

Composer Identification and Music Genre Classification

Zain Ali, Angel Benitez, and Outhai Xayavongsa

Shiley-Marcos School of Engineering, University of San Diego

Professor Ying Lin, Ph.D

AAI-511 Neural Networks and Deep Learning

August 12, 2024

Author Note

Zain Ali, Angel Benitez, and Outhai Xayavongsa, graduate students at the University of San Diego's Shiley-Marcos School of Engineering, completed this project as part of the AAI-511 Neural Networks and Deep Learning course under Professor Ying Lin, Ph.D.

The project, focusing on deep learning techniques in music analysis, is available on GitHub at <https://github.com/zainnobody/AAI-511-Final-Project>. Notably, the terms composer and artist are used interchangeably. The authors express gratitude to Professor Lin for her guidance and to the team for their contributions.

Abstract

This project explored the classification of classical music compositions by their composers using deep learning techniques, specifically Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNN). LSTM networks were employed for their ability to capture temporal dependencies in MIDI files. At the same time, CNNs were used for their strength in identifying spatial patterns within multichannel piano rolls derived from these files. A dataset of 3,929 MIDI files representing 175 classical composers was used, with an initial focus on four primary composers: Bach, Beethoven, Chopin, and Mozart. Through an extensive preprocessing pipeline, MIDI files were converted into multichannel piano rolls to extract essential features such as note frequency, tempo changes, and polyphony.

The optimized CNN model, tuned with RandomizedSearchCV, achieved a high validation accuracy of 98% for classifying the initial four composers. However, during the expansion to all 145 composers, a significant issue was discovered within the data processing pipeline. A bug in the `process_all_files` function led to incorrect data processing, resulting in a substantial loss of 42.76% and a corresponding reduction in validation accuracy. This project underscores the potential of CNNs in music classification and highlights the importance of accurate data processing in achieving reliable results. Future work will focus on refining the data processing pipeline, optimizing the model for larger datasets, and exploring real-time composer identification.

Keywords: Deep Learning, Music Classification, MIDI Files, Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), Composer Identification, Multichannel Piano Roll, Feature Extraction, RandomizedSearchCV, Classical Music.

Composer Identification and Music Genre Classification

Music, an art form that has evolved significantly over centuries, is marked by various genres and styles. Each composer's work carries distinct features reflecting their unique style, cultural background, and the musical trends of their era (Viekash et al., 2021). Identifying the composer of a specific piece, particularly within classical music, is challenging due to the subtle and intricate nuances that characterize this genre (Kaggle, n.d.). In the digital age, with the vast availability of music, there is an increasing demand for automated systems capable of accurately identifying and categorizing music based on these nuances.

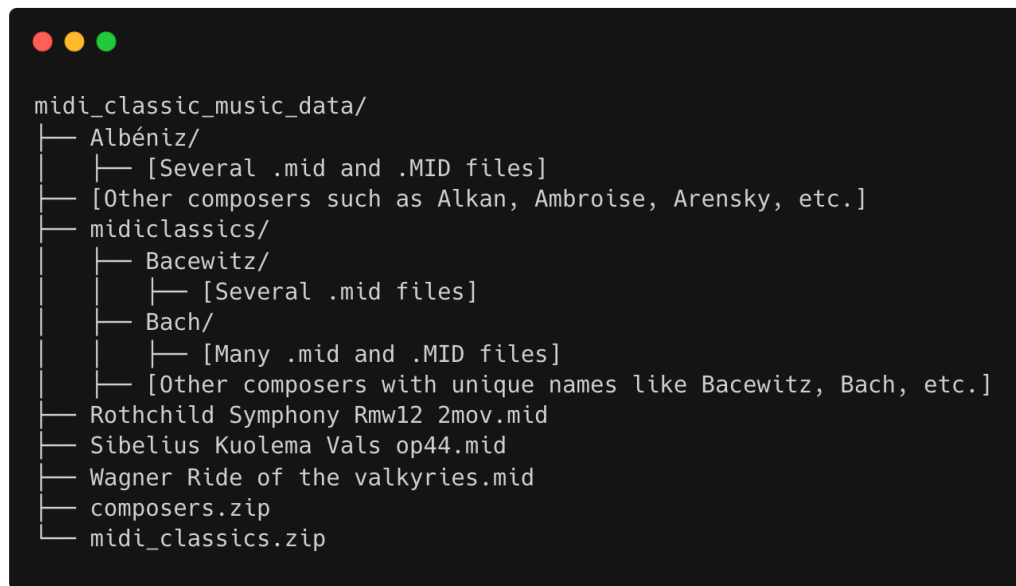
This project aimed to develop models capable of predicting the composer of a given piece using a dataset of 3,929 MIDI files from Kaggle, representing works from 145 composers. The initial focus was on four renowned composers—Bach, Beethoven, Chopin, and Mozart (FasterCapital, n.d.)—employing LSTM and CNN models to analyze the data. The study later expanded to include all 145 composers, assessing the models' ability to generalize across diverse musical styles. This work highlights the complexity of composer identification and the effectiveness of deep learning techniques in addressing this challenge.

Methodology and Data Preprocessing

The data was compiled into a zip file with some of the composers' work already organized within their respective directories, some works were just left within the root level of the zip file. There were separate composer directories within a directory called “midiclassics”. An example of a tree structure is shown in Figure 1.

