COMP 576 Final Project - Artificial Art
Image Synthesis using Generative Adversarial Networks

Jonathan Cai
Rice University Computer Science
6100 Main Street, Houston, TX 77005
jmc31@rice.edu

Skylar Xu
Rice University Computer Science
6100 Main Street, Houston, TX 77005
yx48@rice.edu

Abstract

Introduced by Ian Goodfellow et. al. (2014), Generative Adversial Networks (GANs) have become a field of deep learning that has seen advancements in recent years. Utilizing a combination of two artificial neural networks, a generator and discriminator, GANs are able to learn a distribution of input data, and has been shown to have its applications in many areas including art - generating creative pieces such as paintings, music, portraits, etc.

We are interested in seeing if we can replicate some of these studies on different datasets, with lesser time/resources, and examine the functionality of particular layers in the neural network architectures utilized in SOTA research. We hope to explore the many difficulties and best practices in training GANs for this particular task of art generation, and determine which perform the best in the notoriously difficult task of training a GAN.

In particular, we attempt to replicate the results found in Elgammel et. al., CAN: Creative Adversarial Networks, Generating "Art" by Learning About Styles and Deviating from Style Norms, on a dataset of landscape images sourced from WikiArt.

1 Introduction

1.1 Importance

The general task of learning a given distribution is an important one. It has its applications in a huge variety of different fields of study, industries, etc. It has shown its usefulness in generating images with incredibly real looking objects, scenes, including the intricacies of a human face. This poses a philosophical question - what is art? Does it require a painters skilled hand and creative mind to produce a painting, or can a machine do the same? The implications of a computer learning such distributions also lead to some more insidious implications, the possibility of using this kind of artificial intelligence for less than pure purposes, like the proliferation of "Deepfake" images/videos. One must wonder if there will be disastrous consequences of this type of technology.

Were not sure if we can ever answer these questions but we can sure hope and work to understand the methodology and underlying mechanisms of such an intelligent system that is a GAN.

1.2 Novelty

We experiment with the effectiveness of different architectures consisting of convolutional and transposed convolutional layers to generate fake images. In the end, we propose a modified GAN model with a combination of various techniques to suppress the rapid learning process of the discriminator to achieve relatively stable training and produce quality fake paintings under a reasonable amount of resources and time, especially given that we are two undergraduate students with little to no budget for this project.

Replicating these impressive results found in papers like (Elgammal et al., 2017) is an important and typically overlooked task, especially with the constraints of time and resources. Wired (Barber, 2019) has highlighted this particular phenomenon in artificial intelligence research as a "Reproducibility Crisis", describing that "networks also are growing larger and more complex, with huge data sets and massive computing arrays that make replicating and studying those models expensive, if not impossible for all but the best-funded labs." We believe it is critical to the understanding of deep learning models that impressive "state-of-the-art" research in GANs and deep learning as a whole, is replicable and feasible under reasonable constraints. If we can only ever achieve great results with the best hardware and computational resources, do we truly have a grasp of how these results are derived? And if results are hardly repli-

cable, how can researchers 1. verify and check each other's work, and 2. seek to iterate and improve on each other's work?

2 Data

We used images from WikiArt, the Visual Art Encyclopedia (wik). Self-described as the "the online home for visual arts from all around the world", WikiArt features some 250,000 artworks by 3,000 artists from museums, universities, town halls, and other civic buildings of more than 100 countries. It serves a large variety of image "genres", including the landscape style, which we decided to use.

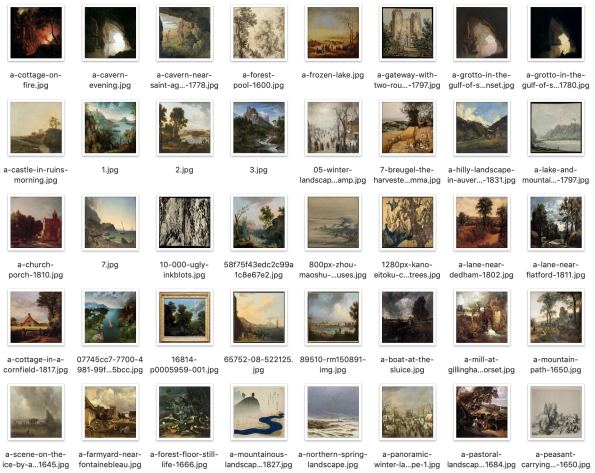


Figure 1: Sample landscape images from WikiArt.

We decided to use landscape images for a few reasons. First of all, it had one of the largest number of images in the genre with 20,000+ images, second behind the portrait genre. We also decided that we wanted images with some structure and distinctive objects/features, as opposed to more abstract styles. Most of the landscape images have a sense of structure, with the land below and sky above, and various distinctive objects like trees and people in between. Generally, we thought that there was a distinct set of learnable features found in the typical landscape image that we could have our GAN simulate.

2.1 Preprocessing

The images we scraped from WikiArt turned out to have a wide range of dimensions, ranging from 100x100, all the way to thousands of pixels wide and high.

