Department of Computer Science

BSCCS Final Year Project Report

2020-2021

20CS034

Artificial Intelligence for Classical Music composition in different eras

(Volume 1 of 1)

Student Name : **KUO Chia-Tse**

Student No. : **54889895**

Programme Code : **BSCEGU3**

Supervisor : **Dr CHAN, Antoni Bert**

1st Reader : **Dr CHAN, Chung**

2nd Reader : **Dr WONG, Hau San Raymond**

For Official Use Only

Student Final Year Project Declaration

I have read the project guidelines and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I hereby declare that the work I submitted for my final year project, entitled:

Artificial Intelligence for Classical Music composition in different eras

does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Project Guidelines.

Student Name:	KUO Chia-Tse	Signature:	Chia-Tse, Kuo
Student ID:	54889895	Date:	March 29, 2021

Abstract

Artificial Intelligence could bring music composition to another level with limitless possibilities as an assistant for human musicians or an AI musician itself. Living in a digital era, Classical Music plays a dominant role in commercial films, movie trailers, game soundtracks, and more. In general, Classical Music can be subdivided into Baroque, Classical, Romantic, and Modernist eras. Among all Classical Music of different periods, musicians nowadays might be proficient in only one of them. With this AI it would be possible for composers, musicians, or even non-specialists without prior knowledge of Classical Music theories and backgrounds could quickly compose Classical Music according to their favorite musical era for many practical purposes.

In recent years, AI arts, especially the AI music generation, have become popular, and related technologies have become available to people. However, most AI music generator platforms mainly focus on creating modern music, and they usually categorize all genres of Classical Music into a single category – Classical Music. Most research on AI music generation adopted LSTMs or other types of algorithms published earlier to generate music. Thus, it is worthwhile to discover different types of newer machine learning methods such as CNN-GANs for this task, and perhaps the proposed models could set successful precedence in this research area.

This project aims to generate Classical Music using generative models – Bi-LSTM and CNN-GAN to compose Classical Music for some particular Classical Music genres and evaluate their performance respectively and collectively. Also, to further explore and strengthen the area of Artificial Intelligence in music composition.

Acknowledgements

I want to express my appreciation to my supervisor Dr. CHAN, Antoni Bert, for his guidance and support throughout the whole project. The valuable suggestions and encouragement from him helped me improve my work quality and better manage my project.

Also, I am grateful to all survey respondents who took valuable time to evaluate my proposed neural network models' performance which allowed me to refine my designs on the proposed neural networks.

Table of Contents

1. Introduction

1.1 Motivation

As an assistant for musicians, Artificial Intelligence could bring music composition to another level with unimaginable possibilities. Living in a multimedia era, Classical Music plays a dominant role in commercial films, movie/ game soundtracks, and more. The realm of Classical Music can be subdivided into Baroque, Classical, Romantic, and Modernist era, and so forth. Among all styles of classical music, musicians might only be proficient in some classes. However, if assisted by AI composers, musicians, or even non-specialists without prior knowledge of music theories or related backgrounds could quickly generate Classical Music with desired genres as they wish, whether for self-entertainment or commercial use.

1.2 Limitations

Though some studies have shown fine results, the state-of-the-art approach(es) of AI music generation is still uncertain. Besides, most music generation systems are mainly focused on recreating music that mimic specific composers' styles. Composers like J.S Bach, Beethoven, and Mozart are some of the most iconic artists studied. Nevertheless, individuals or commercial entities interested in composing Classical Music with AI's help might just be looking for a broader scope, as the typical musical eras, of music styles instead of merely mimicking specific composers' styles. For instance, it would be reasonable and more manageable for a filmmaker to fit a piece of Romantic Music into part of a romantic film rather than accompanying it with a selection from Chopin or Schubert. Similarly, a Baroque or Classical style of music could be used as soundtracks for video games set in antiquity, so that game designers could have more options than just using Bach's music.

1.3 Aims and Objectives

This project aims to identify different deep neural network architectures best suitable for generating Classical Music of different eras/ styles – Baroque, Classical, Romantic, and Modernist. In this project, variations of two deep neural network architectures will be evaluated – Long Short-Term Memory Network (Hochreiter & Schmidhuber, 1997) and Generative Adversarial Network (Goodfellow et al., 2014).

1.4 Scope

1.4.1 In scope

- Collect and preprocess midi files to matrix format.

- Set up new designs of Bi-LSTM and CNN-GAN networks for model training.
- Generate polyphonic piano music for a single channel.
- A comparative study of each model and identify the optimal one for each music style.

1.4.2 Out of scope

- Dynamics (e.g., Forte (f, ff, fff) or Piano (p, pp, ppp)) in music will not be considered since most midi files do not include such information. Thus, it could not be trained.

1.5 Organization of the paper

The following sections is a literature review on existing methods for generating music and the proposed neural network designs. The results and conclusion follow the implementation of the systems.

2. Literature Review

2.1 Existing approaches and their limitations

Previous studies have shown potential in generating music by implementing Markov Chain, Recurrent Neural Network, and Generative Adversarial Network. However, each study has its limitations; therefore, there are still rooms to be improved from the earlier approaches.

2.1.1 Markov Chain

According to specific probabilistic rules, a Markov Chain is a system that changes its state from one to another in fixed timesteps. When applying the Markov Chain to music generation, each note in the training set is assigned a unique state. The model will then generate new states (notes), learning from the past states sequentially. The most straightforward design of a Markov Chain is to limit the model to use a single previous state to predict another. It is reasonable and feasible to apply Markov Chain for assisting real-time improvisational performance like jazz as musicians usually play spontaneously.

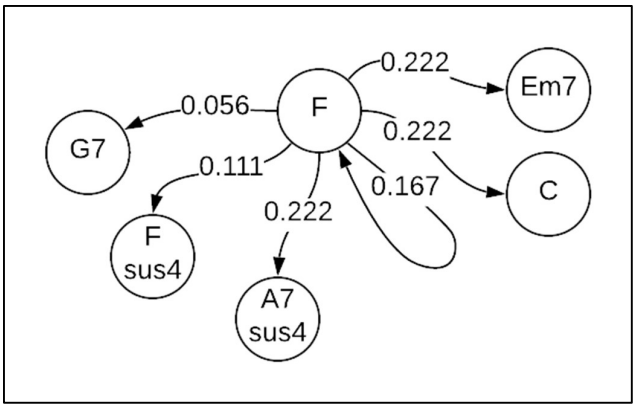

Figure 2-1 An example of weighted next possible notes. The next note after 'F' could be Em7, C, A7, F, G7 or itself: F.

However, when Markov Chains generated music from an existing musical corpus with full compositions, the results were not satisfying. It showed weird arrangements of notes and chords that were unmusical (Moorer, 1972).

2.1.2 Recurrent Neural Network

A Recurrent Neural Network (RNN) is a neural network where the current input is fed in with output from the previously hidden units. Each node of an RNN has a hidden state which allows the network to memorize a sequence of outcomes from the past. This feature of memorization means that RNN can be applied to tasks like Natural Language Processing (NLP), Time Series Pattern Predictions, and more. Since the traditional RNNs have drawbacks of not being able to

memorize long sequences of data and suffering from problems of exploding and vanishing gradient, most studies applied Long Short-Term Memory (LSTM, a kind of RNN) as an alternative to generate longer sequential information like text, speech, and music. Typically, an LSTM consists of a forget gate that removes useless information, an input gate that adds useful information together, and finally an output gate that extracts valuable information as an output of the current cell. Eck and Schmidhuber (2002) demonstrated that an LSTM model was able to compose novel blues music in styles that were similar to its trained (input) data. However, Hadjeres et al. (2016) commented that the music lacks flexibility and could not perform reharmonization in Bach, for instance.

Choi et al. (2016) conducted another study that showed a text-based LSTM algorithm for automatic music composition using different characters as unique notes to learn the relationships between chord progressions and drum tracks. Nevertheless, they discovered that the character-based RNN failed to generate meaningful structures of drum tracks by producing undesired arbitrary 0's and 1's. as output.

F:9 |	F:9 F:9 F:9 F:9 D:min7
D:min7 G:9 |	D:min7 G:9 G:9 C:maj C:maj
C:maj F:9 |	F:9 F:9 C:maj C:maj C:maj
C:maj |	C:maj

Figure 2-2 An example of 4-bar text-based LSTM training data. The right-hand side represents the score from the left with removal of the bars.

Boulanger-Lewandowski et al. (2012) suggested a combined approach that improved their previous study in generating polyphonic music by introducing the RNN-RBM model. Each RNN hidden unit is combined with an RBM. An RBM or Restricted Boltzmann Machine is a bidirectional connected and stochastic (or generative) system with visible nodes and a hidden node where all visible nodes are connected, but the hidden nodes do not. Each state of the RNN unit gets both inputs from its previous state and the RBM observation vector, and this allows the model to better predict the coming notes by learning from the prior timestep of notes. However, the Magenta project (Shiebler, 2016) commented that this model had difficulty generating rhythmic and melodic music for longer timesteps.

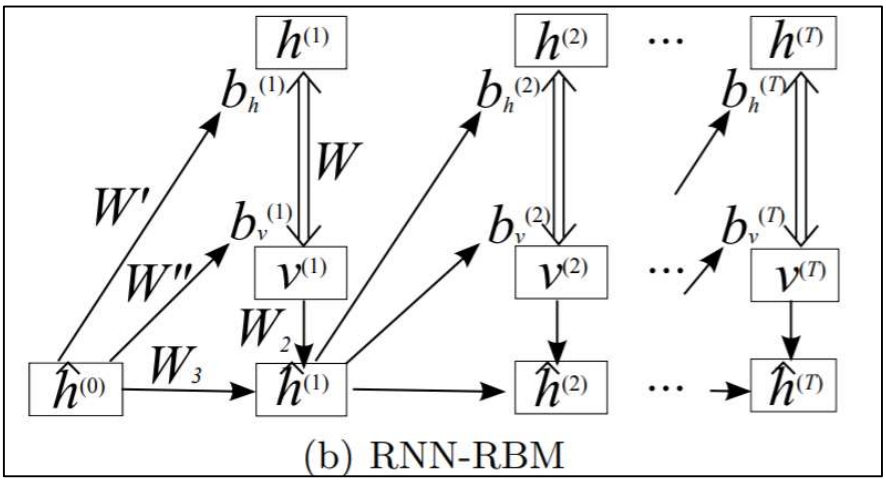

Figure 2-3 The structure of RNN-RBM model. each RNN hidden unit (lower-half of the diagram) is combined with an RBM (upper-half of the diagram)

2.1.3 Generative adversarial network

A Generative Adversarial Network (GAN) (Goodfellow et al., 2014) is a type of neural network that produces plausible data from scratch. There are two systems in a GAN – the generator and the discriminator. The generator tries to fool the discriminator by producing fake data while the discriminator distinguishes real and fake samples. By competing, the discriminator penalizes the generator for generating fake data, and the generator gets better at creating new instances that look real.

A study by Dong et al. (2017) adopted Convolutional GAN with Wasserstein Loss to generate four bars of multi-track piano-rolls. The input data size is set as 96 (timesteps) times 84 (notes) times 5 (tracks) in piano-roll format for the model training. Akin to most image classification problems, each note on the piano-roll is considered a valid pixel indicating a note's presence. Therefore, using convolutional layers combined with GAN's concept, the system is like generating images from scratch then transforming them back into audio formats. The study showed that the method is feasible and can learn the structures and characteristics of short chord progressions. However, it is incapable to produce music for longer timesteps.

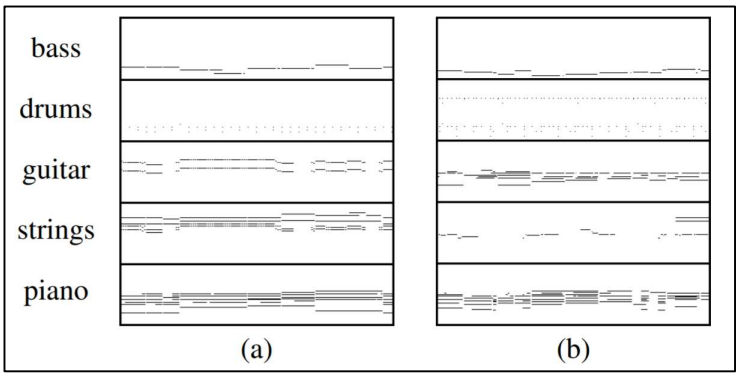

Figure 2-4 An example of the training data for the Convolutional GAN model. There are two instances (a & b) of music fragments of four bars with five tracks.

2.2 Proposed Methods

This project will evaluate two neural network architectures – the Deep Bidirectional LSTM model and a GAN model with convolutional layers. It is hypothesized that the Deep Bidirectional LSTM model could improve shortcomings of existing studies by predicting musical structures with longer timesteps and more meaningful melodies. On the other hand, the GAN models try to solve the LSTM models' problem of getting stuck in a loop (generating the same musical structure) by developing music from scratch (random noise). Both proposed models are supposed to tackle the drawbacks of current studies, and all results will be compared and evaluated by appropriate evaluation metrics – the Fréchet Inception Distance (FID) score and nearest neighbor search – and human survey.