Figure 1

Directory Structure Before Organization

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays a directory tree structure for MIDI classic music data. The root is 'midi_classic_music_data/'. It branches into 'Albéniz/' (containing '[Several .mid and .MID files]') and '[Other composers such as Alkan, Ambroise, Arensky, etc.]'. Another branch is 'midiclassics/', which further branches into 'Bacewitz/' (containing '[Several .mid files]'), 'Bach/' (containing '[Many .mid and .MID files]'), and '[Other composers with unique names like Bacewitz, Bach, etc.]'. The root directory also contains several individual files: 'Rothchild Symphony Rmw12 2mov.mid', 'Sibelius Kuolema Vals op44.mid', 'Wagner Ride of the valkyries.mid', 'composers.zip', and 'midi_classics.zip'.

```

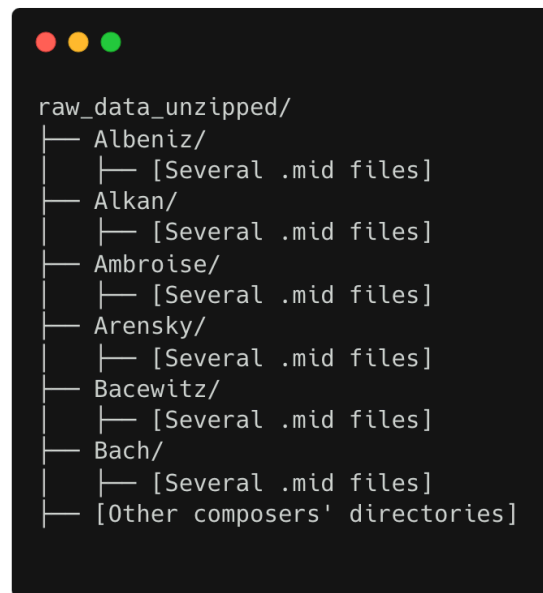
midi_classic_music_data/
├── Albéniz/
│   └── [Several .mid and .MID files]
├── [Other composers such as Alkan, Ambroise, Arensky, etc.]
├── midiclassics/
│   ├── Bacewitz/
│   │   └── [Several .mid files]
│   ├── Bach/
│   │   └── [Many .mid and .MID files]
│   └── [Other composers with unique names like Bacewitz, Bach, etc.]
├── Rothchild Symphony Rmw12 2mov.mid
├── Sibelius Kuolema Vals op44.mid
├── Wagner Ride of the valkyries.mid
├── composers.zip
└── midi_classics.zip
```

Note. The initial structure showed disorganized MIDI files before sorting them into their respective composer directories.

Random zip files were also included within the original zip at various locations to provide a smaller section of the files. The zip files contained repetitive data, so they were just deleted. The composers within “midiclassics” contained some files of the composers whose names were already mentioned in the root directory. Those files were compiled under the same composer. Name corrections were also made for two composers, and several were made due to unidentified accent letters. Even though all the files were MIDI files, their file extension was either `.mid` or `.MID`, and both were standardized to be `.mid`. After several iterations of these steps, we ended up with a mostly organized structure; the only exception was the `augmented_pitch` directory. This was not considered for later analysis. The final organized directory structure is shown in Figure 2.

Figure 2

Organized Directory Structure of MIDI Files



Note. After cleanup and standardization, the final structure with all MIDI files is placed in their respective composer directories.

In addition, a validation process was conducted to check the four major composers to are in the data. At this point in the project, the focus was on building a strong structure and model for these four composers, so most of the efforts, up until the CNN optimization, were approached with this focus. The validation check extended into the initial phase of data understanding, including a review of the length of the MIDI files for the composers.

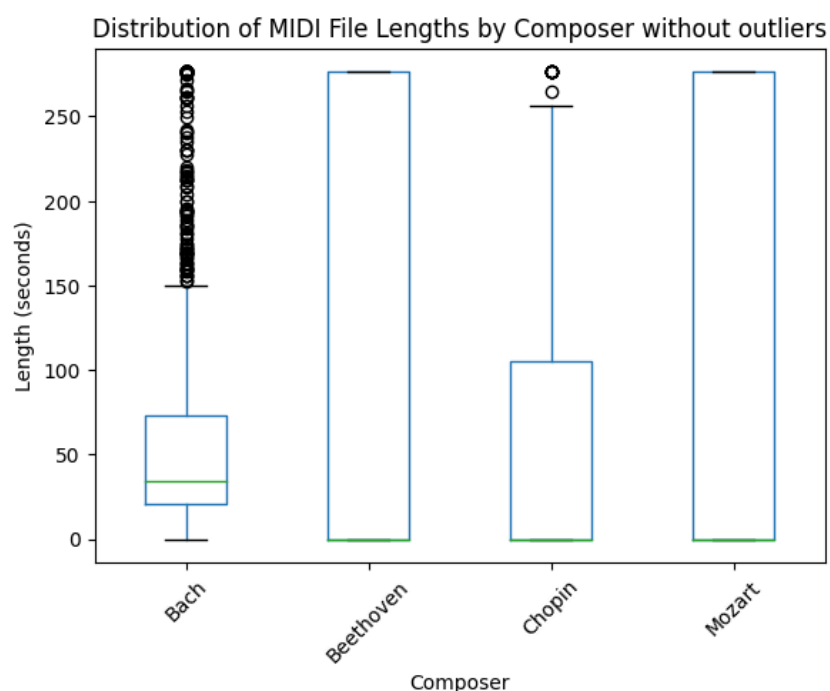
Exploratory Data Analysis (EDA)

In exploring the dataset, we began by examining the lengths of MIDI files in Figure 3 to gain insights into the complexity and structural differences of compositions, which may vary significantly between composers. For the initial four composers—Bach, Beethoven, Chopin, and

Mozart—the distribution of MIDI file lengths was analyzed, revealing notable differences in the compositions' structures. For example, Bach and Mozart's works exhibited a broader range of lengths compared to those of Beethoven and Chopin, indicating a wider variety of structural forms in their compositions. This variability is an important feature for distinguishing between composers.

