computergraphics tübingen

Re-Implementation of Progressive Growing of GANs

Florian Grimm, Vanessa Kirchner, Andreas Specker

April 16, 2018

Winter Term 2017/2018
Practical Course on Computer Graphics

Re-Implementation of Progressive Growing of GANs

Florian Grimm, Vanessa Kirchner, Andreas Specker

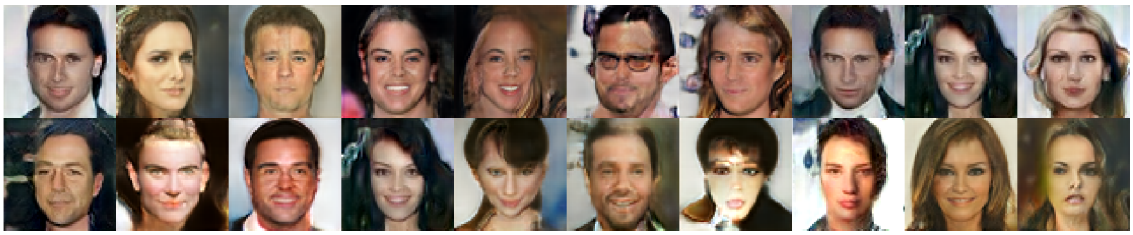

Figure 1: Predicted outputs from the trained generative approach given different random input noise.

Abstract

Traditionally, data-driven approaches minimize a task-dependent objective function containing the sum of a data-fitting and a regularization term. In image processing one ideally seeks to encode the data distribution to ensure the synthesized output has a similar statistics to a natural image. To directly learn such an *image-prior*, Goodfellow et al. [GPAM$^+$14] propose *Generative Adversarial Networks* (GANs) where two neural networks are trained simultaneously by competing in order to produce data that is similar to a given training set. The trained model is able to generate synthesized data from noise and to distinguish between real and fake data. Unfortunately, GANs suffer from instability and limited variation. This project re-implements the work of Karras et al. [KALL17] that tries to overcome these issues by continuously increasing the resolution during the training process to gain reliable and stable results.

1 Introduction

Creating synthesized samples from a given data distribution outlines the goal of this project. For this purpose, a *Generative Adversarial Network* is composed and trained. It consists of a generative model G capturing the data distribution and a discriminative model D which returns the probability that the sample is taken from the training data rather than produced by G [GPAM$^+$14].

The discriminator compares synthetic images to real images to determine the loss (see Section 3.4). Based on the loss the optimizer (see Section 3.5) can adjust the nets' parameters to refine their results.

Our implementation uses the CelebA-HQ dataset produced by Karras et al. [KALL17] showing aligned faces from celebrities. This high-quality version of the CelebA dataset consists of 30.000 images with a 1024×1024 pixel resolution. As the original database contains images with highly differing resolutions and an extremely varying content, Karras et al. [KALL17] applied image processing steps to ensure a consistent quality. The result is composed of high-resolution images produced by a super-resolution network introduced by Korobchenko & Foco [KF17]. All pictures are centered and cropped to the facial region and afterward eye aligned to the same position.

To provide an easy access to this database during this work all data has been converted to several *Lightning Memory-Mapped Databases* (LMDBs) containing the different required resolutions.

Within this project, a set of images is generated which is hardly distinguishable from the used high-resolution celebrity database (see Figure 1) working just with randomly generated pixels as input and training both networks with correctly chosen parameters. By increasing the image size step by

Figure 2: Generator output with increasing resolution from 4×4 to 64×64 pixels.

step during the training process, high-quality results are achieved due to the simplicity of information at the beginning and the continuously adding of further details during transitions to the next resolution (see Figure 2).

2 Related Work

Learning-based approaches to produce novel samples from high-dimensional data distributions can be organized into three main categories – each with different benefits.
Autoregressive methods are able to produce sharp images but are slow to evaluate as for example PixelCNN [vdOKV+16]. Variational autoencoders (VAEs) are comparatively easy to train but create blurry results by reason of restrictions within the model [KW13]. The third approach are GANs [GPAM+14] which produce sharp and natural looking images but consist of limited variation and the training can be highly unstable especially when increasing the resolution of the predictions. These issues of GANs are tried to be resolved in the following approach.

3 Implementation

The training consists of several epochs with 600.000 iterations each trying to learn weights of different sized images. Starting with a low resolution of 4×4 pixels a latent vector of random values is taken as input. After this first initialization epoch, the training alternates between two phases, a *transition phase* and a *stabilization phase* which will now be described in more detail.

