Multi Track Music Generation Using Generative Adversarial Networks

Bergamin Eleonora†, Campagnola Stefano†, Monti Sebastiano†

***Abstract*—Generating coherent data samples autonomously, particularly in the realm of music, poses a significant challenge. To produce enjoyable music it is necessary to consider both a precise temporal structure and the coherence between the different instruments playing together. Furthermore, precise rules are needed even when notes are played consecutively and simultaneously, combined into chords and arpeggios. In this report, we investigate the complex task of generating music using state-of-the-art models, based on generative adversarial networks. We have implemented an architecture capable of generating piano-rolls of mono and multi-instrumental polyphonic music both from scratch and conditioned on a previous sample.We trained our model on two different datasets containing both mono and multi-track MIDI samples. In the first case, the dataset encompasses samples of the piano instrument. In the second case, it includes samples of bass, drums, guitar, piano, and strings instruments. Objective evaluation metrics were then implemented to evaluate the quality of the generated music. The results obtained demonstrate that with the unconditioned architecture, we are able to successfully generate musical pieces, while the conditioning mechanism still requires some refinement. This proves that a future improvement to our model may benefit the quality of generated music and its external control.**

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

I. Introduction

A vast application field of Generative Adversarial Networks (GANs) regards the creation of images or videos, however, these architectures are suitable for processing text and audio data. The goals of this project are to use a GAN architecture to generate short, realistic multi-instrument (or multi-track) music samples from scratch and also, given knowledge of a piece of music, complete it coherently. Objective metrics were also implemented to evaluate the quality of the generated traces.

Many challenges are related to the problem of autonomous music generation. The most relevant ones concern the fact that music, to be enjoyable, requires an underlying temporal structure that defines its rhythm, made up of recurring patterns, such as bars and beats. Furthermore, music is generally composed of multiple instruments that may seem to follow different temporal dynamics when listened to alone, but which instead define a single dynamic or harmony when performed together. Finally, in music, there can also be groups of notes that can be played together, as in chords or arpeggios. Many studies make use of long short-term memory networks (LSTMs) and recurrent neural networks (RNNs) to reproduce musical structures, however, these architectures may not detect note pattern dependencies over longer periods of time.

An interesting solution that has been proposed in recent years to overcome these challenges is given by GAN architectures. These models allow the generation of realistic-sounding music samples by training a generator and discriminator against each other in a minimax game. A recent implementation was performed by Dong et al. [1] with their MuseGAN architecture. In this project, our aim was to replicate the remarkable outcomes attained by the MuseGAN architecture, while also exploring potential enhancements to its structure for generating high-quality music samples. Our approach involved streamlining the architectures of both the generator and discriminator, leveraging diverse datasets for music generation, including a dataset featuring single-track music. Additionally, we implemented a mechanism for conditioned music generation, drawing inspiration from the approach employed by Yang et al. in their MidiNet architecture [2].

The quality assessment of generated samples is a challenging, but still fundamental task when dealing with GANs. While the MuseGAN paper does incorporate objective metrics, we drew inspiration from the *MusPy* study [3] for our evaluation process. Specifically, we implemented a selection of metrics proposed in their work.

This report unfolds in the following manner. In Section I we describe previous work on GANs in general, state-of-the-art architectures in the music generation realm, and music evaluation techniques. The designed pipeline and data preprocessing are respectively presented in Sections III and IV. The proposed GAN model, the training setup, and the evaluation metrics are explained in detail in Section V and the obtained results are shown in Section VI. Concluding remarks are provided in Section VII.

II. Related Work

Generative Adversarial Networks were first introduced by Ian Goodfellow [4]. In this implementation, GANs are conceived as particular types of generative models capable of learning the probability distribution p_{data} of a given dataset and calculating an estimate p_{model} of this distribution. The architecture is basically made up of two parts:

- **Generator:** Its role is to produce synthetic samples that closely resemble real ones. It is trained with the objective of deceiving the discriminator.

†Department of Physics and Astronomy, University of Padova. Emails:
eleonora.bergamin@studenti.unipd.it
stefano.campagnola@studenti.unipd.it
sebastiano.monti@studenti.unipd.it

- **Discriminator:** This component is responsible for classifying the samples, discerning between real and generated ones.

These can be seen as two players in an adversarial minimax game, in which the loss function is given by Equation (1).

$$J^{(D)} = \mathbb{E}_{x\sim p_{data}}\left[\log D\left(x\right)\right] - \mathbb{E}_{z\sim p_z}\left[\frac{1}{2}\log\left(1 - D\left(G\left(z\right)\right)\right)\right]$$
$$J^{(G)} = -J^{(D)} \tag{1}$$

Where x represents a real datum, sampled from p_{data}, z is a random vector, sampled from the latent space distribution p_z and $G\left(.\right)$, $D\left(.\right)$ respectively, represent generator and discriminator networks.

In general, it is difficult to achieve convergence towards the extreme of the objective function in a GAN architecture, since the cost functions are not convex and the parameter space is high dimensional. To overcome this problem, some methods have recently been introduced to improve the stability and convergence of GANs in the training process [5], [6], [7].

A. Convolutional GANs and Wasserstein GANs

A first significant improvement was introduced by Deep Convolutional Generative Adversarial Network (DCGAN) [8], in which it is demonstrated that deep convolutional layers can be used inside generator and discriminator architectures with a resulting increase in performances and stability.

Other studies [9] showed that the training process can be stabilized and the problem of vanishing gradients could be avoided, by two main modifications in the loss function. The first is the introduction of the Wasserstein distance instead of the Jensen-Shannon's divergence, while the other is the addition of a regularization term called gradient penalty. Heuristically, the Wasserstein distance addresses the vanishing gradient problem by enhancing the reliability of gradients. On the other hand, the gradient penalty term promotes gradient norms to stay below one. The new type of GAN architecture is called WGAN-GP. The expression of the loss for this kind of network is:

$$\begin{aligned} L =& \mathbb{E}_{x\sim p_{data}}\left[D\left(x\right)\right] - \mathbb{E}_{z\sim p_z}\left[D\left(G\left(z\right)\right)\right] + \\ &+ \lambda\mathbb{E}_{\hat{x}\sim p_{\hat{x}}}\left[\left(\|\nabla_{\hat{x}}D\left(\hat{x}\right)\|_2 - 1\right)^2\right] \end{aligned} \tag{2}$$

Where λ is the penalty coefficient and $p_{\hat{x}}$ is a uniform sampling distribution along straight lines between pairs of points sampled from p_{data} and p_z.

