Making Art with Generative Adversarial Networks

Loran Knol (s3182541)[a] Thijs Havinga (s2922924)[a]
Elisa Oostwal (s2468255) [a] Jeroen Overschie (s2995697) [a]

[a] *University of Groningen, Nijenborgh 9 9747AG Groningen*

Abstract

Generative Adversarial Networks (GANs) have proven themselves suitable for generating increasingly realistic images. A number of recent developments have shown very promising results, such as the progressively growing StyleGAN [8] that generates high resolution images of human faces. This study aims to use those developments to generate paintings with the style and content of just one artist, namely the Dutchman Vincent van Gogh. Although this artist has produced a large number of works, the number of usable paintings was still too small for training GANs, so data augmentation had to be performed. Two different architectures, DCGAN and StyleGAN, were trained on the augmented data. Of the two, StyleGAN showed the most promising results.

1 Introduction

Generative Adversarial Networks are a relatively new type of technique for generating samples from a learned distribution, in which two networks are simultaneously trained whilst competing against each other. Applications for GAN's are numerous, including image up-sampling [18], image generation, and the recently quite popular 'deep fakes' [11].

In this project[1], we aim to train such a Generative Adversarial Network ourselves, with the purpose of image generation, specifically. As the generation of human faces has been widely studied, we have chosen a different topic, namely: the generation of paintings. While large datasets of paintings are available (e.g., [15]), we have opted to restrict ourselves to one artist, as we believe this will give a better chance at producing realistic paintings. For this, we have chosen the Dutch artist Vincent van Gogh, who is known for his unique style. The dataset is taken from [13] and consists of roughly 2×10^3 images. The goal then becomes to generate paintings that resemble Van Gogh's in terms of color usage and style.

1.1 Generative Adversarial Networks

Generative types of techniques have been existent in Machine Learning literature for some time - we are able to draw samples from distributions using sampling techniques such as rejection sampling or Markov Chain Monte Carlo techniques such as Gibbs sampling. However, we are often obliged to make harsh assumptions about the data distribution at hand, obstructing the goal of learning to generate more samples from a distribution automatically. Whereas discriminative models have seen massive successes in the deep learning field [5], generative models enjoyed success a bit later.

Generative models preceding GANs include denoising autoencoders [10], Deep Boltzmann Machines [16] or Generative Stochastic Networks [1]. Whereas Deep Boltzmann Machines use an undirected graphical model approach with latent variables, Generative Stochastic Networks do not require a Machine Learning practitioner to define an explicit distribution, allowing training by using ordinary back-propagation. Autoencoders also use a latent code using a deep network, forcing the model to compress the essential data distribution information into the bottleneck latent code layer. After the input

[1] Code and instructions available at: github.com/dunnkers/generative-adversarial-networks

data is 'compressed' into a smaller latent code layer, the autoencoder will decode the latent code, trying to best reconstruct the original input.

Generative Adversarial Models [3] add another component: the **adversary** element. While the *generator* model G tries to reconstruct a sample which best matches the original data distribution from a latent code, the adversarial *discriminative* model D tries to classify the generated output as real or fake in a zero-sum game. In this way, two models are simultaneously trained with opposing goals: while the target of G is to 'fool' the discriminator, D is optimized to classify whether a sample is made by G and is in p_G, or whether it came from the real data distribution x. Given some noise input z drawn from p_z, G wants to minimize $\log(1 - D(G(z)))$. The game D and G play can be formalized as

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{1}$$

which means our objective/loss function is a combination of two models: one trying to minimize some quantity whilst the other is trying to maximize it. Note that Equation 1 might be modified to prevent gradient saturation: We might instead train G to maximize $\log D(G(z))$. Though GANs can be effectively trained using this formulation, we are at the risk that one model might excel over the other. In this case training effectiveness will falter, causing the zero-sum game dynamics to fail. Contemporary GANs often take extra measures to best synchronize D and G during training.

2 Dataset

The dataset used in our study is taken from [13]. This collection consists of approximately 2×10^3 paintings by Van Gogh. Upon further inspection we find that the set contains both black-and-white sketches as well as colored paintings. To aid the specialization of the network we opt to only use the colored paintings for training, filtering the images based on high saturation. This reduces the number of images to 744. This is a far too small number of samples, considering usually ten-thousands or even millions of images are used in order to train a GAN, see e.g. [14]. We therefore resolve to data augmentation in order to increase the number of images. The details of this are described in the following subsection.

