Generative Adversarial Networks for Genre-Based Art Generation

Laura Miron
Stanford University
lmiron@stanford.edu

Abstract

This paper evaluates the performance of different GAN architectures on generating new genre-based images. Images are taken from Kaggle/Wikiart and belong to one of six genres: abstract, cityscape, flower, landscape, portrait, and religious. We find that normal DCGAN slightly outperforms conditional DCGAN, and greatly outperforms VanillaGAN.

1. Introduction

Much of visual machine learning over the past decade has focused on image classification, and performance at this task has improved exponentially, especially with the advent of deep convolutional neural networks. More recently, style transfer has become a popular area of study, producing very convincing results transferring various artistic styles to single images and even videos. Yet style transfer always requires a 'content' input, it does not truly invent new images.

Image *generation* remains a challenging problem, although recently generative networks have shown significant promise in generating natural images using the MNIST, CIFAR-10, CUB-200, and LFW datasets. [16] Perhaps the most well-known example is NVIDIA's computer-generated celebrity faces from the celebA dataset.

All of these examples, however, use natural images to generate natural images. Other than Tan *et al.*[16], relatively little progress has been made generating artwork. Generating artwork is more challenging than natural images for several reasons: 1. Artwork has widely varied styles such as drawing and painting. 2. Artwork can be abstract or representational, making it difficult to distinguish foreground and background.

Due to these difficulties, improving the performance of GANs generating artwork could be instrumental to improving the performance of GANs overall. Moreover, there are a myriad of applications for computer-generated artwork such as standalone art, concept art for movies and video games.

Our inputs are images belonging to one of six genres: abstract, cityscape, flower, landscape, portrait, and religious, and labeled by genre. We feed them to three architectures of GAN: VanillaGAN, deep convolutional gan (DCGAN), and conditional deep convolutional GAN (cDCGAN). Our outputs are sample images evaluated qualitatively, as well as inception scores.

2. Related Work

Generative adversarial networks were first proposed by Goodfellow et al. [7] in 2014. Using GANs, the authors are able to generate natural images using the MNIST, TFD and CIFAR-10 datasets that are significantly sharper than images generated by models based on maximum likelihood training.

Since then, many GAN variants have been proposed and applied to natural image generation. In 2014, Mirza et al. extended GANs to a conditional model, cGANs [10], in which both the generator and discriminator are conditioned on extra information y such as class labels, tags or attributes.

These early GANs were notoriously difficult to train, and often suffered from the problem of mode collapse. Introducing convolutional layers alleviated problem. In 2015, Denton *et al.* proposed a cascade of convolutional networks within a Laplacian pyramid framework to generate images in a coarse-to-fine fashion.[6] Also in 2015, *Makhzani et al.* described a set of constraints on the architectural topology of convolutional GANs, name this class Deep Convolutional GANs (DCGAN). DCGAN are stable to train in most settings. [11]

Around the same time, Springenberg [15] and Salimans [13] each proposed categorical GANs (CatGAN) models, in which G and D not only receive label information, but D outputs a probability distribution over either K or $K + 1$ (where one category is "fake") classes.

Other generative models that have been used for image generation include PixelCNN [17] and variational autoencoders.

3. Methods

Generative adversarial networks (GANs) consist of two adversarial networks, a generator G and a discriminator D. G and D compete in a two-player minimax game in which the generator generates images from random noise, and the discriminator tries to distinguish between real and synthetic images. More specifically, G learns a mapping from a random noise vector z to an image y, $G : z \leftarrow y$, and D learns a mapping from an image x to a value between 0 and 1 representing the probability that the image is real.

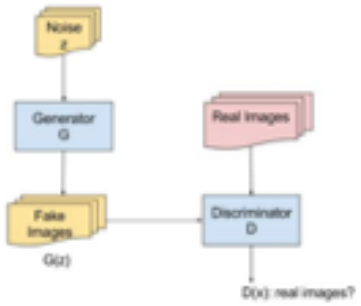

Figure 1. Structure of a GAN

Typically, the minimax game between D and G is defined by the target function:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{x\sim p_z(z)}[\log(1 - D(G(z)))] \quad (1)$$

. Intuitively, the quantity which D tries to maximize and G tries to minimize is the sum of the expected probability of the discriminator classifying a real image as real and the expected probability of classifying a fake image as fake. In practice, in one update step, the discriminator is run once on a batch of all real images, and once on a batch of all fake images.

