COMPOSER STYLE CLASSIFICATION OF PIANO SHEET MUSIC IMAGES USING LANGUAGE MODEL PRETRAINING

TJ Tsai
Harvey Mudd College
`ttsai@g.hmc.edu`

Kevin Ji
Harvey Mudd College
`kji@g.hmc.edu`

ABSTRACT

This paper studies composer style classification of piano sheet music images. Previous approaches to the composer classification task have been limited by a scarcity of data. We address this issue in two ways: (1) we recast the problem to be based on raw sheet music images rather than a symbolic music format, and (2) we propose an approach that can be trained on unlabeled data. Our approach first converts the sheet music image into a sequence of musical "words" based on the bootleg feature representation, and then feeds the sequence into a text classifier. We show that it is possible to significantly improve classifier performance by first training a language model on a set of unlabeled data, initializing the classifier with the pretrained language model weights, and then finetuning the classifier on a small amount of labeled data. We train AWD-LSTM, GPT-2, and RoBERTa language models on all piano sheet music images in IMSLP. We find that transformer-based architectures outperform CNN and LSTM models, and pretraining boosts classification accuracy for the GPT-2 model from 46% to 70% on a 9-way classification task. The trained model can also be used as a feature extractor that projects piano sheet music into a feature space that characterizes compositional style.

1. INTRODUCTION

We've all had the experience of hearing a piece of music that we've never heard before, but immediately recognizing the composer based on the piece's style. This paper explores this phenomenon in the context of sheet music. The question that we want to answer is: "Can we predict the composer of a previously unseen page of piano sheet music based on its compositional style?"

Many previous works have studied the composer classification problem. These works generally fall into one of two categories. The first category of approach is to construct a set of features from the music, and then feed the features into a classifier. Many works use manually designed features that capture musically meaningful information (e.g. [1] [2] [3] [4]). Other works feed minimally preprocessed representations of the data (e.g. 2-D piano rolls [5] [6] or tensors encoding note pitch & duration information [7] [8]) into a convolutional model, and allow the model to learn a useful feature representation. The second category of approach is to train one model for each composer, and then select the model that has the highest likelihood of generating a given sequence of music. Common approaches in this category include N-gram language models [9] [10] [11] and Markov models [12] [13].

Our approach to the composer classification task addresses what we perceive to be the biggest common obstacle to the above approaches: lack of data. All of the above approaches assume that the input is in the form of a symbolic music file (e.g. MIDI or **kern). Because symbolic music formats are much less widely used than audio, video, and image formats, the amount of training data that is available is quite limited. We address this issue of data scarcity in two ways: (1) we re-define the composer classification task to be based on sheet music images, for which there is a lot of data available online, and (2) we propose an approach that can be trained on unlabeled data.

Our work takes advantage of recent developments in transfer learning in the natural language processing (NLP) community. Prior to 2017, transfer learning in NLP was done in a limited way. Typically, one would use pretrained word embeddings such as word2vec [14] [15] or GloVe [16] vectors as the first layer in a model. The problem with this paradigm of transfer learning is that the entire model except the first layer needs to be trained from scratch, which requires a large amount of labeled data. This is in contrast to the paradigm of transfer learning in computer vision, where a model is trained on the ImageNet classification task [17], the final layer is replaced with a different linear classifier, and the model is finetuned for a different task. The benefit of this latter paradigm of transfer learning is that the entire model except the last layer is pretrained, so it can be finetuned with only a small amount of labeled data. This paradigm of transfer learning has been widely used in computer vision in the last decade [18] using pretrained models like VGG [19], ResNet [20], Densenet [21], etc. The switch to ImageNet-style transfer learning in the NLP community occurred in 2017, when Howard et al. [22] proposed a way to pretrain an LSTM-based language model on a large set of unlabeled data, add a classification head on top of the language model, and then finetune the classifier on a new task with a small amount of labeled data. This was quickly followed by sev-

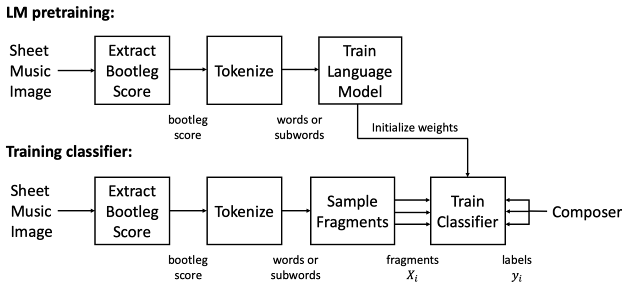

Figure 1. Overview of proxy classifier training. A language model is first trained on a set of unlabeled data, the classifier is initialized with the pretrained language model weights, and then the classifier is finetuned on a small set of labeled data.

