Anatomizing Generative Adversarial Networks

Arnab Sinha

Abstract

Here, we investigate three variants of generative adversarial networks (GANs): vanilla GANs, super-resolution GANs, and conditional GANs. First, we revisit their mathematical formulations. Then, we examine the abilities of vanilla GANs and super-resolution GANs on low-dimensional synthetically generated datasets. For this, we realize generators and discriminators using multi-layer perceptrons (MLPs). We also empirically compare the performance of GANs for super-resolution of data with that of MLPs, when we train MLPs for the same task, considering the associated learning as a regression task. Our limited experiment confirms the superiority of GANs for this task.

Index Terms

Adversarial networks, Discriminators, Generators.

1 Introduction

GENERATIVE adversarial networks (GANs) [1] consist of two neural networks: a generative network and a discriminator. The generator creates counterfeit data while the discriminator, an adversary to the generator, detects the fake ones. The training of a GAN continues until an equilibrium when the discriminator fails to discriminate between fake and real data. In the last decade, GANs have proved themselves to be extremely useful in several tasks including the generation of artificial facial images [2], [3], the creation of super-resolution images from low-resolution images [4], and image translation [5].

The objective of this work is to revisit the working principles of three variants of GANs: (i) vanilla GANs (vGANs) [1] (ii) super-resolution GANs (SRGANs) [4] and (iii) conditional GANs (cGANs) [5]. We experimentally validate the philosophies of both vGANs and SRGANs. Note that, the works in [1] and [4] use convolutional neural networks (ConvNets) to realize the generators and the discriminators. Here, we realize the generators and the discriminators using multi-layer perceptrons (MLPs). Moreover, researchers in [1] and [4] examined the abilities of GANs on real-world image datasets. On the contrary, here we use real-valued datasets sampled from multi-variate Gaussian distributions. These two differences make this investigation distinct from [1] and [4]. Furthermore, we look into three different GANs, which helps us to realize the distinctive nature of these architectures as well as the evolution of GANs. We also experimentally compare GANs with MLPs concerning the super-resolution task.

A technical report submitted for the completion of an internship at the Indian Institute of Information Technology Guwahati in Summer 2020.

- *Arnab Sinha is a student at The LNM Institute of Information Technology, Jaipur, India, 302031.*
  E-mail: 17ucs034@lnmiit.ac.in

2 Methods

2.1 Vanilla GANs (vGANs) [1]

Let $\mathbf{x} \in \mathbb{R}^d$ be d-dimensional data point drawn from the distribution $p_{\mathbf{x}}$, i.e. $\mathbf{x} \sim p_{\mathbf{x}}$. Let $\mathbf{z} \in \mathbb{R}^m$ be a m-dimensional noise vector drawn from the distribution $p_{\mathbf{z}}$, i.e. $\mathbf{z} \sim p_{\mathbf{z}}$. The aim of vGANs is to construct a mapping, $G(\mathbf{z}; \theta_G) \rightarrow \mathbb{R}^d$, such that, $G(\mathbf{z}; \theta_G) \sim p_{\mathbf{x}}$ holds. Here, $G(\mathbf{z}; \theta_G)$ is a differentiable *generator* function that can be realized using a multilayer perceptron (MLP) with the set of parameters (weights) θ_G. As an adversary to $G(\mathbf{z}; \theta_G)$, a second differentiable function $D(\cdot; \theta_D) \rightarrow \mathbb{R}$, called the *discriminator*, is used, where θ_D is a set of parameters (weights). $D(\cdot; \theta_D)$ is indeed a binary classifier that can also be realized using a MLP. $D(\cdot; \theta_D)$ is supposed to discriminate the patterns that are generated by $G(\mathbf{z}; \theta_G)$ from the actual data patterns. Therefore, to train $D(\cdot; \theta_D)$, the desired output for $D(\mathbf{x}; \theta_D)$ is considered to be 1, whereas, the desired output for $D(G(\mathbf{z}; \theta_G); \theta_D)$ is considered to be 0. vGANs trains the generator and the discriminator by optimizing the following problem:

$$\min_{\theta_G} \max_{\theta_D} \; \mathbb{E}_{\mathbf{x} \sim p_{\mathbf{x}}} \log \left[D\left(\mathbf{x}; \theta_D\right)\right] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}} \log \left[1 - D\left(G\left(\mathbf{z}; \theta_G\right); \theta_D\right)\right] \tag{1}$$

This learning process is iterative, where, for every k iterations of discriminator training, 1 iteration of generator training is implemented. This helps to ensure the optimality of the discriminator and the slow training of the generator [1]. However, maximizing $\log\left[D\left(G\left(\mathbf{z}; \theta_G\right); \theta_D\right)\right]$ rather than minimizing $\log\left[1 - D\left(G\left(\mathbf{z}; \theta_G\right); \theta_D\right)\right]$ might be more beneficial [1].

