Final Report of DCGAN, Wasserstein GAN and Self-Attention GAN

Lifan Wang
New York University
lw2435@nyu.edu

Kuan Chen
New York University
kc3422@nyu.edu

Guanhua Chen
New York University
gc2229@nyu.edu

Yueping Wang
New York University
yw1344@nyu.edu

1. Introduction

Recent research and development in General Adversarial Networks (GANs) [3] [2] [14] has enabled a profusion of well-performing machine learning models for generative tasks, such as image synthesis, semantic segmentation and text synthesis. In short, the objective of GANs is to train the generator and the discriminator networks simultaneously in a competitive game: the generator takes in a noise vector and outputs a sample to fool the discriminator, and the discriminator tries to distinguish such sample from real data and return training information to the generator.

However, training GANs is notoriously tricky, and numerous research papers have been dedicated to solving challenges such as vanishing gradient of the generator, model collapse and non-convergence. In our work, we conducted research on three recent improvements in GANs, namely Deep Convolutional GAN [14], Wasserstein GAN [2] and Self-Attention GAN [17], in addition to the vanilla GAN introduced in 2014 [3].

We evaluated these four types of GANs through both theoretical analysis and empirical training against a variety of image datasets, including MNIST handwritten numbers, CIFAR10 and anime faces.

2. Related Work And References

2.1. Early General Adversarial Networks

The first GAN [3] is introduced in 2014 by Ian Goodfellow. It uses fully connected layers as its building blocks, and tries to maximize the similarity of probability distribution between sample data and real data. KL-divergence and JS-divergence are frequently used as the loss function. The early GAN models are know to be very unstable to train. For example, gradient vanishing occurs when the discriminator learns too fast (original improvement by the author is to change minimizing $log(1 - D(G(z)))$ to maximizing $logD(G(z))$ from the equation), and model collapse occurs when the gradient of loss function isn't differentiable.

2.2. Wasserstein GAN

The work of Wasserstein GAN [2] introduces a different way to measure the distance between sample date and real data distribution.

In WGAN, the Wasserstein distance (Earth-Mover distance) replaces JS-divergence.

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

Intuitively, the Wasserstein distance measures how much it takes to shift from one distribution to another. Therefore the loss function of WGAN is always continuous, and the generator can learn even when the two distributions don't overlap. To ensure that the EM loss is K-Lipschitz continuous, weight clipping is applied (w is bounded by some constant K):

$$\max_{\omega \in \mathcal{W}} \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{x \sim p(z)}[f_w(g_\theta(z))]$$
$$\leq \sup_{||f||_L \leq K} \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{x \sim p(z)}[f_w(g_\theta(z))] = K \cdot W(\mathbb{P}_r, \mathbb{P}_\theta)$$

And because Wasserstein distance is not probability distribution, and does the job similar to the regression step, sigmoid function is no longer needed in WGAN.

2.3. Deep Convolutional GAN

Deep Convolutional GAN uses all-convolutional nets [14] as its architecture, and can be trained to generate high-resolution images. DCGAN assumes the that the inputs are images, and takes into account the pixel values in the neighborhood. Hence, ConvNets outperform regular neural nets in the scale of full images. Specifically, the neurons in ConvNet are arranged in 3 dimensions by width, height and depth (the third dimension of activation volume). This architectural "constraints" make the model more stable to train than vanilla GAN.

2.4. Self Attention GAN

Self-Attention GAN [17] focus on modeling long-distance relationships, unlike the earlier GANs that focus on

local neighborhoods. The goal of SAGAN is to apply self-attention [15], to image synthesis via non-local neural network [16]. While self-attention has shown improvements in language translation, its application in image generation is relatively recent.

3. Problem formulation

Here are our problem formulations: What's the difference between these GANs in generating images? What algorithm and technologies are used in each method? How to implement these GANs and conduct data training? What is the difference between the architectures of these methods? How to train the Generator and Discriminator for each GAN? How to optimize the method and improve the picture quality? Why the final results are different?

