Neural Music Style Transfer

Antoine Miquel
CentraleSupélec
antoine.miquel@student.ecp.fr

Luca Serra
CentraleSupélec
luca.serra@student.ecp.fr

Erwan de Lépinau
CentraleSupélec
erwan.de-lepinau@student.ecp.fr

March 1st, 2021

Abstract

Neural style transfer consists in applying the style of an entity to another one while keeping the content of the latter, using Deep Neural Networks (DNN). While the amount of work on style transfer has been significant in the image processing community, very few papers have been produced in the musical domain. The topic seems all the more interesting as there could be numerous musical adaptations of famous songs in a different style: one might be interested in hearing how a Jazz interpretation of a Mozart masterpiece could sound.

1 Introduction

In this work, we are interested in the task of music style transfer defined as keeping the content of a song and applying a different style to it. We define *content* as the melody and harmony of the piece, and *style* as the genre the piece belongs to. The concept of musical genre itself is also somewhat blurry, and given that we will be focusing on piano songs and not trying to change the instrument in this work, we consider genre to be mainly defined by rhythm, density of notes and complexity of the chords played.

Given the challenge that musical style transfer represents, we will first focus on two genres: Classical and Jazz. The reason for this choice is that these two musical genres are the most studied ones in the works on music generation and style transfer we found ([1] [2] [3]) and thus existing datasets are of good quality and easy to find. Additionally, before trying to immediately tackle the harder task of style transfer, we chose to first focus on generating samples in one genre or the other from random noise samples, in line with generative approaches such as Generative Adversarial Networks [4]. The reasoning behind this gradual approach was to first get familiar with effective representations of music data and generative model architectures before jumping to more complex, composite architectures used in neural style transfer.

2 State of the art

2.1 Generative models

The broader task of generating new, artificially synthesized content without human intervention (may it be image, video, text, sound...) has been a hot topic in the deep learning community for several years now. One of the first major breakthroughs in this domain was the invention of Generative Adversarial Networks (or GANs) [4] in 2014. In the original paper, the task at hand is the generation of new images (handwritten digits, human faces, animals...) without simply copying previously seen examples from the training set. The proposed architecture consists of two deep neural networks, a generator G and a discriminator D, trained simultaneously and in an adversarial manner: the generator G tries to generate a realistic-looking image

(for example, a human face) from a random noise input, and is rewarded (*i.e.* its loss decreases) if the discriminator identifies this generated image as a real-world image; the discriminator D, on the other hand, receives as input both real-world images and fake images generated by G, and has to correctly label each image as either coming from a real-world dataset or from G.

However, GANs do suffer from a certain number of common issues [5], including vanishing gradients in the generator (usually because the discriminator improves much faster than the generator), *mode collapse* where the generator learns to always output the same image (or whatever the output is) regardless of the inputs, or failure to converge in the generator. Progress has been made in the following years to try and tackle this problem, usually by modifying the training process and generator loss function. Notable examples are the WGAN [6] or Unrolled-GAN [7] architectures.

Another popular family of generative deep learning models are Variational Auto-Encoders (or VAE) [10], derived from the Auto-Encoders family of neural network models. Auto-encoders in their simplest form are traditional feedforward fully connected neural nets, with N input neurons (N being the number of features of the raw data samples), less than N neurons in the hidden layers, usually monotonically decreasing down to $K < N$ in the innermost hidden layer, and then increasing back up to N neurons in the output layer. Auto-encoders are trained so that they recreate as closely as possible the input values as their output: this results in an interesting technique to carry out unsupervised feature selection and dimensionality reduction, as the innermost K-dimensional hidden layer can be used as a lower-dimension representation of the input, in what is called the *latent space* of the model.

From there, one could try and use the auto-encoder as a generative model by randomly sampling a point in the latent space, and feeding it as input to the decoder half of the model (using the K-dimensional hidden layer as the input layer). The main issue with this approach is that since we have no information about the distribution of the training samples in the latent space (the latent space is said to lack *regularity*), randomly sampling a point in that space provides no guarantee that this point would produce a realistic output once decoded. This is what VAEs attempt to solve, by forcing the distribution of the training samples in the latent space to match a $\mathcal{N}(0, 1)$ distribution, which leads to better regularity of the latent space. Then, randomly sampling a point in that space following the $\mathcal{N}(0, 1)$ distribution, and decoding it with the second half of the auto-encoder allows to generate novel, previously unseen data examples. VAE-based models have seen numerous successful applications in the recent years, for instance on image generation [11] or text generation [12] tasks.

