EVALUATION OF PARALLEL AND SEQUENTIAL DEEP LEARNING MODELS FOR MUSIC SUBGENRE CLASSIFICATION

MIRIA FENG
School of Engineering
Stanford University
Stanford, CA 94305-6106, USA

WENYING FENG
Departments of Computer Science and Mathematics
Trent University
Peterborough, ON K9L 0G2, Canada

(Communicated by Ding-Xuan Zhou)

Abstract. In this paper, we evaluate two deep learning models which integrate convolutional and recurrent neural networks. We implement both sequential and parallel architectures for fine-grain musical subgenre classification. Due to the exceptionally low signal to noise ratio (SNR) of our low level mel-spectrogram dataset, more sensitive yet robust learning models are required to generate meaningful results. We investigate the effects of three commonly applied optimizers, dropout, batch regularization, and sensitivity to varying initialization distributions. The results demonstrate that the sequential model specifically requires the RMSprop optimizer, while the parallel model implemented with the Adam optimizer yielded encouraging and stable results achieving an average F1 score of 0.63. When all factors are considered, the optimized hybrid parallel model outperformed the sequential in classification accuracy and system stability.

1. Introduction. Music is an integral part of our daily lives, and indeed the universal language. Companies such as Spotify, Apple Music, and SoundCloud rely on precise music recommendations in order to better cater to each individual’s listening tastes and provide personalized listening features. Music intelligence companies such as Every Noise and Echo Nest have collected and analyzed over two million music artists and over 35 million songs. This is a first step towards building truly comprehensive world genre maps which allow users to explore areas of music specifically tailored to their personal preferences. Other audio research groups, such as Harvard Music Lab, aim to gain deeper understanding of how precisely users engage with music on a cognitive basis. These musical intelligence platforms rely on machine learning algorithms to correctly classify numerous new songs being added.

2020 Mathematics Subject Classification. Primary: 68T07; Secondary: 68T10.

Key words and phrases. Convolutional neural network, deep learning, GRU, LSTM, music subgenre, optimizer.

The second author is supported by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).
on a daily basis. Definitive classification and analysis of audio data will inevitably affect the music experience of over 10 million people every month.

In the pioneering paper by Tzanetakis and Cook [18], an automatic classification framework of audio signals for ten music genres was proposed using machine learning techniques. Their results demonstrated a classification accuracy of 61%, which is comparable to results reported for human musical genre classification at that time. Following [18], popular learning algorithms such as support vector machines (SVM) and k-means have been applied to music information retrieval (MIR) problems [7, 20]. For instance, using the SVM algorithm, a classification accuracy of 90% is achieved on a dataset containing four largely different musical genres [20]. More recently, unsupervised learning such as convolutional deep belief networks are also shown to be efficient in audio data recognition [12, 19]. Additionally, the sparse representation-based classifiers developed in [15] reported 91% and 93.56% accuracy on two broad music genre datasets respectively.

Convolutional neural networks (CNN) were originally introduced for visual pattern recognition [1, 11], but have proven to also be effective in music classification. Recent studies have shown that CNN models are able to outperform other techniques such as SVM, the k-nearest neighbors, feed-forward neural networks, and the long short-term memory (LSTM) on MIR applications [4, 5, 17]. Therefore hybrid models which utilize combinations of CNN and recurrent neural networks (RNN) have been proposed for improved performance. In [14], a sequential architecture of CNN and RNN is implemented in order to classify music genres based on differing artists. Since human level error for music artist classification is relatively low, this model was able to produce a good level of classification accuracy as expected. On the other side, the parallel recurrent convolutional neural network (PRCNN) introduced most recently in [21] is able to combine feature extraction and time-series data classification at one stage to ultimately optimize classification accuracy.

In contrast to the successful development of music genre classifications by deep learning [4, 14, 20], there has been limited work noticed in the area of music subgenre classifications. Each musical subgenre is a precise qualitative way of grouping together low level musical features which share a large number of similarities at a higher level view. Music recommendations based on subgenres instead of broad genres have much better resonance with listener tastes, and therefore produce more favourable personalized listening features for users. However, the challenge of fine-grain music subgenre classification lies in the fact that despite differences in low level audio features, much of the data is still extremely similar, resulting in exceptionally low signal to noise ratio (SNR). Therefore a more sensitive yet robust machine learning model is needed.

