Project report - StyleGAN2

Luka Boljević (63210482)

1 Introduction

Generative Adversarial Networks (GANs) are a class of deep learning models consisting of two networks that are trained in parallel: the generator and the discriminator, where the generator is constantly competing with the discriminator, which is an adversary that is learning to distinguish between the model distribution (i.e. generated fake images from the generator) and the data distribution (i.e. real images from the training dataset). GANs can be used in areas such as image generation, super resolution, and so on.

Training GANs is notoriously tricky (as we will see in this project as well), however researchers have developed numerous variations from the original GAN [1] to address these challenges. In this project, we will implement and test one of those variations, the famous StyleGAN2 model, proposed by Karras et al. in 2019 [5]. We will evaluate it using the FID metric, and see how the number of training steps affect the quality of generated images, as well as how sensitive this model is to values of hyperparameters used for training.

1.1 Related work

StyleGAN2 belongs to a "family" of GANs, which begins with Progressive GAN, introduced by Karras et al. in 2017 [7]. The key idea of Progressive GAN is to train the generator and discriminator progressively - the generator would slowly, i.e. incrementally learn to add increasingly finer details to its generated images, instead of learning to add them all at once. This was achieved by first training at very low resolution, like 4x4. At such small scales, the generator will first discover large-scale structure of the image distribution (like face color for instance). Then, as scale is increased, the generator learns finer and finer details of the image distribution (like eye color for example). Karras et al. showed this learning regime is more stable and is able to produce images at great resolution (1024x1024).

As a follow up of Progressive GAN, StyleGAN was introduced by the same authors in 2018 [6]. In StyleGAN, they proposed an alternative generator structure, while the discriminator (and loss functions) didn't see any changes. The architecture of the generator borrowed ideas from style transfer literature, hence the name. The idea was that, at each convolution layer/scale, the generator will adjust the "style" of the current generated image - this means that some convolutional layers will affect for example the pose, identity, skin tone, etc., while other layers will affect for example the existence of freckles, hair style, hair color, and so on.

StyleGAN2, again proposed by Karras et al. [5] was an improvement over StyleGAN, mostly modifying the generator architecture, and the discriminator only slightly. They described that StyleGAN would sometimes produce "blob-like" artifacts in its final output image, a problem which they traced to the "AdaIN" (adaptive instance normalization) operation which was introduced in StyleGAN. They removed the AdaIN operation, and introduced something called "weight demodulation", which is essentially a weight normalization technique. Apart from removing those blob-like artifacts, weight demodulation allowed them to slightly simplify the structure of the generator.

Two additional members of this family exist - StyleGAN2-ADA [3] and StyleGAN3 [4], proposed again by Karras et al.

2 StyleGAN2 and dataset

2.1 StyleGAN2

Before diving into the training and results of GAN, we will give a high level overview of how StyleGAN2 works. As described in the Related work section, the core of its architecture was developed first in Progressive GAN paper [7], and then improved in StyleGAN [6], with of course additional improvements introduced in StyleGAN2 itself.

Progressive GAN

One of the main takeaways from Progressive GAN is that learning is more stable and ultimately yields better results if the generator progressively learns to generate features. In other words, we will obtain better final results if the generator first starts by generating coarse features, which it then refines as scale increases. This intuitively makes sense as to why it would yield better results, as the generator does not have to generate everything (all features) at once.

StyleGAN

While the discriminator and training regime was okay, the authors of StyleGAN introduced an improvement to the generator architecture, which could control the style (features) of the generated images better. They achieved this by modeling the generator as a combination of a "mapping network f" and a "synthesis network g".

Traditionally, a random latent code z would be the input to the first layer of the generator, and then it's not used anymore. In StyleGAN, this is a bit different. First, as its input, the mapping network f takes in the random latent code $z \in \mathcal{Z}$ and transforms it to an "intermediate" latent code $w \in \mathcal{W}$.

The synthesis network is the main part of the generator, i.e. the part which actually generates the image. Now, instead of z being the input to the layers of the synthesis network g, w is the input, and not just in the first layer of g, but in all (convolutional) layers of g. With this, the intermediate latent code w will be responsible for controlling the style of the generated images at each scale, i.e. at each convolutional layer. This process is described in more detail (and in a very readable fashion) in the original paper [6], but this is the general idea.

StyleGAN2

Even though StyleGAN improved the quality and diversity of generated images quite significantly, they noticed that some of their images produced "blob-like" artifacts, which, as they state in paper on StyleGAN2 [5], appears at around 64x64 resolution, and becomes progressively more prominent at higher resolutions. They traced this problem to the AdaIN (adaptive instance normalization) operation of convolutional blocks in the synthesis network g, which is why they decided to not use it in StyleGAN2.