Transition Phase: We assume that generator and discriminator have established the learned layer after one stabilization phase. The next step is the introduction of a new image size. The resolution is doubled and two convolutional layers are added to the model. Since the new information may lead to unintended maladjustments, the layers' outputs are faded in smoothly with a slowly increasing emphasis as further described in Section 3.1. At the end of the transition phase, the network produces remarkable results but also some minor distortions and less elaborated details. For that reason, a stabilization phase is needed.

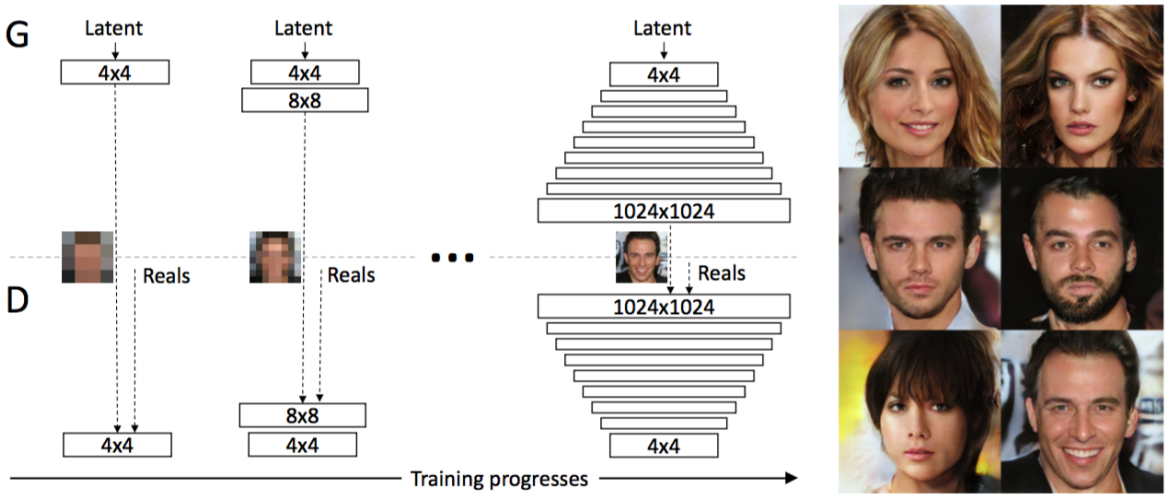

Figure 3: Introduction to the training procedure. Every rectangle marks a convolutional layer. The different layers are added to the generator (G) and the discriminator (D) as the training progresses (Figure taken from [KALL17]).

Stabilization Phase: During the stabilization phase, the new learned layer gets consolidated. In contrast to the previous phase, no new information is introduced to the model, the generator and discriminator's structure remains the same except for skipping the fade in.

After processing the two phases, the generator and discriminator's layers are established and the training-round for the next resolution starts. The generator and discriminator train each other mutually which means the generator tries to create natural looking images while the discriminator compares those synthesized results (see Section 3.4) with real data and returns feedback to the generator as shown in Figure 3. The generator evaluates the feedback (see Section 3.5) and tries to match the predicted output distribution. Thereby, the discriminator learns to distinguish real and fake images and the generator learns to produce more realistic images.

The generator and discriminator are implemented as shown in Figure 10. The increasing and decreasing output shapes as well as the size of the convolution layers are important. Within an epoch, pictures are resized in the generator and discriminator. Here, the nearest-neighbor resizing-algorithm is used as interpolation method for the upsampling and an average-pooling calculation for the downsampling procedure. Figure 10 also indicates where minibatch discrimination (Section 3.3) takes places.

This work has been implemented by using Python in combination with the open-source machine learning framework TensorFlow.

3.1 Transition-Phase: Fading In of New Information

To prevent the generator and the discriminator from being overcharged by new information, the next resolution layer is faded in smoothly. During the transition process where the images are upsampled in the generator (respectively downsampled in the discriminator) a linearly increasing $\alpha \in [0, 1]$ is used to weight the upsampled convoluted layer (see Figure 4). Thereby, the newly generated information is taken into account very little in the beginning and the generator and discriminator can acclimate to the next resolution step by step (see Figure 5). After the transition process, a *stabilization round* is performed to fine tune the weights for a given resolution.

Important during this process is the fade in of the reference images of the CelebA-HQ dataset into the discriminator. In this project, the observation is made that leaving this part out the generator stops producing reliable results at the transition to resolution 64×64 pixels.

