GitHub Link: github.com/jbaxterusd/baxter_final_511

Classifying the Classics: Deep Learning Models to Identify and Predict Classical Music Composers

Jack Baxter

Shiley-Marcos School of Engineering, University of San Diego

AAI-511: Neural Networks and Deep Learning

Dr. Mokhtari

August 11, 2025

Abstract

Modern artificial intelligence algorithms propose advanced potential to classify sequential audio, with large implications in historical research and music composition. Convolutional neural networks (CNNs) and long short-term memory (LSTM) networks are two methods championed for high efficiency in processing sequential data, including time series prediction and natural language processing (Kong, 2020). In the “midi_classic_music” public dataset on Kaggle, thousands of classical music records are represented by MIDI files, capable of use in algorithmic training. For the project, MIDI files of Bach, Beethoven, Chopin, and Mozart are employed in a 50-layer CNN (ResNet50) with strategic LSTM to reduce instances of vanishing gradient. The main purpose of the CNN is to classify each song of a separate test-set of MIDI files by respective composers. To facilitate machine learning, each MIDI file underwent a conversion from the original format to a ‘piano roll’, consistent with requirements of Tensorflow models. Additionally, future directions for “midi_classic_music” analysis and the implications CNNs and LSTM have in music composition are discussed.

Keywords: artificial intelligence, classification, convolutional neural networks, long short-term memory, midi files, piano roll

Data Pre-Processing

The primary dataset for the project was provided by the course instructor, downloaded directly from the public Kaggle website. Titled “midi_classic_music” and composed of 3929 MIDI files, the dataset contained work from 175 different composers, each entry a MIDI file of one individual song. To narrow the scope of the project, the first pre-processing step included the filtration of MIDI files for only the following artists: Bach, Beethoven, Chopin and Mozart. This was performed manually as non-selected composers were deleted. Once filtered, all MIDI files for selected artists were uploaded to a Google Drive folder for simplicity in usage with the Google CoLab application. Each composer had a separate folder to store respective MIDI files.

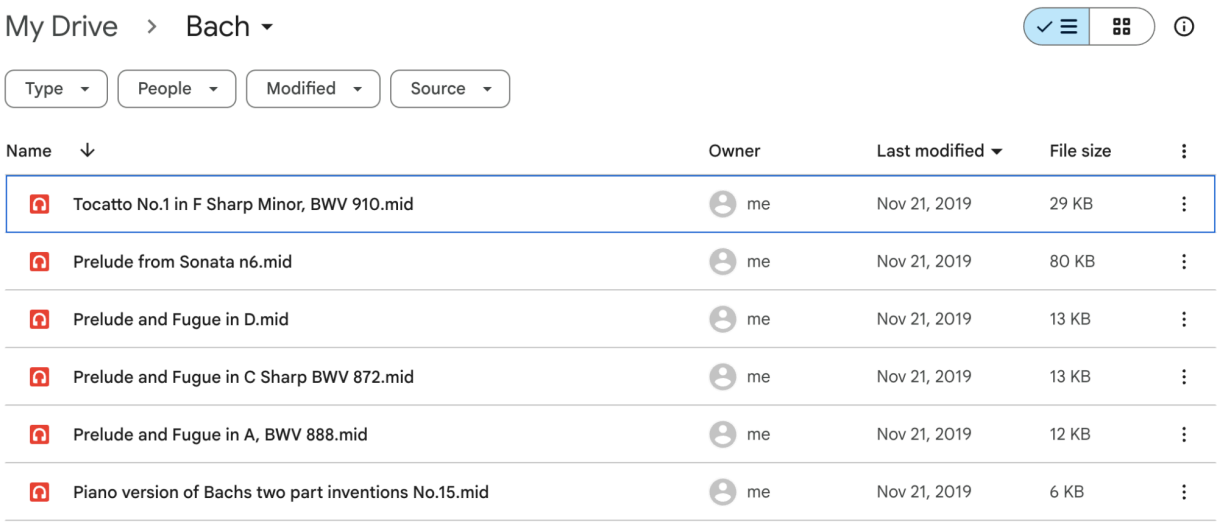

Figure 1. *Data Storage Google Drive.* The figure shows a screenshot representing one of four google folders created for data access and storage in the project.

All song files were initially downloaded in MIDI format, a digital file that stores information on the note pitches, velocities and duration for different components of the song (BassGorilla, 2024). For optimal machine learning performance, each MIDI file was converted to a piano roll, a time-frequency matrix representation of musical data. As a matrix of integers, each piano roll features temporal representation of different pitch captured by the MIDI file (Harte, 2006). This is

an optimal format for analysis and prediction by CNN, as each matrix can now easily be processed by the algorithm to predict the original composer.

```
def get_piano_roll(midi_path, fs=100, fixed_length=4000):
    try:
        midi_data = pretty_midi.PrettyMIDI(midi_path)
        piano_roll = midi_data.get_piano_roll(fs=fs)
        piano_roll = piano_roll / 127.0
        current_length = piano_roll.shape[1]
        if current_length < fixed_length:
            padded = np.zeros((pitchrange, fixed_length))
            padded[:, :current_length] = piano_roll
            return padded
        else:
            return piano_roll[:, :fixed_length]
    except Exception as e:
        print(f"Error processing {midi_path}: {e}")
        return None
```

Figure 2. *get_piano_roll() Function.* The figure shows the function utilized to convert a midi file to a standard piano roll. Function generated by LLM (xAI, 2025).

Once converted, each piano roll was appended as an instance to a master pandas dataframe. With each piano roll, a numerical representation of the respective composer. Part of the data pre-processing included the creation of numerical encoding for each composer, which would also serve as the target feature in model training. Each composer was represented by a single integer under the feature “compid” as follows: Bach=1, Beethoven=2, Chopin=3, Mozart=4. In order to account for the inherent nominal order of 1,2,3,4 and the non-nominal nature of composerID, the ‘compid’ feature also underwent a process of one hot encoded labelling to best represent the target/predicted feature. This was facilitated by the to_categorical() function of the tensorflow.keras.util package.

```
#establish data file path, specifiy piano roll hyper param, initialize list for
#data, iterate through directory and convert midi files to piano rolls
#beethoven data
data_dir = '/content/drive/My Drive/Beethoven'
fs = 100
fixed_length = 4000
compid = 2
for file_name in os.listdir(data_dir):
    midi_path = os.path.join(data_dir, file_name)
    piano_roll = get_piano_roll(midi_path, fs, fixed_length)
    if piano_roll is not None:
        composer_id = compid
        data.append({'song': piano_roll, 'composer_id': composer_id})
```

Figure 3. *MIDI File Loading and Conversion.* The figure above shows the code format utilized to extract MIDI files from their respective Google Drive folder and the processing of the MIDI file through the get_piano_roll() function shown above. Screenshot is for Beethoven data.

The final dataset main_music('song': object, 'compid': integer) is composed of two columns and XXXX rows before being split for testing and training on an 80/20 ratio, consistent with the teachings in our course. Splitting of the processed dataset was facilitated by the train_test_split() function of the sklearn.model_selection package. Hyperparameters were set as follows: train_test_split(X, y, test_size=0.2, random_state=42). The final shape of the X data was investigated for accuracy and displays (X, 128, 4000) consistent with number of instances, dimension of the piano roll and timesteps.

```
#define X and Y vars for training
#one-hot encode y vars for training clarity
x = np.array(main_music['song'].tolist())
y = main_music['composer_id'].values - 1
y = to_categorical(y, num_classes=4)
#investigate format
print(y)
print(x.shape)
```

```
[[1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
(5, 128, 4000)
```

```
#split train/test data
xtrain, xtest, ytrain, ytest =
train_test_split(x, y, test_size=0.2, random_state=42)
```

Figure 4. *Training and Target Dataset Division.* The figure shows the code script associated with the division of X and Y data for training, as well as the main train_test_split() employment on the dataframe.

Feature Extraction

In consideration of storage and training time optimization, each piano roll conversion was adjusted to include only the first 40 seconds of each song, or midi file. For optimal audio classification in CNNs, 30-60 second clips have been suggested as the target length (Kong, 2020). At this length piano rolls are not large enough to overwhelm the algorithm training process, yet include integral parts of the audio to provide identifiable portions in training (Kong, 2020). With the piano roll being the only selectable feature for training, the process of adjusting this length and extracting optimal components of each roll composed most of the

feature-extraction process. In order to specify this adjustment in the project script, two hyperparameters needed to be adjusted (fs=100, fixed_length=4000).

Consistent with traditional CNN models, the majority of the feature extraction process occurs within the convolutional neural network training itself. As each piano roll was processed as input ‘image’ data (128 pitches x 4000 time steps), the convolutional neural network model applies a series of filters and kernels to each piano roll image.

Model Architecture

Tensorflow’s keras convolutional neural network was chosen as the optimal model for the CNN portion of the project. The primary CNN model was designed to feature five different layers: Conv2D, MaxPooling2D, Flatten, Dense, and Dropout. In the context of music composition classification, the Conv2D layer excels at identifying local note patterns and low-pitch features of the piano roll. The MaxPooling2D layer provides dimensionality reduction by reducing the computational size of each piano roll, but retains the dominant melody. The Flatten layer transforms any 2d vector to a 1D vector for simplicity in classification. The dropout layer randomly deactivates neurons in training to prevent potential overfitting.The Dense layer computes prediction probability distribution and ultimately provides the ‘most likely’ composer prediction (Kong, 2020).

```
#initialize CNN model, sequential with multiple layers and activation functions
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(xtrain.shape[1], xtrain.shape[2], 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')
])
```

Figure 5. *Convolutional Neural Network (CNN) Model Implementation.* The figure shows the script of code designated to initialize the convolutional neural network portion of the program. Each layer is featured at least once, sometimes twice in the model.

For the long short-term memory component of the project, the goal was to focus on the long term sequential patterns observable in the dataset. To best reflect this, the LSTM model was initialized first with the LSTM128 layer, dimensions consistent with the size of the piano roll input data. Next an additional LSTM64 layer was added to the model to emphasize higher level abstractions and patterns in the piano rolls. Next a Dense layer to integrate ornamentation or complex music patterns. A 0.5 Dropout layer was included to randomly deactivate neurons and prevent overfitting in the LSTM model. Lastly, a final Dense layer to compute prediction probabilities, emphasizing long term temporal relationships in the input data (Xia, 2020).

```
#initialize LSTM model, sequential with multiple layers and activation functions
lstm_model = Sequential([
    LSTM(128, input_shape=(xtrain.shape[1], xtrain.shape[2]), return_sequences=True),
    LSTM(64),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')
])
```

Figure 6. *Long Short-Term Memory Model Implementation.* The figure shows the script of code designated to initialize the LSTM neural network portion of the program. Each layer depicted is featured at least once, sometimes twice in the model.

Epoch size and activation functions were kept in alignment with standard settings for the first round of model training and fitting. Adjustments to these hyperparameters were made in the model optimization portion of the project and are outlined in sections below.

Training, Reproducibility and Performance Enhancement

Both the CNN and LSTM models were trained on the same ‘xtrain’ portion of the dataset that was identified in the code above. After training, both models underwent a prediction based test process with the ‘xtest’ set. Initial results indicated there was significant room for improvement in both the CNN and LSTM models, hyperparameter adjustments were made and outlined below.

First, to improve the CNN model, batch normalization was added to the program. This allows for the model to preserve different aspects of each layer after processing. This serves as an important inclusion to the CNN as it allows for greater complexity in model training and specificity per piano roll (Kong, 2020)l. Additionally, average pooling was also added to the modified CNN model. Average pooling allows for greater convergence in piano roll classification. Another convolutional layer with 128 filters was also added to the CNN model to promote identification of complex features that the previous layers may have missed (Kong, 2020).

```
#hyperparam adjustments and retraining for CNN:
from tensorflow.keras.layers import BatchNormalization, Activation, AveragePooling2D
cnn_model2 = Sequential([
    Conv2D(32, (3, 3), input_shape=(xtrain.shape[1], xtrain.shape[2], 1)),
    BatchNormalization(),
    Activation('relu'),
    AveragePooling2D((2, 2)),
    Conv2D(64, (3, 3)),
    BatchNormalization(),
    Activation('relu'),
    AveragePooling2D((2, 2)),
    Conv2D(128, (3, 3)),
    BatchNormalization(),
    Activation('relu'),
    AveragePooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')
])
```

Figure 7. *Adjusted CNN Model Implementation.* The figure shows the script of code utilized to adjust the CNN model with chosen optimizations.

To improve the LSTM model, a dropout layer was added to the model in an attempt to assist with any overfitting during the training process. Additionally, the units of the second LSTM layer were increased to 128, in an attempt to capture more intricate relationships in the temporal patterns of each composer (Zhang, 2022). Future directions for both models include an extensive trial at different epoch values to test whether longer training processes improve model performance.

```
#hyperparam adjustments and retraining for LSTM:
lstm_model2 = Sequential([
    LSTM(128, input_shape=(xtrain.shape[1], xtrain.shape[2]),
         return_sequences=True, dropout=0.3),
    LSTM(128, dropout=0.3),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')
])
```

Figure 8. *Adjusted LSTM Model Implementation.* The figure shows the script of code utilized to adjust the LSTM model with final chosen optimizations.

References

BassGorilla. (2024, September 23). *Demystifying MIDI files: A comprehensive guide for music producers* [Web page]. https://bassgorilla.com/midi-files/

Harte, C., Sandler, M., & Gasser, M. (2006). Detecting harmonic change in musical audio. Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia, 21–26. https://doi.org/10.1145/1178723.1178727

Kong, Q., Choi, K., & Wang, Y. (2020). Large-scale MIDI-based composer classification [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2010.14805

xAI. (2025). Grok (Version 2) [Large language model]. https://grok.x.ai

Xia, Y. (2020). Music classification in MIDI format based on LSTM model. arXiv. https://doi.org/10.48550/arXiv.2010.07739

Zhang, J., Pan, J., Yu, S., Zhang, Z., Hu, Z., & Wei, M. (2022). The generation of piano music using deep learning aided by robotic technology. Computational Intelligence and Neuroscience, 2022, Article 8336616. https://doi.org/10.1155/2022/8336616

Appendix A

Participant Contributions

Participant	Jack Baxter
Contributions	Proposal/Preliminary Research - Research topic and algorithm - Written proposal and submission Github Repo/Data Acquisition - Creation of final GitHub repo - Downloading and processing of midi dataset Code/Algorithm Design - Drafted and implemented algorithm for CNN / LSTM with midi data Final Paper & Presentation - Contributed to final paper and code presentation