Music generation with CNN-based GAN and VAE

Erica Brisigotti (2097202), Ekaterina Chueva (2072050)

Abstract—**Music generation task is a longstanding field of research: however, only recently, CNN-based models were proposed to solve this task, leading to new developements. In this project report, we implement preprocessing techniques for symbolic music datasets and we explore two convolutional neural network (CNN)-based models - Generative Adversarial Network (GAN) and Variational Autoencoder (VAE). Lastly, we propose techniques to evaluate the performances of the models both qualitatively and quantitatively. Moreover, we analyze the impact of data augmentation on the quality of the generated music. Our findings suggest that, for this task, GAN architecture outperforms the VAE one. For data augmentations, no significant improvement was noticed, likely due to the limited computing resources. Lastly, we suggest possible prospects for future research on CNN-based VAE to potentially improve its results.**

Index Terms—**Music Generation, Generative Models, Unsupervised Learning, Optimization, Neural Networks, Convolutional Neural Networks, Variational Autoencoders.**

I. Introduction

Historically, music generation has been an area of interest in computer science since 1959, with the first recorded algorithmic composition. This first implementation often resulted in an oversimplification of music, in mechanical sound and, most importantly, in a lack of originality.

To try to solve these issues, starting 1989, shallow neural networks were used for music generation. In recent years, deep neural networks (DNNs) have proven to be able to learn complex patterns from large collections of data also for music generation. In fact, lots of DNN models for music generation have been proposed over the past ten years. [1]

The majority of these models use recurrent neural networks (RNNs) to keep into account the sequential nature of music: in fact, past music events are used to condition the generation of new music.

Fewer attempts have been made with deep convolutional neural networks (CNNs), which are typically faster to train and more parallelizable: the two most notable examples are WaveNet, which showed the possibility of using CNNs for music generation, and MidiNet, which heavily inspired the GAN architecture of this work. [2]

This project aims to solve the music generation task, specifically:

- to implement a GAN architecture, analogous to the MidiNet architecture;
- to explore the possibility of implementing a VAE architecture, CNN-based as well;
- to test if using data augmentation techniques on the training set can improve the performance of GAN and VAE;
- to compare the performance of GAN and VAE, with and without data augmentation, to find the best performing combination.

We carefully discuss the preprocessing techniques we implemented in this work and that can be later used in other similar tasks. Furthermore, we report all the limitations of this work and possible prospects of future research, especially considering VAE architecture.

This report is structured as follows: in section II we describe the models that were previously proposed to tackle music generation task; the bigger picture of our work can be found in III; the dataset and proposed preprocessing techniques, the networks' architectures and training/generating details as well as evaluation metrics are reported in IV; the results are shown in section VI; concluding remarks are provided in section VII.

II. Related Work

Music generation task has been widely explored, resulting in different neural network architectures proposed. The generation tasks can be divided into five groups, depending on having a condition/control: generation from scratch, conditional generation, controllable generation, performance generation, and interactive generation [1]. In this section we briefly discuss methods that have been proposed for music generation historically and focus on conditional melody generation task in particular.

RNN is a suitable model for learning sequential data and it was the first neural network used for music generation in 1989 by Todd [1]. Later, in order to solve the vanishing gradient problem encountered by RNN, a newly proposed special architecture of RNN (LSTM) was used for music generation as well. The capability of a LSTM models to store and recall is useful for generating music features. Nowadays, there are many RNN-based implementations for conditional melody generation, such as CRMCG [3], BebopNet [4] and others.

With advancement of neural networks, other architectures became possible, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). RNN-based GANs and VAEs are a popular solution for music generation.

Among the RNN-based VAEs there are SurpriseNet [5], Music SketchNet [6], InpaintNet [7]. The first neural network, SupriseNet, generates harmonic accompaniments for given melodies and was the first VAE-based melody harmonization work [8]. Other two architectures, InpaintNet and SketchNet, explore the idea that human-composed music is usually iterative and non-sequential, while previous approaches assumed that music is generated sequentially, depending only on the previous segment. The approach of generating music in these two architectures is called music inpainting, referring to adding missing information in a music piece, which resembles more the human process of creating music. VAE's latent space makes possible to explore diversity in music inpainting. SketchNet, superior to the InpaintNet, in fact allows the user to insert their specific idea and fills in missing measures in monophonic music according to them and the surrounding context. Both of these networks suffer from inability to restrict the generating process exactly from the start to the end of the bar.

Beside RNNs, other popular basic architecture is a CNN. The first neural network showing that CNNs are capable of music generation was WaveNet [9]. Other purely CNN-based model, Coconet [10], as well explored music inpainting, however was able to complete only fixed-range music segments. Application of CNNs to the music generation task inspired appearance of CNN-based GANs, such as MIDINet [2]. This model focuses on generating music one bar after another, conditioning on the previous bar and, optionally, chords. In this work we use the approach described in the MIDINet paper for the chosen dataset in order to generate piano melody (we provide further details in the following sections) and extend this approach in order to try the CNN-based VAE for melody generation task.

