Composer Identification in Classical Music Scores

Priscilla Marquez

Department: Graduate Studies - Engineering, University of San Diego

AAI-511: Neural Networks and Deep Learning

Bilgenur Baloglu, Ph.D

August 11, 2025

Authors Note

This paper was completed as part of my MS in Applied Artificial Intelligence at the University of San Diego. I am the sole author (Priscilla Marquez). I have no conflicts of interest to disclose. For any questions, please contact priscillamarquez@sandiego.edu.

Abstract

This research project explores training two different architectures of neural models, Convolutional Neural Networks (CNN) and Recursive Neural Networks (RNN), in the task of composer prediction, where a piece of classical music is provided in MIDI format and the author of that piece is predicted. I will be comparing the technical advantages and challenges of using each type of architecture and describing the results of their evaluation.

Composer Identification in Classical Music Scores

The main goal of this project is to train two neural models using the CNN and RNN architectures on the task of music composer prediction. The two models will be trained on a classical music dataset composed of MIDI files of music written by Beethoven, Bach, Mozart and Chopin.

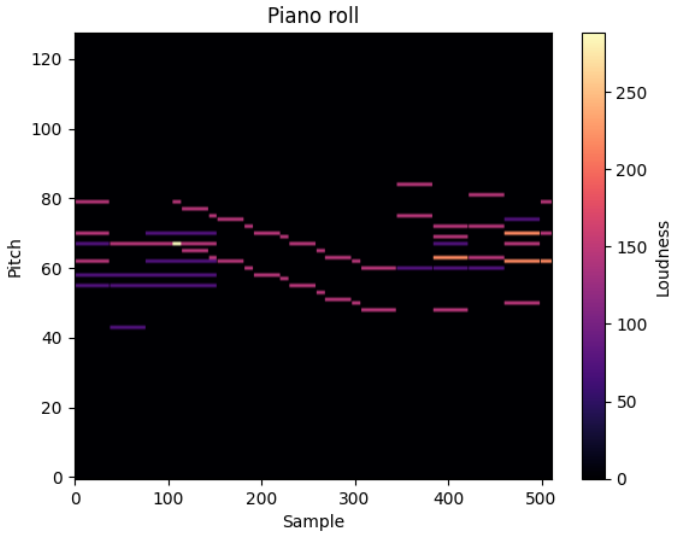
Dataset

The dataset contains multiple MIDI files of music written by a variety of classical composers (blanderbuss, 2018). I will be focusing on those compositions written by Beethoven, Bach, Mozart and Chopin. In total, I ended up with 1530 MIDI files.

The MIDI format is a digital music format that, instead of encoding soundwaves and the sound itself, describes what notes are played and how, for each of the participating instruments. To interface with this data format, I will be using the Python library “pretty_midi” (Raffel & Ellis, 2014), which allows sampling these files into “piano rolls”. This is a way of representing the music data as a time series, measuring the intensity of each of the 128 available pitches over time. This is a very flexible representation that I can be used in both CNNs and RNNs. Figure 1 is an example of how these piano rolls would look:

Figure 1.

Piano roll example



Preprocessing

I will be using a sampling rate of **64 samples per second**, as this should be enough to fully capture even the fastest notes. When loading the whole dataset into memory, I came across two issues: first, the Google Colab machine I was using did not have enough memory to hold the full dataset, and second, the number of samples available for each composer was widely different, with Chopin having the least number of samples (around 8 hours and a half of audio). I ended up settling by using a subset of **8 hours of music per composer**, for a total of 32 hours of audio, as I figured that would be enough to train the models and would remove the complexities of handling disk-backed datasets and unweighted distributions. I normalized the "loudness" value in each track so it would be normalized between 0 and 1. This preprocessing step was carried out using NumPy (Harris et al., 2020).

Then, randomly split the dataset into a "train" (80%) and "test" (20%) sets using "sklearn" (Pedregosa et al., 2011), ensuring that each would get approximately the same number of samples per composer. Because I wanted the models to focus on learning different composer styles instead of learning who wrote each piece, I made the split by whole piece instead of by samples. Otherwise, one single piece may have ended up split between the train and test dataset, and the models could have just learned to identify the music piece itself. This comes with the drawback that each composer may end up with different number of samples in the training dataset:

Table 1.

