In [121…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from dataprep.eda import plot, plot_missing, plot_correlation, create_report

import pretty_midi
import os
import joblib

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.metrics import (confusion_matrix, precision_score, recall_score,
                             roc_auc_score, ConfusionMatrixDisplay, f1_score,
                             accuracy_score, classification_report)
```

In [2]:
```
import tensorflow as tf
tf.__version__
```

Out[2]:
```
'2.12.0'
```

We will do very minor exploration below.

In [38]:
```
midi_file = "bach342.mid"
midi_file = pretty_midi.PrettyMIDI(midi_file)
```

In [39]:
```
midi_file.instruments
```

Out[39]:
```
[Instrument(program=6, is_drum=False, name="Track 1"),
 Instrument(program=6, is_drum=False, name="Track 1"),
 Instrument(program=6, is_drum=False, name="Track 1"),
 Instrument(program=6, is_drum=False, name="Track 1"),
 Instrument(program=6, is_drum=False, name="Track 1"),
 Instrument(program=6, is_drum=False, name="Track 1"),
 Instrument(program=6, is_drum=False, name="Track 1"),
 Instrument(program=6, is_drum=False, name="Track 1"),
 Instrument(program=6, is_drum=False, name="Track 1"),
 Instrument(program=6, is_drum=False, name="Track 1"),
 Instrument(program=6, is_drum=False, name="Track 1"),
 Instrument(program=6, is_drum=False, name="Track 1")]
```

In [40]:
```
midi_file2 = "handel107.mid"
midi_file2 = pretty_midi.PrettyMIDI(midi_file2)
```

In [41]:
```
midi_file2.instruments
```

Out[41]:
```
[Instrument(program=70, is_drum=False, name="Joshua"),
 Instrument(program=40, is_drum=False, name="Strings I"),
 Instrument(program=48, is_drum=False, name="Strings II"),
 Instrument(program=41, is_drum=False, name="Viola"),
 Instrument(program=42, is_drum=False, name="Cello"),
 Instrument(program=43, is_drum=False, name="Double Bass"),
 Instrument(program=6, is_drum=False, name="Harpsichord Treble"),
 Instrument(program=6, is_drum=False, name="Harpsichord Bass")]
```

In [42]:
```
midi_file3 = "handel147.mid"
midi_file3 = pretty_midi.PrettyMIDI(midi_file3)
```

```
C:\Users\yousi\anaconda3\lib\site-packages\pretty_midi\pretty_midi.py:100: Runtime
Warning: Tempo, Key or Time signature change events found on non-zero tracks.  Thi
s is not a valid type 0 or type 1 MIDI file.  Tempo, Key or Time Signature may be
wrong.
  warnings.warn(
```

In [43]:
```
midi_file3.instruments
```

Out[43]:
```
[Instrument(program=68, is_drum=False, name="Oboe I"),
 Instrument(program=68, is_drum=False, name="Oboe II"),
 Instrument(program=70, is_drum=False, name="Bassoon"),
 Instrument(program=48, is_drum=False, name="Violin I"),
 Instrument(program=48, is_drum=False, name="Violin II"),
 Instrument(program=41, is_drum=False, name="Viola"),
 Instrument(program=42, is_drum=False, name="Cello"),
 Instrument(program=43, is_drum=False, name="Double Bass"),
 Instrument(program=6, is_drum=False, name="Harpsichord RH"),
 Instrument(program=6, is_drum=False, name="Harpsichord LH")]
```

From the quick explorations above in 3 separate songs, we notice that the names of the instruments vary widely between them. Thus, we will not be using this information in the model, but we will use use the notes from those instruents without allowing for the ability to tell which instrument is which. This is obviously not a great way to proceed since each instrument is played differently even within the same composer but it seems like the better option for now.

Note: after further research, we found out that the program attribute is standardized. It's an integer between 0 and 127 inclusive. We can include this information for future work and we think that it will be a great indicator feature to add to classify composers.

Below is a sample of the notes that we're going to extract.

In [138…
```
midi_file.instruments[0].notes[:50]
```

Out[138]:
```
[Note(start=0.449219, end=0.597656, pitch=49, velocity=127),
 Note(start=1.821615, end=1.970052, pitch=61, velocity=127),
 Note(start=2.295573, end=2.444010, pitch=61, velocity=127),
 Note(start=3.667969, end=3.816406, pitch=73, velocity=127),
 Note(start=4.576823, end=4.710938, pitch=73, velocity=127),
 Note(start=7.343750, end=9.472656, pitch=73, velocity=127),
 Note(start=14.266927, end=14.358073, pitch=73, velocity=127),
 Note(start=14.430990, end=15.522135, pitch=73, velocity=127),
 Note(start=15.653646, end=15.714844, pitch=73, velocity=127),
 Note(start=16.283854, end=16.535156, pitch=73, velocity=127),
 Note(start=15.778646, end=16.673177, pitch=61, velocity=127),
 Note(start=16.789062, end=17.040365, pitch=73, velocity=127),
 Note(start=17.799479, end=18.050781, pitch=61, velocity=127),
 Note(start=18.304688, end=18.555990, pitch=61, velocity=127),
 Note(start=19.062500, end=19.313802, pitch=73, velocity=127),
 Note(start=19.567708, end=19.819010, pitch=73, velocity=127),
 Note(start=20.514323, end=20.597656, pitch=73, velocity=127),
 Note(start=20.670573, end=21.399740, pitch=73, velocity=127),
 Note(start=21.837240, end=23.100260, pitch=49, velocity=127),
 Note(start=26.890625, end=27.141927, pitch=49, velocity=127),
 Note(start=27.395833, end=27.647135, pitch=49, velocity=127),
 Note(start=28.911458, end=29.162760, pitch=49, velocity=127),
 Note(start=29.921875, end=30.173177, pitch=61, velocity=127),
 Note(start=30.427083, end=30.678385, pitch=73, velocity=127),
 Note(start=30.427083, end=30.678385, pitch=61, velocity=127),
 Note(start=32.953125, end=33.204427, pitch=73, velocity=127),
 Note(start=33.458333, end=33.709635, pitch=73, velocity=127),
 Note(start=34.468750, end=34.720052, pitch=61, velocity=127),
 Note(start=34.972656, end=35.222656, pitch=61, velocity=127),
 Note(start=35.981771, end=36.233073, pitch=49, velocity=127),
 Note(start=36.486979, end=36.738281, pitch=49, velocity=127),
 Note(start=37.497396, end=37.748698, pitch=61, velocity=127),
 Note(start=38.255208, end=38.506510, pitch=61, velocity=127),
 Note(start=40.781250, end=41.032552, pitch=49, velocity=127),
 Note(start=40.023438, end=41.536458, pitch=61, velocity=127),
 Note(start=41.791667, end=42.042969, pitch=49, velocity=127),
 Note(start=42.296875, end=42.548177, pitch=49, velocity=127),
 Note(start=44.570312, end=45.832031, pitch=61, velocity=127),
 Note(start=46.084635, end=46.335938, pitch=61, velocity=127),
 Note(start=48.358073, end=48.609375, pitch=61, velocity=127),
 Note(start=46.084635, end=48.861979, pitch=37, velocity=127),
 Note(start=48.863281, end=49.114583, pitch=49, velocity=127),
 Note(start=50.884115, end=51.135417, pitch=73, velocity=127),
 Note(start=51.389323, end=51.640625, pitch=73, velocity=127),
 Note(start=51.894531, end=52.145833, pitch=61, velocity=127),
 Note(start=52.904948, end=53.658854, pitch=49, velocity=127),
 Note(start=54.417969, end=54.669271, pitch=73, velocity=127),
 Note(start=55.175781, end=55.427083, pitch=61, velocity=127),
 Note(start=55.933594, end=56.183594, pitch=61, velocity=127),
 Note(start=57.445312, end=57.696615, pitch=61, velocity=127)]
```

In [19]:
```python
def extract_features(midi_file: str, max_num_of_points: int, stride: int):
    """
    This function will extract features from the midi files and extracts their feat
        Parameters:
            midi_file: a string that has the file path to the midi file
            max_num_of_points: the maximum number of notes and durations to extract
            stride: the index jump value used in returning the notes, i.e., every s
        returns:
            ([notes], [durations]): a list with notes and their respective duration
    """