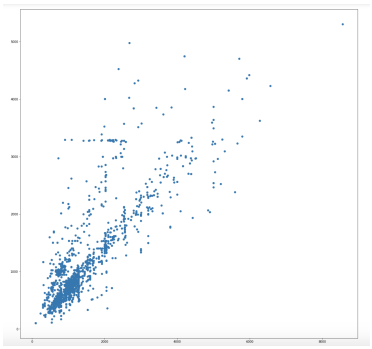
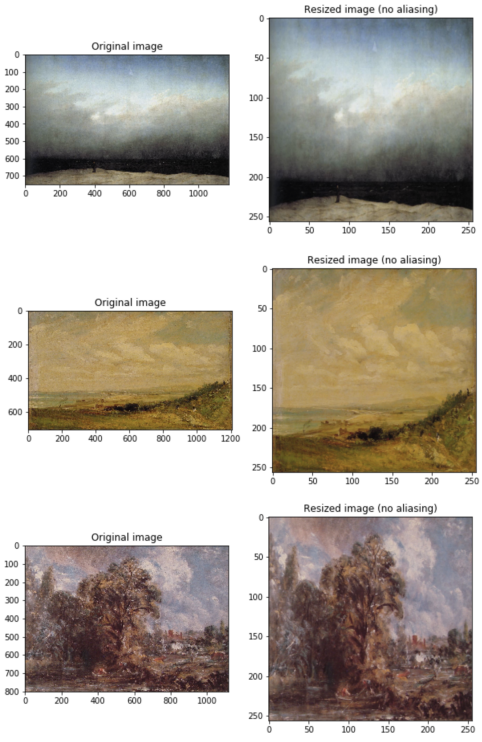


Figure 2: Plot of dimensions for each image in a sample of 2,000 from the landscape dataset (x is width, y is height). Not clear is a cluster of 100x100 images in the bottom left.

When training the model on 128x128 or 256x256 dimensions, we also removed the large number of outlier 100x100 images.

The first step of our preprocessing involved scaling all of the images to the same dimensions. As discussed later, we will eventually do several training runs on different dimensions, including 32x32, 64x64, 128x128, and 256x256, so we ended up with many scaled versions of the same dataset.



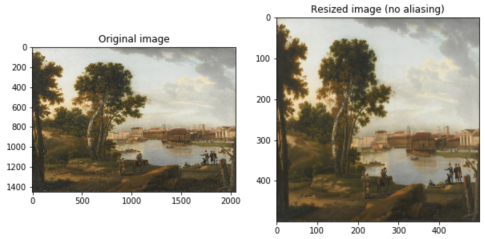
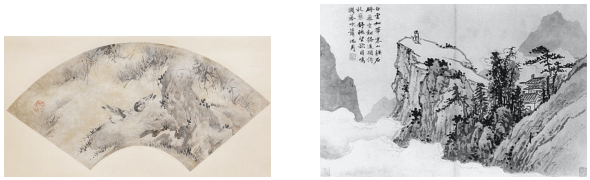


Figure 3: Four examples of the re-scaling process without aliasing.

In addition, after briefly skimming through the images, we realized that many of the images are not very representative of landscape images. In particular, there was a large number of images that were simply pictures of traditional Chinese landscape paintings on scrolls. We simply manually removed these images.



(a) (b)

Figure 4: Examples of the bad images manually removed.

3 Architecture

The proposed architecture is a variation of DCGAN, which others have proven to generate descent quality paintings.

In a traditional DCGAN model, the generator takes in a noise vector and up-samples it to a convolutional representation of a large filter size. Then it applies a series of transposed convolutional layers to

Elgammal et. al (2017) similarly uses the following architecture:

Generator

- Input: Noise (100x1)
- Transpose Convolutional $4 \times 4 \times 1024$
- Transpose Convolutional $8 \times 8 \times 1024$
- Transpose Convolutional $16 \times 16 \times 512$
- Transpose Convolutional $32 \times 32 \times 256$
- Transpose Convolutional $64 \times 64 \times 128$
- Transpose Convolutional $128 \times 128 \times 64$
- Transpose Convolutional $256 \times 256 \times 3$

Discriminator

- Convolutional 32 4×4 filters
- LeakyReLU
- Convolutional 64 4×4 filters
- LeakyReLU
- Convolutional 128 4×4 filters
- LeakyReLU
- Convolutional 256 4×4 filters
- LeakyReLU
- Convolutional 512 4×4 filters
- LeakyReLU
- Convolutional 512 4×4 filters
- LeakyReLU
- Dense 1024
- Dense 512
- Dense 1

Our first, rather ambitious attempts were aimed at generating 256×256 images. However, after days of tedious trial and error, we found it very challenging to train a GAN. In particular, it took a considerable amount of computational resources to train such a model after trying some Rice servers (yogi, garuda), Google Colab, and AWS. The resource exhaustion errors were largely caused by a lack of memory - the RAM required to allocate the large tensors in the models.

After finally finding a machine that could support training, the generated images were filled by the same pattern of pixel squares that are about the same size as the transposed convolution filters after thousands of iterations/epochs. We suspected this phenomenon was due to a sequence of multiple layers of transposed convolutional layers significantly increasing the complexity of learning for the generator. Thus, it became apparent to us that reducing the number of transpose convolutional layers was an appropriate decision to make in regards to our architecture.

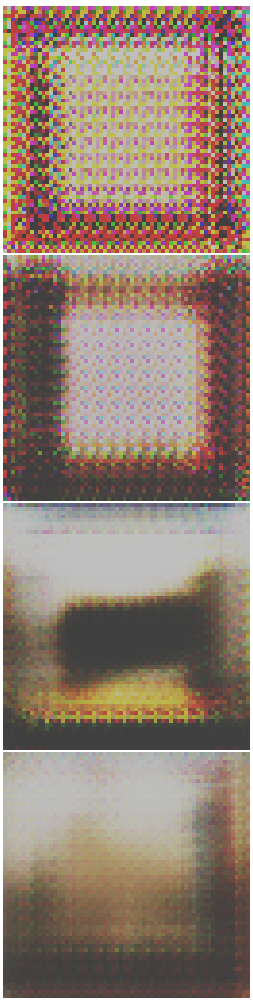


Figure 5: Generated paintings using only transpose convolutional networks in the generator. Images are 32x32, and generated at 2100, 3700, 4600, and 10800 iterations, in order.

