Symbolic-domain virtuosic piano melody generation using a MIDINET inspired architecture

Gabriele Boscarini, Ali Esmaeili nasab lahijan

***Abstract*—In this work, we built a generative artificial network (GAN) architecture for symbolic music generation, taking inspiration from the MidiNet model. Our model is designed to generate a sequence of consecutive bars representing MIDI melodies, exploiting the information given by a bar, to generate the next one. We used a dataset of virtuosic piano pieces, that has been pre-processed to extract melodies with a complex pattern. We adapted the original MidiNet model to handle this kind of melody, adding also additional strategies and data augmentation to prevent mode collapsing and stabilize the training of the GAN. We also experimented with the Wasserstein GAN (WGAN) to improve training stability and enhance the quality of generated sequences. Finally, we made a comparison between melodies generated by the standard GAN and the WGAN, observing that the latter obtain the best result.**

***Index Terms*—Generative Adversarial Network, MidiNet, symbolic music generation, data augmentation, Wasserstein Loss.**

I. Introduction

Music generation using neural networks is an expanding field within deep learning, where symbolic representations of music, such as notes, tempo, and instruments, are commonly stored in MIDI files. This project focuses on generating new music based on virtuosic piano compositions by training a neural network to learn the intricate patterns and styles of these pieces. With the ability to automatically create complex musical content, this approach highlights the growing potential of AI in creative domains. We utilize a GAN architecture [1], inspired by MidiNet [2] which is a successfull attempt to generate consecutive bars of MIDI melodies, exploiting the information given by a bar to condition the generation of the next bar. Our work aim to adapt this model to handle the generation of virtuosic piano bars. Specifically, our model can handle piano roll bars that contains melodies composed by fast notes, up to sixty-fourth notes. We use piano rolls derived from the MAESTRO dataset, which contains complete musical pieces, including both melodies and chords. In our case, we focused on generating symbolic music representing only melodies. During preprocessing, we aimed to isolate the melodic content from the full piano roll. However, the separation process was not perfectly precise, resulting in bars that sometimes contain chords along with melodies. Training GANs presents several challenges, particularly when it comes to conditioning the generation process. We faced issues with ensuring the model accurately generated music conditioned on previous bars while maintaining consistency across generated sequences. To address this, we experimented extensively with various strategies to stabilize the training and improve the quality of the outputs. These strategies range from model regularization methods, such as feature matching [3]and minibatch discrimination [4], to various data augmentation techniques, including time shifting and octave shifting of the bars. We obtained the best result by switching from the standard GAN architecture to the Wasserstein GAN (WGAN) [5], which offered improved stability and quality in generated sequences. This shift helped mitigate issues like mode collapse and produced more coherent musical structures, leading to a clear enhancement over the original GAN model.

II. Related Work

”MidiNet: A Convolutional Generative Adversarial Network for Symbolic-Domain Music Generation” [2] introduces a CNN-based approach for generating melodies in the symbolic domain, departing from the typical use of recurrent networks. The model generates melodies bar by bar, using a GAN framework with a conditional mechanism to incorporate chord sequences or prior melodies. In user studies, MidiNet’s melodies were found to be more interesting than those generated by Google’s MelodyRNN, while maintaining a realistic and pleasant sound. This work highlights the potential of CNNs in symbolic music generation, particularly for single-track melody creation.

”MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment” [6] introduces three GAN-based models for generating multi-track music, trained on a large rock music dataset. The models can produce coherent four-bar sequences and enable human-AI collaboration by generating accompanying tracks. While the work is notable for multi-track generation, it is limited to short sequences and specific genres. Our project extends this by focusing on longer, more complex piano compositions, advancing symbolic music generation in the realm of virtuosic performance.

The paper on Variational Autoencoders (VAE) [7] tackles the challenge of applying VAEs to sequential data, particularly when dealing with long-term structures. By introducing a hierarchical decoder, the model generates subsequences from latent embeddings, addressing the ”posterior collapse” issue that hinders recurrent VAEs. In music generation, this method shows significant improvement in sampling, interpolation, and reconstruction compared to standard flat VAE models. Unlike our approach, which aligns more closely with CNN-based architectures like MidiNet for melody generation, this work

emphasizes latent space efficiency and long-term sequence modeling. While the hierarchical VAE excels in structural