    midi_data = pretty_midi.PrettyMIDI(midi_file)
```

```
    notes = [0] * max_num_of_points
    durations = [0] * max_num_of_points
    # this doesn't take into account the instrument name. We took this approach bec
    # different composers have different instrument name and it's not unified
    # thus, it will introduce a level of complexity and it might not be all that be
    # if we have too many different instruments which is what we've observed from d

    i = 0
    max_reached = False

    for instrument in midi_data.instruments:
        for note in instrument.notes:
            if i == max_num_of_points:
                max_reached = True
                break
            notes[i] = note.pitch
            durations[i] = note.end - note.start
            i += 1
        if max_reached:
            break

    # If the number of notes exceed max_num_of_points, the rest will be truncated.
    # If we have less than max_num_of_points points, we resort to padding with zero

    return notes[::stride], durations[::stride]
```

In [30]:

```
def build_dataset(dataset_directory: str, max_num_of_points: int, stride: int):
    composers = [] # this will hold the composers' names

    features_list = []
    labels_list = [] # composer name corresponding to features in features_list

    for composer in os.listdir(dataset_directory):
        if composer.startswith("."): # to avoid processing ".DS_Store"
            continue

        composers.append(composer)
        composer_directory = os.path.join(dataset_directory, composer)
        for midi_file in os.listdir(composer_directory):
            if midi_file.startswith(".") or not midi_file.endswith(".mid"): # to av
                continue

            file_path = os.path.join(composer_directory, midi_file)

            notes, durations = extract_features(file_path, max_num_of_points, strid
            features_list.append(np.column_stack((notes, durations)))
            labels_list.append(composer)

    return np.array(features_list), labels_list, composers
```

The code above takes a while to run, thus we will persist the data using **joblib**.

In [116…

```
dataset_directory ="data/train"
max_num_of_points = 4000
stride = 5

features_list, labels_list, composers = build_dataset(
    dataset_directory, max_num_of_points, stride
)


joblib.dump(features_list, "features_list.joblib")
```

```
joblib.dump(labels_list, "labels_list.joblib")
joblib.dump(composers, "composers.joblib")
```

```
C:\Users\yousi\anaconda3\lib\site-packages\pretty_midi\pretty_midi.py:100: Runtime
Warning: Tempo, Key or Time signature change events found on non-zero tracks.  Thi
s is not a valid type 0 or type 1 MIDI file.  Tempo, Key or Time Signature may be
wrong.
  warnings.warn(
```

Out[116]:
```
['composers.joblib']
```

In [4]:
```
features_list = joblib.load("features_list.joblib")
labels_list = joblib.load("labels_list.joblib")
composers = joblib.load("composers.joblib")
```

Now we will rescale our data using MinMaxScaler

In [5]:
```
sc = MinMaxScaler()

features_list = sc.fit_transform(features_list.reshape(
                 features_list.shape[0] *
                 features_list.shape[1],
                 features_list.shape[-1]
)).reshape(features_list.shape)
```

Now we will build our data sequences to prepare them for the LSTM

In [6]:
```
X_train = []
y_train = []

seq_length = 20 # window of 20 consecutive notes

for song_index in range(len(labels_list)):
    for note_index in range(len(features_list[song_index]) - seq_length):
        curr_seq = features_list[song_index, note_index:note_index+seq_length, :]
        if np.isin([0], curr_seq)[0]: # if we reach the zero padding in the end, we
                                      # skip the song as we no longer have any note
            break
        X_train.append(curr_seq)
        y_train.append(labels_list[song_index])
```

We will encode the composers using one-hot-encoding

In [8]:
```
composer_encoder = OneHotEncoder(categories=[composers], sparse_output=False)

y_train = composer_encoder.fit_transform(np.array(y_train).reshape(-1,1))
y_train = tf.constant(y_train, dtype=tf.float32)
```

In [25]:
```
# Build the LSTM model
model2 = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(units=150, input_shape=(seq_length, X_train.shape[-1]),
                         return_sequences=False),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(len(composers), activation="softmax")
])

