Image Generation using Generative Adversarial Networks

Sofie de Geus
Supervisor: Prof. Dr. Sandjai Bhulai

Vrije Universiteit Amsterdam
Research Paper Business Analytics
May 2019

Abstract

Generative Adversarial Networks (GANs) have achieved impressive results for many different applications. This paper focuses on creating a GAN for image generation using Keras. The goal is to generate new unique data that resembles the original data. This technique consists of two neural networks, a generator and a discriminator, which are simultaneously trained. The generator tries to capture the data distribution and generate a realistic 'fake' data point, while the discriminator needs to determine whether it is fake or real. The stability and convergence of a GAN are highly dependent on the network architecture and the selection of hyper-parameters. Therefore, different experiments were performed, first testing the effect of different network architectures, second testing the effect of hyper-parameter optimization and third the effect of different initializations. To compare results the Frechét Inception Distance (FID) is used which is shown to be consistent with human judgment of visual quality and can also detect mode collapse.

1 Introduction

Nowadays technology is an essential part of our lives. Machines are programmed that can do tasks that typically require human intelligence. In deep learning, artificial neural networks are used that are inspired by the human brain. Deep learning is used for various tasks like speech recognition, image recognition, playing board and video games and language translation.

Nevertheless, deep learning suffers from some limitations. For example, when creating a discriminative model, which models the decision boundary between the classes, a large amount of labeled data is needed. In some cases, this can be difficult to obtain and will be time-consuming. Also for tasks where human creativity is needed, think of generating new data, there are some issues. Generative models are typically used in unsupervised learning and try to model the actual distribution of the data. Although deep neural networks are good at classifying, they are not good at creating new things.

As a solution, Ian Goodfellow introduced the Generative Adversarial Network (GAN) in 2014 [7]. This is supposed to give machines some 'imagination' [5]. The goal of a GAN is to generate new unique data that resembles the original data. This new technique consists of two neural networks, a generator (G) and a discriminator (D), which are simultaneously trained. The generator tries to capture the data distribution and generate a realistic 'fake' data point, while the discriminator needs to determine whether it is fake or real. As training proceeds both the generator and discriminator improve.

Director of AI Research at Facebook, Yann LeCun, called generative adversarial networks one of the most interesting ideas in machine learning at this moment. The potential of GANs is huge, because they can learn to model any data distribution. This is not only useful for images, as shown in Goodfellow's paper, but also for video's, music, speech, text and more. Since Goodfellow introduced the original GAN in 2014, many GAN-related papers have been written about different applications. They have achieved impressive results in image generation [4], image completion [9], super resolution [13], text to image [16], image to image translation [10] and face aging [1].

Creating a GAN with optimal results for different applications has resulted in the different types of GANs that exist today. One of the most popular is the Deep Convolutional Generative Adversarial Network (DCGAN). To obtain the DCGAN multiple changes were made to the original framework by Radford et al. [15]. This paper focuses on creating a DCGAN for image generation using Keras. The network will be tested on the CelebA dataset containing celebrity faces. Different architectures and hyper-parameters are tested and compared.

2 Literature review

In 2014, Goodfellow et al. created the new framework of generative adversarial networks and compared it to multiple generative models. In this framework, two networks are simultaneously trained. Advantages of this new framework are that no Markov chains are needed and the system is trained on backpropagation. The adversarial nets were applied to a range of datasets including MNIST, the Toronto Face Database (TFD), and CIFAR-10. His paper showed the potential of GANs by showing they were competitive with the better generative models in the literature.

To improve image generation with generative adversarial networks, Denton et al. (2015) introduce a Laplacian pyramid framework. Here the image is generated by breaking the generation into multiple sections instead of immediately creating the image at once. Each level of the pyramid uses the results of the previous stage to better grasp the structure of the image. To compare the Laplacian Generative Adversarial Network (LAPGAN) to the original GAN framework, a human subject experiment was performed. Here the LAPGAN generated samples from CIFAR10 were mistaken for real images around 40% of the time. For the original GAN framework, this was only 10%.

Despite the successes of GANs, this process is highly unstable and does not scale well. Therefore, Radford et al. (2015) made some improvements to the original architecture of a GAN creating the Deep Convolutional Generative Adversarial Network. The DCGAN is one of the most popular and also most successful implementations of a GAN. One modification was replacing the fully-connected layers by convolutional layers, which are generally better at finding spatial correlations. Here convolutional strides are used for down-sampling and up-sampling. Another modification was using batch normalization to stabilize learning and cope with poor initialization.

Currently, many papers exist about different generative adversarial network applications not only image generation.

In Isola et al. (2017) Conditional Generative Adversarial Networks are used for image to image translation. In the CGAN, an additional vector of class information is given as input. Besides learning to map from input image to output image, the network also learns a loss function to train this mapping making it useful for multiple applications without modification. This means no parameter tuning or hand-engineered loss functions are needed. This approach shows good results for synthesizing photos from label maps, reconstructing objects from edge maps and colorizing images.

Also text to image translation is possible with the use of GANs. For this Reed et al. (2016) used a Deep Convolutional Generative Adversarial Network (DCGAN) conditioned on text features. With the conditional DCGAN, Reed

et al. successfully generated 64 x 64 pixel images for birds and flowers based on text descriptions. Another paper by Zhang et al. (2017) created a Stacked GAN (StackGAN) for text to image translation [17]. Here the problem was divided into sub-problems, where each stage has a different purpose. Stage one sketches the primitive shape and colors resulting in a low resolution image. Stage two uses these results and corrects their defects and adds more details. In this paper Zhang et al. succeeded in creating higher resolution images (256 x 256 pixels) with more realistic details.

