Music Genre Transfer with CycleGAN

Christopher Cha
Department of Computer Science
University of California, San Diego
La Jolla, CA 92037
ccha@ucsd.edu

Shirley Qi
Department of Mathematics
University of California, San Diego
La Jolla, CA 92037
shqi@ucsd.edu

Abstract

We implemented a CycleGAN-based model to realize music style transfer between different musical domains, which had extra discriminators to regularize generation to achieve clear style transfer and preserve original melody. In order to evaluate our model, we also trained several genre classifiers separately, and combined them with subjective judgement to have more convincing evaluations. The resulting generated style transfer was successful in preserving aspects and motifs of the original segment, but was still too similar in genre to the original as well. We found that our results improved greatly with the addition of more data and songs in our dataset, and believe that an increase in magnitude of our dataset and a longer training loop would benefit our model greatly. A GAN-like model works well in these types of tasks, as competition between the generator and discriminator lead to more convincing results at the cost of training resources.

1 Introduction

General Adversarial Networks (GANs) are powerful Artificial Intelligence networks that uses an adversarial process to learn to generate realistic data. By training two networks at once, a generator and a discriminator, the model continues to generate more realistic data and discern true data from generated data until it reaches an equilibrium: the generator cannot fool the discriminator, and the discriminator cannot figure out which data is generated [3]. The result is a network that can generate new data without purely memorizing patterns from training data.

After its introduction, GANs have become a popular network to study and use in research. In particular, the use of GANs in style and domain transfer have become more common in recent time. In order for these networks to complete reliable and convincing domain transfers, they must be trained to extract hidden features from the training data.

One such adaptation of GANs is CycleGAN. In the paper "Unpaired Image to Image Translation using Cycle-Consistent Adversarial Networks" [6], the authors showed how they could change the style of an image through the use of GANs and cycle consistency. After comparing their network to the use of just GANs or just cycle consistency, they showed that CycleGAN is a powerful network that is useful in doing style transfers in images. With this inspiration in mind, we explored the use of CycleGAN in style transfer of music, mainly focusing on transferring jazz to classical music.

2 Related Works

The first inspiration for our project was a model called ToneNet, which was a music genre transfer neural network using an RNN-based architecture [5]. While ToneNet was a good starting point for a genre transfer neural network, the generated piece lacked the same structure as the original piece, and the style was not very clear after the transfer.

Our project is heavily based off of the paper "Symbolic Music Genre Transfer with CycleGAN" [1]. In this paper, they built a CycleGAN that can transfer the genre of an input song, and tested it with Jazz & Classical, Classical & Pop, and Jazz & Pop. Using the architecture that they have proposed. Furthermore, they created this model using Tensorflow 1, so we wanted to refactor what they already had into Tensorflow 2, as there have been many changes in the recent updates to Tensorflow.

The structure of this particular CycleGAN was inspired by MuseGAN, which is a music composition neural network that creates multi-track compositions from scratch [2]. MuseGAN captures the essence of musical pieces by training on phrases rather than specific beats or bars in order to capture structure, coherence, rhythm, chords, and more.

3 Dataset

For our dataset we used MIDI files. We specifically looked at Note On/Off, velocity, and pitch parameters in the MIDI files. Note On/Off denotes when a note is being played or not, and will help the model learn duration of notes. Velocity represents the loudness of the note and ranges from 0 to 127. Pitch ranges from 0 to 127, which is C_{-1} to G_9.

In order to pass MIDI files into a neural network, we had to convert these files to piano rolls using pretty_midi and pypianoroll. We set the shortest possible note to a 16th note. In a 4/4 time signature, each bar is 4 beats, so the total number of notes in a bar is 64. We ignore velocity, and instead just keep track of whether a note is on or off to reduce complexity, so that the velocity value only takes on values 0 or 127 rather than 128 different values. Finally, we set the pitch range to be 84, denoting the pitches between $C1$ and $C8$.