As mentioned previously, the main application domain for generative models in the literature so far has been images and pictures. Although in the recent years, automatic music generation using GANs has started gaining interest amongst the deep learning community [8] [9], the amount of published work on the topic remains relatively low compared to the images application field. Possible explanations for this relative absence of applications to music may be the lack of real-world industrial use-cases, or the inherent complexity and amount of pre-processing required to effectively deal with musical content (musical data can rarely be used as-is, and usually requires transcription to MIDI format, multi-track separation, etc.).

2.2 Neural style transfer with paintings

Neural style transfer, first introduced by Gatys et al. in [13] in 2015, is defined in the context of artistic painting as the blending of a *content* image and a *style* image. The *content* is what constitutes the high level description of the image (ex: "Buildings along a river"), and the *style* is a combination of the colors, textures and painting techniques specific to a certain artist or piece of work.

Figure 1: Neural style transfer applied to painting (source: [13])

The architecture used in the paper is based on a pre-trained image classification deep convolutional model (VGG in the original paper), in which the last layers (responsible for classification) are ignored, instead focusing on the intermediate layers. The paper showed that the content of an image could be represented by extracting the feature maps from a well-chosen intermediate layer, and the style could be represented by computing Gram matrices in several intermediate layers. From there, a content loss and a style loss can be defined, which indicate by how much the content (resp. the style) of an input image differs from the content (resp. the content) of the source content (resp. style) image. The model is then used as follows: a source content image (ex: Buildings along a river) and a source style image (ex: Van Gogh's "Starry Night") are selected by the user; the content image is used as the starting input for the model, and its content and style losses are computed by forward propagation in the network; the pixels of the input image are then modified to minimize the content and style losses through backpropagation; this process is repeated until convergence or until the user stops it. Figure 1 shows some example results of this process.

2.3 Towards neural music style transfer

Neural style transfer in the context of music has by far not been studied as much as in the context of visual arts, and there is no - to the best of our knowledge - unique definition of the task that achieves consensus amongst the research community, as pointed out by Dai et al. in [14]. Indeed, although we are tempted to draw a parallel between imagery and music in that they are both artistic content domains, fundamental differences remain in their characteristics and the ways we can represent them in a manner suitable for deep learning tasks, rendering an analogy between the two in the case of neural style transfer even harder. Also according to [14], attempts to reuse architectures similar to the one used in the original painting style transfer paper [13] have proven to produce underwhelming results when applied to music.

As stated in the introduction of the present work, we chose to define content as the key, general melody and harmony of a musical piece, and style as the genre (Classical or Jazz) and interpretation subtleties (including velocity) of a piece. This somewhat resembles what is defined as *Composition style transfer* in [14].

A first approach to this task using Variational Auto-Encoders is described in [15], and involves two VAEs (one for each musical genre) sharing the same latent space, where style transfer from a genre A to a genre B is performed by feeding a musical piece of genre A to the encoder of VAE A, and decoding it through the decoder of VAE B. Another approach from the same author, described in [2] uses a CycleGAN with two musical genres A and B: two GANs are arranged in a cyclic pattern, where generator A-to-B takes an input from genre A and transforms it in genre B; discriminator B is then fed the output from generator A-to-B and tries to determine if it's a real data sample or a fake generated one; then generator B-to-A takes the output from generator A-to-B and transforms it back in genre A; discriminator A is fed the output from generator B-to-A and tries to determine if it's real or fake; finally, the output from generator B-to-A is compared with the original input to ensure it did not change too much. A Seq2Seq approach using recurrent networks (LSTM) with attention mechanism, inspired from state-of-the-art approaches in neural machine translation tasks, is described in [1] for the task of interpreting sheet music in a similar way a professional performer would. This approach is reused with slight modifications in [3] for the task of musical style transfer, but learning a different LSTM model for each musical genre.

3 Experiments

Given the difficulty of music genre transfer task, we first started to build a Jazz and Classical generator. Several works have used Convolutional Neural Network (CNN) in a Generative Adversarial Network (GAN) architecture [2] [3], which seems easier to implement than with Recurrent Neural Networks (RNN) GAN [16].