III. Processing Pipeline

The bigger picture of our project's pipeline is shown on the fig. 1. We start with the original dataset (MIDI files), we then move forward to the preprocessing part (denoted with orange colour), optionally we perform data augmentation, then we finally pass the preprocessed data to the one of the two neural networks (VAE or GAN), which gives us the new music pieces.

In this project we use symbolic representation of music, which is a $h \times w$ matrix of real values. In this representation h represents number of notes and w stands for the number of time steps is used in a bar. For the original matrices we use 1 if a given note is played during a given time step and 0 otherwise. Resulting matrices are called piano rolls. Our first preprocessing step is creating such piano rolls out of the original files.

Then obtained piano rolls need to be further preprocessed. We proceed with separating chords from the melody, since in this project we focus on melody generation. The chords we consider are: 12 major chords (C, C#, D, D#, E, F, F#, G, G#, A, A#, B) and 12 minor chords (Am, A#m, Bm, Cm, C#m, Dm, D#m, Em, Fm, F#m, Gm, G#m, Am, A#m, Bm). These chords were separated from all the octaves, e.g C3 and C4. As a result, we obtained pure melody piano rolls.

These melody piano rolls then face the last step of the preprocessing, which is shifting all the notes to a fixed range of notes: from C4 to B5. As described in [2], this results in a better control of the model.

After this, we optionally perform data augmentation, which consists of shifting melodies to different fixed ranges.

Finally, we use obtained piano rolls as an input to either GAN or VAE, which, once trained, are able to generate new melody piano rolls.

In the next sections we describe all the steps in more details.

IV. Dataset & Preprocessing

The dataset used for this project is the MAESTRO MIDI Dataset, which was obtained by recording the performances of many pianists with Yamaha Disklaviers at the International e-Piano Competition, for a total over 200 hours of paired audio and MIDI recordings.

The MIDI (Musical Instrument Digital Interface) format is a standard protocol for electronic music instruments. It is widely used in industry and in research because it stores information about musical notes and events, rather than actual audio. As a result, MIDI files have a relatively small file size and can be more efficiently processed compared to other music formats (e.g. MP3, MAV). [8]

The dataset was characterized by extracting the distributions of some key quantities, such as:

- the number of instruments playing was always one, and the only instrument playing was the acoustic grand piano;
- the tempo was always 120 bpm;
- the time signature was always $4/4$;
- no keys or lyrics were specified;
- there is a nice variety of lengths of the music pieces (ranging from 100 s to 2500 s);
- the resolution of the MIDI files was either 384 or 480 PPQN (Pulses Per Quarter Note).

The limitations of this dataset will be reflected in the music samples that will be generated by NN models trained on this dataset. Therefore, we expect the generative model to be flexible in terms of music piece length, but will incur in the following limitations:

- using music from only one instrument;
- using only one fixed instrument (acoustic grand piano);
- a fixed tempo (120 quarter notes per minute);
- a fixed time signature $(4/4)$
- an impossibility to specify the key;
- an impossibility to work with lyrics;
- a limited range of resolutions (384 or 480 PPQN).

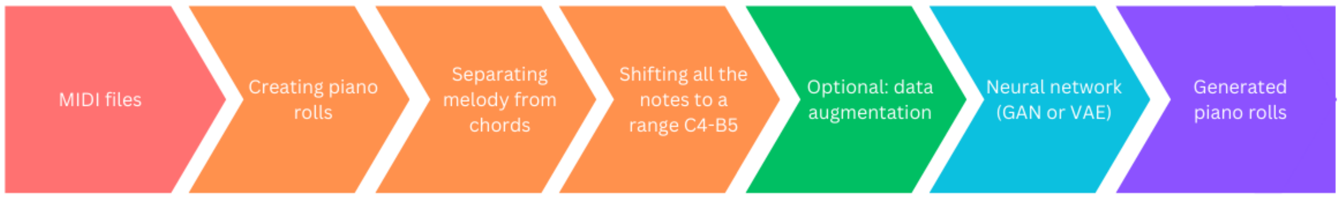
Before being passed to the NN models, each file in the dataset needs to go through a preprocessing algorithm, which can be broken down into the following steps.

A. Discretization

Firstly, the duration of each note is computed as the difference between the end and starting times of the note.

All the notes start times are adjusted by subtracting from them the start time of the first note, to remove the offset given by the difference between the start of the recording and the start of the actual music piece.

The adjusted start time and the duration of each notes are then rounded to discretize the temporal dimension of the note,



MIDI files
Creating piano rolls
Separating melody from chords
Shifting all the notes to a range C4-B5
Optional: data augmentation
Neural network (GAN or VAE)
Generated piano rolls

Fig. 1: Pipeline: original MIDI files go through the preprocessing part (orange), optional data augmentation, a neural network, resulting in the new generated piano rolls.

based on a sensitivity ε which was converted from bars to seconds by dividing it by the tempo:

$$1\ \text{AU} = \frac{1}{16}\ \text{bar} = \frac{\frac{1}{4}\ \text{QN}}{120\ \text{QN/min}} = \frac{1}{480}\ \text{min} = \frac{1}{8}\ \text{s}.$$

The new discretized time index k is computed as follows:

$$x = \begin{cases} k & \text{if} \quad \frac{k}{16} \leq x < \left(\frac{k}{16} + \frac{1}{32}\right) \\ k+1 & \text{if} \quad \left(\frac{k}{16} + \frac{1}{32}\right) \leq x \leq \frac{k+1}{16}. \end{cases}$$

The end time of the last note of the music piece is also adjusted for the initial offset and discretized: this step is necessary to compute the length of the piano roll, which is used to graphically represent the music events. The rows and columns of the piano roll identify, respectively, the pitch and the time (in AU) of a note. Since, following [2], we omitted volume, we used a binary piano roll with 128 rows, corresponding to all possible MIDI pitches.

