NNDL Project: Melody and chord generation through DCGANs

Riccardo De Vidi[†]

***Abstract*—Music generation refers to the creation of music using automated systems rather than manual composition. In recent years, this task has attracted increasing interest from researchers, with primary goals centered around melody and arrangement generation. In this context, MidiNet is a generative adversarial network (GAN) model capable of solving the melody generation task in the symbolic domain. The model can generate a melody bar taking into account a previous melody bar and a single chord (extracted from the underlying chord bar).**

Inspired by the MidiNet architecture, this project presents a generative model capable of solving both the melody and the chord generation task. The introduced model is designed to incorporate full knowledge of the previous melody bar and chord bar into the generation process. The model can also be used to address only the melody or the chord generation task. Moreover, by iteratively using the model, it is possible to generate an arbitrary number of bars, however, experimental results indicate that the model produces acceptable outputs in general up to three consecutive bars. To conclude, the quality of the model is assessed by using two metrics, the pitch count and the note count.

***Index Terms*—Unsupervised Learning, Neural Networks, Convolutional Neural Networks, Generative Adversarial Networks.**

I. Introduction

According to a recent review by Wang et al. [1], Generative Adversarial Networks (GANs), and in particular Deep Convolutional GANs (DCGANs), represent a state-of-the-art approach for image generation. Recently, research has begun to explore the application of GANs to sequential data generation as well. In the context of symbolic music generation, an important contribution is represented by MidiNet, an architecture capable of generating symbolic jazz melodies one bar at a time by leveraging side information and random Gaussian noise.

Beyond MidiNet, several other models have addressed symbolic melody generation, including JazzGAN [2], Yu et al. [3], and SSMGAN [4]. Similarly, MuseGAN [5] have focused on the generation of symbolic arrangements. However, none of these models is capable of separately generating symbolic melody and chord bars one at a time, while ensuring that each new bar is influenced by the previous melody and chord ones. Moreover, they do not support the independent generation of melody and chord bars, when required.

In this project, taking inspiration from the MidiNet architecture, a two-DCGAN-based model for symbolic melody and chord generation is developed. Splitting each music bar into a melody bar and a chord bar, the model can generate a new melody bar, a new chord bar, or both exploiting

- a uniform random vector and a previous melody or chord bar;
- a uniform random vector and both previous melody and chord bars.

Furthermore, the model can work iteratively, allowing flexible generation of both melody and chord bar sequences.

This report is structured as follows. In Section II the state of the art in symbolic music generation is described. The processing pipeline and the used dataset are presented in Sections III and IV, and the learning framework is presented in Section V. In Section VI the results obtained and in Section VII the conclusions are reported.

II. Related Work

MidiNet is a melody generation model that combines Convolutional Neural Networks (CNNs) with GANs. It generates monophonic melodies leveraging prior musical context, but its capabilities are limited to producing a single melody line without pauses or support for multiple instruments. Although MidiNet uses complete melody sequences for conditioning, it uses a simplified representation of chord progressions. Specifically, it is trained only on the 24 basic chord triads, and from each chord bar only a single chord is kept. The latter is then encoded as a 13-dimensional vector, before being used in the conditioning. These simplifications result in the loss of significant information, leading to chord embeddings that are not fully representative of the original bar.

JazzGAN is a melody generation model that integrates Recursive Neural Networks (RNNs) with GANs. It focuses on generating monophonic jazz melodies conditioned on chord progressions. While JazzGAN demonstrates promising results across various evaluation metrics, it does not address the problem of chord generation, limiting its scope.

MuseGAN, on the other hand, tackles the broader task of multi-track music generation with minimal data simplification. It simultaneously generates full arrangements across five instrument tracks: bass, drums, guitar, strings, and piano. This approach enables MuseGAN to produce complex and rich musical pieces. However, it lacks the ability to separately generate a melody line and a chord progression for the same instrument, and thus does not support separate melody and chord script generation.

This work focuses on parallel melody and chord generation using CNN DCGANs. The trained model is capable of solving all the tasks solved by MidiNet (excluding the

[†]Department of Information Engineering, University of Padova,
email: `riccardo.devidi.1@studenti.unipd.it`
student id: `2152869`

melody generation without conditioning) and more, such as chord generation conditioned on the previous melody tab. Moreover, the model, differently from MuseGAN, could be used to generate a melody line and a chord line for the same instrument.

