Comparing Different Neural Network Architectures for Music Generation

Candidate Numbers: 42903, 52559, 53480

Abstract

Music generation is a rapidly developing area of generative deep learning with many potential applications, such as automating routine work in music creation (e.g., mixing) and facilitating the creative process by helping make new sounds and melodies. In this area, there are numerous approaches to both data preprocessing and neural network architectures. In this project, we are going to try several approaches to transforming the raw files, implement RNN and GAN architectures for the task of music generation, compare the results for each and make conclusions about the benefits and drawbacks of the methods mentioned above. As for the data, we used the MAESTRO dataset, which includes 200 hours of classical piano music, and Classical Music MIDI, which is comprised of music by famous composers.

1. Introduction

Music generation is unique from other deep learning modalities in several ways: there is a temporal element, it involves different instruments unfolding in different ways simultaneously, and musical groups - i.e. chords, arpeggios, etc. - that occur, such that ordering notes in a chronological way is not always suitable. These elements all impact the architecture and training methods behind music generation, and these complexities can make the task more challenging than other modalities.

While there has been great success in music generation over the past several years, there are many limitations to previously developed models and how the quality of music is quantified. While specific limitations in RNN and GAN models have been found, we plan to further evaluate the benefits and drawbacks of different neural network architectures for generating music. Given measures for music quality are scarce and inconsistent, our goal is to also develop a framework to thoroughly quantify the quality of music generated.

RNN and GAN models are most commonly used for music generation. RNN has limitations regarding variability in notes and creating longer songs that retain repetition, while GAN has limitations in the diversity of music products, maintaining a uniform structure, and generating longer songs as well. There exist many other limitations as well, based on subjective measures of melody, harmony, expressivity, and style transfer (Briot et al., 2017).

In addition, there are problems in how the "quality" of music is measured, making it challenging to compare which neural networks are best. Many papers only use subjective measures to evaluate music. For instance, in a study on symbolic music generation with transformer-GANs, evaluators were asked which music they preferred (of the different models).

We plan to use Recurrent Neural Networks (RNN) (specifically LSTM) and Generative Adversarial Networks (GAN) to generate music and further evaluate the benefits and drawbacks of each model. We will also develop an evaluation chart that allows us to evaluate the elements of music in a more objective way.

Two different datasets are used in our implementation: the Classical Music Midi and the MAESTRO dataset. The Classical Music Midi dataset contains classical piano midi files from 19 famous composers, whereas the MAESTRO dataset contains 200 hours of paired audio and midi recordings from the 17th-20th century. To maintain uniformity for our analysis, both are classical, piano-based datasets.

2. Related work

There are several studies that have examined different models for music generation and their benefits and drawbacks. (Chou & Peng, 2019) researched the benefits and drawbacks of LSTM architecture, using the MAESTRO dataset in their model. They found that a batch size of 128 and 1000 epochs produces the best accuracy; however, there are fluctuations in training loss that marks models challenging to compare. They also discovered that LSTM has underfitting accuracy and that more epochs, more tuning, and a deeper network might improve the accuracy of this architecture. Another LSTM RNN study was able to produce music indistinguishable to composer-produced music. However, the model was based on one instrument and researchers recommended further research on several instruments and a higher repertoire of training data to continue to test efficacy (Ingale et al., 2021). An additional study using LSTM used piano MIDI

files to generate new music (Abadi et al., 2022), which split the note into three variables during model training. A custom loss function based on mean squared error was used in this model. A portion of this code was recycled for our own implementation and research.

(Dong et al., 2018) explored music generation using GAN architecture via the Lakh MIDI dataset. Unlike (Chou & Peng, 2019), (Dong et al., 2018) developed additional objective metrics (i.e. the ratio of empty bars, number of pitch classes per bar, the ratio of qualified notes, the ratio of drum patterns, and tonal distance to gather more information on the quality of music. They found their model successfully generates harmony in chords, and rhythmic patterns, and played music within a reasonably musical scale and melody. While the music matched up with these quality metrics, the quality was still not up to par with musical-produced music, given several of the objective measures referenced inconsistently with certain elements at a time. Likewise, there was some noise and sparsity in the data, which can be adjusted through cleaning and filtering. Another study discovered that a specific GAN model (SSMGAN) was able to produce self-repetitive music, addressing potential concerns in a computer's ability to generate longer songs that maintain appropriate melody and mood (Jhamtani & Berg-Kirkpatrick, 2019). However, given the models are not backed up by basic music theory, there can be inappropriate pauses and a lack of rhythm in the music.