Another interesting application of GANs is image completion. To obtain a good fully-convolutional neural network for this, Iizuka et al. (2017) looked at both local and global context, each having their own discriminator. Results were compared to multiple techniques and it showed significant improvements over a wide variety of scenes. Their approach was able to generate objects that do not yet appear in the image. Problems did occur when holes in the image were too large or were located at the edge of an image, but it is suggested this could be solved by more dilated convolutions.

Despite the successes and different applications, the training of a GAN remains very unstable. Multiple problems can occur during training. One important problem is unbalance between the generator and the discriminator. This could lead to non-convergence or overfitting. Another thing that can happen is the discriminator getting too successful causing the generator gradient to vanish and learn nothing. Mode collapse is also problematic for GANs, here the generator collapses and produces limited varieties of samples. Additional GANs are highly sensitive to hyper-parameter selections.

3 Dataset

An open source dataset with celebrity faces and attributes, called CelebA, is used in this research [18]. The dataset consists of 202,599 celebrity images with 5 landmark locations (eyes, mouth corners, nose) and each 40 binary attribute annotations. The original size of these images is 218 x 178 pixels with 3 channels for the RGB colors. A few of the celebrity images are plotted with their landmarks in figure 1.

Figure 1: Original images with landmarks

The images in the dataset were already aligned which can be seen in Figure 3. Here all landmarks are plotted in one figure. However, there still exist some background clutter around the face. For this research all image sizes are reduced to 32 x 32 pixels. By reducing the images the quality decreases as well which can be seen in Figure 2. However, this allows for a smaller model and therefore also less computation time. The dataset was divided into a train set (190,000) and a test set (10,000). The test set will be used to compare results from the models to the actual data.

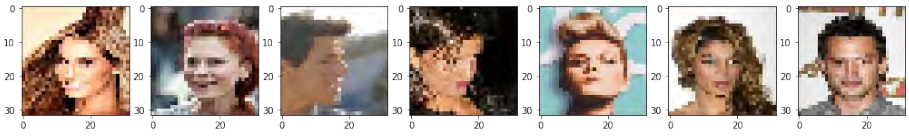


Figure 2: Resized images

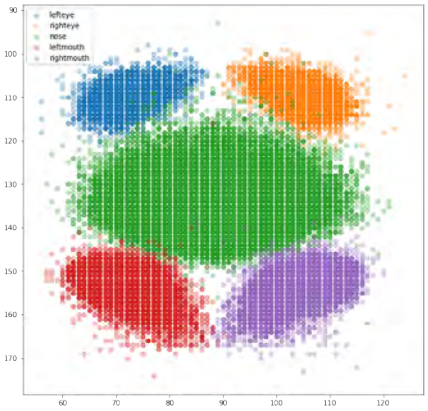


Figure 3: Plot of all landmarks

4 Methods

In this section, the methods and procedures used for our research are explained. The study focused on creating a GAN to generate new facial images similar to the CelebA dataset, in particular using a Deep Convolutional Generative Adversarial Network (DCGAN). Even when the metric is fixed, a generative network can achieve different scores when varying the architecture and/or hyper-parameters. Therefore, multiple DCGAN architectures were created and discussed in this section. Another thing discussed in this section is selecting appropriate hyper-parameters to obtain optimal results, which is an important part of training machine learning models.

4.1 Generative Adversarial Network

The original GAN, also called vanilla GAN, introduced by Goodfellow consists of two neural networks, a generator $G(z;\theta_g)$ and a discriminator $D(x;\theta_d)$, which are simultaneously trained. The generator takes noise vector z as input, which is a vector that follows a certain distribution $p_z(z)$. It tries to capture the distribution of the data and with this generates a realistic 'fake' image. The discriminator needs to determine the probability $D(x)$ whether this image x is from the data (real) rather than produced by the generator (fake). In this process, both networks are trying to optimize a different objective function/loss function which can be described as a two-player minimax game, see Equation 1.

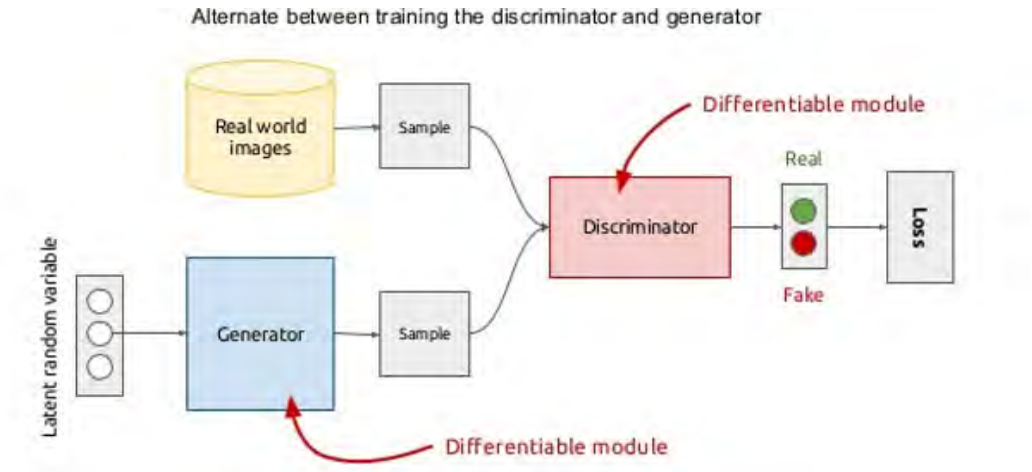


Figure 4: Generative Adversarial Network

In the original GAN, both generator and discriminator are defined as multilayer perceptrons, this represents a fully connected deep neural network, which Goodfellow noted as most straightforward. A neural network consists of an input layer, a number of hidden layers and an output layer. The entire system can be trained with back propagation and sampling from the generative model with forward propagation. No approximate inference networks or Markov chains are necessary [7].

z : noise vector $\quad G(z)$: generator output : x_{fake}

x : data sample : x_{real}

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

4.2 Deep Convolutional Generative Adversarial Network

The Deep Convolutional Generative Adversarial Network was introduced by Radford et al. to improve results for image generation. To do so some modifications were made to the original framework of a GAN [15].