III. Processing Pipeline

Starting from a multi-track MIDI file dataset, for each file a dual-track version is generated. In each dual-track file one track is a melody track, and contains all the melody lines played by all the instruments, and the other one is a chord track, and contains all the chords played by all the instruments.

To begin, the multi-track files are filtered out accordingly to their time signature and tempo changes: only those with a 4/4 time signature and without tempo changes for all the tracks are kept. This step is performed to reduce the complexity of the training data and the model.

Since in each file there are no tempo changes, the length of a quarter note is constant in it. Denoting by $t[BPM]$ a multi-track file tempo in beat per minute, the length of a quarter note is $l_q = \frac{60}{t}[s]$. The latter comes from the fact that a 4/4 time signature implies that in 60 seconds there must be t beats, and one beat corresponds to a quarter note. From the time signature, the length of a bar in the file is defined as four times the length of the quarter note, that is $l_b = 4l_q = \frac{240}{t}[s]$.

Continuing, as in MidiNet, each bar is sampled 16 times, hence the sampling frequency is $f_s = \frac{16}{l_b} = \frac{t}{15}[Hz]$. The sampling of a track is called piano roll. This choice offers a good trade-off between temporal resolution and representation size. Increasing the number of samples per bar would improve accuracy, but at the cost of increased computational requirements and longer model training times. Unlike in MidiNet, where the piano roll notes are modified to remove pauses, in this context the piano rolls are not edited. This decision is made to preserve the full musical content of the bars.

Each piano roll is retained for further analysis only if all of its notes fall within the C4-B5 pitch range, as in MidiNet. This constraint reduces the overall complexity of the data and focuses the analysis. In each piano roll, each bar is classified as a melody bar or as a chord bar by looking at the number of simultaneous notes that are played in it. If the number of simultaneously played notes is fewer than 3, the bar is classified as a melody bar, otherwise it is classified as a chord bar.

The dual-track version of a multi-track file is created concatenating the melody bars and the chord bars, creating two tracks. To conclude, the training dataset is created iterating over each dual-track representation and storing two melody bars and the two corresponding chord bars if each of them has at least 4 playing notes. The processing pipeline is graphically represented in Figure 1.

This pipeline erases the information about which track is playing each note. As a result, each track in the resulting dataset may be more complex than the original individual tracks. Additionally, any notes from melody or chord bars that occur simultaneously are removed. The final dataset can be seen as a hybrid representation, combining characteristics of both multi-track and single-track MIDI files.

IV. Signals and Features

A. Dataset

The Lakh MIDI dataset is a collection of 176,581 unique MIDI files, 45,129 of which have been matched and aligned to entries in the Million Song Dataset [6] [7] [8]. The Lakh dataset *Clean MIDI subset* is a subset composed of 17,259 MIDI files, where and each MIDI file stores data relative to a song, divided into tracks. Each track stores the data of an instrument.

The library `pretty_midi` [9] [10] is used to read and write MIDI files. It interprets the MIDI data as a list of instruments, where each instrument has its own sequence of events: time signatures, tempo changes, notes velocities, and so on. To begin, a MIDI file is loaded into a `PrettyMIDI` object through its path.

B. Time signatures and tempo changes

Given a `PrettyMIDI` object, its instruments time signatures are stored in the attribute `PrettyMIDI.time_signature_changes`, and its instruments tempo changes are obtained with the function `PrettyMIDI.get_tempo_changes()`. The former stores a list of `TimeSignature` objects, where each object stores the numerator and the denominator of the time signature and the time where the time signature is located; the latter returns arrays of tempo changes and their times. The track time signature is 4/4 if the length of `PrettyMIDI.time_signature_changes` is 1 (there is one time signature change along the track), if the numerator and the denominator attributes of the `TimeSignature` object are equal to 4 and if the time of the time signature event is 0.0 (the change occurs at the beginning of the track). The track has no tempo changes if `PrettyMIDI.get_tempo_changes()` returned arrays have length 1 (there is one tempo change along the track) and if the tempo change event time is 0.0 (also in this case, the change occurs at the beginning of the track).