3.1. VanillaGAN

Our VanillaGAN architecture, based off extending the network defined in [4] to three input and output channels, is defined below. The generator uses a fully-connected layer followed by three convolutional layers. In contrast to DCGAN, the convolutional layers are normal forward convolutions, and do not perform upsampling, either maintaining or shrinking the size of the image. Rather, upsampling is performed by the bilinear interpolation layers. Bilinear interpolation is an extension of linear interpolation to two variables where linear interpolation is performed first in one direction and then again the other direction– in the case of VanillaGAN, on the width and then height of the image, performed independently across each channel. Batch normalization is performed before each relu activation layer, as batchnorm has been shown to be critical in training GANs. [13]

The discriminator uses two convolutional layers with average pooling followed by two fully-connected layers, with no final activation layer.

There are several other important differences between VanillaGAN and DCGAN. VanillaGAN uses a gaussian initialization of the z noise vector input to G, claimed by [14] to improve the quality and variety of results. Additionally, the discriminator is pre-trained for 200 iterations: fake image batches are created by running z noise vectors through G as usual, but the generator loss is not computed and the weights are not updated. This is to prevent mode collapse, where the generator learns how to 'trick' the discriminator with images that do not look real to the human eye, because the discriminator never correctly learned to distinguish real from fake. The discriminator is also trained at a higher learning rate than the generator.

Notably, the last generator activation is a sigmoid, meaning the values output are in the interval (0,1).

3.2. DCGAN

Our DCGAN uses the [9] implementation. The discriminator uses 4 convolutional layers with leaky relu activations, and a final fully-connected layer with sigmoid activation output. Each convolutional layers halves the height and width and doubles the number of channels.

The generator architecture is pictured below. It uses one fully-connected and four de-convolutional layers, each of which doubles the height and width of the previous layer's activation, and halves the number of channels. Deconvolution reverses the effects of convolution, that is, the output of a deconvolutional layer is the matrix that, if convolved with the filters, would produce the input to the deconvolutional layer. Thus, each layer of the generator exactly reverses one layer of the discriminator. Upsampling and downsampling is performed exclusively by these convolutional layers, as opposed to the interpolation and pooling in VanillaGAN.

The z noise vector is initialized uniformly rather than by gaussian. Preliminary tests of gaussian initialization showed no noticeably difference in performance, so the uniform initialization uses in the original implementation was used for the majority of trials. The discriminator is not pretrained, and moreover, the generator optimization step is run twice per iteration. The generator and discriminator use the same learning rate. With DCGAN, pre-training the discriminator often led to the discriminator assigning labels with 100% certainty and discriminator loss going to 0. Once in this state, the generator cannot learn and the network cannot recover.