Distribution of composers in the training dataset.

Composer	Percentage
Bach	25.82%
Chopin	23.74%
Beethoven	24.79%
Mozart	25.65%

Still, the difference ended up being minimal, so I proceeded with this setup. I will be splitting each track into non-overlapping **chunks of 512 samples (8 seconds)**.

Models

I trained two main types of models, CNN and RNN, and modified them to try and improve their accuracy. Also, I trained a simple Fully-Connected Neural Network to be used as a baseline. In the worst-case scenario and because all the labels are evenly distributed, I would expect the accuracy to be around 25% for random guesses.

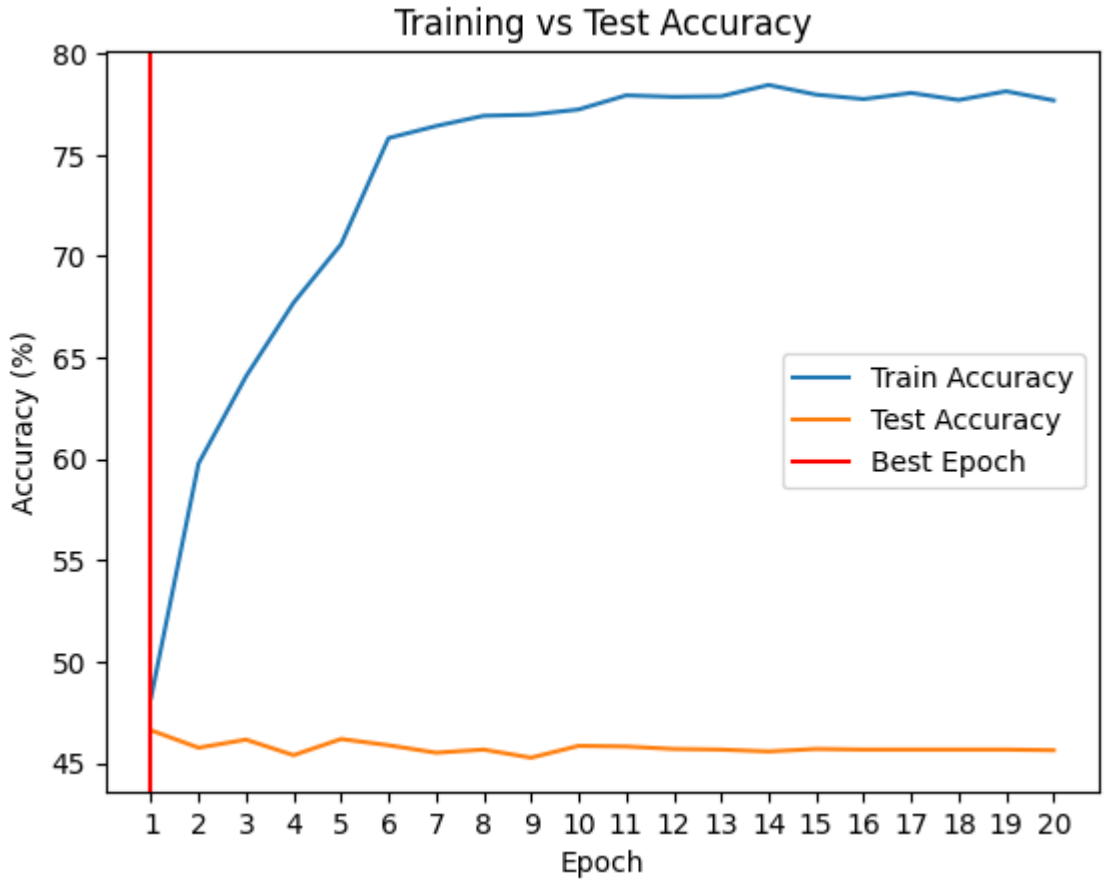
All models were trained for a total of 20 epochs using the Adam optimizer with a starting learning rate of 0.001, using the StepLR learning rate scheduler to reduce the learning rate by 0.1 every 5 epochs. The training was performed on an NVIDIA T4 GPU on Google Colab. The loss function is Cross Entropy Loss. The models and their training were implemented using PyTorch (Paszke et al., 2019).

