Andres Avila
CST 4702
FALL 24

Abstract

In this project, we aim to classify MIDI file composers using sequence attention-based models. We will use a dataset of six composers: Alkan, Schubert, Handel, Mozart, Scarlatti, and Victoria. This research aims to create a machine learning model that accurately predicts the composer of a drawn MIDI file. The rationale behind this specific selection of composers is their relatively uniform dataset sizes, making them the perfect candidates for this whole process. The data was preprocessed using fixed-window sampling and subsequently fed to a bi-directional LSTM model combined with an attention mechanism to learn temporal and contextual dependencies in the musical input.

Introduction

This project explores composer classification using a method I selected of sequence-based attention models on MIDI inputs. Classical music is a fascinating example due to the unique style of each composer, which informs their rhythmic and harmonic components, making the challenge difficult. MIDI data is a structured and digital representation of music, from which it allows us to extract and transform into vector representations that help us analyze musical features (for instance, detailed features like pitch, note lengths, tempo, and dynamics).

In this final project, our goal is to create a machine learning model that correctly predicts the composer of the input MIDI file with a high accuracy score. This project is centered around six composers: Alkan, Schubert, Handel, Mozart, Scarlatti, and Victoria, selected for their balanced representation in the dataset. My approach applies a sampling technique based on a fixed window to convert the actual note system into smaller data sets to go through the model in a reasonable amount of time. We use a bidirectional LSTM model with an attention mechanism to encode both time-dependent behaviors like notes, octave, and note length. That is the function I chose to use.

There are some unique challenges for machine learning models when classifying classical music by composer, particularly when working with MIDI data. The MIDI files are relatively correct most of the time but can contain errors. Some files won't have notes at all; others might be corrupted or lack important metadata such as tempo changes or time signatures. This makes reading and extracting useful information difficult, I had to cater to every edge case imaginable to prevent the model from exploding during training.

Data variability introduced yet another challenge. Classical compositions can vary so much in structure, instruments, and length that MIDI files can contain. Since processing long sequences is computationally expensive, I had to use windowing technique, and since the notes, note length, and octaves were of the same size, I didn't have to worry about padding to bring consistency into the data, which wasn't easy to do. Additionally, the underlying hardware required to efficiently process this vast amount of data was not trivial. If I wanted to avoid running out of memory using test data, I had to process the dataset in parallel using tools like `ipyparallel`.

The dataset imbalance was another huge challenge. Because different composers had different numbers of MIDI files, I had to sample and preprocess the data critically so that each composer would be equally represented. This was important to avoid the model being biased towards the composers for whom there is the most data.