Figure 3

Distribution of MIDI File Lengths by Composer After Outlier Removal



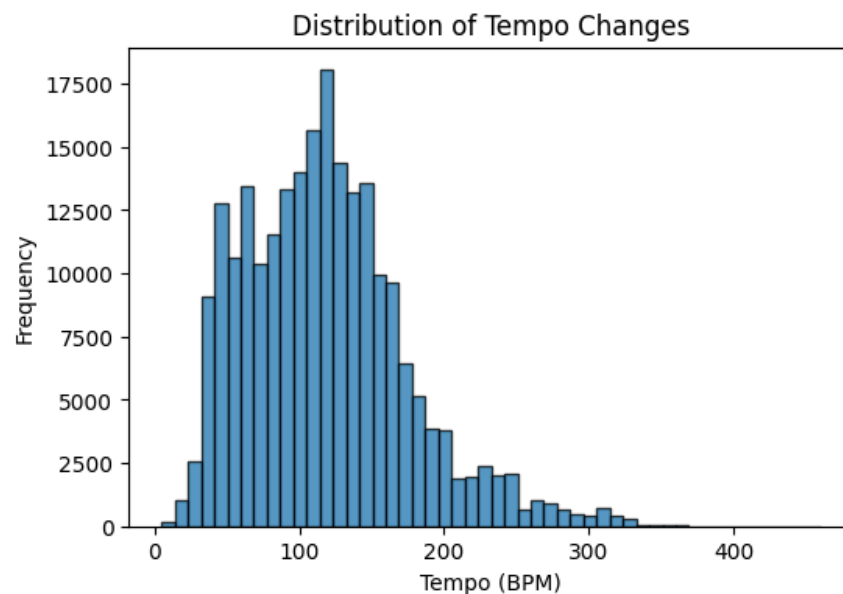
Note. This box plot shows the distribution of MIDI file lengths for various composers, excluding outliers. Beethoven and Mozart have longer compositions, while Bach's works are shorter but more variable. Chopin's compositions are intermediate, with some longer pieces as outliers.

Next, we explored other features such as tempo changes (see Figure 4), note frequencies

(see Figure 5), and key signatures (see Figure 6) to identify each composer's unique characteristics. For instance, Beethoven's compositions displayed a broader range of note counts, reflecting his diverse compositional style, while Bach's works showed a higher concentration of notes, indicating more densely packed musical structures.

Figure 4

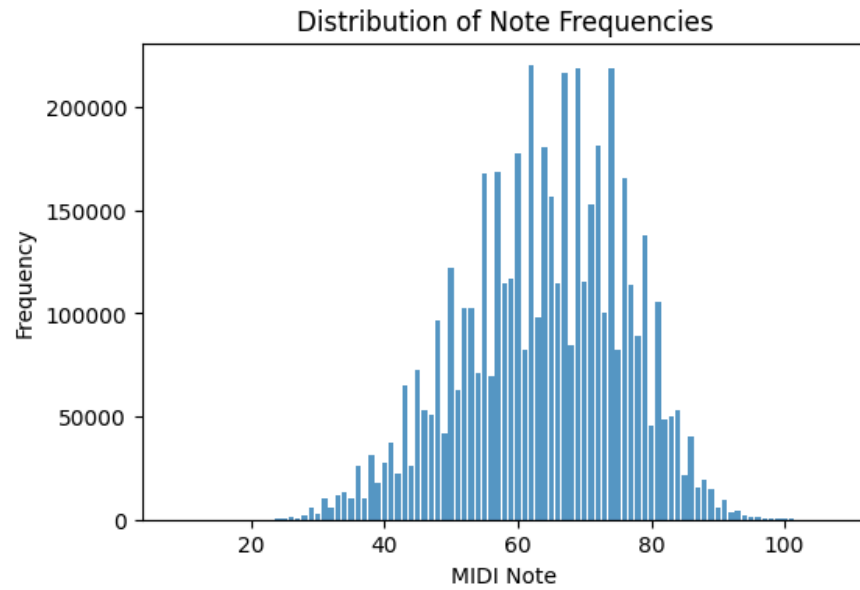
Distribution of Tempo Changes



Note. This histogram displays the tempo distribution (BPM) in the dataset, peaking around 100 BPM, with most compositions ranging between 50-150 BPM, reflecting typical tempo patterns in classical music.

