



Classification of Musical Scores by Composer

Christina Ramsey | Chenduo Huang | Daniel Costa

cmramsey@stanford.edu | cdhuang@stanford.edu | dcosta2@stanford.edu



Abstract

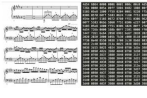
The aim of this project is, given a musical score, to accurately predict which composer wrote it. We believed that this project will be an interesting experiment in audio classification and will potentially demonstrate where composers were influenced by other composers; at the same time, we believe that 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.

Dataset and Features

Our dataset is a collection of 450 classical compositions represented as midi files, hand curated from <http://www.midworld.com/classic.htm>.

 A sample midi file. While audio files potentially include mistakes or performers' interpretations, midi files objectively record instructions for how a piece should be played as the composer instructed. This encoding is visualized and interpreted as shown here.

To standardize our input, we extracted the piano roll, which was a representation of the piece as though played from a single piano, from each piece and split up each piano roll into evenly sized chunks (parameter x) so that each generated sample was now represented as a matrix of size $(128 \times x)$. Each row represents a note; each column is a time slice in which a note is recorded as played.

Features therefore include the the duration and pitch of each note. The piano roll implementation was chosen to standardize across multiple instruments. Chunk size was chosen to standardize piece length. These were necessary to accurately represent the important aspects of each piece while standardizing them with respect to other pieces.

Models and Results

LSTM

We built a 2-4 layer LSTM to classify our input, according to the following standard equations for activation, output, and gate updates.

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$
$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$
$$\Gamma = \sigma(W_x x^{<t>} + U a^{<t-1>} + b)$$

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 the results it achieved.

Roll Size	Window Size	Frequency	Batch Size	Learning Rate	Num Layers	Dropout	L2 Reg	Train Acc	Test Acc
1024	512	10	1	0.05	2	0.5, 0.5	0.01	0.2008	0.0895
1024	512	10	10	0.1	2	0.5, 0.5	0.01	0.2091	0.137
1024	512	10	16	0.1	4	0.3, 0.4, 0.4, 0.5	0.1	0.2307	0.208
1024	512	10	1	0.05	3	0.5, 0.5, 0.3	0.1	0.2277	0.208
1024	512	10	1	0.05	4	0.5, 0.5, 0.5, 0.4	0.1	0.1950	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	3	0.5, 0.4	0.01	0.2335	0.2667
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	0.5, 0.3	0.01	0.1704	0.0925

Baseline: SVM

Influenced by the work done by Lebar et al [1] and Shi [2], we eventually implemented supervised learning via scikit-learn, using both a Multi-layer Perceptron model as well as an Support Vector Classifier model, as our baseline.

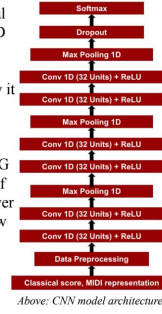
We extracted features from each file including

- Tempo
- the number of time signature changes
- Resolution
- Number of instruments in a piece and other supplements that we thought could be important. We did not normalize these feature vectors. The test accuracy we achieved was 0.33.

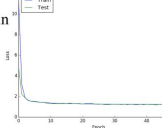
CNN

We implemented a sequential CNN model using mainly 1D convolutional layers. We chose to use 1D convolution because for music intuitively it only makes sense to model interactions along the time axis. The architecture of this model is inspired by the VGG model, where the channels of the layers starts at some power of two and doubles every few layers. In addition, the stride of the convolutional and pooling layers are both kept small. To reduce overfitting, we added a dropout layer before the softmax layer. L2 regularization is also added in all convolutional layers. We defined our convolution as follows and produced the following loss.

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



Above: CNN model architecture



Results

As shown, the CNN vastly outperformed the LSTM, likely due to the higher quantity of training examples.

Architecture	Train Examples	Val Examples	Test Examples	Train Acc	Test Acc
Human Expert (Estimate)	-	-	-	-	>0.95
Baseline	-	-	-	-	0.33
CNN	~30,000	~300	~300	0.72	0.41
LSTM 1	1414	313	354	0.2327	0.23
LSTM 2	1514	370	335	0.2325	0.26

However, none of the methods, including the baseline, were anywhere close to the accuracy of the human expert, who we expect would get at least a 95% accuracy on this task (and there are some humans who we believe should get 100% accuracy on this task).

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.

The 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 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. This is likely due to the difference in preprocessing that led to the increased number of training examples.

Model Shortcomings and Future Research

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.

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/sandershiahacker/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).