C. Instrument piano roll

Each instrument in a `PrettyMIDI` object is managed with the object `Instrument`. It is important to note that, in general, each instrument piano roll could have a different length. In order to obtain a melody and a chord piano rolls for each file, each instrument piano roll is uniformed in length, tacking as reference length the one of the longest piano roll in the file. The longest file piano roll length, in seconds, is obtained with the function `PrettyMIDI.get_end_time()`. Given an instrument, its piano roll is extracted with the function `Instrument.get_piano_roll()`, which transforms the sequence of events of an instrument into a two-dimensional matrix where the i-th column stores the velocities of the notes of the instrument at the i-th sampling instant. The returned array has 128 rows (each row corresponds to a

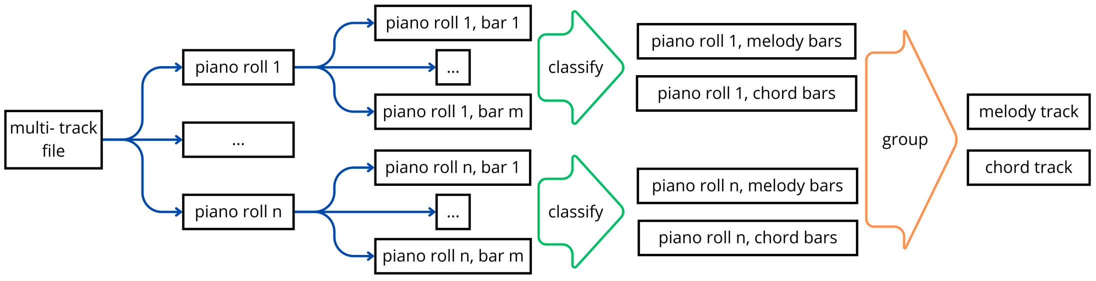

Fig. 1: Processing pipeline

MIDI note) and a variable number of columns. Having that the velocities of the notes are not relevant in this setting, the piano roll matrix is transformed to a boolean matrix with the same shape as before and with `True` entries only where the velocities were greater than zero (playing notes). Each instrument piano roll, if necessary, is zero padded at the end, so all instruments piano rolls have the same shapes. A piano roll is further considered if its notes are in the range B4-C5. The considered piano rolls are stored in a compact representation with only 24 rows (from row 60 to row 79, which correspond to the interval of interest).

D. *Data augmentation*

To conclude, the dataset is augmented by considering a copy of the dataset with all the chord bars empty and a copy of the dataset with all the melody bars empty.

V. Learning Framework

The next melody and chord tabs are generated using two DCGANs that share the same structure.

Each GAN generator takes as input a 100-dimensional uniform random vector and the previous melody and chord bars in compact representation, and outputs a two-dimensional float matrix with 128 rows and 16 columns where each cell stores the probability of playing a note at a specific sampling instant. Each GAN discriminator takes as input the two-dimensional matrix corresponding to the generated bar or the next real bar and the two-dimensional reduced representation matrix of the previous bar, and returns a value that represents the probability of the generated bar or of the next real bar to be real.

The goal of each generator is to generate as many realistic bars as possible, while the goal of each discriminator is to distinguish between the generated bars and the next bars, taking into account the previous bars.

Denoting with G_m the melody GAN generator, D_m the melody GAN discriminator, z_m the melody noise vector, c_m the previous melody bar and c_c the previous chord bar, $G_m(z_m|c_m, c_c)$ the generated melody bar, the melody GAN is learned by optimizing

$$\begin{aligned} min_{G_m} max_{D_m} \ & E_{X_m \sim p_m(X_m)}[log(D_m(X_m|c_m))] + \\ & + E_{z \sim p_z(z)}[log(1 - D_m(G_m(z_m|c_m, c_c))]. \end{aligned} \quad (1)$$

Similarly, denoting with G_c the chord GAN generator, D_c the chord GAN discriminator, z_c the chord noise vector, $G_c(z_c|c_c, c_m)$ the generated chord bar, the chord GAN is learned by optimizing

$$\begin{aligned} min_{G_c} max_{D_c} \ & E_{X_c \sim p_c(X_c)}[log(D_c(X_c|c_c))] + \\ & + E_{z \sim p_z(z)}[log(1 - D_c(G_c(z_c|c_c, c_m))]. \end{aligned} \quad (2)$$

In equations (1) and (2) $X_m \sim p_m(X_m)$ denotes the operation of sampling real melodies from the dataset, and $X_c \sim p_c(X_c)$ denotes the operation of sampling real chords.