3. Design, Solution, and System

3.1 Deep Bidirectional LSTM RNN

3.1.1 Deep neural networks

It is believed that a deeper neural network is more capable of analyzing and processing thorough characteristics of information than a shallow one. Typically, a shallow neural network contains only one or two hidden layer (s), while a deep neural network consists of more than two hidden layers. Therefore, to let the proposed model have a better understanding of input data, a deep neural network is adopted in this project.

3.1.2 LSTM Architecture

Each LSTM layer contains blocks of LSTM cells with multiple gates to memorize sequences of information. The cell states transfer valuable information to other cells and multiple gates with unique functions to keep the LSTM network learning and memorizing information. The below figure illustrates the architecture of an LSTM cell.

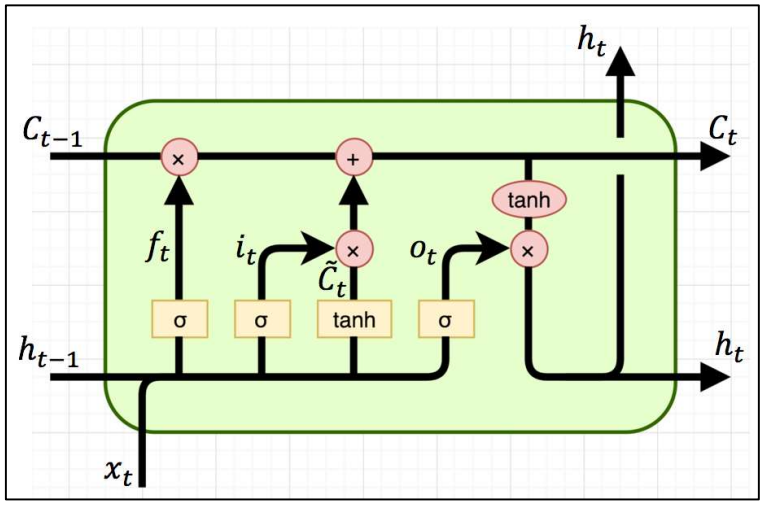

Figure 3-1 An LSTM cell

There are three types of gates in a LSTM cell: a forget gate, an input gate, and an output gate. First, the forget gate will abandon useless information before passing them to future cells. The formula of the forget gate is as follows:

$$f_t = \sigma\big(W_f * [h_{t-1}, x_t] + b_f\big)$$

Second, useful information will be decided by the input gate (i_t) and a candidate cell state ($\tilde{C}_t$) will also be created for regulating the current cell state. The equations are as follows:

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = tanh(W_C * [h_{t-1}, x_t] + b_C)$$

Now, the output of the forget gate, input gate and the candidate cell state are ready, an updated cell state can be derived as follows:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, the output gate will determine the hidden state (h_t). The hidden state is calculated as follows:

$$h_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) * \tanh(C_t)$$

The above equations can be written in python pseudocode as follows:

```
def LSTM(c_t_prev, h_t_prev, input):
    combine = h_t_prev + c_t_prev
    f_t = forgetGate(combine)
    i_t = inputGate(combine)
    c_t_candidate = tanh(combine)
    c_t = c_t_prev * f_t + c_t_candidate * i_t
    h_t = outputGate(combine) * tanh(c_t)

c_t = [array of c_t]
h_t = [array of h_t]

for input in allInput:
    c_t, h_t = LSTM(c_t, h_t, input)
```

Figure 3-2 The algorithm to calculate LSTM's next hidden state

3.1.3 Bidirectional LSTM

A bidirectional LSTM model is a system containing two LSTM layers that pass-through data in opposite directions. Each output from the system receives information from both the hidden states of the backward and forward layers. Comparing with unidirectional LSTMs, bidirectional LSTMs not only learn information from the past but from the future as well. This feature allows the model

to predict sequential information more accurately. From the image below, we can see that it is an acyclic graph. The output for each timestep can be derived as follows:

$$y_t = \sigma\left(W_y\left[\vec{h}_t, \overleftarrow{h}_t\right] + b_y\right)$$

Here, $\vec{h}_t$ denotes the hidden state at timestep t of the forward LSTM layer. On the other hand, $\overleftarrow{h}_t$ represents the hidden state at timestep t of the backward LSTM layer. σ means the activation function for the output y_t.

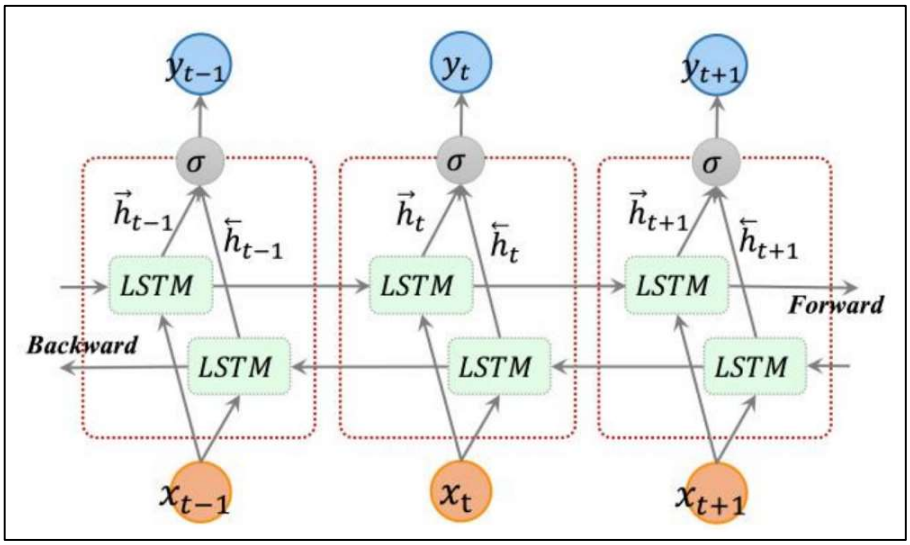

Figure 3-3 An illustration of Bi-LSTM architecture

3.1.3 A combined approach

A Deep Bidirectional LSTM RNN is a model of multiple bidirectional LSTM layers stacked together. It is hypothesized that combining deep neural networks and the feature of bidirectional layers will perform better than methods done by previous studies. Therefore, the first model in this project will be based on this design.

The model contains three bidirectional LSTM layers with 128 neurons each, followed by two fully connected layers to modify the input vectors' dimension and adapt the desired output shape. The initial plan for the activation function and the optimizer is to use Rectified Linear Unit (ReLU) and RMSProp since they are most widely used for LSTM models. However, other optimizers and activation functions will be further tested during the implementation steps.

3.2 CNN based GAN

A GAN is made up of a generator and a discriminator with any neural network architecture that is best for classifying the training data. It is mostly used for generating images with CNN. However, other neural networks like LSTM or RNN have also been used to predict sequential data. In

general, the training process for GAN can be divided into two parts – training of the discriminator and training of the generator.

First, randomly generated data will be fed into the generator. The randomly generated input size does not necessarily need to be the same as the desired generated output. Since the dimension of the input data can be changed in the generator, usually a smaller size than the dimension of the output data will be used. After the generator has produced a fake (generated) output, the fake output will be passed into the discriminator to be compared with the real data. The discriminator, at this point, will learn to distinguish data between real and fake throughout all iterations of the training process according to the discriminator loss function. The generator will also learn from gradients obtained from backpropagation through both the discriminator and the generator according to the generator loss function connected to the discriminator only. The figure below is a general GAN architecture (The "Real images" represents real-world data).

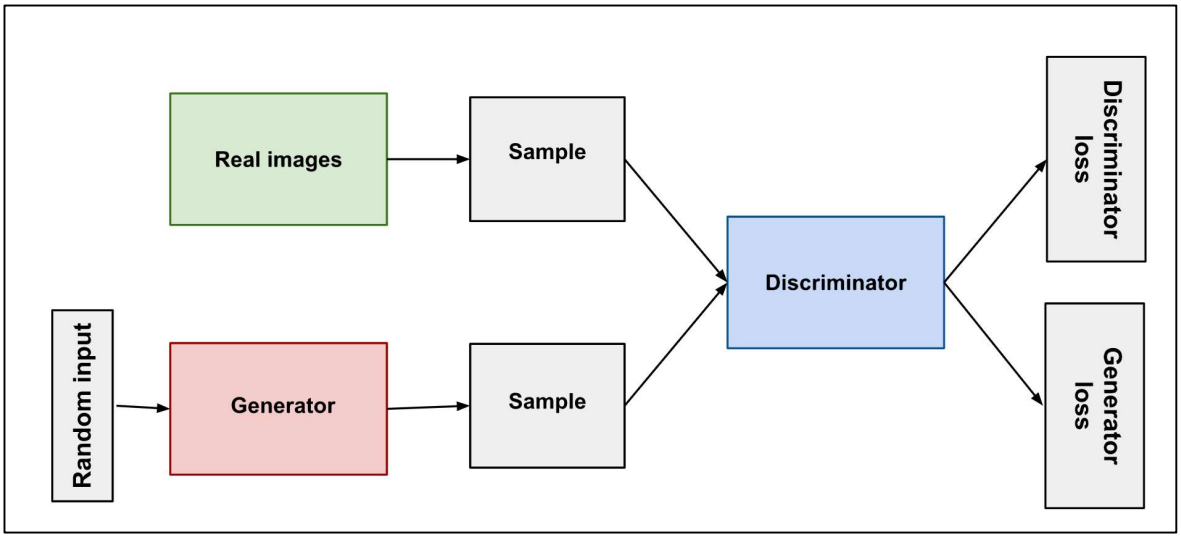

Figure 3-4 The main architecture of a GAN model

This model utilizes convolutional layers as the main neural networks for both the discriminator and the generator. Since a midi file is a file with different frequencies that progress with time, each midi file can be transformed to a 2-D array where the x-axis denotes the timesteps, and the y-axis represents different pitches of notes. In this case, we can treat the 2-D arrays as images to be trained in a convolutional neural network-based GAN model easily.

The generator consists of a fully connected layer with many neurons followed by four convolutional layers that gradually modify the shape of the data vectors into the desired data shape. The final output shape of the generator will be the shape of the generated music. Here, we set the output shape to be 1000 times 84, where 1000 represents the number of timesteps in the music

and 84 stands for the number of pitches. It is worth noting that the number of pitches is set to be 84 in order to save computational cost. Since the notes and chords of almost all pieces fall between in midi index 24 to 108, which contains in total 84 notes, it is better to truncate the unnecessary 0s to speed up the training process. After the training and generating process, the generated pieces will then be added two arrays of 0s to become arrays of size of 1000 by 128. The kernel size for each convolutional layer is 5 by 5, with strides of either (2, 1), (1, 2), or (2, 2) to fit the final image shape. The last convolutional 2D transpose layer uses tanh as the activation function to determine the final output of each neuron.

The discriminator contains two convolutional layers and one fully connected layer. Each convolutional layer has a kernel size of 5 by 5 and a stride of (2, 2), which is similar as the generator's settings.

3.3 One-hot encoding on Midi files

One-hot encoding is the process of transforming categorical information into binary form, indicating that during a specific timestep, only that information has a higher value (1) than the values (0s) of other irrelevant information. For instance, in specific timestep t, note "C4" is played, then the data vector on timestep t will contain a one at a specific position in that vector that corresponds to the note "C4" with 127 0s considering 128 unique notes. Since the notes and chords are stored categorically in all midi files, all such information will be processed through one-hot encoding for better model training performance. Both neural network models will be trained with the same set of training data using this technique.

3.4 Evaluation metrics

3.4.1 Fréchet Inception Distance (FID)

The Fréchet Inception Distance (FID) is designed for evaluating the generated results (fake data) produced by the GAN models with the real data. The equation to calculate the evaluation score is shown below.

$$FID\ score = d^2 = \|\mu_1 - \mu_2\|^2 + tr\left(C_1 + C_2 - 2\sqrt{C_1 C_2}\right)$$

Here μ_1 and μ_2 denotes the mean of all feature vectors of the true and fake data, respectively. Similarly, C_1 and C_2 denotes the covariance matrix of all feature vectors of the true and fake data, and tr denotes the trace linear algebra operation. A lower FID score means the generated results are more similar to the real data, which is desired in most cases. On the other hand, a higher FID score signifies that the generated results are less like the real data.

3.4.2 Pitch Histogram

Note consistency and note diversity can be visualized by plotting the pitch histogram. A pitch histogram is plotted by summing up each unique frequency count of notes and chords in a histogram style. It is believed that it is an effective measure to classify different genres of music. In this project, it can be used as a tool to distinguish the eras among all generated pieces of music by all models.

3.4.3 Nearest Neighbor search

A way to check whether there is any similarity between a generated piece and the pieces from the training set is by searching the nearest neighbor of that generated piece. By calculating the root mean square error between that generated piece and all pieces from the training set, we should retrieve the closest piece by searching for the lowest error.