In this paper, our classification objects lie specifically in the context of fine-grain subgenre classification. In particular, we evaluate the performance of two convolutional and recurrent deep learning architectures (sequential and parallel) for four subgenres of heavy metal music classification. The sequential model is a modification of the architecture proposed in [2, 14], where we have replaced the initial gated recurrent units (GRU) with a more powerful LSTM unit to better capture temporal correlations from music samples. In comparison to the numerous sequential models for deep learning tasks, our proposed parallel model is fairly novel. We adopt the parallel design from the most recent work of [21], which utilizes a hybrid architecture of CNN and RNN blocks. Our new implementation of the parallel model is designed with consideration of the low SNR data, in order to achieve optimal
performance on our specific application. To evaluate the performance of these two dramatically different learning structures, we examine the metrics of three learning optimizers, four initialization distributions, and three regularization methods. Our original dataset is created by downloading 657 songs from four subgenres of metal music, selected for having exceptionally low SNR. To our knowledge, this study is probably the first to evaluate metal music subgenre classification with a focus on the effects of some lower level mathematical aspects of the learning algorithms. Our results show that the more uncommon parallel model was able to outperform the sequential, in both sensitivity and robustness.

The rest of the paper is organized as follows. In section 2, we introduce the design and architecture of the two learning models. Section 3 discusses the optimization functions and evaluation metrics. Next, dataset and experiments are presented in section 4, and finally section 5 summarizes some conclusions, limitations and future work.

2. Sequential and parallel deep learning architecture. We start with a baseline CNN model that is implemented with two-dimensional convolution and the most commonly applied rectified linear activation function (ReLU) [1]. The architecture is composed of three convolutional layers (Figure 1). A key consideration of the design is the kernel size. In general, a kernel of size $r \times c$ means the kernel can learn features of size $r \times c$ (where $r$ and $c$ represent rows and columns of kernels respectively). Smaller kernels reduce computational cost, but see only a small area of the input and therefore risk being unable to recognize inherent data patterns since the features extracted will be too local. This results in little to no information being captured from neighboring input patches. Some previous research such as Hinton et al. [10] proposed large constitutional kernels of size 11$\times$11$\times$3 to detect patterns. However, larger kernels dramatically increase the number of hyperparameters and therefore consume much longer time in training. Even kernels are typically not preferred since they are not symmetrical around a centre point. This lack of symmetry creates the risk of distortions across neural network layers. After experimentation, we choose a 3$\times$3 kernel size for layer one, then increase to 3$\times$5 for higher layers to allow the model to learn nonlinear features across larger regions of the state space. The layers are designed with 8, 32, and 64 filters respectively. Since some data is lost during each convolutional layer, valid padding (or same padding) is applied with maxpooling layers of size 2$\times$4 to accumulate features from maps generated by convolving layers. This design choice serves to reduce the computation cost, while slightly mitigating the risk of overfitting by providing an abstracted form of the data representation. The output is then fed into a final dense layer with softmax activation for classification [4].

Recurrent neural networks (RNN) are significantly different from CNN, and instead scan through each input sequence from left to right with internal memory cells to remember the beginning of each data sequence. This ensures that even later in the input, the beginning of the data is still taken into account. Intuitively this is desirable when analyzing audio data since we want to capture patterns not only from one instant of a song, but also from all previous time-steps of the same song. Therefore, integrating RNN with CNN for the task of music classification becomes a natural design choice. We select two commonly utilized RNN units: the GRU [3] and LSTM [8] to be integrated with the parallel and sequential models respectively. Although both units are able to learn long range connections and dependencies in a
sequence, the GRU is simpler since it only incorporates two gates, while the more complex LSTM involves three gates and is thus able to capture more expressiveness under certain circumstances [8]. We take into consideration that the simpler GRU is computationally faster with greater ease when building a bigger model, as well as the inherent added complexity of the parallel architecture, and select to integrate the GRU with the parallel model. In contrast, the LSTM is selected for integration with the simpler sequential model in order to balance the computational complexity and learning flexibility of the whole system.

