CS230

The Classification of Musical Scores by Composer

Christina Ramsey
Department of Computer Science
Stanford University
cmramsey@stanford.edu

Chenduo Huang
Department of Computer Science
Stanford University
cdhuang@stanford.edu

Daniel Costa
Department of Computer Science
Stanford University
dcosta2@stanford.edu

Abstract

The aim of this project was, given a musical score, to accurately predict which composer wrote it. We believed that this project would be an interesting experiment in audio classification and would potentially demonstrate where composers were influenced by other composers; at the same time, we believe that this project's importance lies also in its easy generalizability to other musical recognition tasks.

Here, we built an LSTM and a CNN that determine who composed a piece. The input is a section of a score, extracted from a midi file, while the output is a specific composer.

Based on our results, the CNN outperformed both the LSTM and the baseline. The LSTM's overall weak performance is likely a result of an issue with the processing of input data.

1 Introduction

We recognized early on that it was an ambitious task to train our model to recognize a specific person from a wide range of genres, musical styles, and composers; for this reason, we decided to deal only with a preselected list of classical composer in order to limit the scope of the project. That being said, we accumulated enough data to expand into additional eras, genres, or composers if desired.

We foresee that this project will be an interesting experiment in audio classification and will potentially demonstrate where composers were influenced by other composers. However, as stated, we were also excited at the prospect of its easy generalizability to other musical recognition tasks. As our project has trained neural networks to recognize and respond to specific features of classical scores, it could thus be frozen and used as the first half of an expanded architecture that could classify a wider list of composers or genres, identify specific songs, or generate new midi scores, for example.

Input and Output

Originally, we had planned to use data from `http://kern.humdrum.org`, which includes .krn file encodings of many compositions we will use as training and test data. We expected that we would take numeric representations of this data as input into our model; however, due to difficulties in parsing and interpreting the .krn format, we eventually decided to pursue a different method.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

Instead, we decided to use midi file representations. Unlike MP3 files, these midi representations don't contain actual audio data; they instead contain musical information that encodes how the sound it represents should be produced. Midi files represent the score of a piece as that piece was specifically written; unlike audio files, which feature a performance of the piece and potentially include mistakes or the performer's interpretations, midi files objectively record instructions for how a piece should be played as the composer envisioned. This information is encoded as a series of numbers, as shown in the Dataset section below. However, python packages that allow you to extract information about certain features of midi files, including instruments used, the piano roll, key and time signatures, and etc, exist to help us interpret these files in our project.

Each input to our model is a midi file. The output was a number corresponding to a particular composer who we predict composed the piece. Information about the specific composers and the preprocessing for these midi files, as well as a specific example of a midi file representation, is included in our Dataset section below.

2 Related work

We were influenced strongly by a paper by Lebar et al [1] that also explored identifying composers. However, the paper itself was written in 2012 and the methods that it used do not reflect current deep learning implementations. We believe that applying more modern methods should significantly improve the results reported in the paper. In addition, the previous paper only considered pitches and their duration as input into the models. We believed that we could find more predictive features or experiment with sequential deep learning models to improve the classification accuracy they stated. Our exploration was further enhanced by the work of Sander Shi [2], who implemented supervised learning using both a Multi-layer Perceptron model and a Support Vector Classifier model for the task of classifying midi files. While he too dealt with the identification and extraction of important features, he achieved success with his approach. His work heavily influenced the approach to our baseline.

We also took time to understand the research Cataltepe et al [3], who experimented with the use of midi files and their audio features for the task of midi classification. They worked specifically with a representation of the distance between midi files, using the Normalized Compression Distance to achieve a similarity score between two given pieces. This gave us insight into information about the structure and encoding of midi files, but did not inform our future implementation.

We looked instead to the project done by Huang and Wu [4], who used an RNN to generate music with both harmony and melody. Unlike the first few papers, this relied primarily on deep learning methods instead of solely feature extraction. However, their project, unlike ours, was generative in nature. Likewise, work done by Kalingeri et al [5] to generate music, was based off of midi files. It too used no specific information about musical structure, including anything about notes or cords, in their learning process. This was an approach that we wanted to achieve, although we aimed for the goal of classification rather than generation. Finally, Lee et al [6] inspired us to implement a CNN imitating the VGG net. Structures with similar to the VGG net had been used extensively for music audio tagging. While the input and output of this task is slightly different from our own, we were convinced that there were many things we could borrow from the paper to train a successful CNN.