3. Datasets

3.1. MAESTRO

The MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organisation) dataset (https://magenta.tensorflow.org/datasets/maestro) is a dataset composed of about 200 hours of paired audio and MIDI recordings from ten years of International Piano-e-Competition, totalling to 1276 midi files.

The MAESTRO dataset will be used in MidiRNN (Section 4.1) and MidiGAN (Section 4.2).

3.2. Classical Music MIDI

The Classical Music MIDI dataset (http://www.piano-midi.de/) is a dataset composed of 295 classical music recordings from 19 famous composers such as Bach, Beethoven, Mozart, etc.

The Classical Music MIDI dataset will be used in ImageGAN (Section 4.3).

4. Training Models

4.0.1. Activation functions

- $sigmoid$ function

$$sigmoid(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

- $tanh$ function

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2}$$

- $ReLU$ function

$$ReLU(z) = max(0, z) \tag{3}$$

- $LeakyReLU$ function

$$LeakyReLU(\mathbf{z}) = max(\beta * \mathbf{z}, \mathbf{z}), \tag{4}$$

where β is a small constant and z is the name of the variable of an activation function.

- $softmax$ function

$$\sigma(\overrightarrow{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{5}$$

where K is the number of classes.

4.0.2. Loss functions

- Binary cross-entropy loss function

$$H_B(y, \hat{y}) = -(ylog\hat{y} + (1 - y)log(1 - \hat{y})) \tag{6}$$

- Sparse categorical cross-entropy function

$$J_\theta(h) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - h(x^{(i)}))^2 = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2 \tag{7}$$

where m is the number of rows, $\hat{y}$ is the predicted class probability, and y is the true class label.

4.1. MidiRNN

MidiRNN follows TensorFlow's tutorial on generating music (Abadi et al., 2022).

4.1.1. Pre-processing

All the notes are first extracted from selected midi files, and then three attributes of the notes are used in training the model. These attributes are the pitch, the step, and the duration of each note, where the pitch is the perceptual

quality of the sound as a midi note number, the step is the time elapsed from the previous note or the start of the track, and the duration is how long the note will be playing.

The attributes of each note are extracted and stored in a dataframe, with each row representing each note. This dataframe is then converted into a tensor object. Training sequences are then made, where the pitches are normalised, with a sequence length of 50 and a vocab size of 128. Then, this training sequence is shuffled and batched with a batch size of 64, creating the training dataset. This training dataset has a shape of (64, 50, 3), where the 3 represents pitch, step, and duration.

4.1.2. The architecture

Long short-term memory (LSTM) architecture is a subtype of recurrent neural networks (RNNs), in which a sequence of notes is inputted and the model predicts the next note given the elements and our pitch vocabulary. In our adaptation of the MidiRNN model, notes are extracted from MIDI files and imputed into the training model. A certain number of midi files is extracted for training. Initially, the number of files is set to 5, and then experimentation with other values (i.e. 3, 6, and 10) are performed. The input involves an LSTM layer with 128 units. The output has three branches representing pitch, step, and duration (elements of the notes), each involving a dense layer. The first branch has 128 units and the next two branches have one unit, ending with an output shape of [None, sequence length, 3]. We experimented with different sequence lengths (i.e. 25 and 50) to understand how this impacts the quality of music generated.

See Appendix A.1 Figure 1 for the model plot.

4.1.3. Training methods

The batch size was set to 64, and trained over 50 epochs. While these are lower than the recommended amount (Chou & Peng, 2019), setting a lower batch size prevents overfitting and allows for efficient running during these experiments. However, a lower epoch value might also lead to a slower convergence (Radford et al., 2015). The dataset being trained takes on a shape of (100, 1), such that it takes in 100 notes and predicts the upcoming note in the song. The learning rate was set to 0.005 and Adam optimizer was used. Sparse categorical cross entropy (Equation 7) was used as a loss function for the pitch branch, and the mean squared error loss function was used for the step and duration branches. The MSE loss function enforces the model to output non-negative values. In this LSTM model, pitch loss tends to be higher than step and duration loss. Given the output model is passed into a softmax function (Equation 5, a softmax distribution of notes and their probabilities, we are able to pick a note that has a high probability of being played and then store this for our note generation. The generated notes are combined and converted to a midi file. The quality of music was evaluated through the loss function and evaluation by the listener on several objective measures.

