Painting with Generative Adversarial Networks

Sam Nadjari, Jeffrey Yoo, Clark Benham
University of Virginia, Charlottesville, VA 22904
[san7st, jy2ma, ce5ye]@virginia.edu

Abstract

In this project, we have trained a model to create works of art using a Generative Adversarial Network (GAN). A GAN consists of two neural networks: a discriminator and a generator. Our discriminator, D, was trained on paintings created by 50 of the greatest artists of all time, while our generator, G, was trained to produce images that resembled the paintings. We were successful in have the generator create images, but encountered mode collapse, which is a common problem with training GANs. We tried to rectify this using a variation of GAN called Wasserstein GAN (WGAN). This turned out to be unsuccessful, but we wish to use this opportunity to highlight the difficulty of training GANs.

1. Introduction

Producing creative works has long been a goal for the field of artificial intelligence. In this project, we aimed to build a neural network model that generates artwork similar to famous historical paintings. To do so, we tried to trained a deep-convolutional generative adversarial network that accepts a noise input and uses it to produce an image of an artwork. The idea is that our network models the distribution of the historical painting and that upsampling from it would produce images that appear similar to a human-drawn painting. For this project, our models were trained on a dataset of over 8,027 works of art by 50 of the most influential artists of all time [4].

Generative Adversarial Networks (GAN) were introduced by Goodfellow et. al. in 2014 [3]. The goal of GAN is to model a distribution of data based on a dataset by pitting two models against one another: a generator and a discriminator. The generator, which we will call G, is a neural network that takes in a random value z as noise input and uses it to upsample from a target distribution [3]. The discriminator, which we will call D, then takes in a sample and tries to determine whether the sample came from the real data distribution or from the "fake"[1] distribution modeled by G [3].

However, despite its promises, GANs are in practice hard to train. This is because a common issue with GANs is mode collapse, which is where the generator only learns to produce a small variety of samples. Numerous methods have been proposed such as unrolled GANs and Wasserstein GAN (WGAN).

WGAN utilizes a different distance metric to measure the similarity between G's distribution and the target distribution. Original GAN uses a loss function based on Jensen-Shannon divergence [3]. WGAN on other hand uses a loss function that is based on Wasserstein distance and is demonstrated to solve some of original GANs training problem [1]. Thus, in this project, we later chose to use WGAN as an alternative to the original GAN.

2. Related Work

There have been previous works where neural network is trained to modify a given image in a creative manner. In their 2015 paper "A Neural Algorithm of Artistic Style", Leon Gatys et. al. describe their work using a single convolutional neural network (CNN) to create art from two input images: the content image and the style image [2]. The CNN extracts a representation of content from the content image, while feature spaces in the CNN capture texture information from the style image by computing correlations between different features in different layers of the CNN.

We can see that most previous work has focused on differentiating between content and style in images, then applying the extracted style to other content representations. Our goal in this project was to go beyond simply learning to represent different components of artwork and train a set of networks to produce novel content.

3. Method

For our GAN, we decided to adopt the deep-convolutional GAN (DCGAN) framework introduced by Radford et. al. [6]. Here, G is composed of a series of transposed convolutional layers that takes in noise input z

[1]We'll use the word "fake" to describe the samples generated by the generator.

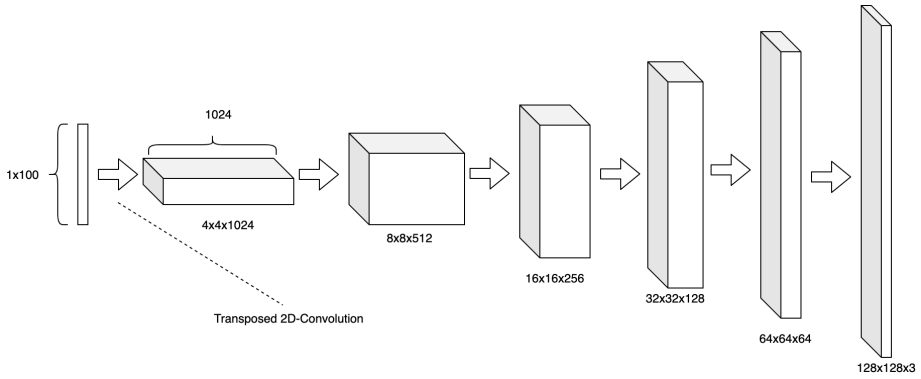

Figure 1. Architecture of generator model - Transposed 2D convolutional layers are used sample 128x128 RGB images.

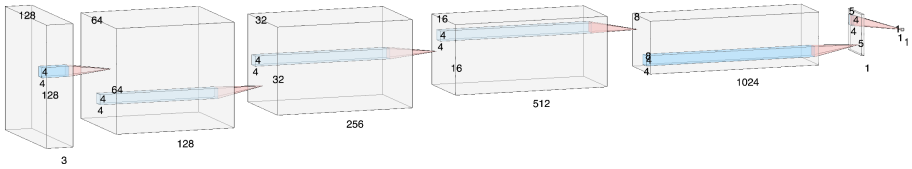

Figure 2. Architecture of discriminator model

to create an image x. D consists of a series of convolutional layers that classifies whether given x is from the dataset or from G.

We used PyTorch's DCGAN tutorial as the foundation for our model [5]. The tutorial was designed to take 64x64x3 images of faces as input. However, because the width and height of our images ranged from 122 pixels to 2,296 pixels. We decided to resize all images to 128x128x3. These dimensions would allow us to resize nearly every image without having to add dummy pixels, since nearly every image had a width and height of at least 128. Upsizing the images would also allow us to display higher resolution artwork. A sample of the training images is shown in figure 3.