2.1 Data augmentation

From our data set, we generate square patches as data samples for our GAN model. Through various methods, we produce square pictures of a fixed size in pixels. This makes it easier to use the samples in a model. Using various methods, we generate patches that are slightly different while retaining the visual contents of the painting. The methods used, in order, are:

1. First, we adjust the brightness of the picture by -5%, 0% or 5%.
2. Then, we apply a *shear* operation on the picture in the x- and/or y-axis, or neither.
3. From the resulting pictures, we cut square patches that cover as much of the picture as possible. This yields several possible square sub-images.
4. We then resize the square patches to a fixed size (for example, 212×212).
5. To the resulting patch, some noise from a normal distribution ($\sigma = 5$ for integer pixel values) can be added or not.

When combined these steps can produce a large number of slightly different patches of the same image. To keep computation for our small project feasible, we cut off the process at 20,000 augmented samples, which we empirically find to represent 162 different paintings.

3 Methods

To generate paintings with a GAN, two architectures have been selected: the relatively simple architecture of DCGAN [14], as well as the more advanced and recent architecture of StyleGAN [8], which allows for an interesting comparison across network design complexity. Their mechanisms will be described below. The Plug & Play architecture [12] was also considered, as it showed promising results for several classes of images, indicating it had good generalized generative properties. This method however relied on the *caffe* interface, which was not installed on the Peregrine cluster.

3.1 DCGAN

Described in [14], the DCGAN model refers simply to using deep convolutational networks for a GAN architecture. As a first experiment to use our augmented data set, we create a design from scratch from the two essential components in a GAN, the generator network G and the discriminator network D. We also added an encoder component E later to better allow the generator to learn useful weights: Training this additional encoder network with the generator network as an autoencoder based on the same data set allows these networks to learn latent representations which the generator is able to translate to the original paintings. By using a subset of 3 paintings from the original data set and applying our data augmentation routine, we attain our final data set X for use in our DCGAN. Training the combination of E and G with X as both the input and the expected output, E learns to generate latent representations h from samples from X, while G learns to transform h back to the original sample. At the same time, G and D are trained with zero-sum adversarial optimizer functions as described in Section 1.1. As such, D learns to distinguish output from G produced with any random latent code from original data samples from X, while G learns to generate pictures that look increasingly like the data samples.

3.2 StyleGAN

Aside from DCGAN, a more recently developed GAN architecture called StyleGAN [8] was also used. StyleGAN is based on Progressive GAN [7], which, for both the generator and discriminator components, adds layers during training, yielding generated images of increasingly higher resolution. This progressive growing of image resolution is beneficial for both speeding up and stabilizing the training process. StyleGAN, in turn, modifies the architecture grown by Progressive GAN in a number of ways.

The first modification concerns the latent code, $\mathbf{z} \in \mathcal{Z}$, which is not fed directly to the network but is first preprocessed by an eight-layered fully connected (FC) multilayer perceptron (MLP) f to produce the intermediate latent vector $\mathbf{w} \in \mathcal{W}$. Learned affine transformations A then convert $\mathbf{w}$ to so-called styles $\mathbf{y}$, which in turn are fed into adaptive instance-normalization (AdaIN) units. With this set-up, the AdaIN units instance-normalize the feature maps $\mathbf{x}_i$ of a second input $\mathbf{x}$, and ensure that the mean and variance of every feature map $\mathbf{x}_i$ becomes aligned with the style $\mathbf{y}_i$ [6, 8].

As a final modification, StyleGAN injects independent images of Gaussian noise after every convolutional layer, before the input is fed into the AdaIN units. The noise is scaled per channel with a learned transformation B, and is hypothesized to relieve the network from the burden of having to generate its own pseudorandom numbers from the network input to induce the necessary stochasticity at the different resolution levels in the generated images [8]. See Figure 1 for an overview of the StyleGAN architecture.

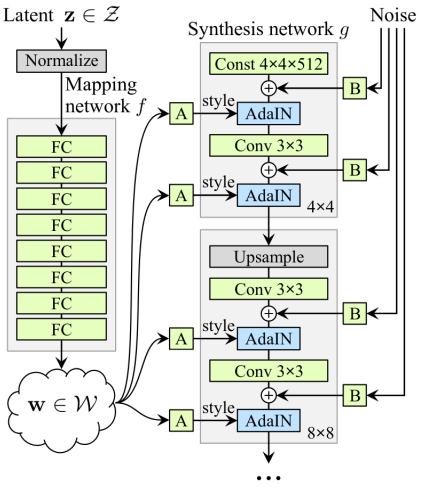

Figure 1: StyleGAN architecture [8]. A latent code $\mathbf{z}$ is transformed through the MLP f to $\mathbf{w}$, which then controls the AdaIN units within the generator g. Independent noise images are injected before every AdaIN unit. Every block of two convolutional layers (and their respective AdaIN units) is responsible for upsampling the image from the previous block; the resulting resolution is indicated in the lower-right corner of the gray outlines.

