Project 2 - Composer Classifier on MuseScore Sheet Music

Nathan Peluso Jamie Mickaill Erwan Umlil

***Abstract*—Composer classification is a research area that involves using machine learning techniques to identify the composer of a piece of music. Digital music scores are a viable source of data for composer classification as they offer a more explicit and precise description of music than other semantic representations. In this project we explore the task of composer classification using machine learning methods applied to digital music scores. An abundance of digital music score data is available online. The successful labelling of this data has potential to aid our understanding of music through learning the stylistic characteristics of composers and finding new ways to analyze music.**

I. Introduction

Composer classification from music involves using machine learning methods to identify the composer of a piece of music from its musical features. These features may be embedded in various semantic representations of the music, such as machine readable formats, symbolic translations of sheet music, sheet music images and audio files. This project explores the processing of digital score representations of music. Digital music scores provide a more detailed and explicit representation of music compared to other semantic representations making them a promising source of information for composer classification.

Our goal during this project was to train a Recurrent Neural Network based model (LSTM, see [9]) on a set of digital scores to learn to classify the composer of any further piece. A symbolic representation of the notes in the score were processed along with the composer labels extracted from the score metadata. Below we discuss the previous work in this field along with details of the pre-processing methods, model architecture and training. Finally we present our results and discuss our findings and limitations.

This project has been supervised by Johannes Hentschel, Ren Zeng and Christoph Finkensiep from the Digital and Cognitive Musicology Lab at EPFL.

II. Related work

hch used various feature extraction techniques to extract relevant information from music representations, in order to accurately identify its composer. Most previous composer classification has been performed with machine-readable symbolic music representations, such as MIDI. Features extracted and learned from these digital representations include rhythm, melody, texture, tempo, tonality, harmonic intervals and n-grams [3] [5]. Sheet music can be used to obtain machine-readable encodings using Optical Music Recognition systems or alternatively, directly processed with computed vision methods [10] . Additionally, research exists for models that work with audio data. Machine learning models used to learn from symbolic representations include Neural network approaches [6] [11] [10] [1] [2] and Markov models [8]. Based on the symbolic nature of our data and the success of recent neural network methods [7] [9], we selected our neural network model architecture While successful classification results have been achieved with these previous representations, there exists little research [4] directly processing digital music score data.

III. Initial data format

The available raw data was a set of around 15 million scores in Musescore format (.mscz): each file contains all 'musical' information of the score, plus some metadata in XML format. From these 15 million scores, corrupted or outdated (older versions of Musescore that could not be converted) files were filtered. Once done, every .mscz file was parsed by a library developed by the DCML (website), 'ms3', to extract the contained musical content in several formats (III-A), and all metadata available that might contain the composer name (III-B). This first step was conducted on a HPC computer in a highly parallelized manner, given the amount of scores to be converted: even then, this processing task took around 72hours of computation to complete.

A. Music

The musical information contained in the Musescore files was extracted as a list of notes, characterized by their onset (beginning of note), duration, pitch and octave. This list of notes stored in a .tsv file could then be processed by our model (V).

B. Metadata

Upon creation of a score on MuseScore's website or on their software, various pre-defined fields can be filled in, such as Composer, Lyricist, Date of composition, etc. To make this system as generic as possible, MuseScore also makes it it possible to add any number of custom-defined fields. As a result, there are as many ways of filling in a score's metadata as there are users: the resulting metadata presents very little structure, and needs heavy preprocessing to extract usable labels. Moreover, this preprocessing is hard to completely automatize, given the variety of possible labels referring to the same composer. The table I displays the top 10 most common 'Composer' field data, without any processing, which reveals some of the ways available labels will have to be modified: merging different spellings, removing generic values, ignoring numeric values, etc. The following section explains how we processed this messy data to generate a clean set of scores with confident labels.

Composer	Count
Composer	24810
anon.	4274
Pierangelo Fernandes Carera	3855
Traditional	3162
Yohei Kato * 加藤 洋平	2794
Yohei Kato * 加藤洋平	2353
Trad.	2223
Toby Fox	1757
Johann Sebastian Bach	1757
Trad	1699

TABLE I
TOP 10 EXTRACTED COMPOSERS, RAW 'COMPOSER' FIELD

IV. LABEL EXTRACTION

Composer names extracted from the "Composer" field are considered reliable, but possible composer names have also been extracted from other, less-structured fields to account for incorrect user input formatting. The extracted names were cleaned using various string processing rules, detailed in Appendix A.