4.1.4. Numerical results

Experiments were run in the MidiRNN model to determine how certain factors, including the number of files in the training set and sequence length, impact the quality of music produced. The original file contains num_files set to 5, and seq_length set to 25. The generated music contains notes that are aligned in a way that sounds like a song; however, there is some hypertonia at the end which sounds out of place. The beginning of the song has a suspenseful mood with a uniform rhythm.

Changing the sequence length from 25 to 50 induces a longer run time for training the model (28 versus 93 minutes). An increased sequence length causes higher loss in the model (0.29 versus 0.22, Figure 2), as well as more oscillation in the later epochs. Our generated song plays several notes at the same time however, there is dissonance. The melody sounds off, as there is a random switch between an increase and a decrease in pitch.

Changing the number of files from 5 to 6 reduces the run time and loss in the model (0.22 to 0.18, Figure 3). The generated music has more variation in notes and a faster tempo that sounds relatively consistent. The mood is upbeat and remains consistent throughout the song. Experiments with num_files = 3 and 10 lead to early stopping, suggesting num_files of 5 or 6 fit better with our model. Overall, this model seems to generate music most similar to how musician-produced music would sound in terms of melody, rhythm, and mood (Table 1).

	Melody	Rhythm	Harmony	Uniform mood
Model 1	✗	✓	N/A	✓
Model 2	✗	✗	✗	✓
Model 3	✓	✓	N/A	✓

Table 1. Evaluation of MidiRNN music quality

4.2. MidiGAN

4.2.1. Pre-processing

The pre-processing is similar to TensorFlow's tutorial (Abadi et al., 2022) on generating music with RNN. Similarly to MidiRNN, all the notes are first extracted from selected midi files, and then the three attributes (pitch, step, and duration) of the notes are used in training the model.

Similarly, the attributes of each note are extracted and stored in a dataframe, with each row representing each note. This

is where the pre-processing differs from MidiRNN. This dataframe is then normalised by scaling it to have a zero mean and a unit standard deviation, ensuring that the range of the input data is similar, helping the neural network to converge faster. This normalised dataframe is then used to create a training dataset by converting the dataframe into a tensor object, which is then batched with a batch size of 64.

4.2.2. The architecture

The GAN architecture consists of two main components: the generator model and the discriminator model. The generator's function is to generate fake data while the discriminator's model is trained to distinguish the generated fake data from the real data.

The generator model begins by taking in a 100-dimensional random noise vector as input and passes it through a series of dense layers with varying numbers of neurons of 128, 256, and 512. Each of these dense layers is followed by a batch normalisation layer, which helps in the faster convergence and the overall performance of the model. After batch normalisation, a Leaky ReLU (Equation 4) activation function is applied which allows a small non-zero gradient when the unit is not active. This helps the problem where neurons stop learning and die. The output layer of the generator is a dense layer with 3 neurons representing the 3 attributes of a note with a '$tanh$' activation function (Equation 2). The '$tanh$' activation function is used in this case as '$tanh$' centred its output around 0 by outputting values in the -1 to 1 range, which is suitable as our training data is normalised.

The architecture of the discriminator is somewhat similar to that of the generator but in reverse order. The discriminator model begins by accepting an input size of 3, representing the three attributes of a note. Similarly to the generator, it has dense layers with 512, 256, and 128 neurons. To prevent overfitting, dropout layers with a rate of 0.3 are introduced after each dense layer. The output layer of the discriminator is a single neuron with a sigmoid activation function (Equation 1, outputting the probability that the input sample is real.

See Appendix A.2 Figure 4 for the model plot.

4.2.3. Training methods

Both the generator and discriminator use the binary cross-entropy loss for their loss functions. Both also use the Adam optimisers with a learning rate of $1e-4$. The Adam optimiser is used because it is generally a suitable optimiser for GAN as it has an adaptive learning rate that can help the model converge faster and more reliably. In our case of music generation, where the patterns and structures can vary greatly, adaptive learning rates can be beneficial.

4.2.4. Numerical results

Note that the labelling of Table 2 is based on the files chosen randomly from the existing 1276 available midi files. Thus, reproducing the training dataset will vary from Table 2. Therefore, Table 2 is based on a specific randomly generated case.