We started with 256 classical piano MIDI files and 439 jazz MIDI files. We then chose the pieces that were in 4/4 time, only had 1 time signature, and started on the first beat of the first measure. This left us with 64 classical piano pieces and 352 jazz pieces. We also condensed all of the tracks to a single piano track, and removed any percussion from the pieces. We also went through each piece and cut off extra bars at the end of the piece so that the overall number of bars was divisible by 4. After cleaning the original dataset, we split the jazz and classical pieces into a training and test set in a 90:10 split. Then for each of these sets, we combined all of the pieces into a single numpy array so that we could split everything into phrases, or 4 measures. These phrases are the data that is passed into the neural network. After all the preprocessing, we had 5508 phrases for jazz train, 684 phrases for jazz test, 1636 phrases for classical train, and 96 phrases for classical test.

We then tested the same model on a larger classical piano dataset that we found called MAESTRO [4]. The MAESTRO dataset is a collection of piano performances taken from international piano competitions. In total, there are over 200 hours of audio recordings. This gave us plenty of classical piano data to work with in order to train our model. After preprocessing the files using the same steps as before, we ended up with around 51000 phrases in the training set and 5100 phrases in the test set. We ended up having much more classical phrases than jazz phrases, which potentially caused an imbalance in learning during the training procedure.

4 Architecture

The neural network that we have used is CycleGAN. CycleGAN has 2 GANs that are being trained simultaneously: one taking input from genre A, x_A and generating genre B music, $\hat{x}_B$, and the other taking input from genre B, x_B and generating genre A music, $\hat{x}_A$. Each of these GANs consist of a generator and a discriminator, where the generator is trying to generate music that fools the discriminator, and the discriminator is trying to classify which music is real data and which music is generated.

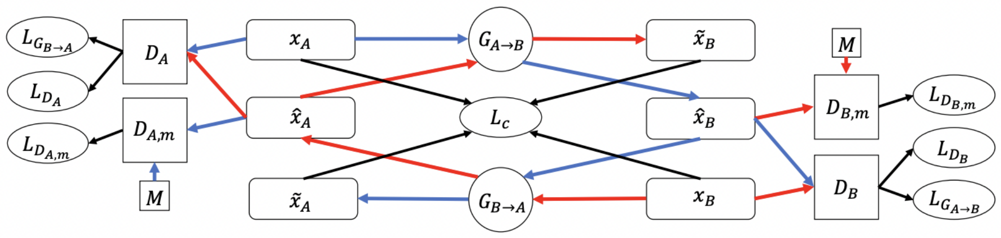

Figure 1: Architecture of CycleGAN. The two cycles are shown in red and blue arrows. Black arrows point to loss functions. $D_{A,M}$ and $D_{B,M}$ are extra discriminators to enforce higher level feature learning.

The discriminator is a convolutional neural network comprised of three convolutional layers. The generator is a mixture of a convolutional neural network and ResNet, and is comprised of three convolutional layers, 10 ResNet layers, and three deconvolutional layers. The inputs for both of these networks is a phrase, or 4 bars, of either the classical or jazz piece, which has been reshaped to $(1, 64, 84, 1)$. For nonlinearization, we used the LeakyReLU activation function for the discriminators, and the ReLU activation for the generators, with a sigmoid activation on the last layer. Details of these models can be found in Tables 1 and 2.

Input: (batchsize x 64 x 84 x 1)					
layer	filter	stride	channel	instance norm	activation
conv	4x4	2x2	64	False	LReLU
conv	4x4	2x2	256	True	LReLU
conv	1x1	1x1	1	False	None
Output: (batchsize x 16 x 21 x 1)					

Table 1: Discriminator Architecture

Input: (batchsize x 64 x 84 x 1)					
layer	filter	stride	channel	instance norm	activation
conv	7x7	1x1	64	True	ReLU
conv	3x3	2x2	128	True	ReLU
conv	3x3	2x2	256	True	ReLU
10x ResNet	3x3	1x1	256	True	ReLU
	3x3	1x1	256	True	ReLU
deconv	3x3	2x2	128	True	ReLU
deconv	3x3	2x2	64	True	ReLU
deconv	7x7	1x1	1	False	Sigmoid
Output: (batchsize x 64 x 84 x 1)					

Table 2: Generator Architecture

Each discriminator and generator have their own loss functions. The loss function we have chosen to use is L2 norm, which is

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

for the generators and

$$L_{D_A} = \frac{1}{2}(||D_A(x_A) - 1||_2 + ||D_A(\hat{x}_A)||_2)$$

$$L_{D_B} = \frac{1}{2}(||D_B(x_B) - 1||_2 + ||D_B(\hat{x}_B)||_2)$$

for the discriminators.