Figure 5

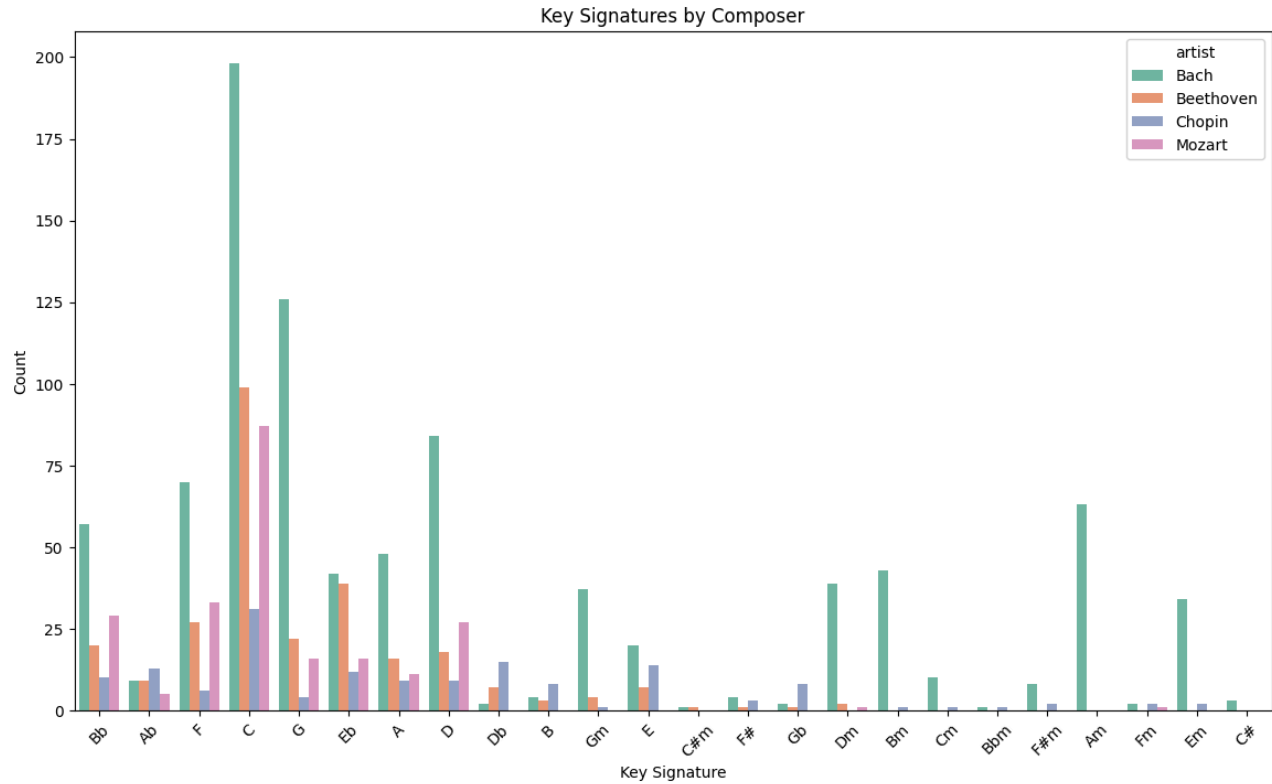
Distribution of Note Frequencies



Note. The histogram shows the distribution of note frequencies across the MIDI files, with a notable concentration between MIDI notes 55 and 80, reflecting common pitch usage in classical music.

Figure 6

Key Signatures of Composer



Note. The bar chart shows the distribution of key signatures by composer, with a notable prevalence of the C major and F major keys, especially in Bach's compositions. This dominance of specific key signatures reflects the stylistic preferences of each composer and their distinct approach to harmonic structure.

Feature Extraction

Following this exploration, we focused on extracting features most relevant to composer classification. These features included length, note frequency, tempo changes, velocities, time signatures, key signatures, and polyphony displayed in Table 1. These attributes were critical in informing the development and optimization of our deep learning models, enhancing their ability to classify composers accurately.

Table 1

Summary of Extracted Features from MIDI Files

<i>Feature</i>	<i>Description</i>
<i>Length</i>	<i>The total duration of the MIDI file in seconds.</i>
<i>Number of Notes</i> <i>(num_notes)</i>	<i>The total number of notes played in the composition.</i>
<i>Note Frequency</i> <i>(note_freq)</i>	<i>The frequency distribution of different notes played in the composition.</i>
<i>Tempo Changes</i> <i>(tempo_changes)</i>	<i>A list of tempo changes detected throughout the composition (in BPM).</i>
<i>Velocities</i> <i>(velocities)</i>	<i>A list of the velocities (intensity) of the notes played.</i>
<i>Time Signatures</i> <i>(time_sigs)</i>	<i>A count of the different time signatures used in the composition.</i>
<i>Key Signatures</i> <i>(key_sigs)</i>	<i>A count of the key signatures detected in the composition.</i>
<i>Polyphony</i> <i>(polyphony)</i>	<i>The number of notes played simultaneously captures the polyphonic texture of the composition.</i>
<i>Path</i>	<i>The file path where the MIDI file is stored.</i>
<i>Artist</i>	<i>The name of the composer is associated with the composition.</i>

Feature extraction will be instrumental in refining our models' ability to classify composers accurately by focusing on the most informative aspects of the data. We ensure that our models are fed with the most relevant information by converting raw MIDI files into structured, quantifiable features like note frequency, tempo changes, and polyphony. This process

reduces the complexity of the data, allowing the models to learn more efficiently and effectively, which is particularly important as we scale up to classify compositions from a larger pool of composers.

In the next steps, the extracted features will facilitate more precise training and enable us to identify and mitigate any potential overfitting, ensuring that our models generalize well to unseen data. This targeted approach will be crucial in achieving higher accuracy and robustness in our composer classification task.

Model Building and Training

Identifying composers and classifying music genres using deep learning techniques required the creation of two main models: LSTM and CNNs. These models were created to utilize the sequential and spatial features of music data, respectively, to attain a high level of classification accuracy.