The sequential architecture is an adaption of the learning models presented in [2, 14]. In contrast to the datasets used in these studies for music artist classification, our low SNR musical subgenre dataset benefits from the replacement LSTM unit in order to better capture the nuances in audio features between each time step of the input. As shown in Figure 2, the CRNN architecture uses 4 layers of CNN with maxpooling before being passed into the LSTM then a final dense layer with softmax activation for classification. In the CNN substructure, the sizes of the convolutional layers and maxpooling layers are $3 \times 3$ and $(2 \times 2) - (3 \times 3) - (3 \times 5) - (3 \times 5)$ respectively. The ultimate underlying assumption is that our data patterns can be better aggregated with RNN than CNN.

The configuration of the parallel CNN-RNN model is inspired by the recent work of [21]. The hybrid architecture consists of parallel CNN and bidirectional GRU (Bi-RNN) blocks, whose outputs are then fused into one representation and fed into the softmax function for classification. The CNN block of this architecture is composed of five convolutional layers with maxpooling. Each kernel size is defined to be $3 \times 1$ with padding. In order to achieve the most meaningful representations, the five convolution layers have filters of size 16, 32, 64, 128, and 64 respectively. Figure 3 shows the overall design. The design implementation is intended to extract
both temporal features from the input data which may be lost in the CNNs, and representations of past and future information of sequences. The Bi-RNN block consists of first processing the input through a 2D maxpooling layer and an embedding layer for dimension reduction to decrease parameters. Finally, the outputs of the two parallel branches are fused into one single representation to be fed into a softmax layer for subgenre classification.

3. **Evaluation metrics.** A core consideration for the design of learning models is the selection of the optimizers, since these functions ultimately define the iteration process for optimization. In our study, effects of the following three optimizers on each of the three models: the baseline CNN, sequential CRNN and parallel CNN-RNN are examined.

Let $\theta$ denote the set of parameters for the update process, $g_t$ represent the gradient at time step $t$, and $\varepsilon = 10^{-8}$ be the smoothing term to avoid division by zero. The Adadelta optimizer [22] applies a stochastic gradient optimization technique involving the root mean square (RMS) of the variables. We initialize $E[g_t^2]_0 = 0$, $E[\Delta \theta_t^2]_0 = 0$, and the update is defined as:

\begin{align*}
E[g_t^2]_t &= \gamma E[g_t^2]_{t-1} + (1-\gamma)g_t^2 \quad (1) \\
\Delta \theta_t &= -\frac{RMS[\Delta \theta_{t-1}]}{\sqrt{E[g_t^2]_t + \varepsilon}} g_t \quad (2) \\
E[\Delta \theta_t^2]_t &= \gamma E[\Delta \theta_t^2]_{t-1} + (1-\gamma)\Delta \theta_t^2 \quad (3) \\
\theta_{t+1} &= \theta_t + \Delta \theta_t. \quad (4)
\end{align*}

We choose $\gamma = 0.9$, and note that we do not need to set a default learning rate since it has been eliminated from the update rule.

Next, the RMSprop optimizer is similar to the gradient descent with momentum algorithm, yet different on the method of calculating the gradients. Its key advantage is faster convergence to solutions of better quality, particularly in regards to LSTM applications. In the following update process, the learning rate $\eta = 0.001$ is selected.
as a starting point:

\[ E[g^2_t] = 0.9E[g^2_{t-1}] + 0.1g^2_t \]

\[ \rightarrow \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2_t] + \varepsilon}}g_t. \]  

\[(5) \quad (6)\]

Lastly, the Adam [9] optimizer combines the benefits of RMSprop and stochastic gradient descent with momentum. It performs particularly well on noisy problems with sparse gradients. Let \( v_t \) denote past squared gradients, and \( m_t \) denotes past gradients. The update rule is as follows:

\[ \hat{m}_t = \frac{m_t}{1 - \beta_1^t} \]

\[ \rightarrow \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \]

\[ \rightarrow \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon}\hat{m}_t. \]  

\[(7) \quad (8) \quad (9)\]

We choose \( \beta_1 = 0.9, \beta_2 = 0.99 \) and an initial learning rate of \( \eta = 0.001 \).