Fully Connected Neural Network:

The reference model used only a dropout layer at 10% and a fully connected layer that mapped the (128, 512) input tensor to the 4 labels. This model achieved **46.75% accuracy**, which is still quite high relative to a random model and considering its simplicity, but started overfitting almost from the start. This can be seen on the train accuracy getting close to 80% while the test accuracy only got worse or did not change:

Figure 2.

*Fully Connected Model Training (*only took 1 minute to train.)

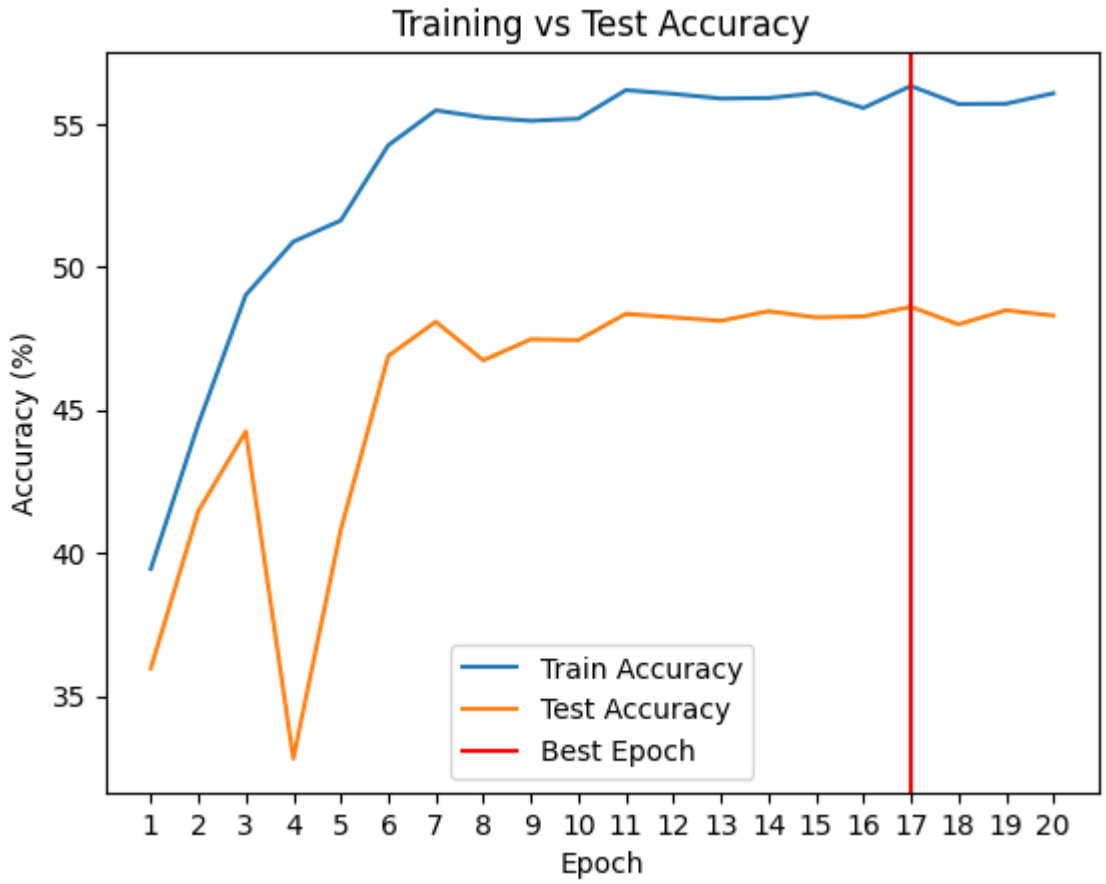


Convolutional Neural Network:

The Convolutional Neural Network (CNN) I used was made out of three convolutional blocks, each made out of a 2D convolutional layer, Batch Normalization and ReLU activation layer, followed by a 10% dropout and a fully connected layer that maps to the 4 labels (Krizhevsky et al., 2012). The input is a 2D tensor where the X axis represents the pitch, the Y axis is the time, and the magnitude is the “loudness” of each note. The number of blocks, its convolution and number of channels was different on each experiment. As a starter, I used a small (3, 7) convolution that was larger on the time axis than the pitch axis. It took 5 minutes to train:

Figure 3.

