IKT450-G Deep Neural Networks Project Report

Using generative adversarial networks on places dataset to generate new images of scenes

LUKASZ FILIP MROZIK

SUPERVISOR
Morten Goodwin

University of Agder, 2021
Faculty of Engineering and Science
Department of Engineering and Sciences

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Abstract

Multiple GAN architectures were evaluated on "MIT Places Database for Scene Recognition", including DCGAN, DCGAN trained as WGAN-gp, residual networks trained as WGAN-gp and StyleGAN2-ada. All used for generating 64x64 RGB images. The amusement park and corridors categories were used. Most GANs struggled with the high randomness of the content of pictures taken inside amusement parks, but StyleGAN2-ada produced decent results on both datasets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of this project is to generate images using the "MIT Places Database for Scene Recognition" dataset[1].
This dataset contains images from different scenes. In this project two scenes were used: amusement parks and corridors. In the pictures of amusement parks we see very many different kinds of pictures as seen in Figure 1.1. Many of the corridor images are empty inside corridors which should make the task of image generation easier, but as seen in Figure 1.3, there are exceptions. The resized variant of the Places Database was used which contains 256x256 jpeg files, but they were later converted into 64x64 png files. The dataset was supposed to have 18178 images of corridors and 42286 images of amusement park, but I later discovered that I only have 15100 image for both categories. This could be caused by using python's tarfile module not working on tarfiles that big. On the bright side that is still more than 80% of the corridor images, and now have an direct comparison on how well we can generate images of amusement parks and corridors using the same amount of data available.

In this project the neural networks are only going to be evaluated on 64x64 RGB images.

Figure 1.1: Amusement park

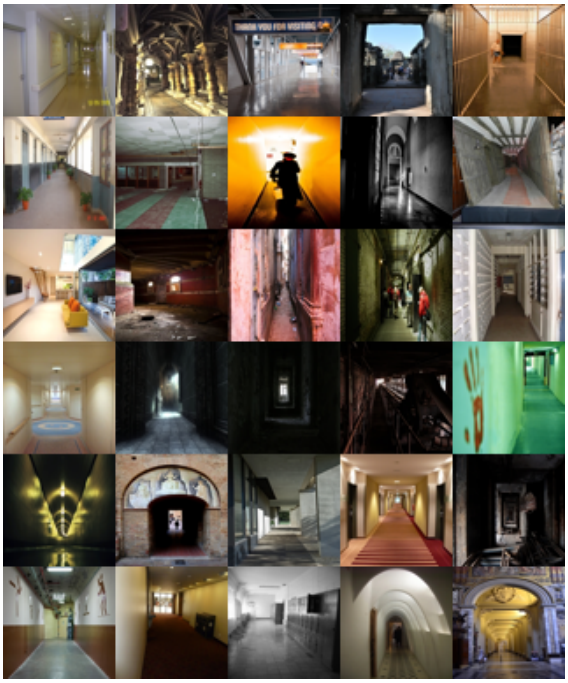

Figure 1.2: Corridors

[1]available on http://places.csail.mit.edu/ (2021-12-8)

Figure 1.3: Outside corridors, pictures of humans inside a corridor and corridors with unusual colors

Chapter 2

Theory

2.1 GAN

Generative adversarial networks (GANs) is an adversarial framework in which two models are trained at the same time. An generator G translates an input noise z into a samples in a probability distribution and a discriminator D that differentiates between samples from the generator G and from the dataset. The generator G doesn't have direct access to the dataset, instead it plays a minimax game with the discriminator D by training to generate samples that the discriminator D mistakes for data from the dataset. Both models can be trained by backpropagation. In the paper "Generative Adversarial Networks" by Ian J. Goodfellow et al. multilayer perceptrons were used.[2]

2.2 DCGAN

Deep convolutional generative adversarial networks (DCGANs) were proposed in 2016 in the paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks" by Alec Radford, Luke Metz and Soumith Chintala[10]. The models for the generator and discriminator do not have fully connected layers apart from the initial projection of the input z into a set of 4x4 feature maps. The models also do not use pooling layer, instead using strided convolutions. Batch normalization layers is an important part of the model as it stabilizes it by dealing with poor initialization of the model. They also found using the Adam algorithm with the beta1 value of 0.9 is less stable than with beta1 value of 0.5. The generator uses mainly ReLU activation functions while the discriminator uses LeakyReLU. This approach gave better results than the original approach which used multilayer perceptrons[10]