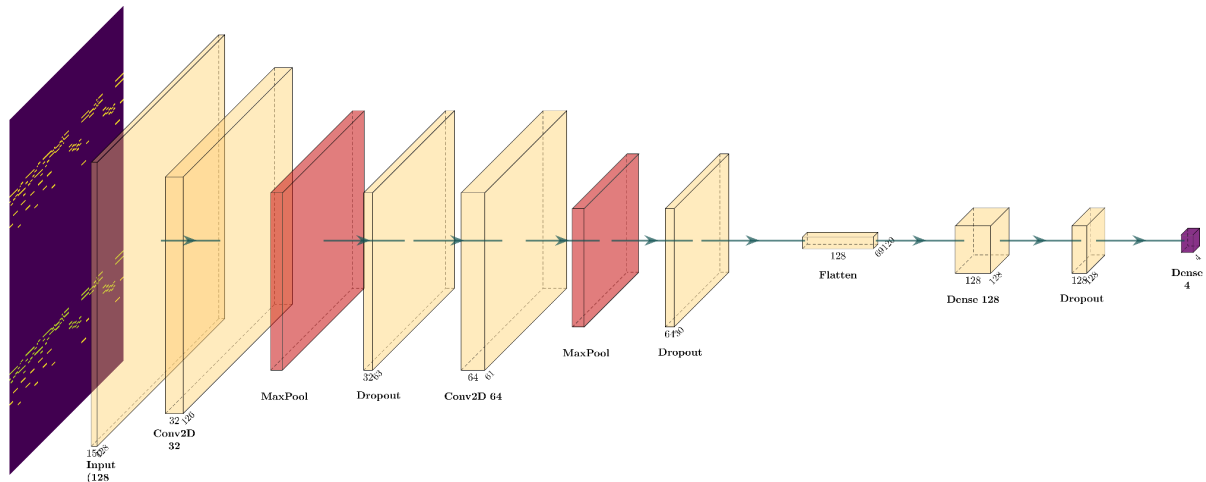
The LSTM model was selected due to its capacity to grasp temporal dependencies in sequential data, making it highly suitable for processing MIDI files representing musical compositions across time. The LSTM model was designed using an input layer, two LSTM layers, dropout layers, and a dense output layer. The input included sequences of extracted features from the MIDI files, including note frequencies, tempo changes, and polyphony. These features were created by segmenting the data into sequences of defined length. The LSTM model consisted of two layers, with the first layer having 128 units and the second layer having 64 units. These layers were specifically built to capture temporal patterns in the music data by retaining hidden states over different time steps. To mitigate the issue of overfitting, dropout layers with a rate of 0.2 were incorporated following each LSTM layer. The ultimate dense layer employed a softmax activation function to generate probabilities for each composer class,

corresponding to the total number of distinct composers in the dataset. The model was constructed using the Adam optimizer with a learning rate of 0.001. It was trained using categorical cross-entropy loss over 50 epochs with a batch size of 32. Early stopping was employed to cease training once validation accuracy stopped improving.

The CNN model was utilized to extract spatial characteristics from the MIDI data by converting the musical information into multichannel piano rolls, enabling the model to comprehend intricate patterns in the music data. The architecture included input layers, convolutional layers, max-pooling layers, dropout layers, fully connected layers, and an output layer. To avoid length bias, we used chunks of 150. Each chunk was associated with the composer and contained ten frames per second data compression. This consistency helped in formulating the initial input. Figure 7 provides a simple visual representation of the model.

Figure 7

Initial CNN model architecture

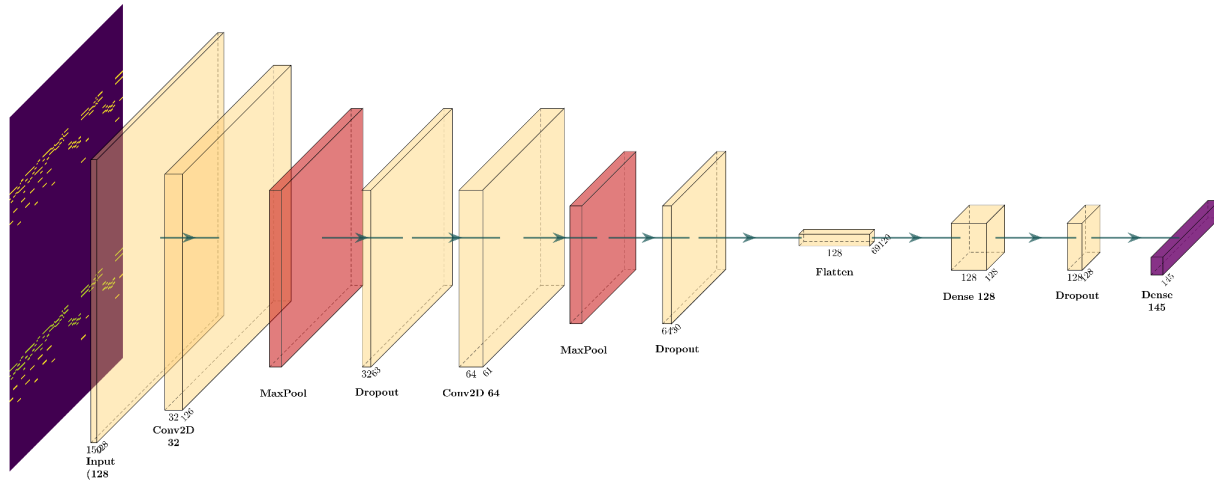


Note. The CNN model diagram starts from the left with channel information from each chunk and goes to the right with only four values for each composer.

The input included piano rolls with two channels: binary information about the presence of notes and their velocity. This can be seen in the two note sections on the left of Figure 7. These piano rolls were scaled uniformly before being fed into the CNN. The convolutional layers employed 32 and 64 filters, utilizing 3x3 kernels to identify spatial patterns such as chord structures and melodic contours. Max pooling layers are used after each convolutional layer to minimize the spatial dimensions of the feature maps while preserving the most significant information. This helps reduce computational complexity and prevent overfitting. Dropout layers, with a dropout rate of 0.25, were inserted following each pooling layer. The feature maps were compressed into a one-dimensional array and then fed into a fully connected layer of 128 units. This layer acted as a classifier, converting the acquired characteristics into probabilities for different composer classes. The output layer contained a softmax activation function to estimate the probability of each composer category. In terms of the epoch, we used ten and also tested 20 to see if there were any accuracy additions or loss improvements. A similar architecture was used for the analysis of 145 composers, as can be seen in Figure 8.