First CNN training

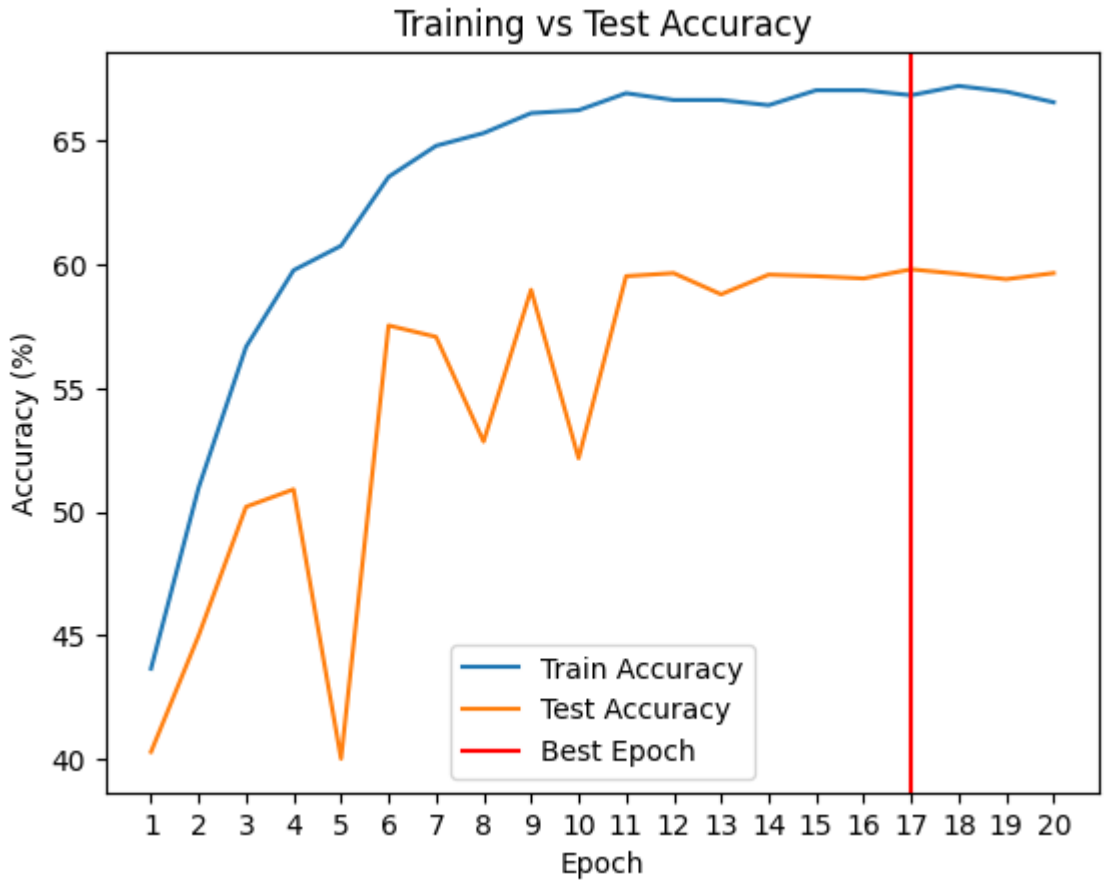


As we can see from Figure 3 this model reached a peak of 56.33% accuracy, which is better than the baseline, although it was relatively slower. There did not seem to be too much overfitting. From here, I wanted to experiment with different convolution shapes, mostly pitch versus time based. I would expect that a pitch-based convolution would perform better as it would be able to encode information about the chords, but the time axis is also important as it encoded musical patterns. The problem would be that the larger the convolution, the slower the training.

