Deep Learning Final Project Report - Progressive Growing of GANs

Abhiroop Gangopadhyay
Columbia University
ag3661@columbia.edu

Saahil Jain
Columbia University
sj2675@columbia.edu

Leo Stilwell
Columbia University
ls3223@columbia.edu

Abstract

We utilize generative adversarial networks (GANs) in order to generate photorealistic images from a training dataset of high quality images. We generate new images of celebrity faces based on existing images, building from recent work on photo generation via the progressive growing of GANs by Karras et al., 2018.

Specifically, we generate semi-realistic images of celebrities of size 128 x 128 based on the CelebA-HQ corpus of celebrity photos in a more constrained environment than Karras et al. due to less computing power and less training time. We attempt to improve baseline results from the Karras et al. architecture at a 128 x 128 resolution, focusing on the following metrics: Sliced Wasserstein Distance (SWD) and Inception Score (IS). Our aim is to increase the quality of the images as well as the variation in the generated images to account for a larger proportion of the training distribution.

Ultimately, via experimentation, we observed a tradeoff between SWD and IS. Our final architecture decreased SWD 4.51 percent, from 109.22 to 104.30. The IS did not change by a statistically significant amount, decreasing marginally by .16 percent. We thus investigated and offered insights into photorealistic image generation by progressively growing GANs.

1. Introduction

Deep learning has exploded in popularity recently and is being applied to many real world tasks such as image classification, speech synthesis and many more fields sure to come.

Generative methods which produce new samples from a high-dimensional distribution especially are finding widespread use, but various problems still remain. GANs typically consist of two main pieces, a generator, which produces the novel samples from a latent vector, and a discriminator, or critic, which judges whether or not that new sample did in fact come from the original distribution. Typically, the generator is of main interest the discriminator is an adaptive loss function that gets discarded once the generator has been sufficiently trained.

The problem with GANs is their quality starts to degrade at high quality images. This problem is made difficult because higher resolution makes it easier to tell the generated images apart from training images, thus drastically amplifying the gradient problem. Further, smaller minibatches must be used since higher resolution images require more memory. This leads to greater training time and instability.

Karras et al. made a key insight into solving this problem by proposing that the Generator and Discriminator should grow progressively starting from easier low-resolution images, and continuing to add new layers that introduce higher-resolution details as the training progresses to improve training stability and greatly reduce training time (Karras et al. 2017).

In our work, we attempt to improve upon the results achieved by Karras et al., 2018 in their work on progressively growing generative adversarial networks (progressive GANs) for improved quality, stability, and variation. Generating high quality and varied images becomes tremendously challenging as the resolution increases. Larger resolution images require smaller minibatches when training due to memory constraints and make it initially easier for the discriminator to differentiate between generated images and training images due to the difference in quality. Moreover, it is difficult to directly map from a latent space to a high dimensional space. By progressively adding layers to the GAN, and leaving previous layers trainable, the mapping from the latent space to low resolution images can be continuously tuned, while also learning weights necessary to transition into high resolutions. Thus, instead of learning one complex function from the latent space to the target resolution, we iteratively add trainable elements to the GAN, increasing both speed and stability of training. We plan on reproducing the results achieved by Karras et al. for smaller resolutions and then attempting to improve upon them by experimenting with different techniques aimed at increasing quality and variation in the images generated by our model.

While Karras et al. trained their network up to 1024 x 1024 resolution images, we will only train up to a resolution of 128 x 128 due to computational restrictions.

2. Related Work

Our work is based on the "Progressive Growing of GANs for Improved Quality, Stability, and Variation" paper by Tero Karras, Timo Aila, Samuli Laine, and Jaako Lehtinen at NVIDIA (Karras et al., 2018). As mentioned above, Karras proposes a new training methodology for GANs that involves growing the generator and discriminator progressively by training on lower to higher resolutions. Although the work speeds the training up, it still involves a relatively high amount of computational power / training time (aka 8 Tesla V100 GPUs for 4 days).