Before removing AdaIN, they first reorganized the structure of the generator to better serve their redesign. Then, Karras et al. introduced a technique called "weight demodulation", which essentially normalized the weights used in convolutional layers to have unit standard deviation. This removes the blob-like artifacts and yet retains full controllability over the generated images.

The authors of StyleGAN2 also wanted to search for an architecture that would still produce high quality images, but without the progressive growing that Progressive GAN and StyleGAN used. After investigating a few architectures, they decided to go for a "skip" generator, and "residual" discriminator. Details are provided in Section 4.1 of their paper [5].

Other than weight demodulation, a skip generator, and residual discriminator, StyleGAN2 has many other components: equalized learning rate introduced in Progressive GAN [7], style mixing regularization, R1 and path length regularization for discriminator and generator losses respectively, mini batch standard deviation, bilinear up/downsampling, truncation, and so on. However, we will not go into details here.

2.2 CelebA dataset

We decided to train StyleGAN2 on the popular CelebA dataset [2]. The CelebA dataset is a face-attribute dataset containing 202599 celebrity images, with 10177 unique identities, partitioned in 3 sets - train (162770 images), validation (19867 images), and test (19962 images). Apart from the identity, each image has annotations for: (1) 40 binary attributes (eyeglasses, wearing hat, mustache, male, narrow eyes, mouth slightly open, no beard, ...), (2) 5 landmark points (left eye, right eye, noise, left mouth, right mouth), and (3) a bounding box around the face itself (in theory, the values are completely wrong in the original dataset). All images are of size 218x178. The dataset is very diverse, covers large pose variation and is widely used to train all kinds of deep learning models, not just GANs.

3 Implementation

We implemented StyleGAN2 using PyTorch. The implementation is available on Github.

Since the published papers don't always provide all implementation details regarding the individual layers, Karras et al's official StyleGAN2 implementation, as well as the official StyleGAN2-ADA implementation provided us with some help. On the other hand, as the official code is slightly overengineered for an average user, we also got help other publicly available source code for StyleGAN2, such as LabML AI's code, lucidrains' code, and ppeetteerrs' code.

From the StyleGAN2 paper (and its siblings), we used weight demodulation in the convolutional layers, lazy regularization for R1 and path length regularization techniques, skip generator and residual discriminator. We didn't use mini batch standard deviation and style mixing regularization. For generating the images, we also used the truncation trick introduced in StyleGAN paper, although we're not entirely sure if it helped (the implementation was most likely off even though we followed NVIDIA's source code, or something else was amiss).

For training, we had access to the Arnes cluster, so training was done on a single NVIDIA V100 32GB GPU (side note: I didn't know/manage to set up training on both GPUs). For all tested models, we use the following. We trained the models on 70000 images from the train split of CelebA dataset. We took one step of training the discriminator, and one step of training the generator and mapping network together. We used the Adam optimizer for all three parts, and set $\beta_1 = 0.0, \beta_2 = 0.99$. The loss function was the non saturating logistic loss introduced in the original GAN paper [1], but as aforementioned we used R1 and path length regularization. We used lazy regularization with an interval of 16 - meaning that the loss regularization terms are calculated only once every 16 steps. The dimensionality of z and w was set to 512, as per the original papers. We used batch size 32. Lastly, unlike in the original StyleGAN2, where images were generated at 1024^2 resolution, we only went up to 64^2, so there was at least hope of training a model in a reasonable amount of time.

4 Results and discussion

We mostly played around with learning rate and the number of steps for accumulating gradients. Although finding the right values was rather tricky (due to mode collapse issues), we managed to find the ones that resulted in a successfully trained model. Besides the aforementioned hyperparameters, that model used 0.002 learning rate for the generator and discriminator, and 0.0002 for the mapping network. It accumulated gradients for 4 steps, and was initially trained for 10000 steps. Training took right around 13 hours. In Figure 1 is a 4x4 grid generated by that very model. All of the models we will mention here are available on this OneDrive link.

Figure 1: A 4x4 grid generated by the model trained for 10000 steps. Each face i.e. image in this grid is of size 64x64. While the faces are evidently rather blurry (not just because this image is upscaled), it's pretty clear the generator has learned to generate something.

As we said, to evaluate the performance of our models, we will use the popular FID (Fréchet Inception Distance) metric. To do so, we first took the test split of the CelebA dataset (19962 images), and resized it to 64x64, as our model generates images of size 64x64. We then generated the same amount of images, and calculated FID. The model achieved a FID of **38.365**, which is a relatively decent result given the "small" amount of training.

To try an improve the result slightly, we continued training the model for an additional 10000, 20000 and 30000 steps (so, 20000, 30000 and 40000 steps in total). In Figures 2, 3, and 4, we can see 4x4 grids generated by those 3 models. The faces got quite a bit clearer when we jumped from 10000 to 20000 steps, and perhaps slightly more going to 30000 i.e. 40000 steps, but it's hard to tell. In any case, we can definitely say that the generator can produce relatively clear faces, with quite a bit of variation.

