Low-shot Image Generation of Landscape Paintings using Generative Adversarial Networks and Differential Augmentation

Haoyu Jia
UC San Diego
San Diego, CA 92092
hyjia@ucsd.edu

1 Objective

1.1 Primary Objective: StyleGAN for painting generation

The objective of this project is to adapt a generative adversarial network (GAN) that can use small datasets of a few hundred images (low-shot) to generate novel and realistic looking images. For context, usual image datasets like CIFAR-10 have tens-of-thousands of images. For the sake of simplicity, StyleGAN2 from Nvidia [1] will be the base architecture used to generate landscape photos, as it's a relatively recent state-of-the-art architecture for image generation. To take advantage of more powerful computing resources, training will be done with a GPU on Google Colab.

1.2 Secondary objective: GAN Architecture Optimization

StyleGAN2 is primarily designed for face generation, and while it has been used for generating non-human images (particularly cars[2]) the architecture has a more difficult time generating realistic images. Therefore, minor changes in parameters or elements within the StyleGAN2 architecture can be made generate more convincing landscapes. Improvement may either come from faster training speed, better resource allocation, or faster convergence.

2 Significance and Challenges

2.1 Project Novelty

While individual process improvements have been proposed for generative adversarial networks, they are often tested in large datasets like CIFAR-10 with established state-of-the-art architectures. However, newer process improvements aren't commonly tested with each other to see how well they stack with other improvements. Furthermore, Most architectures and process improvements rely on very large datasets with a lot of compute power (4+ GPUs and multiple days of training), so many of these architectures and process improvement aren't not tested with small low-shot databases.

2.2 Project significance

While StyleGAN and similar architectures have been thoroughly tested for generating human faces, both realistic and stylized, there has been relatively little exploration into applying these architecture to non-human images. In general, GAN architectures are designed and trained with only face datasets, which have relatively predictable features that a GAN architecture can identify and train with. Non-human images may often times involve different object features, lighting conditions, or other characteristics that aren't usually tested during design. In contexts like artistic works, feature

abstraction makes it more difficult to identify image subject features. Thus, it is important to test the robustness of GAN architectures with datasets of different subjects.

2.3 Implementation challenges: Data Sparsity

In general, generative adversarial networks can require up to 70000 training images to get realistic results, which increases training time. There are however implementations of StyleGAN2 [3] that are able to achieve acceptable results with a few thousand training images. For this project, best practices from data-sparse implementations will be adapted in order to simplify training.

2.3.1 Training speed

Another challenge with generative adversarial networks involves with the time required for training. Part of that has to do with the above issue of training data, but state-of-the-art solutions are more complex than architectures like variational autoencoders. For StyleGAN2, training a 256x256 image on a Tesla V100 GPU using training length of 10000kimg takes 13 full days. This is in contrast with the much slower Tesla P100 available on Google Colab, which on the free edition can only train for 12 hours at a time.

3 Background

A generative adversarial network [4] (GAN) is comprised of two separate neural networks: the generator and the discriminator. The generator attempts to create an image from random input, while the discriminator compares the generator's output with a real image to figure out which image is the real one. The loss is fed back to the generator and the process continues until the generator is able to convincing enough outputs. Since the publishing of the paper in 2014, many GAN architectures have since been created, and two of the more popular architectures are StyleGAN and CycleGAN.

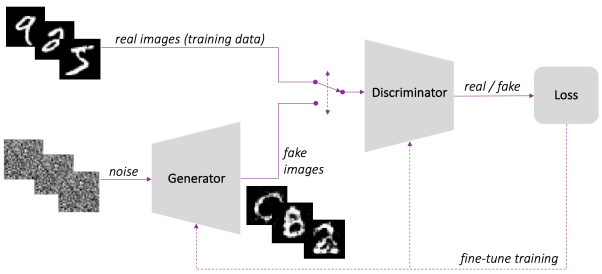


Figure 1: Generalized diagram of GAN architecture

For image generation, StyleGAN2 is the state-of-the-art architecture for most applications. The main differentiating point between StyleGAN and the base GAN architecture comes the generator. Among other changes, StyleGAN generates an intermediate latent space that goes through adaptive instance normalization (AdaIN) after each convolution step [5]. This architecture was further edited in StyleGAN2 to increase accuracy. As mentioned, StyleGAN has provided convincing image generation for various datasets, which has made it a very popular architecture for applications like image generation.