One modification to the framework was introducing convolutional layers, here multi-layer perceptrons were replaced with ConvNets. This means the networks are no longer fully connected making them scale better. For a fully connected network the input image would first have to be flattened, as can be seen in Figure 5. In this new framework, convolutional strides are used for the down-sampling and the up-sampling, see Figure 6. This means the input images will no longer have to be flattened and therefore the spatial relationships in the data remain intact. This is particularly good for image generation.

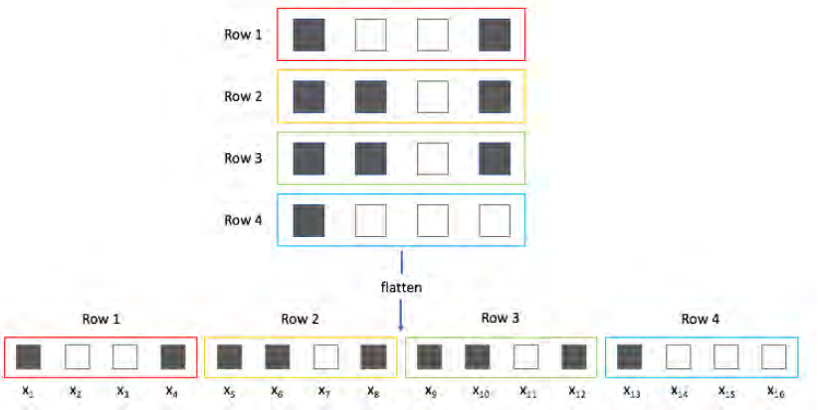


Figure 5: A flat representation of an image

Another modification was using batch normalization to stabilize learning. This overcomes issues with poor initialization and helps gradient flow in deeper models. It also helps to prevent mode-collapse which is problematic during training. When mode collapse occurs, the generating distribution collapses and only produces a few single modes. This results in generated samples with less variety than the data [3]. Now the generated samples are not a good representation because they do not capture the variety from the real data distribution. Kodali et al. (2017) suggest this is due to the existence of undesirable local equilibria [11]. Here sharp discriminator gradients occur around some real data points. To avoid sample oscillation and model instability the batch normalization was not applied to the generator output layer or the discriminator input layer.

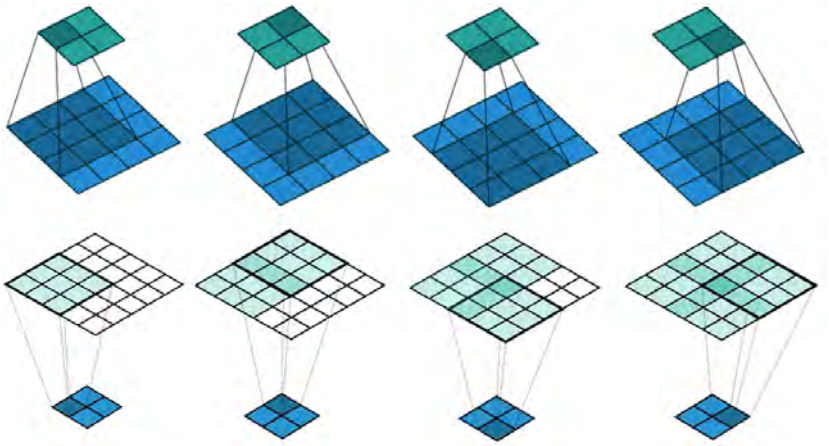


Figure 6: Convolutional strides for up- and down-sampling

Within the DCGAN there are multiple options for activation functions. For the generator, Radford et al. recommend using the ReLU activation except for the output layer which uses the Tanh function. Radford et al. explain a bounded activation allowed the model to learn more quickly to saturate and cover the color space of the training distribution. For the discriminator, a leaky ReLU is recommended.

The optimizer used is called the Adam optimizer which is popular in literature and also recommended/used by Goodfellow himself. The Adam optimization algorithm is an extension to stochastic gradient descent and specifically designed for training deep neural networks. This optimization algorithm uses multiple parameters that would have to be tuned.

4.3 Hyper-parameter search

A hyper-parameter search is very important since GANs are highly sensitive to the hyper-parameter selection. One can either use the best hyper-parameter settings found in literature, or try to optimize them. Lucic et al. (2018) state most models can reach similar scores with enough hyper-parameter optimization and random restarts [14]. The extensiveness of the search depends on the computation budget. Two of the most widely used methods for parameter optimization are grid search and random search. In this research, both grid search and random search are used to find optimal training hyper-parameters for the Adam optimizer (lr_g, lr_d, beta 1, beta 2).

4.3.1 Random Search

Random search is a very popular method for hyper-parameter optimization. Both grid search and random search are computationally expensive. However, in the paper of Bergstra et al. (2012) it was shown that random search outperforms grid search within a small fraction of computation time [2]. In random search each setting is sampled from a distribution over possible parameter values. Here in contrast to grid search a budget can be chosen independently of the number of parameters and possible values. Lucic et al. (2018) optimize by performing a random search on the training hyper-parameters within a chosen range, a similar setup will also be used in this research. This results in finding training parameters that work well for the given dataset and architecture.

4.3.2 Settings from related work

A brute force method like random search is very expensive and can waste time by trying out silly hyper-parameter configurations. Therefore, the best hyper-parameter settings found in literature were also used. To go through this specified subset of hyper-parameter space the grid search method is used. This means all possible combinations will be tested. An overview of these chosen values can be found in Table 3.