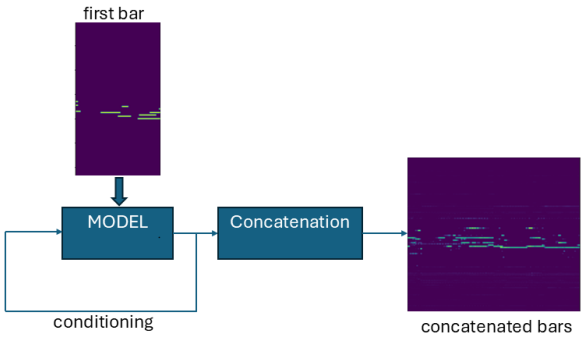


Fig. 1: music generation workflow

coherence, our work focuses on learning stylistic patterns from complex piano compositions using a GAN framework.

III. Processing Pipeline

A. Sequential melody generation workflow

The core of our music generation workflow revolves around sequential bar generation, where each bar influences the creation of the next. The process is shown in Fig. 1 and works as follows: A first bar is sampled from the dataset and it is feed to the model as conditioning input. The output of the model is used as conditioning input for generating the next bar. This process of conditioning on prior output allows the model to maintain long-term structure in the music. As the process continues, each newly generated bar is concatenated with the previous ones, creating a sequence of consecutive bars. This sequential concatenation mimics the flow of real music, where one musical phrase naturally leads into another, building up an entire piece.

B. The generative model

Our generative architecture is based on a Generative Adversarial Network (GAN), composed of three key components: the generator, the conditioner, and the discriminator, each of which leverages convolutional neural networks (CNNs) to handle the symbolic music generation task.

1) Generator: The generator G is responsible for producing new bars of music. It starts by taking a latent vector $z \in \mathbb{R}^l$ with l = 100 as input, which is passed through a series of fully connected and convolutional layers. These layers progressively transform the latent space into a piano roll matrix that represents a bar of music. The generator's architecture uses transposed convolution layers (also known as deconvolution) to upscale the latent space into the final bar-shaped output.

2) Conditioner: The conditioner in our GAN architecture is a CNN that processes the conditioning bar, which represents the previously generated bar of music. The goal of the conditioner is to downsample the conditioning bar at each layer, ensuring that the output dimensions of the conditioner layers match those of the layers of the generator. At every stage, the output of each conditioner layer is carefully matched with the corresponding layer in the generator. This allows for concatenation between the conditioner output and the generator output at each level. The concatenated feature maps (from the conditioner and the generator) are then used as inputs to the deconvolution layers of the generator, ensuring that the temporal and harmonic coherence of the generated music is preserved.

3) Discriminator: The discriminator is tasked with distinguishing between real and generated bars. It also consists of a series of convolutional layers, which analyze the input piano roll and predict whether it is real or generated. The convolutional structure of the discriminator enables it to capture local and global patterns in the piano roll, helping to evaluate the authenticity of the generated bars. This feedback is used to iteratively improve the generator.

C. Training strategy

To address the training of the GAN we followed the typical GANs training logic [4], where the objective is to solve

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$$

where $X \sim p_{\text{data}}(X)$ denotes the operation of sampling from real data, and $z \sim p_z(z)$ the sampling from a random distribution. To stabilize this adversarial process, and to avoid mode collapsing [8] we used the following strategies. First, the discriminator is trained twice with respect to the generator. This prevent the discriminator from being to powerful and overpower the generator. Then we used the one side label smoothing [9]. We inserted also a minibatch discrimination layer at the end of the discriminator, with 64 learnable features. Regarding the loss used to train both generator and discriminator, we used binary cross entropy loss with Logit. Following the work of Midinet, we used the feature matching technique. The idea of feature matching is to add additional L2 regularizers to the generator loss, such that the distributions of real and generated data are enforced to be close. The resulting loss of the generator is:

$$\mathcal{L}_G = -\frac{1}{N}\sum_{i=1}^{N}\log(D(G(z_i))) + \lambda_1\|\mathbb{E}[X] - \mathbb{E}[G(z)]\|_2^2 + \lambda_2\|\mathbb{E}[f(X)] - \mathbb{E}[f(G(z))]\|_2^2$$

where f denotes the first convolution layer of the discriminator, and λ_1, λ_2 are parameters to be set empirically.
The combination of these techniques helped to stabilize the training and improved the overall quality of the generated sample, but they weren't enough to solve the problem of mode collapsing. Indeed, with this configuration, the generated bar resulted to be too much similar to the previous bar, causing a repetitive melody arrangement and a progressive degradation of the quality of the output, bar after bar. To address this issue, we tried to reduce the complexity of the conditioner network, first by decreasing the number of parameters, then adding a dropout layer to it. As the problem persisted, we decided to apply data augmentation to the conditioning bar, at each step of the training. By diversifying the conditioning data, the model was encouraged to learn more general patterns, improving its ability to generate coherent and varied musical outputs without overfitting. We used two strategies for data augmentation:

1) Time change: Changes the duration of an arbitrary note from the melody. It can increase or decrease note time, clipping it to a suitable time division.

