GENERAL ADVERSARIAL NETWORKS TO CREATE ABSTRACT ART

Tanish Ambulkar, Harsh Chobisa, Yash Tandon, Hari Senthilkumar

1. Abstract

This paper details the process undergone to build a Generative Adversarial Network (GAN) for the generation of abstract art. The different GANs (DCGAN, WGAN, etc.) and output image sizes (64 x 64, 128 x 128, 600 x 600) tested are explained in the sections that follow. Discriminator accuracy, discriminator loss, generator loss, and the human eye test were used to determine DCGAN for 128 x 128 sized images to be the best abstract art generating model we created.

1. Introduction

This project aims to teach a computer to generate abstract art resembling images from a given dataset. Creativity has been core to what it means to be a human and until recently, this was one of the biggest challenges for artificial intelligence. It is rather difficult for a machine to generate original art by itself and we will be sharing our findings as we tackled this problem.

We decided to use a GAN (General Adversarial Networks) to create abstract art. Our group decided to start with abstract art because other forms of art require a more in-depth understanding of the physical world

GANs or General Adversarial Networks are a type of neural network that utilize machine learning to generate new content. Based on the data that is fed into the neural network, GANs can generate new content that resembles the data fed in but is new in some way, shape, or form.

A GAN has two neural networks, a generator and a discriminator The generator learns to create fake images using random noise as input. The discriminator is a classifier that attempts to classify a given image as a real image from our real-world dataset or a fake image created by our generator.

Both the generator and discriminator come together in the final model where they try to outdo each other by competing with one another. The generator tries to decrease its loss, so it does a better job of fooling the discriminator, and the discriminator attempts to decrease its loss and maximize its accuracy to do a better job of identifying when it is given fake images. The worse the discriminator is doing, the better the generator is doing.

GANs have their benefits and drawbacks. They work rather well for supervised learning models as both the discriminator and generator try to trick and avoid getting tricked until equilibrium is reached. GANs can make use of convolution layers in their generators and discriminators, making them well suited for tasks involving images. However, some of their shortcomings include the long computation time due to multiple neural networks and their adversarial components. There is also a chance that equilibrium between the generator and discriminator won't be reached in which case the GAN will have failed. It is also often difficult to numerically judge the outputs since there is no methodology or objective scale to measure how good the "original" content generated by the GAN is.

To effectively train our model, we found a dataset containing over 2500 images of abstract art.

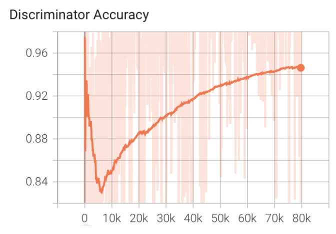
2. Results

When considering the different possible options for our GAN, we wanted to implement a more advanced model than just the vanilla GAN. We tried a variety of different setups, which will be described in more detail in this section along with the steps we took and rationale for making the modifications in the order we did. With every GAN we set up, testing was performed with hyperparameters adjusted to different values. This was to ensure we were giving each model an adequate number of configurations to determine whether it could perform as we desired. The following descriptions, pictures, and analyses of the GANs in this section are for the best hyperparameters tested for each GAN produced.

The first GAN we implemented was the DCGAN (Deep Convolutional GAN) as it removes max pooling and adds batch normalization. "The DCGAN paper mentions it is a good practice to use strided convolution rather than pooling to downsample because it lets the network learn its own pooling function. Also, batch norm and leaky relu functions promote healthy gradient flow which is critical for the learning process of both G and D" [2].

For this reason, we tried to create a DCGAN as our first model and have it output 600 x 600 images. Our initial method chosen was to train the discriminator and generator separately -- first training the discriminator, then later training the generator. This did not work well as it took very long to train. Thus, we elected to train the discriminator and generator simultaneously. We ran the model for 250 epochs, but at the end of it, the artwork being outputted did not look very similar to the training data as it was mostly noise (see example of picture generated below).

Additionally, if we look at the discriminator accuracy, it is clear that the generator was not doing a very good job fooling the discriminator as the discriminator accuracy remained above 90% for the majority of the training.

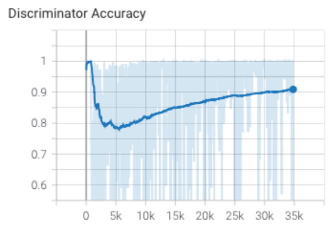


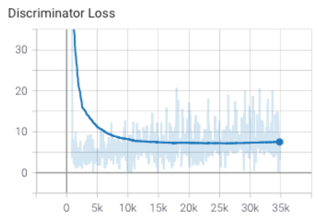
Furthermore, we believed that our model performed poorly because we were asking to do too complex of a task in generating 600 x 600 images. We didn't have enough data or computing resources to train a model to generate 600 x 600 images. For this reason, we decided to try generating 64 x 64 images. After making this change, we obtained decent results quickly, but still encountered an issue with images appearing "very brown" or "washed out" (see picture that follows).