The last generator activation is $tanh$, clipping the output

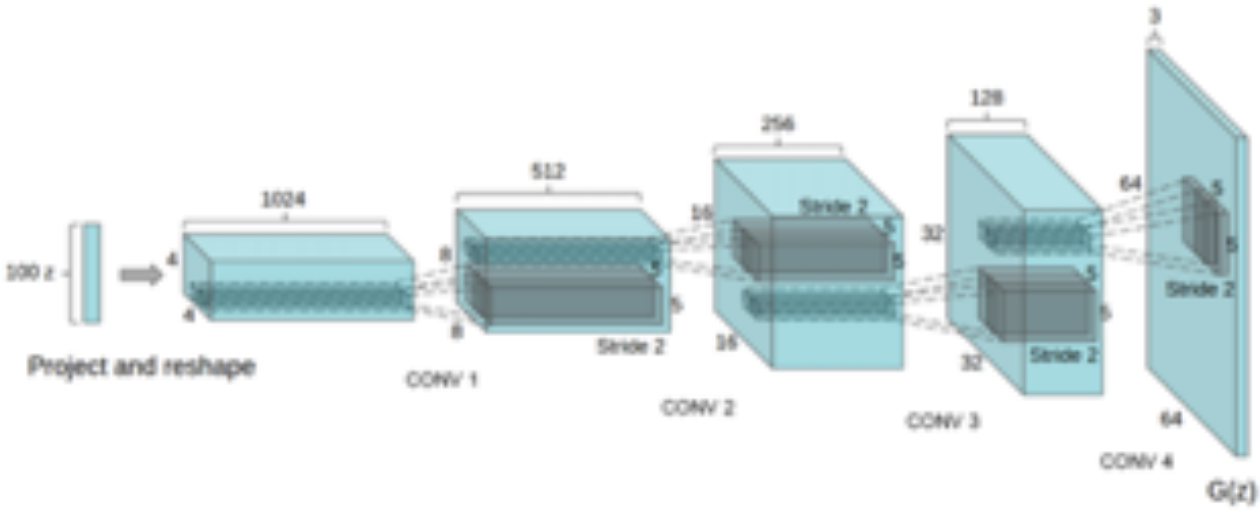

Figure 2. Architecture of DCGAN generator

values to the interval (-1,1). Many papers [14] have shown normalizing the inputs to both G and D to improve results, and we confirm these findings: DCGAN with sigmoid inputs and outputs and all other parameters unchanged failed to converge. Although we ran out of time to test VanillaGAN with (-1,1) inputs, we suspect the sigmoid activations to have contributed to VanillaGAN's non-convergence.

3.3. cDCGAN

In a conditional GAN, or cGAN, a label vector y which encodes information about either the attributes or classes of the data is fed into D and G as an additional input layer. The discriminator does not classify images by label, as in categorical GAN [15], and still outputs a probability of the input being real or fake. *Tan, et al.* have seen that providing label information improves the performance of artwork-generating GANs.

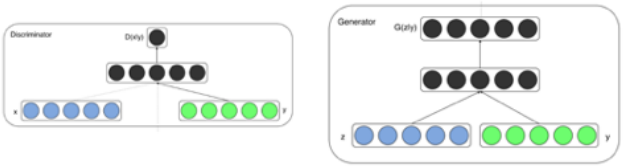

Figure 3. Conditional GAN

We choose to encode y as a 6-dimensional label vector indicating the genre. We feed y to D and G by concatenating y to the channel dimension of both the input and the activation at each layer of both.

4. Dataset and Features

We use the Painter by Numbers dataset curated and maintained by Kaggle [2]. It is a database of painting and sketches mostly taken from wikiart.org [1]. The paintings are annotated and categorized by artist, genre, and style. Although we performed some experiments with painting categorized by artist, the small number of painting per artist made the network difficult to train, and we focus this paper on the results of paintings categorized by genre.

We use a subset of 55,734 examples from the 100,000+ examples in the full kaggle dataset, limited to a subset of genres: abstract, cityscape, flower painting, landscape, portrait, and religious painting. We use a subset of genres both to speed up training and to improve the results of conditional GAN. We chose among genres that contained the most examples, and attempted to choose genres with relatively low variability within the genre and clear distinctions from the other genres. For example, we did not choose both flower and bird-and-flower paintings, or both portraits and self-portraits.

For conditional DGCAN, the image data is concatenated with the label data; for VanillaGAN and DCGAN no label data is provided, but the paintings are still limited to this subset of genres.

We use the new tensorflow data pipeline to encode and preprocess the images: We use cv2 to read the raw jpegs and resize to 64x64 pixels, and then write the image and labels to a tfrecords file. The resize is done by interarea interpolation. Critically, we originally used interlinear interpolation and the quality of the results noticeably improved with inter area interpolation. Interlinear interpolation applies a lowpass filter over the image in order to achieve a trade-off between visual quality and edge removal (lowpass filters tend to remove edges). Interarea interpolation resamples using pixel area relation, and therefore achieves better results when downsampling.

abstract	cityscape	flower	landscape	portrait	religious
15978	7031	2147	19296	21571	9020

Figure 4. Number of examples in each class

Figure 5. Sample of each class of training images

We used a distorting resize rather than a crop because Wang *et al.* [18] found doing so did not negatively affect generated image quality. Moreover, any crop is likely to remove important features from paintings, such as the heads of portraits.

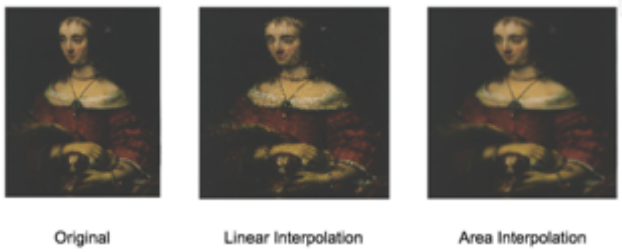

Figure 6. Quality of training images after downsampling

In additional preprocessing, we linearly transform the RGB values from (0,1) to (-1,1), but we do not perform any other normalization. On some trials we perform random data augmentation by repeating the dataset to the total number of examples is 2-3x the true number, and by applying random left-right flips, brightness, and saturation. However, this had no discernible effect on the generated image quality.