2) Octave change: Translate all melody up or down (chosen randomly) one octave

D. Wasserstein GAN (WGAN)

The last improvement that we made has been switching from the standard GAN architecture to the Wesserstein GAN (WGAN). Unlike conventional GANs, which minimize the Jensen-Shannon divergence between real and generated data distributions, WGAN leverages the Wasserstein (Earth Mover's) distance, a metric that more effectively captures distributional differences. By using this distance, WGAN provides a smoother gradient signal for the generator, even when the real and generated data distributions have limited overlap. To enforce the 1-Lipschitz continuity required by the Wasserstein distance, we used the Gradient Penalty method, that consist in adding a term to the loss. This term penalizes the norm of the gradient of the discriminator output with respect to its input, specifically for points interpolated between real and generated samples. The Loss is defined as follows:

$$L = \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] + \lambda_{GP}\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(|\nabla_{\hat{x}}D(\hat{x})|_2 - 1)^2]$$

where D(x) represents the output of the discriminator for input x, $\mathbb{P}_g$ and $\mathbb{P}_r$ are the generated and real data distributions, respectively, and λ_{GP} is a regularization coefficient controlling the strength of the penalty. Additionally, $\mathbb{P}_{\hat{x}}$ denotes the distribution of samples interpolated between real and generated data points, with $\hat{x} = \epsilon x + (1-\epsilon)\tilde{x}$, where x is a real sample, $\tilde{x}$ is a generated sample, and ϵ is a random scalar between 0 and 1.

The term $\left(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1\right)^2$ penalizes the discriminator gradient norm when it deviates from 1 for the interpolated samples, encouraging the discriminator to satisfy the 1-Lipschitz constraint.

IV. Dataset and pre-processing

A. Dataset

For our project, we required a high-quality MIDI dataset, and we selected the MAESTRO-v3.0.0 dataset [10]. The dataset consists of approximately 200 hours of paired audio and MIDI recordings from ten years of the International Piano-e-Competition. The MIDI data includes detailed information such as key velocities and pedal positions, with precise alignment to the corresponding audio files (within 3 ms). Each musical piece is annotated with composer, title, and year, and the dataset is organized into train, validation, and test sets, ensuring that the same composition, even if performed by multiple contestants, does not appear across subsets.

To develop a model capable of generating both melody and chords, it was necessary to separate the melody from the accompanying chords in the MIDI data. For this task, we utilized an existing codebase available on GitHub[1]. However, the separation process was not perfectly precise, resulting in bars that sometimes contain chords along with melodies.

B. Representation

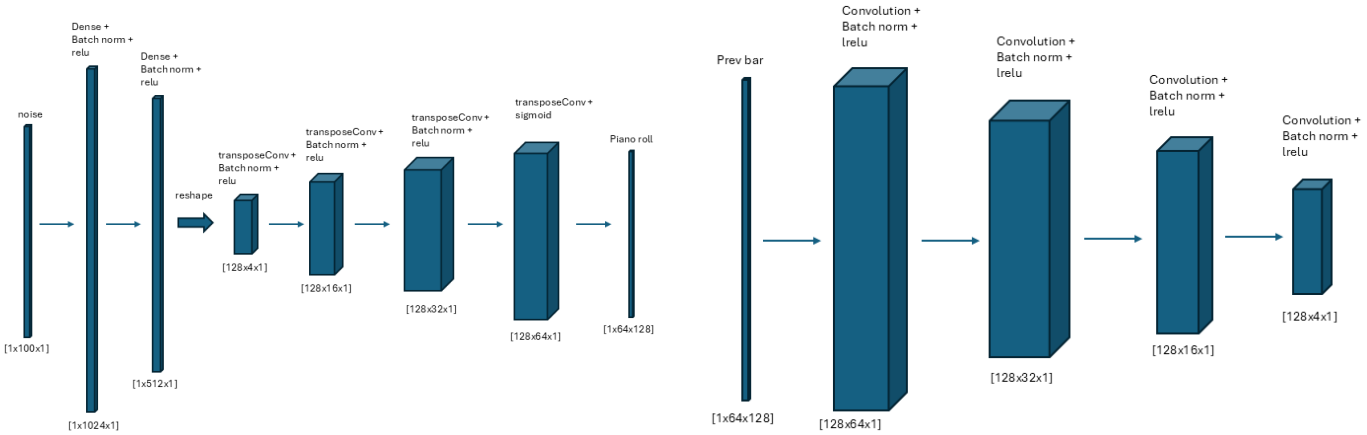
In our model, we used a symbolic representation of music by dividing each MIDI file into bars. We considered all 128 MIDI notes, from C0 to G10, to construct a tensor X of shape w-by-h, where $h = 128$ represents the number of MIDI notes and $w = 64$ represents the time steps per bar. The matrix generated from the note events in each bar is binarized, ensuring at most one active note per time step. However, one limitation of this approach is that it does not distinguish between a long sustained note and multiple consecutive short notes with the same pitch, which may result in a loss of certain musical subtleties.