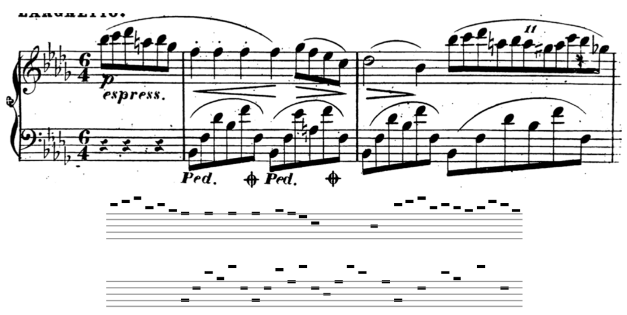

Figure 2. A short section of sheet music and its corresponding bootleg score. Staff lines in the bootleg score are shown for reference, but are not present in the actual feature representation.

eral other similar language model pretraining approaches that replaced the LSTM with transformer-based architectures (e.g. GPT [23], GPT-2 [24], BERT [25]). These pretrained language models have provided the basis for achieving state-of-the-art results on a variety of NLP tasks, and have been extended in various ways (e.g. Transformer-XL [26], XLNet [27]).

Our approach is similarly based on language model pretraining. We first convert each sheet music image into a sequence of words based on the bootleg score feature representation [28]. We then feed this sequence of words into a text classifier. We show that it is possible to significantly improve the performance of the classifier by training a language model on a large set of unlabeled data, initialize the classifier with the pretrained language model weights, and finetune the classifier on a small amount of labeled data. In our experiments, we train language models on all piano sheet music images in the International Music Score Library Project (IMSLP) [1] using the AWD-LSTM [29], GPT-2 [24], and RoBERTa [30] language model architectures. By using pretraining, we are able to improve the accuracy of our GPT-2 model from 46% to 70% on a 9-way classification task. [2]

2. SYSTEM DESCRIPTION

We will describe our system in the next four subsections. In the first subsection, we give a high-level overview and rationale behind our approach. In the following three subsections, we describe the three main stages of system development: language model pretraining, classifier finetuning, and inference.

2.1 Overview

Figure 1 summarizes our training approach. In the first stage, we convert each sheet music image into a sequence of words based on the bootleg score representation [28], and then train a language model on these words. Since this task does not require labels, we can train our language model on a large set of unlabeled data. In this work, we train our language model on all piano sheet music images in the IMSLP dataset. In the second stage, we train a classifier that predicts the composer of a short fragment of music, where the fragment is a fixed-length sequence of symbolic words. We do this by adding one or more dense layers on top of the language model, initializing the weights of the classifier with the language model weights, and then finetuning the model on a set of labeled data. In the third stage, we use the classifier to predict the composer of an unseen scanned page of piano sheet music. We do this by converting the sheet music image to a sequence of symbolic words, and then either (a) applying the classifier to a single variable length input sequence, or (b) averaging the predictions of fixed-length crops sampled from the input sequence. We will describe each of these three stages in more detail in the following three subsections.

The guiding principle behind our approach is to maximize the amount of data. This impacts our approach in three significant ways. First, it informs our choice of data *format*. Rather than using symbolic scores (as in previous approaches), we instead choose to use raw sheet music images. While this arguably makes the task much more challenging, it has the benefit of having much more data available online. Second, we choose an approach that can utilize unlabeled data. Whereas labeled data is usually expensive to annotate and limited in quantity, unlabeled data is often extremely cheap and available in abundance. By adopting an approach that can use unlabeled data, we can drastically increase the amount of data available to train our models. Third, we use data augmentation to make the most of the limited quantity of labeled data that we do have. Rather than fixating on the page classification task, we instead define a proxy task where the goal is to predict the composer given a fixed-length sequence of symbolic words. By defining the proxy task in this way, we can aggressively subsample fragments from the labeled data, resulting in a much larger number of unique training data points than there are actual pages of sheet music. Once the proxy task classifier has been trained, we can apply it to the full page classification task in a straightforward manner.