For each file, an empty (i.e. zero-filled) piano roll is initialized and filled, note by note, with 1 in all the cells corresponding to a note played.

B. Separating melody and chords

The following step consists of identifying the chords present in the music. Since all chords are a combination of 3 notes, we inspected only columns (i.e. time indexes) with at least 3 notes playing, and we computed, for each of these column, all possible triplets. The triplets $(p_1, p_2.p_3)$ were then filtered based on two of the differences in pitch $d(p_1, p_2)$ and $d(p_2, p_3)$, which must be equal to either (3,4) or (4,3). Lastly, the remaining triplets were filtered and labelled based on a chords database.

The chords and melody of each file are separated by, firstly, building a piano roll with the previously identified chords: this chords piano roll is then subtracted from the initial piano roll, to obtain the melody piano roll (see fig. 2).

C. Encoding

The last step of preprocessing is reducing the pitch range of the melody piano roll from [A0, C8]=[21, 108] out of all possible values [0,128], to a fixed range [C4, B5]=[60, 83]. [2] The approximation is based on the octave o: if $o \leq 4$ the note is approximated with the corresponding note in the fourth octave. Otherwise, if $o \geq 5$, the note is approximated with the corresponding note in the fifth octave.

D. Features

We decided to focus on melody generation, particularly on the fixed range melody, and therefore discarded the chords from piano rolls. The dataset was split only in two parts: the majority was used for training (80%), in some instances together with data augmentation, and the remaining portion (20%) was used for testing the quality of the generated samples.

E. Data augmentation

Each NN model we trained on augmented and non-augmented dataset. When implemented, data augmentation was performed randomly, with a probability $p = 0.5$ of performing data augmentation on each piano roll.

This data augmentation technique exploits the fixed note range [C4, B5] = [60, 83] by shifting all notes in the piano roll by $[-3, -2, -1, +1, +2]$ octaves: these values were chosen to ensure that the notes stay in the pitch range allowed for this particular instrument [A0, C8] = [21, 108].

V. Learning Framework

In this section we provide a more in-detail description of the two models' architectures and discuss the training procedures. All the code was written using PyTorch library [11].

A. GAN architecture

For the CNN-based GAN architecture, we mainly followed [2] with some smaller modifications.

1) Generator: A scheme for the generator is presented on fig. 3. As an input, the generator takes a Gaussian noise vector of dimension 100 and the previous bar.

Firstly, the noise vector goes through the fully connected network with layers of 1024 and 512 neurons, both followed by batch normalization and LeakyReLU activation function, and a final linear layer, which output has dimension of 512. We denote the result of the noise vector going through this part of the network as x.

Secondly, we pass the previous bar through four 2D convolutional layers with 256 filters each and kernel sizes $k_1 = (128, 1), k_2 = k_3 = k_4 = (1, 2)$ and strides $s_1 = 1, s_2 = s_3 = s_4 = 2$. We save the result after each convolution and denote them with c_1, c_2, c_3, c_4.

Then we move to the four 2D transposed convolutional layers. Three first layers have input dimension 512, output

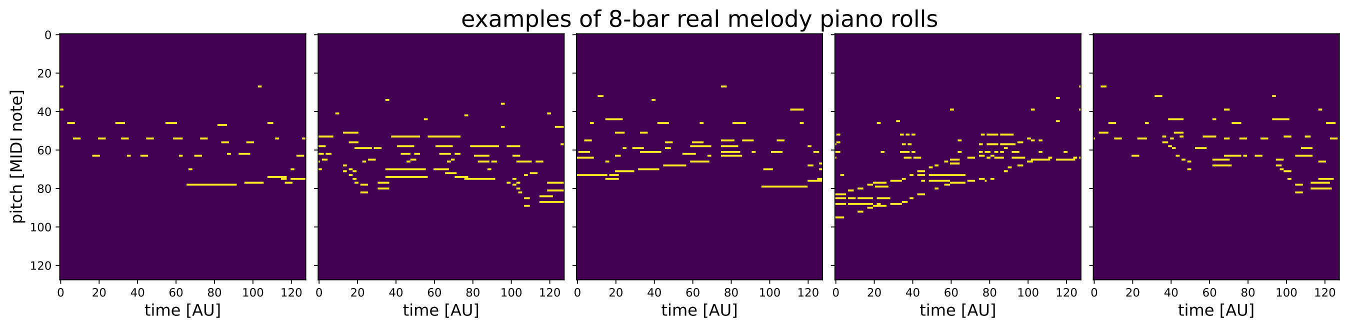


Fig. 2: Examples of real piano rolls.