5. Results

Of the three architectures, DCGAN performs the best. In the samples in fig. 7, all models have been trained for 30 epochs.

VanillaGAN seems to have experienced mode collapse- it has converged, and outputs nearly identical images no matter what the z input is. This output successfully fools the discriminator because the discriminator has not been sufficiently or correctly trained. Despite extensive hyperparameter searching, we were unable to find parameters that outperformed the default suggest parameters for this model– Adam optimizer, D learning rate: .0003, G learning rate: .0001, batch size: 100, z dimensions: 100.

We also performed hyperparameter searching on DCGAN and cDCGAN, and were unable to find parameters that outperformed the suggested parameters– Adam optimizer, D learning rate: .0002, G learning rate: .0002, batch size: 64, z dimensions: 100.

The DCGAN results, while not sufficient to fool the human eye, have many successful characteristics. First, there is significant variation between different samples. This indicates that the generator output is correctly dependent on the z noise input. Second, the images are smooth, and do not contain the ghostly grid that can be seen in the VanillaGAN results and more faintly in the cDCGAN results. Lastly, even though the generator output is not conditioned on any category, many of the the samples clearly belonging to one of the six genres in the input data (abstract, cityscape, flower, landscape, portrait, and religious). For instance, the first and seventh paintings in the dcgan column are clearly landscapes, while the second and sixth samples are clearly portraits.

Given that Tan *et al.*[16] found conditional information improved art generation results, it was somewhat surprising that for us cDCGAN performed significantly worse. We are not sure why this is the case, but suspect genres were not easily distinguishable categories to the discriminator. Although we partially evened out the distribution of examples across genres by hand-selecting the 6 specific genres, some genres are still much more common than others within the dataset. Moreover, 'landscape' and 'cityscape' paintings are often very similar to each other, as well as portraits and religious paintings (both often figures against dark background).

To quantitatively evaluate our results, we calculate the *inception score* of samples drawn from each model. Incep-

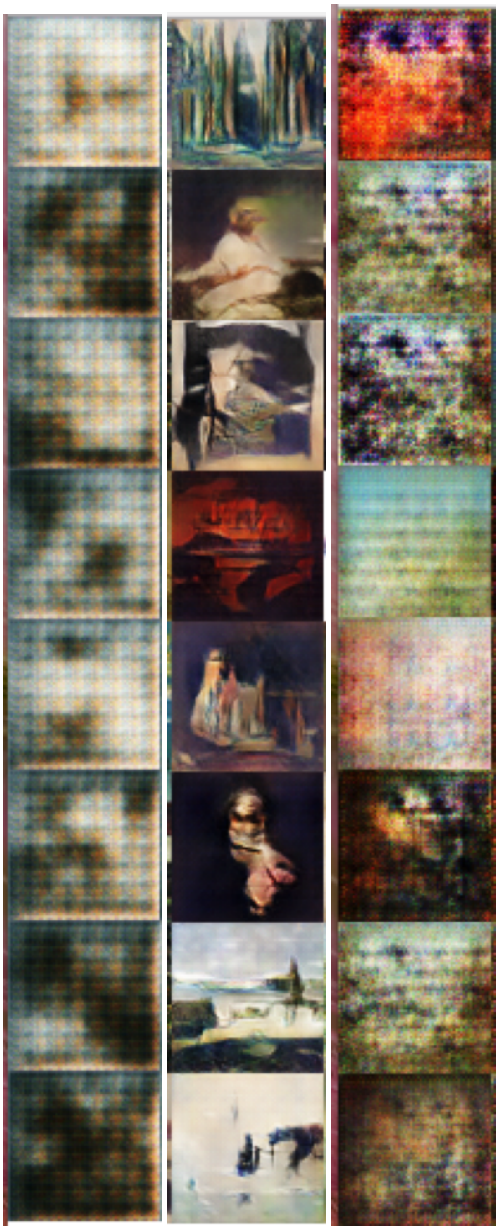

Figure 7. Samples produced by VanillaGAN (top), DCGAN (middle), and cDCGAN (bottom)

tion score is a metric proposed by Salimans *et al.*, which they show correlates very well with human judgement, although it does not produce good results when used as an objective for training. Inception score rewards low entropy within a class and high entropy between classes, which is fittingly also the criteria we strove for when choosing genres for our real input images.