B. MuseGAN and MidiNET architectures

The techniques explained above are at the basis of one of the most important models using GANs for music generation: the MuseGAN architecture [1], on which our architecture is mainly inspired. MuseGAN generates multi-track sequences of polyphonic music, either from scratch or conditioned on a given track by working with a symbolic representation of MIDI tracks, specifically as piano-rolls. The model relies on a WGAN-GP architecture, with the temporal structure derived from a temporal generator that maps a noise vector into a series of latent vectors. These vectors are then passed to a bar generator, responsible for sequentially creating each bar.

The paper introduces three distinct methods for managing track interactions, enabling the architecture to generate music through jamming, composing, or a combination of both techniques. Additionally, the study implements a set of intra- and inter-track objective metrics to provide a comprehensive evaluation of the models.

Another architecture which we took inspiration on for its conditioned generation mechanism, is the MidiNet model [2]. MidiNet has in its core a DCGAN implementation and is able to generate single-track melodies either from scratch, by following a chord sequence, or by conditioning on the melody of previous bars. The conditioning mechanism is different depending on the specific task. Melody generation with no chord condition, encodes the 2D condition represented by the melody of the previous bar, by using repeated convolution and then concatenates the encoded vectors to the (mirrored) layers of the generator. Melody generation with chord condition instead, adds also a 1D condition in the transposed convolutional layers of the generator, to take into account the chord progression of the track.

C. Music evaluation

Model evaluation is a fundamental aspect when assessing the quality of generated music samples. However, due to the subjective nature of musical experiences, employing conventional quantitative metrics may not yield highly meaningful results. This is primarily because music, being an art form, lacks rigidly defined rules. However, there is still a need to assess the performance of the model, and quantitative evaluation of music generation remains a challenge. Usually, these metrics are broadly categorized into pitch-related and rhythm-related metrics [3].

III. Processing Pipeline

In this section, we present a concise overview of the complete model pipeline and its design.

As illustrated in Fig. 1, songs from the dataset are partitioned into musical samples, which are then fed into the discriminator. In parallel, the generator produces synthetic samples by transforming random latent code, and optionally, conditioning elements, into a musical excerpt. Subsequently, the discriminator assesses both real and fake samples, and these scores are used iteratively to train both architectures.

Finally, the trained generator generates a substantial number of samples. The distribution of these samples is then analyzed and compared with that of the original dataset.

IV. Signals and Features

Two primary datasets were considered for this project: the *VGMIDI* unlabelled dataset [10] and the *LPD-5 Cleansed* dataset [11]. The former comprises 3850 MIDI files, while the latter consists, of 21425 MIDI files.

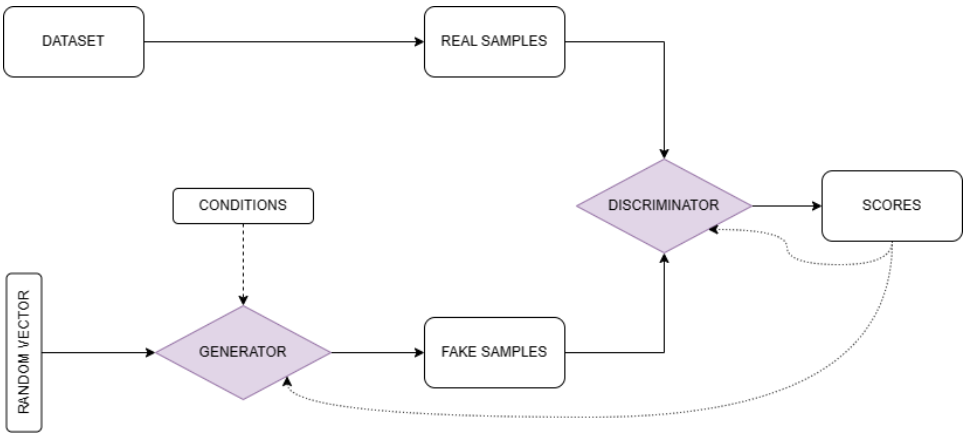


Fig. 1: Processing pipeline of the project. Continuous lines represent the forward pass, while dotted lines the information flow that backpropagates. The dashed line stands for the eventual use of conditional generation.

MIDI format is a protocol used to encode musical information and instructions. The main messages, that describe how a note should be played, are note-on, note-off, pitch, and velocity. MIDI representation offers more straightforward handling compared to audio waves, as the specific identity of the instrument playing the notes is not crucial.

The *VGMIDI* dataset primarily contains song excerpts featuring either a single piano track or at most two piano tracks playing together. In contrast, the *LPD-5 Cleansed* dataset presents a more intricate composition with five instrumental tracks: *Drums*, *Piano*, *Guitar*, *Bass*, and *Strings*. For simplicity, second piano tracks in the *VGMIDI* dataset are treated as separate songs, resulting in a collection of exclusively 1-track songs. Additionally, from the *LPD-5 Cleansed* dataset, we extracted the *LPD-5 1-track* dataset, which exclusively consists of the piano tracks.

A. MIDI conversion to piano-roll

The preprocessing steps for both datasets are identical. Firstly, all notes in each song are binarized, meaning the intensity of all played notes is set to 1. This decision was guided by the need for more uniform data. Secondly, the beat resolution of all tracks is set to a constant value, in this case, 4, making a quarter of beat our temporal unit. This means that a quarter of a beat corresponds to a sixteenth note. While this represents a significant temporal compression, as sixteenth notes become the smallest allowed structures, a higher beat resolution proved more challenging to train within a reasonable timeframe. Given that most basic songs do not feature shorter note durations or odd tuplets of those, we opted for a more straightforward approach to music generation.