3.1 Data and preprocessing

Sequential data can be represented as a grid if we set the length of data used. The idea here is that we want to transpose a visual representation of a Musical Instrument Digital Interface (MIDI) file as training data, so CNN structure seems relevant for this task. Thus, the training samples have one dimension corresponding to the number of notes

which can be played (pitch) and the other corresponding to the subsequent time steps. For the time dimension, a reasonable choice about the length is to choose one phrase, which is long enough to grasp the content of a song. A phrase is composed of 4 bars and a bar is composed of 4 times. Given the granularity of a midi file, which is 1/4 of time, there are 64 time steps in the training data. Concerning the pitch dimension, we choose to limit the range of notes between C1 and C8, because the notes bellow C1 or above C8 are not very common in songs [2], which corresponds to a range of 84 notes (even if the range of notes playable in MIDI format is 128). (See figure 2.) The possible values (called "velocity", which corresponds to the loudness of the note played) for the terms of the 64 x 84 matrix are between 0 and 127 in the MIDI format (can be normalized between 0 and 1). A first approach can be to consider that these values are binary: either the note is played (equal to 1) or not (equal to 0). This simplifies the task by changing this regression problem into classification problem for each term. Another drawback of this simple modeling is that it does not represent the information regarding the way the note is played, i.e. if the note is played once more or sustained. This drawback can be solved by using a finer representation, which differentiates a note articulated or sustained, for example using a 2D-vector (see [1]) or several values (e.g. -1, 0 and 1).

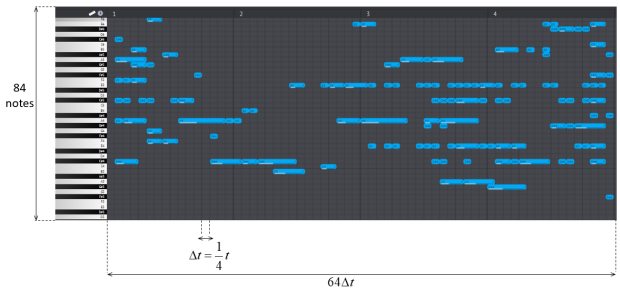

Figure 2: MIDI file (input)

3.2 Generating Jazz and Classical music

3.2.1 First implementations

The first GAN model which was implemented to generate music was based on a Multi-Layer Perceptron (MLP) architecture. Using this simple architecture was a good way to rapidly validate the feasibility of this project and to assess the necessary training time to have satisfying results. Given the 64 x 84 matrix as input, which is called a piano roll, the preprocessing consisted in flattening the matrix in order to have a (1, 64 x 84) vector, to feed the discriminator. By a try-and-error method, we find out the best architecture for our MLP-GAN. Generator and discriminator have respectively 2 and 3 hidden layers, with a Leaky-RELU activation function, which helps the gradients flow easier through the architecture [17]. We use dropout for the discriminator to prevent over-fitting. (For more information about the models architecture, please see our code [18].) With a training set of 1000 samples and a batch size of 32, a Stochastic Gradient Descent Optimizer (learning rate of 10^{-3}), the GAN go through ≈ 1 epoch / sec. The number of epochs necessary to have satisfying results (arbitrary criteria, by hearing the generated MIDI file) is around several hundreds. The chosen generated matrix is then processed in order to round the values: we choose a precision ε and then we apply to each term the following function: $f : x \mapsto 0$ if $x < \varepsilon$ else 1. In this project, we chose $\varepsilon = 10^{-1}$ (see figure 4: data close to 0 is inferior to ε). The results can be heard here for Jazz and here for Classical. Even if the results are encouraging (almost no false note, the style of the song is easily recognizable, the visual aspect of the MIDI file seems like a real song, as it can be seen in figure 3), we experienced different failure modes, which are common issues when training GANs [5]. This topic will be tackled more in depth in the next subsection.