Another feature of CycleGAN is the cycle consistency loss, where the output of one of the generators is fed as input to the other generator to get the original genre of the music, $\tilde{x}_A$, $\tilde{x}_B$. Then, this is compared to the original piece, and weights are updated accordingly to make sure that the generated music has similar styles to the original piece. The loss function we have chosen to use for this is L1 loss, which is

$$L_c = ||\tilde{x}_A - x_A||_1 + ||\tilde{x}_B - x_B||_1$$

The total loss for generators is

$$L_G = L_{G_{A\to B}} + L_{G_{B\to A}} + \lambda L_c$$

where λ is a scalar to weight the cycle consistency loss.

Finally, to force the model to learn higher level features, we have two extra discriminators that discriminates generated data from real data that contains both genres, M. The loss for these discriminators is

$$L_{D_{A,M}} = \frac{1}{2}(||D_{A,M}(x_M) - 1||_2 + ||D_{A,M}(\hat{x}_A)||_2)$$

$$L_{D_{B,M}} = \frac{1}{2}(||D_{B,M}(x_M) - 1||_2 + ||D_{B,M}(\hat{x}_A)||_2)$$

The total loss for discriminators is

$$L_D = L_{D_A} + L_{D_B} + \gamma(L_{D_{A,M}} + L_{D_{B,M}})$$

where γ is a scalar to weight the extra discriminator loss. We used the Adam optimizer to minimize loss.

Gaussian noise $\mathcal{N}(0, \sigma_D^2)$ is also added to improve robustness and generalization of the model.

5 Experiments

We ran two different models: one without the extra discriminators, which we called base, and the other with the extra discriminators, which we called full. Hyperparameters for each model can be found in Tables 3 and 4.

hyperparameter	value
epoch	20
batch size	16
learning rate	0.0002
β_1	0.5
λ	10
γ	1
σ_D	1

Table 3: Hyperparameters for Base Model

hyperparameter	value
epoch	20
batch size	16
learning rate	0.0002
β_1	0.5
λ	10
γ	1
σ_D	0

Table 4: Hyperparameters for Full Model

At first, we trained locally on our own machines, but quickly found out that training a model of this complexity would take days. So we made use of UCSD's Datahub, which had a GPU that quickened training to a few hours. Still, because of time constraints, we weren't able to train the model for too long, and could not fully experiment with the hyperparameters and data to determine what would give our model the best results. Detailed code of our model and training and testing procedures can be found here: `https://github.com/sq19/CSE-190`

6 Results

In order to evaluate our model and the success of genre transfer, we used a pre-trained binary classifier $C_{A,B}$ that outputs a probability distribution over domains A and B.

Input: (batchsize x 64 x 84 x 1)					
layer	filter	stride	channel	instance norm	activation
conv	1x12	1x12	64	False	LReLU
conv	4x1	4x1	128	True	LReLU
conv	2x1	2x1	256	True	LReLU
conv	8x2	8x1	512	True	LReLU
conv	1x7	1x7	2	False	Softmax
Output: (batchsize x 2)					

Table 5: Classifier Architecture

The implementation of the classifier is the same as that of the paper "Symbolic Music Genre Transfer with CycleGAN", and we follow the same testing methodology and pipeline when working with our results. We used the same 90/10 train/test split during the training of the CycleGAN model, but we used the original dataset in the "Symbolic Music Genre Transfer with CycleGAN" paper to train our classifier. Our jazz dataset was classified as jazz with an 82.11% accuracy, which was comparable to the 88.89% accuracy of the original dataset that the classifier was trained on.

We considered a domain transfer $A \rightarrow B$ to be successful if the transfer accuracy was less than 0.5 and the cycle accuracy was greater than 0.5. This signified that the model was able to change from the original genre during the transfer and was able to turn itself back into the original genre after the cycle.