Next, the MIDI encoding for each file is translated into a piano-roll representation, denoted as $\{0,1\}^{M\times T\times P}$, where M represents the number of tracks, T is the number of elementary time steps, and P is the number of pitches in the song. Although piano-roll provides a visual representation of music structures akin to an image, it has limitations. Notably, it does not provide information about note-off events, making it impossible to distinguish between rapidly repeated short notes and sustained long notes [12].

The MIDI files require additional processing before they can be fed into our architecture. In order to ensure consistency and improve the quality of the generated samples, we restrict the pitch range to fall within the notes from $C0$ to $C7$. This adjustment helps in avoiding extremely high or low tones, which are relatively less common in most musical compositions.

Furthermore, we exclude files that contain less than 10 notes in every track from the dataset.

B. Songs slicing mechanism

From the resulting piano-rolls, a specified number of 4-bar chunks (either 16 for the *VGMIDI* dataset or 8 for the *LPD-5 Cleansed*) are selected. These starting points are intentionally aligned with the natural structure of the bars, each consisting of 4 beats, and are chosen randomly from the available options. Additionally, for single-track datasets, we only consider samples where at least 10 notes are played.

In the case of conditioned generation, we take two adjacent 4-bar chunks. The first chunk is used to condition the generator, while the second is employed to compute the loss for the discriminator.

The resulting dataset from this process is substantial, comprising approximately $N \simeq 70000$ samples for the *VGMIDI* dataset and $N \simeq 100000$ samples for the *LPD-5 Cleansed* dataset. Given the ample size of the data collected, there are currently no immediate plans to implement additional data augmentation techniques.

In the case of unconditioned generation, the resulting binary dataset has a final shape of $N \times M \times 64 \times 84$. In contrast, for conditioned generation, it has a shape of $N \times 2 \times M \times 64 \times 84$.

V. Learning Framework

In this section, we introduce the architectures for both the generator and the discriminator, as well as outline the training algorithm and metrics utilized for evaluating the

generated samples. Notably, our investigation covers both an unconditioned GAN and a conditioned GAN. It's essential to emphasize that the training procedure and evaluation remain consistent across both setups, with identical datasets provided to both models.

A. Generator

The generator's structure is constructed using transpose convolutional blocks, where a 2D transpose convolution is succeeded by a 2D batch normalization layer and a *ReLU* activation function. Starting from a latent code of dimension 32 (Fig. 2), the input passes through several transpose convolutional blocks, which consecutively unroll one dimension (temporal and pitch) at a time. This characteristic is pivotal in distinguishing piano-roll generation from conventional image generation. The last layer of the architecture requires, contrary to the blocks, a *Tanh* activation function. While a *Sigmoid* activation might seem a more intuitive choice due to its output range of 0 to 1, we observed that in practice, the model failed to learn effectively. This decision aligns with the approach in our reference paper [1].

B. Discriminator

The discriminator's structure mirrors that of the generator (see Fig. 2, to the right). The primary distinction lies in the dimensionality of the convolutional kernels, set to 3. This addition corresponds to introducing an extra dimension associated with the considered bar of the sample, in order to roll up the bars singularly. The objective of the discriminator is again to pack a dimension at a time in each convolutional layer, starting from the temporal one. Unlike the generator, however, each layer is succeeded solely by a *LeakyReLU* activation function. The last blocks of the net are two feed-forward layers, whose final output the score associated to that given sample. The reason we call it "score" instead of classification will be more clear in the next sections.

C. Conditioned Generator

This version of the generator is a modified iteration of the aforementioned one. It incorporates the use of a sample from the dataset to condition the generation of a new sample [2]. Consequently, the conditioned generator accepts both a random vector and a dataset's sample as input, extracting meaningful information from the latter to "continue" the sample. The architecture of the conditioned generator can be obtained simply by attaching to the layers of the standard generator some intermediate products. These intermediate products are derived by a small encoder network that tries to extract relevant features from the conditioning sample.

D. Training setup

Our GAN architecture is designed to operate within a WGAN-GP framework, thereby adopting the loss function outlined in Section II (Eq. 2). The generator's objective is to maximize the scores of its samples, while the discriminator seeks to maximize the gap between scores of real and fake samples. Crucially, the gradient is regulated by the third term in the loss function, with λ set to the value of $\lambda = 10$.

The training process employs two Adam optimizers, one for the generator and one for the discriminator. In both cases, learning parameters are configured with $lr = 10^{-3}$, $\beta_1 = 0.5$, and $\beta_2 = 0.9$, aligning with the approach in [1].

Differently from MuseGAN architecture, in which the authors trained the generator after 5 iterations of the discriminator, we trained both architectures at the same speed. This training mechanism at different rates was not tested, however the similar TTUR [13] strategy of training generator and discriminator at different learning rates was considered, without however showing again any significant improvements on training process and quality of generated samples.

While hyperparameter tuning would have been preferable, limited computational resources prevented us from conducting a GridSearch. For this reason, these and other hyperparameters were instead heuristically tuned in preliminary tests and then kept fixed in all performed trainings (unless differently specified).

The agreed value for batch size is kept to 64 samples in order to have a reasonable computation time, while the needed amount of training is set to 50 epochs.

E. Evaluation metrics

The developers of *MusPy* [3] introduced an extensive range of quantitative metrics encompassing pitch and pattern characteristics, offering valuable insights into the quality of generated music. Notably, this set surpasses the scope of metrics outlined in our reference paper. From this comprehensive collection, we carefully selected and implemented specific metrics for our in-depth analysis of results.

The pitch-related metrics we have selected delve into various facets of the music, including:

- pitch range: range of the used pitches in the composition
- pitch classes used: number of unique pitch classes employed
- polyphony: average number of pitches being played concurrently
- polyphony rate: ratio between the number of timesteps when multiple pitches are played together to the total number of timesteps. We have set a threshold for the number of concurrent pitches at 3 or more in order to have a meaningful metric.
- scale consistency: highest pitch-in-scale rate across all major and minor scales
- pitch entropy: Shannon entropy of the normalized note pitch histogram
- pitch class entropy: Shannon entropy of the normalized note pitch class histogram

In the realm of rhythm, our chosen metrics consider:

- non empty beat rate: ratio of the number of non empty beats to the total number of beats
- drum-in-pattern rate: assesses the alignment of the drum track with a standard 8/16-beat pattern.