dimension 256, kernel size $(1,2)$ and stride 2. Each of these three layers are followed by batch normalization and LeakyReLU activation function. The fourth layer has an input dimension of 512, kernels size of $(128,1)$ and stride 1. The input to the first layer is a concatenation of x and c_4, resulting in the output y_1, which is then concatenated with c_3 and passed to the next layer. We continue the same procedure further. In this fashion we implement the conditioning on the previous bar, we continue to input a concatenation of the previous output and the result of the convolution to the subsequent layers. Then the final output is passed to the hyperbolic tangent activation function. We tried to use sigmoid function as well, but found that with hyperbolic tangent the values of the generated piano rolls (before converting it to the binary one) are bigger and thus it promotes the generator to be more "sure" in its generations.

2) Discriminator: Like generator, discriminator in this GAN is CNN-based. It has two 2D convolutional layers (with output channels equal to 14 and 77, kernel sizes $k_1 = (128,2)$ and $k_2 = (1,4)$, strides $s_1 = s_2 = (2,2)$), followed by batch normalization and LeakyReLU activation function. Then the result is passed through a linear layer of size 1024, batch normalization and LeakyReLU. The final layer consists of a linear layer with one output feature, followed by sigmoid function. This layer was added compared to what was described in [2] to allow the discriminator to output a single probability value.

As an output, we saved the result after going through all the layers and the first convolutional layer specifically, as this value is used in the loss function.

B. VAE architecture

Besides the GAN described above, we also decided to try a CNN-based VAE architecture. As a standard VAE, it consists of encoder and decoder parts, but also has a small additional network for implementing the condition on the previous bar. The dimension of the latent space was set to 256.

The encoder consists of four 2D convolutional layers with output channels equal to $512, 256, 256, 128$, kernel sizes $k_1 = (128,1), k_2 = k_3 = k_4 = (1,2)$ and strides $s_1 = 1, s_2 = s_3 = s_4 = 2$. Each 2D convolutional layer is followed by batch normalization and LeakyReLU activation function.

The first part of the decoder consists of one linear layer with 256 neurons, batch normalization and LeakyReLU. The second part is formed by two 2D transposed convolutional layers (with output channels equal to $256, 512$, kernel sizes $k_1 = k_2 = (1,2)$, strides $s_1 = s_2 = 2$), followed by batch normalization and LeakyReLU. After this, we have a final 2D transposed convolutional layer with $k_3 = (128,10), s_3 = 2$ and a sigmoid function. Unlike GAN, for this architecture we found that with the sigmoid function we obtain better results.

To conclude the description of the CNN-based VAE, we need to comment on how we implemented the condition on the previous bar during training. Inputs of our VAE are a real current bar and a real previous bar that we take from the dataset. The real bar goes through encoder, then the latent vector is reparametrized. The previous bar goes through a specific condition encoder (linear layer with output dimension 112, batch normalization, LeakyReLU). The result is concatenated to the latent variable and passed to the first part of the decoder. In this way we implement conditioning on the previous bar. During generating new piano rolls, we used one real bar as the first bar and then in a similar fashion perform conditioning on it.

C. Training

In this section we describe in details the training procedure. In both cases we used a helping class for loading the dataset. In particular, its methods load the piano rolls (obtained after the preprocessing part), add a zero bar in the beginning of the piano roll, then select all the pairs of two subsequent bars found in a given piano roll; if chosen, performs data augmentation as described above for the given pair of bars. At the training loop, we give one pair of subsequent bars at a time. Besides that, for both models we used Glorot initialization of weights. Below we provide details for training two models that are different.

1) GAN: Besides classical loss for GANs, as stated in [2], for addressing instability and and model-collapse issues, there added two terms to the generator loss:

$$\lambda_1 \left\|\mathbb{E}[X] - \mathbb{E}[G(z)]\right\|_2^2 + \lambda_2 \left\|\mathbb{E}[f(X)] - \mathbb{E}[f(G(z))]\right\|_2^2$$

where f denotes the first convolutional layer of the discriminator, and $\lambda_1 = 0.1, \lambda_2 = 1$ are set based on [2].

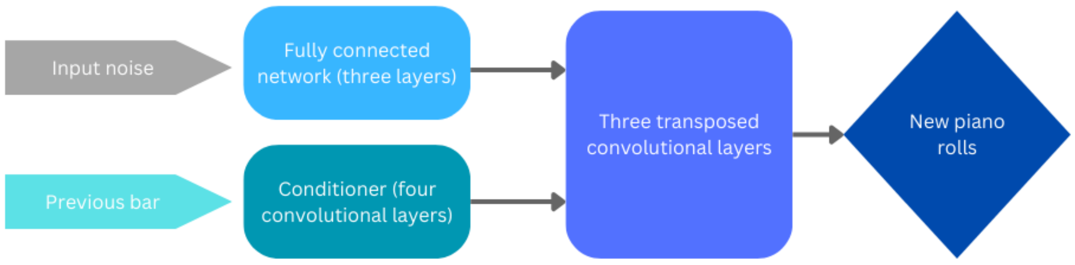

Input noise
Previous bar
Fully connected network (three layers)
Conditioner (four convolutional layers)
Three transposed convolutional layers
New piano rolls

Fig. 3: Scheme for the generator architecture in GAN.