2.2 Super Resolution GANs (SRGANs) [4]

SRGANs are used for the construction of high-dimensional representation of a low resolution data point. Let $\mathbf{x}^{\mathrm{LD}} \in \mathbb{R}^L$ be a L-dimensional data point drawn from the distribution $p_{\mathbf{x}^{\mathrm{LD}}}$, i.e., $\mathbf{x}^{\mathrm{LD}} \sim p_{\mathbf{x}^{\mathrm{LD}}}$. Let $\mathbf{x}^{\mathrm{HD}} \in \mathbb{R}^H$ be a H-dimensional data point drawn from the distribution $p_{\mathbf{x}^{\mathrm{HD}}}$, i.e., $\mathbf{x}^{\mathrm{HD}} \sim p_{\mathbf{x}^{\mathrm{HD}}}$ and $H > L$. Let $(\mathbf{x}^{\mathrm{LD}}, \mathbf{x}^{\mathrm{HD}})$ form a tuple of paired data. SRGANs aims to construct a mapping $G(\mathbf{x}^{\mathrm{LD}}; \theta_G) = \mathbf{x}^{\mathrm{HD}}$. Here, θ_G is the parameters of the generator $G(\mathbf{x}^{\mathrm{LD}}; \theta_G)$. SRGANs' primary application is to form super-resolution images from low-resolution images. Further, an adversary in the form of a discriminator $D(\cdot; \theta_D) \rightarrow \mathbb{R}$ is used. Similar to vGANs, here $D(\cdot; \theta_D)$ is trained with the aim of classifying $D(\mathbf{x}^{\mathrm{HD}}; \theta_D)$ as real data with the target value 1 and $D(G(\mathbf{x}^{\mathrm{LD}}; \theta_G); \theta_D)$ as fake data with the target value 0. Overall, the following minimax game is played:

$$\min_{\theta_G} \max_{\theta_D} \; \mathbb{E}_{\mathbf{x}^{\mathrm{HD}} \sim p_{\mathbf{x}^{\mathrm{HD}}}} \log \left[D\left(\mathbf{x}^{\mathrm{HD}}; \theta_D\right)\right] + \mathbb{E}_{\mathbf{x}^{\mathrm{LD}} \sim p_{\mathbf{x}^{\mathrm{LD}}}} \log \left[1 - D\left(G\left(\mathbf{x}^{\mathrm{LD}}; \theta_G\right); \theta_D\right)\right] \tag{2}$$

The generator, on the other hand, uses a modified loss function, called *perceptual loss*. One half of the perceptual loss constitutes the generative component of the adversarial loss presented in (1). For N training samples, it can be expressed as:

$$l_{Gen} = -\frac{1}{N} \sum_{k=1}^{N} \log \left[D\left(G\left(\mathbf{x}_k^{\mathrm{LD}}; \theta_G\right); \theta_D\right)\right] \tag{3}$$

Here, minimizing over $-\log\left[D\left(G\left(\mathbf{x}^{\mathrm{LD}}; \theta_G\right); \theta_D\right)\right]$ usually yields improved performances compared to minimizing over $-\log\left[1 - D\left(G\left(\mathbf{x}^{\mathrm{LD}}; \theta_G\right); \theta_D\right)\right]$ [1].