4.4 Metric

To compare different models and their performance a certain metric is needed. Within the generative adversarial network, it is measured how well the generator is fooling the discriminator. This however is not a good metric in measuring the image quality or its diversity compared to the original data. In this research, a recently proposed metric well suited to the image domain, called Frechét Inception Distance (FID), is used.

4.4.1 FID

The Frechét Inception Distance (FID) is a metric introduced by Heusel et al. (2017). To compute this metric, the assumption is made that the data follows a multivariate Gaussian distribution. For both the generated data and the original data, the mean and covariance are estimated.

$$X \sim \mathcal{N}(\mu_x, \sum_x)$$
$$X_{generated} \sim \mathcal{N}(\mu_g, \sum_g)$$

The FID is calculated by computing the Frechét distance between two Gaussians with the following equation:

$$FID(x, g) = ||\mu_x - \mu_g||_2^2 + Tr(\sum_x + \sum_g - 2(\sum_x \sum_g)^{\frac{1}{2}})$$

In the paper of Heusel et al. (2017) the effect of different types of disturbances (e.g., Gaussian noise, Gaussian blur, swirled images) on the FID are researched [8]. Figure 7 shows the influence of these disturbances. It shows that the FID captures the disturbance level very well and is consistent with human judgment of visual quality. Also important is that FID can detect mode collapse and therefore is a good measure for image diversity [14][12].

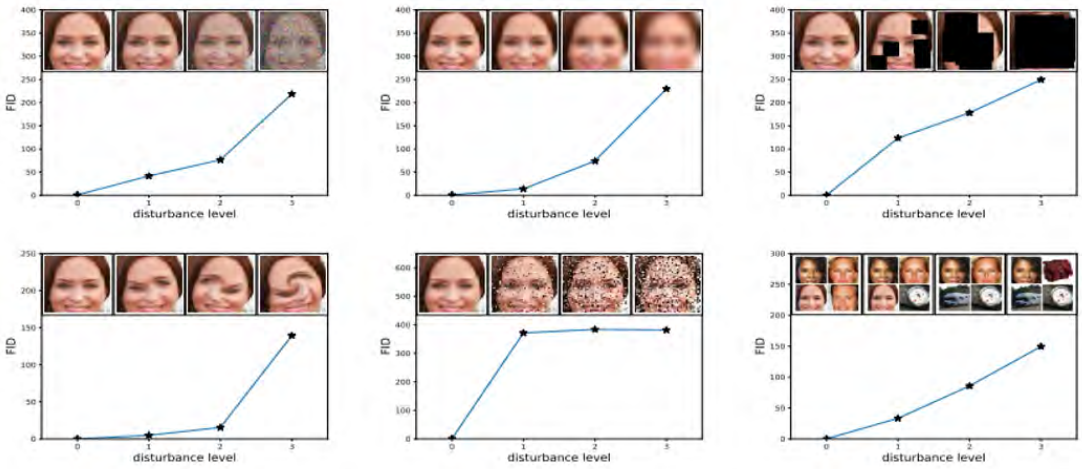


Figure 7: The effect of disturbances on FID

To calculate the FID, an existing code was used from GitHub [6]. An important thing to keep in mind is that the number of samples to calculate the Gaussian statistics should not be too small. Otherwise it can result in the covariance not being full rank. It is recommended that the minimum number of samples is 10,000.

5 Experiments

A generative adversarial network can achieve very different scores, when varying the architecture and hyper-parameters. To see which DCGAN performs best on the CelebA dataset, multiple architectures and hyper-parameters were tested. Three experiments were performed, first testing the effect of different network architectures (experiment 1), second testing the effect of hyper-parameter optimization (experiment 2) and third testing the effect of different initialization seeds (experiment 3). In this section, we discuss the experiments that were performed and their results.

5.1 Setup

For all models created in the different experiments, the number of epochs was set to 20,000. Throughout the 20,000 epochs the model generates multiple (65) evaluation sets which consist of 10,000 generated images to get a good idea about the sensitivity and instability of the model. To compare results, the FID is calculated between these generated images and the test set. For all models the min_fid and avg_fid is calculated. The noise input z for the generator is defined as a vector of length 100 following a normal distribution. Since it is a binary classification we use the binary crossentropy loss function.

The parameters from the Adam optimizer that were considered for optimization are the learning rate (generator, discriminator), beta 1 and beta 2. The learning rate used for the generator and discriminator does not have to be equal in the first experiment, but will be in the others. The beta's stand for the exponential decay rate for the first and second moment estimates. A similar setup from [14] was used.

5.2 Experiment 1: the effect of different network architectures

In the first experiment, the effect of different network architectures was examined. All networks consist of three hidden layers, but the hidden layers differ in depth. In total three different networks where chosen. The details of generator and discriminator structure are summarized in Table 1 and 2.

Layer	Layer	Kernel	Strides	Output
Input	z	-		100
Hidden 1	Linear, BN, ReLu	-		4 x 4 x G_{depth}
Hidden 2	Deconv, BN, ReLu	5	2	8 x 8 x $G_{depth}/2$
Hidden 3	Deconv, BN, ReLu	5	2	16 x 16 x $G_{depth}/4$
output	Deconv, Tanh	5	2	32 x 32 x 3

Table 1: Generator structure

In total three different model architectures will be compared. In one network the generator and discriminator have the exact same size. This is done by keeping the G_{depth} and D_{depth} equal in the structures from Table 1 and 2. Next a network where the generator is bigger than the discriminator ($G_{depth} > D_{depth}$) is created. And at last a network where it is the other way around ($G_{depth} < D_{depth}$). See a representation of these networks in Figure 8, 9 and 10.