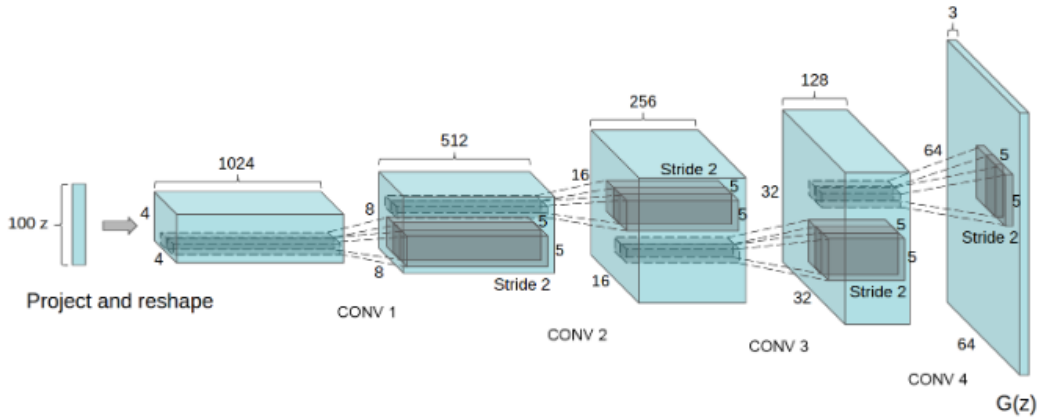

Figure 2.1: DCGAN generator[10]

2.3 Wasserstein GAN

The main feature of Wasserstein GANs it the use of Wasserstein-1/Earth Mover distance between the probability distribution of the generated data and real data. The discriminator of Wasserstein GAN is often called critic because it does not classify the output. While discriminators in vanilla GANs classify samples as true or false, critic is instead trained to output big values for real samples and small values for generated samples while it is a K-Lipschitz function. If the critic is trained like this training the generator should result in decrease in Wasserstein-1 distance. Instead of balancing the training of generator and discriminator, we ideally want to train the critic to optimality. The paper "Wasserstein GAN" by Martin Arjovsky, Soumith Chintala, and Léon Bottou proposed training the critic 5 times more often than the generator.[1]

2.3.1 WGAN with Weight Clipping

The first proposed method of enforcing the Lipschitz constraint was weight clipping. This was the first proposed method of implementing WGAN. In this method the weights of the critic are clipped into a range -c to c. [1] This works, however there was room for improvement; to quote the Wasserstein GAN paper:

> Weight clipping is a clearly terrible way to enforce a Lipschitz constraint. If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing gradients when the number of layers is big, or batch normalization is not used (such as in RNNs). [1]

2.3.2 WGAN-gp

Gradient Penalty (gp) alternative to Weight Clipping which was proposed as response to the original WGAN paper which used weight clipping. It is demonstrated on toy datasets that there exists situation where weight clipping does not work while gradient penalty works. In this method we penalize the critic for the squared difference between 1 and the gradient of the result of applying the critic to an interpolation between the fake and real samples. The downside of this method is that it assumes that critic is a function that maps a single input to a single output, something that is not correct if batch normalization is used. Layer normalization was recommended as a replacement for batch normalization layers in the paper "Improved Training of Wasserstein GANs" by Ishaan Gulrajani et al..[3]

2.4 StyleGAN

StyleGAN is a GAN architecture inspired by the Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization paper by Xun Huang and Serge Belongie[4] which proposed adaptive instance normalization. While the original purpose of the adaptive instance normalization layers was for use in style transfer, the Style-Based Generator Architecture for Generative Adversarial Networks(StyleGAN)[6] is used to generate new images.