For both discriminator and generator we use Adam optimizer with learning rate equal to 0.0002 and $\beta_1 = 0.5, \beta_2 = 0.999$. Besides that, we added one-side label smoothing, resulting in value 0.9 for the real labels instead of 1. In each epoch we updated the generator twice and the discriminator once to prohibit the situation when the discriminator outpowers the generator. Since we have limited computing resources, we trained the GAN for just 10 epochs, which for the augmented dataset took a bit more than an hour on the Google Colab GPUs. For resulting training losses, see fig. 4.

2) VAE: We took a classical loss function for VAE, which consists of a reconstruction term (binary cross-entropy between the reconstructed output and the input data) and Kullback-Lieber divergence between a learned latent variable distribution and a standard Gaussian distribution. At each epoch, we add Gaussian noise to a real current bar to promote exploration of the latent space. In the case of VAE we use Adam optimizer as well, with learning rate parameter set to 0.0001. We perform the training for 30 epochs. For resulting training losses, see fig. 4.

D. Generation

After training the models, we save them and the obtained losses for the further use. Similar to [2], we implemented a function to generate particularly eight bars of music, from which the first bar is provided from the dataset. This function generates one bar at a time, given the previous bar, both for GAN and VAE. The generated piano rolls are converted into binary piano rolls to obtain the final result.

E. Evaluation metrics

Since the NN algorithms used are unsupervised, standard accuracy-based metrics could not be implemented. We compare the samples generated by the two models, firstly qualitatively and later quantitatively, using specific approaches.

The first step of the qualitative evaluation consisted in the visual comparison of piano rolls from real data (in fig. 2), and of piano rolls produced by GAN and VAE, both with and without data augmentation.

Another qualitative comparison involved the distributions of the note pitch, which were computed for real data and for samples generated by the GAN and the VAE, both with and without data augmentation.

Lastly, a quantitative approach was used: out of all possible GAN evaluation metrics, we chose to implement the Fréchet Inception Distance (FID), which evaluates the quality of artificially generated data by comparing it to the distributions of real data and artificially generated data. More technically, the distributions are fed to a pre-trained network (called Inception v3, which input data must be 299 by 299, and outputs 2048 dimensional feature vectors). The mean and covariance of the output features of this network are then extracted. Lastly, the FID is computed as the distance between the two multivariate Gaussian distributions that are characterized by the mean and covariance of the feature vectors extracted from real and artificially generated data.

VI. Results

Firstly, we took a look at the trends of the loss over the epochs, for both GAN and VAE, with and without data augmentation. We noticed that a decreasing trend emerges in all of the plots, particularly in the VAE's plot. But, in the GAN's plots, this decreasing trend is less visible due to the limited number of epochs.

Then, we qualitatively compared some examples of piano rolls generated by GAN and VAE, with and without data augmentation (see fig.5 and fig.6), to some real melody piano rolls (see fig.2).

The GAN melody piano rolls, especially the non-augmented ones, resemble the real and training melody piano rolls. At the same, the GAN piano rolls also show distortions, mainly in the form of abnormally long note duration. (i.e. horizontal lines).

On the other hand, the VAE melody piano rolls show no resemblance to the real and training melody piano rolls. Instead, the generator learned a pattern mainly consisting of vertical lines, which in real life, corresponds with playing all MIDI notes at the same time.

Similarly to above, in fig.7 we qualitatively compared the distributions of note pitch generated by GAN and VAE, with and without data augmentation, to the distributions of note pitch obtained from real data, with and without fixed range.

We noticed that the real note distribution is almost uniform, and covers a majority of the notes that can be played on a real-life piano. Therefore, potential future biases in music generation can not be directly blamed on the original data.

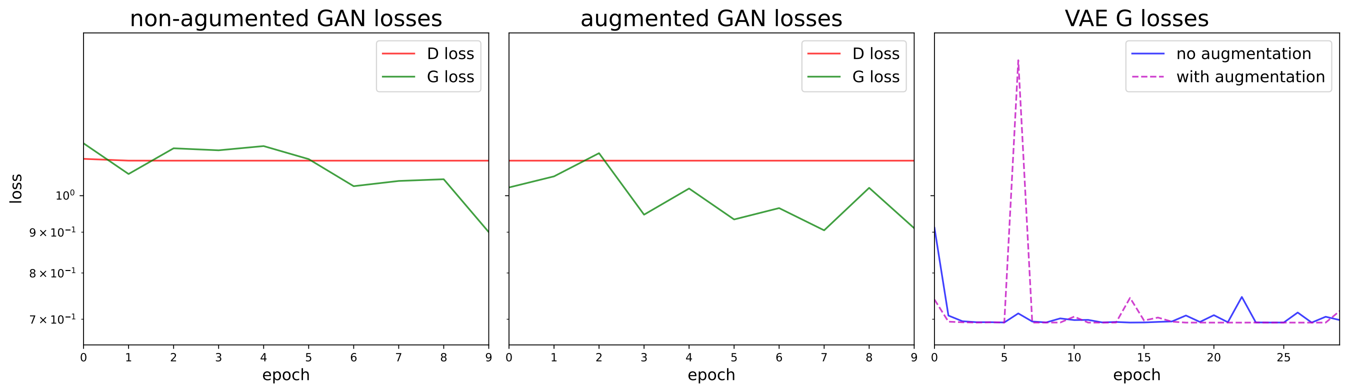


Fig. 4: Training losses obtained for all the architectures.