The learning process is driven by the discriminator, whose output is in the interval $(0,\ 1)$. The discriminator is trained to output 1 for real tabs and 0 for generated tabs. This is achieved by optimizing a Binary Cross-Entropy (BCE) loss function. On the other hand, the generator is trained to fool the discriminator by generating bars that the discriminator classifies as real. In other words, the generator aims to maximize the discriminator output for generated bars, making it approach 1, hence minimizing the BCE loss from the generator perspective. The model structure is reported in Figure 2.

Notice that all the generator inputs are not needed to solve all the tasks listed in Section I. This problem is simply solved by providing a zero input in place of the input component that is not needed to solve a task (i.e. in melody generation without the previous chord bar conditioning the conditioning is replaced by a zero two-dimensional matrix with 24 rows and 16 columns).

A. *GAN Generator*

Each GAN generator is made up of a conditioning network, an innovator network, and a generative network. The conditioning network processes the side information that will be incorporated in the generative process to take into account during the next bar generation the previous bars. The innovator network processes a 100-dimensional normal random vector with zero mean and unit variance that will introduce randomness in the next bar generation. The generative network generates the next bar by combining and processing all the data generated by the other two networks, hence combining the information of the previous bar with the randomness given by the random vector.

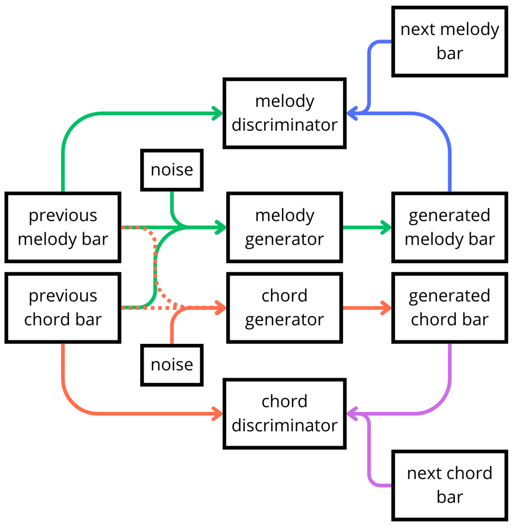

Fig. 2: Model structure

Each conditioning network takes as input the concatenation of the previous melody and chord bars. In the melody generation GAN the chord bar is concatenated to the melody bar, in the chord generation GAN the melody bar is concatenated to the chord one. Using the reduced representations, the input of these networks has dimension $2 \times 24 \times 16$. The networks are composed of a convolutional layer with 256 filters of size $(24,\ 2)$ and stride 2 followed by two fully connected layers of size 2048, 1024 and 1024, respectively. Each layer is followed by a batch normalization layer and a leaky ReLU activation function.

Each innovation network takes as input the 100-dimensional random vector and is made up of two fully connected layers. The first layer has 2048 neurons and the second one 1024. Also in this case, each layer is followed by a batch normalization layer and a leaky ReLU activation function.

Finally, each generative network takes as input the concatenation of the results of the conditioning and the innovation network. In detail, the outputs of the other two networks are reshaped from $1 \times 1024 \times 1$ tensors to $512 \times 1 \times 2$ tensors, and are then concatenated along the channel dimension. Each network is composed of four transposed convolutional layers: the first one has 512 filters with shape $(1,\ 2)$ and stride 2, the second one has 256 filters with shape $(1,\ 2)$ and stride 2, the third one has 128 filters with shape $(1,\ 2)$ and stride 2, and the last one has 1 filter with shape $(128,\ 1)$ and stride 1. The first three layers are followed by a batch normalization layer and a leaky ReLU activation function, and the last one is followed by a batch normalization layer and a sigmoid activation function. The latter limits the output components in the interval $(0,\ 1)$, and the output components can be interpreted as the probability that a note will be played.