Model	Origin	Transfer	Cycle
Base	0.8211143695014663	0.5381231671554252	0.6715542521994134
Full	0.8211143695014663	0.2859237536656892	0.5645161290322581

Table 6: Model Accuracy

We found that the base model did not succeed in stylistic transfer, as the transfer accuracy was greater than 0.5 after 20 epochs. However, the full model that used extra discriminators was able to successfully transfer style and cycle back to the original.

We initially trained our model on a dataset of 256 classical piano MIDI files and 439 jazz MIDI files. However, we found that this amount of data was insufficient in proper training of the CycleGAN model. This lead to samples generated that sounded blocky and rigid, where chords would be played in short, quick successions sequentially with none of the original notes and structure of the song being recognizable.

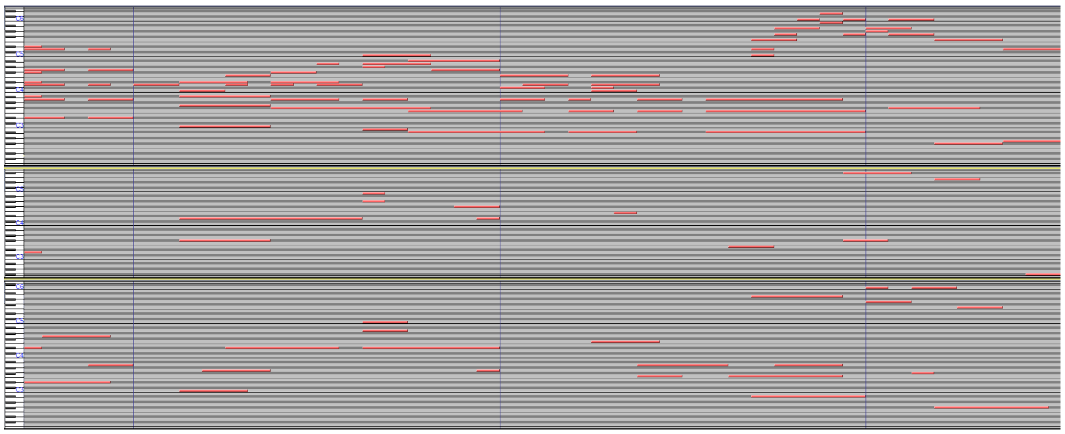

Figure 2: Top: Original Sample, Middle: Transferred Sample, Bottom: Cycled Sample

From the our results, we could see that our generated samples lacked much of the density of both notes and chords, while including many gaps. We found that an increase in data for our training set would improve performance significantly, and included a larger classical piano dataset called MAESTRO. While this did increase our dataset, we were still limited by the number of jazz samples we had available. The new generated samples retained more of the original notes and structure, but they did not sound like the new genre after the transfer. We also haven't reached a point where overfitting had become an issue as well, so an increase from our 20 epochs in the training loop would need to be tested.

7 Conclusion/Discussion

The implementation of the CycleGAN model created many paths for continuation for the project. As said in the results section, more data and a longer training loop would benefit our model and the samples it generates. However, one main issue that arose was hardware resources to train our model, as GANs take much longer to train than most standard models. Local training on our machines would take 15+ hours to complete, and this created a constraint where we could not often make small changes to our project. A future solution would be relying more on high performance computing resources, such as computing allocations like UCSD's DataHub.

There are also many avenues in generalizing and incorporating new features into this model. Our initial processing of data removes note duration and velocity, but these are components that can be included in the training loop. Furthermore, expansion to more instrumentation and potentially between different instruments could add to the genre transfer and make changes more noticeable for human listeners.

References

[1] Gino Brunner, Yuyi Wang, Roger Wattenhofer, and Sumu Zhao. Symbolic music genre transfer with cyclegan, 2018.

[2] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment, 2017.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[4] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the maestro dataset, 2019.

[5] Suraj Jayakumar. Tonenet : A musical style transfer, 2019.

[6] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.

Appendix

Link to code: https://github.com/sq19/CSE-190
Link to media results: https://drive.google.com/drive/folders/18-pE9MddNWyh5pZeiwGOeNDmNFmG_dm8?usp=sharing