The second half of the perceptual loss, called *content loss*, is added in the formulation of the generator's optimization problem. For a low-dimensional input pattern, it helps the generator to construct a pattern that is close to the associated high-dimensional data pattern. The loss is expressed as the Euclidean distance between the generated high-dimensional data point $G(\mathbf{x}^{\mathrm{LD}};\theta_G)$ and the associated high-dimensional data point $\mathbf{x}^{\mathrm{HD}}$ [4]. The content loss, for N training samples, is defined as follows:

$$l^{\boldsymbol{\Phi}}_{content} = \frac{1}{N}\sum_{k=1}^{N}\left(\boldsymbol{\phi}\left(\mathbf{x}_k^{\mathrm{HD}}\right) - \boldsymbol{\phi}\left(G\left(\mathbf{x}_k^{\mathrm{LD}};\theta_G\right)\right)\right)^2 \tag{4}$$

Here, $\boldsymbol{\Phi} = (\Phi_1, \Phi_2, \ldots, \Phi_\kappa)$ denotes κ number of feature extractors. In [4], the authors realized $\boldsymbol{\Phi}$ using the the VGG19 [6] network. However, in this work, we choose to use mean squared error (MSE) as the loss function. This is so because here we do not use images, and hence, we do not find using a ConvNet to be an appropriate choice. Thus, for N training samples, we use the following content loss:

$$l_{content} = \frac{1}{N}\sum_{k=1}^{N}\left|\left|\mathbf{x}_k^{\mathrm{HD}} - G(\mathbf{x}_k^{\mathrm{LD}};\theta_G)\right|\right|_2^2 \tag{5}$$

Thus, the total perceptual loss is a weighted sum of the content loss and adversarial loss expressed as follows:

$$l = \underbrace{l_{content}}_{\text{content loss}} + \lambda \times \underbrace{l_{Gen}}_{\text{adversarial loss}} \tag{6}$$

where $\lambda > 0$ is a coefficient.

2.3 Conditional GANs (cGANs) [5]

Let $\mathbf{x}^{\mathrm{C}} \in \mathbb{R}^{d_C}$ be a d_C-dimensional data point drawn from the distribution $p_{\mathbf{x}^{\mathrm{C}}}$, i.e., $\mathbf{x}^{\mathrm{C}} \sim p_{\mathbf{x}^{\mathrm{C}}}$. Let $\mathbf{z} \in \mathbb{R}^m$ be an m-dimensional noise vector drawn from the distribution $p_{\mathbf{z}}$, i.e., $\mathbf{z} \sim p_{\mathbf{z}}$. Let $\mathbf{x}^{\mathrm{O}} \in \mathbb{R}^{d_O}$ be a d_O-dimensional data point drawn from the distribution $p_{\mathbf{x}^{\mathrm{O}}}$, i.e., $\mathbf{x}^{\mathrm{O}} \sim p_{\mathbf{x}^{\mathrm{O}}}$. Here, $\mathbf{x}^{\mathrm{C}}$, one of the many possible representations of a particular type of data, is an input and acts as the condition. $\mathbf{x}^{\mathrm{O}}$ is the desired output of $\mathbf{x}^{\mathrm{C}}$ and $\mathbf{z}$ is used to introduce diversity in the generated output. cGANs attempts to create a mapping, $G(\mathbf{x}^{\mathrm{C}}, \mathbf{z}; \theta_G) \rightarrow \mathbb{R}^{d_O}$, where θ_G denotes the parameters of the generator $G(\mathbf{x}^{\mathrm{C}}, \mathbf{z}; \theta_G)$. cGANs primarily finds its application in domain-to-domain image translation, for instance, in converting the sketch of a shoe to the image of an actual one. A discriminator, $D(\mathbf{x}^{\mathrm{C}}, \cdot; \theta_D)$, is trained considering the target output of $D(\mathbf{x}^{\mathrm{C}}, \mathbf{x}^{\mathrm{O}}; \theta_D)$ to be 1 and the target output of $D(\mathbf{x}^{\mathrm{C}}, G(\mathbf{x}^{\mathrm{C}}, \mathbf{z}; \theta_G); \theta_D)$ to be 0. The generator and discriminator is indeed trained to solve the following minimax problem:

$$\min_{\theta_G}\max_{\theta_D} L_{\mathrm{cGAN}} \tag{7}$$

where

$$L_{\mathrm{cGAN}} = \mathbb{E}_{\mathbf{x}^{\mathrm{C}}\sim p_{\mathbf{x}^{\mathrm{C}}}, \mathbf{x}^{\mathrm{O}}\sim p_{\mathbf{x}^{\mathrm{O}}}} \log[D(\mathbf{x}^{\mathrm{C}}, \mathbf{x}^{\mathrm{O}}; \theta_D)] + \mathbb{E}_{\mathbf{x}^{\mathrm{C}}\sim p_{\mathbf{x}^{\mathrm{C}}}, \mathbf{z}\sim p_{\mathbf{z}}} \log[1 - D(\mathbf{x}^{\mathrm{C}}, G(\mathbf{x}^{\mathrm{C}}, \mathbf{z}; \theta_G); \theta_D)] \tag{8}$$