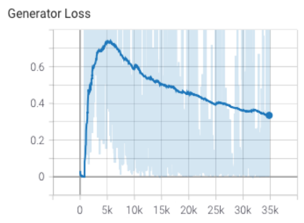
To combat this issue, we modified our data preprocessing by normalizing the training images' red, green, and blue values to means of 0.5, 0.5, 0.5 and standard deviations of 0.5, 0.5, 0.5 respectively.

This modification mostly solved the issue of output images being "washed out". However, the images were still very small and very pixelated. Since our model handled 64 x 64 quite well, we decided to make modifications to generate larger images of size 128 x 128. We also performed some data augmentation here to give our model more data to learn from. We performed three augmentations on the dataset: a 90 degree rotation, a horizontal flip, and a vertical flip. Finally, we also added in the weight initialization for GANs that is suggested in the GAN paper to our model.

As seen by the much nicer abstract art image produced and the graphs of discriminator accuracy, generator loss, and discriminator loss, it is clear that this model worked a lot better than previous ones.





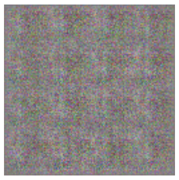


3. DISCUSSION

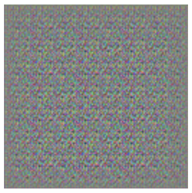
When evaluating the performance of the different models discussed, we had a ranking of what factors we prioritized. As important as the graphs were to understanding a model's performance, our goal was primarily to ensure that the images being produced seemed to be real abstract art to the human eye, not a super pixelated computer generated image. Fortunately, however, the other factors we used such as discriminator accuracy, discriminator loss, and generator loss seemed to line up well with the quality of images a given model produced. These were the 3 main metrics we chose to observe as we knew what to expect their trends to be for a good model. For a high performing model generating abstract art, the loss graphs for the discriminator and generator should both be going down over time to illustrate the generator getting better at fooling the discriminator and the discriminator getting better at deciding which images are real abstract images and which images are the fake generator produced ones. Discriminator accuracy was looked at as if it never experienced a significant drop, such as in the 600 x 600 image model case, then that indicated the generator never effectively fooled the discriminator. This was an indicator that the images produced by the generator were likely dissimilar to the training data.

After building the model for 128 x 128 images, we were able to produce many abstract art images of the desired caliber. We noticed however that when we trained this model for a much longer duration, we would frequently generate images with many similar features and very slight changes. This is a phenomenon known as “mode collapse” in a GAN. In an effort to improve our model further and create greater variety in the produced images, we tried a few different approaches to fix this issue.

The first of these approaches was implementing a Wasserstein GAN (WGAN) as we had read that WGANs often train quickly and achieve stellar results. A WGAN is a “is a GAN variant which uses the 1-Wasserstein distance, rather than the JS-Divergence, to measure the difference between the model and target distributions” [3]. However, when we implemented a WGAN in our model, training slowed significantly, and the images being produced were very pixelated and reminded us of static noise (see image below as an example). Furthermore, the WGAN took very long to train due to change in loss function and that in our WGAN, we performed five gradient descent steps on the discriminator for every single step we took on the generator. For this reason, we stopped experimenting with WGANs and attempted a different approach.



The next modification we tried was to freeze the discriminator and only train the generator. We tried this but the images outputted seemed to look very similar to those outputted by our model when we implemented the WGAN -- very pixelated (see image below). We believe that this happened because once the discriminator was frozen, the generator was able to overfit itself to noise that would fool the discriminator.

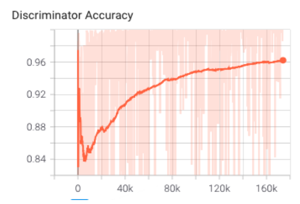
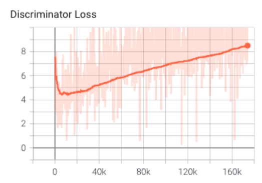
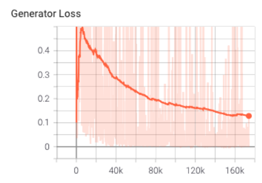
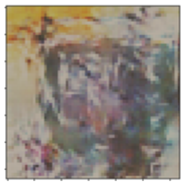
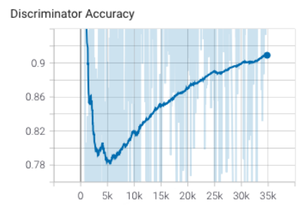
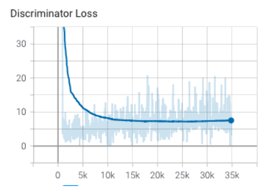
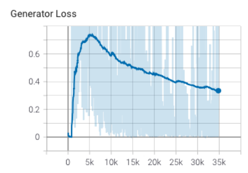
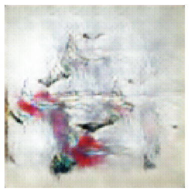
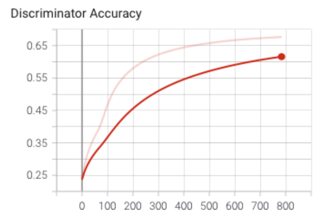
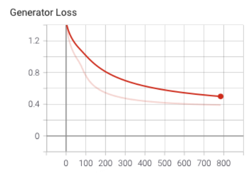
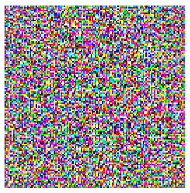
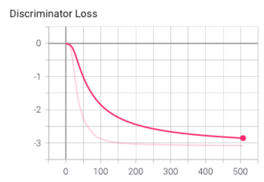
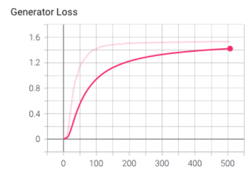
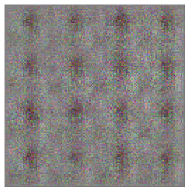