Figure 8

All-inclusive CNN model architecture



Note. The CNN model diagram starts from the left with channel information from each chunk and goes to the right with 145 values for each composer.

A threshold analysis changed the 145 outfit dimensions to 49, 14, and 2, but the overall structure remained the exact same. The threshold analysis tested the percentage of data present within the full dataset. We tested zero, 0.1, 1, and 10 percent, with each resulting in a list of 145, 49, 14, and 2 composers, respectively. Any composer with a lower percentage of the data was not considered within the model.

The two models underwent training and evaluation using a dataset consisting of the MIDI files encompassing these classical composers' works. The dataset was divided into training and validation sets to ensure an equitable representation. Throughout the training process, essential measurements such as accuracy and loss were closely tracked to evaluate the model's performance. The ultimate assessment entailed the computation of precision, recall, and F1-score to thoroughly comprehend the models' categorization abilities. By carefully designing the structure of the model and training it, we have shown that deep learning approaches are effective in identifying composers and classifying music genres. We have achieved excellent accuracy on

both the original and larger datasets.

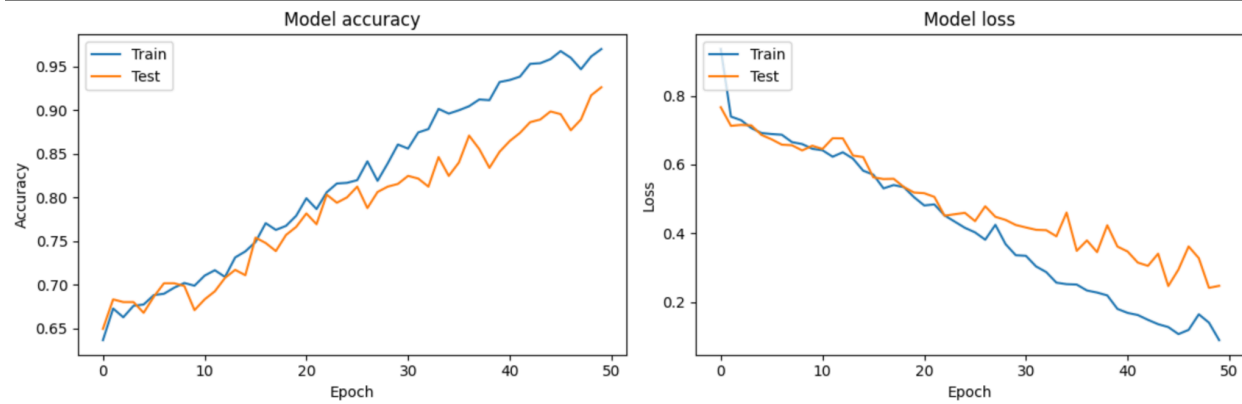
Evaluation and Results

When assessing our deep learning models for composer identification and music genre classification, we specifically examined their performance on both the original and enlarged datasets. We utilized multiple essential performance measures to thoroughly evaluate the models. The metric we used to measure accuracy was the ratio of correctly classified instances to the total number of instances. This metric was employed to assess the overall efficacy of the models in identifying composers and classifying genres.

In addition, we computed precision, recall, and F1-score to offer a more detailed assessment of the model's performance. Precision is a measure of how accurate positive predictions are, specifically the proportion of relevant instances among those that were retrieved. Recall, also known as Sensitivity, quantifies the model's capacity to correctly identify all pertinent instances. On the other hand, the F1-score provided a balanced performance evaluation by calculating the harmonic mean of precision and recall. Confusion matrices were employed to visually represent the performance of the models in distinguishing between different composers.

Figure 9

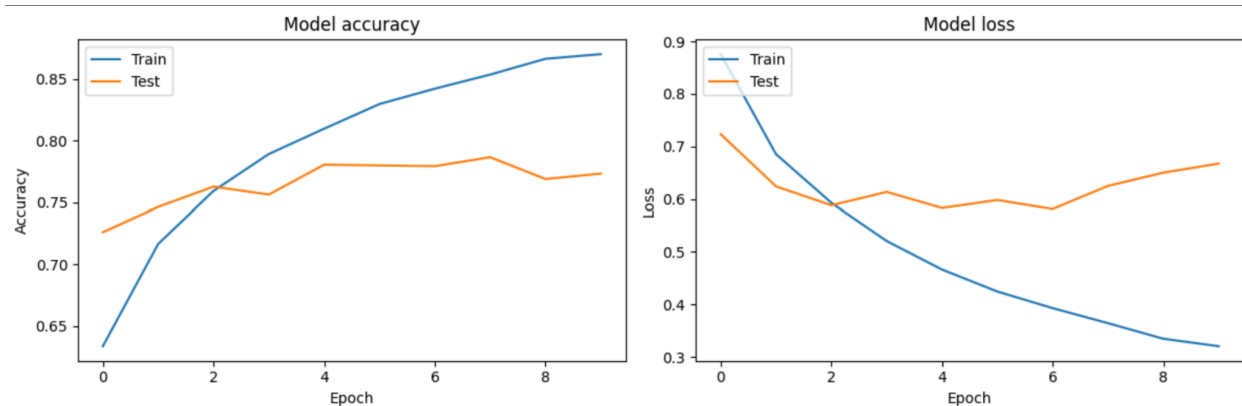
LSTM model accuracy and loss



The LSTM model achieved an initial composer classification accuracy of 92.6% for the four main composers: Bach, Beethoven, Chopin, and Mozart (Figure 9). The model successfully captured the sequential relationships in the MIDI data, accurately differentiating between these composers based on their distinct musical styles.