The generator's job, however, is not only to fool the discriminator, but also to create a data distribution $p_{G(\mathbf{x}^{\mathrm{C}}, \mathbf{z}; \theta_G)} = p_{\mathbf{x}^{\mathrm{O}}}$. Hence, the generative loss consists of an L_1 loss term as well. L_1 loss is used instead of L_2 loss, since the former encourages less blurring and more concrete loss values [4].

$$L^G_{L_1} = \mathbb{E}_{\mathbf{x}^{\mathrm{O}}\sim p_{\mathbf{x}^{\mathrm{O}}}, \mathbf{x}^{\mathrm{C}}\sim p_{\mathbf{x}^{\mathrm{C}}}, \mathbf{z}\sim p_{\mathbf{z}}}\left[\left|\mathbf{x}^{\mathrm{O}} - G(\mathbf{x}^{\mathrm{C}}, \mathbf{z}; \theta_G)\right|\right] \tag{9}$$

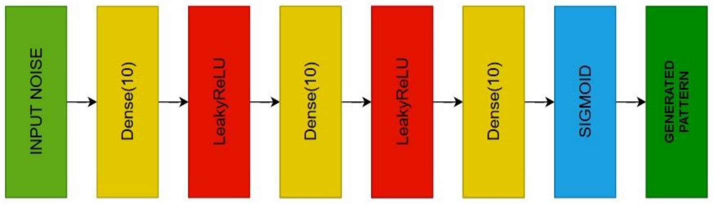

Figure. 1. Network architecture of the generator

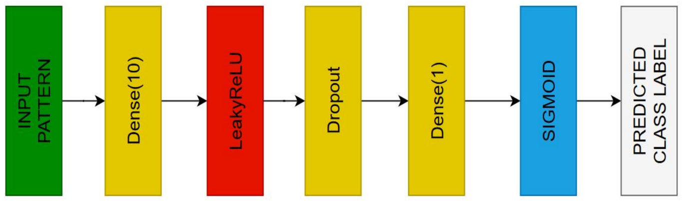

Figure. 2. Network architecture of the discriminator

The final generative loss is a weighted sum of conditional loss and the L_1 loss:

$$L_{\text{Gen}} = L_{\text{cGAN}} + \lambda L^G_{L_1} \tag{10}$$

where $\lambda > 0$ is a coefficient.

In [4], the noise $\mathbf{z}$ was introduced in the network by either concatenating it with the input $\mathbf{x}^C$ or by applying dropout at several layers in the generator. However, it was observed that the generator simply learned to ignore the noise [5]. We choose not to investigate cGANs further (in an empirical way) to keep this work concise.

2.4 Network Architecture

We examine both vGANs and SRGANs. For all the experiments, we use MLPs to realize the generators and the discriminators. As illustrated in Figure 1, the generator consists of two hidden dense layers with Leaky ReLU [7] activation functions, followed by another dense layer with a sigmoid activation function. As illustrated in Figure 2, the discriminator consists of a single hidden dense layer, with Leaky ReLU [7] activation and dropout [8]. When we investigate vGANs, the output layer of the generator has 2 nodes, whereas, when we examine SRGANs the output layer of the generator has 3 nodes.

3 Experiments

3.1 Datasets

For the inspection of vGANs, we use a 2-dimensional dataset of a total of 1500 points; 500 points are drawn from each of the three normal distributions $\mathcal{N}((1,1)^T,(0.5,0.5))^T$, $\mathcal{N}((6,6)^T,(0.6,0.6)^T)$, and $\mathcal{N}((11,11)^T,(1,1)^T)$. We show the dataset in Figure 5a. In 5a, we use three different colors to show points drawn from three different distributions. Note that, we use the same dataset for training as well as for testing.

