Symbolic Music Genre Transfer with CycleGAN

Sanatan Sharma
sas001@eng.ucsd.edu

Shobhit Trehan
shobhitt@eng.ucsd.edu

Prakhar Pandey
p4pandey@eng.ucsd.edu

Palash Agrawal
p6agrawa@eng.ucsd.edu

Abhilash Srivastava
a8srivas@eng.ucsd.edu

Abstract

Deep generative models such as Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) have recently been applied to style and domain transfer for images, and in the case of VAEs, music. We have taken the inspiration from GAN-based models employing several generators and some form of cycle consistency loss that have proven to be among the most successful for image domain transfer. A similar model is used and applied to symbolic music which shows the feasibility of our approach for music genre transfer. The fidelity of the transformed music is improved using additional discriminators that cause the generators to keep the structure of the original music mostly intact, while still achieving strong genre transfer. Visual and audible results further show the potential of our approach. Experiments were carried for genre transfer from Pop to Classical and Jazz to Classical domains. The implementation in Pytorch is open sourced [1].

1 Background and Introduction

The concept of neural style transfer is well known. It has been shown that pre-trained CNNs can be used to merge the style and content of two images. Approaches such as CycleGAN[6] do not require the extraction of explicit style and content features, but instead uses a pair of generators to transform data from a domain A to another domain B. The nature of the two domains implicitly specifies the kinds of features that will be extracted. For example, if domain A contains photographs and domain B contains paintings, then CycleGAN should learn to transfer any painting into a photograph and vice versa. We use the same structure as CycleGAN and apply it to music in the MIDI format. The general idea of CycleGAN has been further developed and improved [1]. In the future we plan on using a more complex architecture and incorporate improvements from these works, but in this paper we focus on showing the feasibility of a CycleGAN approach to domain transfer for symbolic music.

Existing work on music style transfer includes Malik et al. [2], who introduce a model that learns to play music in the style of a human musician. Their model adds velocities to flat MIDI files which results in more realistic sounding music. While their model can indeed play music in a more human-like manner, it can only change note velocities, and does not learn the characteristics of different musical styles/genres. Brunner et al. create MIDI-VAE, a multi-task Variational Autoencoder model with a shared latent space that is capable of changing the style of complete compositions from, e.g., Classic to Jazz. In addition to note pitches, MIDI-VAE also models most other aspects of music contained in MIDI files, i.e., velocities, note durations and instrumentation. In contrast to MIDI-VAE, we do not limit the number of simultaneously played notes, which leads to richer sounding music. Furthermore, when only considering the note pitches, our method achieves a more convincing style transfer. GANs, while very powerful, are notoriously difficult to train and have generally not been applied to sequential data. However, Yu, Lantao, et al. [3] have successfully used

[1]https://github.com/shobhitd11/MUS206_CycleGAN_Music_Style_Transfer

SeqGAN with Policy Gradient. The focus of this paper lies on musical genre transfer. However, genre transfer can only be successful if the resulting music sounds pleasant. Generative models such as the Variational Autoencoder (VAE) and Generative Adversarial Networks (GANs) have been increasingly successful at generating music. We use CNN-based CycleGANs to model music and perform domain transfer.Our experiments are heavily inspired by the Symbolic Genre Transfer [4]. In addition to implementing the existing model in Pytorch, we have experimented with multiple architectures. The best results and plots are discussed in this paper in Section 3.