To revolve the above issue, Utkarsh Desai (2018) proposes an architecture, being the model we find at the later time that is easier to train. It consists of a generator with a large transposed convolution and a series of convolutional networks that follow. Such an architecture would reduce the difficulty for the generator to learn with only a single transposed convolutional layer as opposed to DCGAN model. The following convolutional networks, as we suspect, would then add variation to the paintings generated. This architecture consists of the following (for 32x32 input images):

Generator

- Input: Noise (100×1)
- Dense and Reshape: $16 \times 16 \times 128$
- Convolutional: 128 5×5 filters with stride 1
- Transpose Convolutional: 128 4×4 filters with stride 2
- Convolutional: 128 5×5 filters with stride 1
- Convolutional: 128 5×5 filters with stride 1
- Convolutional: 3 5×5 filters with stride 1
- Activation: tanh

Discriminator

- Convolutional: 1283×3 filters with stride 1
- Convolutional: 644×4 filters with stride 2
- Convolutional: 324×4 filters with stride 2
- Dense: 2048×1

4 Challenges

As expected, the training process for this DCGAN was difficult and arduous, and turned out to be the bulk of our research focus. We ran into a few common pitfalls when training a GAN, and we will organize the discussion about our training process into sections regarding each difficulty we faced, and how we resolved or attempted to resolve the issue.

Now, we will move on to discuss the various difficulties we faced during training, separated into sections that are likely related or correlated.

4.1 Non-convergence

The first, and probably greatest difficulty that we faced was the generator diverging, in terms of its cross-entropy-loss function. This was evident when we monitored the loss for both the discriminator and the generator, as seen below.

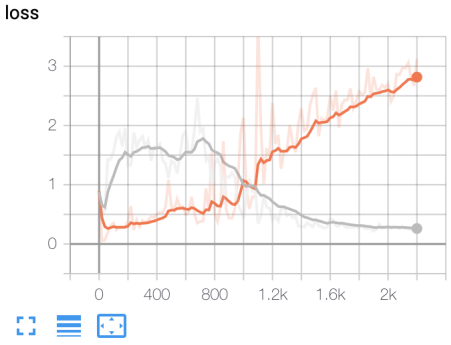


Figure 6: Example loss function divergence. We can observe that the discriminator loss (grey) continues to converge, whereas the generator loss

(orange) continues to diverge. While the discriminator is learning, we can consider the GAN as a whole to be diverging. This is a very typical thing to occur during GAN training, and should be the focus of efforts to improve training and results.

This is probably the biggest issue when developing a GAN, and can be largely attributed to the fact that there is no true objective function. The loss of both models are relative to one another, so it is incredibly difficult to even determine the progress of the GAN as a whole. The result is, we have a dynamic system where the ideal situation is that both players in the system converge to some sort of Nash Equilibrium (in game-theory terms), but typically, the discriminator is able to out-learn the generator. The typical consequence of this is that the discriminator is then able to classify nearly all samples from the generator as fake, leaving the generator without meaningful gradients and without the ability to learn the data distribution.

4.2 Mode collapse

The concept of "mode collapse" refers to a problem that occurs when the generative model loses variance in its output. As described earlier, the generator is learning a mapping from the noise input to the image output. So this mode collapse occurs when the generator maps this noise input, or latent space which is R^100 in our case of a 100x1 vector, to a very small set of output images (as seen when all the images generated in a single batch are very similar.

For us, we observed this occurring rather early on in training, as evidenced by the iterations below:

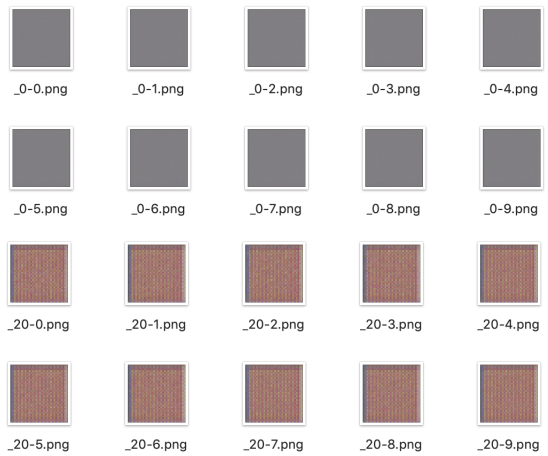


Figure 7: Example of mode collapse across several iterations. Image generation for each sample of 10 images in an iteration are created from a randomly generate noise vector (100x1). Effectively, in the occurrence of mode collapse, there is no variation in the output of the generator, and the generator has significantly reduced the size of the vector space mapped to from the latent noise space.

4.3 Lack of resources

Through our many many attempts at training our GAN, we realized the vast amount of computational resources necessary to train this model. In the higher dimensional training runs, with 128x128 and 256x256 images, many of our attempted architectures used almost ten layers, with tens of hundreds of millions of trainable parameters, even sometimes reaching a billion.

As a result, on these particular runs, we struggled to find a server that could host our training. We attempted to use our own machines, Google Colab, AWS, a friend's high-performance GPU laptop, Rice servers (yogi/garuda), and unfortunately, many of these options could not support the resources/memory necessary to run train our convolutional GAN model. Practically, the task of training a GAN to learn a distribution with images of 128x128 and above dimensionality is clearly not trivial.

As a result, we had to resort to training with lower dimensional input, typically at 32x32. The architecture was less complex, and the amount of memory required to perform the computation was far less, allowing us to get a far shorter run time and tune parameters to seek convergence faster.