The first experiment used a larger convolution of (9, 7), which drove the training time up to 30 minutes. This achieved the best accuracy at 59.80%. I also tried convolution sizes (5, 15) and (12, 7), achieving 59.56% and 59.22% accuracy respectively, at around the same training time.

Figure 4.

Training of the best CNN model

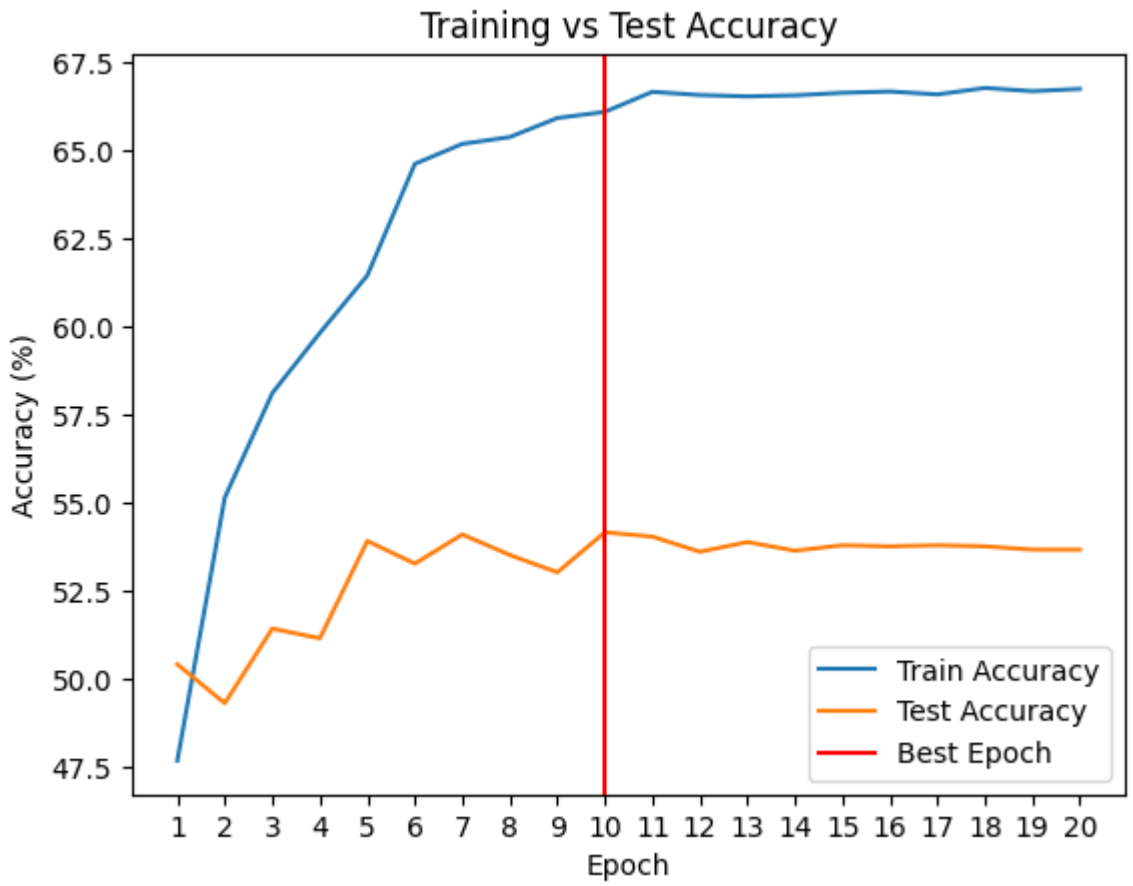


Recurrent Neural Network:

I started with a bidirectional GRU RNN with 2 layers and 128 hidden neurons, a dropout of 10% and using the mean value of the temporal layers to predict the category (Cho et al., 2014). In figure 5 this model trained faster than the CNN, taking 3 minutes, but also performed worse, achieving 54.10% accuracy, which is still better than the Fully Connected Network.

Figure 5.