4 Methodology and Implementation

4.1 Midi files Pre-processing

This project uses the Music21 and Pypianoroll python libraries to extract information from all midi files into notes and chord sequences. Their pitches and octaves correspond to each time step. Since this project's scope only covers a single instrument's generation, the training data will only contain pieces of music played by the piano.

All training data is acquired from the GiantMIDI-Piano repository on the ByteDance GitHub repository. According to their documentation, there are 10854 midi files of classical music available. However, duplicates and empty files were found in the acquired dataset after the data cleaning and categorization process. The final dataset contains 546 pieces of baroque music, 412 pieces of classical music, 637 pieces of romantic music, and 238 pieces of modernist music.

Pieces of music from each four-category are then processed independently. Extracting and concatenating all notes and chords, the program will form a dictionary that records each unique note and chords (a combination of notes) with an index. For instance, a note "A2" is paired with the index "0", and a chord "C#3" is paired with the index "1". It will simplify the process of one-hot encoding later.

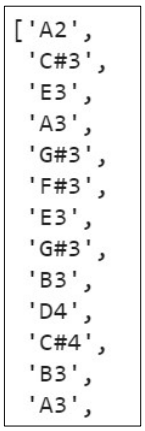

Figure 4-1: An excerpt of unique notes and chords set from the baroque music dataset.

Since LSTM models learn from given sequences and their next prediction(s), an additional process is needed to build series of sequences and their corresponding output (next forecast). This project sets the sequence length as 50, meaning that the LSTM network learns to predict the next note given its previous 50 notes or chords. First, the program will store the first 50 notes or chords, then transform them into integers according to the unique note-chord dictionary built in the previous into the input sequence array. The next corresponding note or chord of the last sequence

is then stored in the output array. Retrieving all sequences of notes and chords, each element in the output array is normalized between 0 and 1 for the later training process's ease.

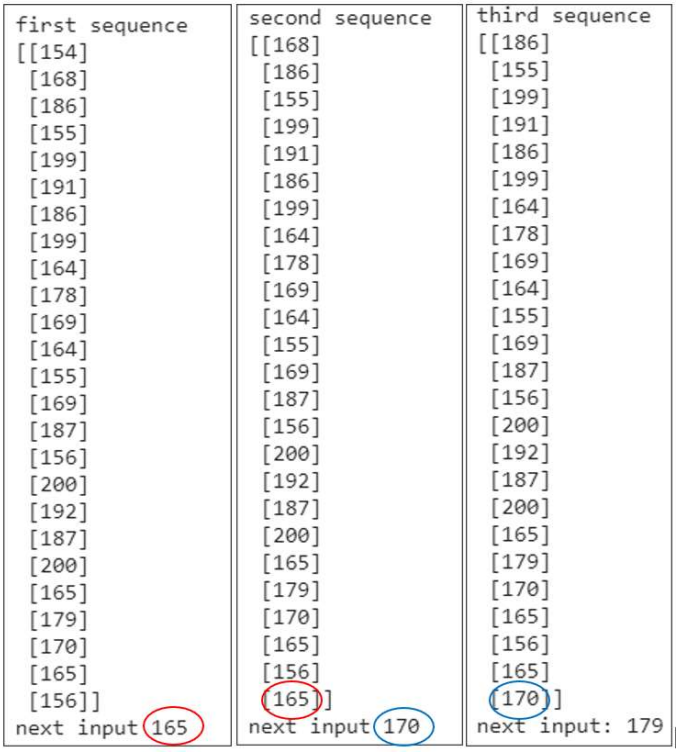

Figure 4-2: An example of sequence of notes and chords stored in arrays and their corresponding outputs in integers format.

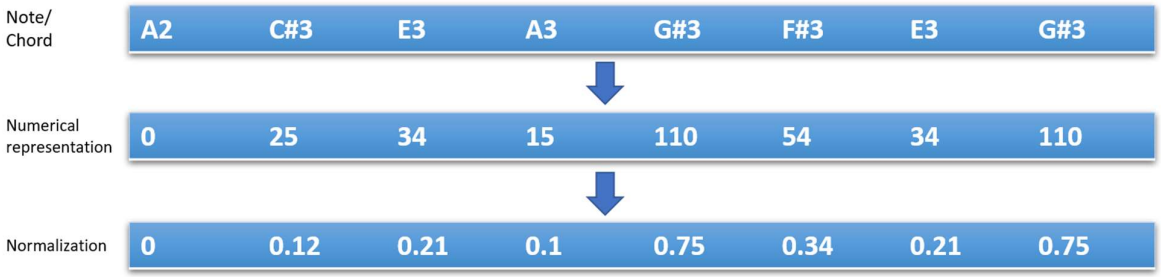

Figure 4-3: The process of converting sequences of notes and chords into normalized sequence.

Treating midi files like images, the CNN-based GAN model can efficiently generate music with reasonable structures both in vertical and horizontal perspectives. The input training data contains 2-D arrays, which can be considered as arrays of images with only two colors. The x-axis represents timesteps, and the y-axis denotes different pitches of notes (and chords). Suppose a note or chord occurs in any timestep, the array of that particular timestep store a '1'. On the other hand, empty notes are denoted by a '-1'.

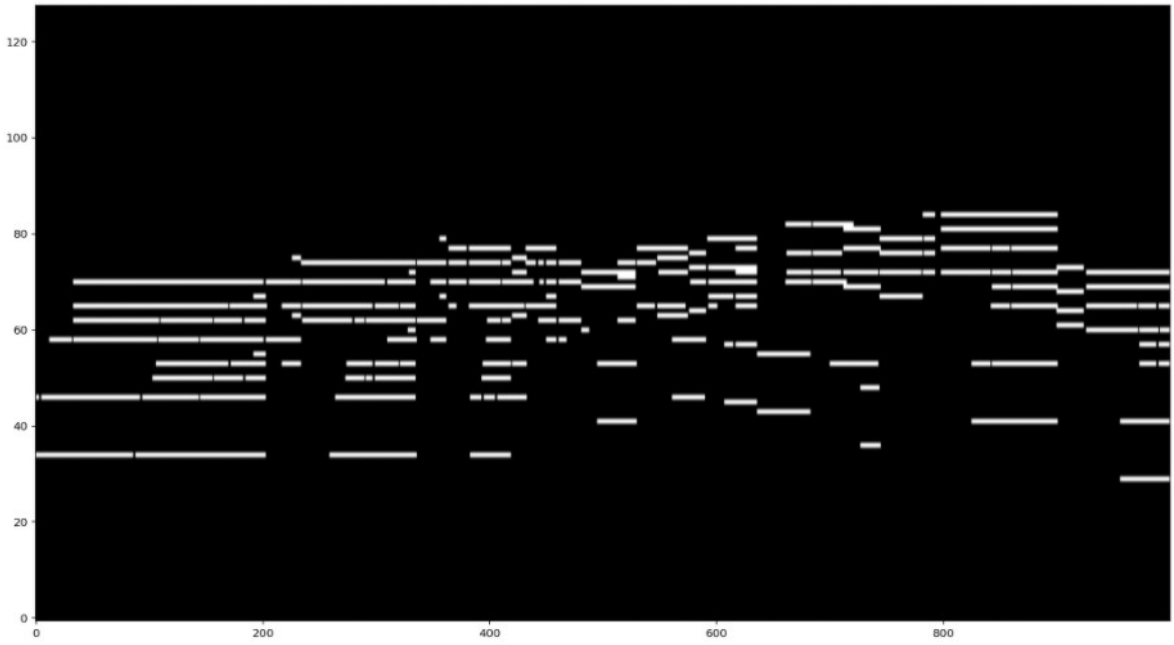

Figure 4-4: An example of a piece of music represented in 2-D array image format (Valse carnavalesque, Op.73 by Chaminade, Cécile). Here, white dots or lines indicate a presence of notes.

The final step is to store all images into a NumPy array with a shape of (x, y, z) where x denotes the number of unique pieces of music, y indicates timesteps of each training piece, and z means the range of notes. In this project, the array shape setting is (x, 1000, 84) for simplicity since the objective is to assess the CNN-GAN model's capability to generate structured music and meaningful melodies instead of generating long pieces that usually require excessive computing power to finish the task. In standard Midi expression, there are 128 notes in total. However, by observing all images from the training set, most of them only have notes span from minimum C1 (index 24) to maximum C7 (index 108). This means that notes below index 24 and above index 108 can be ignored in that they are empty notes which are not conducive for the latter training process. Hence, a trimming process is done for all images before training. It results in the final input data shape of (x, 1000, 84), where x denotes the number of unique pieces of music.

Figure 4-5: Trimming the image above to have a range of notes from C1 to C7.

4.1 Model Building

In this project, all neural networks will be programmed in python using the TensorFlow library and its higher-level API - Keras API. The high-level Keras API allows faster implementation of any neural network. In general, only the input shape of the training data, loss functions, optimizers, and some specific hyperparameters are required to be provided by the developer.

4.1.1 Deep Bidirectional LSTM RNN

The current implementation for this model is to stack one bidirectional LSTM layer and two LSTM layers together. They are followed by two fully connected layers. Each LSTM layer contains 256 nodes with a dropout rate of 0.3 afterward. After the last LSTM layer and the first fully connected layer, batch normalization is added. The first fully connected layer contains 128 nodes with a rectified linear activation function (ReLU). The final layer contains x nodes where x is determined by the distinct notes or chords from the training corpus. Since it is known that if a note or chord is not within the training corpus, it will not show up or be predicted in the future by the neural network, we can safely implement the final fully connected layer in this manner to save memory and computational time. The model is compiled with a loss function of cross-entropy loss and RMSprop for its optimizer.

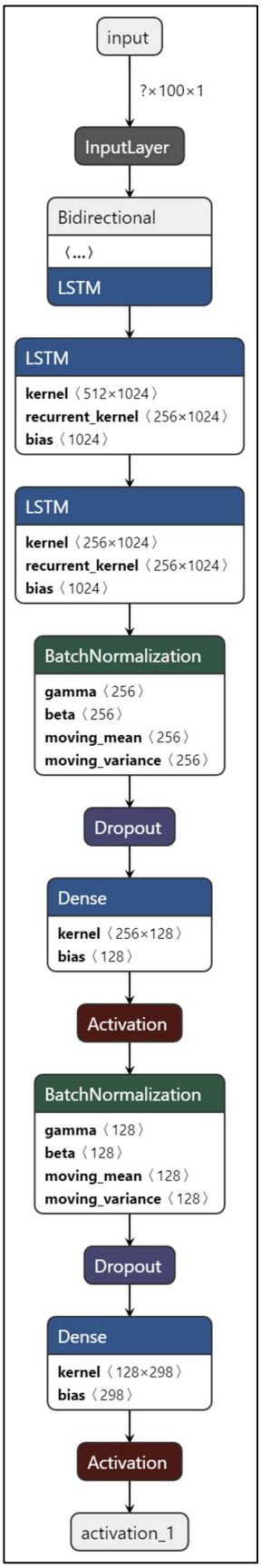

Figure 4-6 The Bi-LSTM network structure

Similar to the training step, the prediction network possesses the same architecture as the training model. The only difference is that the optimal weights from the last epoch of the training process are added to the neural network for sequence prediction.

A random number is generated as an index determining the starting point of the prediction sequence to predict a sequence of music based on the training set. The length of the initial sequence of notes is also determined beforehand. With the above settings decided, the model can predict a single note or a chord (a set of notes) in each iteration of the prediction.

In each iteration, the model predicts the next note or chord of the input sequence by choosing the highest probability of that element (note or chord). That note or chord with the highest likelihood of being the next element of the sequence will then be concatenated to the previous sequence. At the same time, the first element of that sequence will be removed before the next iteration. This allows the window size (length of sequence) to be the same as the window sliding from the beginning of the prediction to the end.

Figure 4-7: generating 2 notes in 2 iterations. The first row (sequence) is the initial sequence determined by the random index. Here, B3 is the first note being generated, and D4 being the second. The process goes on until reaching the maximum number of notes (iterations) specified.

4.2.2 CNN based GAN

This CNN-GAN model consists of 2 neural networks – a discriminator and a generator. Both networks adopt cross-entropy loss as their loss function by computing the difference between predicted labels and true labels. Since the generator is trying to fool the discriminator by generating fake images that make it the discriminator challenging to discern the authenticity, the generator's loss function compares the discriminator's decisions on fake images to an array of ones. As for the discriminator, there are two steps (two losses). It compares real images with an array of ones and fake images with an array of zeros. Combining both losses yields the total loss of the discriminator's loss.

```
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

Figure 4-8: the implementation of the generator's loss

```
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

Figure 4-9: the implementation of the discriminator's loss

The generator's network structure begins with introducing a small random seed as input to a fully connected layer with a much larger output size. After reshaping the output data, four transposed convolutional layers are stacked together to upsample the data. Each convolutional layer has a kernel size of 5 (both height and width) and a stride of (2, 1) or (1, 2), or (2, 2) to upsample the data to meet the same shape of the input data which is (1000, 84). The paddings are the same to preserve the original size of the data. Finally, except the last layer uses tanh as an activation function, all layers are followed by a LeakyReLU activation layer.