[1] `http://imslp.org/`
[2] Code can be found at `https://github.com/tjtsai/PianoStyleEmbedding`.

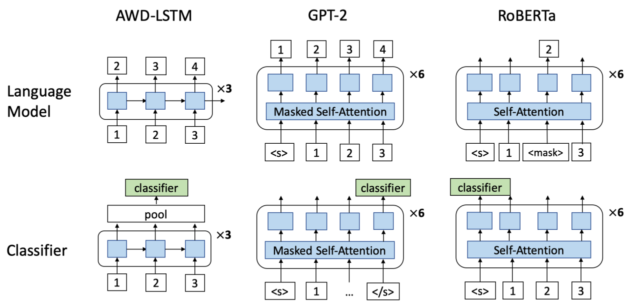

Figure 3. Overview of AWD-LSTM, GPT-2, and RoBERTa language models (top) and classifiers (bottom). Boxes in blue are trained during the language modeling phase and used to initialize the classifier.

2.2 Language Model Pretraining

The language model pretraining consists of three steps, as shown in the upper half of Figure 1. These three steps will be described in the next three paragraphs.

The first step is to convert the sheet music image into a bootleg score. The bootleg score is a low-dimensional feature representation of piano sheet music that encodes the position of filled noteheads relative to the staff lines [28]. Figure 2 shows an example of a section of sheet music and its corresponding bootleg score representation. The bootleg score itself is a $62 \times N$ binary matrix, where 62 indicates the total number of possible staff line positions in both the left and right hands, and where N indicates the total number of estimated simultaneous note onset events. Note that the representation discards a significant amount of information: it does not encode note duration, key signature, time signature, measure boundaries, accidentals, clef changes, or octave markings, and it simply ignores non-filled noteheads (e.g. half or whole notes). Nonetheless, it has been shown to be effective in aligning sheet music and MIDI [28], and we hypothesize that it may also be useful in characterizing piano style. The main benefit of using the bootleg score representation over a full optical music recognition (OMR) pipeline is processing time: computing a bootleg score only takes about 1 second per page using a CPU, which makes it suitable for computing features on the entire IMSLP dataset.[3] We use the code from [28] as a fixed feature extractor to compute the bootleg scores.

The second step is to tokenize the bootleg score into a sequence of word or subword units. We do this differently for different language models. For word-based language models (e.g. AWD-LSTM [29]), we consider each bootleg score column as a single word consisting of a 62-character string of 0s and 1s. We limit the vocabulary to the $30,000$ most frequent words, and map infrequent words to a special unknown word token <unk>. For subword-based language models (e.g. GPT-2 [24], RoBERTa [30]), we use a byte pair encoding (BPE) algorithm [32] to learn a vocabulary of subword units in an unsupervised manner. The BPE algorithm starts with an initial set of subword units (e.g. the set of unique characters [33] or the $2^8 = 256$ unique byte values that comprise unicode characters [34]), and it iteratively merges the most frequently occurring pair of adjacent subword units until a desired vocabulary size has been reached. We experimented with both character-level and byte-level encoding schemes (i.e. representing each word as a string of 62 characters vs. a sequence of 8 bytes), and we found that the byte-level encoding scheme performs much better. We only report results with the byte-level BPE tokenizer. For both subword-based language models explored in this work, we use the same shared BPE tokenizer with a vocabulary size of $30,000$ (which is the vocabularly size used in the RoBERTa model). At the end of the second step, we have represented the sheet music image as a sequence of words or subword units.