A. Overview of extracted composer names

Of the initial 1 504 110 entries, 871 659 contained some information in the 'composer' fields of the metadata (58%). After cleaning, 744 311 contained a valid composer (49.5%). When extending search for clean composer names into alternative metadata fields, an additional 131 261 files were assigned a composer, for a total of 875 571 (58.2%).

Composer	Count	% (of Top 10)	% (all)
J. S. Bach	9498	23.2	1.2
Y. Kato	5156	12.6	0.7
W. A. Mozart	4758	11.6	0.6
P. F. Carera	4438	10.9	0.6
T. Fox	4081	10.0	0.6
L. V. Beethoven	3994	9.8	0.5
J. Williams	3082	7.5	0.4
K. Kondo	3033	7.4	0.4
A. Vivaldi	1403	3.4	0.2
A. Antão	1392	3.4	0.2
Total usable train set	40835		

TABLE II
TOP 10 EXTRACTED COMPOSERS, 'COMPOSER' FIELD ONLY

The table II describes the distribution of the top 10 composers, post cleaning. As indicated in the last column, the composer space is still quite wide. We also noted that the distribution was noticeably unbalanced, as such this was taken into account when creating our training set.

Composer	Count	% (of Top 10)	% (all)
J. S. Bach	9972	22.2	1.1
Y. Kato	5156	11.5	0.6
W. A. Mozart	4899	10.9	0.6
P. F. Carera	4438	9.9	0.5
T. Fox	4187	9.3	0.5
L. V. Beethoven	4133	9.2	0.5
F. Nordberg	3710	8.3	0.4
J. Williams	3205	7.1	0.4
K. Kondo	3066	6.8	0.4
J. Kerr	2220	4.9	0.3
Total usable train set	44986		

TABLE III
TOP 10 EXTRACTED COMPOSERS, 'COMPOSER' AND ADDITIONAL FIELDS

The table III describes the dataset labeled from all available fields. As expected, the distribution of composers did not change radically, but the size of the usable training set increased by around 10%.

B. Retained labels

We decided to have our model classify composers among 10 top composers, to reduce the computational cost of training and to limit the effect of composers with few usable scores. The top 10 composer list was defined by the extracted composers from 'Composer' field only, as more reliable. Using this list instead of an arbitrary list of 'famous' composers allow us to consider more modern composers, or even prolific (several thousands scores composed) MuseScore users that are not "professional composers": P. F. Carera (MuseScore Page) and Y. J. (MuseScore Page) are exactly in this case.

The final list is hence the composers found in II.

V. SCORE INFORMATION PROCESSING

Scores have to be converted to matrices in order to be processed by a machine learning algorithm. As we decided to restrict the data to the pitched events (the notes), we represent the pitch of a note. We use a kind of two-dimensional one-hot encoding to represent a given moment of the piece. The two axis are the octave (11 values, from -1 to 9) and the note name (12 values, from C to B), as shown in Figure 1. If two instruments play the same note at the same time, we decided to still keep a 1 in the one-hot matrix, since it is easier to process for a neural network (zeros and ones embody the activation state of neurons). Note that we use two dimensions rather than a one-dimensional one-hot encoding: we will convolute over each dimension, and in music same notes in different octaves are linked, so we also want to convolute over the octave axis.

Octave	Notes											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 1. Matrix that represents an instant when a G2 and a C4 are simultaneously played

The previous was about representing music at a given moment. We also discretize the time axis. Given the value of parameter δ, which represent a time interval (*e.g.* $1/32$), we compute matrices at every time $n\delta, 0 \leq n \leq N_{score}$ where N_{score} is the number of points in this discretization of the processed score.

Finally, we need to get equally sized data to feed the architecture described in VI. That is why we cut each score into chunks of size $4/\delta$. This size value represents exactly one measure in the typical music format "$4/4$", so we decided to use it for all scores to keep a constant chunk size. At the end

of preprocessing, for each score we have a list of chunks, each containing a 3D tensor of shape $\left(\frac{4}{\delta}, 11, 12\right)$.

VI. Learning Model

A. Architecture

We trained a deep neural network to classify the composer from the list of chunks representing a score. The chosen architecture is the following:

- three 3D-convolutional layers, each one being followed by a 3D-max-pooling layer;
- one LSTM layer;
- two dense layers, the first one being activated by a ReLU.