As mentioned above, the format of the data is relevant for the use of CNN, thus, it was the second type of GAN implemented. This setup is more often named Deep Convolutional Generative Adversarial Networks (DCGAN). The number of layers chosen is quite similar to the architecture of the MLP-GAN, and it was inspired by previous work, on image generation [19], even if the dimensions of the kernels and the number of layers is not the same. The kernel size was chosen in line with the dimensions of our

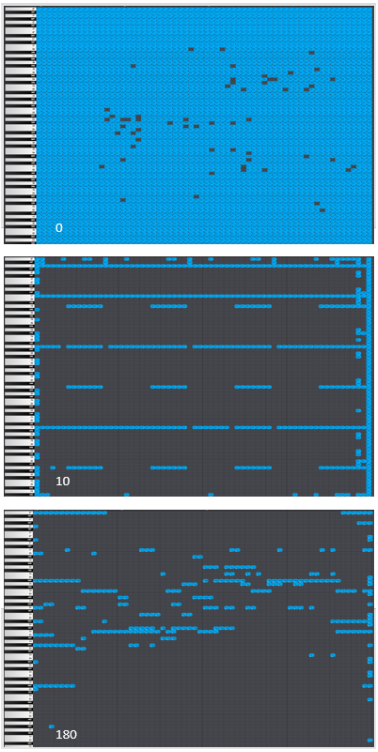

Figure 3: Generated MIDI file evolution (at epoch 0, 10 and 180) for the DCGAN

input matrix: given the fact it was a music generation task, we found intuitive and relevant to have a kernel width of 12, which corresponds to the number of notes in an octave. The same dimension was used for its height, which corresponds to three musical times. The DCGAN proved to be more consistent than the MLP-GAN in its training phase (meaning the quality of generated music was not varying too much from one epoch to another). Moreover, the generated songs seemed more complex than the ones generated with the MLP-GAN (wider range of notes played, actual chords are played and the visual aspect of the generated MIDI file is closer from the training samples) as it can be heard here (Classical genre). Thus, we decided to keep working with DCGAN. However, even though these results are convincing and seem close to what a human player could produce, the inconsistency of the results (failure to converge) and the lack of diversity in the generated samples (mode collapse) lead to some researches of ideas to tackle these issues.

3.2.2 Training monitoring and stability

In order to monitor the stability of our GANs during the training phase, we implemented several metrics (some of them are general in GAN training and some are really specific to this task). A good way to insure that the training was improving was to print the losses of the discriminator on real and fake data and the loss of the generator. Moreover, to assess the realism of generated data, we also decided to plot the data distribution as well as the average number of notes played simultaneously for both discriminator and generator (every few epochs), as shown in figures 4 and 5. We also display the min and max value, which get closer to the extreme values (0 and 1 respectively) as the training evolves.

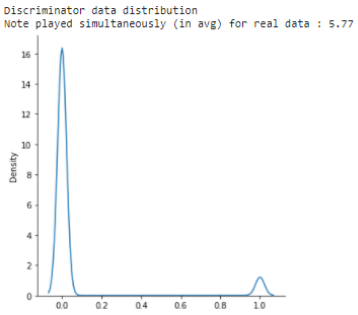

Figure 4: Discriminator input real data distribution

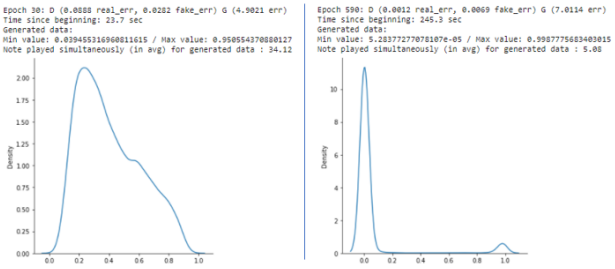

Figure 5: Generator data distribution at epoch 30 (left) and 590 (right) for the MLP-GAN

Surprisingly enough, there is not a very strong correla-

tion between the loss of the generator and the quality of the generated data, as the audio results shew. This is why we would use all theses metrics combined together to infer which epoch would give a satisfying generated sample. Moreover, we found out that the generated data were practically the same (no matter the random seed used by the generator). This issue is know as mode collapse and several techniques allow to tackle this.

The first one which was tried is minibacth discrimination, presented in [20]. When mode collapses, all samples created look similar. Thus, we feed real samples and generated samples into the discriminator separately in different batches and compute the similarity of the sample x with samples in the same batch. We append the similarity in one of the dense layers in the discriminator to classify whether this sample is real or generated. This technique did not actually helped our DCGAN to avoid mode collapse because our discriminator was already surpassing the generator, leading to even more unbalanced interaction. That is why we looked for a method which was more focused toward "helping" the generator.