Layer	Layer	Kernel	Strides	Output
Input		-		32 x 32 x 3
Hidden 1	Conv, LReLu	5	2	16 x 16 x $D_{depth}/4$
Hidden 2	Conv, BN, LReLu	5	2	8 x 8 x $D_{depth}/2$
Hidden 3	Conv, BN, LReLu	5	2	4 x 4 x D_{depth}
output	Linear			1

Table 2: Discriminator structure

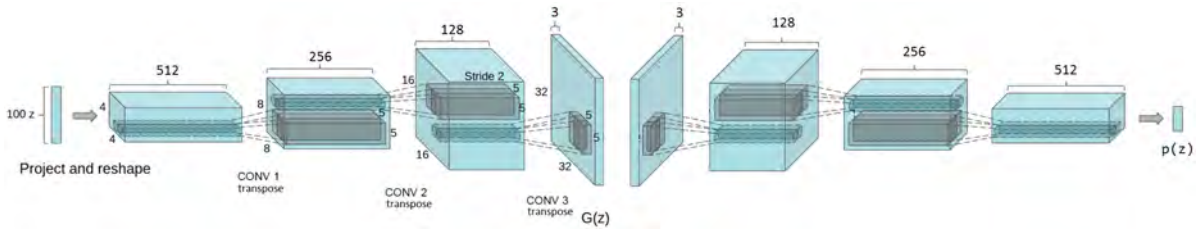


Figure 8: Model 512_512: Generator = Discriminator

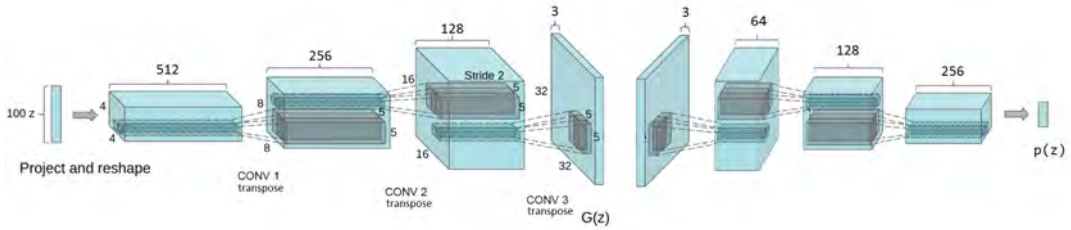


Figure 9: Model 512_256: Generator > Discriminator

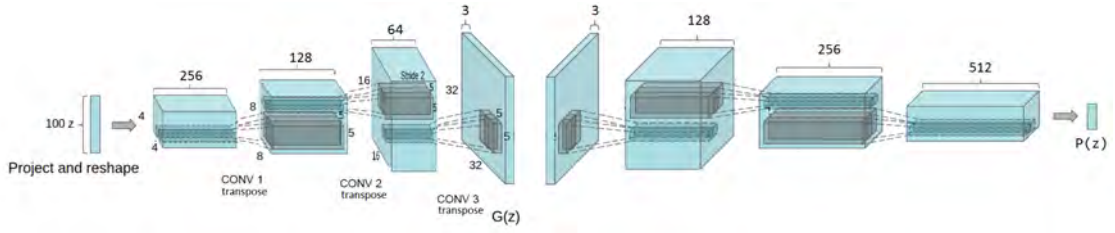
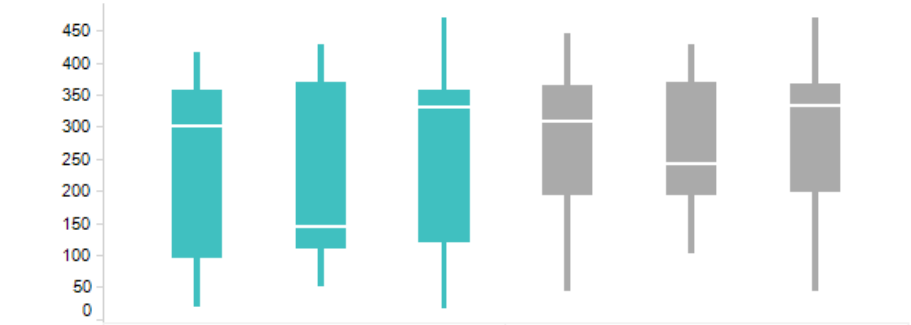


Figure 10: Model 256_512: Generator < Discriminator

Only using the default values for the parameters in the Adam optimizer might result in an incorrect conclusion. Therefore, for all three architectures multiple parameter settings that were obtained from literature were used. In Table 3 the hyper-parameter space is defined. For these values all possible combinations are used which results in 36 different parameter settings for this experiment. For all models the min_fid and avg_fid is calculated, this should give a good overview of the performance. In Figure 11 the results of this experiment can be found. Figure 12 shows the results per parameter value. These figures clearly show that model 512_256 performs better than the other models. Although this model might not have the lowest minimum, the median is significantly lower. Therefore, in the next experiments the architecture from Figure 9 is used. Apparently using a bigger generator than discriminator gives a better balance.