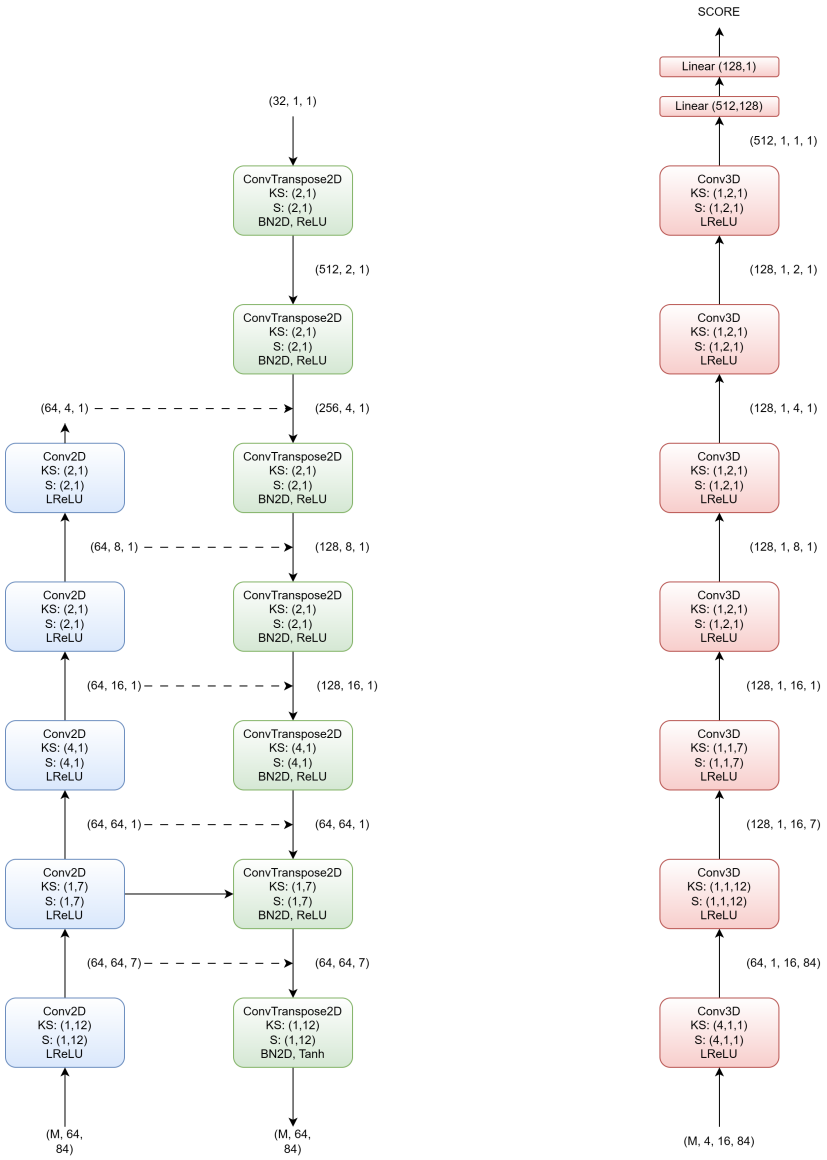


Fig. 2: Architectures of the generator and the discriminator. Dashed lines represent array concatenation in case of conditioned model.

The considered metrics provide a structured framework for objectively assessing the quality of generated music samples. These metrics were employed for both unconditioned and conditioned samples. For the unconditioned samples, we simply visualize the metric distributions of the dataset. However, for conditioned samples, this initial analysis is insufficient. To gain a more profound understanding of the correlations between the generated samples and their conditioning counterparts, we compute the differences in metrics between the conditioning sample and the generated one for each pair of samples. We anticipate that these differences will predominantly center around 0, exhibiting certain variances. Yet, to lend meaningful context, the comparison with a null model is crucial. Consequently, we randomly pair conditioning samples with conditional generated samples and compute the metric differences between the elements within these new pairs. As anticipated, we expect the generated music to exhibit a higher correlation with the conditioning samples as compared to random selections. Consequently, the variance in the first distribution should be smaller than that in the second. This comparative analysis is instrumental in providing valuable insights into the effectiveness of our conditioning approach.

We introduced a weighted version of scale consistency to place greater emphasis on the most pivotal notes within the scale. This adjustment was made to address a scenario where instruments playing fewer notes or primarily single notes, rather than chords, could receive the same scale consistency score across different scales. By assigning slightly higher importance to the first, third, and fifth degrees of the scale, we aimed to provide a more nuanced differentiation between

Number of tracks	Dataset	Track names	Samples per song	Generation
1	*VGMIDI*	*Piano*	16	Scratch Conditioned
1	*LPD-5*	*Piano*	8	Scratch Conditioned
5	*LPD-5*	*Drums Piano Guitar Bass Strings*	8	Scratch Conditioned

TABLE 1: Summary of parameters combination for each dataset.

scales. However, upon evaluation, we found that this technique did not lead to substantial changes in the statistical results and metrics. Consequently, we chose not to incorporate this improvement into the final evaluation.

VI. Results

Several training procedures were carried out on our network architecture, in order to assess its performance on the conditions and purposes we wanted to deal with. The main conditions that were tested, involved the variation of the dataset (*VGMIDI* or *LPD-5*), the number of tracks (1 or 5), the number of samples per song (16 for *VGMIDI* dataset and 8 for *LPD-5*) and the type of generation (from scratch or conditioned on the previous bar). We highlight that the music samples involved in the training of 1 track *LPD-5* dataset, are the ones that represent the piano track. A summary of parameter combinations is summarized in Table 1.

A. Losses trends

Examining Fig. 4, we observe the convergence of generator and discriminator losses across all examined configurations, albeit more distinctly in the cases of unconditioned generation with a single piano track (Panels 4a, 4c, 4d). It is conceivable that the more complex architectures may have benefited from additional training, owing to their larger parameter space.

It is crucial to note that, despite the dataset and model setup, the values of losses at convergence tend to be quite similar. Nevertheless, their interpretation still lacks specific meaning.