A key related work, which Karras et al. and our work directly builds on, is the seminal "Generative Adversarial Nets" paper by Goodfellow et al., which propose GANs as a new framework for estimating generative models through an adversarial process (Goodfellow et al., 2014). Goodfellow et al. demonstrate how a generative model G and a discriminative model D can be situated in a minimax two-player game, where the generative model G can effectively capture a data distribution. However, GANs create images with limited variation and limited sharpness in larger resolutions.

It has long been an open area of research to generate high resolution images that include natural features and details. Wang et al. developed a new method which utilizes multiple discriminators at varying resolutions to guide the discriminator towards a more natural looking image (Wang et. al., 2017). However, this method is much more cumbersome compared to the method proposed by Karras et al., which uses a single generative network as opposed to a hierarchy of networks operating at different resolutions. The single network method proves to be a cleaner and more efficient approach.

Prior work in the realm of increasing the variation of the generated image distribution has also been conducted. One approach, taken by Salimans et al., 2016, was using a form of minibatch discrimination. By computing feature statistics and statistics for each example in a minibatch, the discriminator can learn additional mappings between minibatches and the computed statistics, thus enabling it to use the statistics in its decision making process. This subtly encourages the generator to mimic the training minibatches by attempting to create images with a similar set of statistics as those used in the minibatches. Since the minibatches are randomly sampled across the training data, this effectively increases the variety in the images generated. We expanded upon this technique in our methodology, and describe it in more detail in the Methods section.

3. Problem Formulation

Our first goal in this project is to generate semi-realistic images of celebrities of size 128 x 128 based on the CelebA-HQ corpus of celebrity photos in a more constrained environment. CelebA-HQ is a pre-processed version of the original CelebA dataset which provides a better input into our model. Our constrained environment consists of both less computing power and less time for training. We then seek to extend the results achieved by Karras et al. by experimenting with techniques for increasing quality and variation of images. To address the problems of the current approach, we adapt the neural net architecture and possible loss functions by exploring the literature. We aim to improve the baseline results at a 128 x 128 resolution from the Karras et al. implementation with respect to the metrics of SWD and IS, which we will delve into later.

Since we are aiming for lower resolutions, our computational time will also be lower. Karras et al. use a VGG architecture, with multiple 3 x 3 filters per resolution. Since the size of the architecture is constrained by the number of resolutions (there are $log_2 r$ layers in both the generator and discriminator, to create images of resolution rxr), in order to decrease the number of parameters, we could only decrease the number of convolutions within each resolution layer. However, we feel that this would decrease performance and thus only focus on making improvements that would beat the baseline results achieved by Karras et al. at the 128 x 128 resolution.

We aim to increase variation of the images generated by our GAN by exploring various techniques not currently utilized by Karras et al. Karras et al. opts to use a statistical method similar to the one outlined in the Related Works section, which computes the standard deviation of the features for each spatial dimension, for each minibatch, and creates a feature map relating the features for the minibatch to the computed standard deviation. The techniques we plan to use, which we will describe in Methods, include optimization techniques like a repelling regularizer, which encourages feature orthogonalization in image generation, different loss functions, residual blocks in the generator and discriminator, and experimenting with a richer set of statistics in the minibatch standard deviation layer included in the original implementation.

4. Methods

4.1. Preprocessing

In order to train our own model capable of generating celebrity images from scratch, we needed to create a CelebA-HQ dataset. The CelebA-HQ dataset contains all of the images from the CelebA dataset but alters them by centering images and performing various other preprocessing operations necessary for a good high-quality training dataset. The CelebA-HQ dataset is then composed of the best 30,000 images based on a frequency metric that favors images which are radially symmetric and penalizes blurry images.

We performed the same pre-processing steps as Karras et al. but created images at the 128 x 128 resolution we used for our experiments. However, creating the Celeba-HQ dataset was computationally challenging as it required us to save the data as Tensorflow Tf-records. Creating the CelebA-HQ dataset from the original CelebA dataset required over 10 hours of computation on our machine. Our machine had the following specifications optimized for a budget of 300 US dollars: 8 vCPUs with 30 GB, 1 Tesla K80 GPU, and 500 GB of additional memory.