	Melody	Rhythm	Harmony	Uniform mood
5 files 25 epochs	✓	✓	✗	✗
5 files 50 epochs	✓	✓	✓	✗
10 files 25 epochs	✓	✓	✗	✗
10 files 50 epochs	✓	✗	✗	✗

Table 2. Evaluation of MidiGAN music quality

An observation can be made when repeating the generation procedure with different randomly selected files, which is that generated music using 10 files sounded more dissonant than generated music using 5 files. The songs generated all generally still sound rather dissonant, with the notes still sounding rather random. This is why it results in a cross mark across whether the songs have uniform moods. This could be that the model is overfitting and is trying to replicate dissonant elements present in the training dataset. Another reason could be that the model lacks appropriate regularisation techniques and struggles to capture the underlying structure and harmony of the training data or the model's architecture is just simply not better enough. On the other hand, aside from the general dissonance, there were noticeable melodies with proper rhythms throughout the songs, with the rarer occasions where harmony was ever so slightly present.

4.3. ImageGAN

4.3.1. Pre-Processing

Some other previously implemented GANs (Sim, 2021) dedicated to music generation using an alternative approach to transforming the data input: instead of using MIDI files directly and simply extracting notes from them, they transform them into images where rows represent notes and columns represent a timestamp when it is played. We decided to implement this approach as well to compare its performance to that of other neural networks in this project and to make a conclusion about whether this approach is a viable alternative to using music files directly. To do so, we used the modules img2midi and midi2img (Gatti & Savin, 2019) on our dataset.

One of the features of the module is that if a music file is longer than 100 notes, the program splits it into several images. We have experimented by joining them back, however, for 106x200 images resulted in an unsatisfactory output, while for 106x300 and above the execution of training would result in a crash due to memory overload. Therefore, we have decided to proceed with 106x100 images.

4.3.2. The architecture

The neural network has an extensive architecture, a scheme for which is provided in the Appendix. The generator accepts an input of 10 latent dimensions into a Dense layer followed by a LeakyReLU (Equation 4) activation with slope 0.2 and a Dropout layer with a dropout rate 0.2. The architecture features another 3 Dense layers, a Conv2DTranspose and a Conv2D layer also with LeakyReLU activations and Dropout layers between them. Since in the current approach the data is represented in the form of images, convolutional layers are appropriate and convenient as they preserve the main features in the data while significantly reducing the number of parameters. LeakyReLU has led to better performance than the conventional ReLU, perhaps due to the fact that it preserves information for negative values. As for the discriminator, it accepts 106x100 images as an input and has 2 of the following blocks of layers: Conv2D with 16 channels, Dropout with rate 0.2, LeakyReLU with slope 0.2, and another Dropout. These are followed by a flattening layer, batch normalization, a Dropout layer, and a final Dense layer outputting the decision of the discriminator. Since we are working with images here, the choice to implement several convolutional layers makes sense. According to some previous works (Radford et al., 2015), it is recommended to only use LeakyReLU in the discriminator.

See Appendix A.3 Figure 5 for the model plot.

4.3.3. Training methods

As for the optimizer, we used Adam with a learning rate equal to 0.0002 and beta 1 of 0.5 similar to the paper mentioned above. We acknowledge that experimenting with the parameters further could have been beneficial, however, due to limited time resources we decided to keep these values of parameters. Binary cross-entropy

The neural network with the chosen architecture was initially trained for 20 epochs. Its output featured quite a lot of diversity in the notes and pitch, e.g., it was not simply repeating one note or chord and was not monotonous. However it sounded sharp and not very melodic, the pauses were too abrupt. We then retrained the same neural network for 40 epochs, and the quality of the output improved significantly: the generated samples were more musical, melodic, and consistent (see Table 3). However, it appears that the generated tracks were quite similar to each other, and the harmony and the melody were not perfect.