The note distribution changes significantly after switching to fixed range melody piano rolls. For non-augmented data, the distribution is limited to the fixed range. While, for augmented data, it spans most of the piano range. Despite their differences, these distributions share a similar asymmetric shape and differ only for the tails, which values can only be obtained through data augmentation.

Both GAN distributions, with and without augmentation, almost correctly show a resemblance to the training distributions. More specifically, the majority of the notes fall within the fixed range. However, we see that model without data augmentation incorrectly explores the note range, though it should produce notes only in a given fixed range in order to resemble the training data. We need to mention here that this issue can arise from the fact that GAN was trained only for 10 epochs.

For VAE, we observe very similar distributions between the augmented and non-augmented implementations. The non-augmented distribution is unrealistically uniform across the whole MIDI range, which corresponds to playing all MIDI notes at the same time. Therefore, it seems like the algorithms has an idea of when to play notes, but doesn't know how to choose between the notes. This trend was shown above qualitatively as well.

Lastly, the FIDs were computed based on 1000 samples: the results are shown in 1. Unlike before, only the fixed range version of the real data was examined.

generative model	FID metric
GAN, no augmentation	2.4
GAN, with augmentation	6.1
VAE, no augmentation	168.1
VAE, with augmentation	255.2

TABLE 1: FID values obtained by comparing fixed range piano rolls with samples generated by the models above.

VII. Concluding Remarks

In this project we implemented two CNN-based generative models: GAN and VAE. The piano music dataset, which was used in this work, was carefully preprocessed before being passed to these two models.

To conclude, we want to compare performance of GAN and VAE based on the generated piano rolls qualitatively, on the pitch distributions and on the FID values. In all three cases we found that GAN seems to be better at generating realistic music: the FID metric supports these conclusions.

We noticed that GAN, despite being trained on fewer epochs, generated bigger values for the piano rolls (before being converted to binary piano rolls) than VAE. Therefore, a possible explanation for VAE's poor performance can be that VAE was undertrained and/or not optimized enough to successfully solve the music generation task. This hypothesis is supported by the presence of vertical lines in the VAE piano rolls, which mean that the model managed to learn when to play notes, but did not understand how to choose which note to play.

In general, the VAE's poor results could follow from considering that the successes of CNN architectures in music generation does not guarantee a good performance of CNN-based VAE (or, at least, of this particular architecture). Further research can be done on optimization of the VAE and exploration of other CNN-based architectures for VAE.

Regarding data augmentation, which typically improve the performance of the architectures, we found, qualitatively, that the melody piano rolls generated with it looked more realistic. At the same time, quantitatively, the values of FID was higher for the models trained on the augmented dataset. Therefore, we can not report any significant improvement when using data augmentation. This is likely justified by the limited training time available with the computing resources we had.

Lastly, we want to mention limitations of this work. Firstly, some of these come from the chosen dataset: we focused on generating piano rolls just for one instrument, with fixed time signature, fixed tempo and no lyrics. Second part of the limitations consists of more technical issues: since training time of GAN and VAE was long enough even for limited number of epochs, we could not train them further with limited computing resources that we had.

References

[1] S. Ji, X. Yang, and J. Luo, "A survey on deep learning for symbolic music generation: Representations, algorithms, evaluations, and challenges," *ACM Comput. Surv.*, vol. 56, Aug. 2023.

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

[3] H. Zhu, Q. Liu, N. J. Yuan, C. Qin, J. Li, K. Zhang, G. Zhou, F. Wei, Y. Xu, and E. Chen, "Xiaoice band: A melody and arrangement generation framework for pop music," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2837–2846, 2018.

[4] S. H. Hakimi, N. Bhonker, and R. El-Yaniv, "Bebopnet: Deep neural models for personalized jazz improvisations.," in *ISMIR*, pp. 828–836, 2020.

[5] Y.-W. Chen, H.-S. Lee, Y.-H. Chen, and H.-M. Wang, "Surprisenet: Melody harmonization conditioning on user-controlled surprise contours," *arXiv preprint arXiv:2108.00378*, 2021.

[6] K. Chen, C.-i. Wang, T. Berg-Kirkpatrick, and S. Dubnov, "Music sketchnet: Controllable music generation via factorized representations of pitch and rhythm," *arXiv preprint arXiv:2008.01291*, 2020.

[7] A. Pati, A. Lerch, and G. Hadjeres, "Learning to traverse latent spaces for musical score inpainting," *arXiv preprint arXiv:1907.01164*, 2019.

[8] A. Dash and K. Agres, "Ai-based affective music generation systems: A review of methods and challenges," *ACM Comput. Surv.*, vol. 56, no. 11, 2024.