The selections of the activation and loss functions are first tested using the baseline CNN model. We experiment with parameters via cross validation with the goal of finding optimal starting points. The ReLU activation function is selected for all models in order to achieve non-linear behaviour. In contrast to sigmoid activation, ReLU does not saturate at 1, and the partial derivative of the activation function is never 0. This successfully reduces the vanishing gradient problem that is observed in traditional neural networks. ReLU has also demonstrated a more rapid speed of convergence as opposed to sigmoid or tanh. The categorical crossentropy loss function (as opposed to sparse categorical crossentropy) is used since it is possible that one sample may be identified as multiple classes. Each of the classifier blocks features the softmax function for classification:

\[ P(i) = \frac{\exp(x_i)}{\sum_{k=1}^{k} \exp(x_k)}, \]  

\[(10)\]

where \( P(i) \) and \( x_i \) represent the probability of music subgenre and the \( i \)-th value of the feature vector respectively.

In addition to experimentation with the underline core mathematical optimization functions, we also examine the effects of four different initialization methods: Glorot normal, Glorot uniform, random normal, and random uniform; as well as the effects of regularization using dropout and batch normalization. Results and discussions are given in the next section.

4. Experiments and results.

4.1. Dataset and features. We create the dataset by using Spotify’s API and the python packages of Spotify and Spot-dl [16] to download 657 songs from four subgenres of metal music: ‘Alternative Metal’, ‘Death Metal’, ‘Thrash Metal’, and ‘Power Metal’. These four subgenres of metal music are selected due to their exceptionally low SNR, in order to demonstrate the effectiveness of our sensitive subgenre classification systems.

For audio data, we measure signal as the variance of air pressure over time. By sampling each song, we capture it as a signal waveform which can then be better interpreted and analyzed. Since the audio signal waveform is composed of
several single-frequency sound waves, our samples are only able to capture the resulting amplitudes. Further processing applies the fast fourier transform (FFT) on overlapping windowed segments of the raw signal waveform data to get a resulting spectrogram. Finally, the frequencies of the spectrogram are converted to the mel scale for uniformity, and thus achieve the mel-spectrogram dataset [13]. For visualization, we convert the y axis to a log scale and the colour dimension is converted to decibels (the log scale of the amplitude). Figure 4 shows the visualization of one input song, and the following summarizes the overall process of creating the dataset from downloaded mp3 files:

1. Convert into numpy signal arrays using the audio processing library Librosa.
2. Downsample to 12000Hz in order to reduce and size regularize data.
3. Four non-overlapping twenty second slices were sampled at random from the middle 90% of each song. This not only results in a more manageable feature size, but also augment the number of examples.
4. Convert samples to a mel-spectrogram using Librosa. We chose a hop length of 216 and a number of mels as 96 for consistency with existing work [2].
5. The two dimensional mel-spectrograms are stacked to create an input feature matrix of dimensions 2, 628 by 96 by 1, 112.

For each experiment, the dataset is randomly shuffled and split into train, validation, test sets of 60-20-20% respectively. The training batch size is selected to be 64 due to the size of the dataset.

4.2. Results and discussion. F1 score is the harmonic mean of precision and recall. Precision quantifies the number of positive classifications which truly belong to the positive class, while recall quantifies the number positive class predictions among all positive samples in the dataset. Let \( tp \) denote true positives, \( tn \) denote true negatives, \( fp \) denote false positives, and \( fn \) denote false negatives.

\[
\text{Precision} = \frac{tp}{tp + fp},
\]
\[
\text{Recall} = \frac{tp}{tp + fn},
\]
\[
\text{F1} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}},
\]
\[
\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}.
\]

Throughout this paper we will refer to the F1 score as the macro-averaged F1, which is obtained by taking an arithmetic mean of the F1 scores across the four classes.

Table 1 summarizes the relative performance using the macro-averaged F1 scores for the three optimizers (averaged across three passes). It is noticed that under
Table 1. F1 scores for optimizer evaluation

| Optimizer | CNN  | CRNN | CNN-RNN |
|-----------|------|------|---------|
| Adam      | 0.45 | 0.32 | 0.63    |
| Adadelta  | 0.30 | 0.31 | 0.35    |
| RMSprop   | 0.41 | 0.54 | 0.60    |