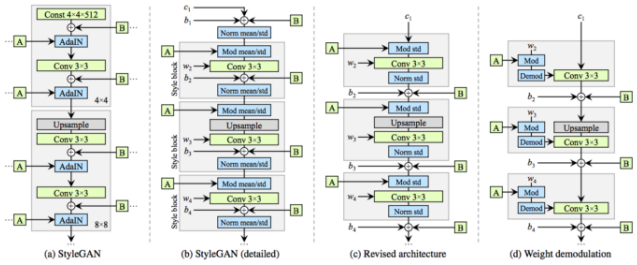


Figure 2: Diagram of StyleGAN architecture (left) and revised StyleGAN2 architecture (right)

In terms of image-to-image translation, CycleGAN is a relatively straightforward architecture that can easily be adapted on top of a StyleGAN-based architecture. The architecture replaces the singular generator with two back-to-back generators, usually in form of autoencoders, that map inputs from one domain to another then back again [6]. The discriminator will compare the generators output with training inputs to determine which on is the real image. This design ensures that the image-to-image translation of the generator can be back-propagated to resemble the original image, and that the generators aren't focused on generating irrelevant images that can fool the discriminator. As mentioned, CycleGAN has been used extensively for image-to-image translation, though there are more recent (and more complicated) architectures that have shown greater accuracy [7].

4 Theory and Method

This section will explain the architecture of a progressively growing GAN (which StyleGAN is based on) as well as the features that comprise of StyleGAN.

4.1 Progressive GAN

As mentioned, any GAN is comprised of a generator (initialized with random input), a discriminator, and a given loss function. The base architecture of StyleGAN uses a progressively growing GAN (a.k.a ProGAN [8]), where a generator begins with generating a small image and and generates a higher resolution at each progressive layer until it reaches the base image resolution. For example, a 1024x1024 image is initialized with a latent random vector and generates a 4x4 image at its first layer. Subsequent layers double the dimensions of layers (8x8, 16x16, etc) until the final layer generates a 1024x1024 image. Conversely, the discriminator trains at a 1024x1024 image, then subsequent layers halve the dimensions until the final layer of the discriminator discriminates from a 4x4 image.

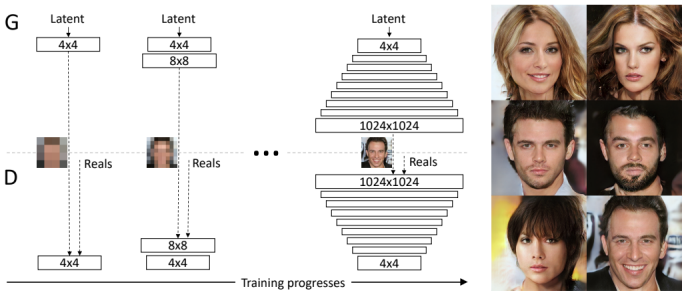


Figure 3: Diagram of StyleGAN architecture (left) and revised StyleGAN2 architecture (right)

ProGAN is able to generate high-definition image that look relatively realistic, but the network is unable to disentangle and isolate features. A random change in inputs can change multiple features of generated images. This will be the base architecture from which changes will be made to create StyleGAN.

4.2 Mapping network

A normalized random vector (For a 1024x1024 image vector is of size 512x1) goes through an 8 fully connected layers to generate a style vector W of same size. This style vector gets incorporated into the generator via adaptive instance normalization.

4.3 Adaptive instance normalization

Adaptive instance normalization aligns the mean and variance of the content features (x) with those of the style features (y) [9]. The adaptive instance normalization for StyleGAN is given as:

$$AdaIN(x_i, y) = y_{s,i} \frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}$$

4.4 Constant input tensor

For initializing the generator of 1024x1024 image, a constant 4x4x512 input tensor is used instead of a random point.

4.5 Adding Noise Inputs

After convolution at each layer, we add single-channel images consisting of uncorrelated Gaussian noise. This will help in making the generated images look realistic, as not adding any noise will make the image look too stylized.