2.4.1 v1

The generator in the StyleGAN architecture has a constant learnable input. Instead the latent space is passed to the adaptive instance normalization layers to control the "style" of the generated output. Latent space z in passed through a mapping network consisting out of fully connected layers to get a latent space w. The latent space w is used as the scales and the biases of the instance normalization part of the adaptive instance normalization layer through learned affline transform A. Sometimes during the training multiple latent spaces are

calculated and used in different places in the network to prevent the network from assuming adjacent styles are correlated. Additionally before the instance normalization layers we add noise scaled by factor B. The noise allows the layers to generate random things such as hair, freckles and skin texture without relying on the input from the previous layer.[6]

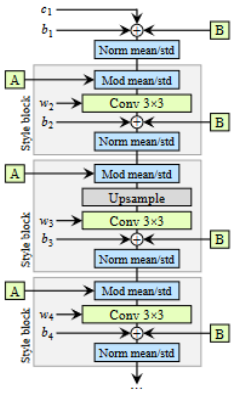

Figure 2.2: StyleGAN[7]

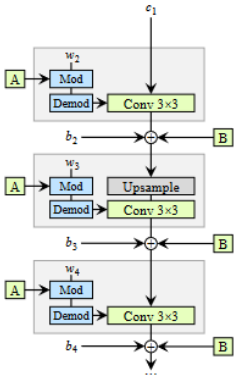

Figure 2.3: StyleGAN2[7]

2.4.2 v2

The second version of StyleGAN architecture has changes which remove some of the visual artifacts that the first version of the network sometimes created. One of the artifacts was a droplet which was caused because the normalization layers destroyed the information in the magnitudes of the feature maps relative to each other. The droplet was the generator sneaking the information past the normalization layer, and after removing the normalization of mean and only keeping normalization of variance these disappeared.
A second change done to improve the quality of the generated images is to replace the modulation of the normalization layers with a modulation of the weights of the convolution layer in a way related to weight normalization. [7]

2.4.3 Adaptive Discriminator Augmentation

The Adaptive Discriminator Augmentation (ADA) variant of StyleGAN2 was created because training the StyleGAN2 to produce images with high resolution required hundreds of thousands of images to be trained and achieve high quality results. Therefore augmentation of the data without "leaking" the augmentations to the generated samples is desired for smaller datasets. Fortunately some types of augmentation do not "leak" (make the generator generate samples with the augmentation) into the generator if we keep the probability of the augmentation happening small enough and augment both during the training of generator and discriminator. The adaptive augmentation detects the amount of overfitting in the discriminator and based on that increases or decreases the change of augmentations being applied.[8]

Chapter 3

Methods

In this project multiple GAN architectures were evaluated on the amusement park and corridor part of the places dataset. This was done without doing hyperparameter search.

3.1 DCGAN

When evaluating the DCGAN, the implementation available on the tutorials part of the PyTorch official website was used.[5] The amount of channels in the convolution layers were not modified to be the same as the ones suggested by Radford et. al. in the paper "Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks"[10], instead using the the ones in the example.

3.2 WGAN-gp

The first method of using the WGAN-gp the DCGAN implementation previously used. The sigmoid layer of the discriminator was removed and the batch normalization layers replaced by layer normalization layers. The network was trained for 97500 generator iteration with the hyperparameters suggested in "Improved Training of Wasserstein GANs" by Ishaan Gulrajani et al.[3], with the same optimizer as in appendix E "Hyperparameters used for LSUN robustness experiments". Hyperparameter beta1 of 0 was also tried for a few thousands generator iterations but produced similar results in the same timespan. The residual network architecture from that paper was also evaluated on the corridor, but only for 27500 generator iterations because it took long time to run with available resources (It took 40 hours on NVIDIA T4).

3.3 StyleGAN2-ada

StyleGAN-ada architecture was evaluated using the PyTorch implementation avaiable in the NVlabs github repository[9]. The automatic config which picks a network architecture was used, despite the generated GAN using over 8 GB of graphical memory to generate 64x64 RGB images. The network was trained for 5 million images in each dataset which took 48 hours on one NVIDIA T4 Tensor Core GPU each. Non-saturating logistic loss with path length regularizer from the paper "Analyzing and Improving the Image Quality of StyleGAN", Karras et al. 2019[7] was picked as the loss function by the preset. The list of layers in the network generated by the automatic preset available in Appendix B.

Chapter 4

Results

4.1 DCGAN

The evaluated combination of hyperparameters resulted in the discriminator being too strong compared to the generator. Adding an if statement that made the discriminator only train if the loss is abose a treshold like 0.1 resulted in the GAN getting stuck at generating random patterns on the amusement park category as shown in Figure 4.1. It did better on corridor category shown in Figure 4.2. Adding noise to labels used to train the discriminator has similar results. On the corridors the generator was jumping back and forth between generating noisy output and blurry empty corridors. See Figure 4.2 for an example.