4.2. Architecture & Design

We work with an initial resolution of 4 x 4, doubling the size of the resolution every n minibatches, for a minibatch size m (these are tunable hyperparameters). The minibatch size m decreases as the resolution increases due to memory considerations. The generator takes in an input from some latent space and applies a 4 x 4 convolution on the input to get a 4 x 4 representation of an image. It then applies a 3 x 3 convolutional layer to the input, passing the result through a leaky ReLU activation function. This is done 3 times per resolution. The discriminator mirrors the generator in its growing process, and takes in an image (either generated by the generator, or a set of training images) and runs it through several convolutional layers, followed by a fully connected layer. This process is shown in Figures 1 and 2.

Furthermore, both in the generator and discriminator, we include a residual block within each resolution layer. This is done before upsampling for the generator, and after downsampling for the discriminator. We feel as though this allows for more flexibility in terms of increasing the depth within each resolution layer. While we did not have the computational resources or time to effectively experiment with a greater number of convolutions in each layer and cascading residual blocks, we feel as though this would be a good point for future work. By adding residual blocks within each resolution layer, we feel as though the network could learn a greater number of features for each resolution layer.

Generator	Act.	Output shape
Latent vector	–	512 × 1 × 1
Conv 4 × 4	LReLU	512 × 4 × 4
Conv 3 × 3	LReLU	512 × 4 × 4
Upsample	–	512 × 8 × 8
Conv 3 × 3	LReLU	512 × 8 × 8
Conv 3 × 3	LReLU	512 × 8 × 8
Upsample	–	512 × 16 × 16
Conv 3 × 3	LReLU	512 × 16 × 16
Conv 3 × 3	LReLU	512 × 16 × 16
Upsample	–	512 × 32 × 32
Conv 3 × 3	LReLU	512 × 32 × 32
Conv 3 × 3	LReLU	512 × 32 × 32
Upsample	–	512 × 64 × 64
Conv 3 × 3	LReLU	256 × 64 × 64
Conv 3 × 3	LReLU	256 × 64 × 64
Upsample	–	256 × 128 × 128
Conv 3 × 3	LReLU	128 × 128 × 128
Conv 3 × 3	LReLU	128 × 128 × 128

Figure 1. Generator Architecture

Conv 3 × 3	LReLU	128 × 128 × 128
Conv 3 × 3	LReLU	256 × 128 × 128
Downsample	–	256 × 64 × 64
Conv 3 × 3	LReLU	256 × 64 × 64
Conv 3 × 3	LReLU	512 × 64 × 64
Downsample	–	512 × 32 × 32
Conv 3 × 3	LReLU	512 × 32 × 32
Conv 3 × 3	LReLU	512 × 32 × 32
Downsample	–	512 × 16 × 16
Conv 3 × 3	LReLU	512 × 16 × 16
Conv 3 × 3	LReLU	512 × 16 × 16
Downsample	–	512 × 8 × 8
Conv 3 × 3	LReLU	512 × 8 × 8
Conv 3 × 3	LReLU	512 × 8 × 8
Downsample	–	512 × 4 × 4
Minibatch stddev	–	513 × 4 × 4
Conv 3 × 3	LReLU	512 × 4 × 4
Conv 4 × 4	LReLU	512 × 1 × 1
Fully-connected	linear	1 × 1 × 1

Figure 2. Discriminator Architecture

Critically, we leave previous layers trainable. To incorporate images of a higher resolution into the GAN, we use nearest neighbor filtering to increase the resolution of the generated images in the generator. During the transition, the weights that are used for new layers are treated as those for a residual block. When a new layer is introduced, these weights are set to 1. At the end of the generator is a toRGB layer, which maps feature vector values to RGB values, using a direct 1 x 1 convolution. This toRGB is always the last layer of the generator. When incorporating higher resolution images into the discriminator, we first convert the RGB values to feature vector elements, using a 1 x 1 convolution. For the discriminator to handle the new, larger resolution image, average pooling is used to reduce the image size. When training the discriminator, the real images

are downscaled accordingly to match the dimensions of the current layer. This transition step is shown in Figure 3.