The discriminator, on the other hand, has a shallower layer of neurons compared with the generator. It has two convolutional layers at the beginning and each with a five-by-five kernel and a stride by 2. Finally, a fully connected layer with a single output determines whether this very input image is from the training set (real) or the generator (fake).

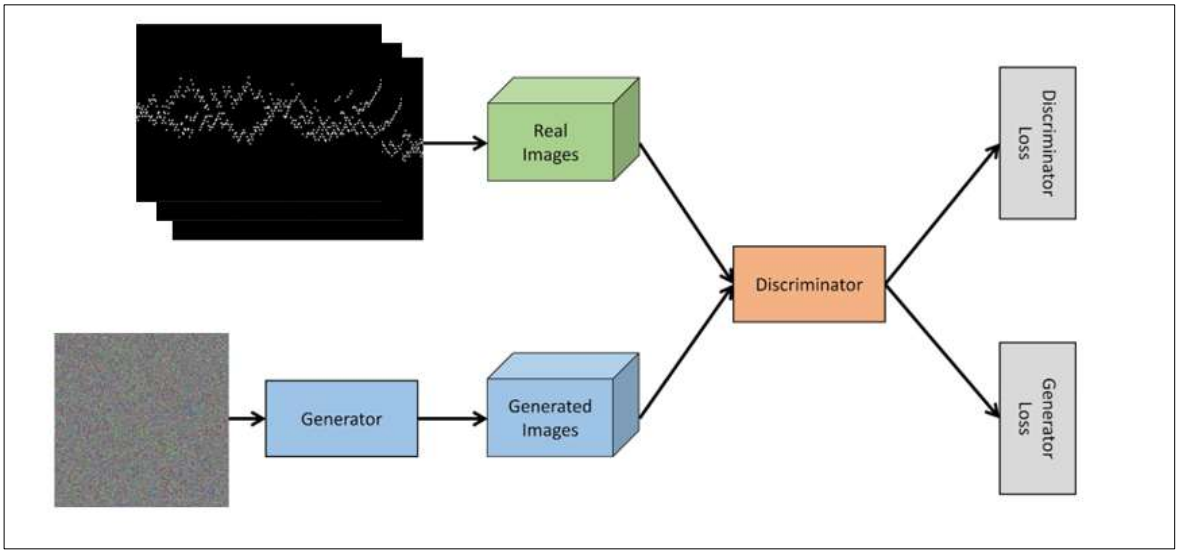

Figure 4-10: The CNN-GAN model structure (The Generator is fed in with some noise)

4.2 Data Conversion

For the Bi-LSTM model, after a new sequence of numbers is generated, all elements from the sequence will first be converted to the representation of either notes or chords accordingly by referencing the unique set of notes and chords from the training set, then be converted to either notes or chords in Midi format. Here, a limit of the note number and chords need to be specified, and so does the tempo. Take modernist music as an example: a maximum of 300 notes and an offset of 0.3 between notes are good settings. Here, the offset controls the tempo of the music. When the number is larger, the tempo is slower, and vice versa.

For the CNN-GAN model, the final step is to convert the array with the size of (1000, 84) back to the size of (1000, 128), the acceptable data format by the Pypianoroll library, then convert it to a midi file. The predicted array concatenates an empty array with a size of (1000, 24), and the combined array is concatenated by the other empty array of size (1000, 20).

4.3 Evaluation

4.3.1 Pitch histogram

The purpose of using pitch histograms to analyze generated music is that this graph is an efficient way to visualize note similarities and diversities from a given piece of music. By counting the most frequent notes occurring in a piece, it is common to determine the music. However, suppose most notes fall in only a few (1 or 2) categories of note. In that case, it may indicate a failure of the generation of music since almost most pieces have a specific diversity of distribution of notes.

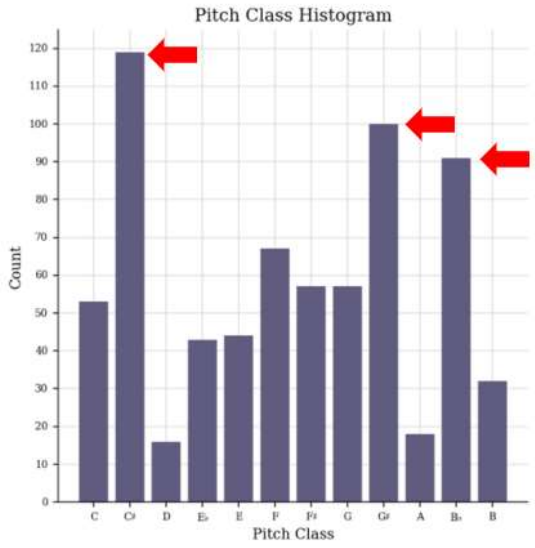

Figure 4-11: A good example of a generated piece in classical style with great diversity of notes. There are three main keys pointed by red arrows.

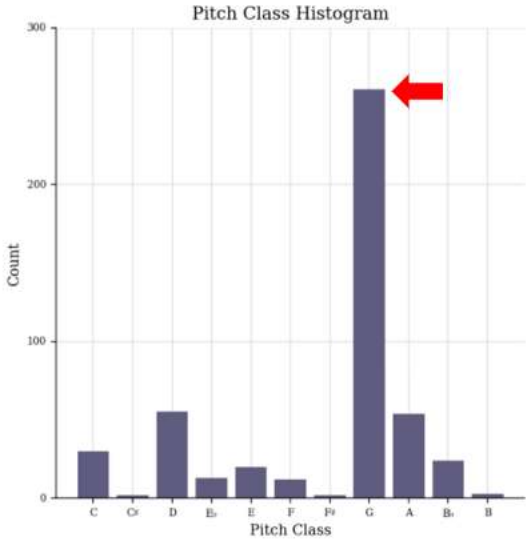

Figure 4-12: An example considered as a failure. It is a generated piece in baroque style with a few training epochs (20 epochs). This indicates that the model did not learn much from the training process and could only produce music with repeated notes. The dominant key is pointed by the red arrow.

4.3.2 FID

There are two required components to calculate the FID scores – the set of real images and the generated image. For each genre, all music from the training set belongs to the 'real' image set, and the generated image belongs to the 'generated' image set. The features that will be extracted for both the 'real' and 'generated' sets are part of the images themselves, and only a segment of each piece will be extracted. The segments are images with a shape of (400, 48), where 400 denotes timesteps and 48 means the range of notes. The reason to set the range of notes to 48 is that from observation, most notes and chords fall in this range, and it saves a lot of computational costs to retrieve features from each piece of music from the training set. Next, the mean of all feature vectors and the covariance matrix of all feature vectors for the 'real' and 'generated' set are first calculated. The FID score can then be retrieved from the below equation.

$$FID\ score = d^2 = \|\mu_1 - \mu_2\|^2 + tr\left(C_1 + C_2 - 2\sqrt{C_1 C_2}\right)$$

```python
def calculate_fid(real, generated):
    # calculate mean and covariance statistics
    mu1, sigma1 = real.mean(axis=0), cov(real, rowvar=False)
    mu2, sigma2 = generated.mean(axis=0), cov(generated, rowvar=False)
    # calculate sum squared difference between means
    ssdiff = numpy.sum((mu1 - mu2)**2.0)
    # calculate sqrt of product between cov
    covmean = sqrtm(sigma1.dot(sigma2))
    # check and correct imaginary numbers from sqrt
    if iscomplexobj(covmean):
        covmean = covmean.real
    # calculate score
    fid = ssdiff + trace(sigma1 + sigma2 - 2.0 * covmean)
    return fid
```

Figure 4-13: The function to calculate FID score.

4.3.3 Survey

Two surveys were conducted to evaluate the results of the Bi-LSTM and the GAN model, and both are hosted on a third-party surveying platform – phonic.ai. Each survey consists of eight multiple-choice questions with only one answer to each question. The respondents need to play the audio provided on each page of the survey and choose whether the music they heard is composed by a human composer or an AI algorithm. If they feel that they cannot decide, they can select the 'I don't know' option. It allows the respondents to answer each question carefully and hopefully minimize their chance of guessing, which might deteriorate the survey results' quality.

One piece from each genre and each category ('real' an 'generated') is chosen, and only an excerpt (between 15 to 30 seconds) of that piece will be available to the respondents since letting the respondents listen to the whole piece in a survey will make them fatigue and therefore deteriorate the survey result. In each survey, the number of 'real' and 'generated' music is balanced and ordered randomly for fairness. The below snippet is an example of a question.

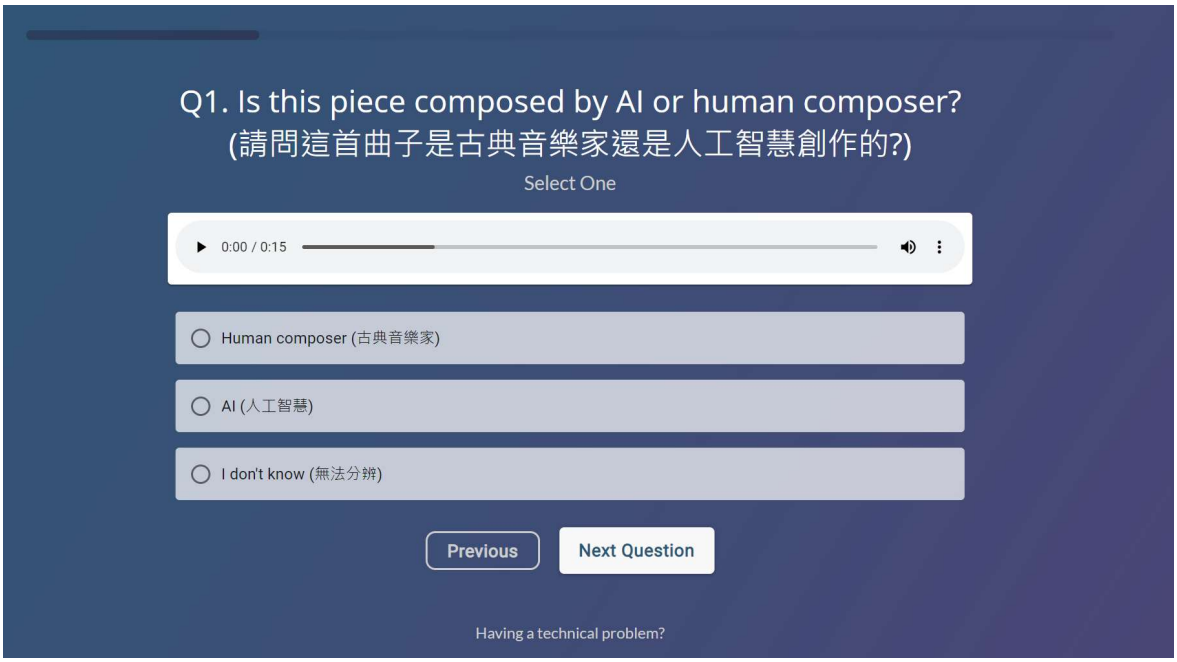

Figure 4-14 A snippet of one of the survey questions.

5 Results and Evaluation

5.1 Deep Bidirectional LSTM RNN

5.1.1 Note Diversity & Analysis

Below are four examples of generated music in different styles. Beginning with the Baroque, Classical styles and followed by the Romantic and Modernist styles. The results shown below demonstrate the Bi-LSTM model's capabilities to compose novel music passages by learning from the given training set.

- An example of a generated Baroque style music

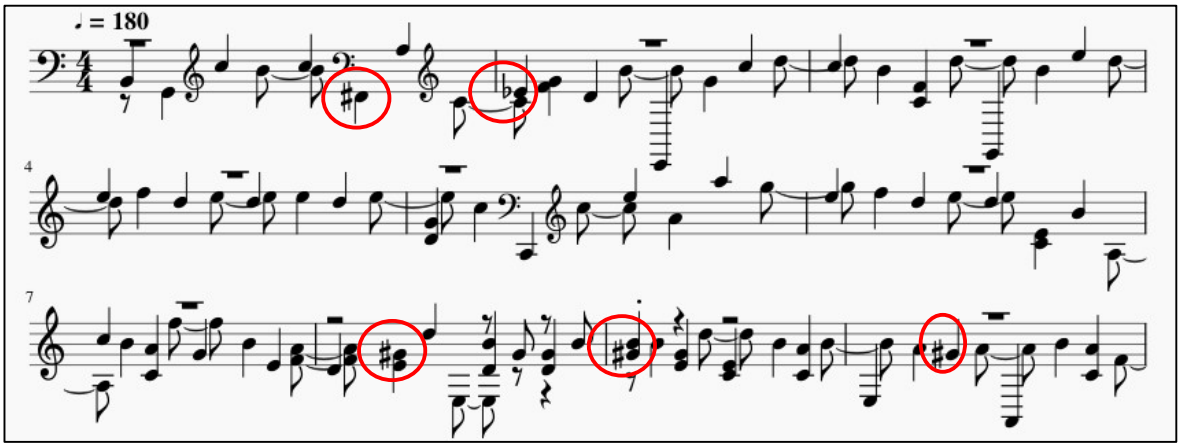

Figure 5-1: Baroque-1-Bi-LSTM

According to the pitch histogram (Figure 5-6 top left), it is evident that notes are distributed evenly in major scales, but there are not many sharps and flats. Though not until the late Baroque period did the key signature evolve to its present state, which usually has defined keys for each piece of music, the note diversity of this generated piece of music is relatively evenly, which is acceptable considering the nature of this genre of music. In addition, from the music sheet excerpt shown above, this piece of music shows signs of monody without complicated polyphonic musical structure.