In an effort to improve the learning process, we attempted training with weights initialization in the architecture, coupled with the introduction of random noise to the input samples fed to the discriminator for the configuration of 1-track from scratch and *VGMIDI* dataset. The resulting losses are illustrated in Figure 3. Given that this approach did not yield significant improvements in convergence or sample quality, we opted not to incorporate it into the training process.

B. Generated music

Generated music is produced by inputting random noise Gaussian vectors with appropriate dimensions into the pre-trained generator. Since the final layer of the generator employs a *Tanh* activation, yielding values within the range

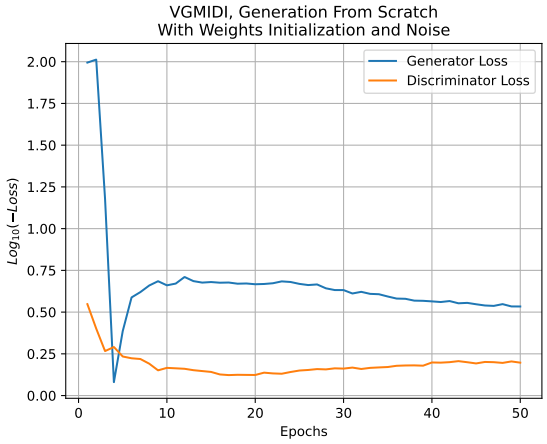


Fig. 3: Plot of the logarithm of the loss for the architecture with weights initialization and random noise.

of $(-1, 1)$, it is necessary to binarize these values before formatting them into a piano-roll. To achieve this, we establish a threshold, such that all outputs above 0.5 are interpreted as ones, while those below are considered zeros.

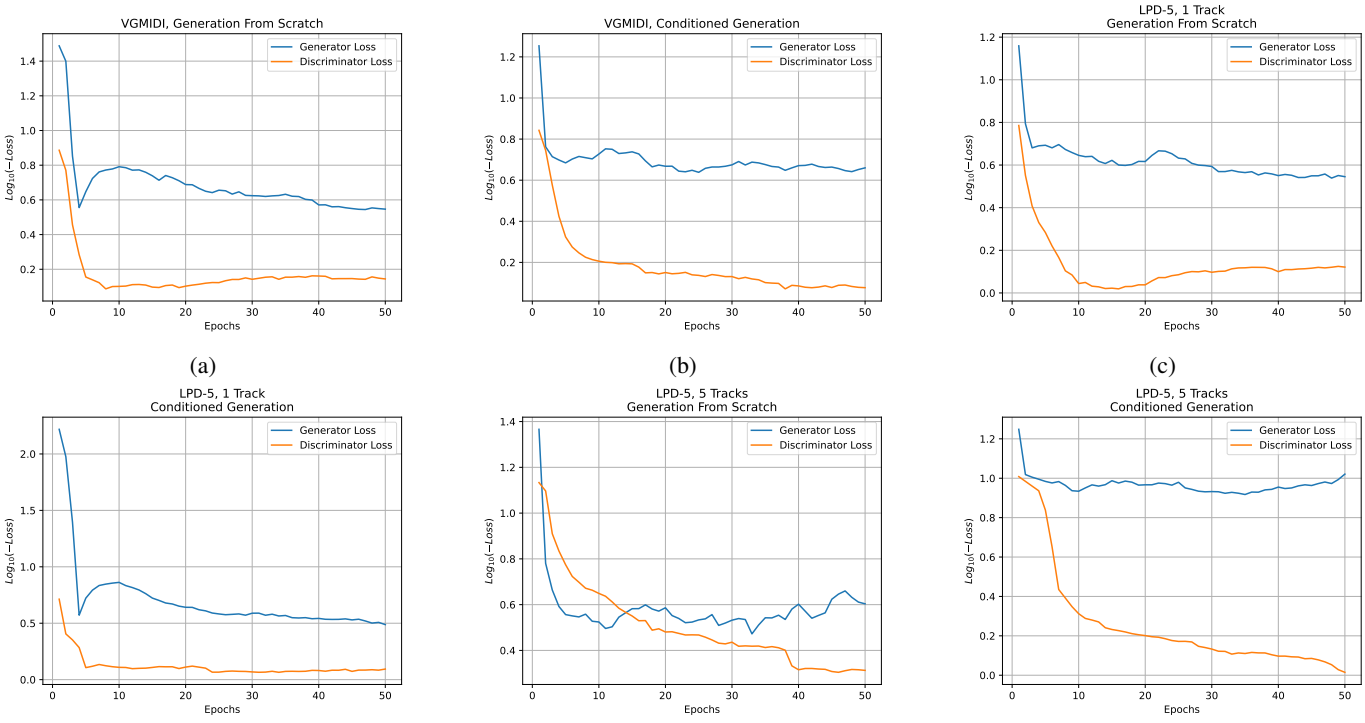
Figures (8a) and (8b) illustrate two examples of generated data samples with 1 and 5 tracks, respectively. In the instrumental tracks, it is evident how notes frequently converge to form chord-like structures, indicating harmonious relationships. Additionally, all tracks appear to adhere to the same musical scale, as evidenced by the consistent pitch range of each track. Moreover, in the samples with 5 generated tracks, one can observe that each instrument exhibits characteristic behavior akin to real-world scenarios. The drums typically adhere to 8- or 16-beat rhythmic patterns. The guitar, piano, and strings often play chords, interspersing moments of silence to provide space for solos from other instruments.

C. Quantitative evaluation of the models

We proceed with a detailed analysis of the generated music across different setups, focusing on the comparison of metric distributions. When examining unconditioned samples, an effective generator is expected to produce samples with metric distributions closely resembling those of the original dataset.

Starting with the unconditioned models trained on *VGMIDI* and *LPD-5 1-track* datasets (Fig. 6a and 6b), we observe that the generated data distributions approximate the dataset's distributions quite accurately. However, it's worth noting that the produced samples tend to utilize a larger number of pitches and a wider pitch range.