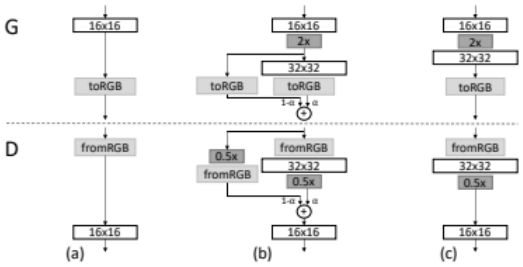

Figure 3. Architecture for transitioning between resolutions, in this case from a 16 x 16 resolution to 32 x 32 resolution in the generator and from a 32 x 32 resolution to 16 x 16 resolution in the discriminator

4.3. Repelling Regularizer

One technique that we will experiment with for greater variation in the images is a "repelling regularizer." Inspired by Zhao et al., 2017 in their work about energy based generative adversarial networks, this essentially adds another term to the objective function for the generator during optimization, in an attempt to encourage the generator to orthogonalize its feature vectors for the minibatch. The discriminator can be "regularized" by having the generator produce contrasting samples. This also keeps the model from only generated images from a subset of the training distribution. This added regularization term is defined by the following:

$$f_{RR}(S) = \frac{1}{N(N-1)} \sum_i \sum_{j \neq i} \left(\frac{S_i^T S_j}{||S_i|| ||S_j||} \right)^2 \quad (1)$$

The vectors S_i refer to the latent vectors initially randomly sampled by the generator in a minibatch. By encouraging orthogonalization of these latent vectors, the generator attempts to a more varied set of images.

4.4. Least Squares

We experimented with the least square loss function as well. We implement the least squares loss function as suggested by Mao et al. and defined below, which they argue would solve the issue of vanishing gradients for results on the correct side of the decision boundary but are still distant from the true data. This allows for increased stability when training and applies quicker updates to the generator. Unlike most loss functions used in GANs which don't apply any penalty for samples which are correctly classified yet lie far from the true distribution, the least square loss still penalizes those samples, thus moving the generated samples closer to the real data.

$$\min_D V_{\text{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}\left[(D(\boldsymbol{x}) - b)^2\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z})) - a)^2\right]$$
$$\min_G V_{\text{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z})) - c)^2\right],$$

The above equation utilizes the $a - b$ naming scheme for the discriminator where a and b are labels for the fake data and the real data respectively, while c is the label G wants to make D believe.

4.5. Additional Statistics

In order to tune the discriminator to the generated images more, we experimented with an increased set of minibatch statistics as well. The current implementation focuses on simply computing the standard deviation per minibatch and per spatial dimension, before replicating the results across the spatial dimensions. We add onto this by also using the third and fourth standardized moments, skewness and kurtosis, defined by the following equations:

$$Skew[X] = \frac{\mu^3}{\sigma^3} \quad (2)$$

$$Kurt[X] = \frac{\mu^4}{\sigma^4} \quad (3)$$

By mapping kurtosis, we can tune the discriminator to any outliers in the minibatch, as kurtosis focuses on the tail ends of a distribution. By mapping skewness, we do the same with the shape of the minibatch distribution. We feel as though increasing the set of statistics will encourage the generator to more closely model the distribution, as the discriminator can use these additional metrics when attempting to differentiate between the generated and training distributions. This statistics layer is added to the end of the discriminator, before the fully connected layer.

4.6. Wasserstein Distance

Lastly, we experiment with the Wasserstein distance as our distance metric comparing the generated and training distributions. After some background research, we have found that Wasserstein distance tends to be the most stable when training. Wasserstein distance is defined as the following (Arjovsky et al., 2017):

$$W(\mathbb{P}_r, \mathbb{P}_g) = inf_{\gamma \in \prod(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||] \quad (4)$$

The Wasserstein distance can be interpreted as the amount of "mass" that must be transported from $x \in r$ to $y \in g$ to transform the distribution $\mathbb{P}_r$ into $\mathbb{P}_g$, and thus represents the cost of making these distributions equal. However,

while the original Wasserstein GAN (WGAN) implementation makes use of weight clipping to avoid vanishing gradients, we used a version that utilizes a penalty on the norm of the gradients for the discriminator. This was found to yield better quality image generation.

4.7. Evaluation Criteria Methods

For a qualitative measure, we will attempt latent space interpolation in order to test the smoothness of transitions between samples.