The forward pass consists in going through the three convolutional/max-pooling pairs for each chunk, so that we get a list of abstract embedding tensors we can flatten. Then we interpret this list of chunk abstractions as a sequence to feed the LSTM layer. Finally we use the last hidden state of LSTM as input for the dense layers.

This architecture comes from both literature and intuition. Indeed, in most of papers that deal with deep learning and music, CNNs and LSTMs are used. Actually, convolutional layers are useful to compute features from a signal. Since our scores are embedded in a 3D signal (time being one dimension), it seems logical to use 3D-convolutional layers. Then, LSTM might benefit from the fact that music is a kind of sequence and help refine, chunk by chunk, the predicted class of the input.

B. Hyperparameters

This model, as well as the preprocessing steps, involves a lot of hyperparameters. Considering the limited amount of time we had to train the model, we decided to set some hyperparameters and to experiment on other ones. Among those we set with no experimentation or without a rigorous one, we set a few ones arbitrarily and others according to intuitive justifications we expose here.

The following hyperparameter was set arbitrarily or without any rigorous experiment:

- batch size: with some quick tests we noticed that our results were better when batch size was 1, so we kept it;

The following values are based on intuition:

- δ: $1/\delta$ must be a power of 2 and it is the shortest note we can represent using our one-hot matrices. According to some musicians advice, $1/128$ should be enough to represent very well our data, but for computational issues and to be able to experiment a lot we chose a default value of $1/32$ which should be enough to capture the most important features of the data;
- chunk size: in many musical styles, similar loops of a certain amount of measures are repeated among time. That is why we decided to consider that our chunk duration will be the equivalent of one measure in the most classical music notation ("4/4"). This means a chunk has $4/\delta$ time points.
- convolutional strides: we set them to 1. Actually, there makes musically no sense to try another convolutional stride than 1, since contrary to images, we want to adjacency is extremely important in the matrices we use.

Finally, the following hyperparameters were our degrees of freedom to experiment:

- learning rate: the best learning rate we found was 0.0001;
- convolutional number of channels and LSTM hidden size: we first tried values from the litterature and then we tweaked.

C. Training

We trained the model using a standard pipeline. The dataset is divided into a train set, a validation set and a test set that is kept for testing only, while validation test can be used to tweak hyperparameters. The loss we use is the multi-class cross-entropy.

We restrict the dataset we use by selecting the top 10 composers who have the most available scores. This reduces the impact of possibly wrong labels in our dataset and makes us sure that every composer has enough score available for training.

When using the whole dataset, we trained our model on 60% of available data, we validated using 20% and finally we tested on the last 20%.

The scripts allow to use either a "zipped mode" or not, meaning that the *ID.tsv* files, that contain the notes, can live inside a zip file. The zipped mode needs less space but is slightly slower.

VII. Results

A. Balanced v.s Unbalanced training

As we observed in IV-A, the distribution of composers in our dataset is unbalanced: J. S. Bach for instance makes up to 22% of the top 10 composers scores. We decided, as the main experiment of this report, to train our model both on a naturally sampled train set and on a balanced one, to remove this bias. Training on naturally sampled set allows our model to rely on this bias, similarly to how a human classifier would first consider famous composers when trying to identify a piece. On the other hand, enforcing a balanced dataset (as many pieces for every composer of the top 10) allows to reduce this 'famousness bias' and enforce a classification agnostic of it. In the case this bias was very specific to our given dataset, or a side-effect of our label extraction, mitigating it before training the model could help get a less over-fitted classifier.

We trained and tested on four available datasets:

- full unbalanced: all scores we have from the top 10 composers
- max balanced: as many scores as possible from the top 10 composers so that the dataset is balanced (same number of scores for each composer)
- small unbalanced: a small batch of scores that respects the proportions of the full dataset
- small balanced: a small balanced batch of scores

B. Accuracies

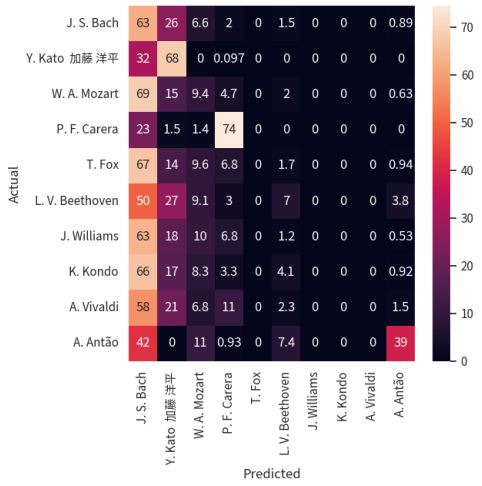
We computed test accuracies on the dedicated test sets. The experiments correspond to the datasets we used.