Figure 1 and 2 shows the architecture of our generator model G and our discriminator model D. While both are very similar to the architecture used in the PyTorch tutorial, we added another 2D transposed convolutional layer in G and a new 2D convolutional layer in D to account for our new output image size. We then used the code provided by the tutorial to accomplish our training. A random vector of size 1×100 is given as input to G, which uses it to produce a 128×128 RGB image. D takes in a 128×128 RGB image and produces a binary classification where 0 represents that the image is fake and 1 represents that the image is real.

With the original GAN, however, we ran into the problem mode collapse. Because there were no implementations of WGAN available, we made significant changes to our training code to implement WGAN as described by the paper [1]. The main changes we made to our training were as

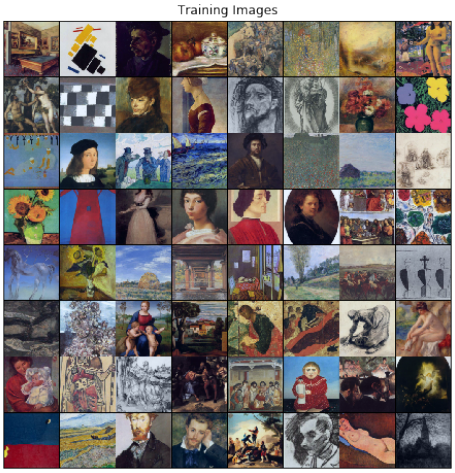

Figure 3. Sample of Training Images after Preprocessing

follows[2]:

- Instead of using binary cross entropy as the loss function, we used $D(x_i)$ and $D(g(z))$ values directly as the loss function to calculate the gradient for back-propagation [1].
- Instead of training D and G in alternating manner, we trained D certain number of steps more before training G.
- We introduced gradient clipping at the end of each

[2]This is based off how training algorithm is described by the paper.

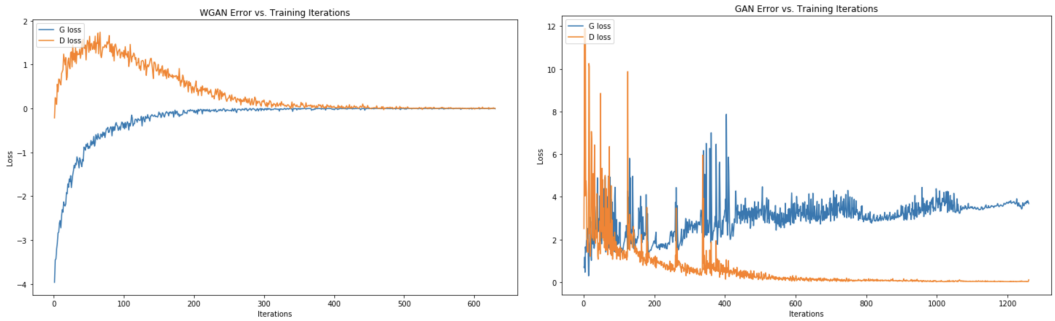

Figure 4. Plot of loss vs training iteration (Left: WGAN, Right: GAN)

back-propagation pass to guarantee Lipschitz constraint required by WGAN [1]

- We change the models to remove any sigmoid activation functions
- We switched from Adam optimizer to RMSprop optimizer.

Then, we tweaked the learning rate, clipping limits, and the number of times we train D as our hyperparameter.

4. Results

Due to the nature of how GANs are trained, we can expect both loss of D and G to oscillate (as seen in figure 4). For a stable GAN, we can expect the discriminator loss to approach 0.5 with high variance as it can do no better than randomly guess whether the output is fake or not. This would imply that the generator is doing a pretty good job. However based on our plot, we can see that in the case of both GAN and WGAN, our losses converge to 0, which indicates that our generator has failed to learn.

Also, a far easier way to determine how well our GAN is training is to just look at its outputs. Figure 5 shows the outputs of our original GAN model when we input three random z vectors. Figure 6 shows the outputs of WGAN. First, we can see that our outputs differ very clearly from human paintings. We can also see that mode collapse is happening in both cases as different z's all still produce very similar images. This means that our generator has only learned to produce a very narrow variety of images to fool the discriminator and indicates that our model has failed to learn the distribution of the historical paintings.

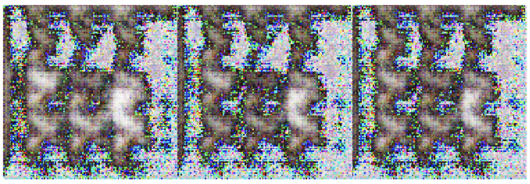

Figure 5. Three Outputs of GAN

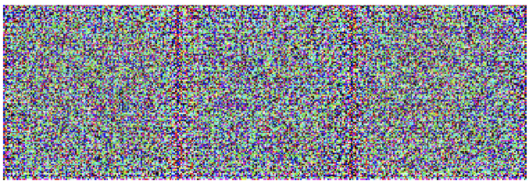

Figure 6. Three Outputs of WGAN

5. Conclusion

In this project, we have tried to build a GAN that is able to generate paintings that resemble historical paintings. While we succeeded in having GANs generate images, results tells us that both our GAN and WGAN implementations have failed to achieve the task of generating painting. However, we believe this was a good opportunity to learn how GANs worked. While we initially relied on PyTorch's tutorial to train GAN, we used it later to implement our own training of WGAN, which also required careful reading of the paper. Also, we believe that this was a good opportunity to demonstrates the difficulty of training GANs. While the idea of GAN is beautiful, the delicate balancing needed when training it reveals that we have far more work to do to make GAN truly practical.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan, 2017.

[2] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[4] Icaro. Best artworks of all time, Mar 2019.

[5] N. Inkawhich. Dcgan tutorial.

[6] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.