Deep Learning for Auto-Generated Music Composition Project Report

Jianqiao Li
Tandon School of Engineering
New York University
jl7136@nyu.edu
https://github.com/superChoi7/DLFinalProject

Zhiying Cui
Tandon School of Engineering
New York University
zc2191@nyu.edu
https://github.com/superChoi7/DLFinalProject

I. Introduction

In recent decade, people have been trying to explore the potential role of artificial intelligence in music production and the capabilities of various machine learning models developed for music auto-generation have shown a promising future in fusing artistic creativity and computation power, suggesting practical applications in fields like film sound scoring, video games, and even short-video-content platforms. As the traditional composition rules transitioned to deep learning techniques based on probabilistic prediction, we expect the technology could aid in mass-producing requirement-oriented musical work that was once considered a human-specific work; while models with outstanding performance are nowadays prevalent, limitations such as style-awareness and theme consistency still yet to be elegantly resolved or sufficiently researched.

There are many rewarding and challenging topics in the field of music generation or music composition with deep learning. Briot et al.(Briot, Hadjeres, and Pachet 2017) uses five dimensions to characterize different ways of applying deep learning techniques to different topics of music generation. Figure 1 summarized features under each dimensions that we are interested. This paper dedicates to touching on possible solutions to one of the obstacles mentioned above: music style(genre) classification. We will build our model based on the DeepJ architecture proposed by Mao et al.(Mao, Shin, and Cottrell 2018) with modifications and specifically, this model aims to auto-generate a 30-second polyphony given a specific composing style and random formatted inputs. Polyphony is the abbreviation of single voice polyphony, a sequence of notes for a single instrument, where more than one note can be played at the same time. The goals of our work are detailed as following.

- Rebuild the DeepJ model on local environment, trained the model with modified input data, and compare the generated music with results from the original model with objective evaluation benchmarks.
- Explore possible genre-classification neural network models and replace the one-hot encoding strategy on style generation in the original model with selected genre-classification model.
- Due to the expensive computational costs with dense matrix manipulation, future research might include developing a sparse representation of data inputs as the various intellectuals previously recommend.

II. Previous Work

A comprehensive review by Hernandez-Olivan et al.(Hernandez-Olivan and Beltran 2021) systematically analyzed several state-of-the-art neural networks applied to music generation. Authors compared strengths and drawbacks of each model and concluded that the Natural Language Processing based models (NLP) and Generative Adversarial Networks(GAN) are proven to be the best alternatives for music auto-generation.

The idea of music auto-generation with machine learning was based on grammar rules and pattern recognition at early stages, and only until recent years the models with probabilistic prediction was brought to attention. Johnson (Johnson 2017) presented two versions of a probabilistic model of polyphony using a set of parallel, tied-weight recurrent networks. These two versions of the model, so called Tied-Parallel LSTM-NADE and Bi-Axial LSTM, haven shown a noticeably higher accuracy on a prediction task in experiments compared to a non-parallel model and successfully reproduces complex melodies, though still, without style persistence.

We decided to base our deep learning architecture foundation on an improved version of parallel model "DeepJ" proposed by Mao. Mao et al.(Mao, Shin, and Cottrell 2018) proposed an end-to-end generative model that is able to compose music based on a specific mixture of composer styles. They build the model DeepJ upon the Bi-Axial LSTM architecture and enforced musical style by adding volume and embedding styles with input data parsed through a 1-D convolution layer. The paper conducted experiments with three selected genres of music styles to evaluate the performance of their model compared to Bi-Axial LSTM by hard-coded classification. The result shows that the DeepJ outperforms Bi-Axial LSTM in overall subjective and objective evaluation by nontrivial margin.

III. Methodology

As we are yet at the stage of error-trailing the effects of input data format on model's performance, we will start our