Parameter	Value		
lr_g	0,0002	0,0001	0,001
lr_d	0,0002	0,0001	0,001
beta_1	0,5	0,9	
beta_2	0,900	0,999	

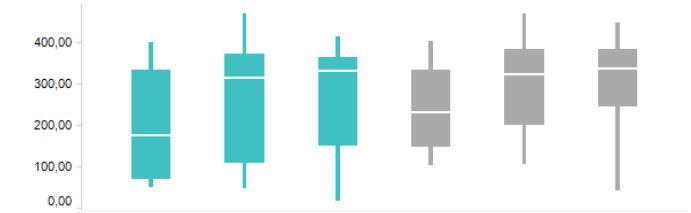
Table 3: Recommended parameter values from literature



	min_fid			avg_fid		
	256_512	512_256	512_512	256_512	512_256	512_512
Min	17,9161	49,8278	16,1392	42,5519	103,374	42,9479
Median	301,767	144,497	331,082	308,611	242,189	333,655
Max	415,633	429,344	470,336	447,088	429,792	470,781
Var	19037,2	18727,7	19085,9	12369,6	10048,6	12661,8

Figure 11: Boxplot of experiment 1

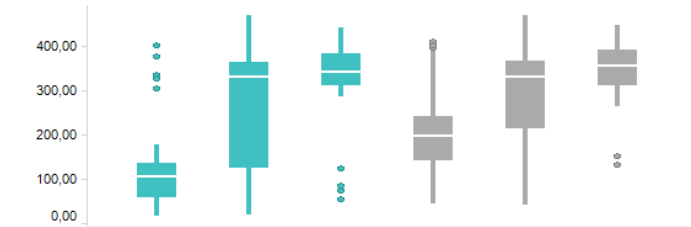
Lr_g



	min_fid			avg_fid		
	0,0001	0,0002	0,001	0,0001	0,0002	0,001
Min	49,8278	45,7788	16,1392	103,374	105,847	42,5519
Median	175,175	313,766	330,522	231,134	323,933	335,962
Max	400,001	470,336	415,923	404,801	470,781	447,088
Var	18574,2	21536,7	16447,4	10336,4	12197,9	10833,2

(a)

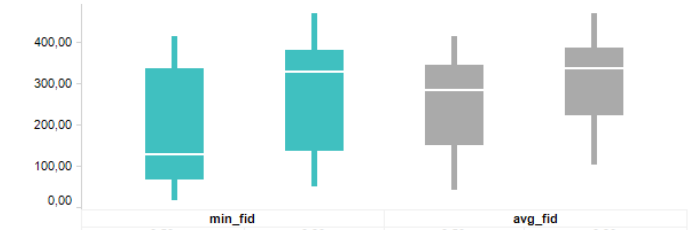
Lr_d



	min_fid			avg_fid		
	0,0001	0,0002	0,001	0,0001	0,0002	0,001
Min	16,1392	17,9161	56,3555	42,9479	42,5519	133,012
Median	106,751	331,595	343,265	198,473	331,882	355,882
Max	402,564	470,336	443,908	413,49	470,781	447,088
Var	9893,93	18860,9	9063,13	9137,35	11228,6	4633,14

(b)

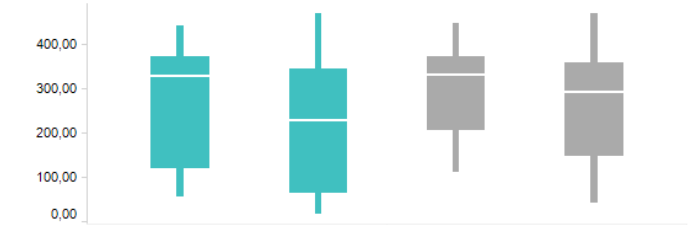
Beta_1



	min_fid		avg_fid	
	0,50	0,90	0,50	0,90
Min	16,1392	49,8278	42,5519	103,374
Median	127,948	329,198	284,46	337,471
Max	415,923	470,336	416,273	470,781
Var	19288,6	16837,3	11530,9	10458,2

(c)

Beta_2



	min_fid		avg_fid	
	0,9	0,999	0,9	0,999
Min	56,1532	16,1392	110,184	42,5519
Median	329,198	228,531	332,152	292,589
Max	443,908	470,336	447,088	470,781
Var	17790,2	20083,5	9253,42	13509,2

(d)

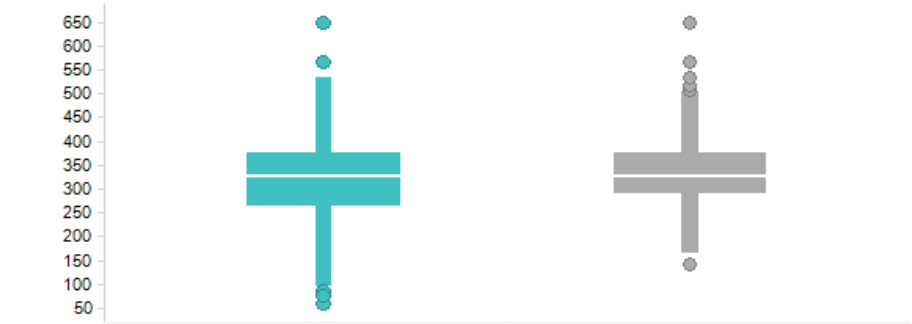
Figure 12: Boxplots of experiment 1 for all parameters

5.3 Experiment 2: the effect of different hyper-parameter settings

In the second experiment the effect of different hyper-parameters was examined. For this, the best architecture from the first experiment was selected, which is where the generator is bigger than the discriminator. For this model, multiple hyper-parameters for the Adam optimizer were optimized by performing a random search. Due to otherwise bad results, the learning rate of both generator and discriminator are kept equal. For the random search, a similar experimental setup was used from Lucic et al. where a wide and narrow search is performed within a certain parameter range and distribution.