4.3.4. Numerical results

As for the numeric results, the following was achieved. The neural network trained for 20 epochs has resulted in a discriminator loss of 0, generator loss of 15.796, and 100% accuracy on both fake and real images. The one trained for 40 epochs had a discriminator loss of 0.691, generator loss of 0.592, accuracy on real images of 17% and accuracy on fake images equal 94%. Considering that the output from the second model had better quality, let us interpret the results. Ideally, we would like to achieve a GAN that is able to generate images that are indistinguishable from real ones, so in a perfect scenario, the accuracy on real images should tend to 100%, while accuracy on fake images should tend to 0%. Therefore, in the first case, our GAN is not doing a great job at fooling the discriminator with the generated images, which shows in the high loss of the generator; in the second case, both losses are quite low, and accuracy on fake images has lowered, meaning that the generator has improved in generating images that look real. However, there is still a lot of room for improvement.

Overall, given the constraints in data and computational power, the neural network has performed relatively well in terms of output quality, however, its main drawback is the speed of training, which was around an hour. We expect that performance could be improved further by experimenting with the data input, e.g., by inputting more data, or limiting the data to music created by one composer to achieve more consistent and homogenous outputs, and training on bigger images once more computational power is obtained.

	Melody	Rhythm	Harmony	Uniform mood
20 epochs	✗	✓	✗	✓
40 epochs	✓	✓	✓	✓

Table 3. Evaluation of ImageGAN music quality

5. Conclusion

Overall, our three models produce promising results that show how deep learning can generate new music that sounds similar to musician-produced tracks. The exploration of music generation through the application of RNN and GAN has illuminated the potential and challenges of using automatic generation of musical content. Our MidiRNN model shows how, with correct adjustments to the training model, we can produce music with a good melody, rhythm, harmony, and a uniform mood. MidiGAN produces music that fits several of our music quality criteria (melody, rhythm, harmony); however, the music that is produced lacks a uniform mood. ImageGAN produces music that fits with all of our

general music quality criteria, although the long training time is a limitation of this model. Another limitation in all three models is that the songs produced are of short time lengths, around 15-30 seconds. Further experiments could be done to analyse how we can generate longer songs that have enough variation in notes while retaining a uniform mood throughout. As these models only focus on one instrument (piano notes), we could also experiment further with music generation for multiple instruments. In future work, we could experiment with stronger, more complex architecture as well. While more complex architectures might be more time-consuming to train, they might generate higher-quality music that is more similar to what a musician would produce. In addition, beyond the pitch, step, and duration attributes some of our models were trained on, we could train our model on additional elements as well (i.e. chord progressions). Further experimentation could also be done to incorporate general music theory rules into our neural network models, to improve harmony and melody.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. Generate music with an rnn, 2022. URL https://www.tensorflow.org/tutorials/audio/music_generation.

Briot, J.-P., Hadjeres, G., and Pachet, F.-D. Deep learning techniques for music generation – a survey, 2017. URL https://arxiv.org/pdf/1709.01620.pdf.

Chou, K. and Peng, R. Deep learning music generation, 2019. URL https://cs230.stanford.edu/projects_fall_2019/reports/26258004.pdf.

Dobilas, S. Deep convolutional gan — how to use a dcgan to generate images in python, 2022. URL https://towardsdatascience.com/deep-convolutional-gan-how-to-use-a-dcgan-to-generate-images-in-python-b08afd4d124e.

Dong, H.-W., siao, W.-Y., Yang, L.-C., and Yang, Y.-H. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment, 2018. URL https://salu133445.github.io/musegan/pdf/musegan-aaai2018-paper.pdf.

Gatti, M. and Savin, M. Midi to image conversion, 2019. URL https://github.com/mathigatti/midi2img/tree/master.

Godwin, T., Rizos, G., Baird, A., Futaisi, N. D. A., Brisse, V., and Schuller, B. W. Evaluating deep music generation methods using data augmentation, 2021. URL https://arxiv.org/abs/2201.00052.

Ingale, V., Mohan, A., Adlakha, D., Kumar, K., and Gupta, M. Music generation using deep learning, 2021. URL https://www.researchgate.net/publication/351708912_Music_Generation_using_Deep_Learning.

Jhamtani, H. and Berg-Kirkpatrick, T. Modeling self-repetition in music generation using generative adversarial networks, 2019. URL http://www.cs.cmu.edu/~jharsh/papers/music_workshop_paper.pdf.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015. URL https://arxiv.org/abs/1511.06434.