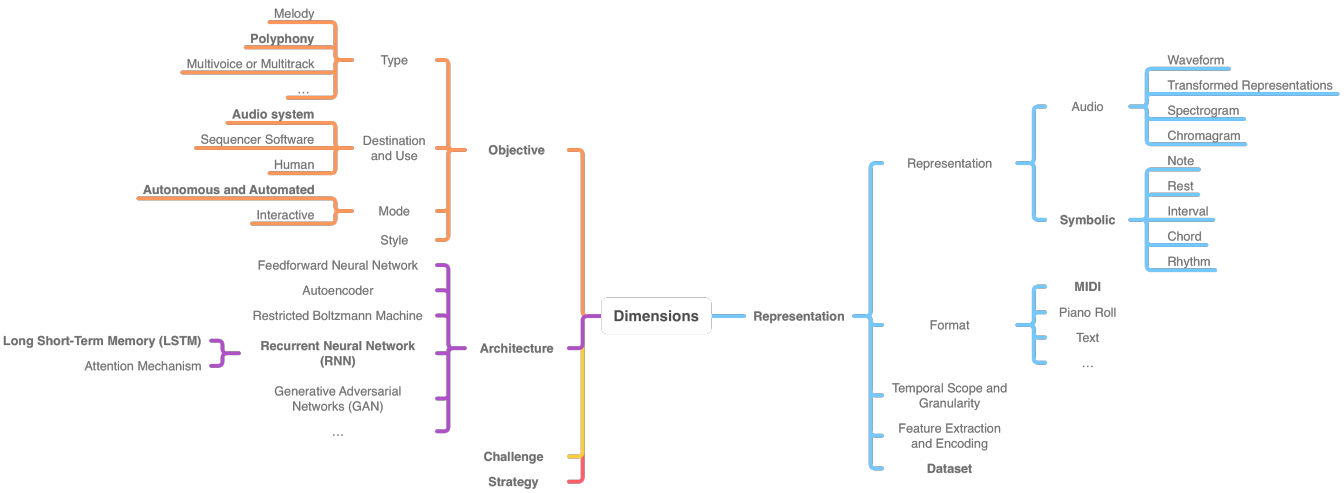

Figure 1: Objective, Representation, Architecture, Challenge and Strategy

project by reconstructing the model architecture designed by Mao et al.(Mao, Shin, and Cottrell 2018) with necessary modifications to accommodate future incorporation of style-classification modules.

A. Data Representation

1) Dataset: We use Musical Instrument Digital Interface(MIDI) files as the training dataset. There are many MIDI libraries online such as piano-e-competition, MuseData, mfiles, Piano-midi[1], etc; and We select the same MIDI library Piano-midi as that trained the DeepJ for model consistency. Piano-midi contains the dataset of classical piano solo pieces composed by different composers throughout different historical stages. The pieces from each composer are recorded by using a MIDI sequencer, with total of 571 pieces written by 26 composers and a total duration of 36.7 hours of MIDI files in this dataset as recorded till Feb. 2020(Kong et al. 2020).

MIDI files represent two important features: "Note-on", which refers to a note is played, and "Note-off", which refers to a note ends. MIDI also contains three attributes: Channel number, which indicates the instrument or track; Note number, which indicates the note pitch, integer values ranged from 0 to 127; Velocity, which indicates how loud the note is played in Note-on, and indicates how fast a note is released in Note-off. In this work, we only consider notes and velocity two attributes of the MIDI and assume channel number is always 1(piano), since we will only work on the pieces recorded in piano. To more visually sense the MIDI file, Figure 2 gives an example of the piece of Prelude and Fugue composed by Johann Sebastian Bach.

2) Representation: In the original Bi-Axial LSTM model proposed by Johnson (Johnson 2017), Johnson adopted the piano roll representation of the notes. It is a binary one-dimensional vector where 1 represents a note being played and 0 otherwise with each adjacent number representing a

[1]http://www.piano-midi.de/

Figure 2: MIDI of Prelude and Fugue in C major BWV 846

semitone (half-step) increase. The reason we decide to reuse the dataset format of MIDI file is in consistence with the concern addressed in Mao et al.(Mao, Shin, and Cottrell 2018): since the piano roll representation is a 1-D vector, it fails to capture the music dynamics (aka velocity), while the Channel component in MIDI files can in some way make up for the loss in details. We also represent the playable notes t_{play} by a $N \times T$ matrix where N is the numbers of notes, T is the number of time steps, values in the matrix is the normalized dynamic scale ranges from 0 to 1, which originally scales between 0 and 127.

$$t_{play} = \begin{bmatrix} 0 & 0 & 0.5 & \dots \\ 0.4 & 0 & 0 & \\ 0 & 0 & 0.2 & \\ \dots & & & \dots \end{bmatrix}_{N \times T} \quad (1)$$

As mentioned above, notes in a standard MIDI file ranges from 0 to 127, but we decide to truncate the original scale to 36-84 (4 octaves) for dimensional reduction in considerations of computational expensiveness. The intuition behind this operation comes from realizing that most musical pieces rarely use pitches outside this range, though any further re-

search or applications could adopt full note scale.