- An example of a generated Classical style music

Music in this era often uses simple harmonic melodies to fulfill a balanced musical structure. This kind of composing idea was often found in Beethoven's music. The music generated below demonstrates a repetition of similar motifs (musical phrases or structures) that are waltz-like, which corresponds to one of the most common music composition ideas in the classical era. In my opinion, this very generated piece is one of the most successful one that resembles music from Beethoven.

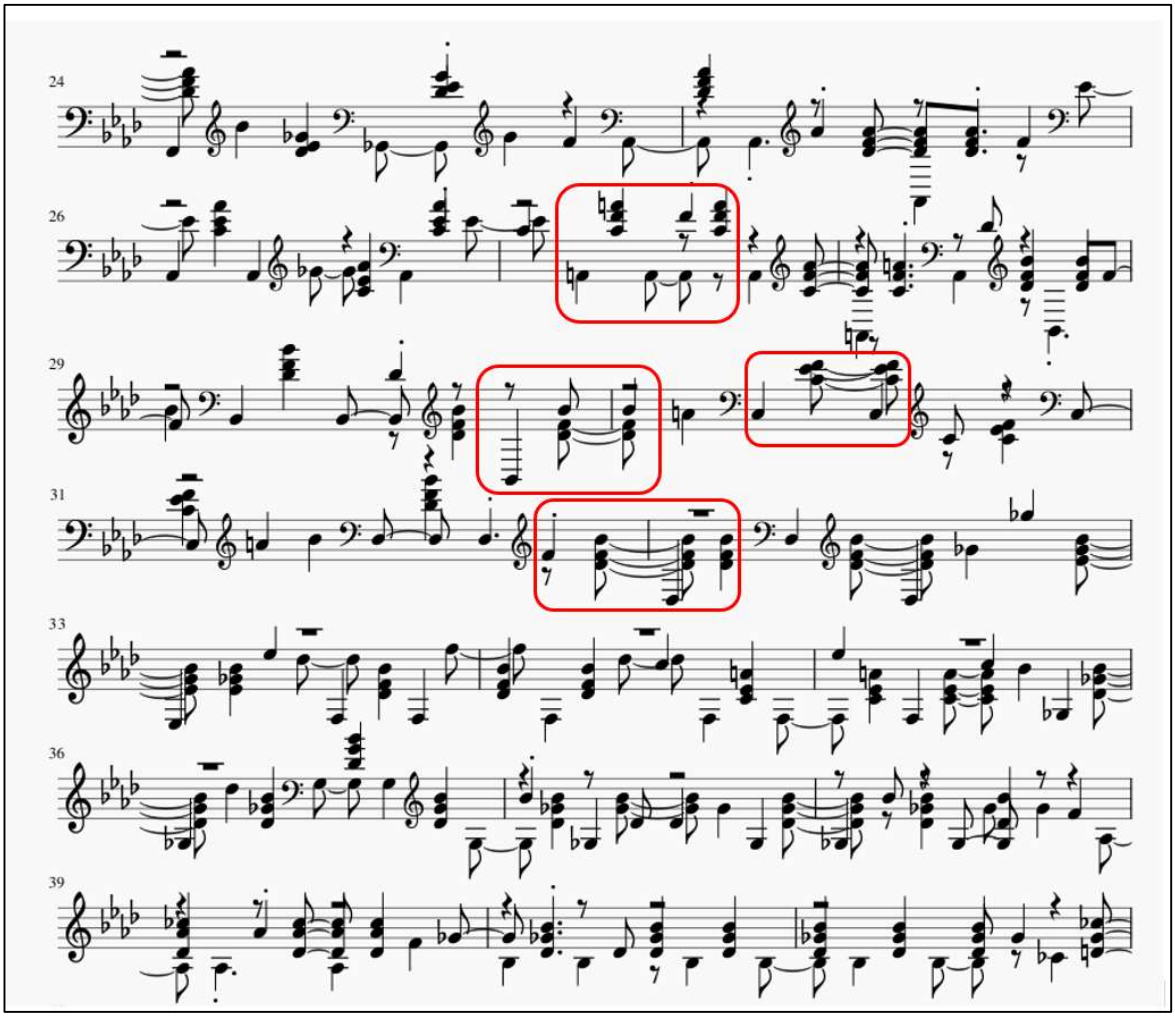

Figure 5-2 Classical-1-Bi-LSTM

- An example of generated Romantic style music

According to the Music 21 library analysis, the expected key for the below piece of music is C sharp minor with a confidence of 70.2%, which corresponds precisely to the pitch histogram below.

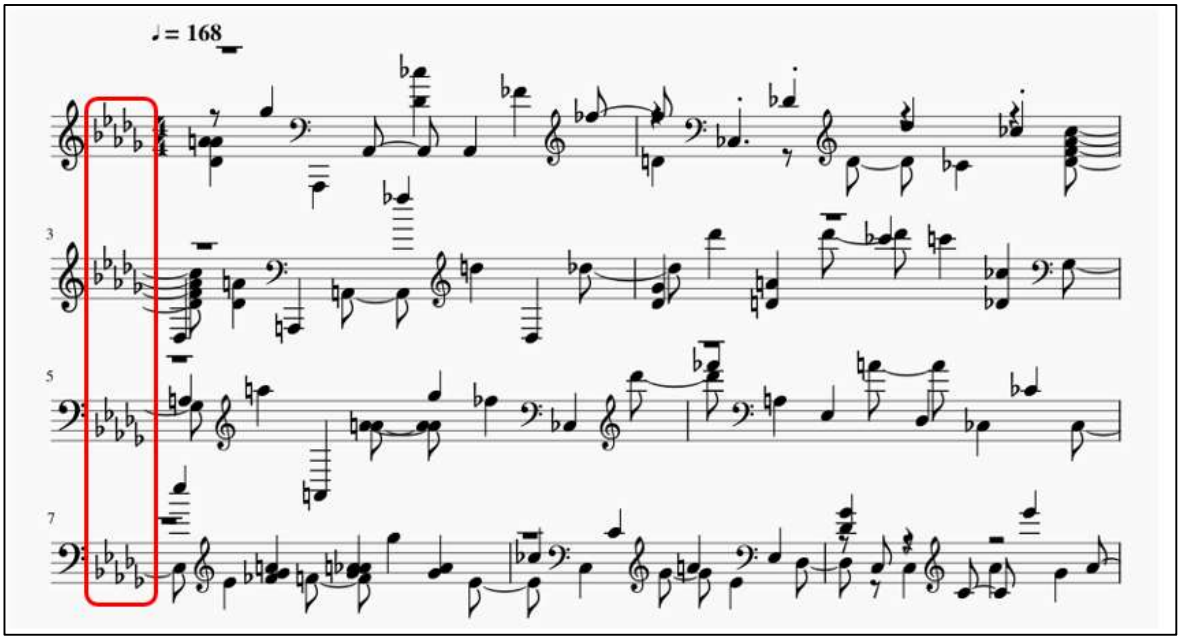

Figure 5-3 Romantic-1-Bi-LSTM. It is very common for a romantic style music to have multiple sharp and flat keys.

```
timeSignature = base_midi.getTimeSignatures()[0]
music_analysis = base_midi.analyze('key')
print("Music time signature: {0}/{1}".format(timeSignature.beatCount, timeSignature.denominator))
print("Expected music key: {0}".format(music_analysis))
print("Music key confidence: {0}".format(music_analysis.correlationCoefficient))
print("Other music key alternatives:")
for analysis in music_analysis.alternateInterpretations:
    if (analysis.correlationCoefficient > 0.5):
        print(analysis)
```

```
Music time signature: 4/4
Expected music key: c# minor
Music key confidence: 0.7027908535567045
Other music key alternatives:
C# major
```

Figure 5-4 Romantic-1-Bi-LSTM. An analysis on the piece shown above. It tries to predict its key with a confidence rate.

- An example of generated Modernist style music

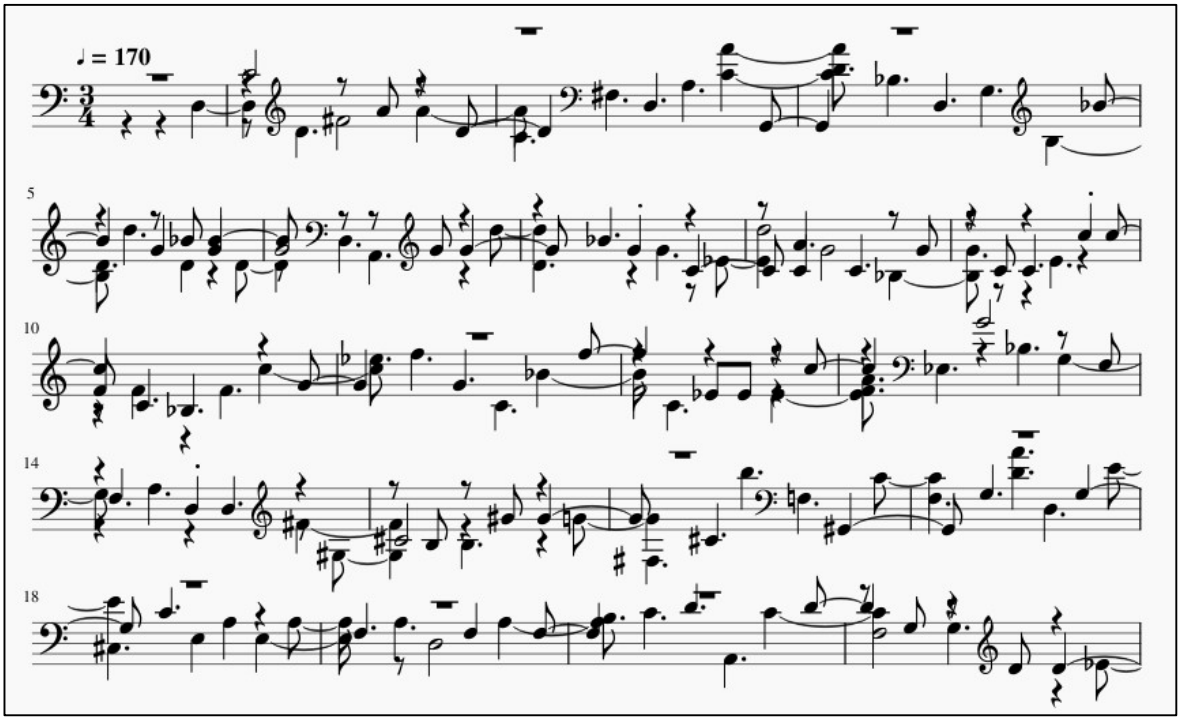

Figure 5-5 Modernist-1-Bi-LSTM

- Pitch histogram of four generated pieces

From the below figures, we can observe that all pieces have diverse distributions of pitches. No single pitch noticeably dominates the whole pieces. Thus, it can be concluded that the Bi-LSTM model handles the note diversity well and does not tend to memorize partial information from the training set.