3 Dataset

We gathered 443 classical songs in midi format from the following 9 composers: J.S. Bach, Byrd, Chopin, Händel, Bartók, Hummel, Mendelssohn, Mozart, and Schumann, with about 50 pieces from each composer. The midi files themselves were collected from http://www.midiworld.com/classic.htm, which is massive collection midi representations of songs from different genres, eras, and composers. We decided on these specific composers because they were all prolific classical, baroque, and romantic composers. Not only was it easy to locate ample data for our model, but these composers are widely regarded and recognized: classification error by experts and musicians would be very low, meaning that it should be possible to build a classifier that should approach this high accuracy. We hope that their styles can be learned and captured by our models.

During our preprocessing, we wrapped these midi files in a python midi wrapper (pretty midi) that allowed us to work with specific features of each midi representation. While this was done for several reasons, our primary goal was to standardize our input; the two main factors we were worried about were the variety in instruments and length within each piece. To correct this, we extracted each piece's piano roll, which was a representation of the piece as though played from a single piano, and split up each piano roll into evenly sized chunks. We also used a sliding window approach as follows: a piano roll represented by 'abcd' with a window size of two would be split into chunks 'ab,' 'bc,' and 'cd.' The specific size of our window and the sample rate from the chunks of each piano roll were paramaters that we tuned within our architecture. While we tried to keep the number of pieces similar for each composer, we noticed a huge imbalance in the average music length. Therefore, we also experimented with evening up the samples of each composer by randomly discarding generated samples from composers who have more than the others.

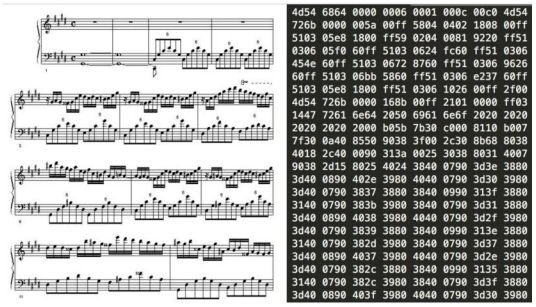


Figure 1: Midi representation for Frederick Chopin's "Fantasie Impromptu" (Piano Solo)

Each of these calculated samples became an input to our model; the size of each of these samples was (128,x) where x was the size of the chunk we wanted (a hyperparameter we tuned, influenced by the piano roll sampling frequency and time length). Because we also varied the window and sample size used, we do not have a specific number of training, validation, and test examples. However, 15% of our data went into our test set. Of the remaining 85%, we treated 15% of that as a validation set, selected as a random subset of the data during training.

4 Methods

Baseline

We noted from our research that many projects saw success using an SVM for classification. In fact, the SVM was consistently the most accurate classifier out of each of the methods that Lebar et al [1] tried and across all of the composers they analyzed. For this reason, we implemented supervised learning via scikit-learn, using both a Multi-layer Perceptron model as well as an Support Vector Classifier model, as our baseline, drawing inspiration from the work done by Sander Shi [2] towards the classification of midi files. At their suggestion, we extracted features including tempo, the number of time signature changes, and resolution, from each file, as well as supplementing this feature extraction with additional information (for example, number of instruments in a piece) that we thought could be important. We did not normalize these feature vectors. The test accuracy we achieved was 0.33.

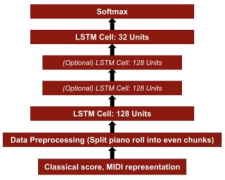
LSTM

We built a 2-4 layer LSTM to classify our input, according to the following standard equations for activation, output, and gate updates. In doing so, we drew inspiration from the work of Rutosi [7] and Dorsey [8], who had likewise worked with music classification via an LSTM and midi files in an RNN, respectively.

$$
\begin{aligned}
a^{<t>} &= g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \\
y^{<t>} &= g_2(W_{ya}a^{<t>} + b_y) \\
\Gamma &= \sigma(Wx^{<t>} + Ua^{<t-1>} + b)
\end{aligned}
$$

We tuned hyperparameters including batch size, learning rate, dropout for each included layer, L2 regularization, and how many layers to include. We also tuned parameters for piano roll cutoff size, sliding window size, and the frequency of samples per second. Below are illustrations de-

picting the architecture of the model and the results it achieved as a result of tuning some of these hyperparameters.