# Compile the model
model2.compile(loss='categorical_crossentropy', optimizer="adam", metrics=["accura

# Train the model
num_epochs = 300
```

```
batch_size = 32 # default batch size
model_path = "model.h5"

history = model2.fit(X_train, y_train, epochs=num_epochs, batch_size=batch_size, va
          callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', mi
                       tf.keras.callbacks.ModelCheckpoint(model_path, monitor='val_
          )
```

```
Epoch 1/300
4918/4918 - 174s - loss: 1.8740 - accuracy: 0.2980 - val_loss: 3.1328 - val_accura
cy: 0.0000e+00 - 174s/epoch - 35ms/step
Epoch 2/300
4918/4918 - 165s - loss: 1.7812 - accuracy: 0.3387 - val_loss: 3.2751 - val_accura
cy: 0.0000e+00 - 165s/epoch - 33ms/step
Epoch 3/300
4918/4918 - 176s - loss: 1.6862 - accuracy: 0.3751 - val_loss: 3.2444 - val_accura
cy: 0.0000e+00 - 176s/epoch - 36ms/step
Epoch 4/300
4918/4918 - 166s - loss: 1.5132 - accuracy: 0.4313 - val_loss: 3.2223 - val_accura
cy: 0.0000e+00 - 166s/epoch - 34ms/step
Epoch 5/300
4918/4918 - 169s - loss: 1.4504 - accuracy: 0.4551 - val_loss: 3.0420 - val_accura
cy: 1.2073e-04 - 169s/epoch - 34ms/step
Epoch 6/300
4918/4918 - 167s - loss: 1.3862 - accuracy: 0.4801 - val_loss: 3.3533 - val_accura
cy: 0.0012 - 167s/epoch - 34ms/step
Epoch 7/300
4918/4918 - 169s - loss: 1.3240 - accuracy: 0.5044 - val_loss: 3.4972 - val_accura
cy: 6.0365e-04 - 169s/epoch - 34ms/step
Epoch 8/300
4918/4918 - 177s - loss: 1.2613 - accuracy: 0.5280 - val_loss: 3.3293 - val_accura
cy: 0.0048 - 177s/epoch - 36ms/step
Epoch 9/300
4918/4918 - 163s - loss: 1.2015 - accuracy: 0.5514 - val_loss: 3.1884 - val_accura
cy: 0.0164 - 163s/epoch - 33ms/step
Epoch 10/300
4918/4918 - 201s - loss: 1.1398 - accuracy: 0.5765 - val_loss: 3.3396 - val_accura
cy: 0.0500 - 201s/epoch - 41ms/step
Epoch 11/300
4918/4918 - 193s - loss: 1.0776 - accuracy: 0.5999 - val_loss: 3.5828 - val_accura
cy: 0.0707 - 193s/epoch - 39ms/step
Epoch 12/300
4918/4918 - 197s - loss: 1.0163 - accuracy: 0.6234 - val_loss: 3.8455 - val_accura
cy: 0.0640 - 197s/epoch - 40ms/step
Epoch 13/300
4918/4918 - 172s - loss: 0.9581 - accuracy: 0.6460 - val_loss: 3.4676 - val_accura
cy: 0.0890 - 172s/epoch - 35ms/step
Epoch 14/300
4918/4918 - 203s - loss: 0.9016 - accuracy: 0.6673 - val_loss: 4.0187 - val_accura
cy: 0.0730 - 203s/epoch - 41ms/step
Epoch 15/300
4918/4918 - 165s - loss: 0.8484 - accuracy: 0.6874 - val_loss: 4.0333 - val_accura
cy: 0.0790 - 165s/epoch - 33ms/step
Epoch 16/300
4918/4918 - 175s - loss: 0.8007 - accuracy: 0.7069 - val_loss: 4.2209 - val_accura
cy: 0.0707 - 175s/epoch - 36ms/step
Epoch 17/300
4918/4918 - 172s - loss: 0.7552 - accuracy: 0.7233 - val_loss: 3.9790 - val_accura
cy: 0.1192 - 172s/epoch - 35ms/step
Epoch 18/300
4918/4918 - 177s - loss: 0.7135 - accuracy: 0.7396 - val_loss: 4.5456 - val_accura
cy: 0.1318 - 177s/epoch - 36ms/step
Epoch 19/300
4918/4918 - 181s - loss: 0.6729 - accuracy: 0.7550 - val_loss: 4.6076 - val_accura
cy: 0.1078 - 181s/epoch - 37ms/step
Epoch 20/300
4918/4918 - 200s - loss: 0.6405 - accuracy: 0.7669 - val_loss: 4.7129 - val_accura
cy: 0.1193 - 200s/epoch - 41ms/step
Epoch 21/300
4918/4918 - 205s - loss: 0.6064 - accuracy: 0.7792 - val_loss: 5.0693 - val_accura
cy: 0.0881 - 205s/epoch - 42ms/step
Epoch 22/300
```