The second technique is called Unrolled GAN and is detailed in [7]. Unrolled GAN plays k steps to learn how the discriminator may optimize itself for the specific generator. In general, $k \in [5, 10]$. At each step, we apply the gradient descent to optimize a new model for the discriminator but we only use the first step to update the discriminator. The unrolling is used by the generator to predict moves and thus it is not used in the discriminator optimization. It is as if the generator was able to see the future state of the discriminator and anticipate how it should behave to trick it. In order to measure the efficiency of this method, we store every few epochs the generated samples based on a seed vector which is a constant defined at the very beginning of the training. This constant seed vector was not used for the training of the generator nor the discriminator, but only to assess the diversity of generated data. Moreover, we print the mean Manhattan distance between the generated samples:

$$\frac{1}{\frac{n(n-1)}{2}} \sum_{1 \leq i,j \leq n, i \neq j} L_1(G_i, G_j)$$

, where G_i is the generated sample based on the element i of the seed vector, of length n (in our case, $n = 10$) and L_1 is the $L1$-norm. This piece of information enabled us to make sure the generated data were diverse with a different random noise.

The unrolled GAN method proved to be quite efficient when seeing the mean distance between the generated data and also double checking by exporting a few samples and observing the variety of the results.

3.2.3 Second iteration: improvements

Even though the results were satisfying, stable and diverse, we thought that the realism of the generated music could be improved. Indeed, the issue was that the notes were always attacked (the program could not express if the note should be played again or sustained, in the case of a same note activated in several consecutive time steps), as it can be heard in the last DCGAN sample. The solution to this was to make a distinction between a note which is hold and a note which is played again. For this, we tried several approaches.

First, we tried 2-dimensional approaches, keeping a one-channel input. The first method was to perform one-hot encoding: [1 0 0] means a new note is played, [0 1 0] means the note is hold and [0 0 1] means the note is not played. Of course, [0 0 1] cannot be followed by [0 1 0] (and if it is the case in the generated sample, we just ignore it). With this representation, we obtain a $(64, 84 \times 3)$ matrix. We found that using a $(64, 84 \times 2)$ matrix gave better results (more realistic songs) (with [1 0] for attacked, [0 1] for sustained and [0 0] for not played). We also tried to use $(64, 84)$ matrix with different values for each state: 1 if the note is attacked, 0 if it is sustained, -1 if it is not played. The $tanh$ activation function was well suited for this modeling of the problem and gave some satisfying results. As it can be heard here, the song seems more natural and the transitions are smoother than with the models of the first iteration because of this use of sustained notes.

Then, we implemented a 3-dimensional method, with a $(64, 84, 2)$ input matrix and thus using 3D convolutions. The reason that we tried this is that intuitively, the relation between the terms $M_{i,j,0}$ and $M_{i,j,1}$ of the data used (the way the note is played or not) is not the same as the relation between the terms $M_{i,j,k}$ and $M_{i,j+1,k}$ or $M_{i,j,k}$ and $M_{i+1,j,k}$ (the next note in the scale or the next note played in time). Here is a result of a generated Jazz song using 3D convolutions. One may notice the play with silences and longer notes, which gives a really realistic aspect to

it. For the rest of the project, we selected this structure as our baseline model.

3.3 Style Transfer

3.3.1 Global Architecture

A CycleGAN is the combination of two generator/discriminator pairs. In our example of transferring Jazz to Classical (and vice versa), we have a generator that takes Jazz as input and returns Classical as output (J2C), a discriminator that classifies Classical samples as real or fake, another generator that takes Classical as input and returns Jazz as output (C2J) and finally a discriminator that classifies Jazz samples as real of fake.

3.3.2 Network Architecture

What really changes from the previous GAN is that generators do not take random noise as input anymore but music samples. That is why we had to change their architecture. We took inspiration from this paper [21] that gives an implementation of a famous CycleGAN which transposes pictures of horses into zebras with astonishing results. In this new architecture, we have convolution layers at the beginning of our GANs in order to treat the input samples. Theses convolution layers are connected to residual blocks connected to our previous architecture. Regarding the discriminator, we did not change the architecture but they might need to be enhanced.

3.3.3 Training method

The goal of the CycleGAN is to change the genre of the sample music wihle keeping the essence of it. To achieve this trade-off, we use the following training method: We feed the J2C generator with Jazz music and feed the output to the Classical discriminator. This allows us to train the discriminator by showing it real and fake data and gives us the feedback for the J2C generator. We then feed the J2C generator with Classical music and compute the difference between the output and the input. Finally we compute the "cycle loss" by feeding the J2C generator with Jazz, feeding the output to the C2J generator and finally computing the difference between the final output and the original sample. We follow a symmetrical process for Classical music at the same time.