B. GAN Discriminator

The GANs discriminators are made up of two convolutional networks and a fully connected network. The convolutional networks process the new and the previous melody/chord bars, and the fully connected network combines this information. Each convolutional network is composed of two convolutional layers: the first with kernel size $(24,\ 2)$ or $(128,\ 2)$ depending on whether the processed bar is the previous or the new one with 256 filters and stride 2, the second one with 512 filters of size $(1,\ 4)$. The fully connected network takes as input the flattened and concatenated outputs of the conditioning network, and it has 3072 neurons. In the end, a sigmoid activation function is applied to the output.

VI. Results

The training is performed on Google Colab, using the T4 GPU environment. In this environment it is possible to use 12GB or RAM, 15GB of GPU RAM and 100GB of disk space. Also the testing is performed on the same environment. On the other hand, the dataset processing is performed on a Google Colab CPU environment. It is observed that all operations require much less resources than those provided by the environments.

A. Training

Both DCGANs are trained with the same learning rates: 1×10^{-4} for the generator and 1×10^{-4} also for the discriminator. The loss curves of the melody generation DCGAN are reported in Figure 3, and the ones of the chord generation DCGAN are reported in Figure 4. Both DCGANs generator go through an initial learning phase where they are overwhelmed by the discriminators, the generator loss is large and the discriminator loss is close to zero; after that, both models generators largely improve, and the losses stabilize.

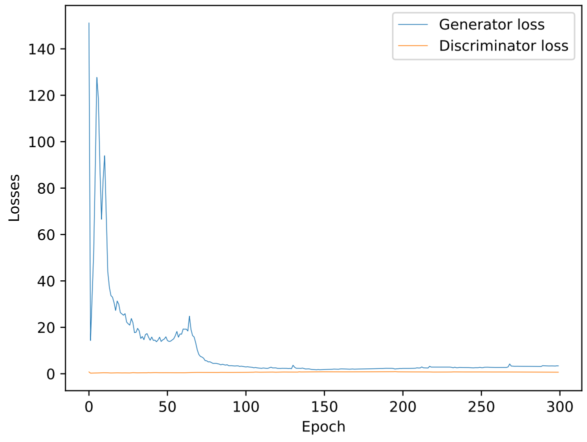

Fig. 3: Melody DCGAN training curves

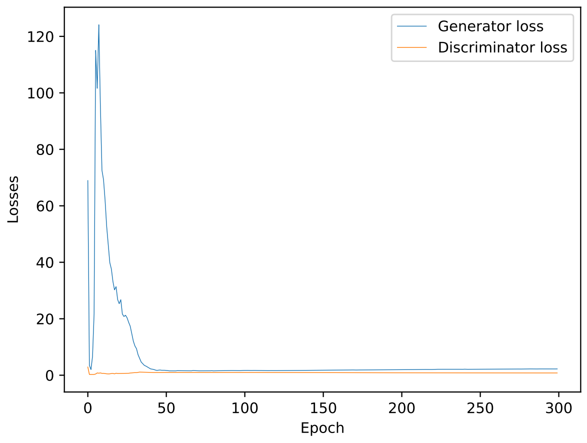

Fig. 4: Chord DCGAN training curves

Some model output through training are reported in Figures 5, 6, 7. Figure 5 shows how, afer 100 epochs, the melody DCGAN generator is still unable to generate realistic data (since the generated samples are simple and similar), even if its loss is much smaller with respect to the first epochs. From the same figure it is observed that the chord DCGAN, instead, seems capable of generating realistic bars after 100 epochs. Looking at the training curves, this fact is justified by observing that the chord DCGAN generator seems to be performing well even from the 50-th epoch, when its losses approach a small value that is kept up to the end of the training. On the other hand, Figure 6 shows that after 200 epochs the melody DCGAN is capable of generating realistic data, and the chord DCGAN is still producing good results. In conclusion, Figure 7 shows how both DCGANs seem to produce acceptable data after 300 epochs.

Fig. 5: 16 generated melody and chord bars after 100 epochs

Fig. 6: 16 generated melody and chord bars after 200 epochs

Fig. 7: 16 generated melody and chord bars after 300 epochs

B. *Testing*

The trained models are used in an iterative way to produce melody and chord tabs of 8 bars, where the first melody and chord bar are given and condition the generation. It is observed that the generation process is relatively fast and that the generation of a bar usually takes much less than 5 seconds. Example results are reported in Figure 8.