Because the training of StyleGAN takes a considerable amount of time and because it was not certain whether the training method would converge within the given time constraints, two separate approaches were taken. The first one takes a StyleGAN instance from [8] that was fully trained on a high-quality faces dataset. With that fully trained network, training is then continued on the augmented Van Gogh dataset. This means that the network does not grow in resolution anymore, since the maximum resolution (1024x1024) has already been reached before the Van Gogh training even commenced. The intuition behind this approach is that the faces generated by the original StyleGAN instance hopefully already lie closer to the manifold of Van Gogh paintings. The second approach is somewhat similar to the first one (use a pre-trained network instance), except

that all training parameters are reset to their starting values - including the resolution. This allows the parameters of the low-resolution parts of the original, fully trained network to be retrained first, and makes the network regrow in resolution.

The training hyperparameters were kept the same as in [8]. The network was trained with non-saturating GAN loss, using the Adam optimizer [9] with $\alpha = 0.001$, $\beta_1 = 0$, and $\beta_2 = 0.99$. However, α increased with resolution size, and its values were 0.0015 for 128^2, 0.002 for 256^2, 0.003 for 512^2 and 1024^2. The minibatch size also varied per resolution: 256 for 4^2 and 8^2, 128 for 16^2, 64 for 32^2, 32 for 64^2, 16 for 128^2 and 8 for 256^2, 512^2, and 1024^2. Both networks were trained with two NVIDIA Tesla K40 GPUs and a total of 24 cores from two Intel Xeon E5 2680v3 CPUs; each network received 24 hours of training. The results of both networks will be compared qualitatively, primarily based on how much the authors think the generated images resemble Van Gogh paintings, and quantitatively, which incorporates calculating the Fréchet Inception Distance (FID) [4] with an Inception-v3 network [17] that has been trained on ImageNet [2].

4 Results

One instance of DCGAN and two instances of StyleGAN have been trained on the augmented Van Gogh dataset. The results of these training runs will be described in the sub-sections below.

4.1 DCGAN

As an initial experiment with GANs, we designed a DCGAN as described in Section 3.1. We did not evaluate the results of this DCGAN formally as we did with our main experiment concerning the StyleGAN model. Some results produced by our DCGAN are shown in Figure 2. The pictures produced by our DCGAN seem to contain areas showing near-perfect copies from one of the source paintings.

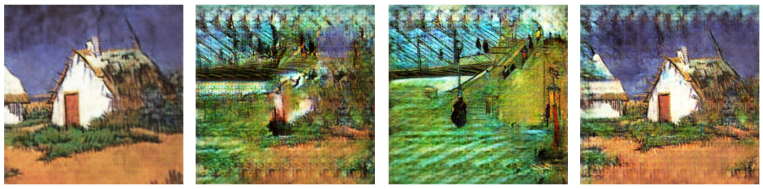

Figure 2: Example results from our DCGAN architecture. The left-most picture was generated with a latent code produced by the encoder and thus represents one of the input samples. The remaining pictures were generated from random latent codes.

4.2 StyleGAN

A sample of the images generated by the StyleGAN instance with constant network size (also referred to as 'training resumed' from here) over the course of training is given in Figure 3 (left). Every row corresponds to a random latent code, every column to the network state at the end of a training tick. The number of images the network needs to be trained on before one tick has passed depends on the training schedule; such a tick thus does not necessarily constitute one full pass through the training data (i.e. a tick is not an epoch). The first column in the figure contains images generated by the original StyleGAN instance from [8]. From the second column onward, the originally generated faces quickly get a painting-like look and change into objects that Van Gogh would often paint, like flowers and (self-)portraits. However, the very last column contains images that do not seem to match the painting style and objects observed in the middle four columns.