2 Method

2.1 Model Architecture

The Vanilla GANs consist of a generator G and a discriminator D. The generator tries to generate real looking data from noise, while the discriminator attempts to distinguish the output of the generator from real data. Since our goal is to transfer music from one domain to another, the generator does not actually get noise as input, but instead real samples from the source domain. As shown in the figure, we will refer to the two domains as A and B, where the two domains correspond to music from two different genres. A CycleGAN consists of two GANs that are arranged in a cyclic fashion and trained in unison. One generator transfers data from domain A to B and the other from B to A. One discriminator is attached to each generator output. In the architecture, blue and red arrows denote the domain transfers in the two opposite directions, and black arrows point to the loss functions. $G_{A\to B}$ and $G_{B\to A}$ are two generators which transfer data between A and B. D_A and D_B are two discriminators which distinguish if data is real or fake. $D_{A,m}$ and $D_{B,m}$ are two extra discriminators which force the generators to learn more high-level features. Following the blue arrows, x_A denotes a real data sample from source domain A. $\hat{x}_B$ denotes the same data sample after being transferred to target domain B, i.e., $\hat{x}_B = G_{A\to B}(x_A)$. $\tilde{x}_A$ denotes the same data sample after being transferred back to the source domain A, i.e., $\tilde{x}_A = G_{B\to A}(G_{A\to B}(x_A))$. Equivalently, following the red arrows describes the opposite direction. M is a dataset containing music from multiple domains.

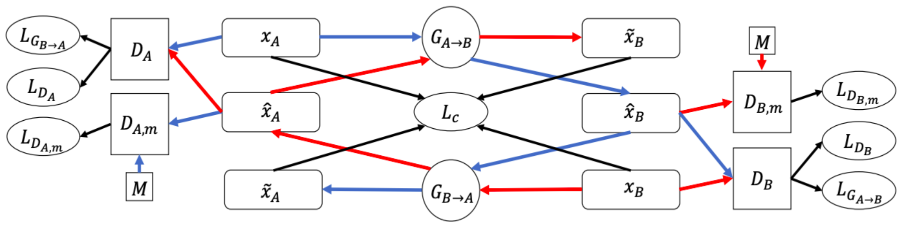

Figure 1: Architecture of our model. The two cycles are shown in blue and red respectively. The black arrows point to the loss functions. We extend the basic CycleGAN architecture with additional discriminators $D_{A,m}$ and $D_{B,m}$.

2.2 Dataset and Preprocessing

To input MIDI files to a neural network, they must first be converted into a matrix, the so-called *piano roll* representation, which can be obtained using the *pretty_midi* and *Pypianoroll* Python packages. Since MIDI notes can have arbitrary lengths, it is necessary to re-sample the MIDI file in order to discretize time and allow a matrix representation. We use a sampling rate of 16 time steps per *bar*, which means that the shortest possible note is the 16th note. A bar is a segment of time corresponding to a specific number of beats, each of which is represented by a particular note value and the boundaries of the bar are indicated by vertical lines (*bars*) on a music sheet. Thus, our final piano-roll representation is a $t \times p$ matrix, where t denotes the number of time steps (e.g., t=16 for a 1-bar piece), and p denotes the number of pitches. Also,every note has the same loudness. Therefore, the piano-roll representation contains a p-dimensional k-hot vector at each time step, where k is the number of simultaneously played notes. Because notes with the pitch below C1 or above C8 are not very common, we only retain notes between this range, i.e., $p = 84$. Thus, the piano-roll for one bar is of size 16×84. Since music has temporal structures we need to consider the relation of consecutive bars.

To avoid cluttering, we do not use highly complex pieces of music such as symphonies, as the number of different voices and instruments is too high. We further omit the drum track, since it often sounds bad when played by another instrument. As the dataset (consisting of Jazz, classical and Pop music) is noisy, we need to perform several preprocessing steps. First, we filter out MIDI files whose first beat does not start at 0. Then we remove songs whose time signature changes throughout the song, or whose time signature is not $\frac{4}{4}$. After these preprocessing steps, we have a clean dataset consisting of 12,341 Jazz, 16,545 Classic and 20,780 Pop samples, where the length of one sample is equal to one phrase, or four bars.

2.3 Architecture Parameters and Training

The batch size used is 64.Before feeding the samples to the models, we normalize the pitch values to the range [0,1]. Adam Optimize with alpha = 0.0002 and momentum decay of 0.5 & 0.99. For cycle consistency loss λ = 10 and γ=1. The model was trained for about 20 epochs. The architecture is shown in Tables 1 and 2.