Apart from the fields required above, In LSTM model we also need contextual input to maintain a better long term structure consistency. Same as the the additional field propose with DeepJ, We use beat position in the current bar.

As for the component of musical style, we adopt the original encoding techniques proposed in the paper at the first stage of rebuilding DeepJ: we use one-hot encoding representation over all artists since different composers have their particular artistic style. During the training process, we encode "mixed styles" through equal-weight distribution, for example, we represent the mixed style of "Classical" pieces across all musicians in this period, shown below as $S_{classical}$. Here we made one modification: in light of the limitation of our computation power, we decide to eliminate the "Baroque" genre and thus reduce the total number of musicians to 6, along with the change of the length of style vector. What's worth noticing is that the original model only generates results with general styles in specific historical periods instead of certain musicians, thus making classification down to musicians itself pointless. As a result, we will incorporate certain changes into the later stage of this project to streamline this inconsistency, for example, using batches of different musical styles with distinctive artistic features to augment differentiability.

$$\begin{aligned} s_1 &= [1,0,0,0,0,0] \\ s_2 &= [0,1,0,0,0,0] \\ s_3 &= [0,0,1,0,0,0] \\ s_{classical} &= [0.33,0.33,0.33,0,0,0] \end{aligned} \tag{2}$$

Meanwhile, Mao et al.(Mao, Shin, and Cottrell 2018) mentioned that the dense data representation they used for DeepJ is quite computationally expensive and we did encounter similar difficulties in our experiment. As authors recommended, we aim to find a rather a spare representation such as the one used in Performance RNN(Simon and Oore 2017), which is closer to the nature of the MIDI file. Thus, we plan to change the input representation after successfully reconstructing the DeepJ with the original input format.

3) Data Pre-Processing: Before we can feed our model with training data, we still need an extra step of properly parsing our input dataset into correct format, and here are several important utility functions listed below.

```
#Load all MIDI files and parse them into
    four inputs, that are note_data,
    note_target, beat_data, style_data, and
    one label note_target.
load_all(styles, batch_size, time_steps)
#Clamp the MIDI based on the MIN and MAX
    notes. In the paper, the authors
    truncates a standard pitch to range
    from 36 to 84 to reduce input dimension.
clamp_midi(sequence)
#Chop the sequence data by time_steps. This
    function returns two variables: dataX
    the sequence of data in the current
    time step, and dataY the sequence of
    data in the next time step which is the
    predicted target.
stagger(data, time_steps)
```

B. Architecture

The DeepJ architecture is illustrated in Figure 3. This architecture based on Bi-Axial LSTM models each note within certain time step as a probability prediction conditioned on previous time steps and notes generated in current time step. The Bi-Axial architecture proposes two axis modules which in the case of music generation is crucial, since we need to train the tempo and note prediction simultaneously. But the modification here is to add style conditioning at every layer.

We first parse our input data into a 1-D convolution layer with Kernel of 24 for feature extraction and concatenated with context input ($beat_{data} \rightarrow beat_{in}$)to feed the first LSTM recurrent layer, then move on to transform the style input. First, we linearly transforms the encoding of the style into embedding, where W is the projection matrix to linearly embed our style classification, due to the fact that feature of different musical styles might overlap. Then, we simply add it to each layer's output as a new input into fully connected hidden layers using $tanh$ activation for latent representation of style h' to enforcer global conditioning, where l represents the specific layer:

$$\begin{aligned} h &= tanh(Ws) \\ h' &= tanh(W'_l h) \end{aligned}$$

The major part of our model consists of two recurrent LSTM hidden layers: time-axis module and note-axis module. Each layer serves separate responsibilities of learning features in timing and note predictions. Finally, we use sigmoid activation function for the output of note-axis layer and render note prediction. With chosen note $chosen_{in}$, we concatenate it with the output of the time-axis layer to perform recurrent training.