The third step is to train a language model on a set of unlabeled data. In this work, we explore three different language models, which are representative of state-of-the-art models in the last 3-4 years. The top half of Figure 3 shows a high-level overview of these three language models. The first model is AWD-LSTM [29]. This is a 3-layer LSTM architecture that makes heavy use of regularization techniques throughout the model, including four different types of dropout. The output of the final LSTM layer is fed to a linear decoder whose weights are tied to the input embedding matrix. This produces an output distribution across the tokens in the vocabulary. The model is then trained to predict the next token at each time step. We use the fastai implementation of the AWD-LSTM model with default parameters. The second model is openAI's GPT-2 [24]. This architecture consists of multiple transformer decoder layers [35]. Each transformer decoder layer consists of a masked self-attention, along with feed-forwards layers, layer normalizations, and residual connections. While transformer encoder layers allow each token to attend to all other tokens in the input, the transformer decoder layers only allow a token to attend to previous tokens.[4] Similar to the AWD-LSTM model, the outputs of the last transformer layer are fed to a linear decoder whose weights are tied to the input embeddings, and the model is trained to predict the next token at each time step. We use the huggingface implementation of the GPT-2 model with default parameters, except that we reduce the vocabulary size from $50,000$ to $30,000$ (to use the same tokenizer as the RoBERTa model), the amount of context from 1024 to 512, and the number of layers from 12 to 6. The third model is RoBERTa [30], which is based on Google's BERT language model [25]. This architecture consists of multiple transformer encoder layers. Unlike GPT-2, each token can attend to all other tokens in the input and the goal is not to predict the next token. Instead, a certain fraction of the input tokens are randomly converted to a special <mask> token, and the model is trained to predict the masked tokens. We use the huggingface implementation of RoBERTa with default parameter settings, except that we reduce the number of layers from 12 to 6.

[3] In contrast, the best performing music object detectors take 40-80 seconds to process each page at inference time using a GPU [31].

[4] This is because, in the original machine translation task [35], the decoder generates the output sentence autoregressively.

2.3 Classifier Finetuning

In the second main stage, we finetune a classifier based on a set of labeled data. The labeled data consists of a set of sheet music images along with their corresponding composer labels. The process of training the classifier is comprised of four steps (lower half of Figure 1).

The first two steps are to compute and tokenize a bootleg score into a sequence of symbolic words. We use the same fixed feature extractor and the same tokenizer that were used in the language model pretraining stage.

The third step is to sample short, fixed-length fragments of words from the labeled data. As mentioned in Section 2.1, we define a proxy task where the goal is to predict the composer given a short, fixed-length fragment of words. Defining the proxy task in this way has three significant benefits: (1) we can use sampling to generate many more unique training data points than there are actual pages of sheet music in our dataset, (2) we can sample the data in such a way that the classes are balanced, which avoids problems during training, and (3) using fixed-length inputs allows us to train more efficiently in batches. Our approach follows the general recommendations of a recent study on best practices for training a classifier with imbalanced data [36]. Each sampled fragment and its corresponding composer label constitute a single (X_i, y_i) training pair for the proxy task.

The fourth step is to train the classifier model. The bottom half of Figure 3 shows how this is done with our three models. Our general approach is to add a classifier head on top of the language model, initialize the weights of the classifier with the pretrained language model weights, and then finetune the classifier on the proxy task data. For the AWD-LSTM, we take the outputs from the last LSTM layer and construct a fixed-size representation by concatenating three things: (a) the output at the last time step, (b) the result of max pooling the outputs across the sequence dimension, and (c) the result of average pooling the outputs across the sequence dimension. This fixed-size representation (which is three times the hidden dimension size) is then fed into the classifier head, which consists of two dense layers with batch normalization and dropout. For the GPT-2 model, we take the output from the last transformer layer at the last time step, and then feed it into a single dense (classification) layer. Because the GPT-2 and RoBERTa models require special tokens during training, we insert special symbols <s> and </s> at the beginning and end of every training input, respectively. Because of the masked self-attention, we must use the output of the last token in order to access all of the information in the input sequence. For the RoBERTa model, we take the output from the last transformer layer corresponding to the <s> token, and feed it into a single dense (classification) layer. The <s> takes the place of the special [CLS] token described in the original paper.

We integrated all models into the fastai framework and finetuned the classifier in the following manner. We first select an appropriate learning rate using a range test, in which we sweep the learning rate across a wide range of values and observe the impact on training loss. We initially freeze all parameters in the model except for the untrained classification head, and we gradually unfreeze more and more layers in the model as the training converges. To avoid overly aggressive changes to the pretrained language model weights, we use discriminative finetuning, in which earlier layers of the model use exponentially smaller learning rates compared to later layers in the model. All training is done with (multiple cycles of) the one cycle training policy [37], in which learning rate and momentum are varied cyclically over each cycle. The above practices were proposed in [22] and found to be effective in finetuning language models for text classification.