Similar trends are evident in the unconditioned model trained on the *LPD-5* dataset (Fig. 6c). While there is a general resemblance in the distributions, pitch variability is not yet managed optimally. Examining other instrumental tracks like *Guitar* and *Strings*, we observe a correct use of chord structures in the polyphonic trend, albeit with some fragmentation. The *Bass* track (Fig. 6d), which is typically expected to play single-note or sparse-note phrases, occasionally exhibits undesired polyphony and high pitch outliers.



(a) (b) (c)

(d) (e) (f)

Fig. 4: Plots of the logarithm of the losses for cases listed in Table 1.

Results from the conditioned models reveal interesting behaviors. It is important to notice that in the conditional generation case, the generator is provided with new samples, distinct from those used during training. Specifically, the generator is fed the samples previously used in the discriminator's training process in the conditional case. This precaution was taken to prevent potential correlations between the training dataset and the generated samples, as our primary focus is on the effects of conditioning. For both *VGMIDI* and *LPD-5 1-track*, the generated music aligns with the previously observed unconditioned distributions, indicating a consistency in the ability to learn similar features. The correlation with conditioning samples, which represent the music excerpts to be "continued", is assessed by comparing the variances of the distributions (Fig. 7a and 7b). This comparison highlights that the model trained with *VGMIDI* generates music samples that closely match the conditioning samples, whereas the model trained with *LPD-5 1-track* doesn't display any notable correlation. This outcome may be attributed to the absence of a loss constraint specifically addressing the utilization of conditioning information. In the case of *LPD-5 5-track*, the conditioned model perfectly mirrors the dataset distribution and exhibits a strong correlation with the conditioning samples (Fig. **??**). This is clearly visible in (Fig5), where the conditioning drum pattern is proposed as guitar track sample. This behavior is indicative of the fact that the generator learned to output the conditioning samples provided to it (although a pattern in one track might be reproduced in a different track), probably exploiting the fact that music is repetitive and has patterns. This phenomenon was not discussed in [2], and it remains unclear whether it can be attributed to an excessive flow of information from the auxiliary conditioner architecture.

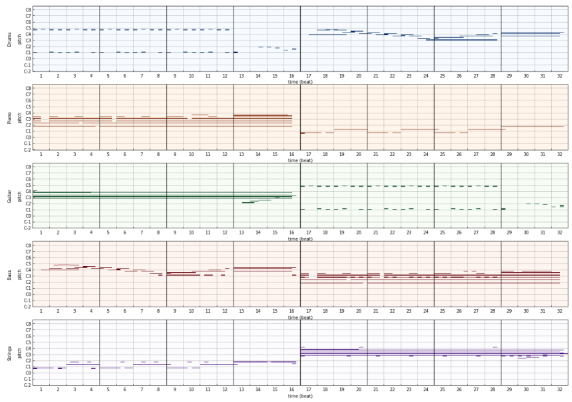
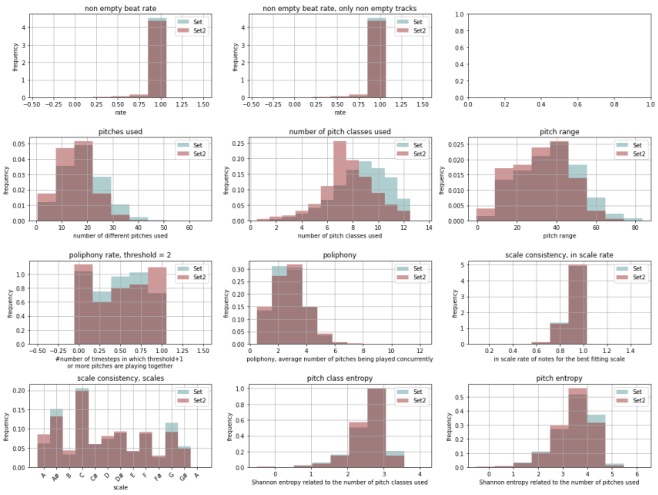


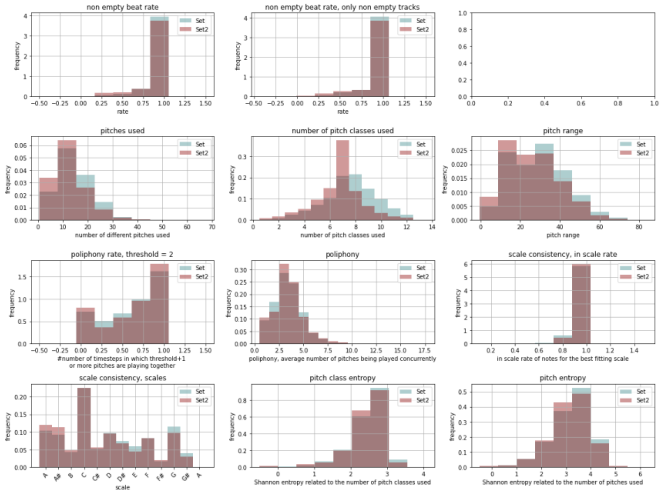
Fig. 5: Comparison between the conditioning sample and the conditioned generated sample for the *LPD-5 5-track* dataset.

VII. Concluding Remarks

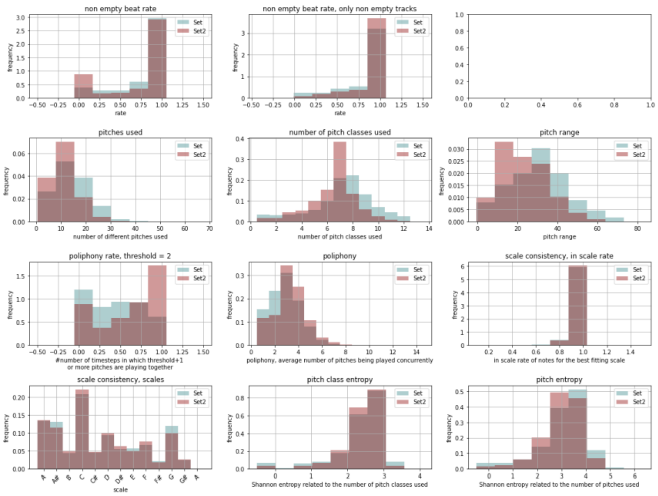
In this study, we tackled the task of music generation, devising architectures capable of producing piano-roll representations of both monophonic and polyphonic compositions. These models demonstrated proficiency in generating music