4.6 Mixing Regularization

The generator uses intermediate vectors at each layer of the generator, which might result in the GAN to learn that adjacent styles are correlated. Mixing regularization randomly mixes the latent style vectors during training, which means the generator will not try to rely on the correlation between levels to take shortcuts in generating higher dimension images. While mixing regularization generally has a relatively small effect on final performance, combining a set of latent style vectors does generate a coherent images that mixes the features of all images.

4.7 Truncation Trick of W

The GAN might have difficulties generating features that are poorly represented in training data, which might result in unrealistic-looking images. StyleGAN truncates the intermediate W vector to force it closer to a more "average" vector. The center of mass of W is calculated as $\hat{W} = E_{z\ P(z)}[f(z)]$, which corresponds to the average image $\psi = 0$. We can thus scale W from the center to get $W' = \hat{W} + \psi(W - \hat{W})$, with negative values of ψ being the opposite of style vector.

4.8 Frechet Inception Distance

To measure the "accuracy" of a generated image, we must be able to compare the distribution of both the generated and original image. This is where the Frechet Inception Distance, or FID, comes in. The distribution of our generated image is given as a multidimensional Gaussian variable $N(\mu, \Sigma)$, and the distribution of real images used for training is given as multidimensional Gaussian variable $N(\mu_w, \Sigma_w)$. The FID is calculated thusly (lower is better):

$$FID = |\mu - \mu_w|^2 + tr(\Sigma + \Sigma_w - 2\sqrt{\Sigma\Sigma_w})$$

4.9 Loss Function

ProGAN uses Wassertein GAN + Gradient Penalty (a.k.a WGAN-GP) as its loss function, and similarly StyleGAN utilizes the same loss function. Below is the psuedocode for WGAN-GP [10]:

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Figure 4: Psuedocode for WGAN from source paper

5 Optimization Practices

StyleGAN2 is optimized enough that training time for a moderately-sized dataset of smaller images (i.e. 256x256) can be done in a few days, but there are hyperparameter and adjustments that can improve training speed and potentially reduce FID. Larger batch-sizes has shown modest improvements in training speed and FID [11]. The increase in training speed comes at the cost of more GPU memory usage, and if computation costs exceed GPU memory capacity the training is not possible.

Another parameter adjustment technique is the Two Time-scale Update Rule (TTUR) for GANs trains with stochastic gradient descent [12]. Since StyleGAN2 uses an Adam optimizer, which is based off of stochastic gradient descent, TTUR is achievable. The paper generally suggests a higher learning rate for the discriminator and a lower one for the generator, though performance might degrade if (under the base learning rate) the discriminator overpowers the generator.

A more involved optimization involves performing data augmentation on both the generated and real image before feeding it into the discriminator [13], which when adapted into StyleGAN2 is able to produce results competitive with the original implementation with only a fraction of the dataset.

6 Algorithm

Below is psuedocode that describes the generator of a StyleGAN algorithm.

Data: latent style vector w
Result: 1024x1024 image
image = const 4x4x512;
for *4x4, 8x8, ... , 1024x1024* **do**
  temp1 = image + noise;
  temp2 = AdaIN(temp1, w);
  temp3 = conv2D(temp2, 3, 3);
  temp3 = temp3 + noise;
  temp4 = AdaIN(temp3, w);
  image = temp4;
end

Algorithm 1: StyleGAN Generator

7 Experiments & Results

7.1 Dataset

Differential Augmentation shines well with smaller datasets in the order of hundreds of images, so for experimentation I will be using a 300-image dataset of Monet landscape paintings. Unlike the

testing datasets of the original papers, this dataset not only has non-human subjects (landscape) but also has to work around a different artistic style. More importantly, the more abstract style of Monet's landscapes will mean that generating convincing paintings will be more challenging, as the GAN needs to discern from the abstraction what the features of a given painting represent.

The images are pre-processed to be square 256x256 images and roughly centered. For the sake of computational speed, the GAN is trained for 128x128 images.

7.2 Experiments

Three different experiments will be done to asses how well GAN architectures work with low-shot datasets. All three will use the revised StyleGAN2 architecture, which makes minor improvements to both the discriminator and generator for greater accuracy. StyleGAN2 should also be faster, which should hasten training. Training length is determined in terms of thousands of images generated/discriminated (kimg), and image samples are taken at 300, 450, and 750 kimg. In general Nvidia Tesla P100 will be used in Google Colab, though some parts of the experiment might use the faster Nvidia T4.