RNN model results



I tried changing some parameters, like using an LSTM network and sampling the last temporal layer instead of calculating the mean, but those did not improve the model. Then, I doubled the size of the hidden layer and increased the number of layers. Both changes improved the accuracy of the model but also drove up the training time. With a hidden layer size of 256 and 3 layers, the RNN model was able to achieve a 57.13% accuracy, taking half the time of the best CNN training.

Evaluation

For the best CNN and RNN models, I calculated a performance report and plotted the confusion matrix shown in figure 6.

Convolutional Neural Network:

Based on the above report, we can see that Chopin is the composer easiest to differentiate by the model, while Mozart and Beethoven seem to get the highest mislabel rate. This is not unexpected, as Beethoven and Mozart belong to the Classical period and Mozart has been reported to be a strong influence in Beethoven (Neff, 2020). A similar style is clearly reflected in this confusion matrix. Chopin stands out as he belongs to the Romantic period, and that seems to be reflected in having a style that is easiest to differentiate. Bach seems to fall somewhere in the middle accuracy-wise, with a good balance of precision and recall. He belongs to the Baroque period, which seems to be distinguishable but not as much as Chopin.

As for the model itself, it was required to use large convolution sizes and long training times to achieve this level of accuracy.

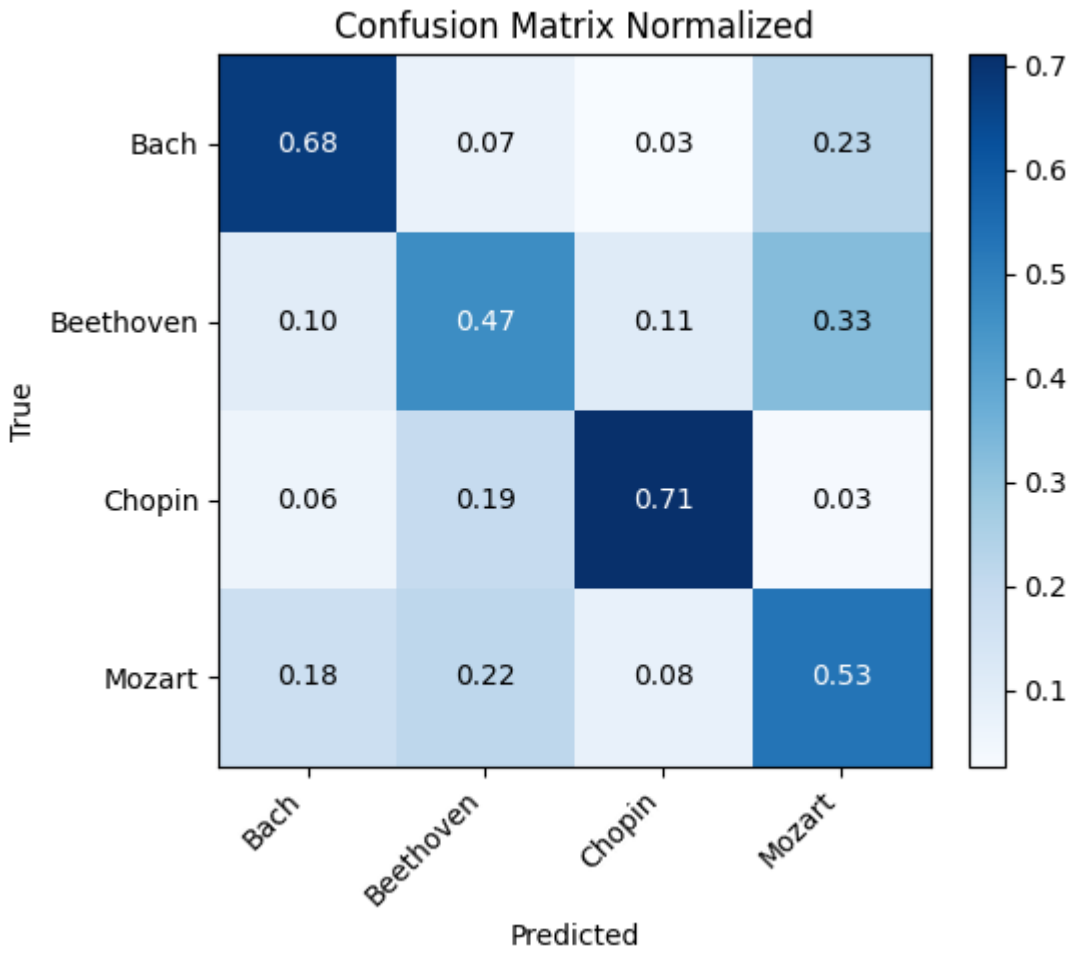
Table 2.