To create the desired matrix representation, we utilized the PrettyMIDI [11] library to convert the MIDI files into piano rolls. A piano roll is a matrix where the vertical axis corresponds to the MIDI note numbers, and the horizontal axis represents time. Each element in the matrix indicates whether a particular note is active at a specific time step. This format enables us to effectively process the MIDI data for our neural network model.

C. Filtering

- **Fast Note Filtering:** For melodies, we fixed the smallest note unit to be the sixty-fourth note, meaning that each bar of music is divided into 64 equal time steps, corresponding to the duration of sixteenth notes.

[1] https://github.com/Rainbow-Dreamer/musicpy/wiki/the-algorithm-to-split-the-main-melody-and-chords-from-a-piece-of-music



(a) generator network

(b) conditioner network

Fig. 2: Generator and Conditioner

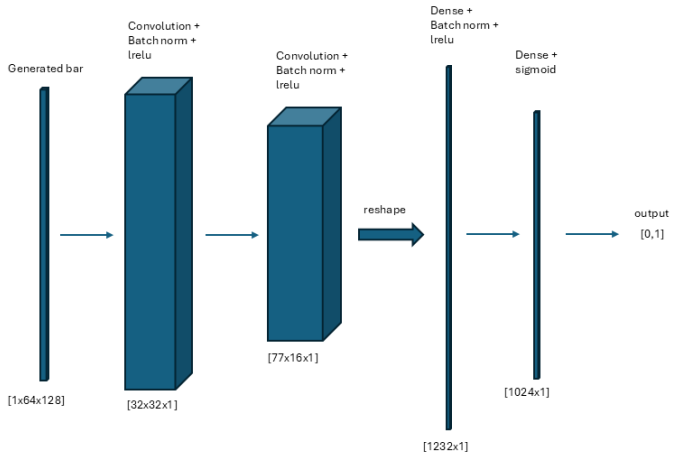


Fig. 3: discriminator network

As a result, $w = 64$ represents the number of time steps in each bar, providing sufficient detail for the generation of complex melodies while maintaining a manageable tensor size.

- **BPMs Filtering:** To ensure consistency in the rhythmic patterns and prevent extreme tempo variations, we filtered the dataset to include only pieces with a tempo ranging between 70 and 110 beats per minute (BPM). This tempo range allows the model to focus on musical pieces with moderate tempos, facilitating better learning of both melody and chord structures without introducing excessive complexity due to extreme rhythmic variations.
- **Bar Splitting:** After analyzing the dataset, we confirmed that all the tracks follow a 4/4 time signature. Consequently, we treated each sequence of 4 beats as a bar and split the piano rolls into individual bars. This allowed us to standardize the temporal structure across the dataset, making it easier to manage and feed into the neural network for training.

Following the above steps, our data preprocessing resulted in a refined subset of 103 MIDI tracks (totaling 38,584 seconds), equating to **31,081** bars of input from the original 1,276 MIDI tracks (715,151 seconds). Since the GAN is a generative model, we omitted dataset splitting into training, validation, and test sets to maximize the use of available data.