C. SVM Genre Classification Model

The DeepJ model uses a simple one-hot encoding to represents the music style. It basically generates music by music genre rather than one specific composer. It mixes all composers' composition styles under the same genre into one-hot encoding. Additionally, the music genres such as Baroque, Classicism, and Romanticism are known as the different periods of ages. It is hard for a human with basic music knowledge to distinguish which period a given piece belongs to. But for a deep learning model, it is probably able to distinguish the certain pattern behind the notes and beats.

Therefore, our goal to improve the DeepJ is to replace the style one-hot encoding with a pre-trained a music genre classification module. There are some excellent models for music genre classification such as spectrograms identification using CNN(Bahuleyan 2018; Rai 2021). All models referred above can achieve the accuracy over 70%. However, we use the representation of notes and beats to denote a MIDI file. The pre-processing to output waveform images is redundant work.

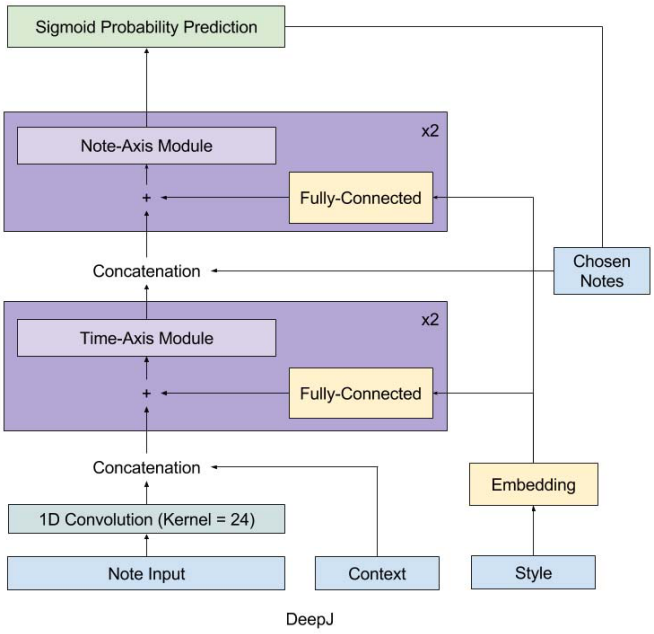

Figure 3: DeepJ architecture

In the previous work by Chet N. Gnegy(Gnegy 2014), the author concluded that by using the right feature extraction, the accuracy of the SVM genre classification model can exceed over 90%. For the style embedding in our model, we literally want to use it to represent what a piece of music sounds like by a human. Therefore, the style encoding is supposed to be a vector with the value of the probability of the music genre. However, the SVM models don't output probabilities natively. We have to convert the output to class probabilities. Among all possible approaches, the Platt scaling is particularly suitable for SVMs(MMA 2019). We are going to pre-train an SVM music genre classification model. In our music genre classification, we adopt the same feature extraction method in the reference(Shi 2019).

D. Training

The DeepJ produces three outputs: play probability, replay probability, and dynamics. We need to train all three outputs in parallel. Play and replay are treated as logistic regression problems trained using binary cross entropy, as defined in the Bi-Axial LSTM by Johnson. Dynamics is trained using Mean-Squared-Error Loss function shown as follows:

$$
\begin{aligned}
L_{play} &= \sum t_{play} \log y_{play} + (1 - t_{play}) log(1 - y_{play}) \\
L_{rply} &= \sum t_{play}(t_{rply} \log y_{rply} + (1 - t_{rply}) log(1 - y_{rply})) \\
L_{dynamics} &= \sum t_{play}(t_{dynamics} - y_{dynamics})^2 \\
L_{primary} &= L_{play} + L_{rplay} + L_{dynamics}
\end{aligned}
\tag{3}
$$

One thing worth noticing is that in preliminary experiments, the replay loss tends to explode largely due to the fact