Table 1: Discriminator architecture

Input: $(batchsize \times 64 \times 84 \times 1)$					
layer	filter	stride	channel	instance norm	activation
conv	4×4	2×2	64	False	LReLu
conv	4×4	2×2	256	True	LReLu
conv	1×1	1×1	1	False	None
Output: $(batchsize \times 16 \times 21 \times 1)$					

Table 2: Generator architecture

Input: $(batchsize \times 64 \times 84 \times 1)$					
layer	filter	stride	channel	instance norm	activation
conv	7×7	1×1	64	True	ReLu
conv	3×3	2×2	128	True	ReLu
conv	3×3	2×2	256	True	ReLu
$10\times$ ResNet	3×3	1×1	256	True	ReLu
	3×3	1×1	256	True	ReLu
deconv	3×3	2×2	128	True	ReLu
deconv	3×3	2×2	64	True	ReLu
deconv	7×7	1×1	1	False	Sigmoid
Output: $(batchsize \times 64 \times 84 \times 1)$					

3 Results and Discussion

The loss functions for the generators and discriminators are mentioned below. Most of the notations used here are self-explanatory. In the interest of space, we are not explaining the notations in detail.

$$L_{G_{A \rightarrow B}} = \left\|D_B\left(\hat{x}_B\right) - 1\right\|_2, L_{G_{B \rightarrow A}} = \left\|D_A\left(\hat{x}_A\right) - 1\right\|_2$$

Thus, the total loss function of the generators is

$$L_G = L_{G_{A \rightarrow B}} + L_{G_{B \rightarrow A}} \tag{1}$$

$$L_{D_A} = \frac{1}{2}\left(\left\|D_A\left(x_A\right) - 1\right\|_2 + \left\|D_A\left(\hat{x}_A\right)\right\|_2\right), L_{D_B} = \frac{1}{2}\left(\left\|D_B\left(x_B\right) - 1\right\|_2 + \left\|D_B\left(\hat{x}_B\right)\right\|_2\right)$$

$$L_{D_{A,m}} = \frac{1}{2}\left(\left\|D_{A,m}\left(x_M\right) - 1\right\|_2 + \left\|D_{A,m}\left(\hat{x}_A\right)\right\|_2\right), L_{D_{B,m}} = \frac{1}{2}\left(\left\|D_{B,m}\left(x_M\right) - 1\right\|_2 + \left\|D_{B,m}\left(\hat{x}_B\right)\right\|_2\right)$$

where M denotes mixed real data from multiple domains (here possibly Jazz, Classic and/or Pop). Thus the total loss for the discriminators is

$$L_{D,all} = L_D + \gamma \left(L_{D_{A,m}} + L_{D_{B,m}} \right) \tag{2}$$

where γ is used to weight the extra discriminator losses.

From the loss plots as shown in Figure 3, we see that the loss continues decreases for both discriminator and generator. We have experimented genre transfer between Jazz to Classical and Pop to Classical. Also, find the generated sample midi plots in section 5. The Pytorch implementation is shared [2].