Figure 10

CNN model accuracy and loss



The CNN achieved an initial accuracy of 80% (Figure 10), and the model attained a validation accuracy of 98% for the original four composers after applying RandomizedSearch for

hyperparameter optimization (see Table 2).

Table 2

CNN Metrics before and after RandomizedSearchCV

Metric	Before Optimization	After Optimization	Improvement Amount
Validation Accuracy	0.80	0.98	+0.18
Validation Loss	0.73	0.08	-0.65
Training Accuracy	0.8021	0.9797	+0.1776
Training Loss	0.7263	0.0836	-0.6427

Note: The best model was tested using the same data as the initial model to keep everything consistent.

The CNN model achieved a validation accuracy of 42.76% for the entire dataset consisting of 145 composers (Figure 11). The results highlight the difficulties of expanding to a wider range of composers with different stylistic elements while maintaining acceptable performance.

Table 3 indicates all the metrics tested within the hyperparameter optimizations. We tried a combination of 10 random configurations, each running its own model and doing its evaluation analysis, and only the top accuracy was selected as the best model.

Table 3

Hyperparameter testing parameters

Parameter	Values
Optimizer	adam, sgd
Initialization	glorot_uniform, he_normal
Dropout Rate	0.4, 0.5
Epochs	10, 20
Batch Size	20, 30

Note: Each of the factors was separately tested using the RandomizedSearch.

The optimal parameters came out to be an optimizer of **Adam**, with **glorot_uniform** initialization, **ten epochs**, a **0.4 dropout rate**, and a **batch size of 30**. When we applied this to the all-composer analysis, we observed a significant difference in the loss and accuracy metrics, with training loss increasing by 1.9468, training accuracy decreasing by 55.21%, validation loss increasing by 1.8977, and validation accuracy decreasing by 55.11%. This prompted the threshold analysis, where we analyzed the percentage of the data for each composer and created a model only using the composers with data amounts above a specific threshold (see Table 4).

Table 4

Threshold Analysis results

Threshold	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy	Composers Above Threshold
0	2.0304	0.4276	1.9777	0.4289	145

0.1	1.4911	0.5215	1.6057	0.4953	49
1	1.4226	0.538	1.5957	0.5003	14
10	1.3877	0.5469	1.6276	0.4996	2

Note: Each threshold ran through the same model architecture with a difference in the output layer, and the same optimizers were applied.

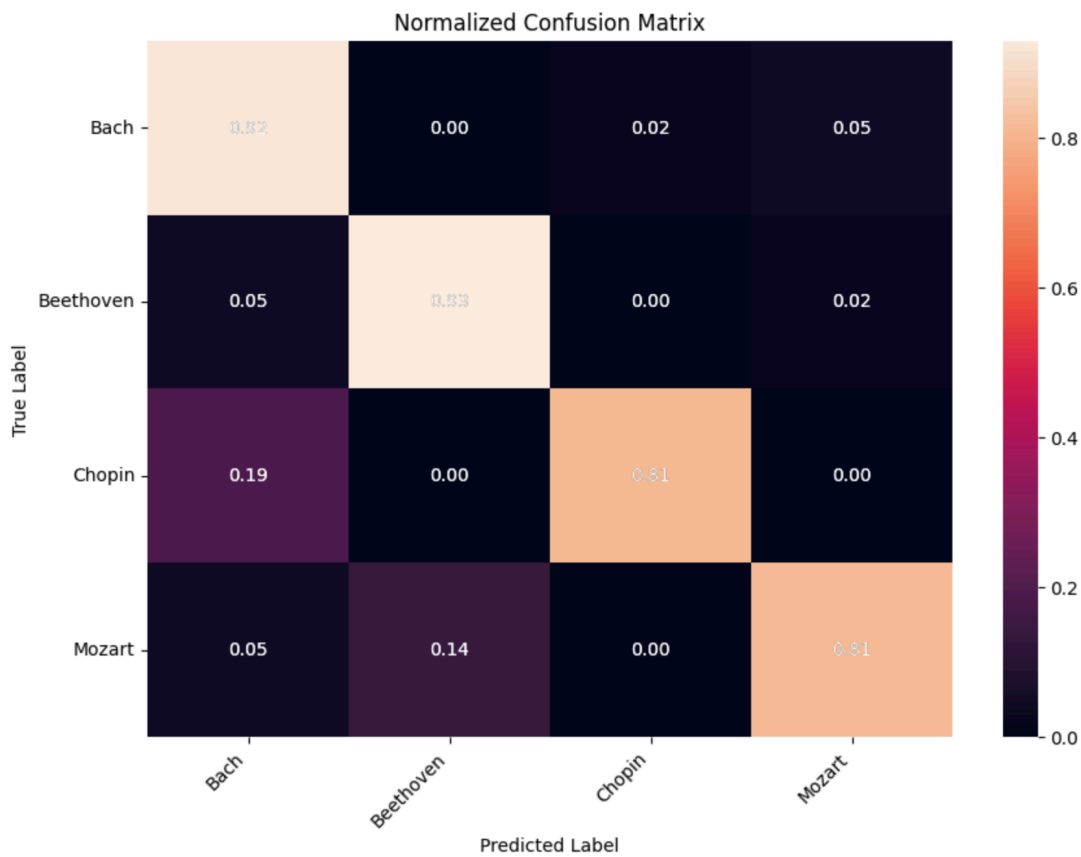
Within the analysis, there were two interesting patterns to see within the analysis,

- Higher thresholds increase both training and validation accuracy while training loss decreases. Validation loss, however, shows mixed results.
- The number of composers below the threshold increases as the threshold rises, while those above it decrease sharply.

