Team 7 Technical Report

Tyler Foreman

Shiley-Marcos School of Engineering, University of San Diego

AAI511: Final Project

Professor Ying Lin, Ph.D

August 12, 2024

Introduction

Archive7 is a data management and archiving company, specializing in providing cloud data archive and cataloging services to the media industry. A core service provided by the company is the archival and cataloging of digital music portfolios. Recently, customers have requested an ability to automatically categorize these music files based on the artist or composer, which would provide more accurate and efficient cataloging, search and retrieval of these portfolios.

The product team at Archive7 took on this request and sought to prototype the capability on a targeted set of music genres and composers. To facilitate this effort, a catalog of classical music was acquired, which contained works from four master composers: Mozart, Beethoven, Bach and Chopin.

The objective of this project was to apply deep learning techniques to create a model that could process a piece of digital music and accurately classify the composer of the work to one of the four listed above. Correctly classifying these works of music is a challenging task, as many composers are influenced by others, resulting in similarities in various characteristics of the music. Due to the difficulty level of this particular genre, a successful outcome would indicate if such an approach was technically feasible and whether it could be generalized to other genres of music.

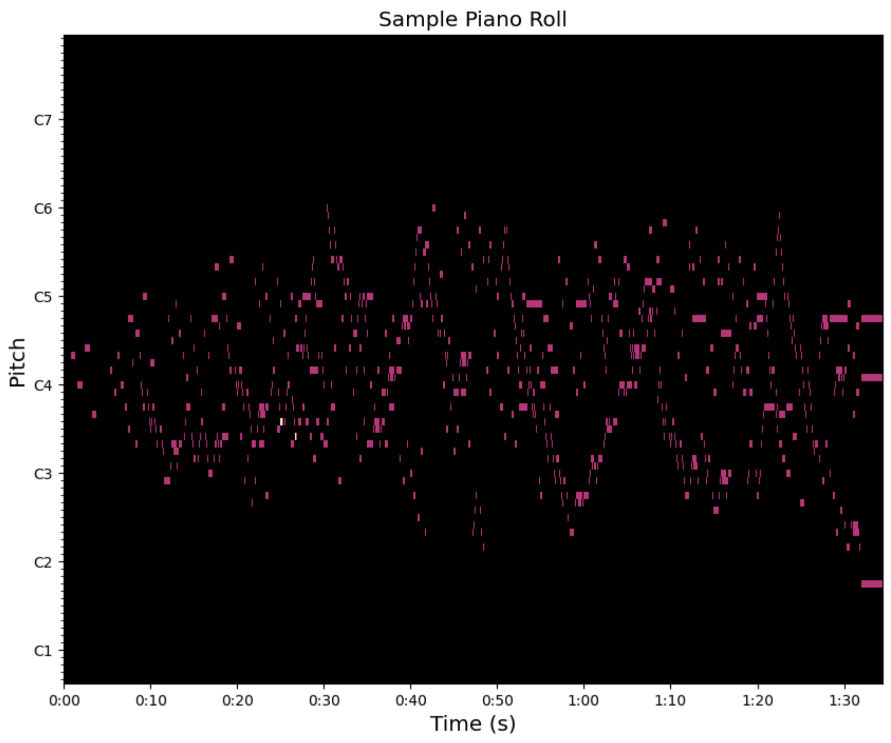
Approach

The team began the project by conducting research into established techniques for analyzing digital music files in order to formulate a technical approach to this problem. Since the objective of this project is ultimately to classify these files by composer, the research focused initially on techniques to extract meaningful features from these digital music files that would provide predictive power to our model. To make the approach as generalizable as possible for future use cases, the team specifically focused on fundamental features that would be readily available across genres and would not be specific to any particular music file format.