Sim, V. Musegan: Using gans to generate original music, 2021. URL https://towardsdatascience.com/bachgan-using-gans-to-generate-original-baroque-m

Tham, I. Generating music using deep learning, 2021. URL https://towardsdatascience.com/generating-music-using-deep-learning-cb5843a9d55e

A. Appendix

A.1. Architecture of MidiRNN

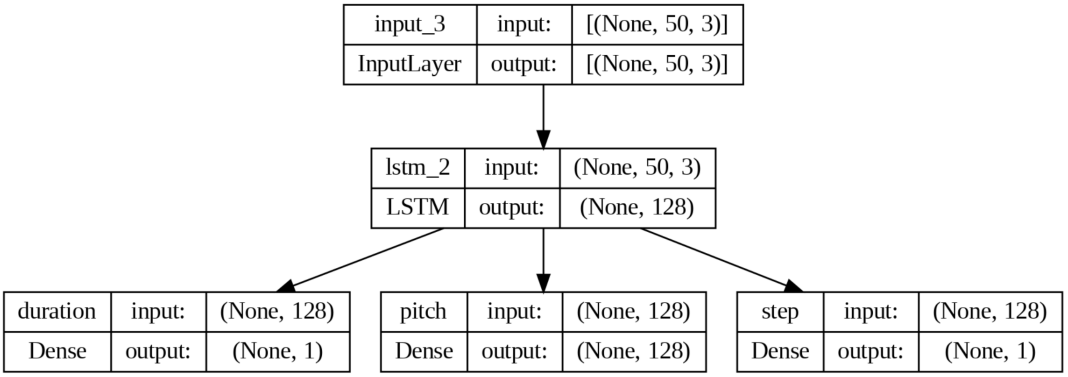

Figure 1. Architecture of MidiRNN

A.1.1. Loss graphs of MidiRNN

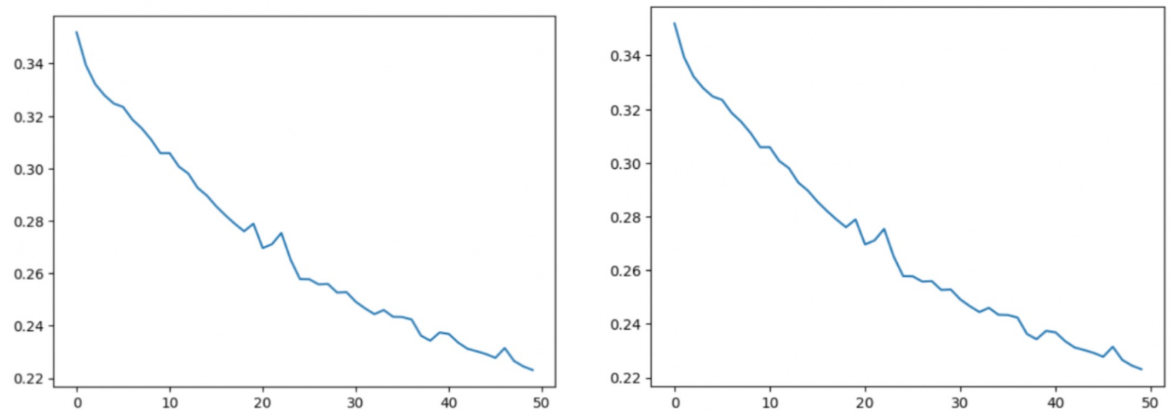

Figure 2. left: num_files = 5, seq_length = 25 right: num_files = 5, seq_length = 50