We calculate the inception score using code from [12], and a model pre-trained on ImageNet [5]. Although this means the model is trained on natural images, many [16, 8] still use it as a metric for artwork.

The inception score results agree with our quantitative analysis: DCGAN was by far the most successful of the three models, and VanillaGAN and cDCGAN suffer in particular from lack of diversity of output.

	mean	std dev
VanillaGAN	0.732	0.028
DCGAN	2.32	0.217
cDCGAN	1.11	0.025

Figure 8. Inception scores

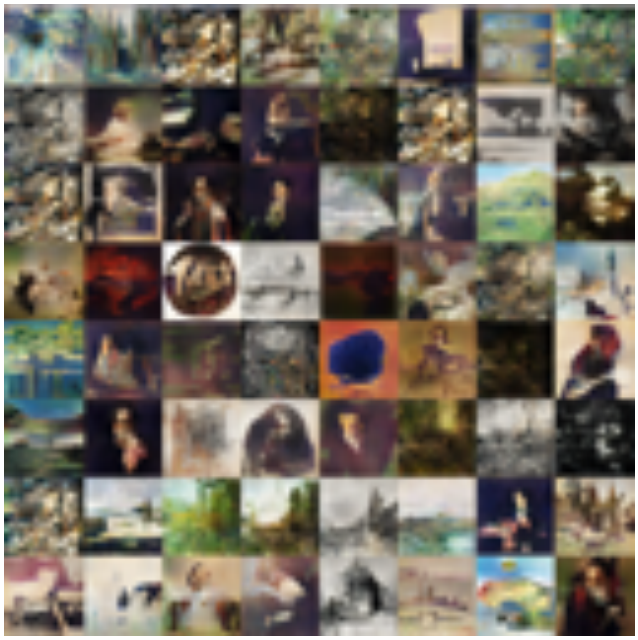

Figure 9. More DCGAN results

We also ran DCGAN on subsets of artwork restricted to a specific genre and performed qualitative analysis. Each genre behaves differently. Abstract artwork produces the widest variety of output, but it is difficult to asses how convincing an abstract form is. Religious artwork was the least successful of the four genres tested, and does not converge to any recognizable forms, likely because there are fewer examples of religious art in our dataset than the other genres. Portraits are certainly the category with least internal variation. Accordingly, the portrait outputs look distinctively like portraits, with a centered figure from the waist up against a dark background. However, there is little variation among the output samples. Lastly, landscapes output a mixture of reasonable results and garbage. One challenge in the landscape category is that the data is a mixture of true landscapes and seascapes, which accounts for the predominantly blue sample outputs.