The team's research uncovered some notable approaches that extract these fundamental characteristics. One such approach was the extraction of the "piano roll" from the music file (Ramsey et al., 2018). The piano roll produces a time-series matrix representation of the music. At each time step, a vector is provided (matrix column), with each dimension of the vector corresponding to a different pitch. The value at each dimension of the vector represents an intensity of the pitch at the corresponding time step (*Pretty_Midi — Pretty_Midi 0.2.10 Documentation*, n.d.). A non-zero value indicates that the note is being played and at what intensity, while a value of zero indicates that the note is not active at the time step.

Figure 1

Visualization of a piano roll matrix (from Bach music sample)



The team determined that the piano roll was an ideal feature set to support the classification task at hand. As it deals with fundamental characteristics of the music (pitch intensity at a given time), it was one that could generalize well across genres.

With the high level technique for feature selection defined, the team was able to frame the deep learning approach as a time-series classification task. That is, the model would need to process, understand and contextualize the time-series piano roll sequences provided and accurately classify the composer based on these sequences. To facilitate this, two model architectures were selected that are well suited to time-series analysis tasks: Long Short-term Memory (LSTM) and Convolutional Neural Network (CNN) (Petit, 2022). These models would be configured to process the piano roll sequences and run the result through a classification head that would output the composer prediction. Multiple experiments were conducted to select the optimal model architecture, as will be explained in more detail below.

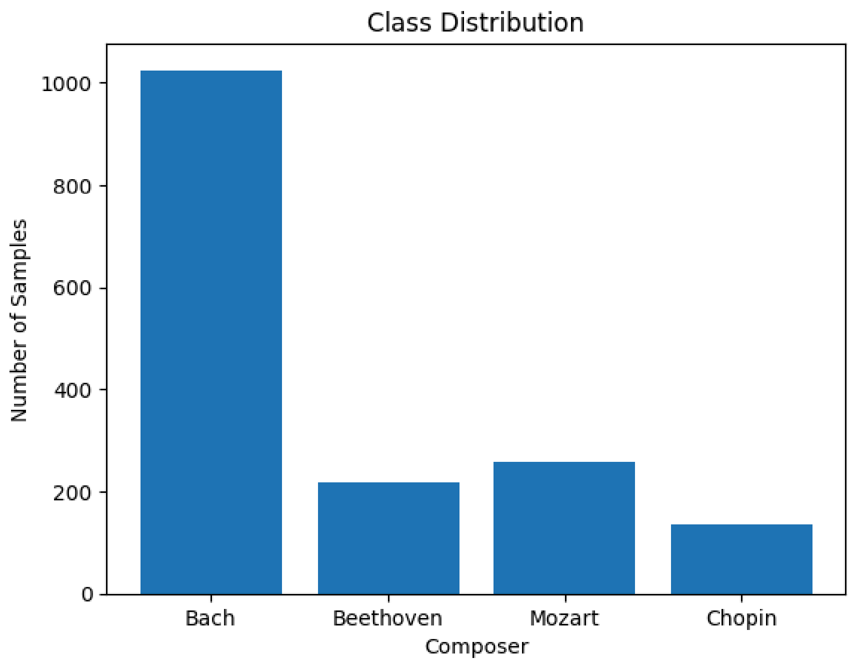
Exploratory Data Analysis

The dataset was provided as a set of Musical Instrument Digital Interface (MIDI) files organized into folders based on the composer (*Midi_Classic_Music*, 2019). The full dataset contained music from many composers outside of our four targets. To streamline the data analysis, the music files from just the four target composers were separated from the full dataset and a routine was executed on each composer directory to flatten the file structure so as to eliminate any sub-directories present. This resulted in a flat hierarchy with only the relevant music files under each composer directory.

Initial interrogation showed that there were a total of 1,637 music files across the four composer classes. Further analysis of the breakdown of these music files by the composer revealed a significant class imbalance. Most notably, there was a significantly higher number of music files for Bach (1,025) when compared to the other three. This indicated work would need to be done downstream to ensure a balanced representation of classes when training our models.