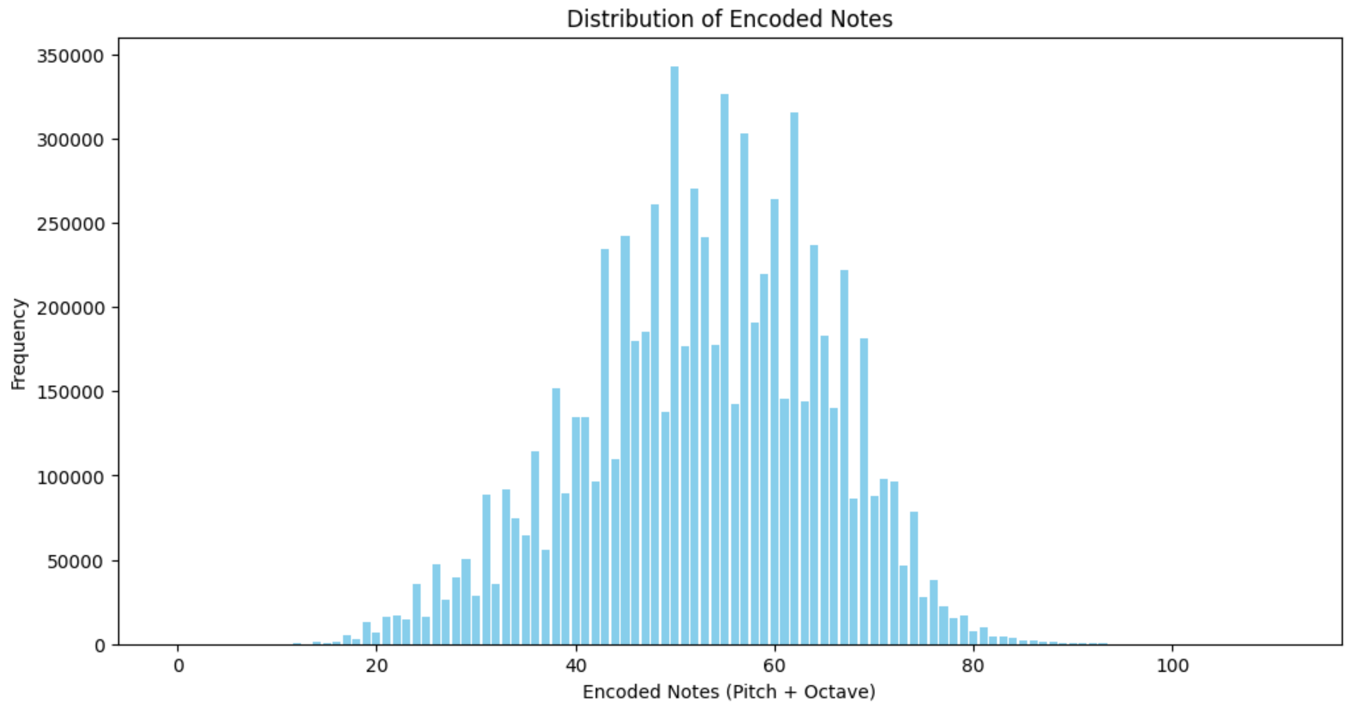
It had also not been easy to figure out how to capture the essence of a composer's musical style. Classical music is extremely complex, having deep patterns and dependencies across notes, rhythms, and harmonic structures. We needed models such as LSTMs and attention mechanisms to capture these details, and they also had their challenges too when it came to tuning and implementation. In short, doing this project required a lot of both technical expertise, as well as trial and error in order to address hardware constraints, anomalous data, and classical music's unique structure. It's certainly not an easy job....