3.3.4 Results

We did not have really satisfying results with style transfer. Our guess is that it comes from the discriminators' architecture that may require enhancement since we have observed a constant error on both discriminators on both of their losses (on real and fake data). This ends up with generators which just remove some notes but do not add anything, for both Classical and Jazz. This behaviours is understandable : removing some notes can trick the discriminator easily at first whereas adding notes is more risky because it may be wrong.

Figure 6: Original Jazz song (a)

Figure 7: Classical song output (a)

Sometimes, the generator adds notes but they are artifacts and do not depend on the input song, as it can be heard here (orignal Jazz song can be listened to here).

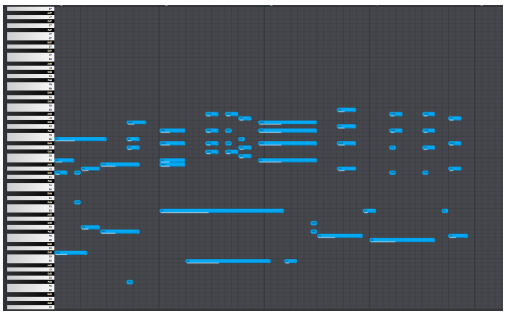

Figure 8: Original Jazz song (b)

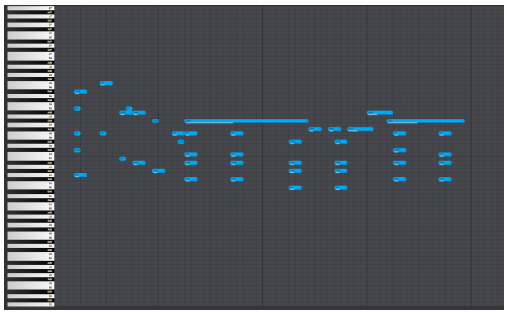

Figure 10: Original Jazz song (c)

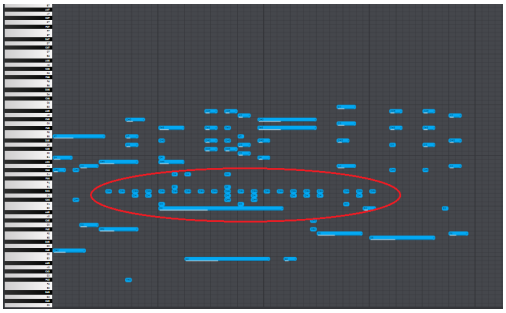

Figure 9: Classical song output with artifact (b)

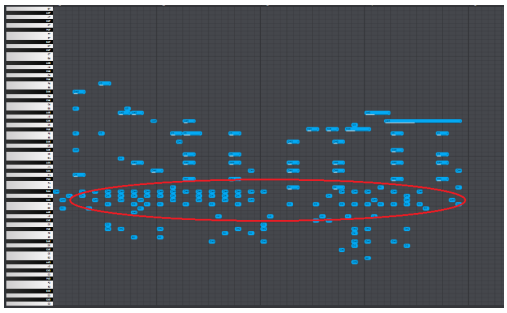

Figure 11: Classical song output with artifact (c)

This is a first step towards style transfer and this is encouraging to have some result with more investigation.

4 Results

During the first stage of the project (music generation), we had some metrics to invalidate our generated data such as the average number of note played simultaneously or the data distribution. But in the second stage (music style transfer), these metrics were not relevant enough to tell us if the samples were realistic or not. For the music generation task, we decided to test the DCGAN outputs by using a Google Form (available here), which collected 104 answers. We mixed real samples from the datasets with fake ones and collected the guesses. The results can be seen on figure 12 (generated songs) and 13 (real songs).

A first insight is that the number of good answers for the real data varies a lot. Some of them are even classified as AI music by the majority. This shows the poor quality of the data when processed as an input for our models and we might want to improve the data processing to get better results. Regarding the generated data, the number of good answers does not vary a lot and none of the songs were qualified as human by the majority. However, one AI sample did convince nearly one third of the guessers. This is far from being a successful Turing test but this is encouraging.