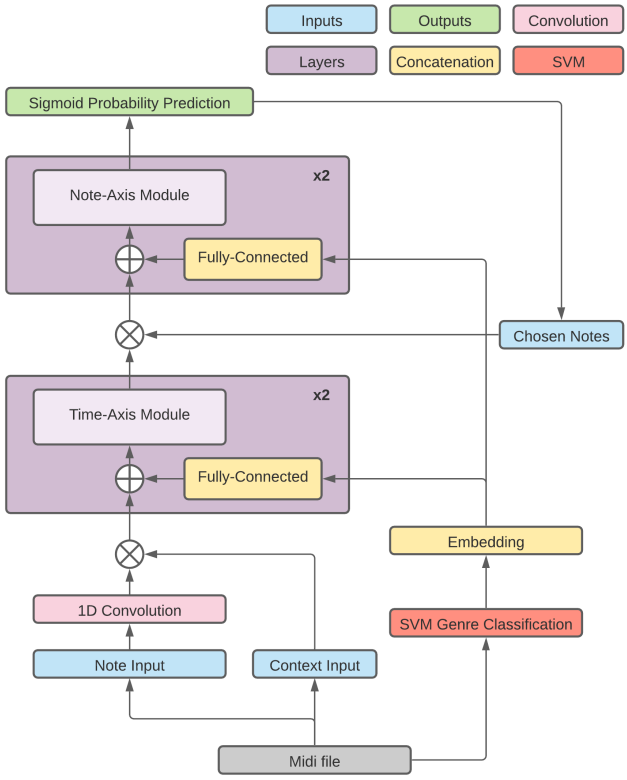

Figure 4: Our model's architecture

of including errors from notes no being played in the first place. Therefore, we fine-tuned the loss function of replay to mask out the loss when $t_{play} = 0$. The learning process used in model training is performed using stochastic gradient descent with the Nesterov Adam optimizer. Dropout is used as a regularizer during training in two parts: 0.5 for non-input layers and 0.2 for input layers.

In our model, there are 1,268,388 trainable hyperparameters in total. The time-axis layer and note-axis layer contains 912,606 and 356,230 parameters respectively. Fig.5 gives some core parameters used in our model. In the preliminary experiment, we select 18 pieces of MIDI files from 6 composers in 2 genres (classical and romantic) with duration of $\sim$85 minutes referring to Table 2. We observe that it typically takes $\sim$800s for training one epoch on our server. And it requires $\sim$120 epochs to get the relatively optimal solution. Figure 5 lists an excerpt of hyperparameters and related features.

E. Generation

With our trained model, we decide to adopt another approach different from what proposed in Mao et al.(Mao, Shin, and Cottrell 2018). Authors performed generation by sampling from the model's probability distribution using a coin flip to determine whether to play a note or not. After deciding to play a note, they sampled from the replay probability to determine if the note should be re-attacked. Never-

Variable	Value/Type	Representation
genre	List	Genre of music
styles	List<List>	Directory of dataset
NUM_STYLES	styles.size()	Numbers of styles
DEFAULT_RES	96	Resolution
MIDI_MAX_NOTES	128	Notes range [1, 128]
MAX_VELOCITY	127	Velocity range [0, 127]
NUM_OCTAVES	4	Number of octaves
OCTAVE	12	Notes in every octave
MIN_NOTE	36	Minimum note
MAX_NOTE	MIN_NOTE + NUM_OCTAVES * OCTAVE	Maximum note
NUM_NOTES	MAX_NOTE - MIN_NOTE	Number of notes between MIN_NOTE and MAX_NOTE
BEATS_PER_BAR	4	Number of beats in a bar
NOTES_PER_BEAT	4	Notes per quarter note
NOTES_PER_BAR	NOTES_PER_BEAT * BEATS_PER_BAR	The quickest note is a half-note
BATCH_SIZE	16	Training batch size
SEQ_LEN	8 * NOTES_PER_BAR	Data sequence length
OCTAVE_UNITS	64	Dim of hyperparameter in octave convolution layer
STYLE_UNITS	64	Dim of hyperparameter in style embedding
NOTE_UNITS	3	Three outputs: play prob, replay prob and dynamics
TIME_AXIS_UNITS	256	Dim of hyperparameter used for LSTMs in Time-Axis
NOTE_AXIS_UNITS	128	Dim of hyperparameter used for LSTMs in Note-Axis

Figure 5: Parameters for the DeepJ model

theless, one of the shortcomings of coin-flip initial feeding strategy might be the presence of long-term silence, which is also noted in their work. While the authors came up with an idea of deploying temperature adjustment to avoid long-term silence, we decide to generate our music through simply providing a pre-composed MIDI file or certain cut of training dataset to out model. This generation alternative is very similar to the modern approach of music content production: a complete musical work is built upon certain piece of main theme, rendering our model the potential applicability in industrial useage.