Figure 1

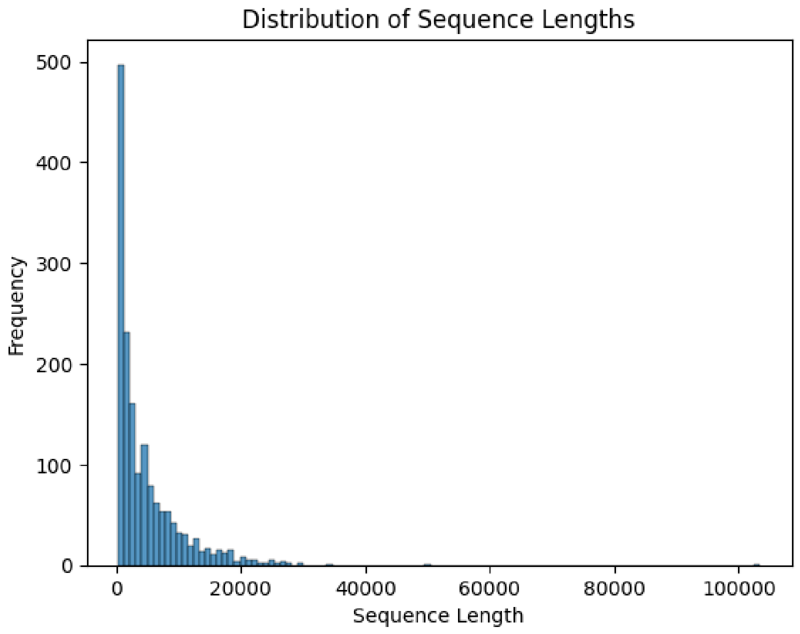
Illustration of number of samples (music files) for each composer class



The composer files were then processed using the pretty-midi library (*Pretty_Midi — Pretty_Midi 0.2.10 Documentation*, n.d.) to extract and analyze the sequences of piano rolls. To facilitate this, a sample frequency of 20 was selected, corresponding to 20 samples per second. As this was a time-series modeling task, understanding the distribution of sequence lengths across the corpus of music files would help determine an appropriate window size to use as input to the deep learning models. The sequences ranged broadly in length from a minimum of 350 time steps and 103,362 time steps, with a mean of 4,838 and standard deviation of 5,900. This wide range of sequence lengths and large variation across the distribution indicates that some preprocessing of these piano rolls will be necessary in order to normalize the sequences into a common length.

Figure 2

Distribution of piano roll sequence lengths across composers



Data Preprocessing and Feature Extraction

The exploratory data analysis revealed that some preprocessing of the datasets was required in order to prepare the dataset and extract useful features for subsequent states of the pipeline. Namely, the class imbalance issue needed to be addressed and the piano roll sequence lengths needed to be normalized to a standard length suitable for input into the models.

Since the sequences needed to be sampled and transformed into standard lengths, this task was taken on first, as any class balancing would need to be performed on these processed sequences. Several techniques were employed to generate these sequences, which ultimately led to experimentation with the models in subsequent stages. One parameter that contributed to this experimentation was determining the optimal sequence length to be generated. The sequence length can be correlated to a time-based “clip” of the audio track. Experiments were conducted with long sequence lengths (corresponding to 1-2 minutes clips) and shorter lengths (10-20s).