```
4918/4918 - 175s - loss: 0.5720 - accuracy: 0.7909 - val_loss: 4.7732 - val_accura
cy: 0.1434 - 175s/epoch - 36ms/step
Epoch 23/300
4918/4918 - 169s - loss: 0.5454 - accuracy: 0.8019 - val_loss: 5.0717 - val_accura
cy: 0.1291 - 169s/epoch - 34ms/step
Epoch 24/300
4918/4918 - 178s - loss: 0.5214 - accuracy: 0.8107 - val_loss: 5.1262 - val_accura
cy: 0.1300 - 178s/epoch - 36ms/step
Epoch 25/300
4918/4918 - 175s - loss: 0.4939 - accuracy: 0.8212 - val_loss: 5.4181 - val_accura
cy: 0.1365 - 175s/epoch - 36ms/step
Epoch 26/300
4918/4918 - 176s - loss: 0.4723 - accuracy: 0.8273 - val_loss: 5.7088 - val_accura
cy: 0.1281 - 176s/epoch - 36ms/step
Epoch 27/300
4918/4918 - 190s - loss: 0.4515 - accuracy: 0.8350 - val_loss: 5.2507 - val_accura
cy: 0.1282 - 190s/epoch - 39ms/step
Epoch 28/300
4918/4918 - 223s - loss: 0.4328 - accuracy: 0.8429 - val_loss: 5.2881 - val_accura
cy: 0.1521 - 223s/epoch - 45ms/step
Epoch 29/300
4918/4918 - 200s - loss: 0.4115 - accuracy: 0.8502 - val_loss: 5.9221 - val_accura
cy: 0.1391 - 200s/epoch - 41ms/step
Epoch 30/300
4918/4918 - 258s - loss: 0.3964 - accuracy: 0.8567 - val_loss: 5.9713 - val_accura
cy: 0.1432 - 258s/epoch - 52ms/step
Epoch 31/300
4918/4918 - 191s - loss: 0.3781 - accuracy: 0.8635 - val_loss: 5.7184 - val_accura
cy: 0.1590 - 191s/epoch - 39ms/step
Epoch 32/300
4918/4918 - 96s - loss: 0.3656 - accuracy: 0.8683 - val_loss: 6.4025 - val_accurac
y: 0.1251 - 96s/epoch - 20ms/step
Epoch 33/300
4918/4918 - 109s - loss: 0.3519 - accuracy: 0.8732 - val_loss: 6.2369 - val_accura
cy: 0.1427 - 109s/epoch - 22ms/step
Epoch 34/300
4918/4918 - 178s - loss: 0.3387 - accuracy: 0.8769 - val_loss: 6.7071 - val_accura
cy: 0.1413 - 178s/epoch - 36ms/step
Epoch 35/300
4918/4918 - 183s - loss: 0.3251 - accuracy: 0.8824 - val_loss: 6.6671 - val_accura
cy: 0.1409 - 183s/epoch - 37ms/step
Epoch 36/300
4918/4918 - 374s - loss: 0.3150 - accuracy: 0.8866 - val_loss: 6.7755 - val_accura
cy: 0.1426 - 374s/epoch - 76ms/step
Epoch 37/300
4918/4918 - 185s - loss: 0.3013 - accuracy: 0.8916 - val_loss: 6.9571 - val_accura
cy: 0.1231 - 185s/epoch - 38ms/step
Epoch 38/300
4918/4918 - 112s - loss: 0.2935 - accuracy: 0.8943 - val_loss: 6.7434 - val_accura
cy: 0.1434 - 112s/epoch - 23ms/step
Epoch 39/300
4918/4918 - 119s - loss: 0.2800 - accuracy: 0.9006 - val_loss: 6.2913 - val_accura
cy: 0.1899 - 119s/epoch - 24ms/step
Epoch 40/300
4918/4918 - 114s - loss: 0.2731 - accuracy: 0.9027 - val_loss: 6.4946 - val_accura
cy: 0.1740 - 114s/epoch - 23ms/step
Epoch 41/300
4918/4918 - 219s - loss: 0.2671 - accuracy: 0.9053 - val_loss: 6.6277 - val_accura
cy: 0.1583 - 219s/epoch - 44ms/step
Epoch 42/300
4918/4918 - 160s - loss: 0.2533 - accuracy: 0.9103 - val_loss: 6.7496 - val_accura
cy: 0.1664 - 160s/epoch - 33ms/step
Epoch 43/300
4918/4918 - 225s - loss: 0.2488 - accuracy: 0.9118 - val_loss: 7.1162 - val_accura
```

```
cy: 0.1601 - 225s/epoch - 46ms/step
Epoch 44/300
4918/4918 - 175s - loss: 0.2401 - accuracy: 0.9140 - val_loss: 7.5469 - val_accura
cy: 0.1391 - 175s/epoch - 35ms/step
Epoch 45/300
4918/4918 - 136s - loss: 0.2368 - accuracy: 0.9158 - val_loss: 7.2108 - val_accura
cy: 0.1551 - 136s/epoch - 28ms/step
Epoch 46/300
4918/4918 - 178s - loss: 0.2258 - accuracy: 0.9193 - val_loss: 7.5707 - val_accura
cy: 0.1443 - 178s/epoch - 36ms/step
Epoch 47/300
4918/4918 - 170s - loss: 0.2192 - accuracy: 0.9231 - val_loss: 6.9862 - val_accura
cy: 0.1711 - 170s/epoch - 35ms/step
Epoch 48/300
4918/4918 - 169s - loss: 0.2169 - accuracy: 0.9228 - val_loss: 7.0599 - val_accura
cy: 0.1757 - 169s/epoch - 34ms/step
Epoch 49/300
4918/4918 - 169s - loss: 0.2101 - accuracy: 0.9262 - val_loss: 8.1344 - val_accura
cy: 0.1392 - 169s/epoch - 34ms/step
```

We can see that the validation accuracy we achieved was a measly 19% approximately even though the training accuracy is about 92%. This tells us that the sequence that we have built for the training data is not informative enough of the composer. We can make the case that the model isn't performing well because it's not complex enough but I don't believe that this is the case. I've experimented with different architectures and layers but it didn't help in any significant way. I don't think hyperparameter tuning will give us a very big boost in the accuracy, either.

Now, we will rebuild the data sets and take consecutive notes rather than with a stride of 5 (every 5th note). We'll see if this imrpoves the performance. We will also increase the window size for the sequences from 20 to 40 to see if this helps increase the accuracy of the validation set.

In [12]:
```
# Build the LSTM model
model2 = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(units=150, input_shape=(seq_length, X_train.shape[-1]),
                         return_sequences=True),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv1D(filters=32, kernel_size=3, activation="relu"),
    tf.keras.layers.MaxPool1D(pool_size=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(len(composers), activation="softmax")
])