Our goal is to compare the performances of the four GAN models in image synthesis. First, we will design and implement the different neural network architectures for vanilla GAN, WGAN, DCGAN and SAGAN. We will then run training sessions to assess the output images with qualitative and quantitative criterion, such as quality of image, losses and training time. After that, we will reflect on the training process and outcome, and make adjustments and improvements to the GANs architectures and functions.

Through training the GANs models against multiple image datasets, we hope to analyze how the different theoretical improvements proposed in the GANs papers actually perform in practice.

4. Methods and Algorithms

4.1. Applying DCGAN for Generator & Discriminator

Originally for WGAN we can use fully connected layers for the Generator, but in order to create a more intuitive architecture, we modifies the Generator by putting in more than 4 layers of Deconvolution modules. We also balanced the Discriminator with the Generator by putting in equal amounts of layers in it. Due to some weakness of unstable training (gradient diminishing / gradient explosion / covariance shift), we apply batch normalization into each layer right before each activation function.

4.2. Designing WGAN-GP loss function

The original GAN's loss function is weak at some gradient singularity point, so we introduce the WGAN-GP loss function, that is Earth-Mover Distance(Wasserstein metric). But its not easy to calculate the Wasserstein metric. We need use K-Lipschitz function to transfer it to the problem:

$$\max_{\omega \in \mathcal{W}} \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{x \sim p(z)}[f_w(g_\theta(z))]$$
$$\leq \sup_{||f||_L \leq K} \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{x \sim p(z)}[f_w(g_\theta(z))] = K \cdot W(\mathbb{P}_r, \mathbb{P}_\theta)$$

Then it can be proved that this process is principled under the optimality assumption. Hence the loss of discriminator and generator can be calculated.

$$\bigtriangledown_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z \sim p(z)}[\bigtriangledown_\theta f(g_\theta(z))]$$

4.3. Embedding Self-Attention module

SAGAN uses hinge loss to train the network:

$$L_D = -\mathbb{E}_{(x,y) \sim P_{data}}[min(0, -1 + D(x, y))]$$
$$- \mathbb{E}_{z \sim p_z,) \sim P_{data}}[min(0, -1 - D(G(z), y))]$$
$$L_G = -\mathbb{E}_{z \sim p_z,) \sim P_{data}} D(G(z), y)$$

The PAPER shows there are WGAN-GP architecture used for image generation isn't good at capturing global features because of the locality property for its Convolution/Deconvolution layers. So we add self attention module into it.

Basically the self attention module divides the original convolutional feature map into three sub feature-maps, and then computes the covariance matrix for each two pixels in the feature map. This covariance matrix is intuitively called Attention, simply because each row of the matrix represents how each pixel views its relation towards the whole picture. After processing with the Attention matix ,the original input (convolutional feature map) would be able to customize an attention mask for each pixel. Now we create Self attention result by applying the each pixels attention mask onto the feature map and by shifting the original pixel feature to its attention fields. It would achieve similar performance as fully connected layer ,but its intuitive architecture outweighs fully connected layer thus computationally more efficient.

Here are few details for Self-Attention GAN:

1. They used this self-attention layer in both the generator and discriminator
2. They applied spectral normalization to the weights in both generator and discriminator, unlike the previous paper which only normalizes the discriminator weights. They set the spectral norm to 1 to constrain the Lipschitz constant of the weights. Its just used for controlling the gradients. This spectral normalization idea was first introduced by Miyato et. al.
3. They used a two-timescale update rule (TTUR) which is simply using different learning rate for both discriminator and generator

4.4. GAN's Training Algorithm

Then we will follow Wasserstein Generative Adversarial Network algorithm[2] to build our model. Here is our brief algorithm:

For _ in enough iteration

1. we train D for 1 time, using images coming from the Real and Synthesis data. And maximize loss function
2. we train G for 1 time ,using random noise and compare it with Real data, then minimize loss function
3. if loss function == 0, then end iteration.