It is observed that the generated bars tend to be more realistic in their first half, and their second half usually sounds more random and chaotic. This suggests that the model has not fully captured the distribution of the training data, which is much more complex than that of MidiNet due to the revised conditioning mechanisms. This problem could be solved by increasing the complexity of the generators and the discriminators and performing a longer training, and additionally by increasing the tabs involved in the conditioning mechanism (but this would lead to even more complex models).

The quality of the generations is objectively assessed by using two metrics: the pitch count and the note count [11]. The former is defined as the number of different pitches played in a bar, and is a pitch-based metric; the latter is defined as the number of notes played in a bar, and is a rhythm-related metric. Even if these metrics are easy to evaluate, they are capable of detecting if the generated samples are realistic. Figures 9, 10 show that the pitch count and the note count for the real melody bars and the generated ones are at least comparable, suggesting that the melody DCGAN is really generating good samples. The same cannot be said for the chord DCGAN: Figures 11, 12 show that the metrics for the generated bars are completely different from the ones evaluated on the real bars.

VII. Concluding Remarks

In this project a melody and chord generation model has been developed. The model shows results that seem acceptable, but the pitch count and the note count metrics show that the generated chord bars have features that are not comparable with the ones of the training data.

The presented work could be extended by implementing melody and chord bar generation without conditioning, extending the acceptable note range over the B4-C5 one, by using more complex metrics to evaluate the model results, and by using metrics to assess the learning phase of the model.

In doing this project I have learned to manage midi data, the models that are typically used to solve melody generation tasks, the metrics that are used to evaluate such models. Moreover, I experienced the difficulty of training a conditional

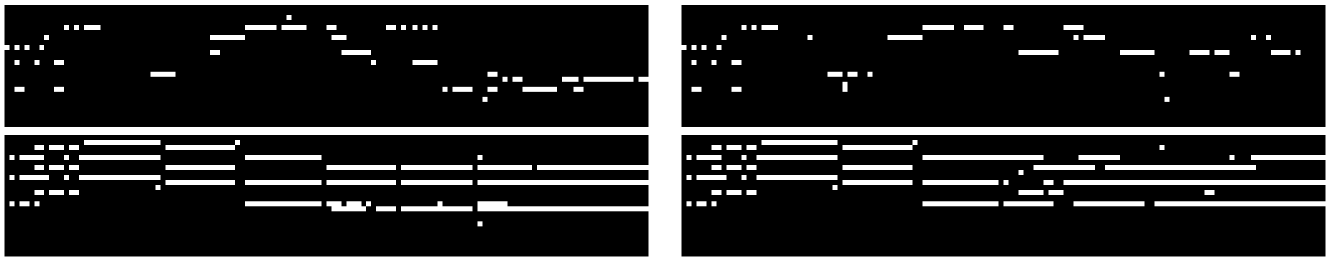

Fig. 8: Two examples of 8-bars tab generation (notes in the range C4-B5). In the first line the melody tabs are reported, in the second line the chord tabs are reported