# Compile the model
model2.compile(loss='categorical_crossentropy', optimizer="adam", metrics=["accura

# Train the model
num_epochs = 300
batch_size = 32 # default batch size
model_path = "model2.h5"

history = model2.fit(X_train, y_train, epochs=num_epochs, batch_size=batch_size, v
          callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', mi
                       tf.keras.callbacks.ModelCheckpoint(model_path,monitor='val_
          )
```

```
Epoch 1/300
4918/4918 - 208s - loss: 1.8337 - accuracy: 0.3140 - val_loss: 3.1390 - val_accura
cy: 0.0000e+00 - 208s/epoch - 42ms/step
Epoch 2/300
4918/4918 - 205s - loss: 1.5445 - accuracy: 0.4234 - val_loss: 3.0820 - val_accura
cy: 0.0000e+00 - 205s/epoch - 42ms/step
Epoch 3/300
4918/4918 - 199s - loss: 1.4396 - accuracy: 0.4626 - val_loss: 3.2794 - val_accura
cy: 0.0000e+00 - 199s/epoch - 41ms/step
Epoch 4/300
4918/4918 - 201s - loss: 1.3663 - accuracy: 0.4902 - val_loss: 3.1983 - val_accura
cy: 2.4146e-04 - 201s/epoch - 41ms/step
Epoch 5/300
4918/4918 - 203s - loss: 1.3091 - accuracy: 0.5110 - val_loss: 3.3322 - val_accura
cy: 0.0041 - 203s/epoch - 41ms/step
Epoch 6/300
4918/4918 - 201s - loss: 1.2481 - accuracy: 0.5356 - val_loss: 3.1706 - val_accura
cy: 0.0264 - 201s/epoch - 41ms/step
Epoch 7/300
4918/4918 - 199s - loss: 1.1850 - accuracy: 0.5602 - val_loss: 3.0658 - val_accura
cy: 0.0652 - 199s/epoch - 40ms/step
Epoch 8/300
4918/4918 - 207s - loss: 1.1142 - accuracy: 0.5874 - val_loss: 3.3405 - val_accura
cy: 0.0435 - 207s/epoch - 42ms/step
Epoch 9/300
4918/4918 - 202s - loss: 1.0429 - accuracy: 0.6146 - val_loss: 3.5916 - val_accura
cy: 0.0695 - 202s/epoch - 41ms/step
Epoch 10/300
4918/4918 - 205s - loss: 0.9766 - accuracy: 0.6397 - val_loss: 3.6420 - val_accura
cy: 0.1027 - 205s/epoch - 42ms/step
Epoch 11/300
4918/4918 - 205s - loss: 0.9056 - accuracy: 0.6666 - val_loss: 3.7637 - val_accura
cy: 0.1047 - 205s/epoch - 42ms/step
Epoch 12/300
4918/4918 - 205s - loss: 0.8410 - accuracy: 0.6919 - val_loss: 3.9008 - val_accura
cy: 0.1617 - 205s/epoch - 42ms/step
Epoch 13/300
4918/4918 - 209s - loss: 0.7806 - accuracy: 0.7124 - val_loss: 4.2165 - val_accura
cy: 0.1679 - 209s/epoch - 43ms/step
Epoch 14/300
4918/4918 - 206s - loss: 0.7272 - accuracy: 0.7347 - val_loss: 4.1552 - val_accura
cy: 0.1697 - 206s/epoch - 42ms/step
Epoch 15/300
4918/4918 - 226s - loss: 0.6739 - accuracy: 0.7541 - val_loss: 5.1045 - val_accura
cy: 0.1202 - 226s/epoch - 46ms/step
Epoch 16/300
4918/4918 - 240s - loss: 0.6238 - accuracy: 0.7723 - val_loss: 5.4320 - val_accura
cy: 0.1187 - 240s/epoch - 49ms/step
Epoch 17/300
4918/4918 - 201s - loss: 0.5843 - accuracy: 0.7871 - val_loss: 5.3464 - val_accura
cy: 0.1607 - 201s/epoch - 41ms/step
Epoch 18/300
4918/4918 - 191s - loss: 0.5452 - accuracy: 0.8032 - val_loss: 5.5761 - val_accura
cy: 0.1636 - 191s/epoch - 39ms/step
Epoch 19/300
4918/4918 - 216s - loss: 0.5118 - accuracy: 0.8140 - val_loss: 5.9796 - val_accura
cy: 0.1582 - 216s/epoch - 44ms/step
Epoch 20/300
4918/4918 - 212s - loss: 0.4830 - accuracy: 0.8251 - val_loss: 6.6841 - val_accura
cy: 0.1369 - 212s/epoch - 43ms/step
Epoch 21/300
4918/4918 - 209s - loss: 0.4525 - accuracy: 0.8356 - val_loss: 6.3043 - val_accura
cy: 0.1847 - 209s/epoch - 43ms/step
Epoch 22/300
```