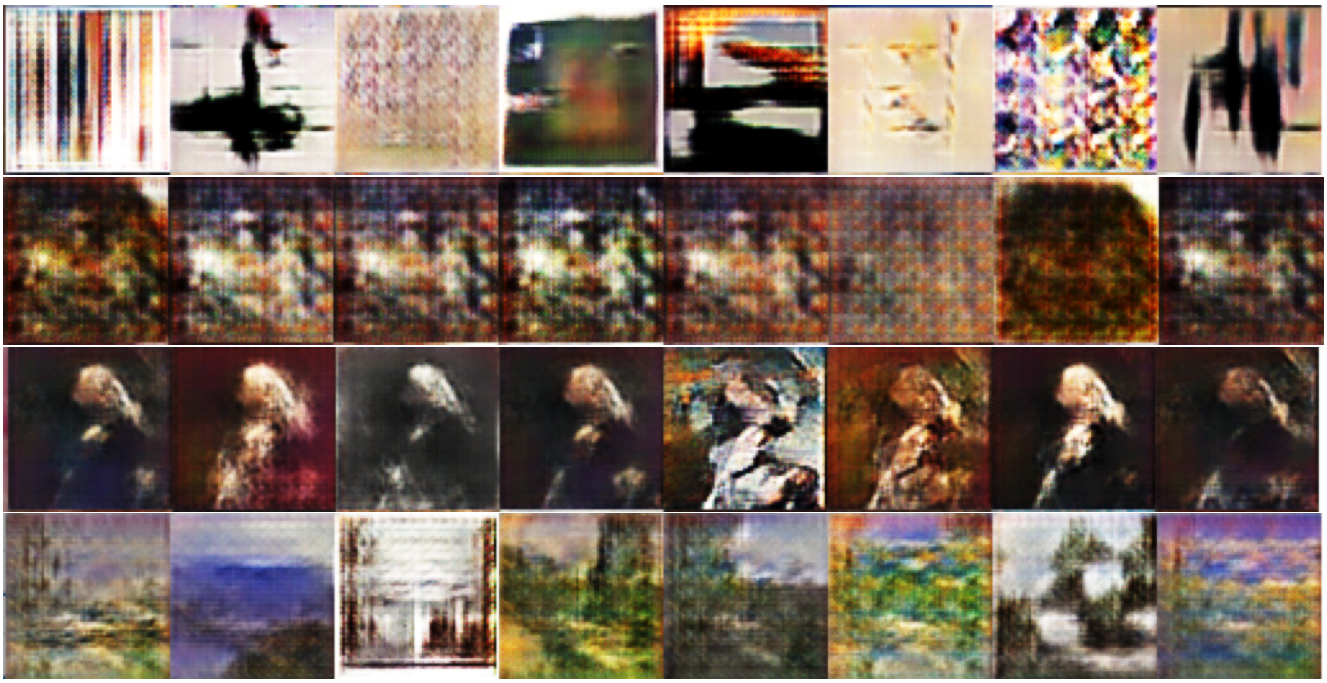

Figure 10. Samples produced by DCGAN for specific genres (top to bottom): abstract, religious, portraits, landscape

One reasonable question is whether the models are truly generating new images, or simply memorizing and outputting slight variations of the training data. We perform k nearest neighbors analysis to show that this is not the case. We can see by comparing DCGAN output samples to their nearest neighbors that the two images differ significantly, and the generator is therefore creating entirely new images.

Figure 11. KNN results. Top: DCGAN output, bottom: closest neighbor from training data

6. Conclusion

We conclude that DCGAN is an effective model for producing generated artwork, and also that, when training on a single genre of art, the choice of genre heavily influences the quality of the generated images. There is still huge potential for improvement. Although our trials of cDCGAN were unsuccessful, other research suggests that cDCGANs are a powerful tool capable of producing even better outputs than normal DCGAN, and we would therefore like to experiment further with making cDCGAN work. Perhaps using more genres would be helpful, or differently selected genres. Random crops and flips could be used to augment genres with fewer examples. Similar genres could be combined, such as portraits and self-portraits, or landscapes and cityscapes. Many sources suggest that WassersteinGAN [3, 18] is even more effective at art generation than any of the models we tested. We would like to explore these possibilities in future papers.

7. Contributions and Acknowledgements

I worked on this project alone. (I use 'we' through the paper because 'I' for some reason did not sound correct). I use code from the following repositories, which are also cited throughout the paper: https://github.com/carpedm20/DCGAN-tensorflow, https://github.com/willtwr/ArtGAN, https://github.com/openai/improved-gan/tree/master/inception_score.

References

[1] Wikiart visual art encyclopedia. https://www.wikiart.org/.

[2] Painter by numbers (data). https://www.kaggle.com/c/painter-by-numbers/data, 2016. Labeled dataset of wikiart paitings.

[3] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan, 2017.

[4] J. Bruner and A. Deshpande. Generative adversarial networks for beginners. https://github.com/willtwr/ArtGAN, 2017.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database, 2009.

[6] E. Denton, S. Chintala, A. Szlam, and R. Fergus. Deep generative image models using a laplacian pyramid of adversarial networks, 2015.

[7] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[8] K. Jones. Gangogh: Creating art with gans. https://towardsdatascience.com/gangogh-creating-art-with-gans-8d087d8f74a1, 2017.

[9] T. Kim. Dcgan-tensorflow. https://github.com/carpedm20/DCGAN-tensorflow, 2018.

[10] M. Mirza and S. Osindero. Conditional generative adversarial nets, 2014.

[11] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[12] T. Salimans. improved-gan. https://github.com/openai/improved-gan/tree/master/inception_score, 2018.

[13] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans, 2016.

[14] M. A. Soumith Chintala, Emily Denton and M. Mathieu. How to train a gan? tips and tricks to make gans work. https://github.com/soumith/ganhacks, 2016.

[15] J. T. Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks, 2015.

[16] W. R. Tan, C. S. Chan, H. Aguirre, and K. Tanaka. Artgan: Artwork synthesis with conditional categorical gans, 2017.

[17] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders, 2016.

[18] G. Wang, Y. Chen, and Y. Chen. Chinese painting generation using generative adversarial networks. http://cs231n.stanford.edu/reports/2017/pdfs/311.pdf, 2017.