5 Conclusion

The use of DCGANs on MIDI representation of music has a huge potential that we just touched with this project. The main challenge on this project was to add the ”sustained” note status and the results obtained with that approach were a good achievement. Even though these results are satisfying, they could be enhanced with a longer training time and better data processing. For instance, a future work could be to model the velocities as values between 0 and 1, and not just 0 or 1, which could enable us

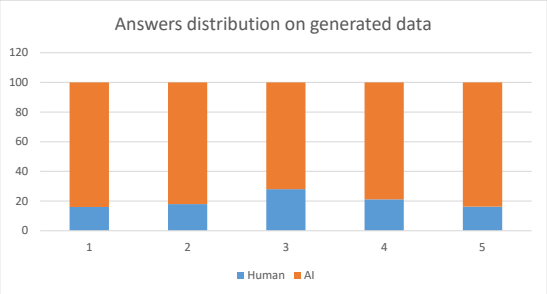

Figure 12: Answers distribution for each five generated songs

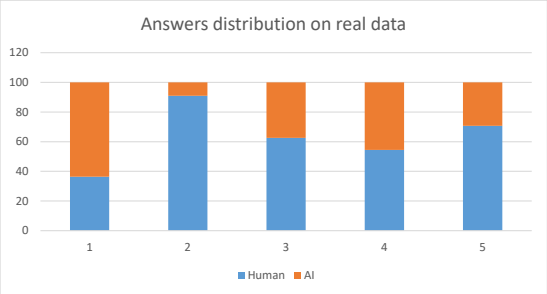

Figure 13: Answers distribution for each five real songs

to grasp more subtleties in artistic pieces. Regarding style transfer, we would have liked to explore more this part. With some changes in hyperparameters and architectures (for discriminators especially), we believe that better results are possible.

References

[1] Malik et al. *Neural Translation of Musical Style*. ArXiv.org, 2017. 1, 3, 4

[2] Brunner et al. *Symbolic Music Genre Transfer with CycleGAN*. ArXiv.org, 2018. 1, 3, 4

[3] Jayakumar. *ToneNet : A Musical Style Transfer*. Towards Data Science, 2017. 1, 3

[4] Goodfellow et al. *Generative Adversarial Networks* ArXiv.org, 2014. 1

[5] Google. *Real World GANs: Common Problems*. Google Developers website. 2, 4

[6] Arjovsky et al. *Wasserstein GAN* ArXiv.org, 2017. 2

[7] Metz et al. (Google). *Unrolled Generative Adversarial Networks*. ArXiv.org, 2017. 2, 6

[8] Dong et al. *Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment* Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 32. No. 1, 2018. 2

[9] Chen et al. *Generating Music Algorithm with Deep Convolutional Generative Adversarial Networks* IEEE 2nd International Conference on Electronics Technology (ICET), 2019. 2

[10] Kingma et al. *Auto-Encoding Variational Bayes* ArXiv.org, 2014. 2

[11] Razavi et al. *Generating Diverse High-Fidelity Images with VQ-VAE-2* ArXiv.org, 2019. 2

[12] Hu et al. *Toward Controlled Generation of Text* ArXiv.org, 2017. 2

[13] Gatys et al. *A Neural Algorithm of Artistic Style* ArXiv.org, 2015. 2, 3

[14] Dai et al. *Music Style Transfer: A Position Paper* ArXiv.org, 2018. 3

[15] Brunner et al. *MIDI-VAE: Modeling Dynamics and Instrumentation of Music with Applications to Style Transfer* ArXiv.org, 2018. 3

[16] Yang et al. *Midinet: a Convolutional Generative Adversarial Network for symbolic-domain music generation*. ArXiv.org, 2017. 3

[17] Thalles' blog. *A Short Introduction to Generative Adversarial Networks*. https://sthalles.github.io/, 2017. 4

[18] Erwan de Lépinau, Antoine Miquel, Luca Serra. *Neural music style transfer Github page*. https://github.com/luca-serra/neural-style-transfer, 2021. 4

[19] Gluon. *Deep Convolutional Generative Adversarial Networks*. gluon.mxnet.io. 4

[20] Goodfellow et al. *Improved Techniques for Training GANs*. ArXiv.org, 2016. 6

[21] Jason Brownlee *How to Develop a CycleGAN for Image-to-Image Translation with Keras*. https://machinelearningmastery.com/cyclegan-tutorial-with-keras/, 2019. 7