[9] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, vol. 12, 2016.

[10] C.-Z. A. Huang, T. Cooijmans, A. Roberts, A. Courville, and D. Eck, "Counterpoint by convolution," *arXiv preprint arXiv:1903.07227*, 2019.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

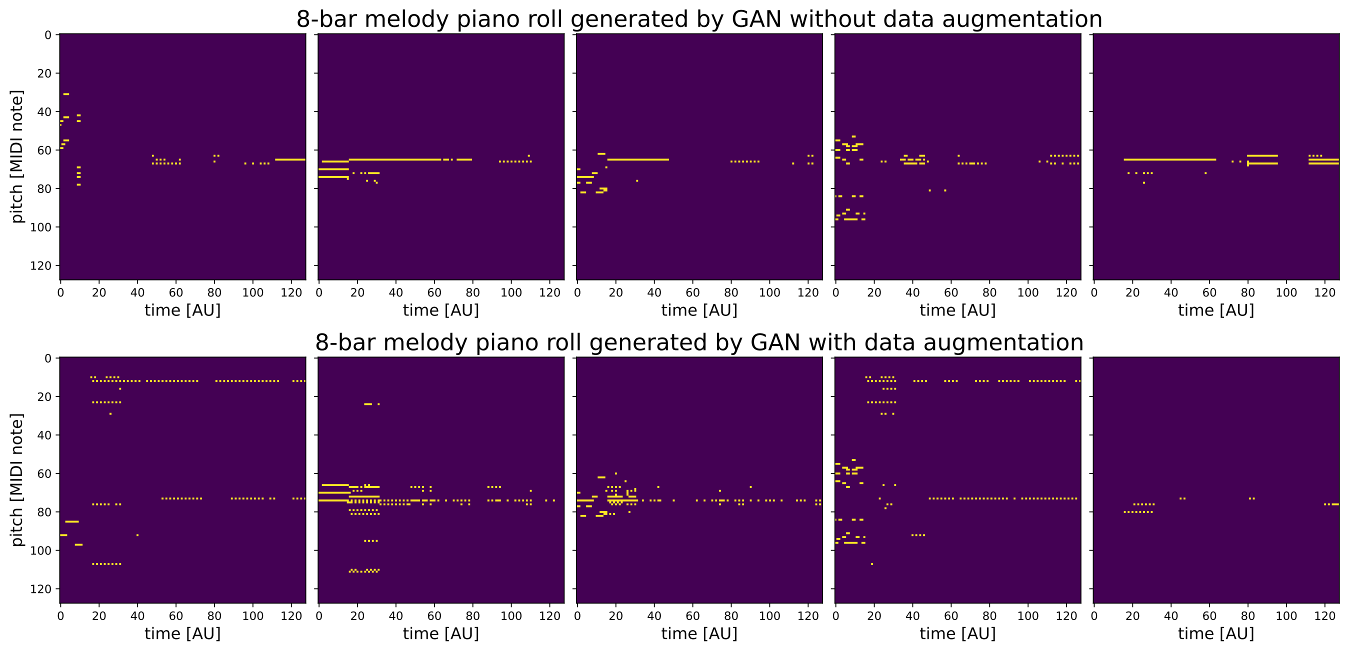


Fig. 5: Examples of samples generated by GAN, trained on both augmented and non-augmented data.