5. Technical Depth and Innovation

5.1. Technical Innovation

- Replaced the traditional NN generator by CNN(ResNet) generator, applied batchnormalization at each level.
- Simulated a image scaling stage by using a linear module – a fully connected layer(without batch normalization) , at the entrance/exit of generator/discriminator.
- Implemented Relaxed-WGANs by applying gradient penalty into traditional WGANs.
- Analyzed the training speed, performance of DCGAN, WGAN, Relaxed WGAN, and compared their loss function
- Applied Relaxed GAN to various dataset of mnist, cifar10, customized dataset.

5.2. Architecture Innovation

- We applied Self-Attention architecture into the previous WGAN-GP model to create a SAGAN.
- We fed the model with Mnist, Cifar10, Celeba, comic faces dataset, and achieved decent results.
- We applied DCGANs generator architecture into traditional WGAN model and modified the loss function to create a WGAN-GP model.

5.3. Training Innovation

- We analyzed the training speed, performance of DCGAN, WGAN-GP, SAGAN, and compared their loss function.
- We modified the parameters(learning rates, batch size, and exponential decay rate of ADAM optimizer) and balanced the architecture differences between Generator & Discriminator.
- For some training set, we found that our modified parameters produce better results than parameters proposed in the original SAGAN paper: We compared TWO types of image rescaling methods: (neuron network+refactoring & opencv resizing function) and find neuron network method has more interesting training results

6. Architecture and Design

The core to the DCGAN architecture uses a standard CNN architecture on the discriminative model. For the generator, convolutions are replaced with upconvolutions, so the representation at each layer of the generator is actually successively larger, as it mapes from a low-dimensional latent vector onto a high-Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).

Use batch normalization in both the generator and the discriminator.

Remove fully connected hidden layers for deeper architectures.

Use ReLU activation in generator for all layers except for the output, which uses Tanh.

Use LeakyReLU activation in the discriminator for all layers.dimensional image.

The architecture of our project is "$deconv$" $\Rightarrow$ "$conv$", where generator is deconv and discriminator is conv. Take Cifar10 as an example, here is our Architecture graph:

6.1. GAN's Generator

Here is our design of generator:

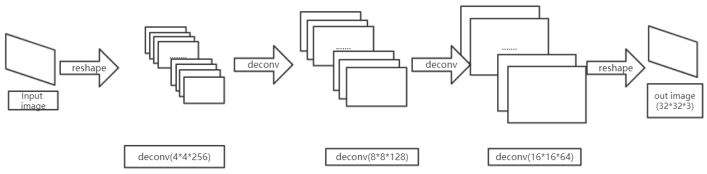

Figure 1. Generator Design

generator: deconv [batch,1024] $\Rightarrow$ [batch,4096] $\Rightarrow$ [batch,4,4,256] $\Rightarrow$ batch,8,8,128] $\Rightarrow$ [batch,16,16,64] $\Rightarrow$ [batch,16,16,32] $\Rightarrow$ [batch,32,32,3]

It reshapes the input image to a 4096 length array. Then after three times deconv, it becomes a 16*16*32 layer. Finally the layer is reshaped to a 32*32*3 output image.

6.2. GAN's Discriminator

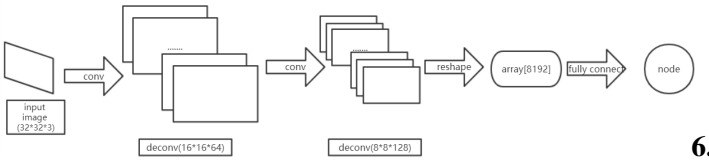

Figure 2. Discriminator Design

Discriminator: conv [batch,32,32,3] ⇒ [batch,16,16,64] ⇒ [batch,8,8,128] ⇒ [batch,8192] ⇒ [batch,1]

After three times conv, it becomes a 8*8*128 layer. Then the layer is reshaped to a 8192 array then fully connected to a tag node.