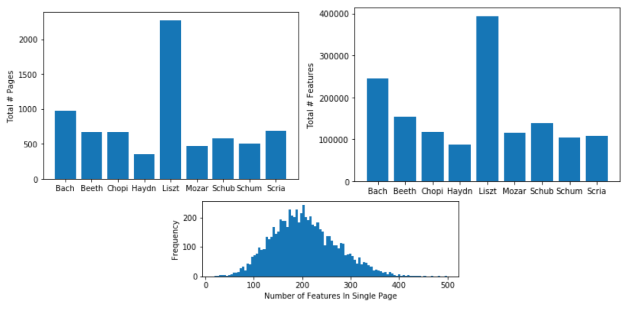

Figure 4. Statistics on the target dataset. The top two histograms show the distribution of the number of pages (top left) and number of bootleg score features (top right) per composer. The bottom figure shows the distribution of the number of bootleg score features per page.

2.4 Inference

The third main stage is to apply the proxy classifier to the original full page classification task. We explore two different ways to do this. The first method is to convert the sheet music image into a bootleg score, tokenize the bootleg score into a sequence of word or subword units, and then apply the proxy classifier to a single variable-length input. Note that all of the models can handle variable-length inputs up to a maximum context length. The second method is identical to the first, except that it averages the predictions from multiple fixed-length crops taken from the input sequence. The fixed-length crops are the same size as is used during classifier training, and the crops are sampled uniformly with 50% overlap. [5]

3. EXPERIMENTAL SETUP

In this section we describe the data collection process and the metrics used to evaluate our approach.

The data comes from IMSLP. We first scraped the website and downloaded all PDF scores and accompanying metadata. [6] We filtered the data based on its instrumentation in order to identify a list of solo piano scores. We

[5] We also experimented with applying a Bayesian prior to the classifier softmax outputs, as recommended in [36], but found that the results were not consistently better.

[6] We downloaded the data over a span of several weeks in May of 2018.

then computed bootleg score features for all of the piano sheet music images using the XSEDE supercomputing infrastructure [38], and discarded any pages that had less than a minimum threshold of features. This latter step is designed to remove non-music pages such as the title page, foreword, or table of contents. The resulting set of data contained $29,310$ PDFs, [7] $255,539$ pages and a total of 48.5 million bootleg score features. This set of data is what we refer to as the IMSLP dataset in this work (e.g. the IMSLP pretrained language model). For language model training, we split the IMSLP data by piece, using 90% for training and 10% for validation.

The classification task uses a subset of the IMSLP data. We first identified a list of composers with a significant amount of data (composers shown in Figure 4). We limited the list to nine composers in order to avoid extreme class imbalance. Because popular pieces tend to have many sheet music versions in the dataset, we select one version per piece in order to avoid over-representation of a small subset of pieces. Next, we manually labeled and discarded all filler pages, and then computed bootleg score features on the remaining sheet music images. This cleaned dataset is what we refer to as the target data in this work (e.g. the target pretrained language model). Figure 4 shows the total number of pages and bootleg score features per composer for the target dataset, along with the distribution of the number of bootleg score features per page. For training and testing, we split the data by piece, using 60% of the pieces for training (4347 pages), 20% for validation (1500 pages), and 20% for testing (1304 pages). To generate data for the proxy task, we randomly sampled fixed-length fragments from the target data. We sample the same number of fragments for each composer to ensure class balance. We experimented with fragment sizes of 64/128/256 and sampled 32400/16200/8100 fragments for training and 10800/5400/2700 fragments for validation/test, respectively. This sampling scheme ensures the same data coverage regardless of fragment length. Note that the classification data is carefully curated, while the IMSLP data requires minimal processing.

We use two different metrics to evaluate our systems. For the proxy task, accuracy is an appropriate metric since the data is balanced. For the full page classification task – which has imbalanced data – we report results in macro F1 score. Macro F1 is a generalization of F1 score to a multi-class setting, in which each class is treated as a one-versus-all binary classification task and the F1 scores from all classes are averaged.

4. RESULTS & ANALYSIS

In this section we present our experimental results and conduct various analyses to answer key questions of interest. While the proxy task is an artificially created task, it provides a more reliable indicator of classifier performance than the full page classification. This is because the test set of the full page classification task is both imbalanced and

[7] Note that a PDF may contain multiple pieces (e.g. the complete set of Chopin etudes).