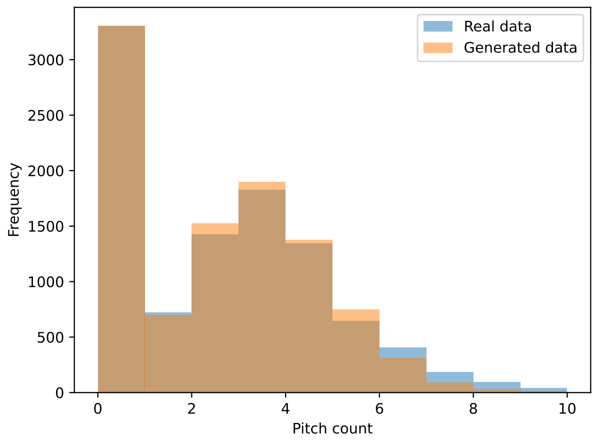

Fig. 9: Melody pitch count

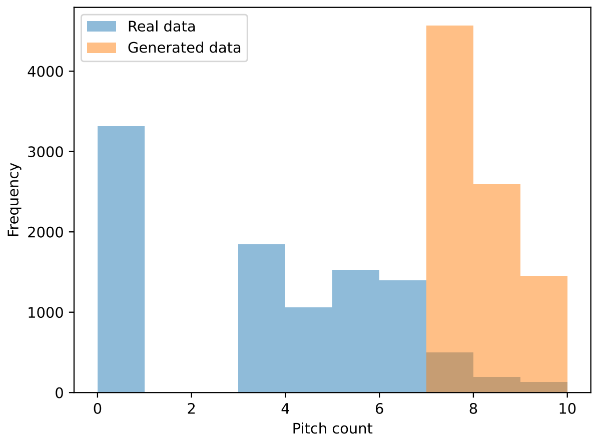

Fig. 11: Chord pitch count

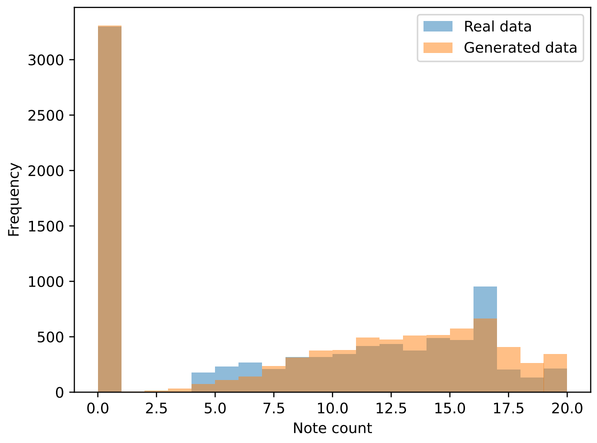

Fig. 10: Melody note count

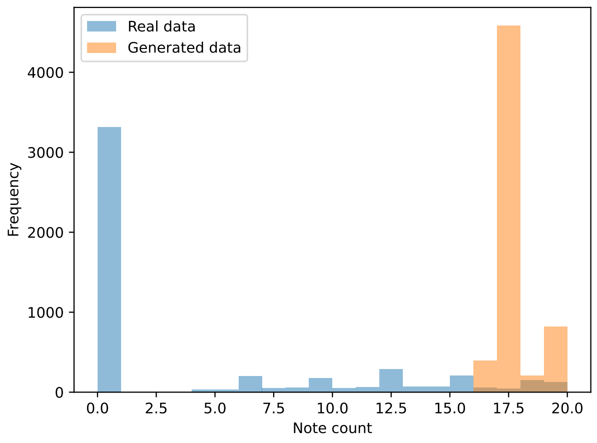

Fig. 12: Chord note count

DCGAN model, of choosing how to encode a conditioning side information and of exploring the literature.

References

[1] L. Wang, Z. Zhao, H. Liu, J. Pang, Y. Qin, and Q. Wu, "A review of intelligent music generation systems," *Neural Computing and Applications*, vol. 36, no. 12, pp. 6381–6401, 2024.

[2] N. Trieu and R. Keller, "Jazzgan: Improvising with generative adversarial networks," 08 2018.

[3] Y. Yu, A. Srivastava, and S. Canales, "Conditional lstm-gan for melody generation from lyrics," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 17, no. 1, pp. 1–20, 2021.

[4] H. Jhamtani and T. Berg-Kirkpatrick, "Modeling self-repetition in music generation using generative adversarial networks," in *Machine learning for music discovery workshop, ICML*, 2019.

[5] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan: Symbolic-domain music generation and accompaniment with multi-track sequential generative adversarial networks," *arXiv preprint arXiv:1709.06298*, 2017.

[6] C. Raffel, *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD thesis, Columbia University, 2016.

[7] C. Raffel, "The lakh midi dataset v0.1." https://colinraffel.com/projects/lmd/, 2016. Accessed: 2025-07-19.

[8] B. McFee, T. Bertin-Mahieux, D. P. Ellis, and G. R. Lanckriet, "The million song dataset challenge," in *Proceedings of the 21st International Conference on World Wide Web*, pp. 909–916, 2012.

[9] C. Raffel, "pretty˙midi." https://github.com/craffel/pretty-midi, 2014. Accessed: 2025-07-19.

[10] C. Raffel and D. P. Ellis, "Intuitive analysis, creation and manipulation of midi data with pretty_midi," in *15th international society for music information retrieval conference late breaking and demo papers*, pp. 84–93, 2014.

[11] L.-C. Yang and A. Lerch, "On the evaluation of generative models in music," *Neural Computing and Applications*, vol. 32, no. 9, pp. 4773–4784, 2020.