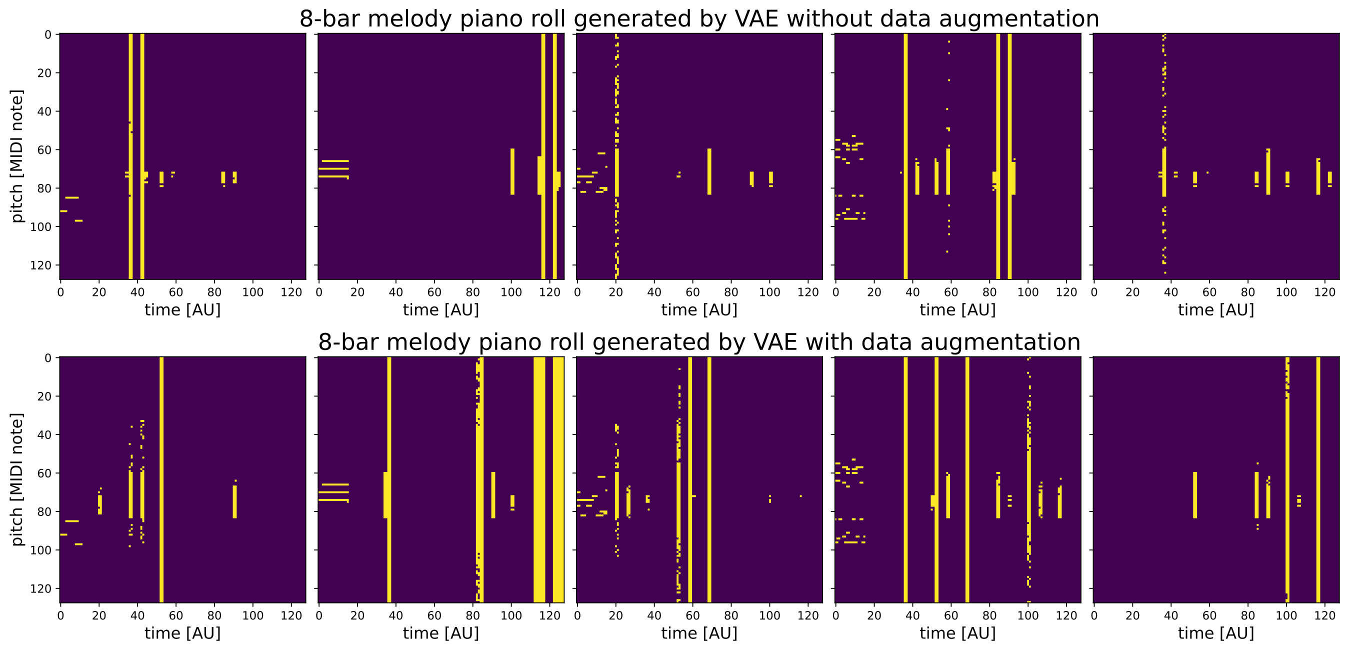


Fig. 6: Examples of samples generated by VAE, trained on both augmented and non-augmented data.

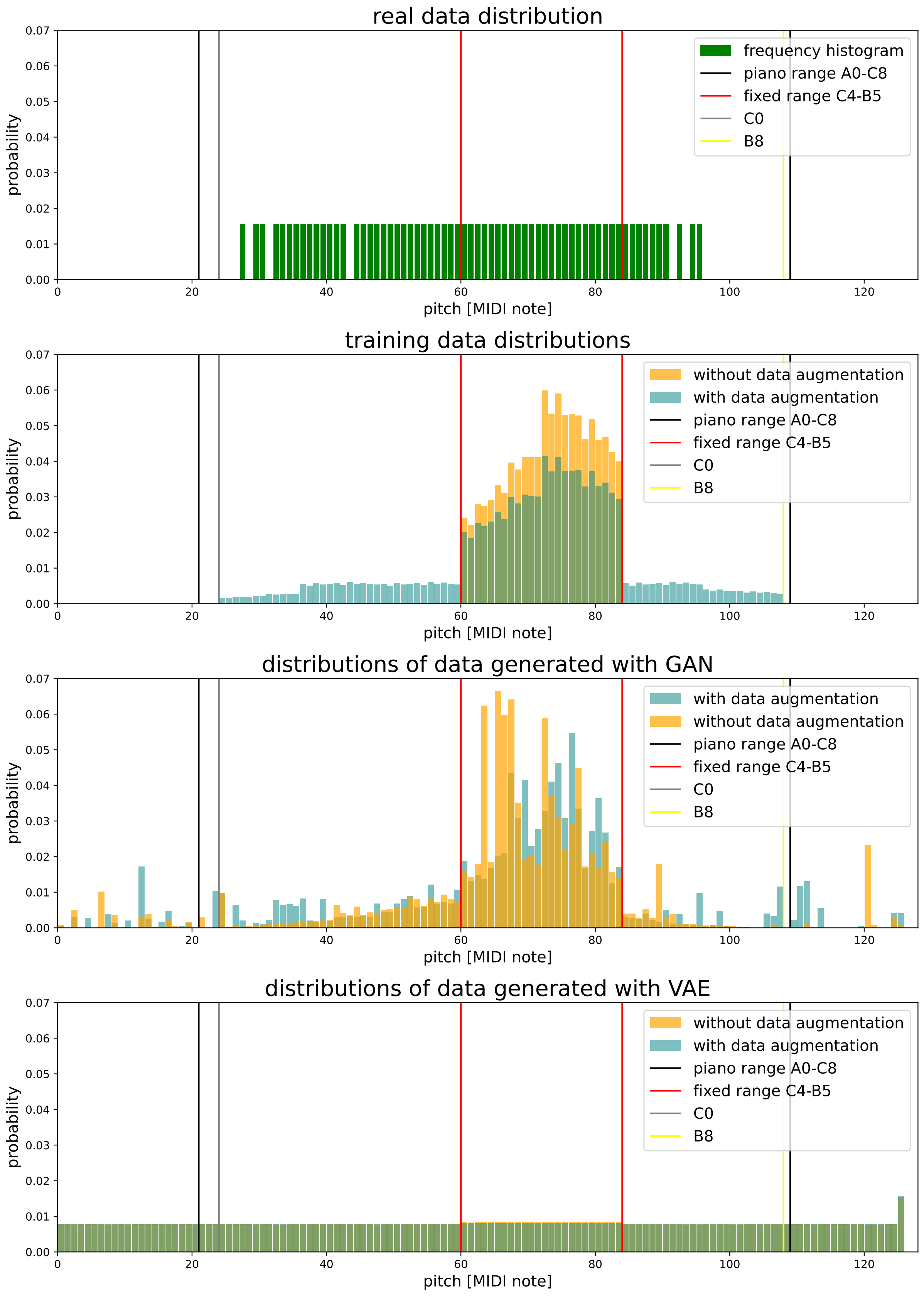


Fig. 7: Pitch distributions of the samples used at the different stages of the project