V. Learning Framework

A. Generator and Conditioner networks

For the *generator*, we used as input random vectors of white Gaussian noise of length l = 100. The random vector go through two fully-connected layers, with 1024 and 512 neurons respectively, before being reshaped into a 4-by-1 matrix. We then used four transposed convolution layers: the first one use 128 filters of shape 10-by-1 and two strides, the second and third layer uses 128 filters of shape 2-by-1 and 2 stride, the last one use 128 filters of shape 1x128 and stride (1,2).

The *conditioner* takes as input a tensor X_{prev} of dimension w-by-h, that is the previous bar.

The network has a first convolution layer that use 128 filters of shape 1-by-128, stride 2, followed by a dropout layer to perform regularization. The second and the third convolutional layers use 128 filters of shape 2-by-1, with stride 2. The last convolutional layer use 128 filters of shape 10-by-1, with stride 2. The outputs of the intermediate layers of the conditioner are concatenated with the outputs of the intermediate layers of the generator. The overall output is a generated piano roll X_{gen} of shape w-by-h, taking values between [0,1]. The topology of the networks can be seen in Fig. 2a 2b.

B. Discriminator network

The *discriminator* network takes as input the tensor X_{gen} and it is composed first by two convolutional layer that use respectively 32 and 77 filters of shape 2-by-128 and 2-by-1, both with stride 2. After that, there are two linear layers with 1232 and 1024 neurons respectively, followed by a minibatch

discrimination layer[], with 64 learnable features. The output of the discriminator is a scalar between [0,1]. A value is close to 1 if the discriminator classify it to be from real data (i.e., X) and close to 0 if it classify it to be from generated data (i.e., $G(z)$). The topology of the network can be seen in Fig. 3

In our architecture, each Conv2D layer is followed by batch normalization and a Leaky ReLU activation, while each ConvTranspose2D layer is paired with batch normalization and a ReLU activation. In both the discriminator and generator, the final layer applies a logit activation without batch normalization.

C. WGAN architecture

To implement this version, we had to slightly modify the topology of the models presented in V-A and V-B. Specifically, given the nature of the Wasserstein loss training, we removed the sigmoid activation function from the last layer of both generator and discriminator and, for the generator, we replaced it with the Tanh activation function. We also removed the batch normalization layers and replaced them with instance normalization layers. In the WGAN original paper [], this structure has been proven to be more efficient.

VI. Results

While standard metrics for GAN evaluation, such as the Frechet Inception Distance (FID) [12], exist, they are less suited for sequential music generation. Consequently, we assessed our model's performance through visual inspection and auditory analysis, focusing on the melodic appeal and consistency with music theory. The following results reflect careful hyperparameter tuning and extensive effort to determine the optimal complexity balance between the three networks involved in the adversarial training. The optimal topology of the networks is shown in Sec. V. In the following, we present the results from the two models: Model A, the standard GAN, and Model B, the WGAN with GP.

A. Model A: Standard GAN

These are the optimal parameters that emerged from our experiment.

Parameter	Optimal Value
Learning rate	0.0002
Batch size	72
Epochs	30
λ_1, λ_2	0.01, 0.1
num of G updates	2

TABLE 1: Optimal hyperparameters for model A.

The tuning of the parameters λ_1 and λ_2 was essential for the stability of the training. The evolution of the losses is shown in 4. The melody generated by this model resulted to be fluent, and with an accetable grade of variety between one bar and another. However, some artifacts are present in the piano roll images, leading to sections with dissonant or off-key music. Moreover, the melodic structure lacks the complexity expected for a virtuosic piano piece. This may be due to excessive regularization aimed at preventing the model from mode collapsing. An example of the output composed by 3 bars is shown in Fig. 5a.

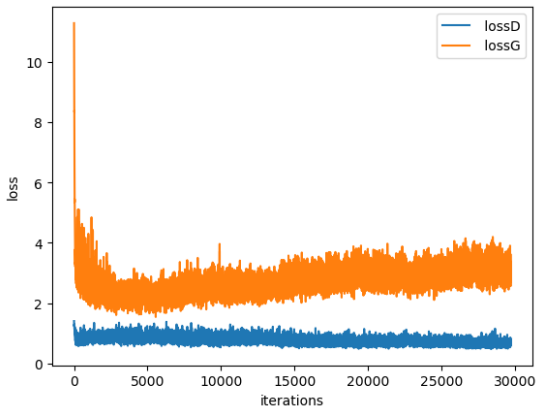


Fig. 4: loss for model A

B. Model B: Wasserstein GAN

The value of the following Hyperparameters reflect the one of the original paper of WGAN [referenced before].