4.4 Dimensionality

Trivially, as we increase the dimensionality of the data in powers of two as described, we increase complexity exponentially, both in the required computational power, but also in the function that we are approximating (the function that maps from the latent space to the space of generated images). The higher the resolution, the more detailed the features in the image distribution, the more complicated and non-linear the function becomes, the more layers and neurons required for a network to learn this function. Thus, we are faced with a dilemma, do we sacrifice more realistic, higher resolution images for more feasible computation and training runs?

5 Training Techniques

Common among all of our training runs was the general method of the training process. The training process for the discriminator involved generating a batch of "fake" images from the generator, and taking a batch of "real" images from the dataset, predicting the label, a 0-1 value where 0 indicates "real", and 1 indicates "fake". The cross-entropy-loss is then calculated as an average from the predicted labels vs the actual labels for the fake and real batches, as the goal for the discriminator is to accurately discriminate between the two. The training process for the generator involved again, generating a batch of fake images, that are then given to the discriminator to predict the label for. The cross-entropy-loss is then calculated from the results of the discriminator, and the desired label (0), as goal for the generator is to simulate images that appear as real as possible to the discriminator.

Directly from Goodfellow (2014), we can model the value of the GAN as a whole as $V(G, D)$ equal to

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

Dependent on both the generator G, and the discriminator D, this value function defines the GAN as a zero-sum game, where our two players compete to learn the distribution faster.

We start with configuring and training our model on the benchmark MNIST dataset consisting of simple handwritten digits, as below:

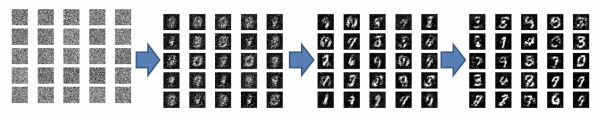


Figure 8: Visualization of training on the MNIST dataset.

Having had this baseline on MNIST, we set our sight on repulicating model from Elgammel et al. (2017) applied to landscape paintings. The original paintings taken from Wiki-Art dataset range from 200 to 2000 pixels in width and height, which leads to our decision of setting the generated image size as 256 and preprocess all input paintings accordingly. However, the learning is slow and unstable due to issues discussed in earlier sections. **Figure 9** shows generated images from this attempt. As a solution, all later experiments take in input images of 32×32 pixels and generate paintings of the same size.

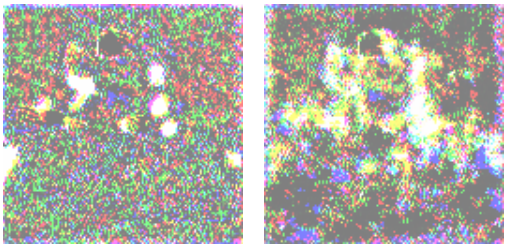


Figure 9: Results of applying full CAN model directly to 256×256 images. The amount of required computation and time is too considerable that we decided to go with resized 32×32 paintings instead as our baseline.

5.1 Less Transposed Convolution and More Convolution

As mentioned earlier, one way to improve the learning is to reduce the complexity and difficulty to learn on the generator side. We suspect that a sequence of transposed convolutional networks that scales the noise step by step from $100 \rightarrow 4 \times 4 \times 256 \rightarrow 8 \times 8 \times 128 \rightarrow 16 \times 16 \times 64 \rightarrow 32 \times 32 \times 32 \rightarrow 32 \times 32 \times 3$ would indeed reverse the convolutional process yet become too expensive to train regarding the resources/time necessary. Consequently, we employ the architecture

that scales up the noise straight to $32 \times 32 \times 128$ and uses convolutional layers to add more variation in the generator.

5.2 Soft Labels

Instead of using hard labels 0 and 1 for real and fake paintings, the proposed model randomly generates numbers in range $[0, 0.1)$ and $(0.9, 1]$ to indicate real and fake images. This prevents the loss of the discriminator to quickly go down to 0 and thus gives the generator more potential to learn and catch up with the discriminator. **Figure 10** shows the comparison between using soft and hard labels.

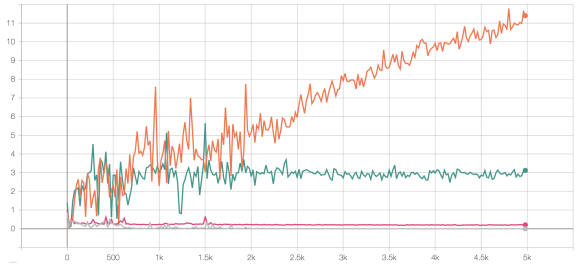


Figure 10: Comparison of losses between using hard and soft labels. Green/red lines are generator/discriminator losses for using soft labels. More stable learning can be observed for soft labels where the discriminator loss decreases more slowly and stays at around 0.2 most of the time. What happens is that when the discriminator loss approaches 0 quickly, it would be very difficult to recover the generator loss to balance out the discriminator's, as the generator loss increases linearly with hard labels shown in the graph.

5.3 Learning Rate

With Adam optimizer and soft labels used by both the generator and discriminator, learning rate 0.0001 results in slower learning than 0.001, but the generator and discriminator approaches similar loss values in both cases.

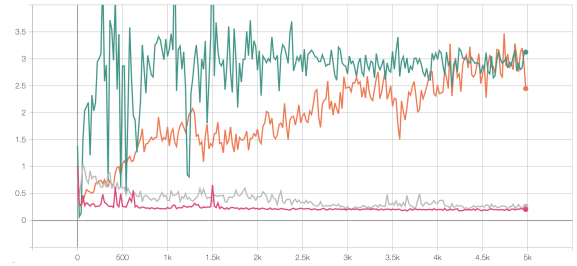


Figure 12: Smaller learning rate 0.00001 seems to be more stable at earlier batches but approaches loss values. Learning rate 0.001 leads to faster convergence.