Experiment	Loss	Accuracy %
Random on full unbalanced	NA	13.4
Small unbalanced	1.8607	34.3
Small balanced	1.9450	30.8
Full unbalanced	**1.8041**	**34.9**
Max balanced	1.8904	33.4

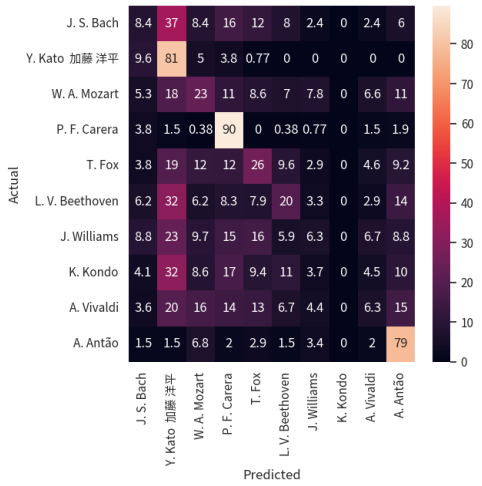
TABLE IV
RESULTS ON THE PROVIDED DATASETS

We can notice in Table IV that the results are better on balanced datasets since the network can learn from the distribution of composers.

C. Confusion Matrix



(a) Dataset 'Full unbalanced'



(b) Dataset 'Max balanced'

Fig. 2. Confusion matrix, p($predicted|actual$) in %

The figure 2 displays in a confusion matrixes how our model performs: which composer(s) it classifies easily and which ones he struggles with. In the "unbalanced dataset" matrix, we can observe here that our model tends to classify a lot of scores (more than 26% of scores for all composers) as J. S. Bach, due to the bias identified earlier. This effect disappears completely when training on a balanced dataset: he even becomes one of the less often classified composers. What's interesting to note is also that some composers such as T. Fox or K. Kondo are significantly more often classified as J. S. Bach than their own label, in the unbalanced dataset: this might mean the weight of their own 'intrinsic style' is less than the weight of the identified 'Bach bias'. Another interesting feature of our classifier is the lack of symmetry: for instance, W. A. Mozart is very often misclassified as J. S. Bach (twice more than the correct label), while J. S. Bach is misclassified as W. A. Mozart in less than 10% of classifications (unbalanced dataset). On both datasets, some composers seem to be confidently identified, such as Y. Kato or P. F. Carera: one could interpret this as these composers having a more precise style of composing. By simply looking at some scores by P. F. Carera, this idea seems to be plausible, as the temporal structure and the notes he uses (especially at the scale of a single bar) appear to be very redundant. This hypothesis could also be correlated by the case of K. Kondo. Indeed, this composer worked for Nintendo and as such he is registered as composer for hundreds of video game soundtracks: due to the variety of games, environments to describe, etc, it seems plausible that defining his exact style would be harder than other composers.

VIII. CONCLUSION

We demonstrated that our method allows to predict the composer from a score with an accuracy which is much better than at random. However this accuracy is not that high and one might argue an expert human may be better at this task. We propose a few points to explain these results and how we could improve the method.

First, we made a very strong assumption: we said that every composer has an inherent style that can be characterized. But this might not always be true. One composer may have composed more various pieces than another one. Our model seems to be able to define a strongly identifiable style for some composers, while struggling to characterize some others.

Secondly, we restricted the data we used to the pitched events (the notes). One might try to improve the method by incorporating, for instance, several types of events (e.g. drums, rests,...).

Finally, the initial data used and the label extraction task could be perfected. As a matter of fact, we started from an unreliable dataset, trying to find the best composer for each score from its metadata, but there are probably many false composers associated to scores. A way to improve this would be to use a more reliable dataset, even if smaller. Furthermore, scores are uploaded on MuseScore by users, with no control. There are two drawbacks here. First, the scores could be of low quality. Second, the dataset probably contains multiple variants of the same scores, e.g. for very famous composers. So when we split our dataset between train, validation and test sets, we might have very similar scores that are used for training and for testing, leading to a kind of overfitting.