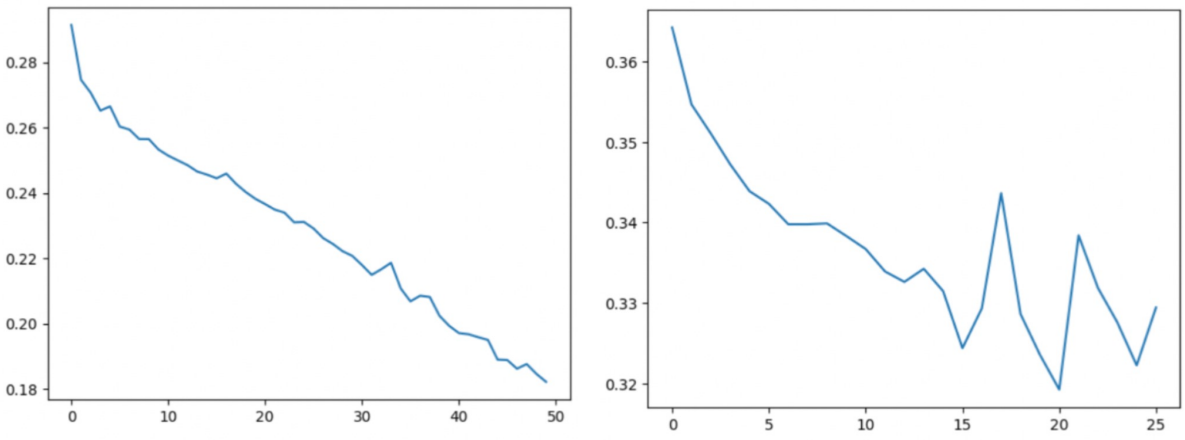

Figure 3. left: num_files = 6, seq_length = 25 right: num_files = 3

A.2. Architecture of MidiGAN

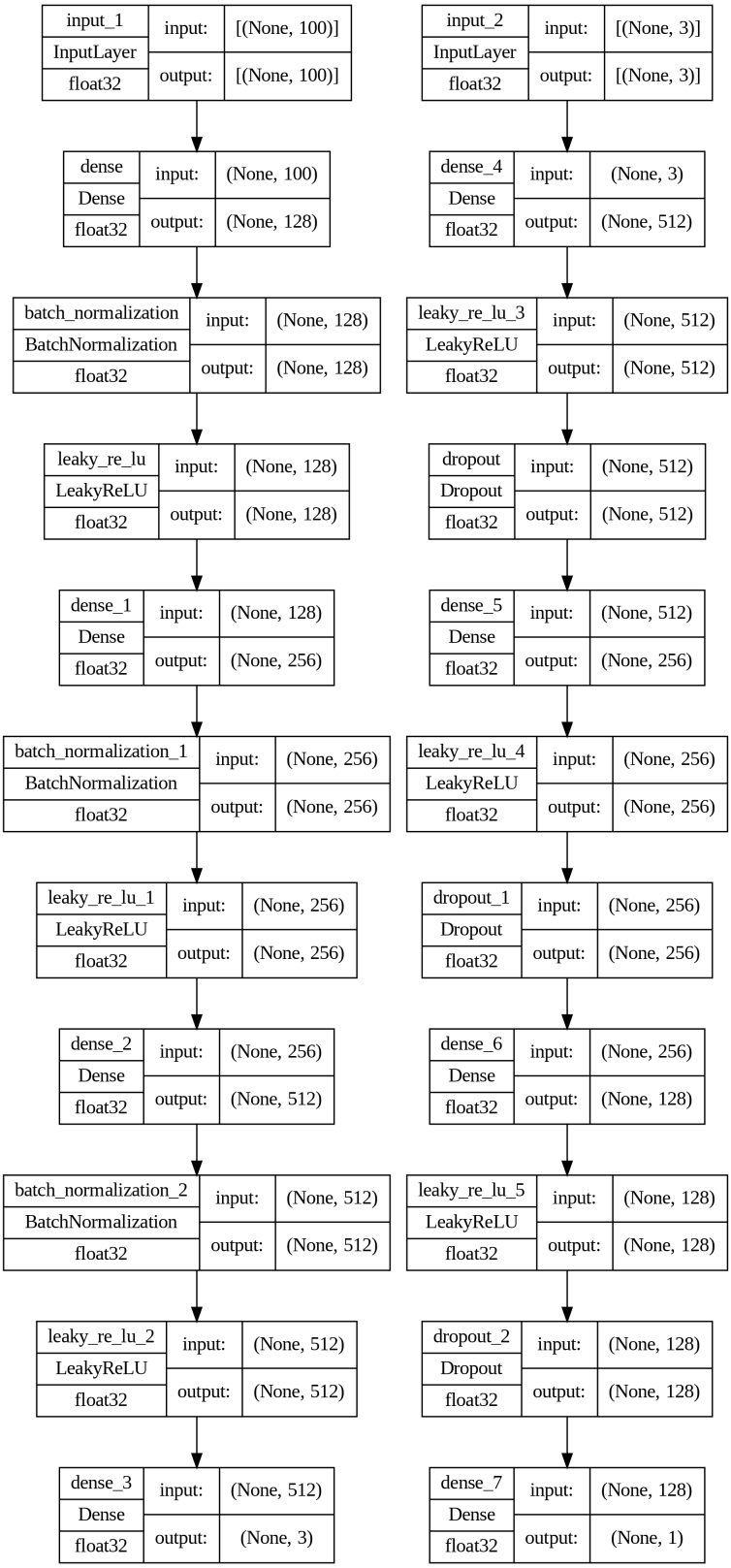