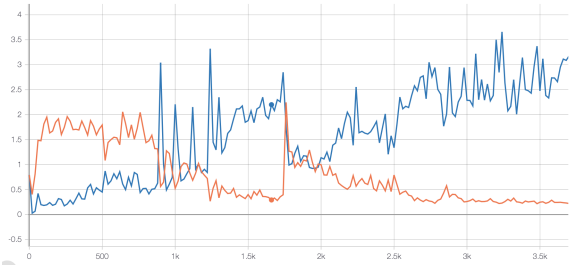


Figure 13: Divergence at around 2k with learning rate 0.0001.

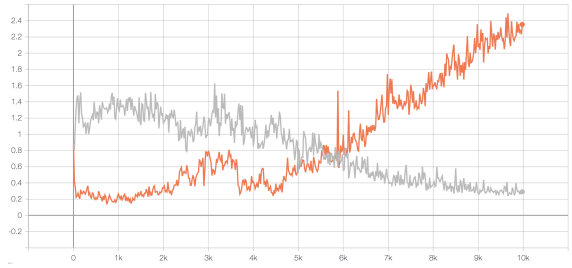


Figure 14: Divergence at around 6k with learning rate 0.00001.

5.4 Activation Function

We compare the effectiveness of using Sigmoid and Tanh as the last activation layer in the generator to scale the output to range $(0, 1)$. It turns out that this last activation is critical for the learning process, which is probably due to the fact that Tanh function scales more values towards both ends. With 5000 batches and everything else maintained the same, Tanh outperforms Sigmoid by a significant amount.

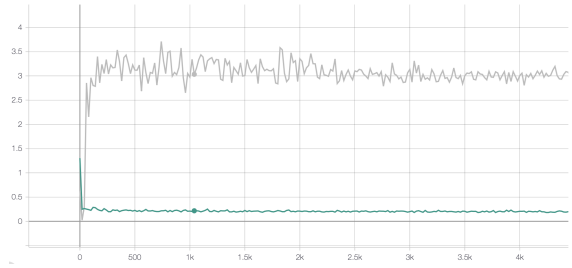


Figure 15: Using Sigmoid as the last activation in the generator.

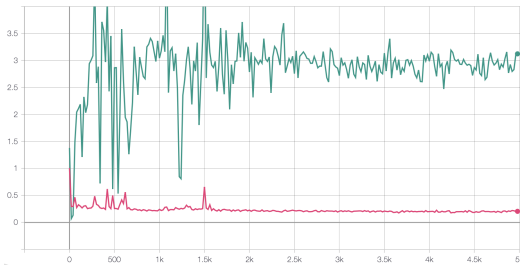


Figure 16: Using Tanh as the last activation in the generator.

5.5 Training the Discriminator Periodically

In order to slow down the learning process of the discriminator, we also attempts to stop its training during some time period, and also at different rates - these techniques described more in detail below:

Stopping discriminator training when the generator loss is large:

As the discriminator tends to always be smarter than the generator after the beginning stage, we attempts to stop its training at some point to see whether the generator's loss would decrease and balance out the discriminator's. Once the discriminator stops learning, its loss stays the same while the generator's loss drops dramatically.

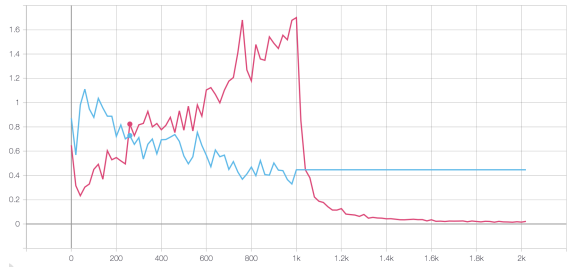


Figure 17: The blue curve is the discriminator loss, and the red curve is the generator loss.

The result indicates that the generator and discriminator are dynamically dependent on each other. The criteria of how long to stop the discriminator in order to slow down its learning is a matter of a judgement call during the training. Therefore, we propose the solution of training less frequently the discriminator once for every several batches, which slows down its learning by a moderate degree.

Training the discriminator once after training the generator x times:

This technique is employed to try to handicap the discriminator, not letting it train at the same rate as the generator. The results of this technique follow:

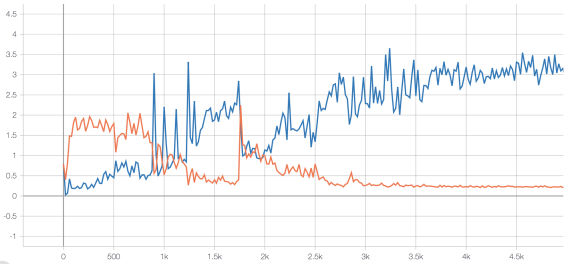


Figure 18: Training the discriminator once for every four batches.

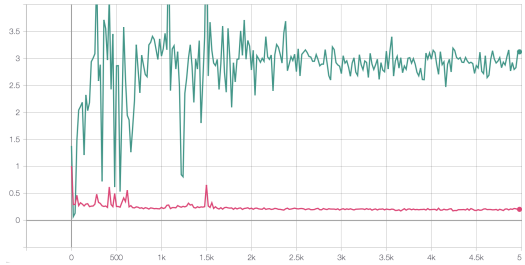


Figure 19: As a comparison, running the model without handicapping the discriminator.