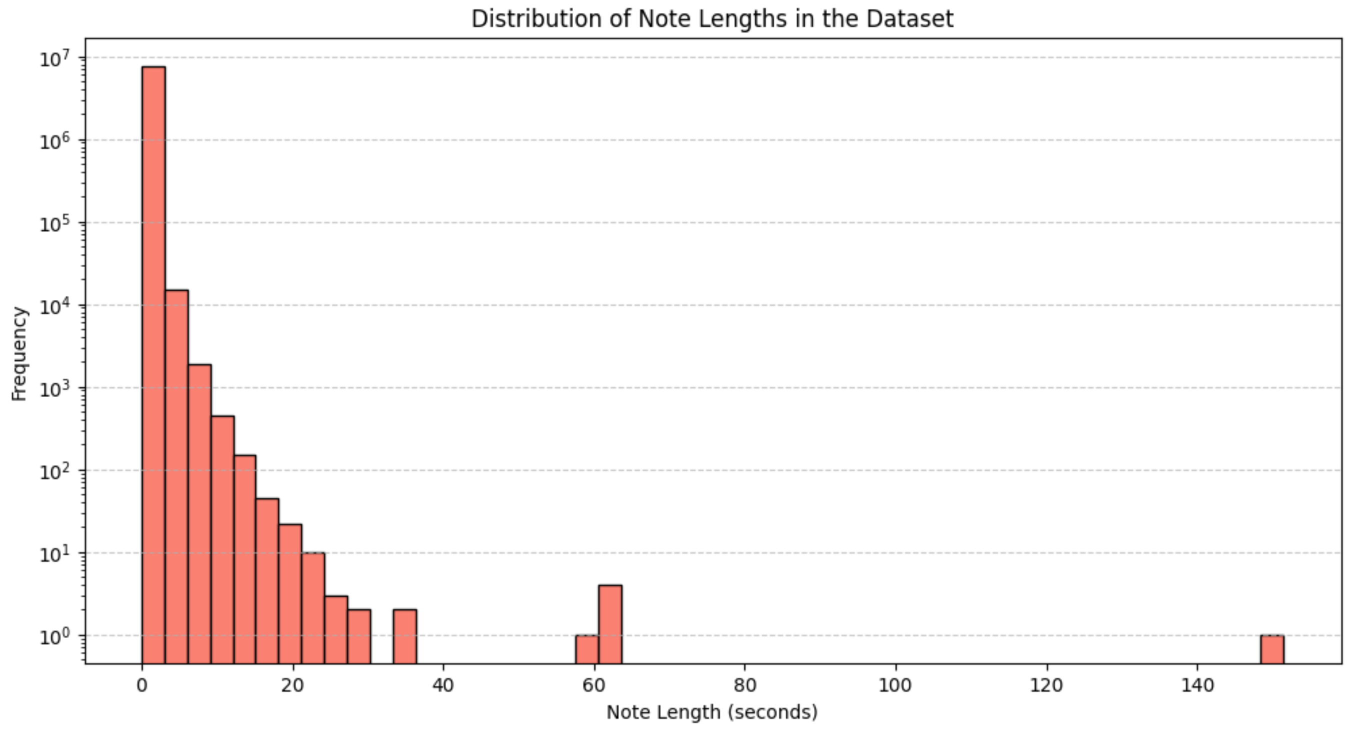
Dataset

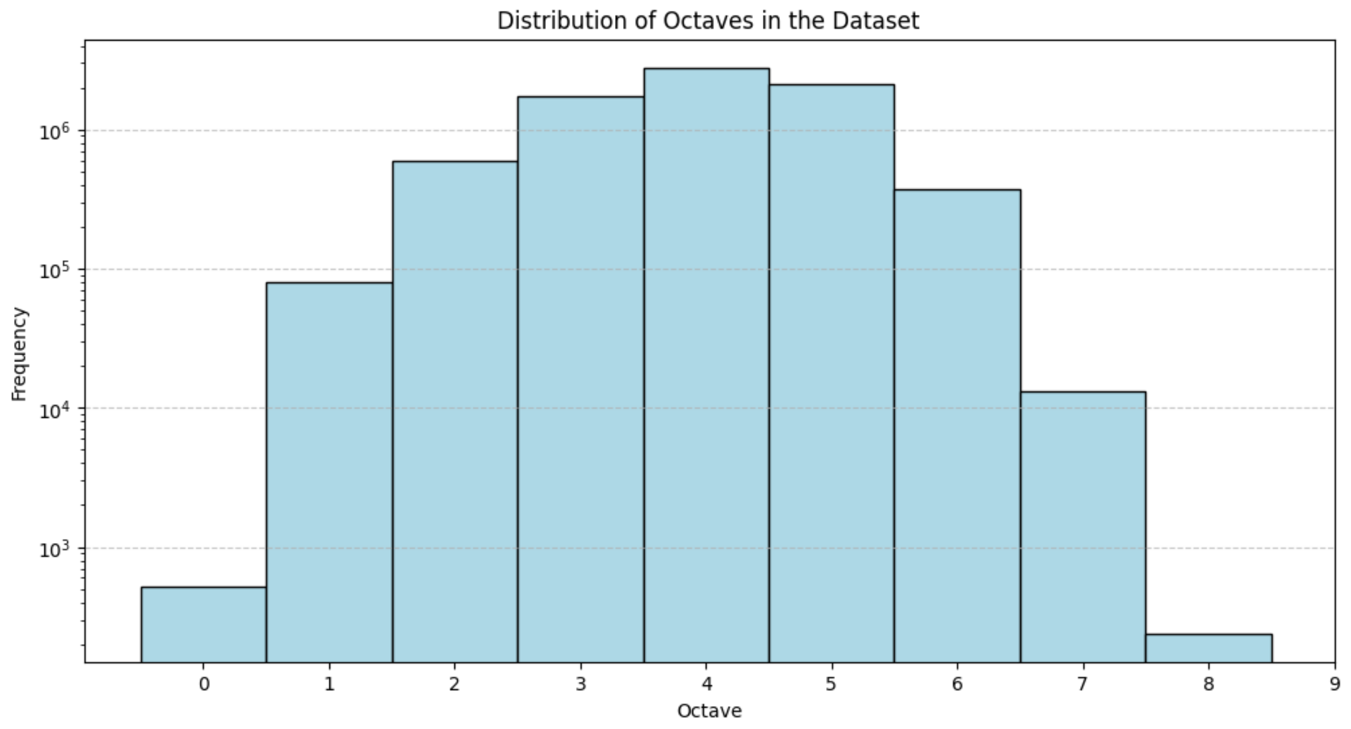
The MIDI dataset was specifically chosen to contain recordings of compositions from six classical composers Alkan, Handel, Mozart, Scarlatti, Schubert, and Victoria, going by their influence in classical music. Excluding four erroneous and 26 blank files, we were left with 2,564 valid MIDI files. These characteristics reflect musical properties, such as the note pitch (expressed in numbers), melody length (expressed in milliseconds), note octaves, key, tempo (as BPM), and musical meters (the time signature). Due to the sequential nature of the data, we employed a sliding window approach, we used a window size of 600 notes, and a step size of 15 notes, making sure to use padding to obtain a consistent input dimension. Note lengths and tempos were normalized, and a unique label was given to each composer. Data was divided into training (80%) and testing (20%).