Figure 4. Architecture of MidiGAN left: Generator right: Discriminator

A.3. Architecture of ImageGAN

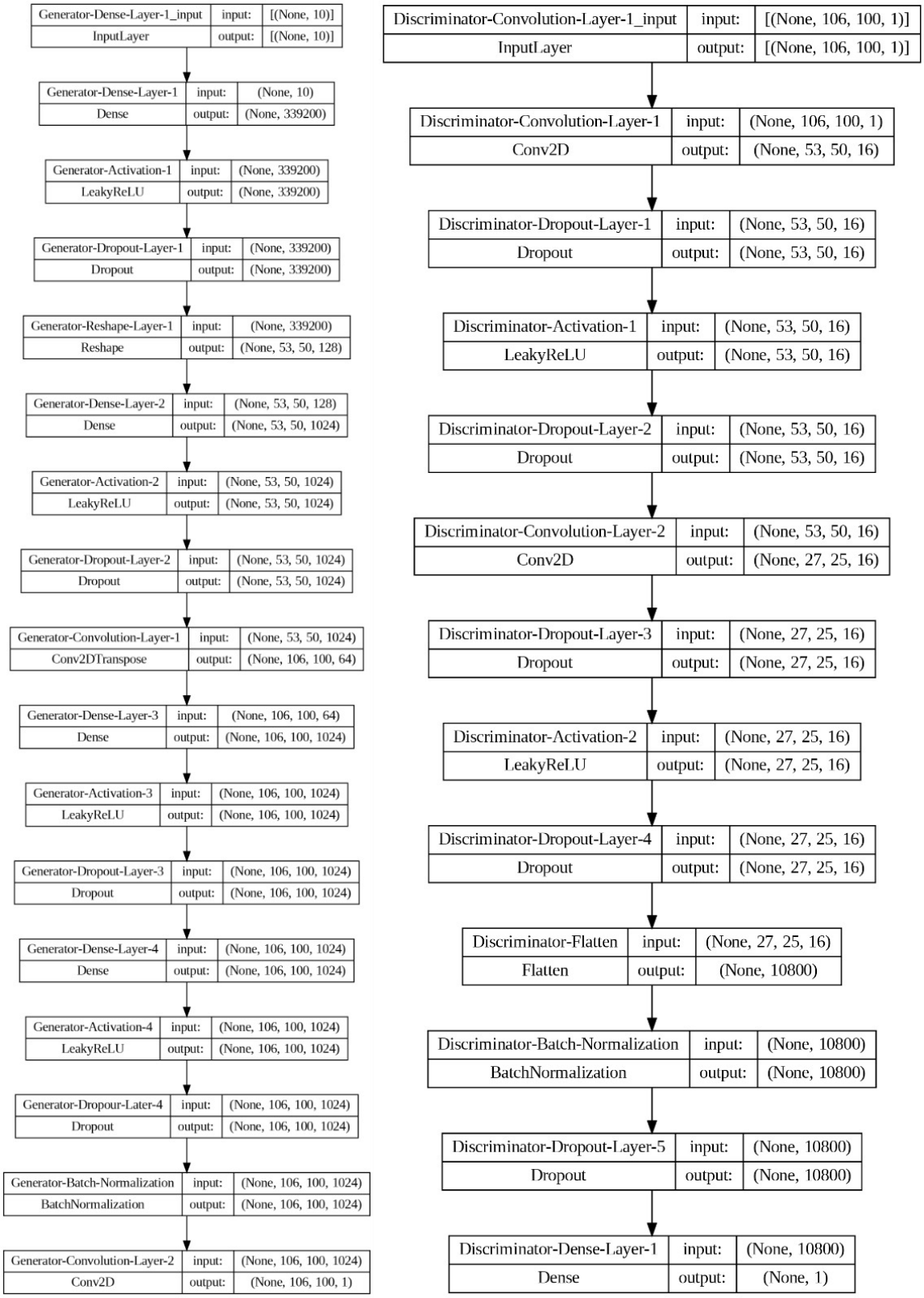

Figure 5. Architecture of ImageGAN left: Generator right: Discriminator