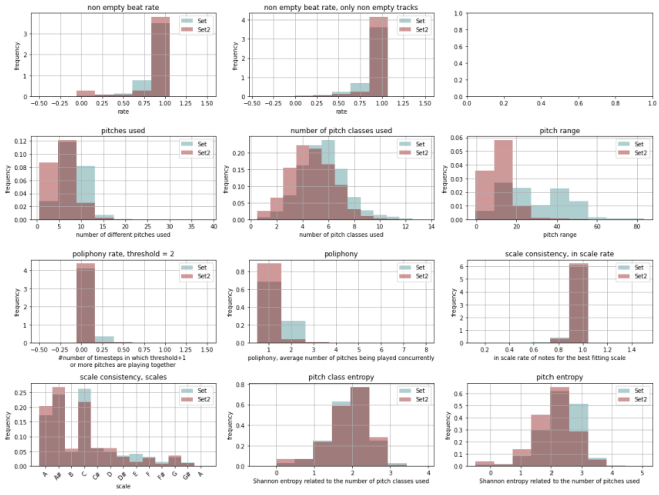
(a) *VGMIDI*, *Piano*



(b) *LPD-5 1-track*, *Piano*



(c) *LPD-5*, *Piano*



(d) *LPD-5*, *Bass*

Fig. 6: The figure displays the distributions of the music generated by the unconditioned model, represented in blue, in comparison to those of the dataset, shown in red.

from scratch or conditioned on preceding samples, and exhibited consistent performance across two distinct datasets.

Our incorporation of an extensive array of metrics allowed for a quantitative evaluation of the generated compositions, providing a robust foundation for further advancements in our model. Future endeavors may involve a more comprehensive exploration of hyperparameters, given the availability of ample computational resources. Additionally, deeper investigations are needed into the role of the conditioner and the type of information it should provide to the generator, aiming to prevent potential over-reliance on conditioning samples. A remaining challenge lies in effectively handling the observed diversity in pitch and pitch range across various instrumental tracks. An ongoing challenge lies in effectively managing the observed diversity in pitch and pitch range across various instrumental tracks. It is plausible that nuanced architectures designed to tailor the behavior of generated tracks could lead to significant strides in this domain.

REFERENCES

[1] 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," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.

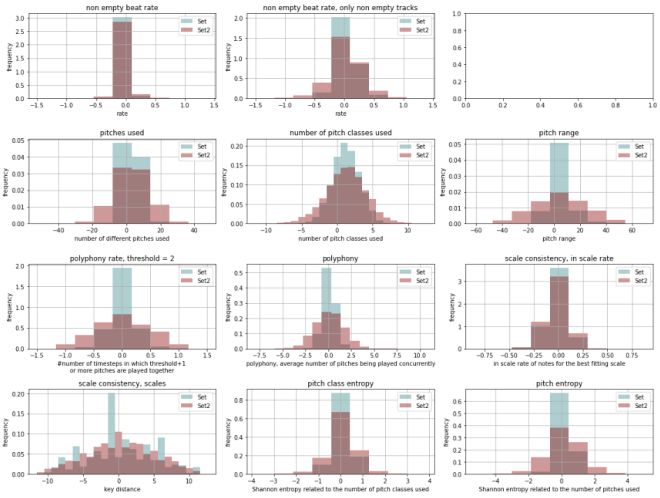
[2] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, "Midinet: A convolutional generative adversarial network for symbolic-domain music generation," arXiv preprint arXiv:1703.10847, 2017.

[3] H.-W. Dong, K. Chen, J. McAuley, and T. Berg-Kirkpatrick, "Muspy: A toolkit for symbolic music generation," ArXiv, vol. abs/2008.01951, 2020.

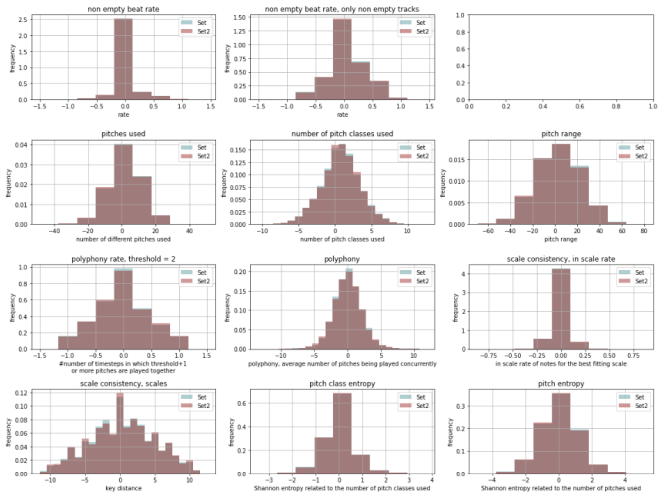
[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," Communications of the ACM, vol. 63, no. 11, pp. 139–144, 2020.

[5] 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.

[6] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," arXiv preprint arXiv:1710.10196, 2017.



(a) *VGMIDI*, *Piano*



(b) *LPD-5 1-track*, *Piano*

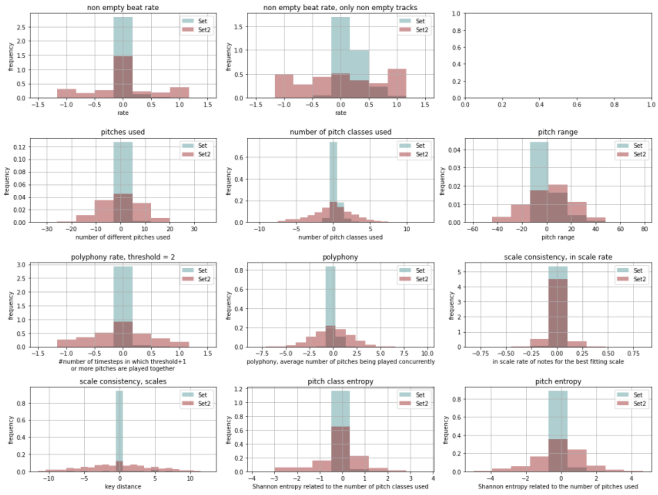


Fig. 7: The figure displays the distributions of the music generated by the conditioned model, represented in blue, in comparison to those of the dataset, shown in red. It is worth noting that in Figure **??**, the results may appear satisfactory simply because the network is occasionally reproducing the input itself.