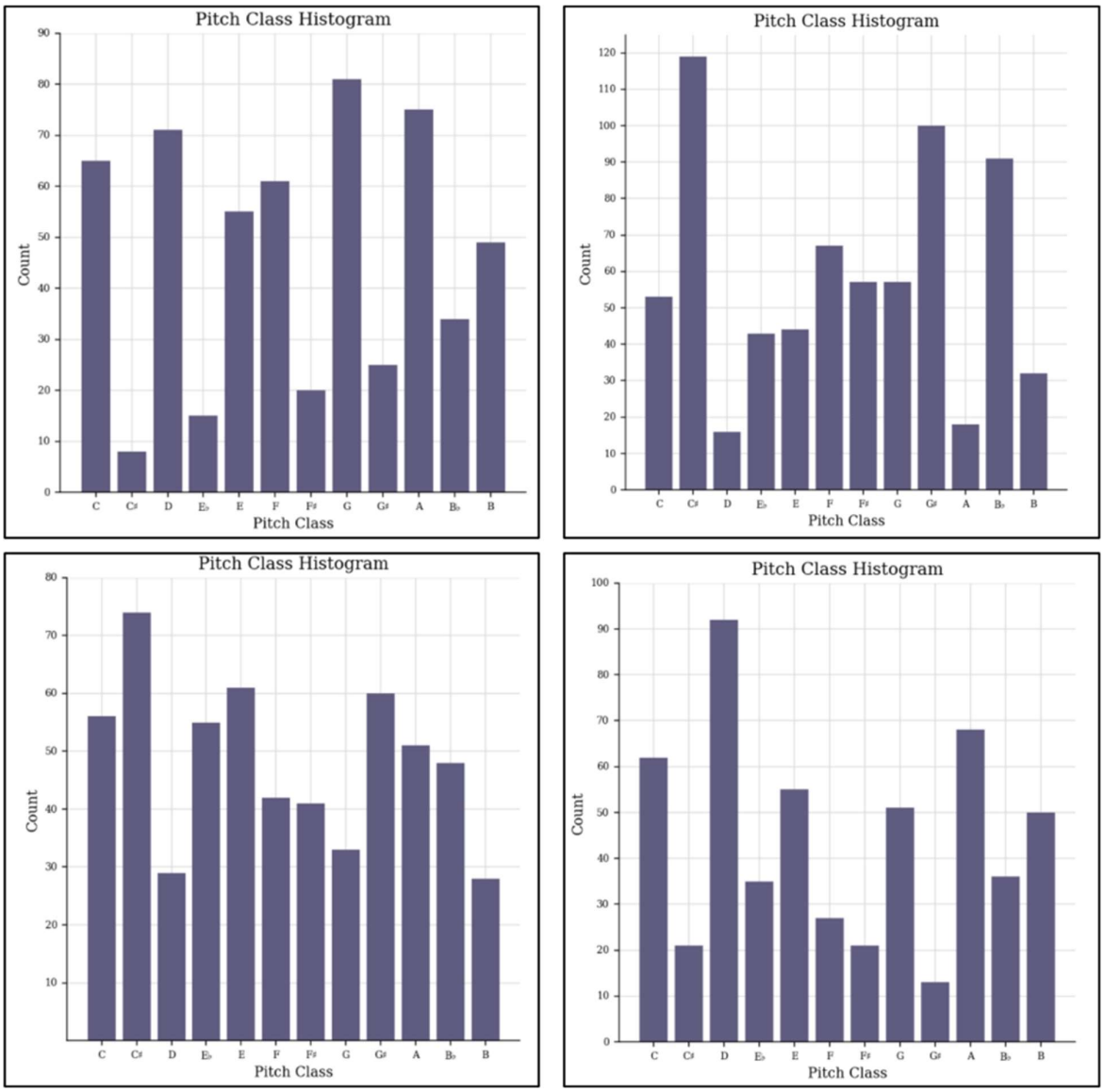

Figure 5-6 Four pitch histograms of the generated pieces from the Bi-LSTM models. Clockwise from top-left: Baroque-1-Bi-LSTM, Classical-1-Bi-LSTM, Modernist-1-Bi-LSTM, and Romantic-1-Bi-LSTM.

5.1.2 Survey

From the stacked column chart below, it is clear that over half of the respondents have difficulty in telling whether the given piece of music is composed by AI or human. Among all human-composed pieces, only half of which receives over fifty percentages of confidence indicating them to be composed by the AI. The results are the same for all pieces composed by the AI. Though some respondents regarded 'real' music as composed by AI, the statistics showing that most respondents picked 'Real' when they are given to listen to a piece composed by the AI. This indicates that the proposed Bi-LSTM model have the capability to trick humans' judgment.

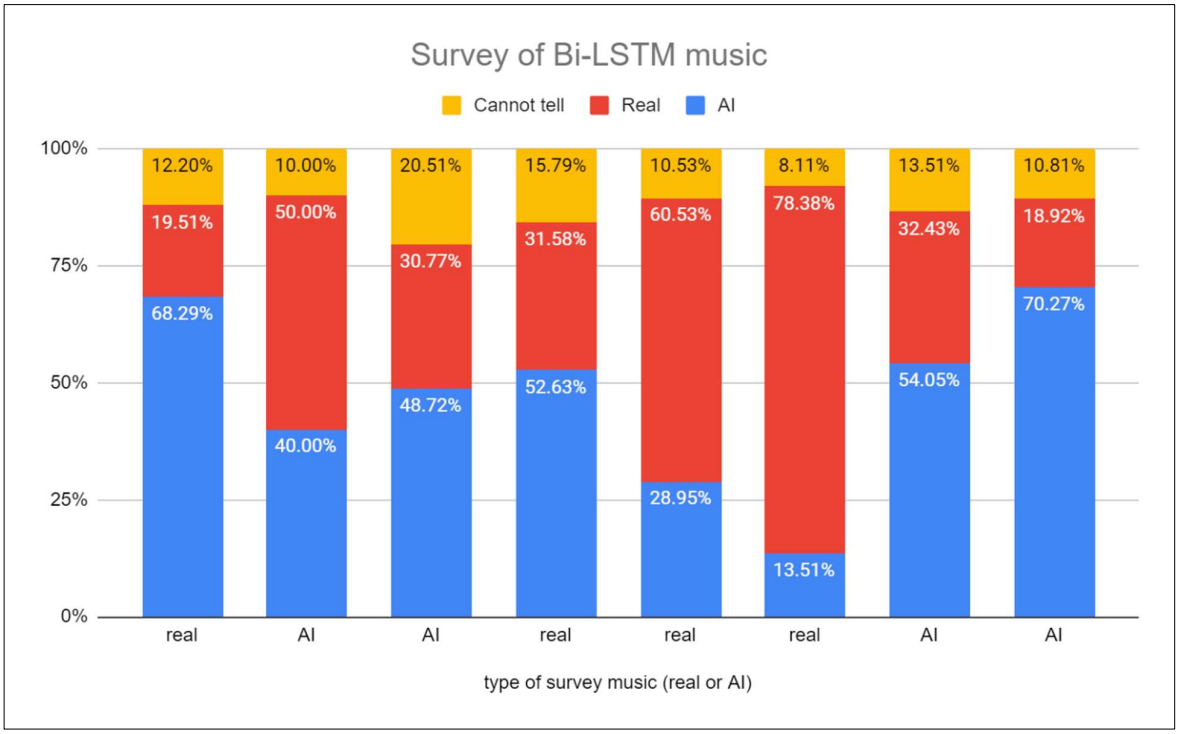

Figure 5-7 Survey of Bi-LSTM music

5.2 Generative Adversarial Network

5.2.1 Fréchet Inception Distance

The four generated pieces are evaluated by the FID score. In the below table, the first row, FID, is the FID score of the generated pieces and the training set, and the second row, FID (random), is the FID score of a random piece (image), which represents the initial states of the GAN training process, and the training set. Since the lower the FID scores, the similar the pieces (images) are to the training set, we can observe that each generated piece has a far lower FID score than the random one that corresponds to it. Therefore, we can say that the generated pieces look similar to those in the training set, which means that the GAN model mimics music from the training set well.

Major things that might affect the FID score could be the number of pieces in a particular training set, the piece itself, and the features that are being extracted for the FID evaluation. Also, unlike other image generating tasks which normally use the pretrained InceptionV3 model to evaluate generated image qualities, this project does not adopt this model as part of the FID score calculation due to the project's nature. The generated pieces are not exact the same as traditional images that have RGB color channels and have association with ImageNet labels. Hence, in this case, I consider the FID score as a reference of the model performance, instead of a hard standard to determine the goodness of a generated piece of music.

	Baroque-1-GAN	Classical-1-GAN	Romantic-1-GAN	Modernist-1-GAN
FID	2437.382	3596.256	4408.329	5939.203
FID (random)	20947.692	21013.249	20440.701	20610.990

5.2.2 Note Diversity & Analysis

Below are four examples of generated music in different styles. Beginning with the Baroque, Classical styles and followed by the Romantic and Modernist styles. The results shown below demonstrate the GAN model's capabilities to compose novel musical notes/ chords and structures by learning from the given training set.

- An example of a generated Baroque style music

Figure 5-8 Baroque-1-GAN (excerpt). Both the upper (boxed in red) and the lower (boxed in blue) staff has their own chord progression.

From the above *Baroque-1-GAN* music sheet excerpt, we can observe some impressive musical structures that are more complicated than the *Baroque-1-Bi-LSTM*. A noticeable feature is that when looking at a single measure, both the upper and the lower stave possess their melody, or at least the lower staff acts as the companion for the upper one. In contrast, the Bi-LSTM model does not showcase this kind of feature. Instead, it combines all chords or notes coinciding as a whole and represents them as a single chord. Though this might not simplify the musical structures, the GAN model might be a better option for generating more complex musical structures and producing more readable music sheets.

A way to check whether there is any similarity between a generated piece and the pieces from the training set is by searching the nearest neighbor of that generated piece. By calculating the root mean square error between Baroque-GAN-1 and all piece from the training set, we get the closest piece by searching for the lowest error. Below is a comparison between Baroque-GAN-1 (the generated piece) and the closest piece being found by the previously mentioned calculation. Observing the melody patterns of both pieces, we could say that both pieces have similar tendencies in developing continuous arpeggios. It is a good sign of showing the GAN model's learning capability, and not entirely copying the musical phrases from the training set.

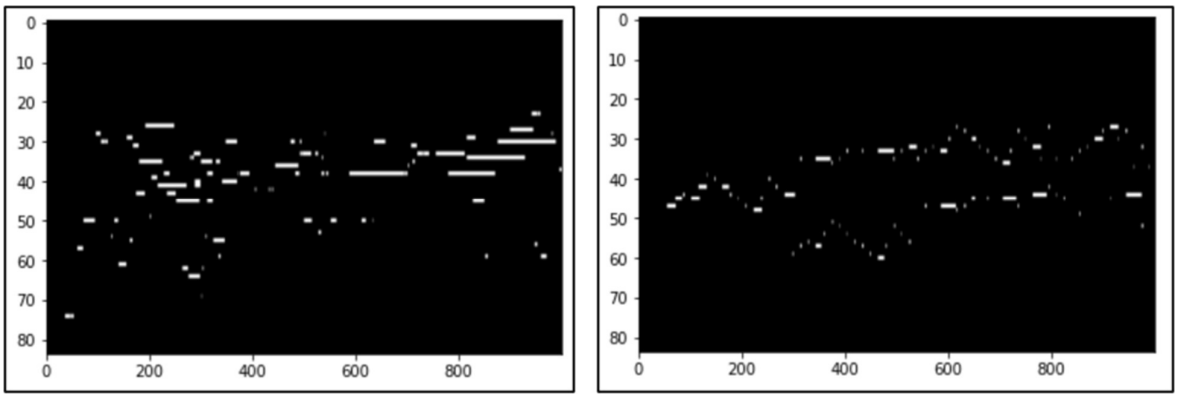

Figure 5-92 The nearest neighbor of Baroque-1-GAN. Left: Baroque-1-GAN; Right: a piece of music from the training set.

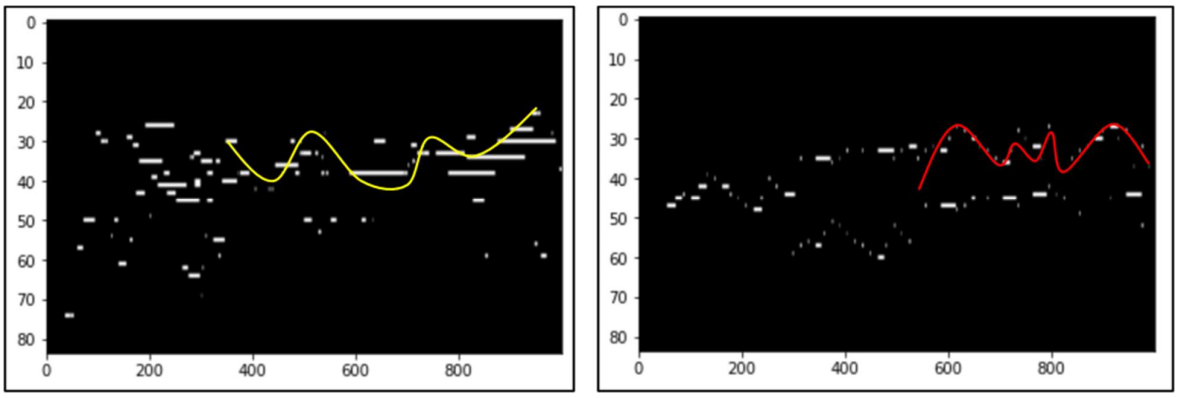

Figure 5-3 Both pieces have similar tendency in developing arpeggios going up and down continuously.