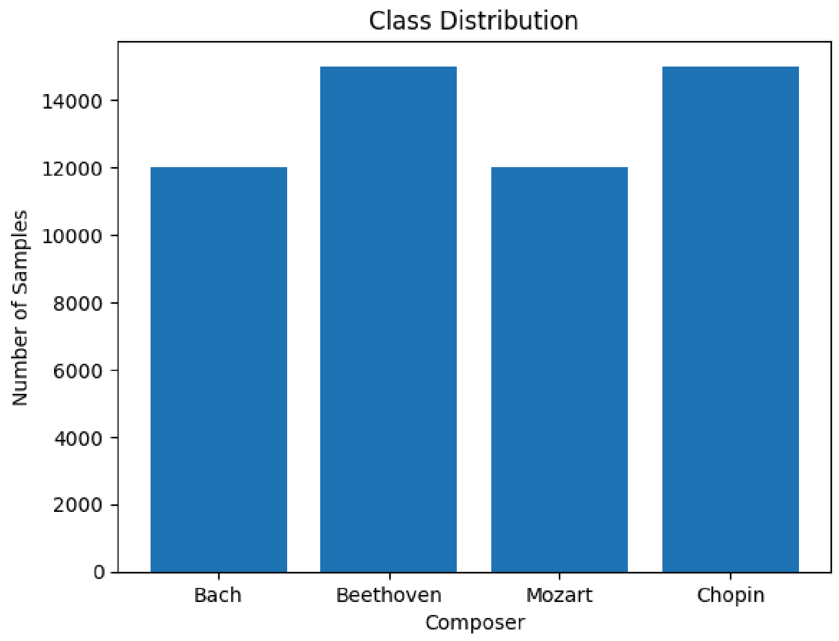
The first feature extraction technique used to generate sequences was to directly break up each music track based on the sequence length, padding any incomplete sequences at the end of the track with zeros. For example, if a track was 60 seconds long total and our sequence length was 10 seconds, this approach would produce 6 sequences, each consisting of 10s worth of timesteps. This approach worked reasonably well. However, this approach had limitations. For one, it produced a relatively low number of sequences, which did not help in any way with the class imbalance. Second, downstream experimentation with these sequences on baseline models illustrated performance that was not optimal, as there were relatively few samples per composer available to train the model with.

A second feature extraction approach was adopted instead that added a “strided sampling” technique to generate the sequences. Instead of just breaking up each track into even parts, the strided technique would apply a moving kernel to each sequence. At each stride, a sequence would be sampled with the proper sequence length, then the kernel would be shifted by the stride amount, where another sample would be taken. The stride number was a hyperparameter that through experimentation was set to an optimal number of 10.

The strided sampling approach for feature extraction solved several challenges. First, it generated sequences that provided the model more context between sequences, as there were overlapping time steps between them. This seemed to improve overall model performance in baseline experiments. Second, it resulted in a much larger number of overall sequences. In fact, it generated more samples than were manageable by the compute environment. However, this helped solve the class imbalance problem, as the sequences were randomly sampled to produce a fixed number of samples for each class. Two composer classes - Beethoven and Chopin - were slightly oversampled compared to the other two composers. The reasoning for this will be explained further in the results section.

Figure 3

Balanced classes after feature selection and strided sampling of sequences



The resulting samples were then combined, with appropriate labels, into a consolidated dataset. The labels were one-hot encoded into 4 dimensional vectors. The combined dataset was then split into train/test/validation datasets at an 80/10/10 split.

Model Definition, Training and Performance Evaluation

As mentioned previously, two fundamental model architectures - LSTM and CNN - were used as the foundation for several experiments to determine the optimal model architecture and configuration for this task. For each model architecture, a baseline model was defined for two reasons. First, the baseline model enabled quick experimentation during data preparation to validate that the dataset was generated in a format suitable for the model to interpret. Second, these simple models set a baseline performance level from which to create improvements in subsequent experiments. Full model configurations used are outlined in Table 1.

Table 1

Summary of model architectures and configurations used in experimentation

Name	Norm. Layer	# Hidden Layers	# Dense Layers	# Units/ Hidden Layer	# Units/ Dense Layer	Dropout
LSTM Baseline	No	1	1	256	64	0.2
LSTM Improved-1	Yes	2	2	128->32	32	0.3
LSTM Improved-2	Yes	4	2	128->32	64	0.3
CNN Baseline	No	3	2	256->64	64/32	0
CNN Improved-1	Yes	5	3	512->32	64/32	0.2
CNN Improved-2	Yes	6	2	256->64	64	0.3