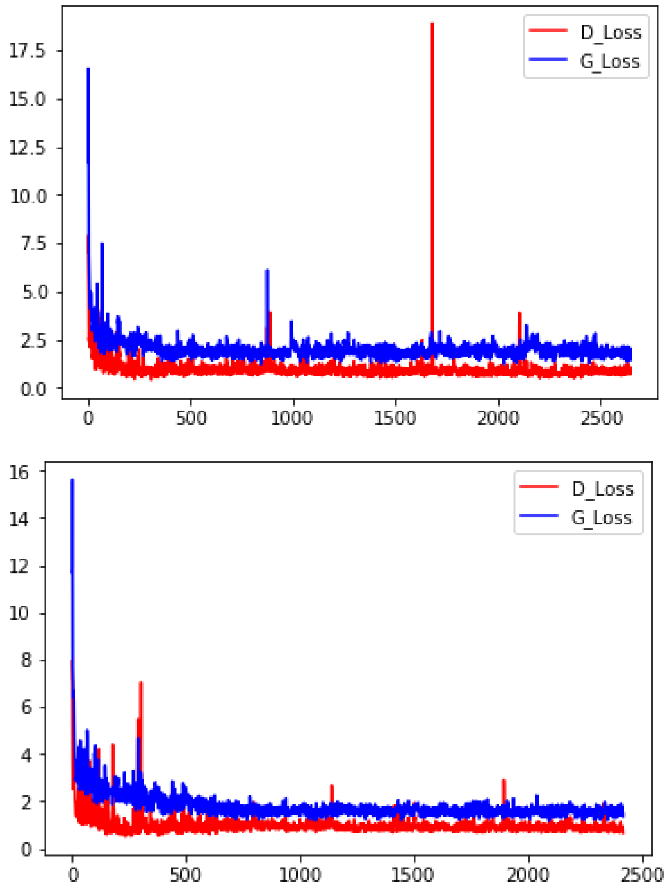

Figure 2: The plot on the top shows the loss for genre transfer from Jazz to Classical while the one below is from Pop to Classical.

4 Future Work and References

In the future we plan to develop more objective genre transfer metrics. Incorporating richer features such as velocities, note durations and instrumentation could further improve the results and make genre transfers more convincing and realistic. While in this paper we mainly focused on evaluating the basic CycleGAN architecture, more sophisticated architectures should be explored as well. Using LSTM with Reward Function for genre style transfer instead on Conv-Nets may be a good approach.

[1] Yang, Li-Chia, Szu-Yu Chou, and Yi-Hsuan Yang. "MidiNet: A convolutional generative adversarial network for symbolic-domain music generation." arXiv preprint arXiv:1703.10847 (2017).

[2]https://github.com/shobhitd11/MUS206_CycleGAN_Music_Style_Transfer

[2] Malik, Iman, and Carl Henrik Ek. "Neural translation of musical style." arXiv preprint arXiv:1708.03535 (2017).

[3] Yu, Lantao, et al. "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient." AAAI. 2017.

[4] Brunner, G., Wang, Y., Wattenhofer, R., & Zhao, S. (2018). Symbolic Music Genre Transfer with CycleGAN. arXiv preprint arXiv:1809.07575.

[5] DeepJazz, https://github.com/jisungk/deepjazz

[6] CycleGAN, https://hardikbansal.github.io/CycleGANBlog

5 Appendix

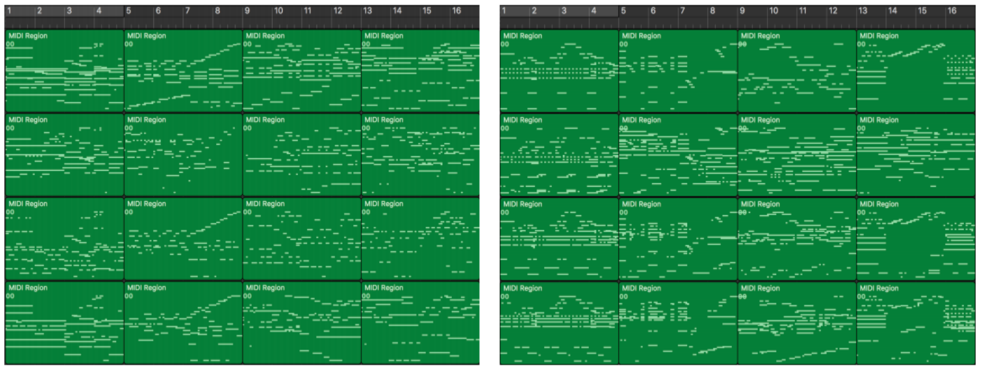

Figure 3: The figure on the left shows the samples transferred from Jazz to Classic, while that on the right shows the samples transferred from Pop to Classic.