- An example of a generated Classical style music

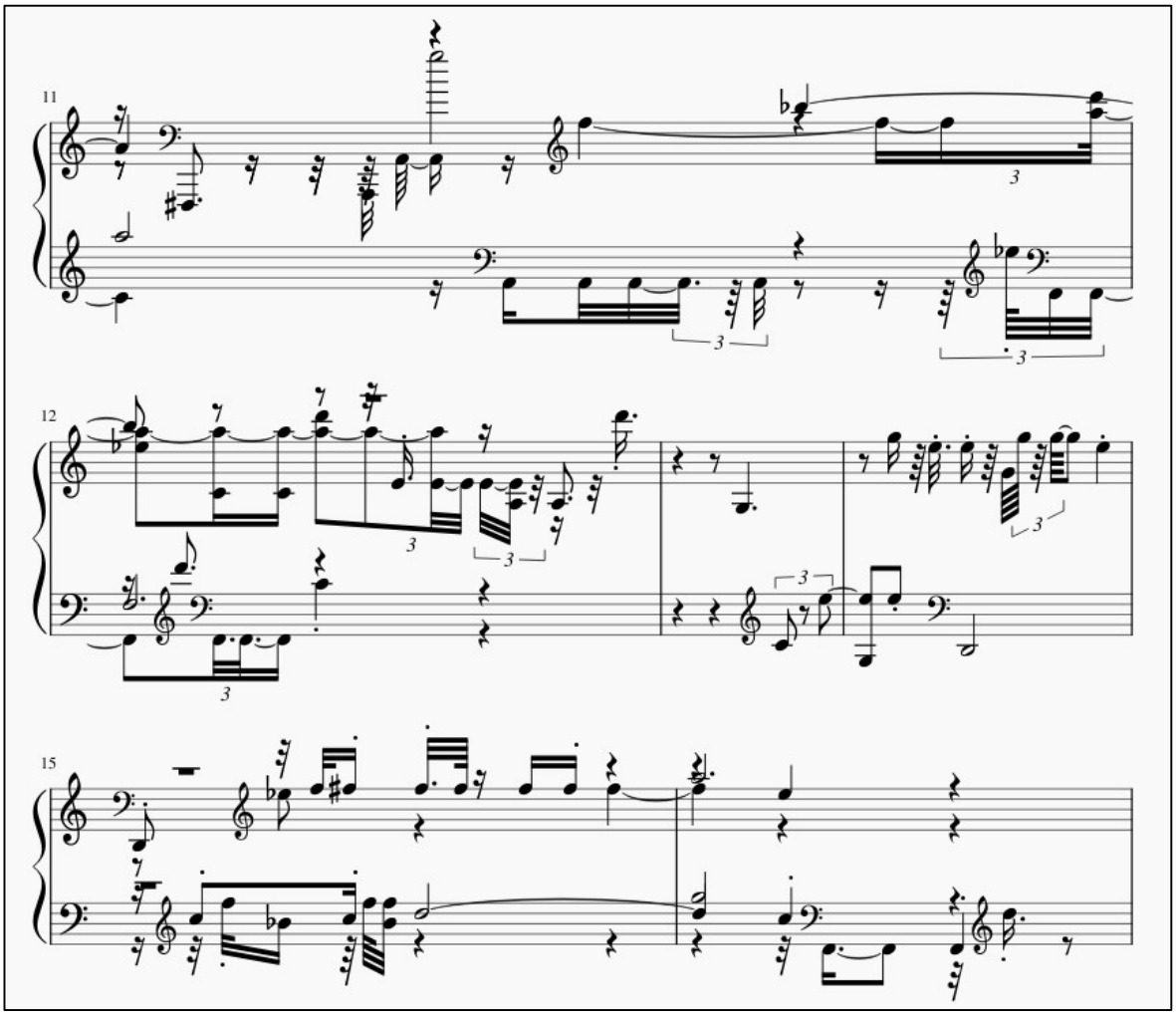

Figure 5-4 Classical-1-GAN

From the nearest neighbor search, it appears insignificant sign of similarity between Classical-1-GAN and its closest piece.

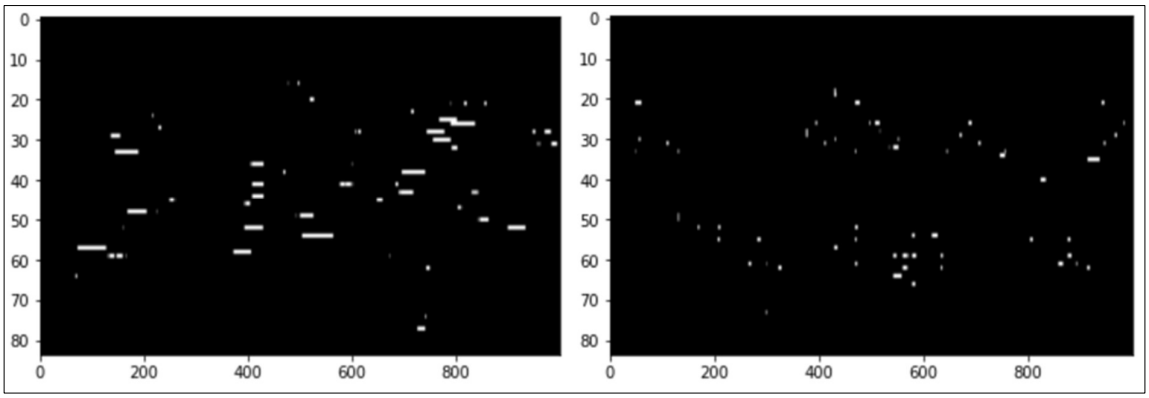

Figure 5-5 The nearest neighbor of Classical-1-GAN. Left: Classical-GAN-1; Right: a piece of music from the training set.

- An example of generated Romantic style music

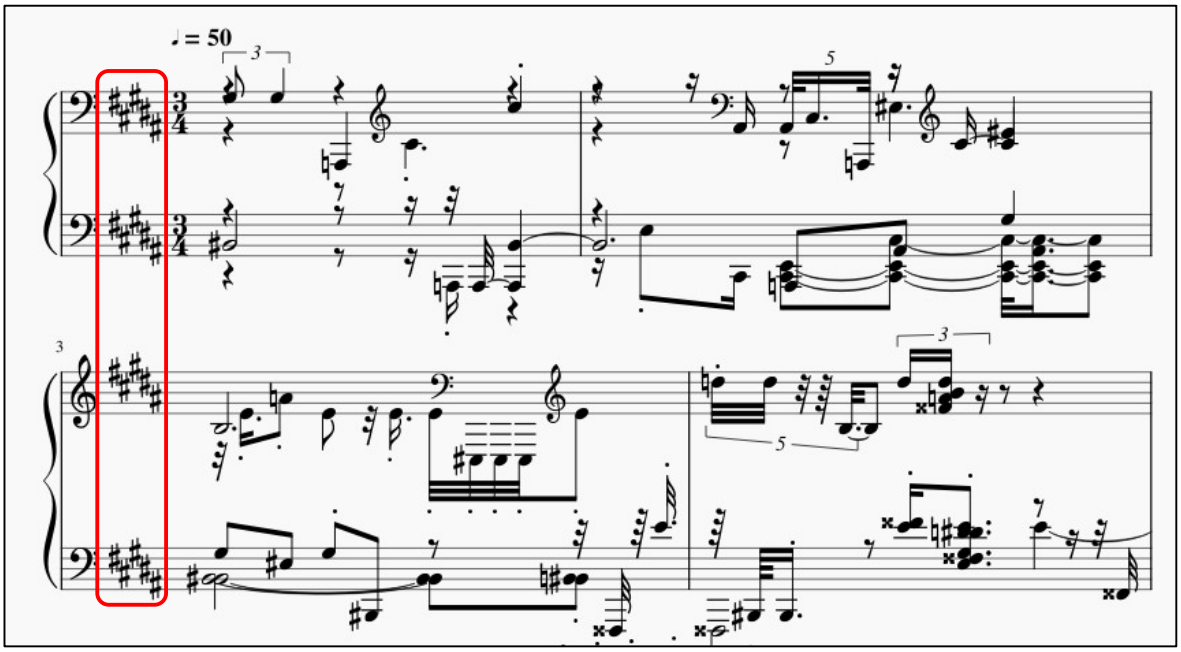

Figure 5-13 Romantic-1-GAN

One thing worth noting from the above music sheet is the keys boxed in red. As mentioned before in the Bi-LSTM section, it is common for romantic style music having multiple flat for sharp keys. Also, comparing Romantic-1-GAN and its nearest neighbor piece, we can observe that both pieces progress along the timesteps in rather stable pitches without much dramatically vertical pitch changes.

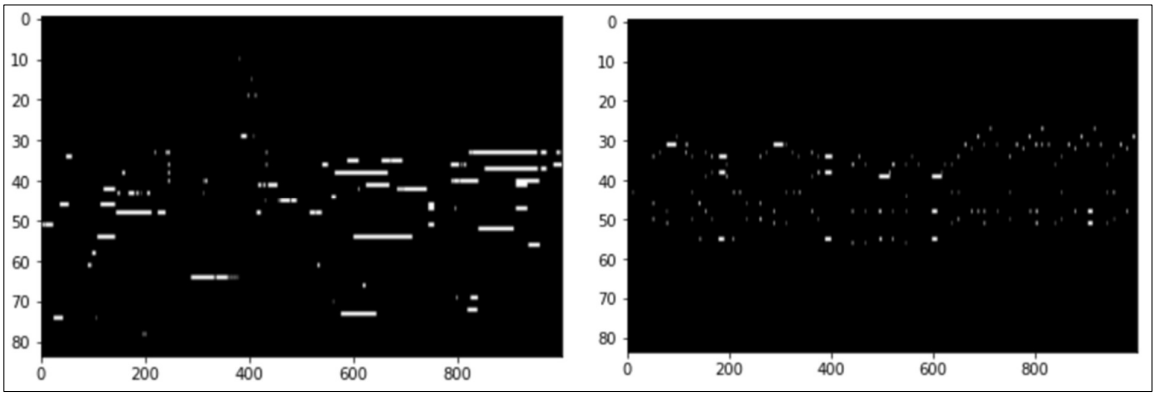

Figure 5-14 The nearest neighbor of Romantic-1-GAN. Left: Romantic-GAN-1; Right: a piece of music from the training set.

- An example of generated Modernist style music

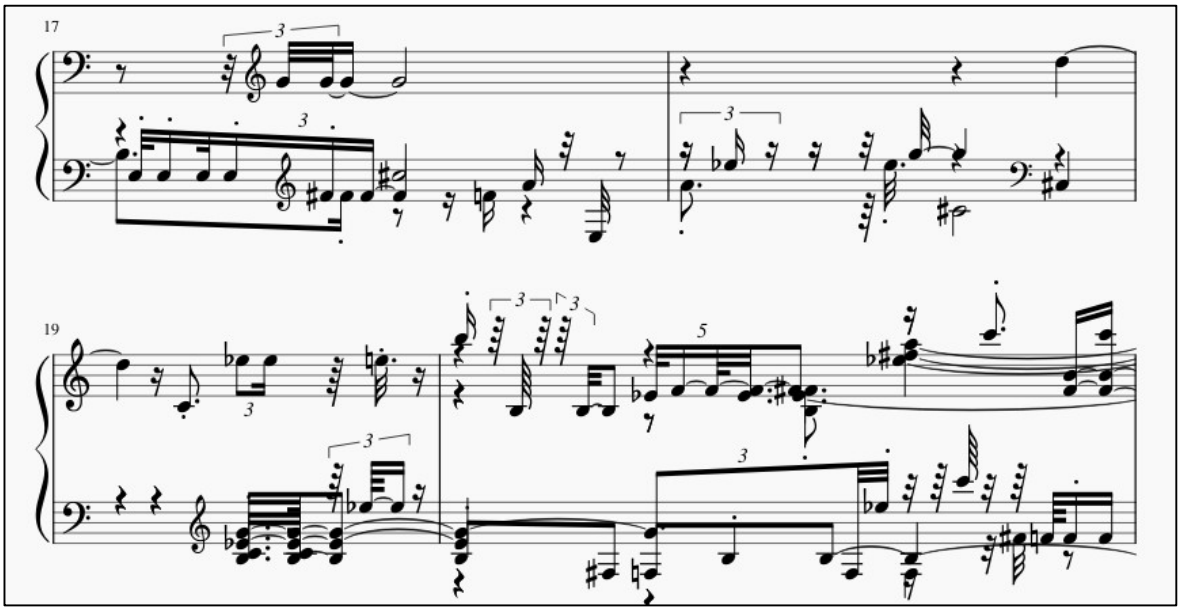

Figure 5-65 Modernist-1-GAN

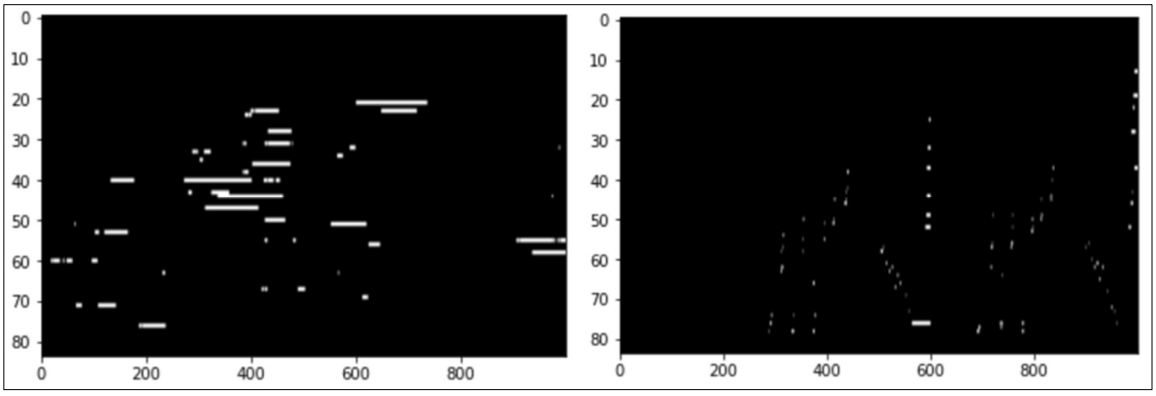

Figure 5-16 The nearest neighbor of Modernist-1-GAN. Left: Modernist-GAN-1; Right: a piece of music from the training set.