```
4918/4918 - 214s - loss: 0.4274 - accuracy: 0.8454 - val_loss: 7.1567 - val_accura
cy: 0.1531 - 214s/epoch - 43ms/step
Epoch 23/300
4918/4918 - 195s - loss: 0.4020 - accuracy: 0.8550 - val_loss: 6.3295 - val_accura
cy: 0.1822 - 195s/epoch - 40ms/step
Epoch 24/300
4918/4918 - 207s - loss: 0.3812 - accuracy: 0.8619 - val_loss: 7.7104 - val_accura
cy: 0.1416 - 207s/epoch - 42ms/step
Epoch 25/300
4918/4918 - 208s - loss: 0.3587 - accuracy: 0.8707 - val_loss: 7.9100 - val_accura
cy: 0.1510 - 208s/epoch - 42ms/step
Epoch 26/300
4918/4918 - 213s - loss: 0.3426 - accuracy: 0.8755 - val_loss: 8.0786 - val_accura
cy: 0.1618 - 213s/epoch - 43ms/step
Epoch 27/300
4918/4918 - 205s - loss: 0.3261 - accuracy: 0.8822 - val_loss: 8.3402 - val_accura
cy: 0.1553 - 205s/epoch - 42ms/step
Epoch 28/300
4918/4918 - 145s - loss: 0.3102 - accuracy: 0.8880 - val_loss: 7.8664 - val_accura
cy: 0.1939 - 145s/epoch - 29ms/step
Epoch 29/300
4918/4918 - 139s - loss: 0.2958 - accuracy: 0.8934 - val_loss: 9.5004 - val_accura
cy: 0.1288 - 139s/epoch - 28ms/step
Epoch 30/300
4918/4918 - 194s - loss: 0.2820 - accuracy: 0.8975 - val_loss: 8.8794 - val_accura
cy: 0.1689 - 194s/epoch - 39ms/step
Epoch 31/300
4918/4918 - 134s - loss: 0.2695 - accuracy: 0.9023 - val_loss: 8.7574 - val_accura
cy: 0.1745 - 134s/epoch - 27ms/step
Epoch 32/300
4918/4918 - 144s - loss: 0.2586 - accuracy: 0.9067 - val_loss: 8.7788 - val_accura
cy: 0.1895 - 144s/epoch - 29ms/step
Epoch 33/300
4918/4918 - 220s - loss: 0.2453 - accuracy: 0.9115 - val_loss: 9.7165 - val_accura
cy: 0.1505 - 220s/epoch - 45ms/step
Epoch 34/300
4918/4918 - 261s - loss: 0.2393 - accuracy: 0.9143 - val_loss: 10.8443 - val_accur
acy: 0.1415 - 261s/epoch - 53ms/step
Epoch 35/300
4918/4918 - 249s - loss: 0.2313 - accuracy: 0.9160 - val_loss: 9.4684 - val_accura
cy: 0.1649 - 249s/epoch - 51ms/step
Epoch 36/300
4918/4918 - 281s - loss: 0.2212 - accuracy: 0.9201 - val_loss: 9.8643 - val_accura
cy: 0.1726 - 281s/epoch - 57ms/step
Epoch 37/300
4918/4918 - 205s - loss: 0.2134 - accuracy: 0.9227 - val_loss: 9.6423 - val_accura
cy: 0.1829 - 205s/epoch - 42ms/step
Epoch 38/300
4918/4918 - 140s - loss: 0.2076 - accuracy: 0.9257 - val_loss: 10.0819 - val_accur
acy: 0.1700 - 140s/epoch - 29ms/step
```

We notice that we get very similar results which tells us that the complexity we introduced isn't really catching much more of the underlying patterns. We will now change the way we extract the information from the midi files. We will not use a stride (greater than 1) as opposed to extracting every 5th note as we did before.

We will also increase our sequence length when building the data for the model. We will use a length of 50 notes. Hopefully, the increase of the window will make it easier for the model to classify the composer accurately.

In [21]:
```
dataset_directory ="data/train"
max_num_of_points = 800
stride = 1

features_list, labels_list, _ = build_dataset(
    dataset_directory, max_num_of_points, stride
)

features_list = np.array(features_list)
```

```
C:\Users\yousi\anaconda3\lib\site-packages\pretty_midi\pretty_midi.py:100: RuntimeWarning: Tempo, Key or Time signature change events found on non-zero tracks.  This is not a valid type 0 or type 1 MIDI file.  Tempo, Key or Time Signature may be wrong.
  warnings.warn(
```

In [24]:
```
sc = MinMaxScaler()

features_list = sc.fit_transform(features_list.reshape(
                 features_list.shape[0] *
                 features_list.shape[1],
                 features_list.shape[-1]
)).reshape(features_list.shape)
```

In [25]:
```
X_train = []
y_train = []

seq_length = 50 # window of 50 consecutive notes

for song_index in range(len(labels_list)):
    for note_index in range(len(features_list[song_index]) - seq_length):
        curr_seq = features_list[song_index, note_index:note_index+seq_length, :]
        if np.isin([0], curr_seq)[0]: # if we reach the zero padding in the end, we
                                      # skip the song as we no longer have any note
            break
        X_train.append(curr_seq)
        y_train.append(labels_list[song_index])

X_train = tf.constant(np.array(X_train), dtype=tf.float32)
y_train = composer_encoder.transform(np.array(y_train).reshape(-1,1))
y_train = tf.constant(y_train, dtype=tf.float32)
```

In [27]:
```
# Build the model
model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(units=150, input_shape=(seq_length, X_train.shape[-1]),
                         return_sequences=False),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(len(composers), activation="softmax")
])

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

# Train the model
num_epochs = 300
batch_size = 32 # default batch size
model_path = "model.h5"

history = model.fit(X_train, y_train, epochs=num_epochs, batch_size=batch_size, val
          callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', min
                       tf.keras.callbacks.ModelCheckpoint(model_path, monitor='val_
          )
```