Three types of Loss Functions : Relax-WGAN loss (EM distance+gradient penalty), WGAN loss(EM distance) , DCGAN loss (JS divergence)

6.3. Self Attention Layer

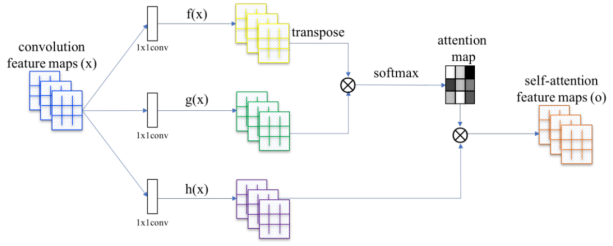

Figure 3. Self-Attention Framework[17]

For each convolutional layer, we refine each spatial location output with an extra term o computed by the self-attention mechanism.

$$y_i = \gamma o_i + x_i$$

where x is the original layer output and y is the new output.

we apply the self-attention mechanism to each convolutional layers The self-attention composes of:

- Compute the attention map β.
- Compute the self-attention output.

On the left of the below image, we get our feature maps from the previous convolutional layer. We first pass the feature map through three 1x1 convolutions separately. We name the three filters f, g and h. Next, we multiple x with W_h (model parameters to be trained also) and merge it with the attention map β to generate the self-attention feature map output(o). The final output of this convolutional layer is:

$$y_i = \gamma o_i + x_i$$

6.4. Parameter

We use hinge loss and wgan-gp loss as loss function. Generator's learning rate is 0.0001, Discriminator's learning rate is 0.0004. $G_\beta = 0.0, d_\beta = 0.9$.

7. Preliminary Results

7.1. Github Repo and Training Abstract

`https://github.com/StarryBar/image_synthesis-WGAN-`

We train DCGAN, WGAN and SAGAN on cifar10 [9], Anime faces dataset [11] and CelebA [12]. Then compare the result and loss of them. The codes and jupyter notebook can be found in the Github repo. Here is some parameters of our training: learning rate = 0.001 β_1 = 0.5

dataset	picture size
cifar10	32*32*3
Anime faces	96*96*3
CelebA	178*218*3

7.2. Cifar10-deer

7.2.1 DCGAN

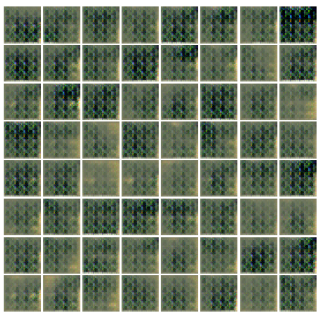

Figure 4. DCGAN on cifar10-deer after 500 iteration

Figure 5. DCGAN on cifar10-deer after 2500 iteration

7.2.2 WGAN

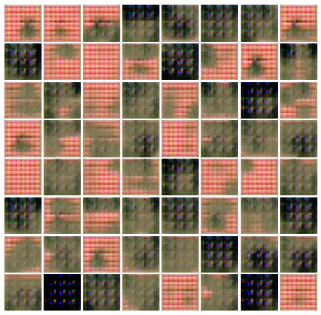

Figure 6. WGAN on cifar10-deer after 1000 iteration

Figure 7. WGAN on cifar10-deer after 4000 iteration

Figure 8. WGAN on cifar10-deer after 60000 iteration

Figure 9. Generator Loss of WGAN for cifar10 dataset

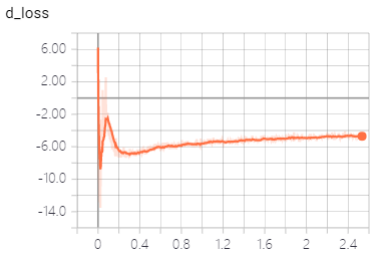

Figure 10. Discriminator Loss of WGAN for cifar10 dataset

From the loss figures, we can see that both generator and discriminator are converged to a small range. And the result pictures seem good. After long enough iteration, we can figure that the result is deer easily.