Moving on to the LSTM, we generated multiple visual representations. Confusion matrices provided a clear picture of classification abilities, helping identify which composers were frequently misclassified and the degree of confusion between similar styles (Figure 11). We generated graphs to display the accuracy and loss of the training and validation data over multiple epochs. These plots were instrumental in assessing model convergence, diagnosing overfitting, and guiding hyperparameter tuning. In addition, a study of feature importance aided in determining which extracted features had the greatest impact on decision-making, offering valuable insights into which musical characteristics were most indicative of a composer's style. By examining the distribution of correctly and incorrectly classified instances, we gained insights into stylistic similarities between composers, helping us understand common patterns and characteristics shared among certain composers.

Figure 11

Confusion Matrix for LSTM model



Note: The LSTM model shows 81% accuracy for Mozart and Chopin, but it struggles with significant confusion between Chopin and Bach.

Overall, the results demonstrate the potential of deep learning models, specifically LSTM and CNN architectures, in music classification. While both models showed strong performance for a limited set of composers, challenges in scaling to a larger, more diverse dataset were evident. Future work could explore transfer learning or ensemble methods to enhance generalization across a broader range of musical styles. Developing an interactive demo could also facilitate real-time composer identification, offering practical applications for music analysis

and discovery.

Conclusion and Future Work

This study demonstrated the potential of deep learning techniques in composer identification and music genre classification. Our models achieved notable results when classifying works from a limited set of renowned composers, with the optimized CNN model reaching a validation accuracy of 98% for Bach, Beethoven, Chopin, and Mozart. This high accuracy underscores our approach's effectiveness in capturing unique stylistic elements.

However, expanding the analysis to 145 composers revealed significant challenges. The decrease in validation accuracy to 42.76% for the full composer set highlights the complexity of distinguishing between a larger number of potentially similar musical styles. Our threshold analysis provided valuable insights into the relationship between data representation and model performance, suggesting that the quality and quantity of data per composer significantly impact the model's ability to learn and generalize.

The limitations encountered open several avenues for future research. For starters, an optimization is definitely needed for the all composer model, similar to the one for four composers. We performed some initial threshold analysis as well, but after trying multiple strategies, we think we might need to add additional layers within the model and also increase the backward analysis to check if the CNN picked up on the supposed patterns. Also, similar work is needed for the LSTM.

Additionally, it will be great to create a dashboard in which any MIDI file can be added, and it performs the analysis to check the probability for each composer. Also, there can be an analysis to see the closest composers and the furthest in terms of their style.

References

- Ali, Z., Benitez, A., & Xayavongsa, O. (2024). *Composer identification and music genre classification using deep learning*. GitHub.
<https://github.com/zainnobody/AAI-511-Final-Project/blob/main/Music%20Genre%20and%20Composer%20Classification%20Using%20Deep%20Learning.ipynb>
- Briot, J.-P., Hadjeres, G., & Pachet, F.-D. (2020). *Deep learning techniques for music generation*. Springer. <https://doi.org/10.1007/978-3-319-70163-9>
- Choi, K., Fazekas, G., & Sandler, M. B. (2017). *Automatic tagging using deep convolutional neural networks*. arXiv preprint arXiv:1606.00298.
<https://doi.org/10.48550/arXiv.1606.00298>
- FasterCapital. (n.d.). Frédéric Chopin. <https://fastercapital.com/keyword/frdric-chopin.html>
- Fu, Z., Lu, G., Ting, K. M., & Zhang, D. (2011). *A survey of audio-based music classification and annotation*. IEEE Transactions on Multimedia, 13(2), 303-319.
<https://doi.org/10.1109/TMM.2010.2098858>
- Hosseini, P., Qraitem, M., Symes, T., & Khoshmagham, S. (2021). *Spatiotemporal short-term traffic speed prediction using machine learning techniques with probe and weather data*. Paper presented at the Transportation Research Board 100th Annual Meeting, Washington, DC. Transportation Research Board.
<https://annualmeeting.mytrb.org/OnlineProgram/Details/15461>
- Kaggle. (n.d.). MIDI Classic Music Dataset.
<https://www.kaggle.com/datasets/blanderbuss/midi-classic-music>
- Tokozume, Y., Ushiku, Y., & Harada, T. (2018). *Learning from between-class examples for deep sound recognition*. In ICLR. <https://openreview.net/forum?id=HyRVBzap->

- Viekash, V. K., Jothi Balaji, J., & Lakshminarayanan, V. (2021). FAZSeg: A new software for quantification of the foveal avascular zone. *Clinical Ophthalmology*, 15, 4817-4827.
<https://doi.org/10.2147/OPTH.S346145>
- Vo, Q.-H., Nguyen, H.-T., Le, B., & Nguyen, M.-L. (2017). Multi-channel LSTM-CNN model for Vietnamese sentiment analysis. *Proceedings of the 2017 IEEE Ninth International Conference on Knowledge and Systems Engineering (KSE)*, 24-29.
doi:10.1109/KSE.2017.8119429. <https://github.com/ntienhuy/MultiChannel>
- Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.
https://doi.org/10.1007/978-3-319-10590-1_53