The project highlights the importance of considering prior assumptions, event selection, preprocessing methods, and dataset reliability when processing digital music score information. It also shows the potential for computational models to learn composer-specific features from music score data.

APPENDIX

Composer strings were cleaned with the following process:

- Whitespace and newlines are stripped from the outside of the string
- String is split by newline and comma characters, keeping only the first substring. This removes any additional composers, retaining only the first mentioned composer.
- Non alphanumeric characters are removed except those often used in names (" ","","","-","."). Non-ascii characters are retained.
- String format is set to Title case i.e. each separate word is capitalized
- '.' period characters are replaced by '. '. This consolidates composers of the format (A.B. Composer, A.B.Composer and A. B. Composer)
- Any double spaces introduced (e.g '. ') are removed. - Presence of specific words at certain indices (e.g. 'Arr' at index 0) were determined to be disqualifying conditions and the composer removed. This was to account for arrangers listed as the only entity in the composer field.
- A check is performed for distinct known composer substrings. This is used to reduce variance in labels for common, easily identifiable composers (e.g. Mozart, Beethoven, Debussy). If a known substring is found, the cleaning process terminates and the composers name is returned. This short list was computed by a first pass without this option.
- The string split on spaces is trimmed before the first numeric character if any (e.g. ['Johann', 'Sebastian', 'Bach', '17th' 'Century'] $\rightarrow$ ['Johann', 'Sebastian', 'Bach']) This is to remove unnecessary dates
- Irrelevant words ('by', 'in', ...) and numerical terms are from this substring list. Irrelevant words list was iteratively constructed from manual inspection of processing results.
- Any substring containing only " ',.- " characters are removed
- Composer strings $\geq$ length of 6 words are trimmed to the first two words. (This allows for titles such as 'The x of the y').
- If the length of the joined composer string is strictly less than 4 characters it is removed.
- Cleaned composer string is reduced to an initial-ised form. Each non-surname word is initialized (e.g. Johann Sebastian Bach $\rightarrow$ J. S. Bach) This reduces variation in labels, better pooling together compositions from the same composer, at the cost of possible mislabelization of "A. Name" homonyms.
- Only for alternative text fields: only composers with a minimum of 2 names are retained. This increases the quality of composers derived from alternative text fields.

REFERENCES

[1] Keunwoo Choi, Gyorgy Fazekas, and Mark Sandler. Convolutional recurrent neural networks for music classification.

[2] Keunwoo Choi, Gyorgy Fazekas, and Mark Sandler. Automatic tagging using deep convolutional neural networks. 2016.

[3] Dorien Herremans, David Martens, and Kenneth Sörensen. Composer classification models for music-theory building. *Computational Music Analysis*, page 369–392, 2015.

[4] Qin Lin. Music score recognition method based on deep learning. *Computational Intelligence and Neuroscience*, 2022:1–12, 2022.

[5] Corey McKay, Julie Cumming, and Ichiro Fujinaga. Jsymbolic 2.2: Extracting features from symbolic music for use in musicological and mir research.

[6] Chaitanya Prasad N, Krishnakant Saboo, and Bipin Rajendran. Composer classification based on temporal coding in adaptive spiking neural networks. *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015.

[7] Zain Nasrullah and Yue Zhao. Music artist classification with convolutional recurrent neural networks. *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019.

[8] E. Pollastri and G. Simoncelli. Classification of melodies by composer with hidden markov models. *Proceedings First International Conference on WEB Delivering of Music. WEDELMUSIC 2001.*

[9] Pallabi Saikia, Dhwani Dholaria, Priyanka Yadav, Vaidehi Patel, and Mohendra Roy. A hybrid cnn-lstm model for video deepfake detection by leveraging optical flow features. *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022.

[10] Dnyanesh Walwadkar, Elona Shatri, Benjamin Timms, and Gyorgy Fazekas. Compidnet: Sheet music composer identification using deep neural network, 2022.

[11] Wenbo Yi. Classifying classical piano music based on composer's native language using machine learning. *https://mct-master.github.io/machine-learning/2021/09/19/wenbo-ml-nl.html.*