```
Epoch 1/300
7486/7486 - 704s - loss: 1.8244 - accuracy: 0.3362 - val_loss: 3.0907 - val_accura
cy: 0.0000e+00 - 704s/epoch - 94ms/step
Epoch 2/300
7486/7486 - 634s - loss: 1.7032 - accuracy: 0.3845 - val_loss: 3.0977 - val_accura
cy: 2.3794e-04 - 634s/epoch - 85ms/step
Epoch 3/300
7486/7486 - 644s - loss: 1.5024 - accuracy: 0.4586 - val_loss: 2.9599 - val_accura
cy: 7.1383e-04 - 644s/epoch - 86ms/step
Epoch 4/300
7486/7486 - 634s - loss: 1.2057 - accuracy: 0.5621 - val_loss: 2.7481 - val_accura
cy: 0.0798 - 634s/epoch - 85ms/step
Epoch 5/300
7486/7486 - 647s - loss: 0.9479 - accuracy: 0.6573 - val_loss: 2.5599 - val_accura
cy: 0.1622 - 647s/epoch - 86ms/step
Epoch 6/300
7486/7486 - 630s - loss: 0.7435 - accuracy: 0.7321 - val_loss: 3.6125 - val_accura
cy: 0.0911 - 630s/epoch - 84ms/step
Epoch 7/300
7486/7486 - 630s - loss: 0.5828 - accuracy: 0.7912 - val_loss: 3.5736 - val_accura
cy: 0.1620 - 630s/epoch - 84ms/step
Epoch 8/300
7486/7486 - 773s - loss: 0.4600 - accuracy: 0.8359 - val_loss: 4.1894 - val_accura
cy: 0.1887 - 773s/epoch - 103ms/step
Epoch 9/300
7486/7486 - 708s - loss: 0.3663 - accuracy: 0.8712 - val_loss: 4.8920 - val_accura
cy: 0.1504 - 708s/epoch - 95ms/step
Epoch 10/300
7486/7486 - 729s - loss: 0.2973 - accuracy: 0.8962 - val_loss: 5.0600 - val_accura
cy: 0.1812 - 729s/epoch - 97ms/step
Epoch 11/300
7486/7486 - 729s - loss: 0.2444 - accuracy: 0.9151 - val_loss: 4.7596 - val_accura
cy: 0.1919 - 729s/epoch - 97ms/step
Epoch 12/300
7486/7486 - 699s - loss: 0.2041 - accuracy: 0.9305 - val_loss: 4.7317 - val_accura
cy: 0.2303 - 699s/epoch - 93ms/step
Epoch 13/300
7486/7486 - 667s - loss: 0.1695 - accuracy: 0.9426 - val_loss: 5.6806 - val_accura
cy: 0.2036 - 667s/epoch - 89ms/step
Epoch 14/300
7486/7486 - 653s - loss: 0.1453 - accuracy: 0.9517 - val_loss: 6.2090 - val_accura
cy: 0.1624 - 653s/epoch - 87ms/step
Epoch 15/300
7486/7486 - 609s - loss: 0.1274 - accuracy: 0.9581 - val_loss: 5.9134 - val_accura
cy: 0.2075 - 609s/epoch - 81ms/step
Epoch 16/300
7486/7486 - 617s - loss: 0.1088 - accuracy: 0.9640 - val_loss: 5.9122 - val_accura
cy: 0.2000 - 617s/epoch - 82ms/step
Epoch 17/300
7486/7486 - 640s - loss: 0.0959 - accuracy: 0.9688 - val_loss: 6.0176 - val_accura
cy: 0.1735 - 640s/epoch - 85ms/step
Epoch 18/300
7486/7486 - 634s - loss: 0.0854 - accuracy: 0.9728 - val_loss: 5.4178 - val_accura
cy: 0.2127 - 634s/epoch - 85ms/step
Epoch 19/300
7486/7486 - 643s - loss: 0.0774 - accuracy: 0.9753 - val_loss: 6.8923 - val_accura
cy: 0.1524 - 643s/epoch - 86ms/step
Epoch 20/300
7486/7486 - 623s - loss: 0.0718 - accuracy: 0.9775 - val_loss: 6.2679 - val_accura
cy: 0.1989 - 623s/epoch - 83ms/step
Epoch 21/300
7486/7486 - 694s - loss: 0.0660 - accuracy: 0.9792 - val_loss: 6.3301 - val_accura
cy: 0.2105 - 694s/epoch - 93ms/step
Epoch 22/300
```

```
7486/7486 - 747s - loss: 0.0633 - accuracy: 0.9807 - val_loss: 6.1760 - val_accura
cy: 0.1764 - 747s/epoch - 100ms/step
```

We achieved a slightly better resut here. It's not great by any means and if we used more of the data- here we only extracted the first 800 notes and their durations- we can possibly receive better accuracy.

Now, let's build our test set and evaluate the model performance on it.

In [31]:
```
dataset_directory ="data/test"
max_num_of_points = 800
stride = 1

features_list_test, labels_list_test, _ = build_dataset(
    dataset_directory, max_num_of_points, stride
)


features_list_test = sc.transform(features_list_test.reshape(
                features_list_test.shape[0] *
                features_list_test.shape[1],
                features_list_test.shape[-1]
)).reshape(features_list_test.shape)
```

In [32]:
```
X_test = []
y_test = []

seq_length = 50 # window of 50 consecutive notes

for song_index in range(len(labels_list_test)):
    for note_index in range(len(features_list_test[song_index]) - seq_length):
        curr_seq = features_list_test[song_index, note_index:note_index+seq_length
        if np.isin([0], curr_seq)[0]: # if we reach the zero padding in the end, we
                                      # skip the song as we no longer have any note
            break
        X_test.append(curr_seq)
        y_test.append(labels_list_test[song_index])

X_test = tf.constant(np.array(X_test), dtype=tf.float32)
y_test = composer_encoder.transform(np.array(y_test).reshape(-1,1))
y_test = tf.constant(y_test, dtype=tf.float32)
```

In [35]:
```
model.evaluate(X_test, y_test)
```

```
750/750 [==============================] - 22s 28ms/step - loss: 3.6692 - accurac
y: 0.4875
```
Out[35]:
```
[3.669236421585083, 0.4875046908855438]
```

We see that we've gotten 50% accuracy approximately

In [110…
```
predictions = model.predict(X_test)
class_predictions = np.argmax(predictions, axis=1)
```