Figure 4.1: DCGAN results amusement park (discriminator updated only when loss D $>$ 0.1)

Figure 4.2: DCGAN results corridors (discriminator updated only when loss D > 0.1)

4.2 WGAN-gp

Turning the DCGAN into a Wasserstein GAN made the training of it more stable. However training the network for more than than 35000 generator iteration had little effect and the network still have not reached images of amusement park that look any realistic, while it did suffer from the patter like artifacts like the classical GAN training algorithm, it still failed to create realistic looking images. The network trained on corridors had smaller amount of artifacts

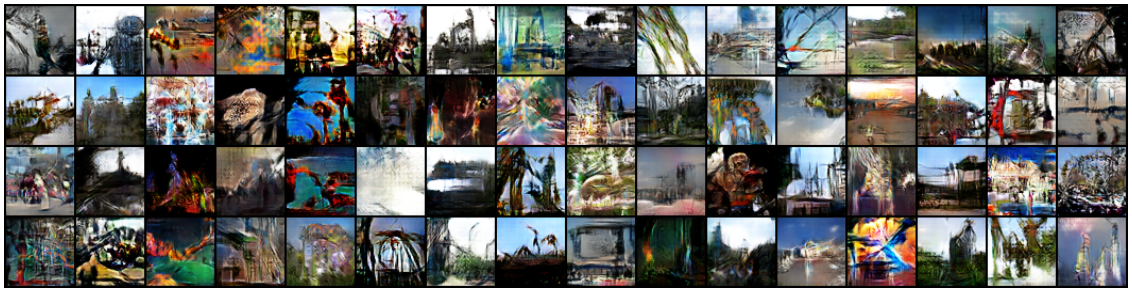

Figure 4.3: DCGAN trained as WGAN-gp after 35000 iterations (amusement park)

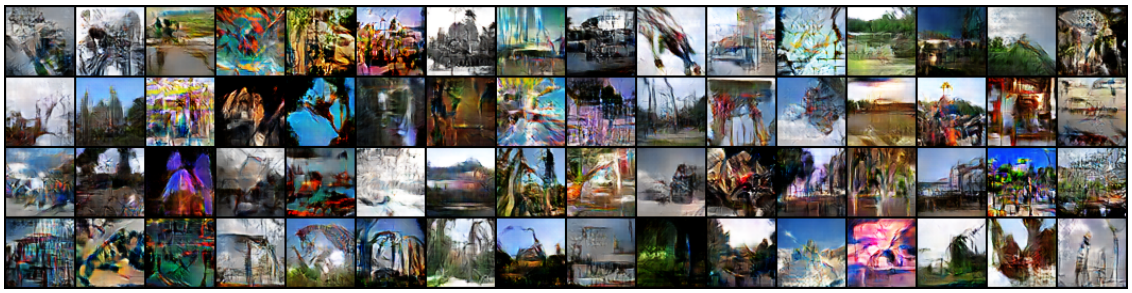

Figure 4.4: DCGAN trained as WGAN-gp after 97500 iterations (amusement park)

The architecture proposed in the paper "Improved Training of Wasserstein GANs" by Ishaan Gulrajani et al.[3] which are convolutional networks with residual connections, was trained on the corridor dataset for 28000 generator iterations. The result is visible in Figure 4.7.

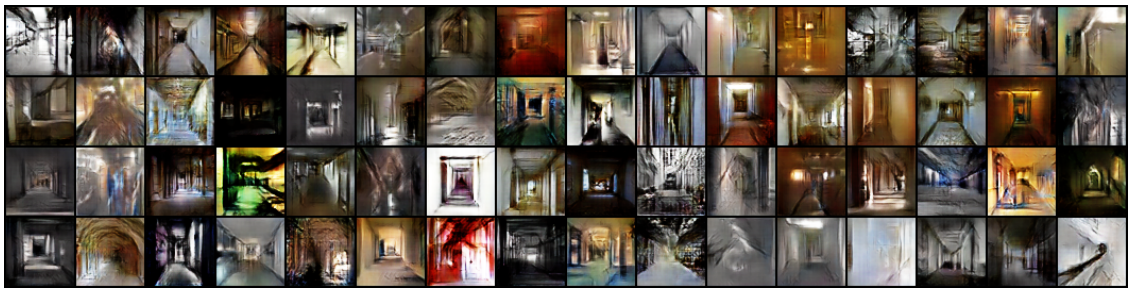

Figure 4.5: DCGAN trained as WGAN-gp after 35000 iterations (corridors)