Table 2. Optimal classification accuracy

| Optimizer | CNN  | CRNN | CNN-RNN |
|-----------|------|------|---------|
| Adam      | 0.31 | 0.57 | 0.64    |
| RMSprop   |      |      |         |

any other optimizer (besides the RMSprop), the CRNN model seems to start with an initial value, then make the correct prediction, and does not deviate further from the same prediction. This is likely due to the low SNR in the dataset. The singular success of the RMSprop optimizer here is credited to its ability to resolve radically diminishing learning rates. This is particularly interesting since Adam [9] is broadly considered to be the more effective optimizer, yet the sequential CRNN model is specifically unable to make meaningful classifications under any optimizer besides RMSprop. Aiming to understand this behaviour, we consider the key differences between Adam and RMSprop. While both optimizers involve keeping an exponentially weighted average of the squares of the gradients, Adam has an additional element of momentum. This added momentum seems to prevent the sequential CRNN model from making reasonable classifications among data with low SNR. This is also validated by the fact that RMSprop has generally been shown to be more successful in training RNN. In contrast, the parallel CNN-RNN model is able to make reasonable classifications with both RMSprop and Adam optimizers, and converges at a steady pace. The CNN baseline model also performed similarly between the RMSprop and Adam optimizers, yet all three models performed poorly with the Adadelta optimizer [22]. This is potentially due to the moving averages of Adadelta being biased by initialization of decay parameters. Table 2 gives the optimal classification accuracy for each of the three models. The optimized parallel CNN-RNN model with the Adam optimizer outperformed the other two models in terms of accuracy and robustness. Unlike the CRNN, it is able to make meaningful classifications even without meticulous fine-tuning of hyperparameters, and ultimately surpasses the accuracy of the comparative models.

To further investigate the notable success of RMSprop optimizer on the CRNN model, a visualization of the optimization on two axis is shown in Figure 5. In reality we note that the process is on a high dimensional space of parameters. Equations (11)-(14) show the simplified parameter update for two dimensions. The goal is to slow down/damp out the oscillations in the vertical \( b \) direction while speeding up learning in the horizontal \( w \) direction for faster convergence. Therefore \( S_{db} \) should be relatively small (division by a smaller number results in a faster learning rate), while \( S_{dw} \) will be relatively large (division by a larger number to slow down oscillations). The net affect is that the updates in the vertical direction are damped out. Therefore it is possible to use a larger learning rate and achieve faster learning without divergence in the vertical direction under the RMSprop optimizer.
Next, we examine the effects of dropout and batch regularization on the CRNN and CNN-RNN architectures. We selected the Adam optimizer for the parallel model since it yielded the most consistent results, and the RMSprop optimizer for the sequential model. Batch normalization is designed at every layer after each activation function and before maxpooling. By normalizing hidden unit activations, it reduces covariant shift and therefore speeds up learning. It also forces later units in deeper layers to not rely too much on any single previous hidden unit by adding slight noise. Overall, batch normalization makes the learning in later layers easier.

To show the significant improvement on training set performance after adding batch normalization, we list detailed results for training, validation and testing in Table 3. Due to this sharp increase and simultaneous decrease in accuracy on testing and validation sets, we suspected overfitting. This is likely due to the fact that batch normalization may have worsened model generalization to out-of-domain data seen during training. In other words, the distribution of the test data may be different from the distribution of the training data. This is usually unexpected since batch normalization transforms data during training by scaling and shifting with the mean and variance of each data batch to reduce overfitting. However, since our dataset is composite of particularly low SNR data, it is understandable that a small shift in distribution might result in significant changes in performance. Ultimately a combination of batch normalization and dropout yielded the most stable results.

Experimenting with various dropout values led to the optimal dropout results for each of the models shown in Table 3, where the dropout is selected to be 0.2 and 0.6 for the parallel and sequential models respectively. Clearly, effects of dropout are less dramatic than batch normalization. By randomly setting some input values to zero with a frequency defined by the dropout rate, this technique implies each unit cannot rely on any one value overwhelmingly. Instead we spread out the weight among all inputs, therefore achieving the effect of shrinking the squared norm of the weights. Dropout can also be formally shown to be an adaptive form of $L^2$ regularization, thus preventing overfitting. Additionally, dropout is only applied during training so no values are dropped during inference. We can also see from Table 3 that when dropout is applied, the accuracy for both models improved. Careful tuning of these hyperparameters eventually allowed the CRNN model to achieve a similar level of performance as the CNN-RNN model. However, it is noted
Table 3. Marco F1 scores for effect of regularization

| Model   | Data       | Dropout | Batch Normalization | Dropout + Batch Normalization |
|---------|------------|---------|---------------------|------------------------------|
| CRNN    | Train      | 0.67    | 1.00                | 0.98                         |
|         | Validation | 0.65    | 0.58                | 0.60                         |
|         | Test       | 0.62    | 0.57                | 0.41                         |
| CNN-RNN | Train      | 0.65    | 1.00                | 0.90                         |
|         | Validation | 0.65    | 0.58                | 0.60                         |
|         | Test       | 0.63    | 0.61                | 0.63                         |