Figure 3 (right) shows sampled images from the progressively growing StyleGAN instance (which we will refer to as 'training restarted') over the course of training. Since this rendition of the network starts off with generating lower-resolution images, it is faster to train, hence it can do more ticks over the same training period; only a few ticks are showcased here for clarity. The first column again shows the faces generated by the original network, while in the second column the resolution is reset to 8×8

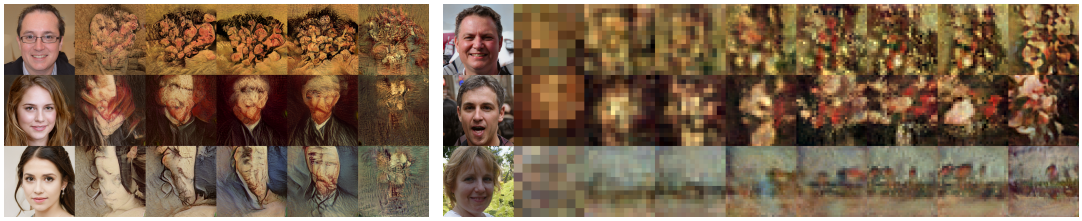

Figure 3: Generated images over the course of training the constant-sized (left) and progressively grown (right) StyleGAN instance. Every row corresponds to a randomly generated latent code, every column corresponds to a training tick.

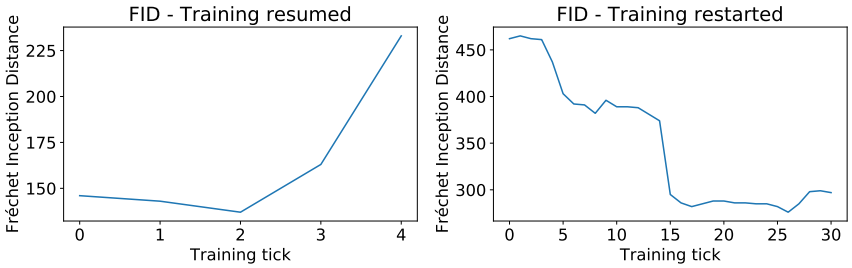

Figure 4: FIDs of the constant-sized (left) and the progressively grown (right) StyleGAN instances. The FID of the final state of the StyleGAN instance provided by [8], that would precede the FID of tick 0, is *not* included in the figures to allow for easier comparisons.

and increases from there. As training progresses, the pixelated blobs of color slowly coalesce into resemblances of flowers (first and second row) and landscapes (third row).

To also evaluate the progress of the StyleGAN instances quantitatively, the FID has been calculated for all training ticks of both instances (see Figure 4). The StyleGAN instance for which training was resumed shows a gently declining FID for the first three ticks, which then rises rapidly. Especially the high FID in the last tick matches well with the last column of Figure 3 that has lost some of its resemblance to Van Gogh paintings.

As for the StyleGAN instance whose training schedule was reset, FIDs are higher overall than the instance for which the schedule was simply resumed, but the descend of the FID over training ticks is more persistent. There seem to be two moments at which the FID takes a relatively large step downwards, namely when the resolution is increased from 8×8 to 16×16 and from 16×16 to 32×32.

5 Conclusion

We have trained two types of GANs, DCGAN and StyleGAN, on a dataset of paintings by Van Gogh with the aim of developing a network which is able to generate new paintings that have a style similar to Van Gogh's. Of the two, StyleGAN gave the most promising results. While the pretrained (constant-sized) instance in general gave better results than the progressively grown network in terms of the FID, we note that it appears to be prone to overfitting as the FID quickly grows after half a day of training. This is also visible in the results: the paintings produced in later epochs look worse than those created after half a day of training. The trend of the FID of the progressively grown StyleGAN on the other hand steadily decreases throughout the course of training. It would have been worthwhile to train this network for even longer to observe the minimal FID we could achieve, and compare this to the best fit produced by the constant-sized StyleGAN instance.

While the final results do not portray humans or landscapes accurately, we are pleased with the generated paintings, which definitely show similarity to Van Gogh's work in terms of colour usage, topics, and style.

References

[1] Yoshua Bengio, Eric Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *International Conference on Machine Learning*, pages 226–234. PMLR, 2014.

[2] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[3] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[4] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6629–6640, Red Hook, NY, USA, 2017. Curran Associates Inc.

[5] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[6] X. Huang and S. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1510–1519, 2017.

[7] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

[8] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[10] Xugang Lu, Yu Tsao, Shigeki Matsuda, and Chiori Hori. Speech enhancement based on deep denoising autoencoder. In *Interspeech*, volume 2013, pages 436–440, 2013.

[11] Francesco Marra, Diego Gragnaniello, Davide Cozzolino, and Luisa Verdoliva. Detection of gan-generated fake images over social networks. In *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 384–389. IEEE, 2018.

[12] Anh Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. Plug play generative networks: Conditional iterative generation of images in latent space, 2017.

[13] Kaggle: Van Gogh Paintings. `https://www.kaggle.com/ipythonx/van-gogh-paintings`.

[14] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[15] Kaggle: Rijksmuseum. `https://www.kaggle.com/lgmoneda/rijksmuseum`.

[16] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455. PMLR, 2009.

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[18] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.