[7] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," Advances in neural information processing systems, vol. 30, 2017.

[8] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," arXiv preprint arXiv:1511.06434, 2015.

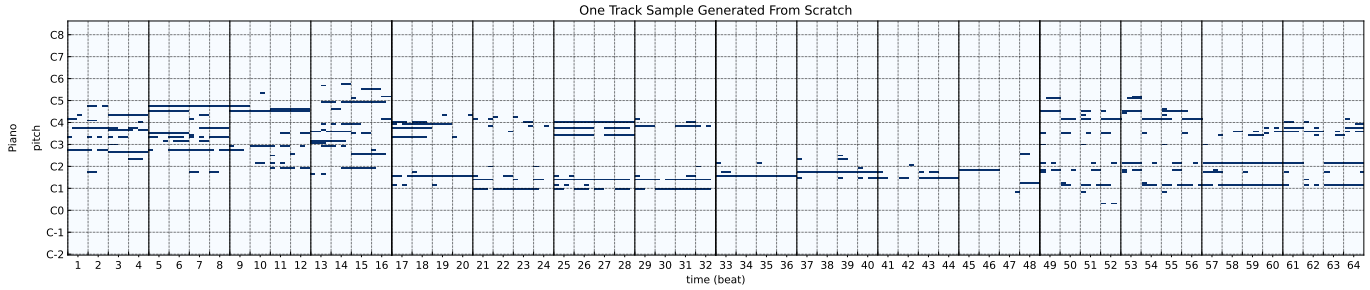
[9] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in International conference on machine learning, pp. 214–223, PMLR, 2017.

[10] L. N. Ferreira and J. Whitehead, "Learning to generate music with sentiment," 2019.

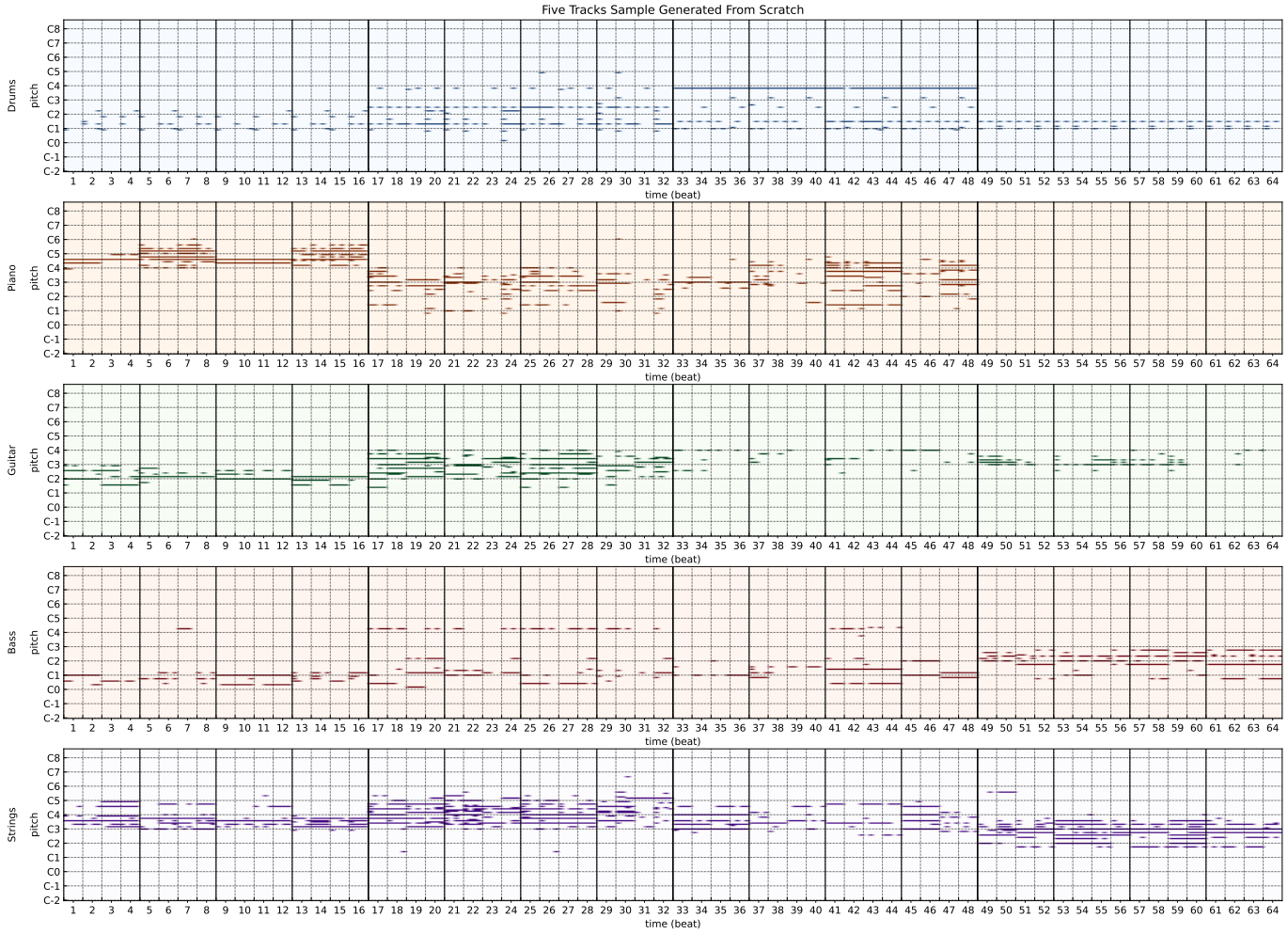
[11] C. Raffel, Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching. PhD thesis, Columbia University, 2016.

[12] C. Walder, "Modelling symbolic music: Beyond the piano roll," 2016.

[13] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," 2018.



(a)



(b)

Fig. 8: Examples of generated piano-rolls after training. They are constructed joining 4 different generated samples of 4 bars.