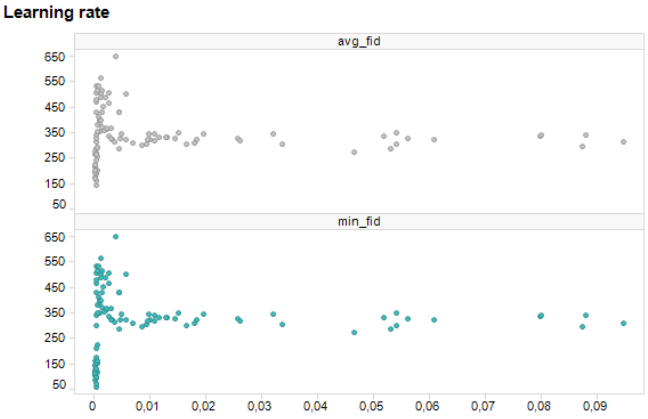
5.3.1 2a: Wide search

First, the wide setup was done where 100 samples of hyper-parameters were selected within a wide range. An overview of the chosen ranges and distributions for the parameters can be found in Table 4. Both beta's are sampled over a uniform distribution and the learning rates over a log scale distribution. For all different parameter settings the min_fid and avg_fid is calculated. The sensitivity of the model to hyper-parameters can be seen in the boxplot of Figure 13. Figure 14 is created to select better ranges for the next part of the experiment. This contains a scatter plot showing the FID against the parameter values. For the learning rate, the lowest FIDs occur at a lower value. This corresponds with the results from the first experiment in Figure 14a and 12b. Because of this, the lowest range is chosen for the learning rate. For the two beta values, the scatter plots give no obvious pattern. Therefore, the choice for these ranges is based on results from the experiment 1. The boxplots in Figure 12 show that for beta_1 and beta_2 the optimal value is 0.5 and 0.999, respectively.

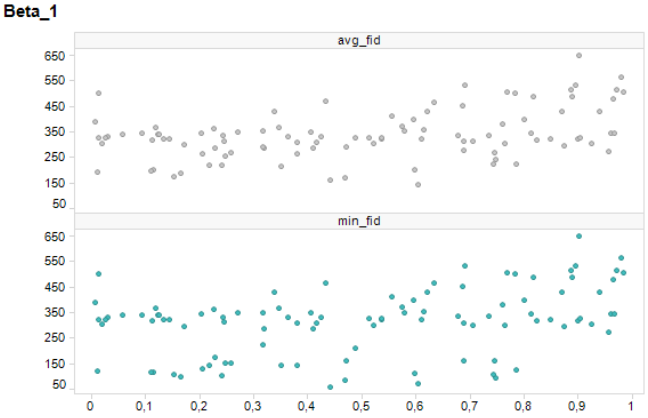


	min_fid	avg_fid
	512_256	
Min	59,9775	143,808
Median	326,128	327,076
Max	649,006	649,006
Var	16456,5	9355,04

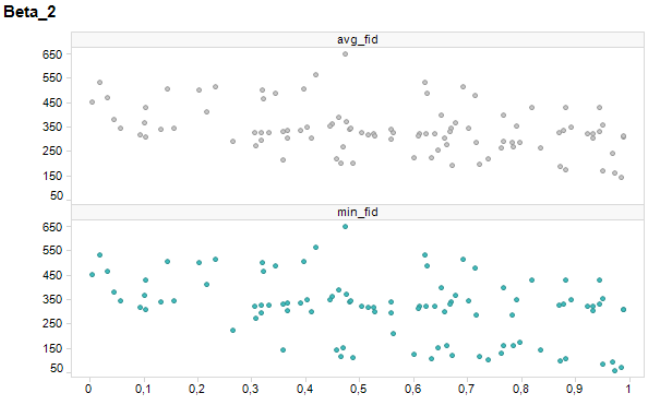
Figure 13: Boxplot of experiment 2a



(a)



(b)



(c)

Figure 14: Scatter plots of experiment 2 for all parameters

Parameters	min	max	distribution
lr_g, lr_d	10e-5	10e-2	L
beta_1	0	1	U
beta_2	0	1	U

Table 4: Wide parameter ranges

Parameters	min	max	distribution
lr_g, lr_d	10e-5	10e-4	L
beta_1	0,5	-	U
beta_2	0,999	-	U

Table 5: Narrow parameter ranges

5.3.2 2b: Narrow search

Next, the narrow setup was done where 50 samples were selected from a more narrow range derived from experiment 2a. These ranges can be found in Table 5. Again for all settings the min_fid and avg_fid is calculated. The boxplot representation can be seen in Figure 15. These new parameter settings result in a significant better FID score compared to experiment 2a.

	min_fid	avg_fid
	512_256	
Min	44,8591	108,898
Median	68,6373	158,197
Max	423,814	424,336
Var	10641,9	7712,62

Figure 15: Boxplot of experiment 2b

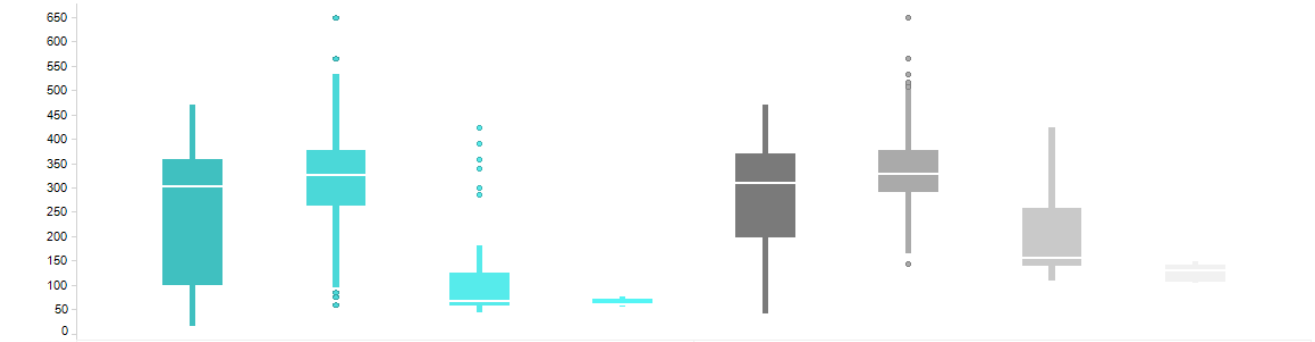
5.4 Experiment 3: the effect of different initialization seeds

After these experiments, the best architecture with its optimal hyper-parameter setting was found. To get an idea of the stability of training and convergence rate of one unique model, it was re-run 10 times with different initialization seeds. The seeds were used to create unique initial network weights, which means every starting point of the model will be different. The results from running this so-called best model resulted in the values from Table 6. It shows that even keeping the architecture and hyper-parameter settings equal, the model converges differently.