As **Figure 18** and **Figure 19** manifest, the generator and discriminator counteract with each other more strongly and frequently with only selected training of the discriminator, whereas the discriminator dominates in the situation when it's trained in every batch. Given the more desirable generator-discriminator balance, we decide to incorporate this technique into our architecture and all experiments for creating our final results.

5.6 Spectral Normalization on Convolutional Layers

This is a technique derived from Miyato et al. (2018). A relatively recent development in this area, Spectral Normalization is a novel weight normalization method to stabilize the training of the discriminator in GANS.

To be more specific, explanation about this technique involves discussion about the Lipschitz continuity of a given function. Let's first begin with a discriminator D. In the context of our particular DCGAN, $D : R^{64\times64\times3} \rightarrow R^{[0,1]}$, mapping the space of input images to a label, real or fake. Since the domain and codomain of this function have inner products, we have a distance metric, the L2 norm in both spaces (Cosgrove, 2018).

If D is K-Lipschitz continuous, then $\forall x, y \in R^{64\times64\times3}$, we have

$$||D(x) - D(y)|| \leq K||x - y||$$

If K is a minimum value that meets this constraint, then it is the Lipschitz constant of our func-

tion, the discriminator D. Much of the machine-learning community has pointed out the importance of the importance of Lipschitz continuity in assuring the boundedness of statistics (Miyato et al., 2018).

Miyato proposes a method that controls the Lipschitz constant of the discriminator by constraining the spectral norm of each layer g. By definition, Lipschitz norm $||g||_{Lip} = sup_h \sigma((h))$ where $\sigma(A)$ is the sepctral norm of the matrix A (the L2 norm of A as described previously) (Miyato et al., 2018).

$$\sigma(A) := max_{h:h\neq 0} \frac{||Ah||_2}{||h||_2} = max_{||h||_2 \leq 1} ||Ah||_2$$

is the largest singular value of A (Miyato et al., 2018). The proposed spectral normalization technique normalizes the spectral norm of the weight matrix W, satisfying the Lipschitz constraint $\sigma(W) = 1$:

$$W_{SN}(W) := W/\sigma(W)$$

To utilize these special Spectral Normalization weights, we utilized a pre-existing implementation in Keras (Fang, 2019). The results of using this method can be found in the following section, Results.

6 Results

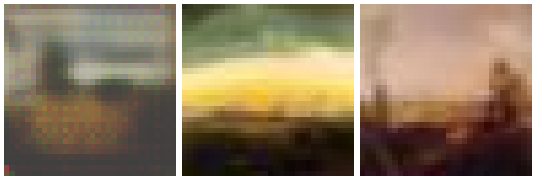


Figure 20: Which one is fake? [1]

Applying all above techniques to our modified architecture, we run our GAN model for 10000 and 50000 random batches respectively on 32×32 input paintings with 0.001 learning rate for the Adam optimizer, training the discriminator only once every four batches. **Figure 21** shows the input distribution as a reminder. According to results in **Figure 22**, even though the discriminator's and generator's losses are way off each other, they stay stable at around 0.3 for the discriminator and 3 for the generator. The two opponents are in a dynamic balance as they do not show tendency of diverging any further. This could be interpreted as the discriminator always learning a few steps ahead of the generator. They not just counteract with each other but also prevent each other from getting diverging too much, indicating some possible differential equations involving the two.

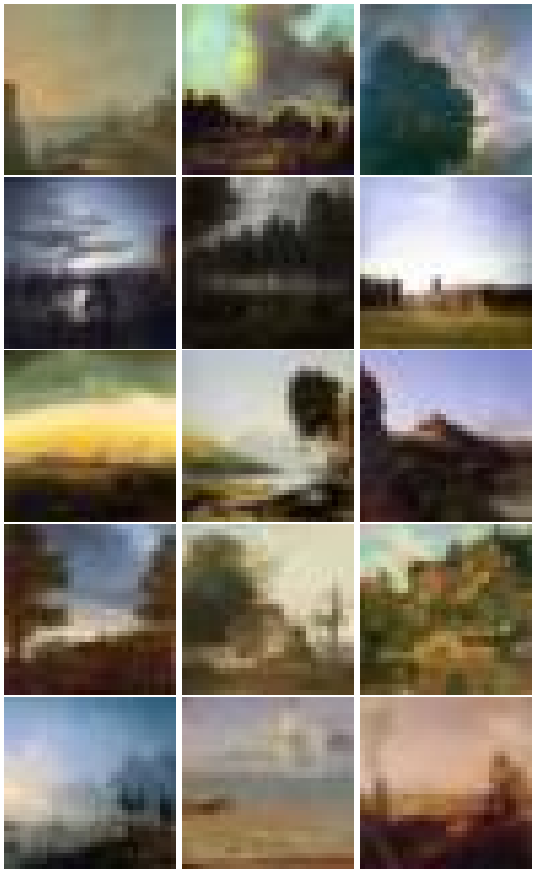


Figure 21: 15 input paintings resized to 32×32 pixels.

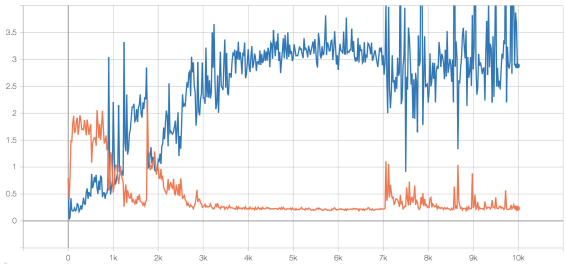


Figure 22: The loss of 10000-batch trial.