For quantitative evaluation metrics, we choose Inception Score (IS) and Sliced Wasserstein Distance (SWD), discussed below.

4.7.1 Inception Score

Inception score is often used to evaluate the quality of image generative models, and correlates well with human perceptions of image quality. The inception score of a generative model G is defined by the following:

$$IS(G) = exp(_{x\sim p_g} D_{KL}(p(y|x)||p(y))) \tag{5}$$

$x \sim p_g$ refers to a sampled image from p_g, the generated distribution, $p(y|x)$ refers to the class conditional distribution, $p(y)$ refers to the distribution of the generated images, and D_{KL} refers to the KL-divergence between $p(y|x)$ and $p(y)$. If $p(y|x)$ is high entropy, then the images should be sharp and the objects clearly identifiable. In our case, this would indicate a better quality image of a newly synthesized face. If $p(y)$ is low entropy, then the model generates a varied set of images. Thus, the ideal model would maximize the KL-divergence between $p(y|x)$ and $p(y)$, and therefore would aim to increase inception score.

4.7.2 Sliced Wasserstein Distance

The Sliced Wasserstein Distance (SWD) is a variation of Wasserstein Distance that generalizes better to higher levels. Given two distributions, it uses linear projections to obtain a family of 1-dimensional marginal distributions for both the input distributions. The distance is then computed as a function of these marginal distributions. The full formulation for SWD is shown below:

$$SW_p(I_x, I_y) = (\int_{\S^{d-1}} W_p^p(RI_x(.,\theta), (RI_y(.,\theta))d\theta)^{\frac{1}{p}} \tag{6}$$

As SWD is an approximation for Wasserstein distance, the generative model attempts to minimize SWD between its distribution and that of the training set. A low SWD thus indicates a high level of generative performance, as well as a greater amount of variation in the synthesized images.

5. Results

5.1. Metrics

After implementing the various techniques and architectural changes detailed in the Methods section above, we then evaluated the efficacy of these techniques and changes with respect to the metrics of Sliced Wasserstein Distance (SWD) and Inception Score (IS) defined above.

Table 1 below indicates the results after we calculated the metrics for our various experiments. The first column, Architecture, refers to the results for the images generated by various architectures that we experimented with, the details of which we fleshed out in the Methods. Reals and Reals2 are real celebrity images, which we compute metrics on so that we could compare the metrics on our generated images to the metrics on a dataset of real celebrity images. Res Block G and Res Block D refer to additional residual blocks in the generator and additional residual blocks in the discriminator respectively. RR stands for the repelling regularizer. Least squares loss refers to the different loss function we experimented with, as opposed to the Wasserstein distance. We use Wasserstein distance in our other architectures, as it optimally performed across both SWD and IS in a range of experiments. Next, in the columns, IS mean and IS std refer to the mean of the inception score and the standard deviation of that inception score respectively.

Note that we generate the images from a particular architecture after training the network corresponding to that architecture through 1.462 million images. This standardizes our results, enabling us to make comparisons across metrics corresponding to different architectures.

Table 1. SWD and IS Scores across Architectures

Architectures	**SWD**	**IS_mean**	**IS_std**
Reals	0	3.5783	0.0759
Reals2	1.65	3.5991	0.0621
Baseline	109.2181	1.118	0.0027
Res Block G	105.9239	1.1132	0.0009
Res Block D	108.1729	1.1145	0.0022
RR	113.236	1.126	0.0017
RR, Res Block G	103.6598	1.114	0.0021
RR, Res Blocks G and D	104.2972	1.1161	0.0021
Least Squares Loss	109.6703	1.1175	0.0017

Table 2 below shows the percent differences in IS and SWD, for the different tested architectures. For example, for RR,

Res Block G, the percent change from the baseline of -5.09% means that the RR, Res Block G architecture decreased SWD by approximately 5% in comparison to the baseline model by Karras et al.