Experiment 1 will use Differential Augmentation on both real images and generated images before they feed into the discriminator of the StyleGAN2 architecture. Batch-size will stay at the default 16, and learning rate for generator and discriminator will be equally set at 0.002. GPU memory usage should be around 8GB. Batch size is set at 16.

Experiment 2 will build off of experiment 1 by increasing the batch size from 16 to 24. Google Colab GPUs have 12-16GB of memory, so this should take as much GPU memory as possible. Since the relationship between resolution, batch-size, and memory usage is non-linear, it is advisable to play around with increasing batch sizes to find the highest value that doesn't exceed memory capacity. A GPU with 16GB of memory can handle 128x128 images with a batch-size of 24, but not 64x64 with a batch size of 48. The aim of increasing batch size is to measure increases in training speed (if any).

Experiment 3 will use the 24-image batch size but will also implement two-timestep update rule. In particular, we will increase discriminator learning rate to 0.03 and decrease generator learning rate to 0.01. This should in theory have trivial effects on training speed, though it might affect how well the GAN architecture converges./

7.3 Log Observations

To train the dataset for 750kimg, StyleGAN2 with Differential Augmentation took 25 hours, 45 minutes, and 56 seconds (122 seconds per kimg). Increasing batch size from 16 to 24 decreases training time to 19 hours and 32 minutes (74.9 seconds per kimg or 127.5seconds per kimg), while implementing variable learning rates (TTUR) took a total of 25 hours, 7 minutes, and 29 seconds (127.6 seconds per kimg). For the second experiment, the two different training speeds are due to the GPUs used at a given moment. Google Colab will sometimes provide the faster Tesla T4 GPUs, but in more congested times will provide the slower Tesla P100 or the even slowr K5. On the surface, increasing batch size does dramatically increase training speed, through implementing Two Time-scale Update Rule does require more computational resources.

In terms of GPU memory usage, the default batch-size 16 requires 8.5GB of memory, while a batch-size of 24 requires 12.3 GB of memory. Including background processes, the combined memory usage at batch-size of 24 is around 15.6GB, or just below the capacity of the 16GB GPUs used. This does seem to have an appreciable effect on training speed trivial effects on the FID score and minor effects on the real/fake image score.

7.4 Training results

(a) 300 $kimg$ (b) 450 $kimg$ (c) 750 $kimg$

Figure 5: Sample images of StyleGAN2 with Differential Augmentation. Results done at various training lengths at seed 1337.

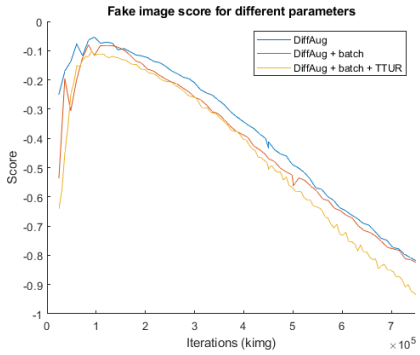
(a) 300 $kimg$ (b) 450 $kimg$ (c) 750 $kimg$

Figure 6: Sample images of StyleGAN2 with Differential Augmentation and increased batch size. Results done at various training lengths at seed 1337.

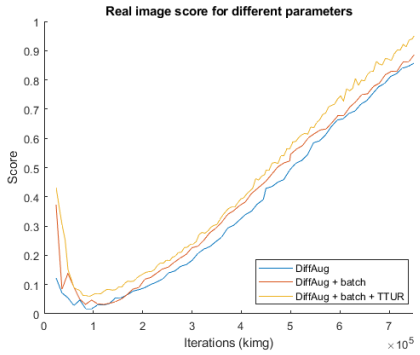
(a) 300 $kimg$ (b) 450 $kimg$ (c) 750 $kimg$

Figure 7: Sample images of StyleGAN2 with Differential Augmentation, increased batch size, and two-timescale update rule. Results done at various training lengths at seed 1337.

Artistic style seems to be easily extracted in all three experiments at 300 kimg training length, with no sign of mode collapse. landscape subjects and filters get more refined with increasing training length, though it's a mixed bad between realistic and unconvincing paintings. The GAN architecture has an easier time painting hills, lakes, and fields, but the inclusion of trees and clouds do throw it in a loop. Interestingly, even though the generation seed is the same across all three experiments, minor changes in architecture result in vastly different images being generated. Of course, longer training time could help with training fine details.