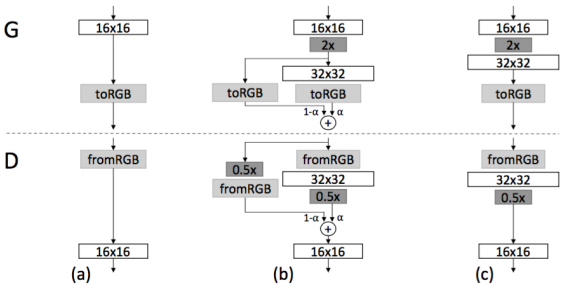

Figure 4: Smoothly fading in of new information (Figure taken from [KALL17]).

3.2 Pixelwise Normalization

The generator and discriminator's magnitudes of losses can get out of control due to the competition between them [KALL17]. In order to prevent this, a pixelwise feature normalization to unit length is performed, introduced by [KSH12]. A slightly adjusted version of this process is used in [KALL17]:

$$b_{x,y} = \frac{a_{x,y}}{\sqrt{\frac{1}{N}\sum_{j=0}^{N-1}(a_{x,y}^{j})^2 + \epsilon}} \tag{1}$$

where $a_{x,y}$ and $b_{x,y}$ denote the original and normalized feature vector of pixel (x, y), N is the amount of feature maps and ϵ is set to 10^{-8}. In this project, the normalization is realized step by step using the basic functions of TensorFlow before recognizing a pre-implemented function which calculates the mean of a specified tensor. As described in [KALL17] this process is applied after every convolution layer in the generator.

During the first implementations, the observation was made that this normalization has diverse effects concerning the use of varying optimizers (see

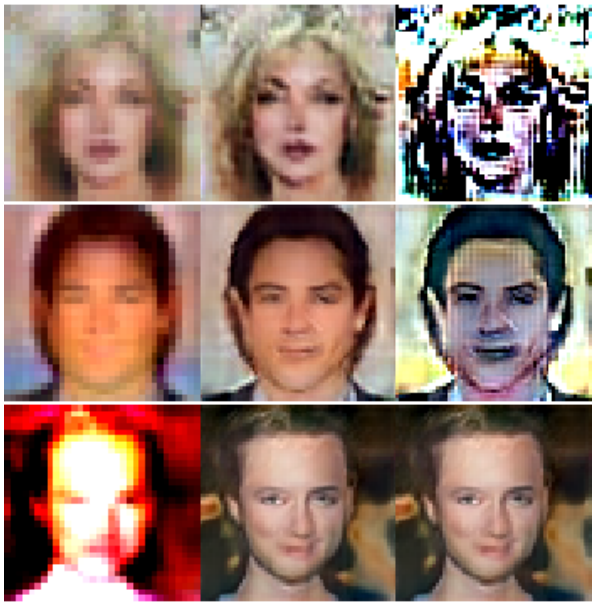

Figure 5: Transition between 32×32 pixels to 64×64 pixels as described in Section 3.1. Left: after 32×32 pixels step. Right: after 64×64 pixels step. Middle: Resulting image of 64×64 pixels. From top to bottom: $\alpha = 0.1, 0.5, 0.9$.

Section 3.5). Whereas the RMSProp Optimizer seems apparently not affected, the Adam Optimizer starts producing mostly uniform colored outputs.

3.3 Minibatch Standard Deviation

It may happen that the generator gets stuck in a state which is called *mode collapse*. In this state, predicted outputs tend to show very similar results - given different input noise - instead of producing a wide variety of different images, as gradients of similar points possess a similar orientation. The discriminator then favors a single point as a desired goal and due to the independent processing of the generator's product, there is no way to achieve a more dissimilar output [SGZ+16].

To gain a larger variation of the generated outcome, Salimans et al. [SGZ+16] suggested a procedure called *minibatch discrimination*. The discriminator takes multiple generated outputs into account instead of processing examples one by one. The concept has been simplified by Karras et al. [KALL17] by concatenating an additional feature map in the channel dimension at the end of the discriminator pipeline. This map is composed of the standard deviation for each feature in each spatial location over the minibatch, the average of these estimates and the subsequent replication of the resulting single value.

As a result, the diversity of generated images increases because the discriminator has more statistical information of the complete minibatch at disposal for his role as a critic. In this project, a function has been implemented which first calculates the variance and afterwards the square root of the variances mean. By replicating and reshaping this resulting value, the additional feature map is created and then appended to the existing vector.