As can be seen from the results, the first "convergence" that generate landscape alike paintings mainly features the abstract distribution of colors: light color at the top and dark color at the bottom. The colors are gradient instead of solid.

We can see that during the "divergence" between around $3k$ to $7k$, the generator seems to be learning some sort of colored shapes around the center. Following this pattern, the second "divergence" happening between $7k$ and $10k$ is able to generate solid object-like shapes with saturated colors in a typical landscape background. In the

[1] The left one is fake.

last two images, it depicts the blueish sky on the top and objects that looks like trees, mountains, and yellowish grassland in the close shot, displaying much more layers and details. **Figure 23** show the formation of paintings and **Figure 24** includes some of the best fake paintings that we observe.

The experiment that runs 50000 batches exhibits the same trend but different "convergence" time, one much earlier and one much later than this 10000-batch run, suggesting that the learning of GAN is largely variant of each run.

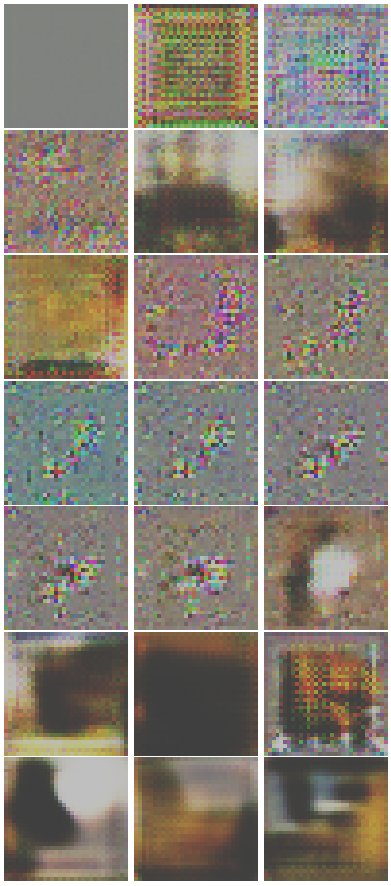


Figure 23: The evolution of generated paintings. Taken every 500 batches. Ordered from left to right, top to down. Better images corresponds to period $2k - 3k$ and $7k - 10k$ where the generator and the discriminator oscillate more towards each other.

Figure 24: Few generated paintings are impressive.

At this point, it seems appropriate to apply the model to a larger scale. We run our modified model with more iterations on Google Colab. With the computation resources and time available, we make it to around $50k$ batches (Due to some Google Colab memory issue, both experiments terminate early). The results are in **Figure 25 26**. It appears that the generator creates high-quality fake paintings between periods of grayish images during which it seems to learn to generate some new features as we suspect.

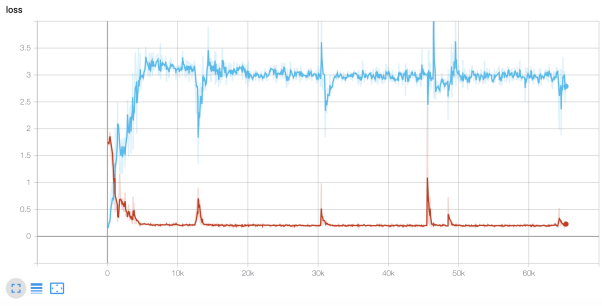


Figure 25-a: Proposed model on 32×32 images with spectral normalization applied.

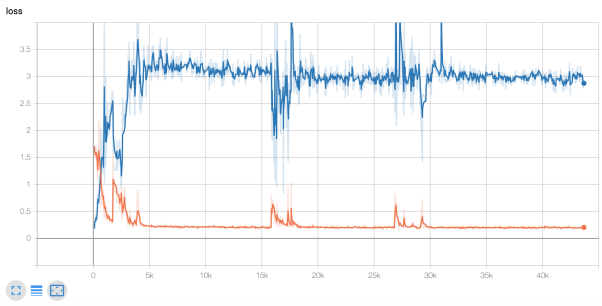
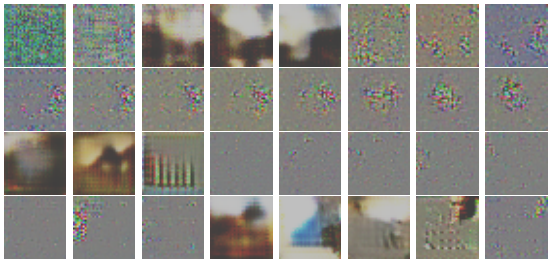


Figure 25-b: Proposed model on 32×32 images without spectral normalization.



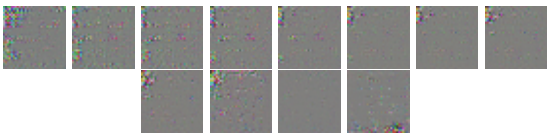


Figure 26-a: Generated paintings of sixe 32×32. Without spetral normalization. Every 1000 batches.

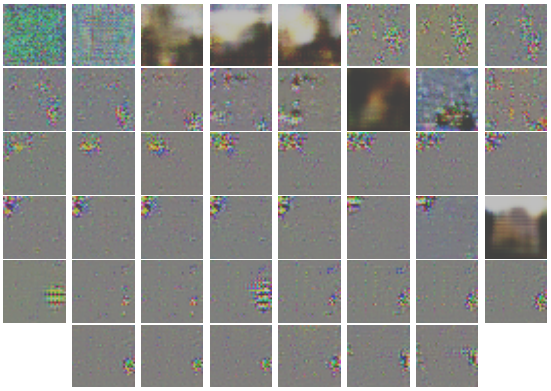
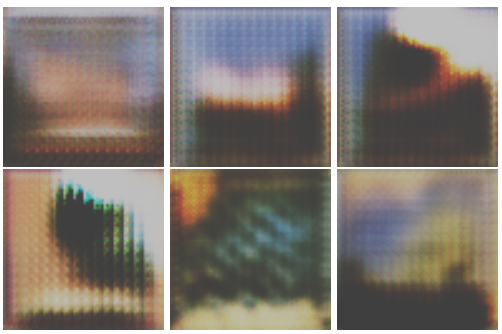


Figure 26-b: Generated paintings of sixe 32×32. With spetral normalization. Every 1000 batches.