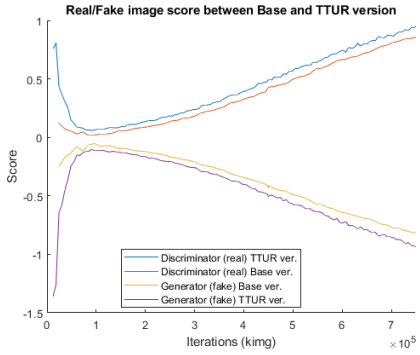


(a) 300 $kimg$

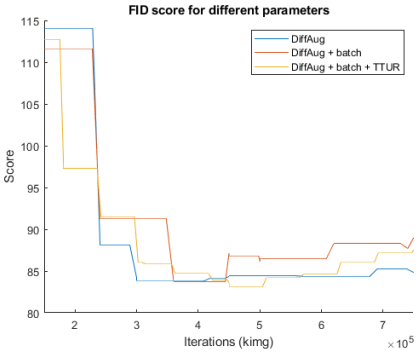


(b) 450 $kimg$

Figure 8: Sample images of StyleGAN2 with Differential Augmentation, increased batch size, and two-timescale update rule. Results done at various training lengths at seed 1337.



(a) Base vs TTUR version



(b) FID score (after 150 kimg)

Figure 9: Sample images of StyleGAN2 with Differential Augmentation, increased batch size, and two-timescale update rule. Results done at various training lengths at seed 1337.

8 Analysis & Conclusions

In aggregate, it seems like Differential Augmentation does allow for fast convergence for any architecture (in this case StyleGAN2). Within just 300,000 images loops of training, the architecture is able to generate images within the artistic style. Generating image subjects that look believable is more difficult, as some portrait subjects seem easier to generate than others, while some instances will still mostly result in noise in the vague painting style. That being said, given the challenges that come from a small dataset of (relatively abstract) landscape portraits, it seems like data augmentation combined with StyleGAN2 does show promise in challenging low-shot datasets. Batch-size seems has the possibility of increasing training speed, though that could also be hardware dependent.

In this instance, the discriminator seems to dominate over the generator, and increasing the discriminator learning rates without changing generator learning rate does make the discriminator dominate more. This does hint towards asymmetric learning rates as a useful tool in increasing convergence, though for this particular instance convergence would've been faster if discriminator learning rate was instead reduced while generator learning rate was increased.

References

[1] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and Improving the Image Quality of StyleGAN," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[2] T. Kramberger and B. Potocnik, "LSUN-Stanford Car Dataset: Enhancing Large-Scale Car Image Datasets Using Deep Learning for Usage in GAN Training," Applies Sciences 2020, 10, 4913.

[3] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila, "Training Generative Adversarial Networks with Limited Data," Computer Vision and Pattern Recognition, Jun. 2020.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," Advances in neural information processing systems, vol. 27, pp. 2672–2680, 2014.

[5] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[6] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," 2017 IEEE International Conference on Computer Vision (ICCV), 2017.

[7] K. Mei, C. Zhu, J. Zou, and S. Zhang, "Instance Adaptive Self-training for Unsupervised Domain Adaptation," Computer Vision – ECCV 2020 Lecture Notes in Computer Science, pp. 415–430, 2020.

[8] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive Growing of GANs for Improved Quality, Stability, and Variation," International Conference on Learning Representations, Feb. 2018.

[9] X. Huang and S. Belongie, "Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization," 2017 IEEE International Conference on Computer Vision (ICCV), 2017.

[10] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," Neural Information Processing Systems, Dec. 2017.

[11] A. Brock, J. Donahue, and K. Simonyan, "Large Scale GAN Training for High Fidelity Natural Image Synthesis," International Conference on Learning Representations, Feb. 2019.

[12] M. Heusel, H. Ramsauer, T. Unterthiner, B. Lessler, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," Advances in Neural Information Processing Systems, Jun. 2017.

[13] S. Zhao, Z. Liu, J. Lin, J.-Y. Zhu, and S. Han, "Differentiable Augmentation for Data-Efficient GAN Training," Neural Information Processing Systems, 2020.