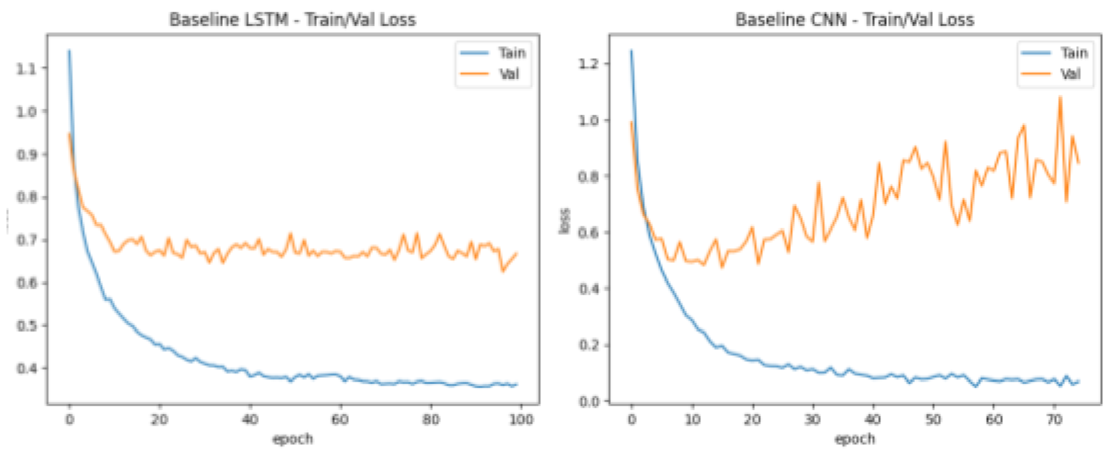
The models were all configured to use CategoricalCrossEntropy for the loss function and Adam for the optimizer. Note that the improved models (non-baseline) all included a normalization layer on the input to normalize the data. For the baseline models, a pre-normalized version of the dataset was used. The models were all evaluated across four performance metrics: categorical accuracy, precision, recall and F1 score. The model checkpoints were captured based on the maximum categorical accuracy score on the validation dataset.

Training was conducted for all models using 75-100 epochs, with a batch size of 32. Some experimentation was done with different learning rates but no real impact on performance was observed so the learning rate was kept at 0.001.

The baseline models for both architectures exhibited signs of overfitting, though achieved a moderately good categorical accuracy score of 0.774 (LSTM) and 0.882 (CNN) after performing some tweaks to the class balances in the dataset (as mentioned previously). Early experiments indicated that the models were getting confused at a higher rate between Beethoven and Chopin. Boosting the samples from these two composers helped improve this problem without having any detrimental effects on the other two classes.

Figure 4

Evidence of overfitting in both LSTM and CNN baseline models loss curves



To address the overfitting issue, the improved LSTM and CNN models were defined with varying degrees of dropout. To improve performance (measured by categorical accuracy on validation data set), the definitions of these models included improvements such as more hidden layers and more units per layer. The impacts of these improvements were then measured in subsequent training runs of each model.

The results of these experiments are captured in Table 1 below. For the LSTM models, we saw an improvement with each experiment, with the best LSTM model being the *LSTM Improved-2*, which had more hidden layers and more units per layer than the other LSTM models. This model also incorporated a higher level of dropout, which seemed to get the overfitting problem under control.

Table 2

Summary of best performance metrics achieved by each model against validation dataset