For the inspection of SRGANs, for training, we need a low dimensional dataset and an associated high dimensional dataset. We use a 2-dimensional dataset of 3000 points;

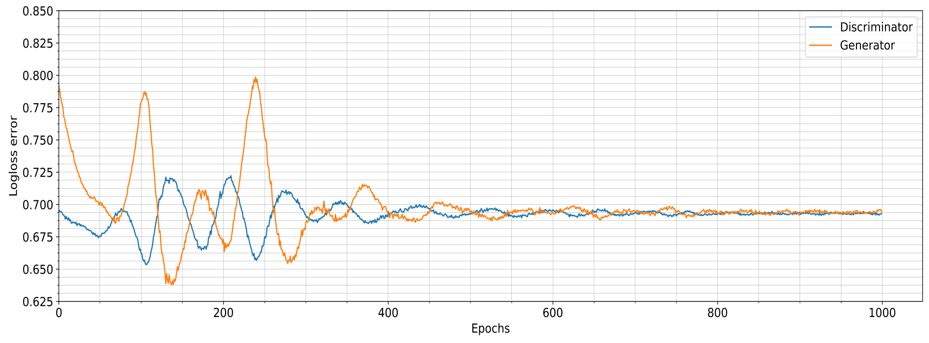

Figure. 3. Training log-loss error graph for generator (in orange) and discriminator (in blue) spanning 1000 epochs in vGANs experiment.

1000 points are drawn from each of the three normal distributions $\mathcal{N}((20,20)^T,(1,1)^T)$, $\mathcal{N}((40,40)^T,(3,3))^T$, and $\mathcal{N}((80,80)^T,(5,5)^T)$, as the lower dimensional training dataset. To construct an associated high dimensional training dataset, we introduce a third dimension $x_3 = 0.25 \times x_1 + 0.75 \times x_2$, where x_1 and x_2 correspond to the two dimensions of the low dimensional training dataset, respectively. We illustrate the low-dimensional training dataset in Figure 5c. We use a separate 2-dimensional dataset, with 600 points, for testing. The corresponding 3-dimensional test dataset is generated in the same fashion as we do for the training dataset. We illustrate this dataset in Figure 5d.

3.2 Pre-processing, Experimental Setting, and Implementation Details

We linearly normalize each feature in $[0,1]$. In the first experiment, for optimization, we use `Adam` [9] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We use a mini-batch size of 100, a learning rate of 20^{-4} and train for 1000 epochs. In the second experiment, we use a mini-batch size of 200 and $\lambda = 1$. In the second experiment, we use a learning rate of 10^{-4} for the first 500 epochs and a learning rate of 10^{-3} of the last 500 epochs. We train the discriminator and generator alternately. The implementation using `PyTorch` and several other graphical comparisons are available on `GitHub`[1].

3.3 Results and Discussions

First, we discuss the results for the experiment related to vGANs. Figure 3.3 shows the log-loss error versus the number of iterations (epochs). Figure 3.3 shows a high degree of fluctuation in the initial stage of learning. In Figure 5b, we illustrate some data points generated from the trained generator. From Figures 5a and 5b, we visually confirm that the generator could generate data points that are somewhat similar to the data points that we use for training.

Now, we discuss the results for the experiment related to SRGANs. Figure 3.3 shows the log-loss error versus the number of iterations (epochs). Figure 3.3 shows a high degree of fluctuation in the initial stage of learning. We find an MSE of 3.4555×10^{-5} on the test data. To compare the performance of GAN with respect to an MLP with the same architecture of the generator network, we train an MLP with the same architecture for

1. https://github.com/arnabsinha99/GANs

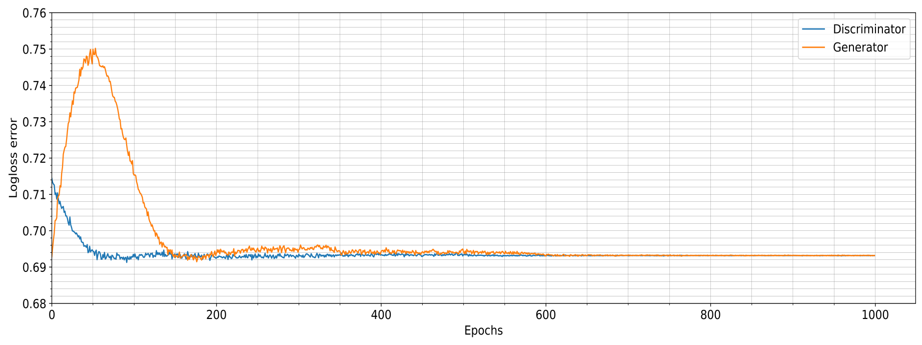

Figure. 4. Training log-loss error graph for generator (in orange) and discriminator (in blue) spanning 1000 epochs in SRGANs experiment.