7.3. Anime Faces

7.3.1 DCGAN

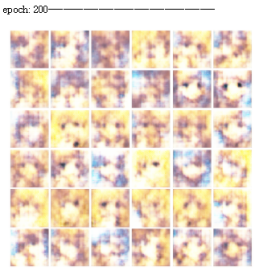

Figure 11. DCGAN on Anime Faces dataset after 1000 iteration

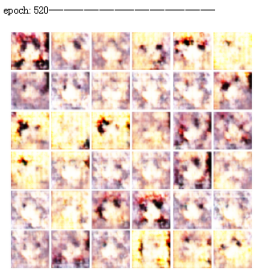

Figure 12. DCGAN on Anime Faces dataset after 3000 iteration

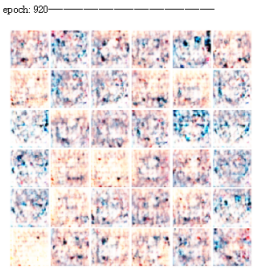

Figure 13. DCGAN on Anime Faces dataset after 5000 iteration

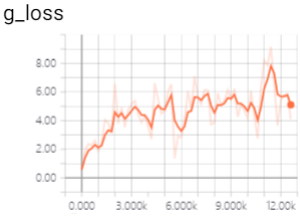

Figure 14. Generator Loss of DCGAN on Anime Faces dataset

Figure 15. Discriminator Loss of DCGAN on Anime Faces dataset

When training with the Anime Faces dataset, Our DCGAN seems not well. The discriminator loss converges quickly while generator is unstable. And after 10000 iterations, it will explode. But the WGAN uses almost same structure, but WGAN perform much better.

7.3.2 WGAN

Figure 16. WGAN on Anime Faces dataset after 1000 iteration

Figure 17. WGAN on Anime Faces dataset after 3000 iteration

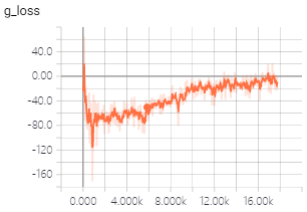

Figure 18. Generator Loss of WGAN on Anime faces dataset

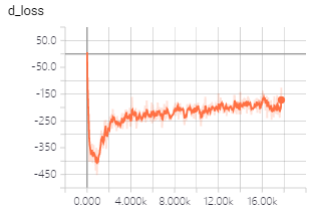

Figure 19. Discriminator Loss of WGAN on Anime faces dataset

Compared with DCGAN, WGAN's generator loss converges quickly and discriminator is also going to good direction.

7.3.3 SAGAN

Figure 20. SAGAN on Anime Faces dataset after 1000 iteration

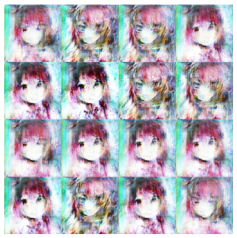

Figure 21. SAGAN on Anime Faces dataset after 3000 iteration

Figure 22. SAGAN on Anime Faces dataset after 50000 iteration

Figure 23. SAGAN on Anime Faces dataset after 80000 iteration

Figure 24. Generator Loss of SAGAN on Anime faces dataset

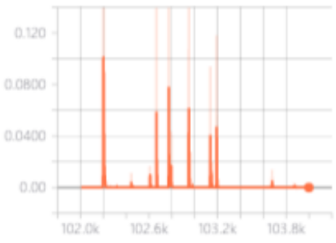

Figure 25. Discriminator Loss of SAGAN on Anime faces dataset

When training with the Anime Faces dataset, SAGAN is able to generate face-like images very quickly, but the problem is that the good performance is unstable. Indeed, as the trainings increase, we found that the image quality and loss of generator oscillate significantly.