```
750/750 [==============================] - 23s 30ms/step
```

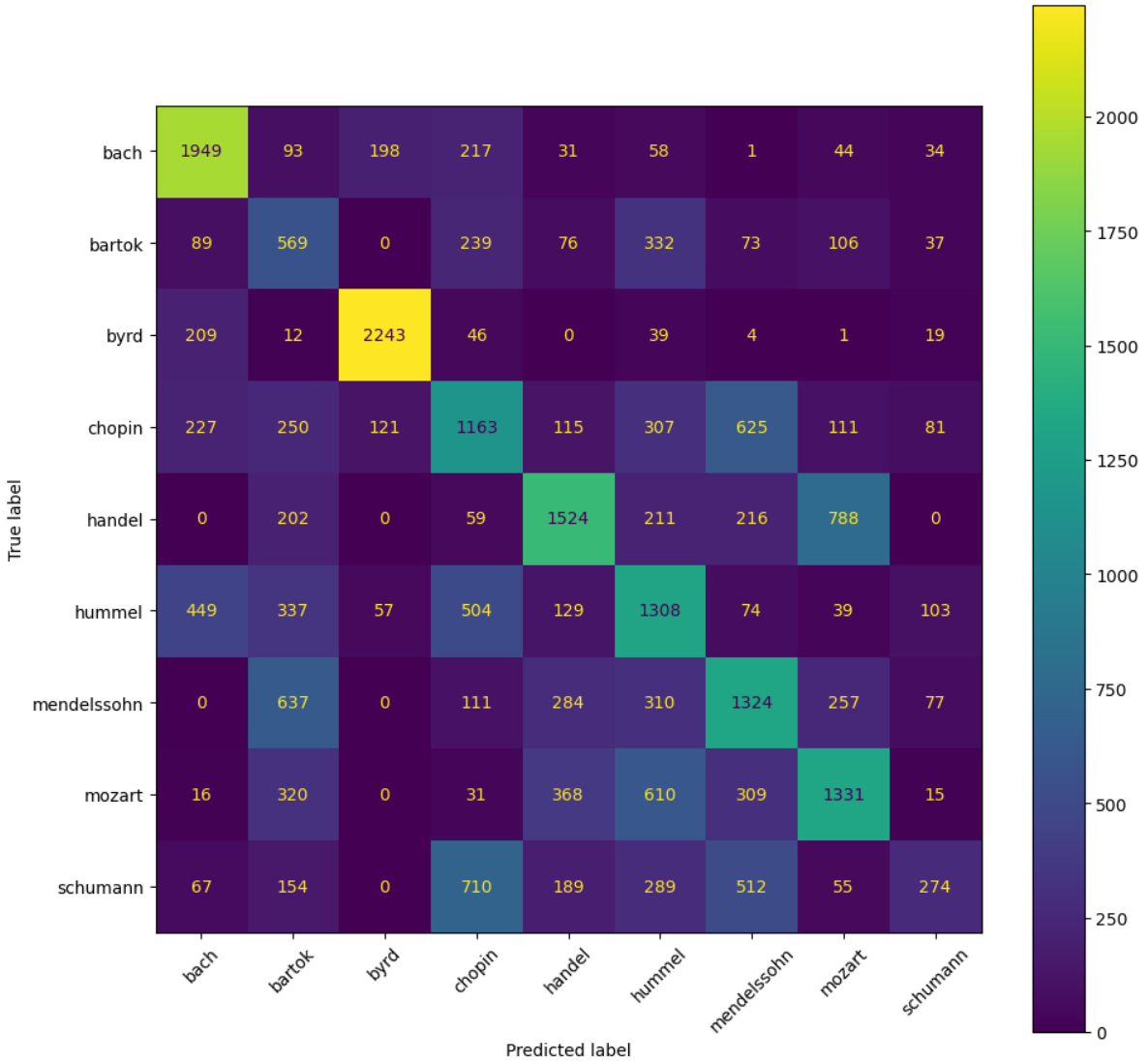
In [115…
```
y_test_classes = np.argmax(y_test, axis=1)

print("accuracy:", accuracy_score(y_test_classes, class_predictions))
print("precision:", precision_score(y_test_classes, class_predictions, average="ma
print("recall:", recall_score(y_test_classes, class_predictions, average="macro"))
print("f1-score:", f1_score(y_test_classes, class_predictions, average="macro"))
```

```
print("auc score:", roc_auc_score(y_test_classes, predictions, average="macro", mu

fig, ax = plt.subplots(figsize=(11, 11))
cm = confusion_matrix(np.argmax(y_test, axis=1), class_predictions)
ConfusionMatrixDisplay(confusion_matrix=cm,
                       display_labels=composer_encoder.categories[0]).plot(ax=ax)
plt.xticks(rotation=45)
plt.show()
```

```
accuracy: 0.48750469356251824
precision: 0.4866015460881798
recall: 0.4807512965662387
f1-score: 0.4711233571751809
auc score: 0.8160568337581541
```



In [125…

```
report = classification_report(y_test_classes, class_predictions, target_names=com
print(report)
```

```
                precision    recall  f1-score   support

          bach       0.65      0.74      0.69      2625
        bartok       0.22      0.37      0.28      1521
          byrd       0.86      0.87      0.86      2573
        chopin       0.38      0.39      0.38      3000
        handel       0.56      0.51      0.53      3000
        hummel       0.38      0.44      0.40      3000
    mendelssohn       0.42      0.44      0.43      3000
        mozart       0.49      0.44      0.46      3000
      schumann       0.43      0.12      0.19      2250

      accuracy                           0.49     23969
     macro avg       0.49      0.48      0.47     23969
  weighted avg       0.50      0.49      0.48     23969
```

We can see that we did somewhat reasonably. Keep in mind that we're only taking a window of 50 notes. If we wanted to do better, we would extend that window further. Also, we would train with a lot more data than just the first 800. We limited these numbers because of our computational restraints. As it is now, it took us about 3 hours to train. This made it harder to experiment. We notice that the best performing classes or composers are `Bach` and `Byrd`.

One more feature we could have added and would probably make the predictions get better is the velocity of each note. It would certainly make training the model take longer but we are confident that it can contribute in a noticeable way to our model's performance.

Another feature is the instrument number. As we mentioned in our **note** above, we found out that the instrument number is a standardized attribute and would have been easily added to the features that we extract. For time considerations, since the model will take so long to retrain. We leave this and the velocity features to be added in future work. We hypothesize that they can contribute very significantly to the performance.

In [ ]: