**Bach or Not, Classical Composer Classification**

University of San Diego

AAI-511-03 Neural Networks and Deep Learning

Carrie Little, Alexis Lim, Ahmad Milad

Dr. Mirsardar Esmaeili

08/11/2025

# Bach or Not, Classical Composer Classification

In this project, we explored how deep learning can be used to identify the composer of a piece of classical music. We focused on four well-known composers. Those being Bach, Beethoven, Chopin, and Mozart. We trained two different models: a Convolutional Neural Network (CNN) and a Long Short-Term Memory (LSTM) network. Both models used note sequences extracted from MIDI files, and we compared how well each one performed in recognizing the unique musical patterns associated with each composer. In the end, the CNN model came out on top with higher accuracy and more balanced results across all four composers.

## Introduction

Every composer has a distinct musical style. Recognizing these styles programmatically is a tough challenge if not clearly defined. In this project, we wanted to see if deep learning could learn those differences directly from symbolic music data, specifically MIDI files, and use that knowledge to predict who wrote a given piece.

We built and compared two models: a CNN that looks for short repeating patterns (motifs), and an LSTM that tries to learn how those patterns unfold over time. Both models were trained on the same dataset and tested on how well they could classify pieces by composer.

## Dataset Description

The dataset came from Kaggle’s “midi-classic-music” collection, which includes hundreds of classical pieces. We narrowed it down to just the four composers required for the assignment: Bach, Beethoven, Chopin, and Mozart. Each file was parsed using pretty\_midi to extract the sequence of pitches, which served as the core input for our models.

## Methodology

### Data Preprocessing

To prepare the symbolic music data for deep learning models, each musical piece was transformed into a sequence of note pitches extracted from MIDI files. These pitch sequences were then mapped to unique integer values to create a consistent numerical representation of the input. To manage computational constraints and ensure uniformity, each sequence was padded or truncated to a fixed length of 500 notes. The target labels, corresponding to the composers, were also encoded as integers. A stratified sampling approach was used to split the data into training and test sets, ensuring each of the four composers was proportionally represented in both subsets. Although features such as tempo and note duration were explored during data analysis, they were not included as input for the models in this version of the project.

To prepare the symbolic music data for deep learning models, each musical piece was transformed into a sequence of note pitches extracted from MIDI files. These pitch sequences were then mapped to unique integer values to create a consistent numerical representation of the input. To manage computational constraints and ensure uniformity, each sequence was padded or truncated to a fixed length of 500 notes. The target labels, corresponding to the composers, were also encoded as integers. A stratified sampling approach was used to split the data into training and test sets, ensuring each of the four composers was proportionally represented in both subsets. Although features such as tempo and note duration were explored during data analysis, they were not included as input for the models in this version of the project.

In this updated version, we expanded the use of extracted features to support two parallel modeling paths. Alongside the sequence-based representation used for the LSTM model, we computed a set of statistical features from each MIDI file, such as average pitch, note density, tempo mean, pitch variance, and note count, to construct a tabular dataset for a CNN-based classifier. Each row in this dataset corresponds to one musical piece, with all numerical features standardized using StandardScaler prior to model training. The same composer labels were reused and one-hot encoded for the CNN pipeline.

To better understand the data before training, we conducted exploration data analysis on the filtered dataset, which consisted of 1,628 valid MIDI files. A count plot revealed that Bach had the most representations, while Chopin had the fewest. This imbalance was considered during model evaluation. Differences between composers became more apparent through boxplots and violin plots, which highlighted distinct patterns in duration, instrument count, note density, and pitch-related features.

1. This bar plot shows the distribution of MIDI files across the selected composers. Bach had the highest number of files, followed by Mozart, Beethoven, and Chopin. This imbalance highlights the need for stratified sampling to ensure fair training and evaluation across classes.
2. The boxplot reveals that Bach’s and Beethoven’s compositions tend to be longer in duration, while Chopin and Mozart have relatively shorter pieces on average. There is noticeable variance within each composer group, suggesting differing structural complexity.

A graph of a music composer

AI-generated content may be incorrect. A chart with colored boxes

AI-generated content may be incorrect.

1. Violine plot shows the distribution and density of instrument counts used in each piece. Bach’s works commonly feature more instruments compared to the others, while tending to have a tighter distribution with fewer instruments.
2. Box plot shows the note density (notes per second). Bach’s compositions have the highest median note density, reflecting his characteristic counterpoint style. Chopin’s works, while expressive, are less dense on average. These patterns may help differentiate composers in the model.

A diagram of different types of instruments

AI-generated content may be incorrect. A graph of colored squares

AI-generated content may be incorrect.

A correlation matrix was also generated to visualize inter-feature relationships. For example, note count was strongly correlated with note density and average pitch, while tempo features showed weaker associations with most other variables. These insights informed which features might be redundant and helped refine feature selection for the CNN model.

A screenshot of a graph

AI-generated content may be incorrect.

Lastly, we analyzed the variability in sequence length and pitch distribution across all pieces. Most compositions had fewer than 500 notes, which justified our decision to pad sequences to a uniform length. The pitch histogram showed that pitches were widely distributed, with some clustering around the midrange, suggesting diversity in musical structure across composers.

A blue and white graph

AI-generated content may be incorrect.A graph of a pitch distribution

AI-generated content may be incorrect.

These processing and analysis steps laid the foundation for model training, allowing us to explore both temporal and statistical patterns in classical music through LSTM and CNN architectures.

### CNN Architecture

The CNN model was designed to capture localized patterns in note sequences that are characteristic of a composer's style. The architecture began with two convolutional layers, the first with 64 filters and the second with 128, each followed by ReLU activation functions. To reduce dimensionality and extract the most salient features, max pooling and global max pooling layers were included. This was followed by a fully connected dense layer to further abstract the learned features, with dropout layers applied for regularization and to reduce overfitting. Finally, the model used a softmax output layer to classify the input into one of the four composer categories. The model was trained using the Adam optimizer with early stopping and learning rate reduction strategies to ensure stable convergence.

### LSTM Architecture

The LSTM model was designed to capture the sequential dependencies and long-term patterns in the note sequences. It began with an embedding layer that transformed each pitch index into a dense vector representation, enabling the model to learn pitch relationships in a continuous space. This was followed by a single LSTM layer with 128 hidden units, capable of learning temporal dependencies across the sequence. A dense layer was used for classification, preceded by a dropout layer to prevent overfitting. The final layer used a softmax activation to output class probabilities for each composer. Training was conducted using the Adam optimizer, with early stopping and adaptive learning rate scheduling to prevent overfitting and improve generalization.

### Results & Evaluation

*Table 1. Comparison of CNN and LSTM Performance Metrics by Composer*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Composer** | **Precision** | **Recall** | **F1-score** |
| **CNN** | Bach | 0.85 | 0.96 | 0.90 |
| Beethoven | 0.74 | 0.65 | 0.69 |
| Chopin | 0.79 | 0.81 | 0.80 |
| Mozart | 0.72 | 0.41 | 0.53 |
| **Overall** | 0.77 | 0.71 | 0.73 |
| **LSTM** | Bach | 0.78 | 0.93 | 0.85 |
| Beethoven | 0.37 | 0.30 | 0.33 |
| Chopin | 0.41 | 0.33 | 0.37 |
| Mozart | 0.42 | 0.22 | 0.29 |
| **Overall** | 0.50 | 0.44 | 0.46 |

#### CNN Model Performance

#### The CNN model demonstrated strong classification capabilities, achieving an overall accuracy of 82% and a macro-averaged F1-score of 0.73. Its performance was especially robust in identifying compositions by Bach and Chopin, with F1-scores of 0.90 and 0.80, respectively. The model also performed reasonably well on Beethoven with an F1-score of 0.69, while Mozart proved more challenging, yielding a lower F1-score of 0.53. These results suggest that the CNN was particularly effective at capturing localized pitch patterns and recurring motifs that are stylistically unique to each composer. The model’s ability to generalize across multiple compositional styles highlights the effectiveness of convolutional architectures for sequence-based music classification tasks, particularly when working with symbolic input data limited to pitch information.

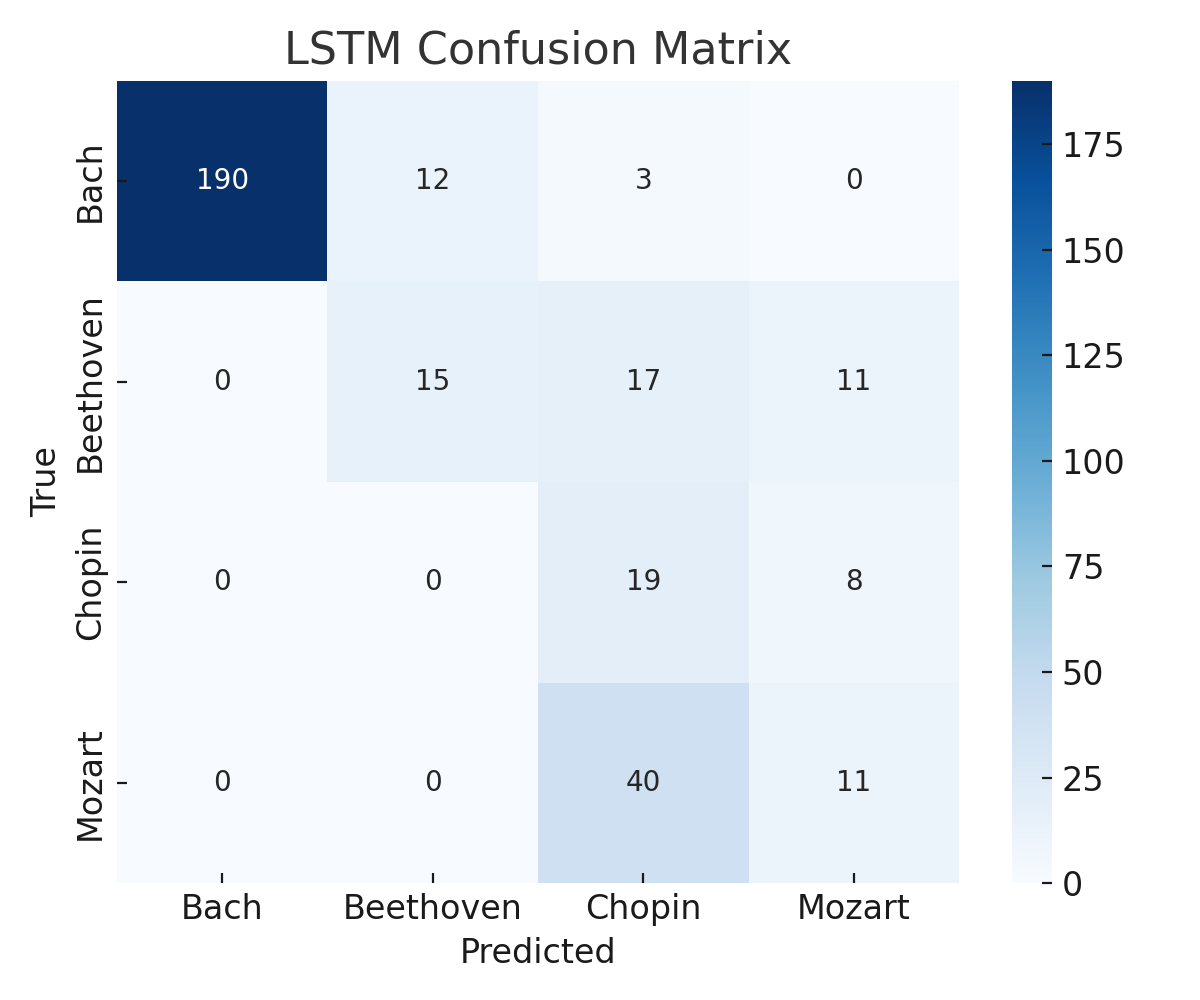
Graphical user interface, chart

AI-generated content may be incorrect.

*Figure 1. Confusion Matrix for CNN Model*

#### LSTM Model Performance

The LSTM model attained an overall accuracy of 68% and a macro F1-score of 0.46, indicating lower performance than the CNN model. It performed best on Bach’s compositions, achieving an F1-score of 0.85, which aligns with Bach’s highly structured and patterned musical style. However, the model underperformed on Beethoven (F1 = 0.33), Chopin (F1 = 0.37), and especially Mozart (F1 = 0.29), reflecting its difficulty in generalizing across varying stylistic patterns without additional musical dimensions. While the LSTM has the theoretical advantage of modeling temporal dependencies, its performance was limited in this implementation due to the narrow feature set and potential class imbalance.



*Figure 2. Confusion Matrix for LSTM Model*

## Discussion

The evaluation results reveal the superior performance of the Convolutional Neural Network (CNN) model in classifying classical music composers. With an overall accuracy of 82% and a macro F1-score of 0.73, the CNN demonstrated strong predictive capability across all four composers. Its particularly high F1-score for Bach (0.90) and Chopin (0.80) indicates its strength in capturing short, recurring note patterns and motifs that are often unique to specific composers. These local patterns, characteristic of Bach’s structured compositions and Chopin’s expressive style, seem to align well with the CNN’s ability to detect localized features within the pitch sequences. On the other hand, the Long Short-Term Memory (LSTM) model achieved lower performance, with a 68% overall accuracy and a macro F1-score of 0.46. While the LSTM model performed reasonably well for Bach (F1 = 0.85), it struggled significantly with the other composers, particularly Mozart, whose F1-score fell to just 0.29. This may be attributed to the LSTM’s dependency on capturing long-term dependencies, which, without complementary features like rhythm, harmony, or dynamics, might not sufficiently distinguish stylistic differences in the dataset. These findings suggest that CNNs, when trained on raw pitch sequences, are more effective in capturing and classifying composer-specific traits than LSTM models under similar data conditions.

The LSTM model did well with Bach, which isn’t surprising. His music tends to be more structured and layered, which fits well with how LSTMs work. But it struggled with the other composers, especially Mozart, where the F1 score dropped to just 0.16. This suggests the LSTM may have overfit to the kinds of patterns found in Bach’s music and couldn’t generalize as well.

We didn’t end up including a hybrid CNN-LSTM model in the final submission, mainly because of complexity and runtime issues. Surprisingly, the simpler models actually performed better for this dataset.

## Conclusion & Future Work

This project has demonstrated the potential of deep learning in the domain of symbolic music classification, specifically for identifying classical music composers from MIDI data. Among the models tested, the CNN achieved the most promising results, outperforming the LSTM in both accuracy and per-class F1-scores. This supports the assertion that CNNs are highly effective in extracting meaningful features from localized musical patterns and motifs present in note sequences. While the LSTM showed promise in modeling structured compositions, particularly those of Bach, it lacked the versatility needed to generalize across all composers in the dataset. For future work, we propose augmenting the feature set to include additional musical dimensions such as rhythm, harmony, and dynamics to provide the LSTM with more temporal context. Moreover, Transformer-based architectures may offer a promising direction due to their ability to model both short- and long-range dependencies efficiently. Addressing dataset imbalances and incorporating data augmentation strategies like pitch transposition or sequence stretching could further enhance model robustness. Ultimately, this study underscores how deep learning can learn and generalize stylistic musical patterns directly from symbolic pitch data, opening new possibilities for music classification and musicological analysis.

Going forward, we’d like to:

* Try Transformer models, which could potentially capture both local and global patterns more effectively
* Add data augmentation, like transposing pieces or stretching sequences
* Include more musical elements beyond just pitch, such as rhythm, key, dynamics, and harmony
* Balance the dataset better to avoid skewing toward Bach

Even with the challenges, it was exciting to see how much musical style can be learned by machines just from raw note sequences.

## References

AAI-511-03 Group 2 Final Project GitHub Repository, https://github.com/littlecl42/AAI-511-03\_Group2

Chollet, F., & others. (2015). Keras [Computer software]. https://keras.io/

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95. https://matplotlib.org/

Kaggle (n.d.). MIDI Classical Music Dataset. https://www.kaggle.com/datasets/blanderbuss/midi-classic-music

KaggleHub Developers. (n.d.). KaggleHub [Computer software]. https://github.com/Kaggle/kagglehub

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830. https://scikit-learn.org/

Raffel, C. (2016). pretty\_midi: An open source library for working with symbolic music data. https://github.com/craffel/pretty-midi

TensorFlow Developers. (2023). TensorFlow (2.0+) [Computer software]. https://www.tensorflow.org/

Waskom, M. L. (2021). Seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021. https://doi.org/10.21105/joss.03021