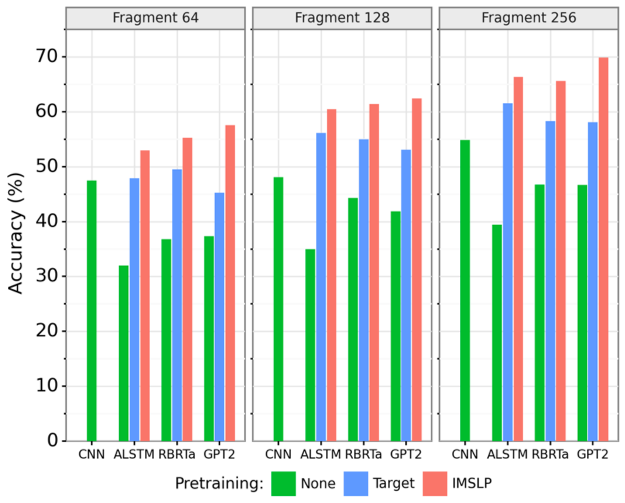

Figure 5. Model performance on the proxy classification task. This comparison shows the effect of different pretraining conditions and fragment sizes.

very small (1304 data points). Accordingly, we will report results on both the proxy task and full page classification task.

4.1 Proxy Task

We first consider the performance of our models on the proxy classification task. We would like to understand the effect of (a) model architecture, (b) pretraining condition, and (c) fragment size.

We evaluate four different model architectures. In addition to the AWD-LSTM, GPT-2, and RoBERTa models previously described, we also measure the performance of a CNN-based approach recently proposed in [7]. Note that we cannot use the exact same model in [7] since we do not have symbolic score information. Nonetheless, we can use the same general approach of computing local features, aggregating feature statistics across time, and applying a linear classifier. The design of our 2-layer CNN model roughly matches the architecture proposed in [7].

We consider three different language model pretraining conditions. The first condition is with no pretraining, where we train the classifier from scratch only on the proxy task. The second condition is with target language model pretraining, where we first train a language model on the target data, and then finetune the classifier on the proxy task. The third condition is with IMSLP language model pretraining. Here, we train a language model on the full IMSLP dataset, finetune the language model on the target data, and then finetune the classifier on the proxy task.

Figure 5 shows the performance of all models on the proxy task. There are three things to notice. First, regarding (a), the transformer-based models generally outperform the LSTM and CNN models. Second, regarding (b), language model pretraining improves performance significantly across the board. Regardless of architecture, we see a large improvement going from no pretraining (condition 1) to target pretraining (condition 2), and another

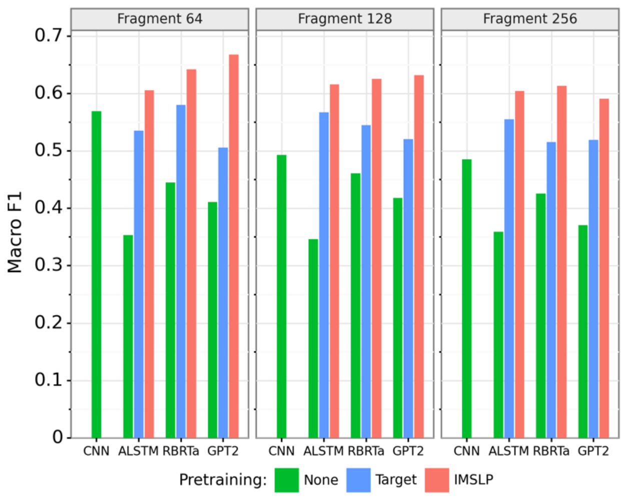

Figure 6. Results on the full page classification task.