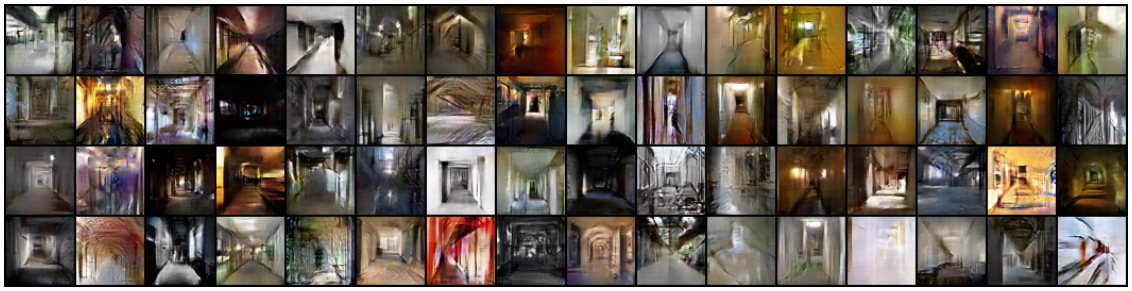

Figure 4.6: DCGAN trained as WGAN-gp after 97500 iterations (corridors)

Figure 4.7: WGAN-gp resnet (corridors)

4.3 StyleGAN2-ada

As seen in Figure 4.8, Figure 4.9 and Appendix A, the model did a better job at both types of images that the previously evaluated architectures. The model which generated the corridor images reached a lower fid (6.994 vs. 10.899), but that was expected since the corridor images are mostly of empty corridors and the amusement park contains many more images that are unique. Figure 4.10 shows the fid50k of the generators after training for X thousands images. Training for longer could produce better results.

Figure 4.8: StyleGAN2-ada results (amusement park)

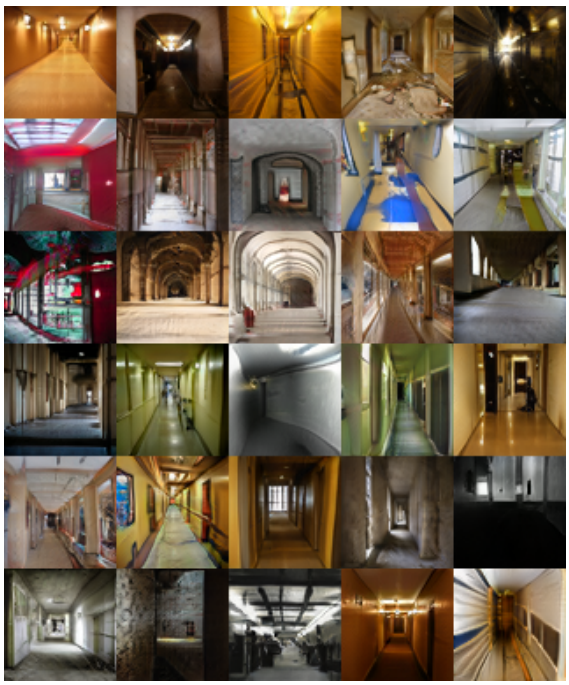

Figure 4.9: StyleGAN2-ada results (corridors)