More general analysis through the dataset helped investigate how different musical attributes were distributed to support the selection of features to investigate or the preprocessing steps to take. The distribution of note counts found across MIDI files exhibited a heavy tail, where the majority of pieces were of low density, but a few outliers greatly went above the average. It highlighted the importance of implementing well-considered windowing parameter tuning. Octave distribution showed a clear peak among mid-range values (octave 4), which aligns with the also identified typical range of classical compositions and indicated that extreme low and high octaves are less frequent. The distribution of note lengths was equally right-shifted (skewed to the right), which showed preference for short durations with occasional spikes at longer (sustained) notes, which can be expected in classical music. Finally, a visual analysis of the distribution of the encoded notes gave a normal distribution shape that reflects the dominance of certain notes in the dataset..

Visualization of the data:







Mathematics

$$\alpha_{t,i} = \text{softmax}(\frac{h_t^T h_i}{\sqrt{n}})$$

The key idea is to use the dot product to calculate the attention score, and then scale it and apply a softmax function to obtain the attention weights. Through this mechanism, in my project, I transformed the model so that it paid attention to critical musical features like notes, octaves, and lengths, which improved its classification capability of the MIDI sequence's composer. The attention weights alpha indicate how much importance the model assigns to each encoded state h while generating the output at time t. The final context vector ct is computed as a weighted sum of these encoded states:

$$ct = \sum_i \alpha_{t,i} h_i$$

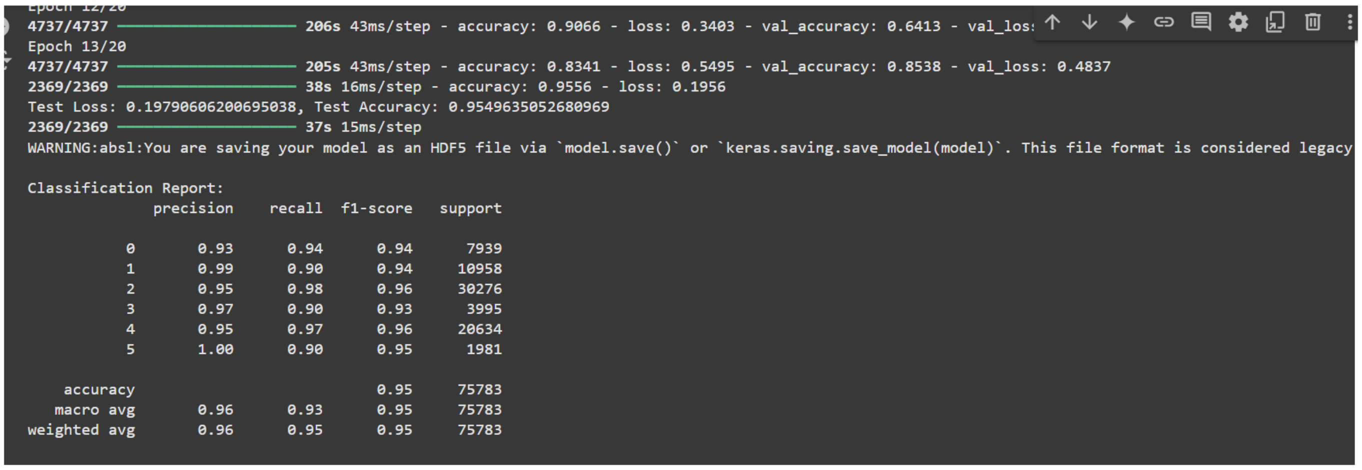
This formula is for attention comes from the **scaled dot-product attention**, where each time step t_t is assigned a score by computing the dot-product between the current hidden state h_t and a series of "cached" states h_i. That's the way I like to think about it.

An example could be a particular "note" set or combination of octaves and note lengths could be highly distinguishable of a specific composer. This would allow the model to put extra emphasis on these patterns to improve classification performance.

The combination of LSTMs and attention helps my model capture both temporal dependencies in the data and focus on the relevant parts of the input sequence to be the better-suited model for the task of composer classification.

Final Results:

This project attempted to classify classical composers in the form of MIDI data, which required the extraction of informative features notes, note durations, octaves were extracted using the `music21` library. These features represented fundamental musical components, that can be indicative for other characteristics of music styles like melody, rhythm, and harmonic organization, which were critical for differentiating composers. Numerous deep learning models, including LSTM, Bidirectional LSTM, and attention mechanisms, were tested for their performance. The results showed that the simpler regularized LSTM model reached an accuracy of 74.7%, while the more computationally expensive Bidirectional LSTM with Attention achieved ~95%.



```
Epoch 12/20
4737/4737 ━━━━━━━━━━━━━━━━━━━━ 206s 43ms/step - accuracy: 0.9066 - loss: 0.3403 - val_accuracy: 0.6413 - val_los
Epoch 13/20
4737/4737 ━━━━━━━━━━━━━━━━━━━━ 205s 43ms/step - accuracy: 0.8341 - loss: 0.5495 - val_accuracy: 0.8538 - val_loss: 0.4837
2369/2369 ━━━━━━━━━━━━━━━━━━━━ 38s 16ms/step - accuracy: 0.9556 - loss: 0.1956
Test Loss: 0.19790606200695038, Test Accuracy: 0.9549635052680969
2369/2369 ━━━━━━━━━━━━━━━━━━━━ 37s 15ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.94      7939
           1       0.99      0.90      0.94     10958
           2       0.95      0.98      0.96     30276
           3       0.97      0.90      0.93      3995
           4       0.95      0.97      0.96     20634
           5       1.00      0.90      0.95      1981

    accuracy                           0.95     75783
   macro avg       0.96      0.93      0.95     75783
weighted avg       0.96      0.95      0.95     75783
```

References

1. Nama, A. (n.d.). *Understanding Bidirectional LSTM for Sequential Data Processing*. Retrieved from Medium.
2. Analytics India Magazine. (n.d.). *Bidirectional LSTM with Python Codes: Implementation for Padding Sequences*. Retrieved from Analytics India Magazine.
3. Vaswani, A., et al. (2017). *Attention Is All You Need*. Retrieved from arXiv.
4. Gentaiscool. (n.d.). *LSTM-Attention Implementation*. Retrieved from GitHub Repository.