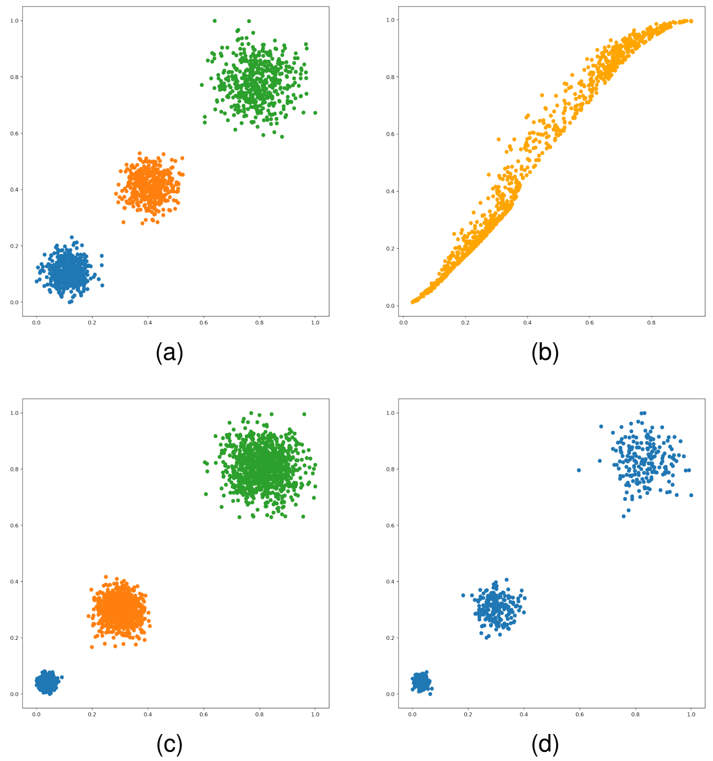

Figure. 5. (a) Training data for the experiment related to vGANs (b) Output generated by vGAN model after training (c) 2-dimensional training data for the experiment related to SRGAN (d) 2-dimensional test data for the experiment related to SRGAN.

super-resolution considering the learning process as a regression task. For making a fair comparison, we use the same set of initial weights as used in the generator to initialize the MLP. We find an MSE of 4.3432×10^{-5} on the same dataset using the MLP. Hence, we find the performance of the SRGAN to be better than the MLP for super-resolution task.

4 Conclusion and Future Work

Here, we investigate three variants of GANs: vGANs, SRGANs and cGANs. We revisit their mathematical formulations. Then, we inspect the abilities of vGANs and SRGANs on low-dimensional synthetically generated datasets. We realize the generators and the discriminators using MLPs. We also experimentally compare the performance of the generator of a GAN with an MLP when we use the MLP for the generation of patterns taking into account the associated learning as a regression task.

First, we observe that the philosophy of GANs can be applied on low-dimensional synthetically generated datasets to attain somewhat acceptable performances. Second, we also found quantitative evidence that GANs usually perform better than MLPs concerning the super-resolution task.

This work has a few shortfalls. First, we do not perform any experiment on cGAN though we have revisited its mathematical formulation. Second, we did not use any real-world but tiny dataset in this work. It would be interesting to examine how GANs work in such cases. Third, we did not repeat our experiments. Therefore, our observations might be an outcome of the randomness associated with the learning. In future, we intend to perform experiments on cGANs and possibly on other variants of GANs including cycleGANs [10] and bicycleGANs [11].

References

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

[2] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

[3] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8110–8119.

[4] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[5] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[7] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[10] J.-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman, "Toward multimodal image-to-image translation," in *Advances in neural information processing systems*, 2017, pp. 465–476.

[11] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.