Roll Size	Window Size	Frequency	Batch Size	Learning Rate	Num Layers	Dropout	L2 Reg	Train Acc	Test Acc
1024	512	10	1	0.25	2	0.5, 0.5	0.01	0.2008	0.0895
1024	512	10	32	0.1	2	0.3, 0.3	0.01	0.2291	0.137
1024	512	10	16	0.1	4	0.3, 0.4, 0.4, 0.5	0.1	0.2327	0.2268
1024	512	10	1	0.05	3	0.3, 0.5, 0.3	0.1	0.2277	0.2268
1024	512	10	1	0.05	4	0.4, 0.3, 0.3, 0.4	0.1	0.1959	0.1917
1536	1024	20	1	0.5	3	0.4, 0.5, 0.5	0.5	0.1446	0.2567
1536	**1024**	**20**	**64**	**0.05**	**2**	**0.3, 0.4**	**0.01**	**0.2325**	**0.2567**
1536	1024	20	32	0.1	3	0.4, 0.5, 0.3	0.1	0.2173	0.1104
1536	1024	20	64	0.05	3	0.5, 0.3, 0.3	0.1	0.2166	0.214
1536	1024	20	64	0.5	2	0.3, 0.3	0.01	0.1704	0.0925

These hyperparameters were chosen specifically to vary the complexity of the model and increase the regularization, especially as, early on in our training, we ran into a recurring error where the training error would be nan early on and the performance of the model would drastically decrease. Once tuned properly, the classifications outputted by the model did become more accurate. However, we still believe that one major issue with this model had to do specifically with the preprocessing of our data. If we had more time, we would try increasing the sampling frequency of each piece and decreasing the sliding window size to allow more overlap in each piece. With the numbers we used, we had between one thousand and two thousand training examples, which was not enough to correctly train a sophisticated model.

CNN

As discussed above, our CNN is greatly inspired by the VGG net. We utilized six 1D convolution layers in total. We chose to use 1D convolution, according to the following equation, because for music intuitively it only makes sense to model interactions along the time axis.

$$y[n] = x[n] * h[n] = \sum_{-\infty}^{\infty} x[k] \cdot h[n-k]$$

The first layer had 32 channels and we doubled the number of channels every two layers similar to what was done for the VGG net. In addition, we also used convolution filters of size 3 and max pooling layers of size 2. A standard softmax layer was used last since we were solving a multi-class problem. For regularization, we added l2 regularization in every convolution layer and a dropout layer right before the softmax layer.

Besides the data augmentations we discussed in the dataset section above, we were also creative when doing test set evaluation. To classify a midi file, we sampled chunks of the piano roll similar to the data augmentation method and then classify each chunk using the trained model. We then treated the softmax scores of each chunk as weighted votes for the classes. Finally we picked the class with the highest accumulated vote as the final classification.

5 Experiments/Results/Discussion

For the most part, we didn't quite achieve the performance that we were hoping throughout this project. After some deliberation, we believe that the primary reason for this was our treatment of the input.

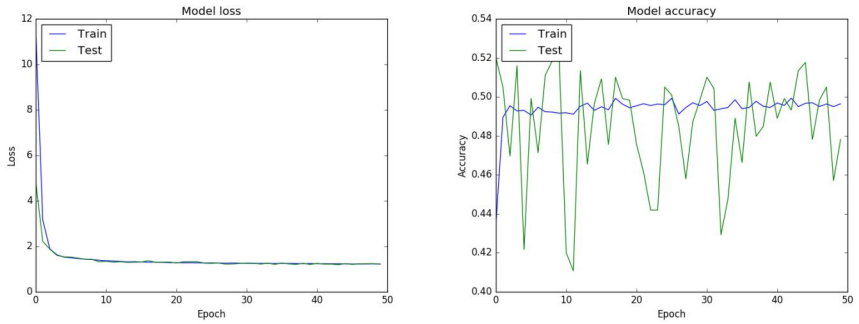
Architecture	Train Examples	Val Examples	Test Exaxmples	Train Acc	Test Acc
Baseline	-	-	-	-	0.33
CNN	**~ 30,000**	**~ 300**	**~ 300**	**0.51**	**0.58**
LSTM 1	1414	313	354	0.2327	0.23
LSTM 2	1514	379	335	0.2325	0.26

The results of training and testing with our different architectures. Here, LSTM 1 refers to the first set of data preprocessing parameters and LSTM 2 refers to the second set. See the chart in the LSTM section for more detail.