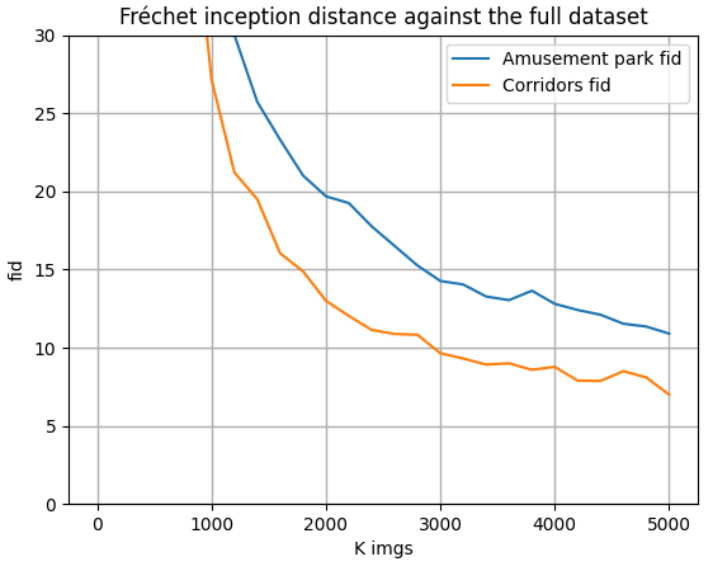

Figure 4.10: fid scores of the images generated by stylegan2-ada

Chapter 5

Discussion

5.1 StyleGAN2-ada vs. DCGAN trained as WGAN-gp

StyleGAN2-ada produced significantly better results than the rest, and there are many possible reasons and combinations of reasons for that.
First if we look at the critic loss of the DCGAN network trained as a WGAN-gp network we see that after a few tens of thousands generator iterations the loss of the critic begins to fall, see Figure 5.1 and Figure 5.2. In the paper "Improved Training of Wasserstein GANs" by Ishaan Gulrajani et al. it is mentioned that the critic is overfitting faster than the generator. The critic overfitting being a problem in all networks but the StyleGAN2-ada would make sense because of the adaptive discriminator augmentation which makes overfitting less of a problem. To check the impact of overfitting on the networks StyleGAN2 without adaptive discriminator augmentation could be compared with the current StyleGAN2-ada results on the same dataset.

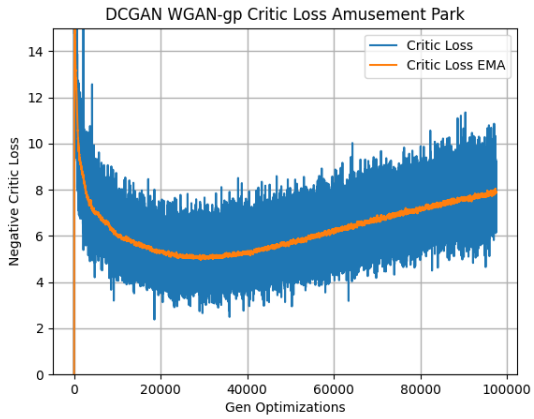

Figure 5.1: Critic loss with exponential moving average of the DCGAN network trained as a WGAN-gp network on amusement park images

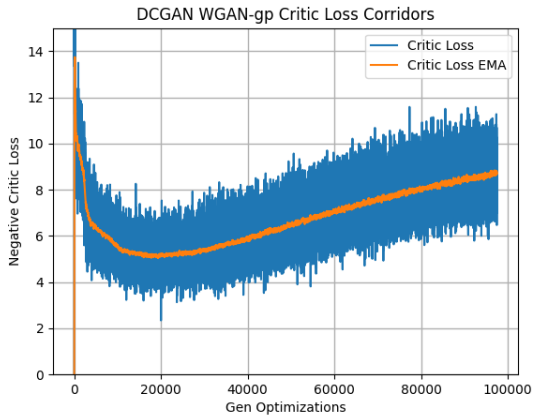

Figure 5.2: Critic loss with exponential moving average of the DCGAN network trained as a WGAN-gp network on corridor images

A second this that could cause the StyleGAN2-ada to perform better is perhaps that the architecture of the networks themselves are better suited to the task.

Another explanation is the loss function of StyleGAN2-ada which uses not only gradient penalty but also R1 regularization and penalty for high gradient in output relative to latent space. The number of channels in the convolution layer of the StyleGAN2-ada network used is also bigger than the DCGAN-gp network, increasing the amount of channels in DCGAN

could cause improved results.

It is also possible that DCGAN with WGAN-gp loss would perform better with some other hyperparameters which were not found during the project.
To see why StyleGAN2-ada performed better evaluating StyleGAN2 without adaptive discriminator augmentation, evaluating DCGAN with a loss more similar to that of StyleGAN2, increasing the number of channels in DCGAN, adding adaptive discriminator augmentation to DCGAN and doing a hyperparameter search.

Chapter 6

Conclusions

The network which showed the best performance was the StyleGAN2-ada. Using the Wasserstein-1 distance improved the result of the DCGAN by some degree, although the results were still significantly worse than StyleGAN2-ada. It is not clear what part of StyleGAN2-ada made it work better than the rest.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[2] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[3] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028 [cs.LG].

[4] Xun Huang and Serge Belongie. *Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization*. 2017. arXiv: 1703.06868 [cs.CV].

[5] Nathan Inkawhich. *DCGAN Tutorial*. URL: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html (visited on 12/08/2021).

[6] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE].