Parameter	Optimal Value
Learning rate	0.0004
Batch size	72
Epochs	50
λ_{GP}	10
num of D updates	5

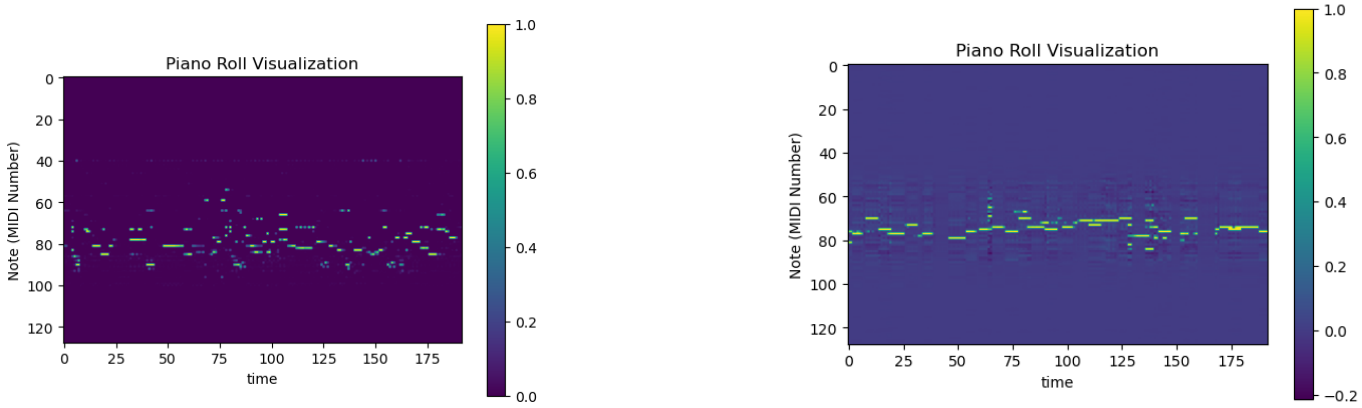
TABLE 2: Optimal hyperparameters for model B.

Using model B, we obtained a melody composed by complex melodic structures and rapid notes, characteristic of virtuosic piano pieces. This result brought us to the conclusion that model B outperformed model A. Some artifacts are still present, likely due to imperfections in the preprocessing steps, but the listening experience of this output is quite satisfactory. An example of output composed by 3 bars is shown in Fig. 5b

VII. Concluding Remarks

In this work, we presented a modified version of the MIDINET architecture, using two methods, the standard GAN and the WGAN, pointing out the fact that the latter outperform the former in terms of similarity with the training dataset of virtuosic piano music.

We put a lot of effort in pre-processing the data to have a meaningfull melody structure, in accordance with music theory. This procedure resulted to be crucial in order to achieve good results using the models. We also had some difficulties stabilizing the training. Therefore, we had to try



(a) output of model A

(b) output of model B

Fig. 5: Generated melody

different approaches. We are reasonably satisfied with the outcomes, which could be improved by using different methods for the pre-processing. The correlation between one bar and another could also be improved by allowing the model to be conditioned on multiple previous bars.

REFERENCES

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[2] L. Yang, S. Chou, and Y. Yang, "Midinet: A convolutional generative adversarial network for symbolic-domain music generation using 1d and 2d conditions," *CoRR*, vol. abs/1703.10847, 2017.

[3] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superglue: Learning feature matching with graph neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4938–4947, 2020.

[4] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[5] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," 2017.

[6] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," 2017.

[7] A. Roberts, J. H. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A hierarchical latent vector model for learning long-term structure in music," *CoRR*, vol. abs/1803.05428, 2018.

[8] Z. Zhang, M. Li, and J. Yu, "On the convergence and mode collapse of gan," in *SIGGRAPH Asia 2018 Technical Briefs*, pp. 1–4, 2018.

[9] Z. Zheng, L. Zheng, and Y. Yang, "Unlabeled samples generated by gan improve the person re-identification baseline in vitro," in *Proceedings of the IEEE international conference on computer vision*, pp. 3754–3762, 2017.

[10] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling factorized piano music modeling and generation with the MAESTRO dataset," in *International Conference on Learning Representations*, 2019.

[11] 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.

[12] Y. Yu, W. Zhang, and Y. Deng, "Frechet inception distance (fid) for evaluating gans," *China University of Mining Technology Beijing Graduate School*, vol. 3, 2021.