Most of the time, Modernist music refuses tonality. Its music does not anchor in any "key" and always tends to feel unsettled or "off." Observing the above music sheet and listening to its soundtrack, I think the same way. Unlike other musical styles like Baroque or Classical, Modernist music does not need to follow strict rules when composing. It is encouraged to defy "the convention" by pouring innovation into music. Thus, by this proposed GAN model's nature, we do not need to worry about the unsettling melody or rhythms of the generated music.

- Pitch histogram of four generated pieces

According to the below pitch histogram, all four pieces have multiple dominant keys, which is a good sign of note diversity. The GAN model does not generate specific notes but predicts notes proportionally with a few keys as major ones. It is a good sign as the GAN model makes it similar to human composing styles.

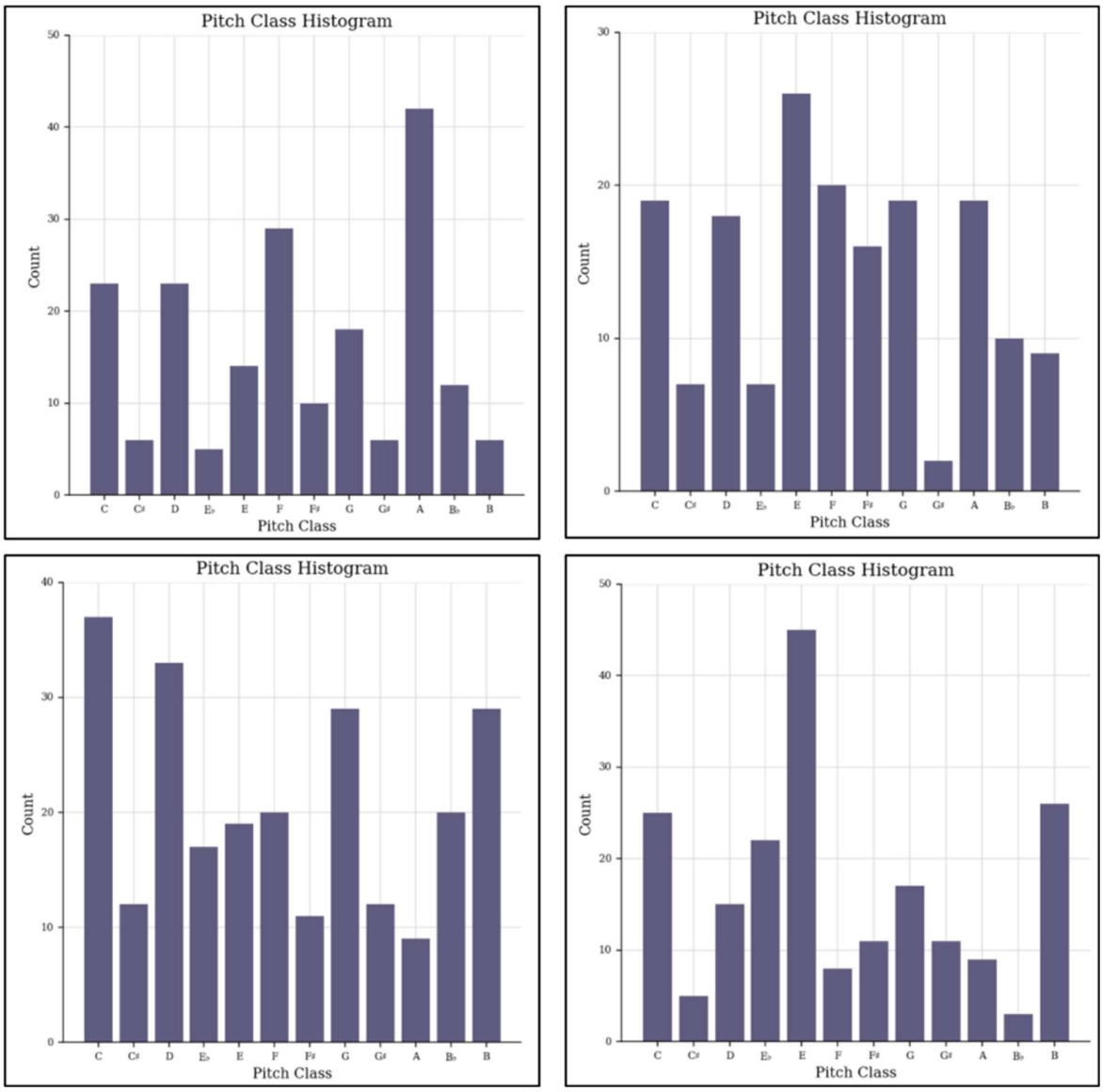

Figure 5-17 Four pitch histograms of the generated pieces from the GAN models. Clockwise from top-left: Baroque-1-GAN, Classical-1-GAN, Modernist-1-GAN, and Romantic-1-GAN.

5.2.3 Survey

The chart below is the survey of the GAN model. Though most respondents could clearly distinguish between 'Real' music or 'AI' music, there are still a portion (33.4% on average) of respondents could not identify whether their given pieces are composed by AI or human. This is calculated by averaging the summation of the percentage of the choices – 'Cannot tell' and the incorrect choices (either 'AI' or 'Human') according to each given piece.

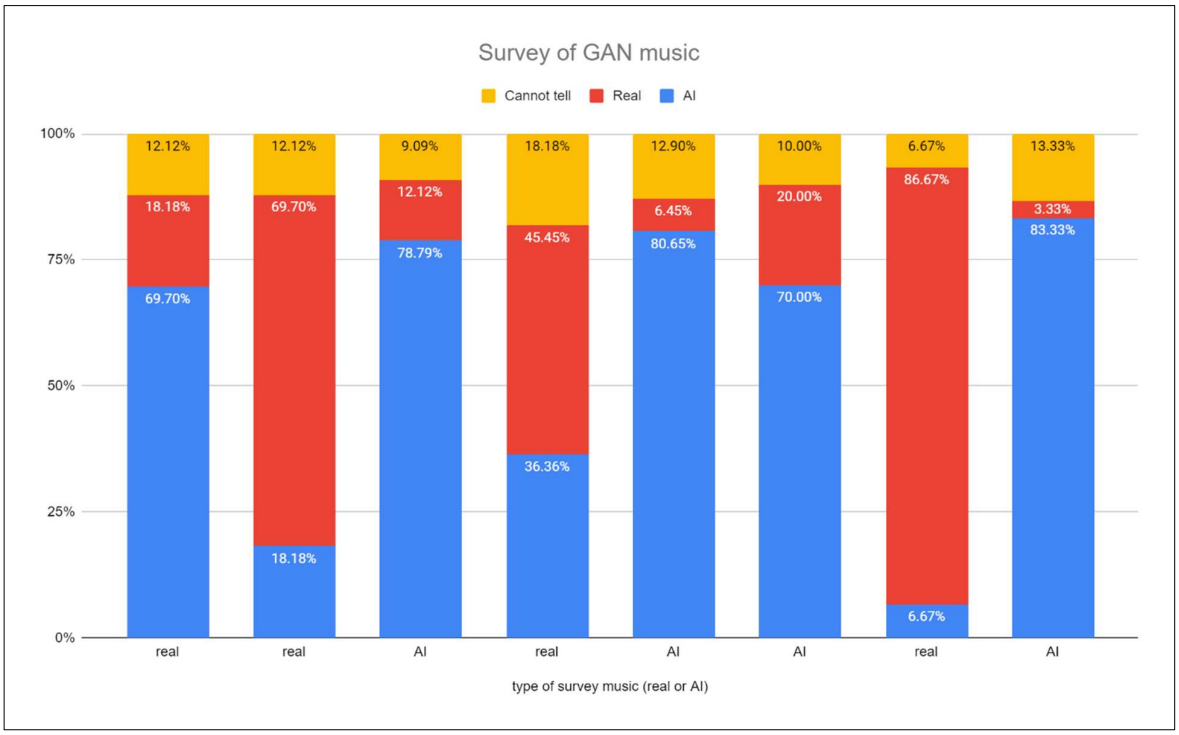

Figure 5-18 Survey of GAN music

6 Conclusion

6.1 Achievements

The generated results from the Bi-LSTM model demonstrate strong capabilities in producing classical music that most human could hardly tell whether they are composed by a human or an AI. The Bi-LSTM model proves to be able to generate music that makes sense in a sequential point of view.

The novel concept of treating sequence of notes and chords as 2D arrays proves to be a feasible design for the training dataset. This kind of input data shape allows the Generative Adversarial Network to generate music in the same way as generating images. By up scaling the dimension (1000 by 84) of the training set, it could also produce music with much longer timesteps than the previous studies (4 by 128).

Among the four Classical Music eras, I found the Bi-LSTM model suitable for generating Baroque, Classical, and some Romantic Music owing to the design of the Bi-LSTM model and the nature of these musical styles. These types of music require strict rules and are less complexed than the Modernist music. The design of LSTMs works just well for doing these tasks. On the other hand, the GAN model seems to be more creative and stochastic in generating music. I therefore consider it suitable for generating Modernist Music.

6.2 Future Improvement

To further improve the quality of the generated music, an augmentation and additional refinement of the training set is recommended. Though this project's training set includes hundreds of classical music pieces in midi format, most midi files are recorded live which means that some background noise might be as well recorded with the music during the original performance, and this could affect the precision of making the midi files. For instance, when visualizing some music sheets from the dataset using the 'MuseScore' software, it is obvious to observe excessive notes and chords comparing with the original music scores. It could be a potential cause of harming the later training process, however, further trials and examinations are needed to prove this hypothesis. Also, the data cleaning process is essential as I found many duplicate and invalid (empty) pieces of music from the dataset.

It is believed that the Bi-LSTM model outperforms the unidirectional LSTM model from the previous studies. However, a comparison of performance between the unidirectional LSTM model and the Bi-LSTM model should be made in the future to confirm this argument.

7 References

Boulanger-Lewandowski, N., Bengio, Y., & Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. arXiv preprint arXiv:1206.6392.

Briot, J. P., Hadjeres, G., & Pachet, F. D. (2017). Deep learning techniques for music generation--a survey. arXiv preprint arXiv:1709.01620.

Choi, K., Fazekas, G., & Sandler, M. (2016). Text-based LSTM networks for automatic music composition. arXiv preprint arXiv:1604.05358.

Dong, H. W., Hsiao, W. Y., Yang, L. C., & Yang, Y. H. (2017). MuseGAN: Demonstration of a convolutional GAN based model for generating multi-track piano-rolls. ISMIR Late Breaking/Demos.

Eck, D., & Schmidhuber, J. (2002). A first look at music composition using lstm recurrent neural networks. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, 103, 48.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

Hadjeres, G., Pachet, F., & Nielsen, F. (2017, July). Deepbach: a steerable model for bach chorales generation. In International Conference on Machine Learning (pp. 1362-1371). PMLR.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

Moorer, J. A. (1972). Music and computer composition. Communications of the ACM, 15 (2), 104–113.

Shiebler. (2016). Magenta/magenta. GitHub.

https://github.com/magenta/magenta/tree/master/magenta/reviews

8 Appendices

8.1 Monthly Logs

Oct 2020

- project plan completed.
- confirm the project scope and preliminary system design with my project supervisor.
- start collecting midi files for baroque and classical era and explore resources of free midi files.
- study and explore ways of preprocessing midi files using different python libraries (music21 and pypianoroll)
- study papers on generating music using LSTM.
- study basic LSTM and Bidirectional LSTM designs
- study GAN models on image processing for later transformation on music to 2D array.

Nov 2020

- cleaned training data. (remove duplicated music files from the dataset)
- Build the training dataset with a shape of (128*1000) for each piece of music for the GAN model training.
- setup the GAN model with convolutional layers and started training on small batches of Baroque music to test whether the model works as expected.
- Continue training with different depths of the GAN model to see their performance differences.

Dec 2020

- Cleaned training data. (remove duplicated music files from the dataset) (classical, romantic, and modernist eras)

- Build the training dataset with a shape of (128*1000) for each piece of music for the GAN model training (classical, era).
- Trained the GAN model with convolutional layers of Baroque music, and a small batch of Classical music.
- Set up the Bi-LSTM model and trained on Baroque music.
- Continue training with different depths and number of nodes for each layer of the GAN model to see their performance differences.

Jan 2021

- Continue training the Bi-LSTM model on Classical, Romantic, and Modernist music.
- Finalize the GAN model's input data shape (1000, 84) after many previous trials and retrain the Baroque music.
- Analyze the generated results from the Bi-LSTM model.
- Update the abstract of this project report.
- Finish the Interim report II.

Feb 2021

- Continue training the GAN model for all four eras.
- Designed and distributed surveys of the Bi-LSTM and the GAN model.

Mar 2021

- Survey collection and analysis.
- Trying to find passages of music from the training set that are most similar to the generated examples by the Nearest Neighbor Search.
- Revising the final report.
- Start making the project video.

8.2 Project Schedule