Table 2. Percent Change from Baseline across Architectures

Architectures	**% Δ SWD**	**% Δ IS**
Res Block G	-3.02	-0.43
Res Block D	-0.096	-0.31
RR	3.68	0.72
RR, Res Block G	-5.09	-0.36
RR, Res Blocks G and D	-4.51	-0.17
Least Square Loss	0.41	-0.04

As can be seen from the results in Table 1 and Table 2, we discovered a tradeoff between IS and SWD across different architectures. Our RR architecture, corresponding to the repelling regularizer, increased the IS from 1.118 to 1.126, a statistically significant amount as the change is outside the standard deviation of both inception scores. Thus, we were able to improve the inception score, but this change was associated with an increase in SWD. Similarly, our RR, Res Block G architecture decreased SWD by 5.09%, a non-negligible amount. So, we could decrease the SWD, but such change was associated with a decrease in the IS. This set of experimentation sheds light on the dynamic between architectural changes and metrics like SWD / IS when progressively growing GANs for image generation.

Ultimately, our final model attempted to strike the optimal balance across both metrics. Our final model, which used the RR, Res Blocks G and D architecture, decreased SWD with no statistically significant changes to IS from the baseline. The final architecture decreased SWD from 109.22 to 104.30, a 4.51% improvement in the SWD score. Although the mean of the IS decreased by 0.16%, the change in IS was not statistically significant, as it was within 1 standard deviation of the mean of the IS of the final architecture as well as the mean of the IS of the baseline.

In other words, our final architecture proved promising, as it improved SWD without any statistically significant changes to IS.

5.2. Generated Images

After experimenting with various architectures, as can be seen above, we then selected our best performing architecture described above and trained it on 128 x 128 images for approximately 13 hours. At various steps along this training process, we generated fake images of 128 x 128 resolution from our model to demonstrate both the stability of training as well as the results obtainable by our model. Figures 4 - 11 show the faces generated by our model at different points in the training process. Figure 11 shows relatively high quality facial images after 13 hours of training and iterating through 3.146 million images. Figure 12 shows the facial images generated by Karras et al.'s pre-trained model after processing the same amount of images as our model in Figure 11 but in 1024 x 1024 resolution. Although we cannot make direct visual comparisons as the images are generated in different resolutions, we include Figure 12 to show that at this stage, even Karras et al's model was not generating the perfectly photorealistic images indicated in their final results.

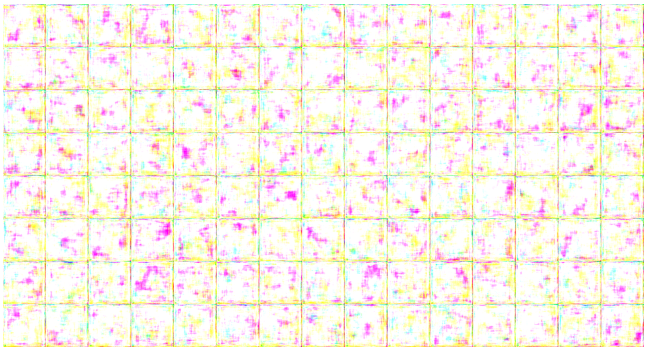

Figure 4. Collection of 128 x 128 celebrity images generated via our model after iterating through 0 images (aka the first set of 'guesses' from the generator)

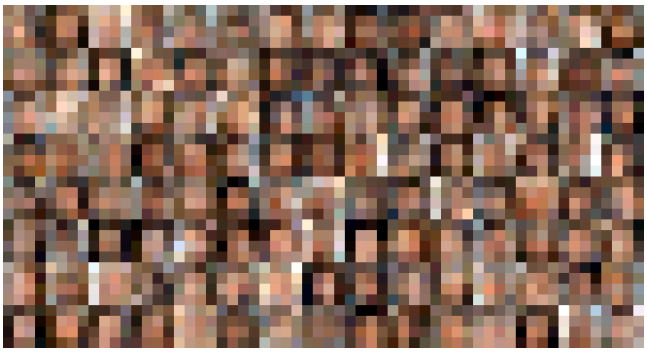

Figure 5. Collection of 128 x 128 celebrity images generated via our model after iterating through 480,000 images in approximately 11 minutes

Figure 6. Collection of 128 x 128 celebrity images generated via our model after iterating through 901,000 images in approximately 52 minutes

