 85)
1/1 [==============================] - 0s 30ms/step
[0 0 3 0]
```

Conclusion and Results of LSTM Model

The model has achieved perfect performance on this dataset, with 98% accuracy, precision, recall, and F1-score across all classes. This is an ideal outcome and suggests that the model has learned to distinguish between the different classes perfectly, at least on the test set provided. However, such perfect scores could sometimes indicate that the model might be overfitting, especially if the dataset is small or not very diverse. Since unseen data provides indifferent results, it can prove more training data set of each of these composers is required. Also, it shows finding differences in music and rhythmic patterns is not an easy task and the above analysis are the basics in building the model. It requires more deeper analysis to get perfect results.