With these methods unable to produce images close to the quality of our earlier models, we decided that our 128 x 128 image producing DCGAN model was by far the best. The images it was producing looked closest to the training data images and the graphs for accuracy and loss followed the expected trends as well. Based on all the adjustments we made, we believe that the 128 x 128 model performed the best due to a few reasons. The greatest reason may have been that it was the ideal size. The 600 x 600 images were too large, and we did not have the training data nor the computing resources to train a large enough neural network that would effectively generate 600 x 600 sized abstract art images. The 64 x 64 images may have been too small, meaning there was less room for artistic creation and expression for the model.

4. References

1. https://machinelearningmastery.com/how-to-code-generative-adversarial-network-hacks/
2. https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
3. https://www.depthfirstlearning.com/2019/WassersteinGAN

Table containing the primary models we tried and their performance:

Model	**Discriminator Accuracy**	**Discriminator Loss**	**Generator Loss**	**Example of Generated Artwork**	**Summary of Performance**
DCGAN for 64 x 64					Artwork generated is decent but looks “washed out”
DCGAN for 128 x 128					Artwork generated looks good
DCGAN for 128 x 128 with discriminator frozen		N/A as discriminator was frozen			Artwork generated is noise, terrible performance
WGAN for 128 x 128 (Final Model)	N/A as discriminator accuracy is not applicable for WGANs				Artwork generated is very pixelated, bad performance

Summary of Final Architecture and Training Performed:

Generator

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1          [-1, 1024, 4, 4]       1,638,400
       BatchNorm2d-2          [-1, 1024, 4, 4]           2,048
         LeakyReLU-3          [-1, 1024, 4, 4]               0
   ConvTranspose2d-4           [-1, 512, 8, 8]       8,388,608
       BatchNorm2d-5           [-1, 512, 8, 8]           1,024
         LeakyReLU-6           [-1, 512, 8, 8]               0
   ConvTranspose2d-7         [-1, 256, 16, 16]       2,097,152
       BatchNorm2d-8         [-1, 256, 16, 16]             512
         LeakyReLU-9         [-1, 256, 16, 16]               0
  ConvTranspose2d-10         [-1, 128, 32, 32]         524,288
      BatchNorm2d-11         [-1, 128, 32, 32]             256
        LeakyReLU-12         [-1, 128, 32, 32]               0
  ConvTranspose2d-13          [-1, 64, 64, 64]         131,072
      BatchNorm2d-14          [-1, 64, 64, 64]             128
        LeakyReLU-15          [-1, 64, 64, 64]               0
  ConvTranspose2d-16         [-1, 3, 128, 128]           3,072
             Tanh-17         [-1, 3, 128, 128]               0
```

Discriminator

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 64, 64]           3,136
       BatchNorm2d-2           [-1, 64, 64, 64]             128
         LeakyReLU-3           [-1, 64, 64, 64]               0
            Conv2d-4          [-1, 128, 32, 32]         131,200
       BatchNorm2d-5          [-1, 128, 32, 32]             256
         LeakyReLU-6          [-1, 128, 32, 32]               0
            Conv2d-7          [-1, 256, 16, 16]         524,544
       BatchNorm2d-8          [-1, 256, 16, 16]             512
         LeakyReLU-9          [-1, 256, 16, 16]               0
           Conv2d-10            [-1, 512, 8, 8]       2,097,664
      BatchNorm2d-11            [-1, 512, 8, 8]           1,024
        LeakyReLU-12            [-1, 512, 8, 8]               0
           Conv2d-13           [-1, 1024, 4, 4]       8,389,632
      BatchNorm2d-14           [-1, 1024, 4, 4]           2,048
        LeakyReLU-15           [-1, 1024, 4, 4]               0
           Conv2d-16              [-1, 1, 1, 1]          16,385
          Flatten-17                    [-1, 1]               0
          Sigmoid-18                    [-1, 1]               0
```

Other models tried:

	64 x 64 DCGAN	128 x 128 frozen discriminator	128 x 128 WGAN	128 x 128 final model
Architecture	Same as above, but remove layers 14-16 from gen and remove layers 13-15 from dis	Same as above	Same as above, but remove layer 18 in discriminator	Same as above
Learning rates	2e-4	2e-4	5e-5	2e-4
Data processing	none	Image normalization, Data augmentation	Image normalization, Data augmentation	Image normalization, Data augmentation