IV. Evaluation

A. Validation

As per the construction of author's model, we managed to rebuild the model and successfully test our model in the following music cuts; comparing wit the outputs of original DeepJ model, we used the following MIDI dataset shown in Table 1 as a shorter version of input given limited training resources.

Given a Beethoven's piece as an input, shown in Fig.6, the generated classical style music from our and authors' model are given in Fig.7 and Fig.8 respectively. Noted that the musical score generated by our training procedure , though with reasonable harmonic resemblance as actual musical pieces in the classic era, it has unnecessary structural complexity not seen in the result from author's output. We've expected such nuance considering the truncation on input dataset. Furthermore, the reason why we chose to used Beethoven's excerpt as an input rather than the coin-flip random feeding that author used is to partly resolve the issue yield from limited training resources. In conclusion, the overall reconstruction is fairly successful, and we can proceed to training our model into next level.

Genre	Composer	Piece number
Classic	beethoven	3
	haydn	3
	mozart	3
Romantics	borodin	3
	brahms	3
	tschaikovsky	3

Table 1: MIDI dataset for validation

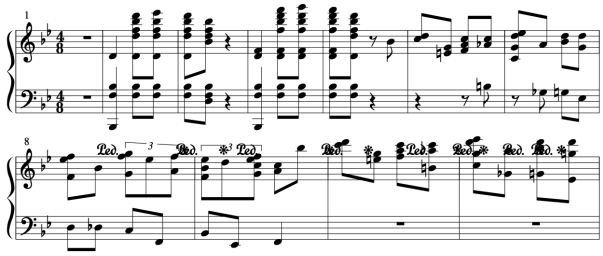

Figure 6: Input cuts

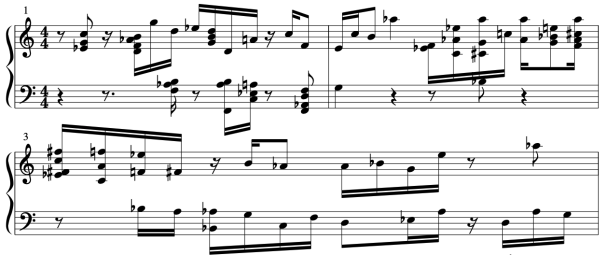

Figure 7: Classical music generated by our model with parameters in Table 1

B. Refining Model with Extended Input

Upon the reconstruction of proposed model and conducted training with limited dataset, we began to feed our model with more musical pieces amongst different composers. The generated musical score is shown in Fig.9. As shown in the figure, we can see that the redundancy in structural complexity does not demonstrate significant improvement, nor does tempo consistency, yet the tempo pattern can be well recognized now, indicating possibilities that our model actually picks up reasonable features from genre classification. Though extending our input size should address some issues arisen in section A, we are more interested in testing how different style classification module could improve the quality and musical consistency of results generated with our model.

Genre	Composer	Piece number
Classic	beethoven	12
	haydn	13
	mozart	12
Romantics	borodin	7
	brahms	10
	tschaikovsky	12

Table 2: MIDI dataset for extensive training

C. Style Embedding

With extensive research and evaluation, we decided to construct a SVM module to replace the one-hot-encoding style embedding in the original model. The dataset for DeepJ is not suitable to evaluate the performance of SVM model, so we test our SVM using another dataset mapping 13360 pieces into 13 labels. The accuracy achieves 75%.

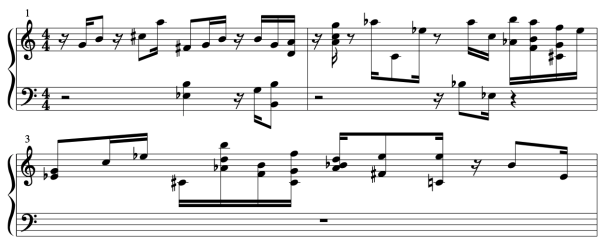

Figure 8: Classical music generated by author's model

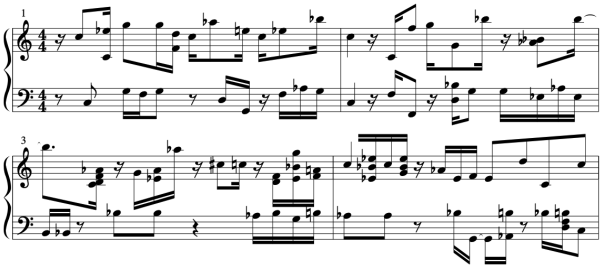

Figure 9: Classical music generated by our model with extended input

We used the piece from Mozart for testing and the following Table 3 gives the probability distribution of different composers our model outputs. Here can see that the classifier accurately predicts the composer with index 4, which represents Mozart. Upon a closer look at the actual probability distribution, we can see that the predicted probabilities of 'Tchaikovsky' and 'Mozart' are pretty close, probably due the reason that both composers are known for symphony composing and Tchaikovsky's work was greatly influenced by Mozart.