Figure 6. Classification accuracy across 50 epochs

that the CRNN architecture is particularly difficult to tune, and is unable to make meaningful classifications unless under extremely specific hyperparameter choices.

Figure 6 compares the learning process of the two models over 50 epochs. It clearly demonstrates the superiority of the parallel model in contrast to the sequential model. As training continues the CNN-RNN model is able to steadily increase performance with low variance, while the CRNN model requires careful hand tuning of hyperparameters to achieve similar performance, yet still exhibits high variance.

Lastly, we investigate the sensitivity of our models to varying initializations. We experiment with the most popular normal and uniform distributions, as well as random initialization options. The Glorot [6] technique is based on the theoretical analysis of finding a good variance for the distribution from which the initial parameters are drawn. This variance is adapted to the activation function and derived without explicitly considering the type of distribution. The results are given in Table 4. Of the three learning models, the baseline CNN model is most non-sensitive to variations in initialization. The sequential CRNN model performs better with the Glorot method, but performs poorly with random initialization (random uniform had the lowest score of 0.37). The parallel CNN-RNN model is moderately sensitive, and achieved its highest testing score 0.63 with the Glorot normal and lowest score 0.53 with the random normal. Overall, both the CRNN and the parallel CNN-RNN models performed optimally with Glorot normal initialization.

5. Conclusion and future work. In this paper, we evaluate three deep learning neural network architectures for fine-grain music subgenre classification. Our
generated dataset includes four subgenres of metal music with particularly low SNR in order to train the most sensitive yet robust classification models. Our experiments revealed that the CNN has limited expressiveness, and the sequential CRNN is only able to make meaningful classifications with the RMSprop optimizer and dropout=0.6. The parallel CNN-RNN model implemented with the Adam optimizer yielded encouraging and stable results with an average F1 score of 0.63. These results are noticeably lower than the classification accuracy of broad music genres (often more than 90%), which is to be expected since human level error is significantly higher on subgenre classification. For example, it is appreciably easier to differentiate between classical music and rock music, yet much more challenging for the average listener to differentiate between alternative metal music and power metal music. Therefore this study leads to promising directions for future research in subgenre classification, with solid potential to improve the preliminary results.

We also studied the effects of three different regularization techniques. Our results demonstrate the significance of batch normalization, which should be used with caution when mitigating overfitting. The implementation of dropout has less dramatic effect. In comparing the two models, the parallel CNN-RNN architecture is noticeably more robust, and able to perform meaningful classifications without meticulous fine-tuning of hyperparameters. Finally, the evaluation of four initialization methods indicate the baseline CNN model is almost unaffected by various initialization. In comparison, the parallel CNN-RNN is less sensitive while the CRNN strictly requires the Glorot initialization and fails under random initialization.

In distinguishing the overall performance of the CRNN and CNN-RNN models, the differences between GRU and LSTM are important factors. Both are able to preserve information via gates during long term dependencies, but in GRU the activations of gates only depend on previous output and current input. Thus, the simpler GRU mitigates the occurrence of overfitting, and converges faster than LSTM with less parameters. However with a larger dataset, LSTM results in much higher expressiveness and therefore will likely lead to better results. The limited size of our dataset is also a factor in the favourable performance with GRU as opposed to LSTM.

Future work will involve creating a larger dataset and explore leveraging a large amount of unlabeled data to generate learned features for more expressiveness. This will lead to the development of richer learning models, such as a hybrid parallel model with LSTM or transformers for more parallelization. The potential of including a combination of low and high level features (with a late stage fusion approach) in order to capture more complexity in audio data should also be considered. Our preliminary results encourage further study aiming to train deeper parallel neural network models to achieve Bayes error rates which will eventually be lower than human level error rates on subgenre music classification.
Acknowledgments. We would like to thank Aidan John Donohue for providing the dataset, and many thanks to Christopher Wolff, Andrew Ng, Chris Re and Tengyu Ma for support throughout and many insightful discussions. We thank the anonymous reviewers for their valuable comments and suggestions.