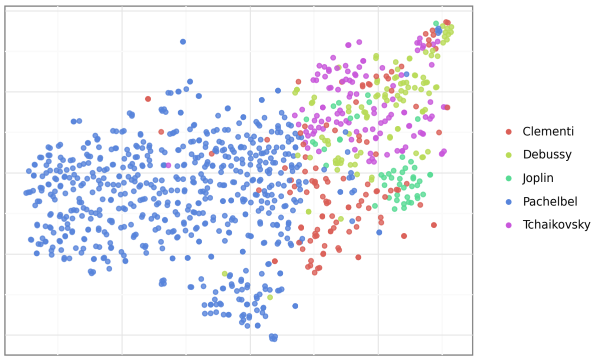

Figure 7. t-SNE plot of the RoBERTa model activations for five novel composers. Each data point corresponds to a single page of sheet music for a composer that was *not* considered in the classification task.

large improvement going from target pretraining (condition 2) to IMSLP pretraining (condition 3). For example, the performance of the GPT-2 model increases from 37.3% to 45.2% to 57.5% across the three pretraining conditions. Because the data in conditions 1 & 2 is exactly the same, the improvement in performance must be coming from more effective use of the data. We can interpret this from an information theory perspective by noting that the classification task provides the model $log_2 9 = 3.17$ bits of information per fragment, whereas the language modeling task provides $log_2 V$ bits of information *per bootleg score feature* where V is the vocabulary size. The performance gap between condition 2 and condition 3 can also be interpreted as the result of providing more information to the model, but here the information is coming from having additional data. Third, regarding (c), larger fragments result in better performance, as we might expect.

4.2 Full Page Classification

Next, we consider performance of our models on the full page classification task. We would like to understand the effect of (a) model architecture, (b) pretraining condition, (c) fragment size, and (d) inference type (single vs. multi-crop). Regarding (d), we found that taking multiple crops improved results with all models except the CNN. This suggests that this type of test time augmentation does not benefit approaches that simply average feature statistics over time. In the results presented below, we only show the optimal inference type for each model architecture (i.e. CNN with single crop, all others with multi-crop).

Figure 6 shows model performance on the full page classification task. There are two things to notice. First, we see the same general trends as in Figure 5 for model architecture and pretraining condition: the transformer-based models generally outperform the CNN and LSTM models, and pretraining helps substantially in every case. The macro F1 score of our best model (GPT-2 with fragment size 64) increases from 0.41 to 0.51 to 0.67 across the three pretraining conditions. Second, we see the *opposite* trend as the proxy task for fragment size: smaller fragments have *better* page classification performance. This strongly indicates a data distribution mismatch. Indeed, when we look at the distribution of the number of bootleg score features in a single page (Figure 4), we see that a significant fraction of pages have less than 256 features. Because we only sample fragments that contain a complete set of 256 words, our proxy task data is biased towards longer inputs. This leads to poor performance when the classifier is faced with short inputs, which are never seen in training. Using a fragment size of 64 minimizes this bias.

4.3 t-SNE Plots

Another key question of interest is, “Can we use our model to characterize the style of any page of piano sheet music?" The classification task forces the model to project the sheet music into a feature space where the compositional style of the nine composers can be differentiated. We hypothesize that this feature space might be useful in characterizing the style of *any* page of piano sheet music, even from composers not in the classification task.

To test this hypothesis, we fed data from 5 novel composers into our models and constructed t-SNE plots [39] of the activations at the second-to-last layer. Figure 7 shows such a plot for the RoBERTa model. Each data point corresponds to a single page of sheet music from a novel composer. Even though we have not trained the classifier on these five composers, we can see that the data points are still clustered, suggesting that the feature space can describe the style of new composers in a useful manner.

5. CONCLUSION

We propose a method for predicting the composer of a single page of piano sheet music. Our method first converts the raw sheet music image into a sequence of musical words based on the bootleg score feature representation, and then feeds the sequence into a text classifier. We show that by pretraining a language model on a large set of unlabeled data, it is possible to significantly improve the performance of the classifier.

6. ACKNOWLEDGMENTS

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. Large-scale computations on IMSLP data were performed with XSEDE Bridges at the Pittsburgh Supercomputing Center through allocation TG-IRI190019. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the GPU used for training the models.

7. REFERENCES

[1] K. C. Kempfert and S. W. Wong, "Where does haydn end and mozart begin? composer classification of string quartets," *arXiv preprint arXiv:1809.05075*, 2018.

[2] P. Sadeghian, C. Wilson, S. Goeddel, and A. Olmsted, "Classification of music by composer using fuzzy min-max neural networks," in *Proc. of the 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2017, pp. 189–192.