3.4 Loss Functions

Optimizing G and D is a zero-sum game. Consequently, one loss increases while the other decreases, where the discriminator has obviously a way more easier task, because altered noise is simply separable from real data.

There follows a dilemma: On one hand, if the discriminator does not behave reasonably, no helpful feedback for the generator exists. On the other hand, if the discriminator works perfectly well, the gradient of the loss function converges quickly to zero and the training gets slow or does not even show any results. By reason of this, the conditions to calculate the costs for the zero-sum game are very important. As seen in Figure 6 the

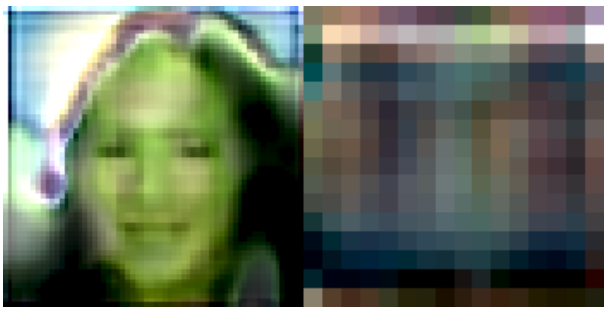

Figure 6: Left: The image shows that the generator was able to keep up with the discriminator for a rather long time but lost in the end. Right: The generator was not able to compete with the discriminator.

choice of the discriminator loss is very important and if chosen unreliable, no useful results will be generated.

Three different losses have been implemented in order to gain reliable results. A graphical visualization of the losses is shown in Figure 7.

Originally, GANs were introduced by Goodfellow et al. [GPAM+14] using the Jensen-Shannon (JS) divergence [Lin91]. The task is to compare the distribution of generated and real data. To overcome weaknesses like small supports between the distribution of generator data and the distribution of real data on low dimensional manifolds, discussed by Arjosky and Botou [AB17], *Wasserstein GANs* (WGANs) were introduced by Arjosky et al. [ACB17]. These provide a smooth representation of the distance between distributions in low dimensional manifolds. The loss is defined as:

$$L = \underset{\tilde{x}\sim\mathbb{P}_g}{\mathbb{E}}[D(\tilde{x})] - \underset{x\sim\mathbb{P}_r}{\mathbb{E}}[D(x)], \tag{2}$$

where $\mathbb{P}_r$ is the data distribution of real data, $\mathbb{P}_g$ the model distribution implicitly defined by $\tilde{x} = G(z)$, with z as uniform noise.

3.4.1 Gradient Penalty Loss

The WGAN in its original form suffers from enforcing the Lipschitz continuity by clipping the discriminator weights into a certain range. But as always, clipping values leads to an information loss. If the limit is too small, the gradient can vanish. Choosing a large limit the weights may take a long time to reach their limit. In order to remove this problem Gulrajani et al. [GAA+17] proposed a gradient penalty (*WGAN-GP loss*):

$$L' = L + \lambda \underset{\tilde{x}\sim\mathbb{P}_{\tilde{x}}}{\mathbb{E}}[(||\Delta_{\hat{x}} D(\hat{x})||_2 - 1)^2], \tag{3}$$

where L' is the new loss, L represents the *Wasserstein loss* and λ is set to 10. $\mathbb{P}_{\tilde{x}}$ are uniformly sampled lines between pair of points sampled from $\mathbb{P}_r$ and $\mathbb{P}_g$. This increases the loss of the generator depending on the current weights. The additional loss term keeps the discriminator within the Lipschitz constraint.

3.4.2 Drift Loss

An additional loss term is used in order to ensure that the output values will not drift too far away from zero and that the training is running more stable [KALL17]:

$$L' = L + \epsilon \underset{\tilde{x}\sim\mathbb{P}_r}{\mathbb{E}}[D(x)^2], \quad \epsilon = 0.001 \tag{4}$$

However, the drift loss term is contributing only very little compared to the other loss terms.

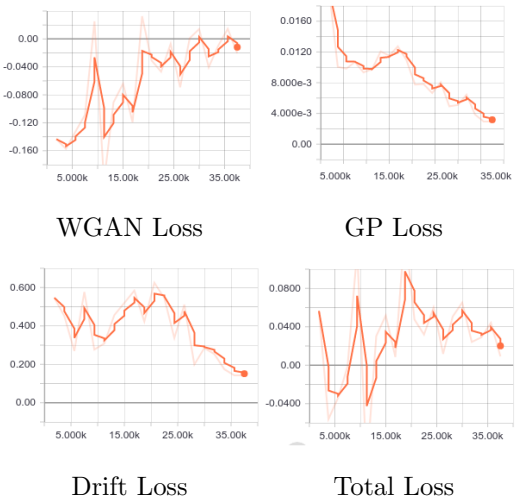

Figure 7: Graphs of single loss terms and the total loss, computed by adding the three weighted single loss terms.