In previous sections, we already discussed the limitations with our data augmentation method used for the RNN pipeline. If we sampled more overlapping data, the results could be different. Another

reason that the RNN did so poorly was because of the format and management of the input via the piano roll method, which looks like it created a sparse matrix represented by notes (rows) over time (columns) where a cell is 1 if a note is played at that time and 0 otherwise. This input is too empty for the model to learn anything meaningful or specific about the input itself, much less about the specifics differentiating each of the composers with whom we dealt.

The CNN, on the other hand, did better than both the LSTM and the baseline. For input of our CNN, we explored and settled with randomly sampling all possible piano roll chunks. We sampled piano rolls at a frequency of 10Hz for length 1024 and only took 10% of all possible samples. And we further discarded samples until all classes have equal number of samples. This more aggressive method of data augmentation greatly increased the training data and definitely contributed to the success of our CNN model. We found that our CNN model performs best with relu activation, batch size of 128, adam optimizer with default parameters, l2 regularization of 0.05 and dropout rate of 0.5.



"Test" in the legend is referring to the dev set.

For our best run, the loss graph seems normal. As for the accuracy graph, while the accuracy of the dev set (seen in green line) fluctuated a lot during training, it stayed on average near 50%. We were not worried about this because the classification on individual chunks does not fully reflect the performance of the prediction of the pipeline due to the presence of the voting method. In addition, we find that if we adjust the model the train set accuracy can reach 80% or more but gap between train set and dev set accuracy becomes significant, indicating a strong chance of overfitting. The voting system for the final classification paid off and we ended up achieving 58% accuracy for midi file classification on the test set.

6 Conclusion/Future Work

After viewing our results, we think that one major shortcoming of our project is that we don't have enough data samples from each composer to properly train our model, even with data augmentation techniques. Going forward, we would either collect more midi files or convert existing audio files to midi format to enhance our dataset. The next step would be to enhance our input; from our results, it seems that the piano roll representation by itself may be perhaps too sparse to train our models to recognize the finer differences between composers.

In addition to collecting more data, we could have spent more time tuning our hyperparameters, particularly in regards to model complexity and number of layers. We think that there is great potential to significantly improve the results of the LSTM.

We would lastly propose experimenting with changed types of models. The first is a dilated convolutional neural net [9], as this would allow the model to focus on the piece at both a specific and a coarser level. Finally, we think that there is potential to improve upon both the results of our CNN and our baseline SVM by incorporating the output of the CNN as input into th SVM.

7 Github Repository

The Github repository containing all of our code can be found here: https://github.com/chramsey/CS230-Final-Project

8 Contributions

Christina worked on data preprocessing, writing and tuning the baseline, writing and tuning the LSTM, and helping to create the poster and paper. Chenduo worked on data augmentation, writing and tuning the CNN model, helping to create the poster and paper. Daniel worked on data preprocessing, error analysis, and helping to create the poster and paper.

References

[1] Lebar, Justin & Chang, Gary & Yu, David. (2012). Classifying Musical Scores by Composer: A machine learning approach.

[2] Shi, Sander. (2018), Github Repository. Midi Classification Tutorial. https://github.com/sandershihacker/midi-classification-tutorial.

[3] Cataltepe, Zehra, Yusuf Yaslan, and Abdullah Sonmez. "Music genre classification using MIDI and audio features." EURASIP Journal on Advances in Signal Processing 2007, no. 1 (2007): 036409.

[4] Huang, Allen, and Raymond Wu. "Deep learning for music." arXiv preprint arXiv:1606.04930 (2016).

[5] Kalingeri, Vasanth, and Srikanth Grandhe. "Music generation with deep learning." arXiv preprint arXiv:1612.04928 (2016).

[6] Lee, Jongpil et al. “Sample-level Deep Convolutional Neural Networks for Music Auto-tagging Using Raw Waveforms.” CoRR abs/1703.01789 (2017): n. pag.

[7] Ruotsi, Ruoho. (2017), Github Repository. LSTM-Music-Genre-Classification. https://github.com/ruohoruotsi/LSTM-Music-Genre-Classification/blob/master/lstm_genre_classifier_keras.py

[8] Dorsey, Brannon. (2017), Github Repository. Midi-Rnn. https://github.com/brannondorsey/midi-rnn/blob/master/train.py

[9] Oord, A.V.D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. and Kavukcuoglu, K., 2016. Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.