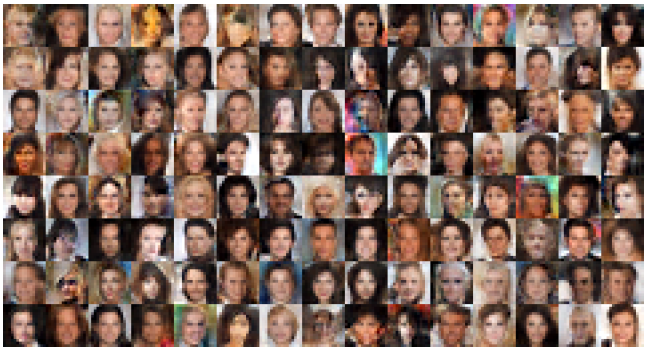

Figure 9. Collection of 128 x 128 celebrity images generated via our model after iterating through 2.705 million images in approximately 8 hours

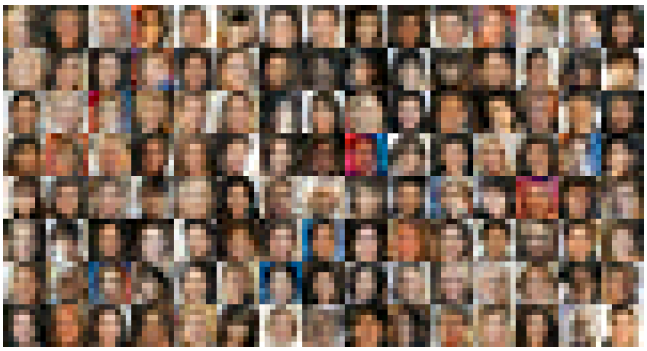

Figure 7. Collection of 128 x 128 celebrity images generated via our model after iterating through 1.462 million images in approximately 2 hours

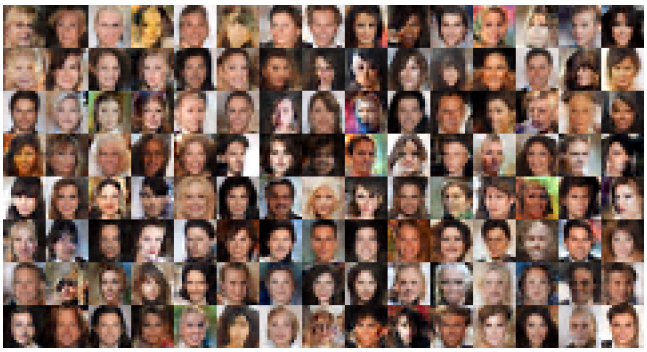

Figure 10. Collection of 128 x 128 celebrity images generated via our model after iterating through 2.946 million images in approximately 9 hours 40 minutes

Figure 8. Collection of 128 x 128 celebrity images generated via our model after iterating through 2.104 million images in approximately 4.5 hours

Figure 11. Collection of 128 x 128 celebrity images generated via our model after iterating through 3.146 million images in approximately 13 hours

Figure 12. Collection of 1024 x 1024 celebrity images generated via Karras et al's pretrained model after iterating through 3.146 million images

6. Future Work

As discussed above in the Architecture section, possible future work could include modifying the architecture to include cascading residual blocks to help the network learn more features than it did with our architecture. Another technique for improved variation that we would like to work with is an optimization technique called "unrolling the discriminator", inspired by Metz et al., 2016. This essentially involves making several gradient descent updates for the discriminator for each update of the generator, and using these "locally optimal" discriminator parameters for the generator's objective function when updating the generator. This allows the generator to get an idea of how the discriminator would react optimally at a given point in training, and forces it to adapt accordingly.

7. GitHub Repository

Our Github repository can be found at `https://github.com/Lstilwell/Progressive-GAN`.

8. References

T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196, 2017

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. In NIPS, 2014.

Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional GANs. CoRR, abs/1711.11585, 2017.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In NIPS, 2016.

Junbo Jake Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. In ICLR, 2017'

Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, and Zhen Wang. Least squares generative adversarial networks. CoRR, abs/1611.04076, 2016.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. CoRR, abs/1611.02163, 2016

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. arXiv preprint arXiv:1701.07875, 2017.