Figure 2: A 4x4 grid generated by the model that was trained for 20000 steps in total.

In Table 1 we can see the FID score for all 4 models. FID increased quite significantly from 10000 to 20000 steps, but

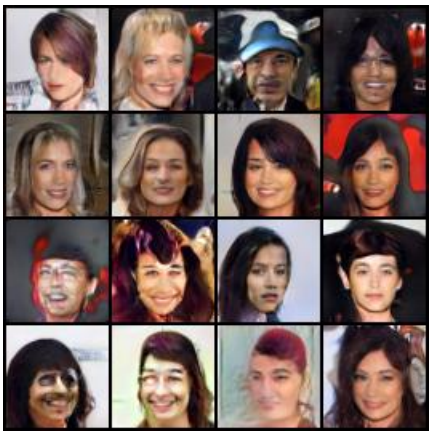

Figure 3: A 4x4 grid generated by the model that was trained for 30000 steps in total.

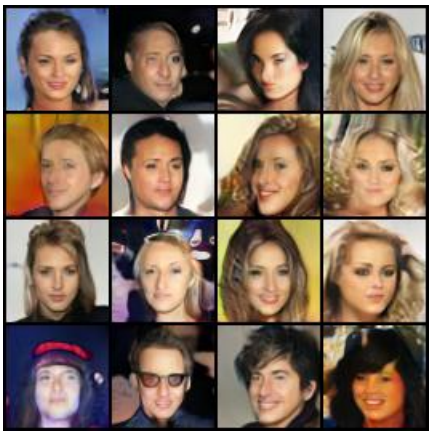

Figure 4: A 4x4 grid generated by the model that was trained for 40000 steps in total.

we didn't achieve that big of an improvement going to 30000 or 40000 steps.

Number of steps	FID score
10000	38.365
20000	23.937
30000	22.957
40000	21.229

Table 1: FID scores for all 4 models, i.e. for the same model trained for 10000, 20000, 30000 and 40000 steps.

4.1 Mode collapse

Unfortunately, most of the models we tried weren't very successful. The specific hyperparameters that didn't work can be found in the repository, but they all experienced mode collapse - whatever the input is, the model would generate exactly the same amount, any number of times. Figure 5 shows a similar 4x4 grid as we've seen already, but generated by a model that experienced mode collapse.

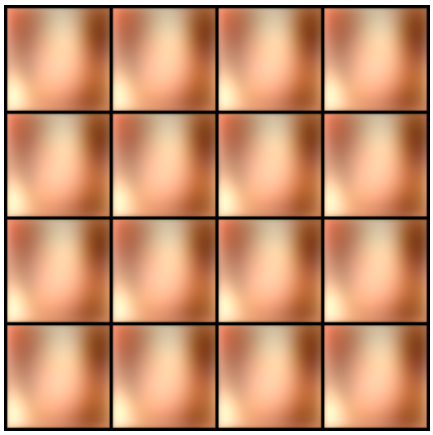

Figure 5: A 4x4 grid generated by a model that went through mode collapse. All generated images are pretty much exactly the same, regardless of the input.

5 Conclusion

In this project, we implemented the StyleGAN2 model, one member of a family of GAN networks. We provided an overview and the main idea of StyleGAN2. We successfully trained one model, while unfortunately the majority experienced mode collapse. We then compared how the number of training steps affects the quality of generated images and FID score on the successfully trained model.

Of course, our implementation of StyleGAN2 is quite "dumbed down" from the original, so it was expected that we would not be able to generate images of the same quality. Although tricky to find the right hyperparameter values to successfully train a model, we still managed to find them and train the model for long enough so faces are generated. In any case, this goes in line with the fact that GANs are quite sensitive to hyperparameter values. Perhaps if we trained the model to generate 128^2 or 256^2 images, and further experimented with different hyperparameters, we would be able to get higher quality images.

References

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

[2] Liu, Ziwei and Luo, Ping and Wang, Xiaogang and Tang, Xiaoou. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[3] Tero Karras and Miika Aittala and Janne Hellsten and Samuli Laine and Jaakko Lehtinen and Timo Aila. Training Generative Adversarial Networks with Limited Data, 2020.

[4] Tero Karras and Miika Aittala and Samuli Laine and Erik Härkönen and Janne Hellsten and Jaakko Lehtinen and Timo Aila. Alias-Free Generative Adversarial Networks, 2021.

[5] Tero Karras and Samuli Laine and Miika Aittala and Janne Hellsten and Jaakko Lehtinen and Timo Aila. Analyzing and Improving the Image Quality of StyleGAN, 2020.

[6] Tero Karras and Samuli Laine and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks, 2019.

[7] Tero Karras and Timo Aila and Samuli Laine and Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation, 2018.