3.5 RMSProp Optimizer

In the field of deep learning optimization algorithms are used to find helpful weights by mostly performing either a minimization or a maximization of a certain function. In order to gain smaller losses during the training process, the recently developed *Adam Optimizer* [KB14] is used in the paper of Karras et al. [KALL17]. This algorithm takes so-called hyper-parameters to adjust the optimization process to the specified problem [KB14]. Karras et al. propose a modified parameter tuning compared to the default settings proposed by Kingma [KB14] with $\alpha = 0.001$, $\beta_1 = 0$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$ which means they do not use any learning rate decay [KALL17].

Another common optimization algorithm is the *RMSProp Optimizer*. Whereas the updates of the Adam Optimizer are estimated by using a running average of the recent gradients moments, the RMSProp Optimizer uses a momentum on the

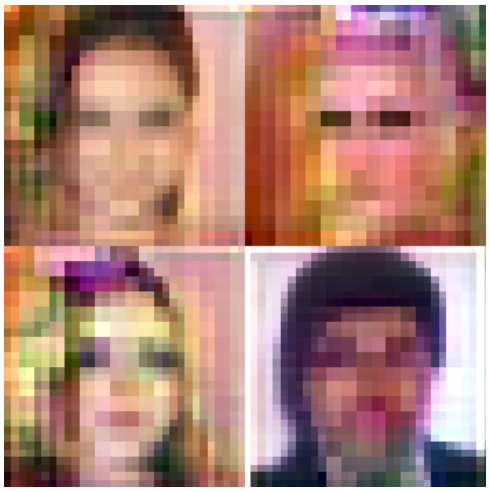

Figure 8: Four generated 32×32 pixel images which are not quite reliable but the eyes are always placed prominently at the same location.

Training Net	Duration (hours)
4×4 (s)	0:47
8×8 (t)	1:20
8×8 (s)	1:10
16×16 (t)	2:26
16×16 (s)	2:05
32×32 (t)	5:43
32×32 (s)	5:27
64×64 (t)	15:10
64×64 (s)	14:23
128×128 (t)	32:05
128×128 (s)	29:58
128×256 (t)	67:13
256×256 (s)	58:13
total:	**236:00**

Table 1: Training time for all sub nets (s - stabilization, t - transition). Times were measured on a single-GPU setup using Nvidia Titan X.

rescaled gradients [KB14]. In addition to that, it has no bias-correction term which may lead to a large stepsize and divergence in certain cases.

Although the Adam Optimizer is meant to perform a superior job due to the main differences between both optimization algorithms, we observed that within our implementation the RMSProp Optimizer produces more reasonable results. Having the same framework and using the Adam Optimizer, the generated images begin to diverge from the reference during the transition from 4×4 pixel to 8×8 pixel resolution. This is not the case if the RMSProp Optimizer is used and all other parameters stay unchanged.

4 Results and Discussion

During this work, different approaches have been investigated, including diverse optimization algorithms, a varying amount of real pictures shown to the discriminator and different configurations of stabilization processes.

The major difference to the paper of Karras et al. [KALL17] is the usage of the RMSProp Optimizer as opposed to the Adam Optimizer. For the training we used the built-in TensorFlow method with an epsilon of 10^{-4} and the default parameter set. We noticed that the quality of the results is more stable compared to the proposed Adams hyper-parameter configuration of the paper.

Another result of this project is the observation about the stabilization process after a resolution change. Showing the discriminator ten times more images than shown by Karras et al. [KALL17] leads to reliable outputs but as well to paying the price of a drastically increased runtime. At this point, further investigations of a suitable trade-off may be useful (see Section 5.1). Tabular 1 shows the training durations for the individual training epochs with 600.000 iterations each.

The generator learns quickly where the eyes are positioned in the face. Figure 8 shows that even if the generated faces are not reliable, the eyes are always at the right position. This occurrence is most likely because of the structure of the real images where the eyes are pixel-perfect aligned.