In our experiment, there's not much of a difference when using spectral normalization except for more smooth loss curves.

Another observation that can be made is the appearance of oscillation of the loss function of the discriminator with the generator. If we observe **Figure 25-a** and **Figure 25-b**, we see a pattern where after every few ten-thousand iterations, we see a trough in the generator loss and a corresponding peak in the discriminator loss. These "oscillations" actually were correlated with the the subjective quality of the images generated - the iterations where the loss functions spiked closer are the same iterations where we found the best generated images. In contrast, the iterations between these spikes were typically static or uniform in details, without any outstanding features like objects. We observe many of such oscillations throughout training.

Further, we attempt training on both 64×64 and 128×128, seen below:



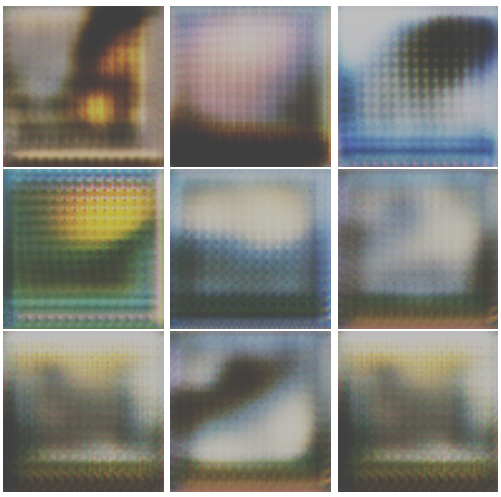


Figure 27: Generated 64×64 images.

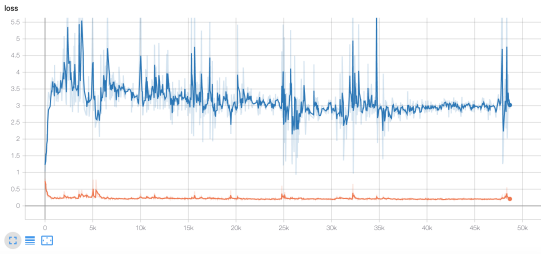


Figure 28: Loss of training on 64×64.

Figure 29: Generated 128×128 images.

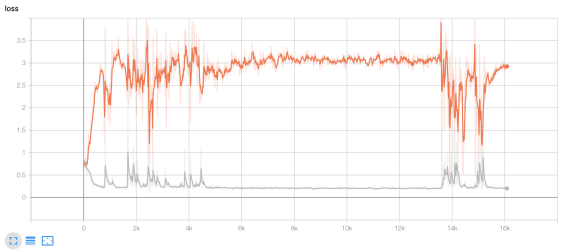


Figure 30: Loss of training on 128×128.

7 Discussion

Evident by simply looking at the resulting images and assessing with qualitative metrics, the results from our many runs do not appear to simulate the input distribution with astounding accuracy. There are certainly features that do not seem to correlate with the original dataset - particularly the pixelated appearance of edges and across the entirety of a few images. The human eye can notice these features and pretty easily attribute these to be computer-made.

We are not yet sure how to resolve this and achieve the impressive results found in state-of-the-art GAN research, but we hope that the various practical techniques/methods that can produce those results may be discovered and popularized, so that this kind of thing may be feasible in the near future.

7.1 Team Member Contributions

Over the course of the past month, all work completed for this project was completed evenly between the two of us - data scraping, data processing, research of related works, architecture modeling, experimentation process, documentation, and presentation.

7.2 Future Work

So far, our research has mainly just focused on the practical techniques that result in more stable training of GANs, but there is much more to research in this particular field, and many questions remain unanswered.

What should the criteria be to determine that training has finished, and we have reached some sort of final equilibrium, where training for both the discriminator and generator have stagnated?

How do we know for sure at a certain point during training, that the generator and discriminator have diverged too far, and will not converge again at a later time? This is especially important, as we realized much later, that most of our earlier runs we simply stopped early, assuming divergence, when they could certainly have converged several thousand iterations later.

And when we have diverged, and the discriminator loss is extremely low in comparison to that of the generator, is it possible to recover from this state?

These areas appear to us as potential areas of research in the future. There still is much more work to be done to fully understand the underlying mechanisms that allow for successful generative models like this DCGAN variation we propose here. We look forward to reading about further development and improvements in this particularly difficult task of training GANs.

References

Visual art encyclopedia.

Gregory Barber. 2019. Artificial intelligence confronts a 'reproducibility' crisis.

Cosgrove. 2018. Spectral normalization explained.

Utkarsh Desai. 2018. Keep calm and train a gan. pitfalls and tips on training generative adversarial networks.

Ahmed M. Elgammal, Bingchen Liu, Mohamed Elhoseiny, and Marian Mazzone. 2017. CAN: creative adversarial networks, generating "art" by learning about styles and deviating from style norms. *CoRR*, abs/1706.07068.

I-Sheng Fang. 2019. Spectral normalization keras. `https://github.com/IShengFang/SpectralNormalizationKeras`.

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

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. 2018. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957.