[7] Tero Karras et al. *Analyzing and Improving the Image Quality of StyleGAN*. 2020. arXiv: 1912.04958 [cs.CV].

[8] Tero Karras et al. *Training Generative Adversarial Networks with Limited Data*. 2020. arXiv: 2006.06676 [cs.CV].

[9] NVlabs. *StyleGAN2-ADA — Official PyTorch implementation*. URL: https://github.com/NVlabs/stylegan2-ada-pytorch (visited on 12/08/2021).

[10] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

Appendix A

StyleGAN2-ada more results

A.1 Amusement park

A.2 Corridors

Appendix B

StyleGAN2-ada layer list

Generator	Parameters	Buffers	Output shape	Datatype
mapping.fc0	262656	-	[32, 512]	float32
mapping.fc1	262656	-	[32, 512]	float32
mapping	-	512	[32, 10, 512]	float32
synthesis.b4.conv1	2622465	32	[32, 512, 4, 4]	float32
synthesis.b4.torgb	264195	-	[32, 3, 4, 4]	float32
synthesis.b4:0	8192	16	[32, 512, 4, 4]	float32
synthesis.b4:1	-	-	[32, 512, 4, 4]	float32
synthesis.b8.conv0	2622465	80	[32, 512, 8, 8]	float16
synthesis.b8.conv1	2622465	80	[32, 512, 8, 8]	float16
synthesis.b8.torgb	264195	-	[32, 3, 8, 8]	float16
synthesis.b8:0	-	16	[32, 512, 8, 8]	float16
synthesis.b8:1	-	-	[32, 512, 8, 8]	float32
synthesis.b16.conv0	2622465	272	[32, 512, 16, 16]	float16
synthesis.b16.conv1	2622465	272	[32, 512, 16, 16]	float16
synthesis.b16.torgb	264195	-	[32, 3, 16, 16]	float16
synthesis.b16:0	-	16	[32, 512, 16, 16]	float16
synthesis.b16:1	-	-	[32, 512, 16, 16]	float32
synthesis.b32.conv0	2622465	1040	[32, 512, 32, 32]	float16
synthesis.b32.conv1	2622465	1040	[32, 512, 32, 32]	float16
synthesis.b32.torgb	264195	-	[32, 3, 32, 32]	float16
synthesis.b32:0	-	16	[32, 512, 32, 32]	float16
synthesis.b32:1	-	-	[32, 512, 32, 32]	float32
synthesis.b64.conv0	1442561	4112	[32, 256, 64, 64]	float16
synthesis.b64.conv1	721409	4112	[32, 256, 64, 64]	float16
synthesis.b64.torgb	132099	-	[32, 3, 64, 64]	float16
synthesis.b64:0	-	16	[32, 256, 64, 64]	float16
synthesis.b64:1	-	-	[32, 256, 64, 64]	float32
---	---	---	---	---
Total	22243608	11632	-	-

Discriminator	Parameters	Buffers	Output shape	Datatype
b64.fromrgb	1024	16	[32, 256, 64, 64]	float16
b64.skip	131072	16	[32, 512, 32, 32]	float16
b64.conv0	590080	16	[32, 256, 64, 64]	float16
b64.conv1	1180160	16	[32, 512, 32, 32]	float16
b64	-	16	[32, 512, 32, 32]	float16
b32.skip	262144	16	[32, 512, 16, 16]	float16
b32.conv0	2359808	16	[32, 512, 32, 32]	float16
b32.conv1	2359808	16	[32, 512, 16, 16]	float16
b32	-	16	[32, 512, 16, 16]	float16
b16.skip	262144	16	[32, 512, 8, 8]	float16

b16.conv0	2359808	16	[32, 512, 16, 16]	float16
b16.conv1	2359808	16	[32, 512, 8, 8]	float16
b16	-	16	[32, 512, 8, 8]	float16
b8.skip	262144	16	[32, 512, 4, 4]	float16
b8.conv0	2359808	16	[32, 512, 8, 8]	float16
b8.conv1	2359808	16	[32, 512, 4, 4]	float16
b8	-	16	[32, 512, 4, 4]	float16
b4.mbstd	-	-	[32, 513, 4, 4]	float32
b4.conv	2364416	16	[32, 512, 4, 4]	float32
b4.fc	4194816	-	[32, 512]	float32
b4.out	513	-	[32, 1]	float32
---	---	---	---	---
Total	23407361	288	-	-