During transition and stabilization rounds, upsampling and downsampling is performed. For the upsampling operation, the nearest neighbor resizing-method of TensorFlow was used whereas for downsampling the average pooling layer was performed. Including pixelwise normalization (see Section 3.2), minibatch discrimination (see Section

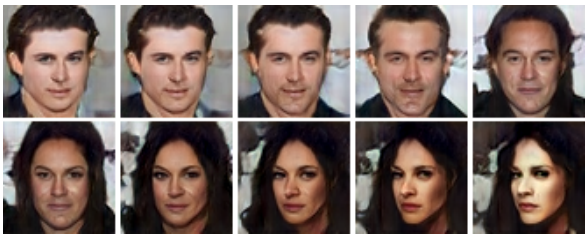

Figure 9: Instead of using noise $\in [-1, 1]$ as input for G, a linear evenly spaced vector from $[-0.25, 0.25]$ was used.

3.3), the fading in of new information (see Section 3.1) and the multiple losses (see Section 3.4) a GAN network has been trained during this project which generates reliable images of celebrities as it can be seen in Figure 1, Figure 9 and Figure 11.

4.1 Computational Effort

Since the whole training process consists of diverse training parts, intermediate results can be seen after a few hours of training. Starting with a low image resolution of 4×4 pixels and training the network up to 256×256 pixels, predictions at higher resolutions can be made relatively early. As shown in Figure 6, the training is inconclusively and will not bring better results. In comparison, Figure 11 shows real-looking synthesized faces and will bring realistic results also within higher resolutions. Changing the parameters results in starting the whole process anew which is rather time-consuming, especially if the changes are experimental to adjust several parameters.

5 Conclusion

A progressive growing of GANs stabilizes the training process a lot, mainly by starting with low resolutions containing less details and going up to high resolutions with fine details. In order to gain reasonable results, the parameters have to be chosen carefully and small changes in the network structure may lead to completely different synthesized results which can be far away from photorealism.

5.1 Future Work

Although, the quality of results can be estimated after a few hours it takes quite long to train higher resolutions such as 64×64 pixels. It would be useful to speed up the process in order to gain an efficient implementation. Furthermore, all images in the CelebA-HQ dataset show faces where the eyes are exactly located in the same place. So another issue would be to test diverse datasets containing for example objects such as pieces of furniture or faces that are not aligned.
Additionally, we made the observation that using $z \in [-1, 0]$ as latent input vector mainly results in male face whereas $z \in [0, 1]$ mostly shows females. This observation is worth further investigation.

References

[AB17] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. 2017.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. 2017.

[GAA+17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5769–5779, 2017.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[KALL17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. 2017.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[KF17] Dmitry Korobchenko and Marco Foco. Single image super-resolution using deep learning.

2017. https://gwmt.nvidia.com/super-res/about.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 2013.

[Lin91] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

[SGZ+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[vdOKV+16] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016.

A Results

Generator	Act.	Output shape	Params
Latent vector	–	512 × 1 × 1	–
Conv 4 × 4	LReLU	512 × 4 × 4	4.2M
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Upsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Upsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Upsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Upsample	–	512 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	1.2M
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Upsample	–	256 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	295k
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Upsample	–	128 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	74k
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Upsample	–	64 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	18k
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Upsample	–	32 × 1024 × 1024	–
Conv 3 × 3	LReLU	16 × 1024 × 1024	4.6k
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 1 × 1	linear	3 × 1024 × 1024	51
Total trainable parameters			**23.1M**

Discriminator	Act.	Output shape	Params
Input image	–	3 × 1024 × 1024	–
Conv 1 × 1	LReLU	16 × 1024 × 1024	64
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 3 × 3	LReLU	32 × 1024 × 1024	4.6k
Downsample	–	32 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Conv 3 × 3	LReLU	64 × 512 × 512	18k
Downsample	–	64 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Conv 3 × 3	LReLU	128 × 256 × 256	74k
Downsample	–	128 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Conv 3 × 3	LReLU	256 × 128 × 128	295k
Downsample	–	256 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Conv 3 × 3	LReLU	512 × 64 × 64	1.2M
Downsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Downsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Downsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Downsample	–	512 × 4 × 4	–
Minibatch stddev	–	513 × 4 × 4	–
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Conv 4 × 4	LReLU	512 × 1 × 1	4.2M
Fully-connected	linear	1 × 1 × 1	513
Total trainable parameters			**23.1M**

Figure 10: Overview of the generator and discriminator containing the shapes of the feature vectors and the size of the convolution layers (Figure taken from [KALL17]).

Figure 11: Results in different resolutions from 32×32 pixels to 256×256 from top to bottom. The higher the resolution the more contrast is visible, but also unrealistic elements increase. The last row shows pictures from the CelebA-HQ dataset as reference.