	CelebA
min_fid	64.71 $\pm$ 8.48
avg_fid	126.30 $\pm$ 18.76

Table 6: mean FID with standard deviation of best model

In Figure 16 an overview of all experiments can be found. This shows the improvements certain architecture choices and hyper-parameter choices had on the resulting FIDs. It can be seen that with these choices the variance reduces, which can be seen as less instability. Also the median FID improves. The resulting generated images can be seen in Figure 17. Here the faces are recognizable although some of them are not of the best quality. However they look similar to the resized images from the test set.



	min_fid				avg_fid			
	1	2a	2b	3	1	2a	2b	3
Min	16,1392	59,9775	44,8591	56,4527	42,5519	143,808	108,898	104,937
Median	301,767	326,128	66,4955	61,3795	308,611	327,076	156,481	129,566
Max	470,336	649,006	423,814	76,4933	470,781	649,006	424,336	148,76
Var	19116,5	16456,5	10205,5	71,9372	11616,9	9355,04	7625,17	351,99

Figure 16: Boxplot of experiment 1, 2 and 3

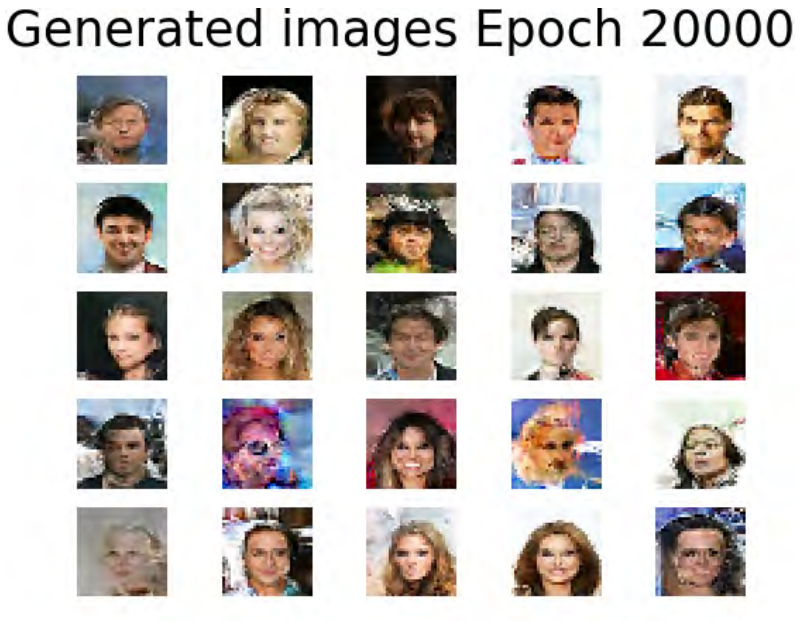


Figure 17: Generated images after 20,000 epochs

6 Conclusion

These experiments confirm that even when the metric is fixed, a given algorithm can achieve very different scores, when varying the architecture, hyper-parameters and random initialization. Only a few choices for architecture and hyper-parameter settings were used in this experiment. When more computation time is available, a more extensive research can be done. Think of other types of GANs and different architecture sizes and hyper-parameter settings. But also different choices are available concerning the activation functions, optimizers, metrics and even loss functions, which might improve results.

Ideally the generated images would also have the same size as the original dataset. This would however result in bigger and more complex networks. This would be possible when more computation time and power is available.

It was found that the most important thing during training is to have a good balance between the generator and discriminator. In this research, a model with a bigger generator than discriminator outperformed the others. This can be explained by the problem of diminished gradients. When the discriminator gets too successful compared to the generator, the generator gradient vanishes and learns nothing.

A thing that could prevent this and help for training the discriminator is using soft and noisy labels. Having hard labels (1 or 0) kills learning early on, to soften these labels fake images could use a random number close to zero (e.g., 0.1) and real images a value close to 1 (e.g., 0.9). Adding noise to the labels could be done by randomly flipping a percentage of labels, i.e., labeling real as generated and generated as real.

Because problems occur during training it might also be good to dynamically adjust training when unbalance is detected. That means if one side is winning, steps should be taken to prevent this. This might be possible by adjusting learning rates or the number of iterations.

References

[1] Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay. Face aging with conditional generative adversarial networks. 2017.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[3] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. *arXiv preprint arXiv:1612.02136*, 2016.

[4] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a Laplacian pyramid of adversarial networks. 2015.

[5] Martin Giles. The GANfather: The man who's given machines the gift of imagination. *MIT Technology Review*, February 2018.

[6] Github. fid code. *https://github.com/bioinf-jku/TTUR*.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. 2014.

[8] 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. *https://arxiv.org/pdf/1706.08500.pdf*, 2017.

[9] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and locally consistent image completion. 2017.

[10] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. 2017.

[11] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. On convergence and stability of GANs. *arXiv preprint arXiv:1705.07215*, 2017.

[12] Karol Kurach, Mario Lucic, Xiaohua Zhai, Marcin Michalski, and Sylvain Gelly. The GAN landscape: Losses, architectures, regularization, and normalization. *arXiv preprint arXiv:1807.04720*, 2018.

[13] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. 2017.

[14] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are GANs created equal? a large-scale study.

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

[16] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[17] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks. 2017.

[18] Xiaogang Wang Ziwei Liu, Ping Luo and Xiaoou Tang. Large-scale celebfaces attributes (celeba) dataset. *http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html*, 2017.