REFERENCES

[1] M. Browne and S. S. Ghidary, Convolutional neural networks for image processing: An application in robot vision, in AI 2003: Advances in Artificial Intelligence, Lecture Notes in Comput. Sci., 2903, Lecture Notes in Artificial Intelligence, Springer, Berlin, 2003, 641–652.

[2] K. Choi, G. Fazekas, M. Sandler and K. Cho, Convolutional recurrent neural networks for music classification, IEEE International Conference on Acoustics, Speech, and Signal Processing, New Orleans, LA, 2017.

[3] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, preprint, arXiv:1412.3555v1.

[4] Y. M. G. Costa, L. S. Oliveira and C. N. Silla Jr., An evaluation of Convolutional Neural Networks for music classification using spectrograms, Applied Soft Computing, 52 (2017), 28–38.

[5] G. Gessle and S. Åkesson, A Comparative Analysis of CNN and LSTM for Music Genre Classification, Degree Project in Technology, Stockholm, Sweden, 2019. Available from: https://www.diva-portal.org/smash/get/diva2:1354738/FULLTEXT01.pdf.

[6] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 9 (2010), 249–256. Available from: http://proceedings.mlr.press/v9/glorot10a.html.

[7] M. Helén and T. Virtanen, Separation of drums from polyphonic music using non-negative matrix factorization and support vector machine, 13th European Signal Processing Conference, Antalya, Turkey, (2005), 1–4. Available from: https://ieeexplore.ieee.org/document/7078147.

[8] S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural Computation, 9 (1997), 1735–1780.

[9] D. P. Kingma and J. Ba, ADAM: A method for stochastic optimization, preprint, arXiv:1412.6980.

[10] A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet classification with deep convolutional neural networks, 26th Conference on Neural Information Processing Systems, NeurIPS, 2012.

[11] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Computation, 1 (1989), 541–551.

[12] H. Lee, Y. Largman, P. Pham and A. Y. Ng, Unsupervised feature learning for audio classification using convolutional deep belief networks, 23rd Conference on Neural Information Processing Systems, NeurIPS, 2009. Available from: https://ai.stanford.edu/~ang/papers/nips09-AudioConvolutionalDBN.pdf.

[13] B. Logan, Mel frequency cepstral coefficients for music modeling, International Symposium on Music Information Retrieval, 2000.

[14] Z. Nasrullah and Y. Zhao, Music artist classification with convolutional recurrent neural networks, preprint, arXiv:1901.04555v2.

[15] Y. Panagakis, C. Kotropoulos and G. R. Arce, Music genre classification via sparse representations of auditory temporal modulations, 17th European Signal Processing Conference, Glasgow, UK, 2009.

[16] Python, Package Index: Spotify and Spot-dl. Available from: https://pypi.org/project/spotify/ and https://pypi.org/project/spotdl/.

[17] A. J. R. Simpson, G. Roma and M. D. Plumbley, Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network, preprint, arXiv:1504.04658.

[18] G. Tzanetakis and P. Cook, Musical genre classification of audio signals, IEEE Transactions on Speech and Audio Processing, 10 (2002), 293–302.

[19] J. Wülffing and M. A. Riedmiller, Unsupervised learning of local features for music classification, International Society for Music Information Retrieval Conference (ISMIR), 2012. Available from: http://ml.informatik.uni-freiburg.de/Former/_media/publications/wuelf2012.pdf.
[20] C. Xu, N. C. Maddage, X. Shao, F. Cao and Q. Tian, Musical genre classification using support vector machines, IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP ’03), Hong Kong, 2003.

[21] R. Yang, L. Feng, H. Wang, J. Yao and S. Luo, Parallel recurrent convolutional neural networks based music genre classification method for mobile devices, IEEE Access, 8 (2020), 19629–19637.

[22] M. D. Zeiler, ADADELTA: An adaptive learning rate method, preprint, arXiv:1212.5701.

Received April 2021; revised May 2021.

E-mail address: miria0@stanford.edu
E-mail address: ufeng@trentu.ca