[3] A. Brinkman, D. Shanahan, and C. Sapp, "Musical stylometry, machine learning and attribution studies: A semi-supervised approach to the works of josquin," in *Proc. of the Biennial Int. Conf. on Music Perception and Cognition*, 2016, pp. 91–97.

[4] D. Herremans, D. Martens, and K. Sörensen, "Composer classification models for music-theory building," in *Computational Music Analysis*, 2016, pp. 369–392.

[5] G. Velarde, C. C. Chacón, D. Meredith, T. Weyde, and M. Grachten, "Convolution-based classification of audio and symbolic representations of music," *Journal of New Music Research*, vol. 47, no. 3, pp. 191–205, 2018.

[6] G. Velarde, T. Weyde, C. E. C. Chacón, D. Meredith, and M. Grachten, "Composer recognition based on 2d-filtered piano-rolls," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2016, pp. 115–121.

[7] H. Verma and J. Thickstun, "Convolutional composer classification," in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2019, pp. 549–556.

[8] G. Buzzanca, "A supervised learning approach to musical style recognition," in *Proc. of the international conference on music and artificial intelligence (ICMAI)*, vol. 2002, 2002, p. 167.

[9] J. Wołkowicz and V. Kešelj, "Evaluation of N-gram-based classification approaches on classical music corpora," in *International Conference on Mathematics and Computation in Music*, 2013, pp. 213–225.

[10] M. Hontanilla, C. Pérez-Sancho, and J. M. Inesta, "Modeling musical style with language models for composer recognition," in *Iberian Conference on Pattern Recognition and Image Analysis*, 2013, pp. 740–748.

[11] R. Hillewaere, B. Manderick, and D. Conklin, "String quartet classification with monophonic models." in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2010, pp. 537–542.

[12] M. A. Kaliakatsos-Papakostas, M. G. Epitropakis, and M. N. Vrahatis, "Weighted markov chain model for musical composer identification," in *European Conference on the Applications of Evolutionary Computation*, 2011, pp. 334–343.

[13] E. Pollastri and G. Simoncelli, "Classification of melodies by composer with hidden markov models," in *Proc. of the First International Conference on WEB Delivering of Music*, 2001, pp. 88–95.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[16] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proc. of the conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[17] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F.-F. Li, "ImageNet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[18] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[21] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[22] J. Howard and S. Ruder, “Universal language model fine-tuning for text classification,” in *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018, pp. 328–339.

[23] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” *OpenAI Blog*, 2018.

[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[25] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[26] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. Le, and R. Salakhutdinov, “Transformer-XL: Attentive language models beyond a fixed-length context,” in *Proc. of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 2978–2988.

[27] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, “XLNet: Generalized autoregressive pretraining for language understanding,” in *Advances in neural information processing systems*, 2019, pp. 5754–5764.

[28] D. Yang, T. Tanprasert, T. Jenrungrot, M. Shan, and T. Tsai, “MIDI passage retrieval using cell phone pictures of sheet music,” in *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2019, pp. 916–923.

[29] S. Merity, N. S. Keskar, and R. Socher, “Regularizing and optimizing LSTM language models,” *arXiv preprint arXiv:1708.02182*, 2017.

[30] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, “RoBERTa: A robustly optimized BERT pretraining approach,” *arXiv preprint arXiv:1907.11692*, 2019.

[31] A. Pacha, J. Hajič, and J. Calvo-Zaragoza, “A baseline for general music object detection with deep learning,” *Applied Sciences*, vol. 8, no. 9, p. 1488, 2018.

[32] P. Gage, “A new algorithm for data compression,” *C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.

[33] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” in *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016, pp. 1715–1725.

[34] D. Gillick, C. Brunk, O. Vinyals, and A. Subramanya, “Multilingual language processing from bytes,” in *Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 1296–1306.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[36] M. Buda, A. Maki, and M. A. Mazurowski, “A systematic study of the class imbalance problem in convolutional neural networks,” *Neural Networks*, vol. 106, pp. 249–259, 2018.

[37] L. N. Smith, “A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay,” *arXiv preprint arXiv:1803.09820*, 2018.

[38] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr, “XSEDE: Accelerating scientific discovery,” *Computing in Science & Engineering*, vol. 16, no. 5, pp. 62–74, Sept.-Oct. 2014. [Online]. Available: doi.ieeecomputersociety.org/10.1109/MCSE.2014.80

[39] L. v. d. Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.