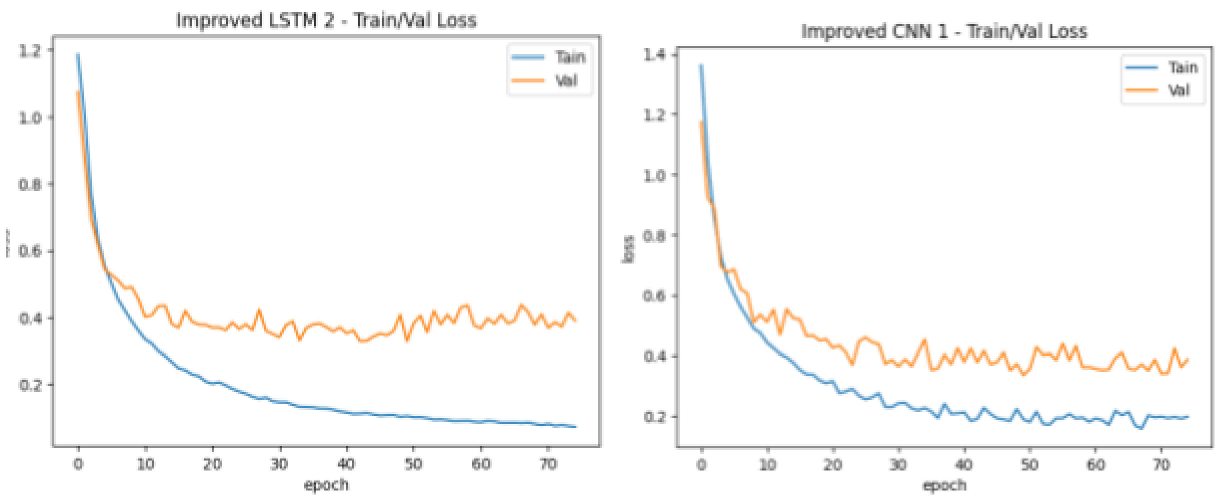
Model	Cat. Accuracy	Precision	Recall	F1 Score
LSTM Baseline	0.774	0.792	0.759	0.772
LSTM Improved-1	0.887	0.892	0.885	0.887
LSTM Improved-2	0.909	0.911	0.909	0.906
CNN Baseline	0.884	0.890	0.879	0.884
CNN Improved-1	**0.911**	**0.922**	**0.897**	**0.908**
CNN Improved-2	0.848	0.902	0.749	0.849

For the CNN models, the results were mixed. The best performing CNN was *CNN Improved-1*, achieving the highest overall categorical accuracy score of 0.911. This CNN did not have the highest number of hidden (convolutional) layers, but it did have a higher number of units in the largest convolutional layer, cascading down from 512 to 32 across 5 layers. Interestingly, the lowest performing model was the CNN Improved-2 model, which incorporated the most hidden (convolutional) layers but a reduced number of units per layer, starting at 256 in the largest convolutional layer cascading down to 64.

Evaluation of the performance metrics reveals a very close level of performance between the best LSTM (*LSTM Improved-2*) and the best CNN (*CNN Improved-1*), with the CNN edging out the LSTM slightly in most of the metrics. The margins of this difference are so narrow that it would be difficult to choose one over the other based solely on this criteria. However, it was also observed during training that the CNN model seemed to exhibit less overfitting behavior, as evidenced by the train/val loss curves shown in Figure 5. This would indicate that the CNN is likely to generalize better in the real-world, making it a more suitable choice.

Figure 5

Train/Val loss curves of best LSTM and best CNN models, with CNN exhibiting les overfitting



The best CNN was thus selected as the model to evaluate against the hold out test dataset. This was done by running the dataset through the model and evaluating performance against the ground truth labels. The model maintained slightly improved performance on the test data as it did against the validation data, bolstering confidence that the model does in fact generalize well.