7.4. SAGAN on CelebA

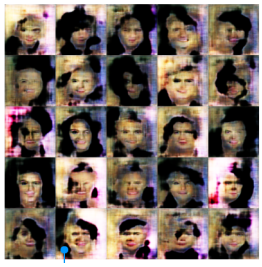

Figure 26. SAGAN on CelebA dataset after 3000 iteration

Figure 27. SAGAN on CelebA dataset after 5000 iteration

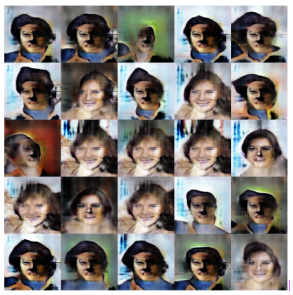

Figure 28. SAGAN on CelebA dataset after 50000 iteration

Figure 29. SAGAN on CelebA dataset after 80000 iteration

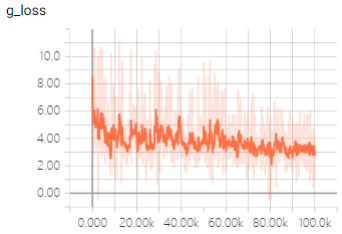

Figure 30. Generator Loss of SAGAN on Anime faces dataset

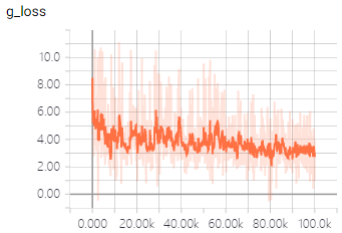

Figure 31. Discriminator Loss of SAGAN on Anime faces dataset

Both the plot and training samples show steady improvements in the generator's ability to produce realistic human faces.

Nonetheless, we do observe a discrepancy of image quality between the face area, and the hair style together with the background. The proportion and resolution within the facial area are very stable and realistic, and the same batch tends to generate consistent facial features. However, the hair contour and background varies quite a bit even for the same batch, and there is a lack of correlation between the face and the hair.

While SAGAN propose benefits of learning long-distance influences, in this Celebrity Faces example, we see that there is still a very pragmatic need for strengthening adjacent pixel correlations.

7.5. Discussion

1. Normally WGAN only need to use fully connected layers for generator, but here we use complicated structure in DCGAN for the generator to achieve better results.

2. Cifar10 dataset is not good for image generation. Firstly, cifar10 dataset has ten labels, each label stands for a different type of image. We cannot put all of the types into training stage. So we choose only one type for training. Secondly, Even if we choose only one label for training, the background for the image varies a lot. So we could not achieve similar performance as mnist dataset whose background is only blackboard.

3. For the comic faces data set, there are 50,000 images, but we only choose 128 images so as to train in a fast way. Unluckily, we find the relevant results to be of low variance.

4. DCGAN requires a strongly restricted architecture for both genertor and the discriminator. Our DCGAN is not so powerful enough so that after some time of training, the model collapse and the loss never changes again.

5. There are two ways to rescale the 96*96 image to feed it into the 32*32 entrance of the generator. One is use OPENCV resize function. Another way is to add another neuron network into the generator. We apply the second method and find that we cannot add batch normalization after the fully connected layer, for the rescaling function is only a linear function but batch normalization destroys the linear model.

6. For GAN models trained with ImageNet, they are good at classes with a lot of texture (landscape, sky) but perform much worse for structure. For example, GAN may render the fur of a dog nicely but fail badly for the dogs legs. While convolutional filters are good at exploring spatial locality information, the receptive fields may not be large enough to cover larger structures. We can increase the filter size or the depth of the deep network but this will make GANs even harder to train. Alternatively, we can apply the attention concept.

References

[1] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. *CoRR*, abs/1701.04862, 2017.

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. 70:214–223, 06–11 Aug 2017.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. pages 2672–2680, 2014.

[4] I. J. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2016.

[5] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In *NIPS*, 2017.

[6] X. Guo, J. Hong, T. Lin, and N. Yang. Relaxed wasserstein with applications to gans. *CoRR*, abs/1705.07164, 2017.

[7] H. Huang, P. S. Yu, and C. Wang. An introduction to image synthesis with generative adversarial nets. *CoRR*, abs/1803.04469, 2018.

[8] J. Kim. Self-attention-gan-tensorflow. `https://github.com/taki0112/Self-Attention-GAN-Tensorflow/`. Accessed Nov 20, 2018.

[9] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research).

[10] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[11] J. Lei. Anime faces dataset. `https://github.com/jayleicn/animeGAN/`. Accessed Nov 10, 2018.

[12] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[13] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018.

[14] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017.

[16] X. Wang, R. B. Girshick, A. Gupta, and K. He. Non-local neural networks. *CoRR*, abs/1711.07971, 2017.

[17] H. Zhang, I. J. Goodfellow, D. N. Metaxas, and A. Odena. Self-attention generative adversarial networks. *CoRR*, abs/1805.08318, 2018.