Models	Vector = ['tschai', 'beethoven', 'haydn', 'borodin', 'mozart', 'brahms']
Our model	[0.220, 0.184, 0.157, 0.047, 0.241, 0.151]
DeepJ	[0, 0.333, 0.333, 0, 0.333, 0]

Table 3: Style embedding for the Mozart's piece

V. Challenge and Strategy

- (During objective 1) Limited computing resources – Use free Google Cloud server and deploy Jupiter environment on it.
- (Resolved in objective 2) Major challenge: most of the existing music classification models center CNN(Bahuleyan 2018) to classify spectrogram images generated by certain genre. To generate another type of input data seems to be redundant and most importantly, the average test accuracy or validation accuracy rate is around low 60% – Pre-train a simple and shallow CNN music classification model by using existed dataset for DeepJ.
- Our VM server was suspended multiple times by google

probably due to IP address leakage for cryptocurrency mining, so numerous data was lost during the re-boosting process. As a result, we could not train our complete model (DeepJ reconstruction and SVM embedding) in a timely manner with alternative computing resources.

VI. Summary

In this paper, We introduced a deep neural network model for auto music generation with embedded style classification based on the work of Mao et al.(Mao, Shin, and Cottrell 2018), DeepJ. The project consists of two main components: reconstruction of the original DeepJ model, and development of a SVM genre classification module in replacement of the inefficient one-hot-encoding strategy. In general, we improved the DeepJ model in the following manners:

- Composer-level classification: the probability distribution amongst different composers from the same musical period is no longer uniform with the introduction of SVM. Therefore, our model demonstrates much higher tolerance on input specification and variation.
- Embeddability: this SVM module could be used for other purpose on its own, or Incorporated into other music generation system.
- Accuracy: the overall genre classification accuracy of our SVM module is 74.5%, and with more training samples and right feature extraction, the theoretical training accuracy could exceed 90% according to Chet N. Gnegy(Gnegy 2014).

The musical samples generated from our model could in practise resolve the issues of the absence of central theme present in DeepJ. However, the problem of enforcing long-term structural consistency still persists, and it would be a challenging yet rewarding field in future research. Further, our SVM genre classification model can also contribute to the GANs model refinement as an input of generative network as well as the output of discriminative network, connecting two parallel training systems in realization of better reinforcement learning and future work could be done in incorporating GANs into music generation model to resolve the issue long-term structural consistency issue.

References

Bahuleyan, H. 2018. Music genre classification using machine learning techniques. *arXiv preprint arXiv:1804.01149*.

Briot, J.-P.; Hadjeres, G.; and Pachet, F.-D. 2017. Deep learning techniques for music generation–a survey. *arXiv preprint arXiv:1709.01620*.

Gnegy, C. 2014. Classification of musical playing styles using midi information.

Hernandez-Olivan, C., and Beltran, J. R. 2021. Music composition with deep learning: A review. *arXiv preprint arXiv:2108.12290*.

Johnson, D. D. 2017. Generating polyphonic music using tied parallel networks. In *International conference on evolutionary and biologically inspired music and art*, 128–143. Springer.

Kong, Q.; Li, B.; Chen, J.; and Wang, Y. 2020. Giantmidi-piano: A large-scale midi dataset for classical piano music. *arXiv preprint arXiv:2010.07061*.

Mao, H. H.; Shin, T.; and Cottrell, G. 2018. Deepj: Style-specific music generation. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, 377–382. IEEE.

MMA. 2019. Can you interpret probabilistically the output of a support vector machine?

Rai, S. 2021. Music genres classification using deep learning techniques.

Shi, S. 2019. Midi classification tutorial.

Simon, I., and Oore, S. 2017. Performance rnn: Generating music with expressive timing and dynamics. https://magenta.tensorflow.org/performance-rnn.