Table 3

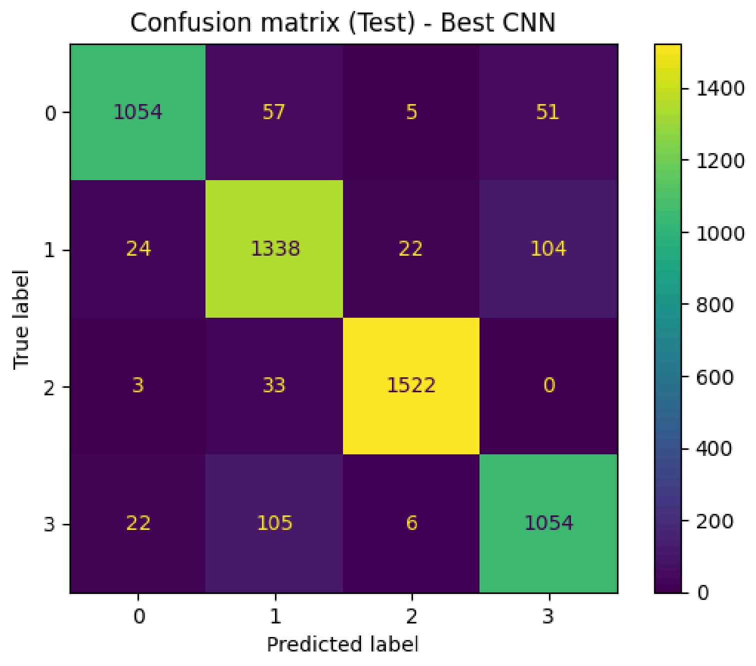
Summary of performance metrics achieved by best CNN model against test dataset

Model	Cat. Accuracy	Precision	Recall	F1 Score
CNN Improved-1	**0.920**	**0.931**	**0.909**	**0.918**

A confusion matrix was also generated (as it was for all previous experiments) to gain insight as to where the model was most commonly making mistakes. As observed during the experiments, the model makes the most mistakes distinguishing between class 1 (Beethoven) and class 3 (Chopin). There appears to be approximately an equal number of misclassifications when the ground truth is either of these classes. However, the results were significantly improved as a result of the dataset balancing performed and described above.

Figure 6

Confusion matrix of best CNN evaluated against the test dataset



Conclusions and Recommendations

The results of this project present encouraging evidence that the approach taken to process digital music files and auto-categorize them is technically sound. The performance of the best model is approaching levels of what would be expected of a production-quality model, with a categorical accuracy score of 0.92 on the test dataset. The feature engineering approach of leveraging the “piano roll” matrix to represent the work of music to the model appears to have been a good design choice, though it was significantly improved when the novel “strided sampling” technique was applied to generate normalized sequences of piano rolls and generate a large volume of sequences. Though this approach resulted in slightly increased computational time and larger overall sample representation for a single piece of music, these were deemed acceptable tradeoffs.

Model architecture choice between LSTM and CNN did not seem to result in a drastic difference in performance, though the CNN’s ability to generalize did give it an edge. Since the ultimate objective of this project is to expand to include a broader range of music genres and artists, this characteristic is important and thus part of the reason for the team's recommendation to adopt the CNN model architecture.

The team would additionally recommend continued exploration of hyperparameter tuning and experimentation with different CNN model configurations (number of layers, number of units) as a method to potentially improve the model further. There was evidence in the CNN experiments that a larger number of units per layer resulted in the best overall performance, so this would be a specific area of experimentation to explore. Other dataset preparation parameters could also be tuned further and experimented with, including the piano roll sample size, the sequence length and the strided sampling window size.

Finally, there is further opportunity to enhance the feature engineering beyond using only the piano roll matrix. Adding additional features available, such as tempo, duration and velocity would be additional areas for experimentation and may yield further improvement in model performance.

References

Ramsey, C., Huang, C., & Costa, D. (2018). The classification of musical scores by composer. *CS230*.

pretty_midi — pretty_midi 0.2.10 documentation. (n.d.). https://craffel.github.io/pretty-midi/

Petit, P. (2022, January 6). PYZZICATO — Piano Instrumental Music Generation using Deep Neural Networks | by Philippe Petit, Gatien Vassor, Emna Bairam and Wladimir Raymond | Towards Data Science. *Medium*. https://towardsdatascience.com/pyzzicato-piano-instrumental-music-generation-using-deep-neural-networks-ed9e89320bf6

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

midi_classic_music. (2019, January 28). Kaggle. https://www.kaggle.com/datasets/blanderbuss/midi-classic-music/data

Appendix A: GitHub Repository

GitHub: https://github.com/t4ai/music-composer-classification