Evaluation of the CNN

Class	Precision	Recall	F1-Score	Support
Bach	0.637	0.677	0.656	715
Beethoven	0.496	0.465	0.480	843
Chopin	0.801	0.710	0.753	955
Mozart	0.458	0.529	0.491	746
Accuracy			0.598	3259
Macro avg	0.598	0.595	0.595	3259
Weighted avg	0.608	0.598	0.601	3259

Figure 6.

CNN confusion matrix normalized



Recurrent Neural Network:

Performance is slightly lower than the CNN baseline, with an overall accuracy of 57.1% compared to the 59.8% achieved earlier but achieved with roughly half the training time. Through different labels, behavior remains similar: Chopin is still the easiest composer to identify, with balanced precision and recall. Beethoven remains the hardest class, showing the lowest scores, likely due to strong stylistic overlap with Mozart. Mozart's recall is relatively high, but its lower precision suggests frequent mislabeling from other classes, especially Beethoven. Bach sits between these extremes, with good precision but somewhat reduced recall, indicating that while its Baroque style is still detectable, certain patterns are being misclassified as Classical.

Table 3.

Evaluation of the RNN

Class	Precision	Recall	F1-Score	Support
Bach	**0.660**	**0.585**	**0.620**	**715**
Beethoven	**0.454**	**0.445**	**0.449**	**843**
Chopin	**0.665**	**0.663**	**0.664**	**955**
Mozart	**0.514**	**0.584**	**0.547**	**746**
Accuracy			**0.571**	**3259**
Macro avg	**0.573**	**0.569**	**0.570**	**3259**
Weighted avg	**0.575**	**0.571**	**0.572**	**3259**

Conclusion

I was able to successfully train two neural models, one CNN and one RNN, in the task of composer prediction and achieve accuracy much better than random chance. Still, there is much room for improvement, but that could be accomplished with a larger dataset or more complex models. Although the CNN model was able to reach the highest accuracy, the RNN achieved similar results but with much less processing time. My prediction is that, if instead of using a temporal window of 8 seconds I used a larger one, the difference in training time would get much larger and the RNN may be able to get better accuracy. This option could be explored in future work.

Regarding the model's accuracy, it was interesting how the two composers that got the highest rate of error belong to the same musical period. I wonder if by including more composers of the same period or including more information like the instruments used in the piece, it could help the models improve their accuracy.

References

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

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. http://jmlr.org/papers/v12/pedregosa11a.html

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), Advances in Neural Information Processing Systems (Vol. 32, pp. 8024–8035). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

Raffel, C., & Ellis, D. P. W. (2014). Intuitive analysis, creation and manipulation of MIDI data with pretty_midi. In Proceedings of the 15th International Society for Music Information Retrieval Conference Late-Breaking and Demo Papers (pp. 84–93).

Neff, T. M. (2020). Enhanced program notes: Beethoven + Mozart: The art of influence. Handel and Haydn Society. Retrieved August 11, 2025, from https://handelandhaydn.org/enhanced-program-notes-beethoven-mozart-the-art-of-influence/

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks.

In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems (Vol. 25, pp. 1097–1105). Curran Associates, Inc.

Cho, K., van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. arXiv. https://arxiv.org/abs/1409.1259

blanderbuss. (2018). *MIDI Classic Music* [Data set]. Kaggle.

https://www.kaggle.com/datasets/blanderbuss/midi-classic-music

https://github.com/Prisze/AAI-511FinalProject