Technische Hochschule
Brandenburg
University of
Applied Sciences

Generative Adversarial Networks & Loss Function

Betreuer: Prof. Keller

Bentyn, Zofia Erasmus student	Zanderstr. 10E Brandenburg an der Havel zofia.bentyn@gmail.com Mart. Nr. 2022-5180
González Sonnenberg, Lucas Master business informatics 2. Semester	Walther-Rathenau-Str. 15 15834 Rangsdorf Gonzalez@th-brandenburg.de Matr. Nr. 2022-3933
Gutowski, Manina Master business informatics 2. Semester	Querstr. 7 14727 Premnitz Gutowski@th-brandenburg.de Matr. Nr. 2020-5234

Brandenburg, 17th of July 2022

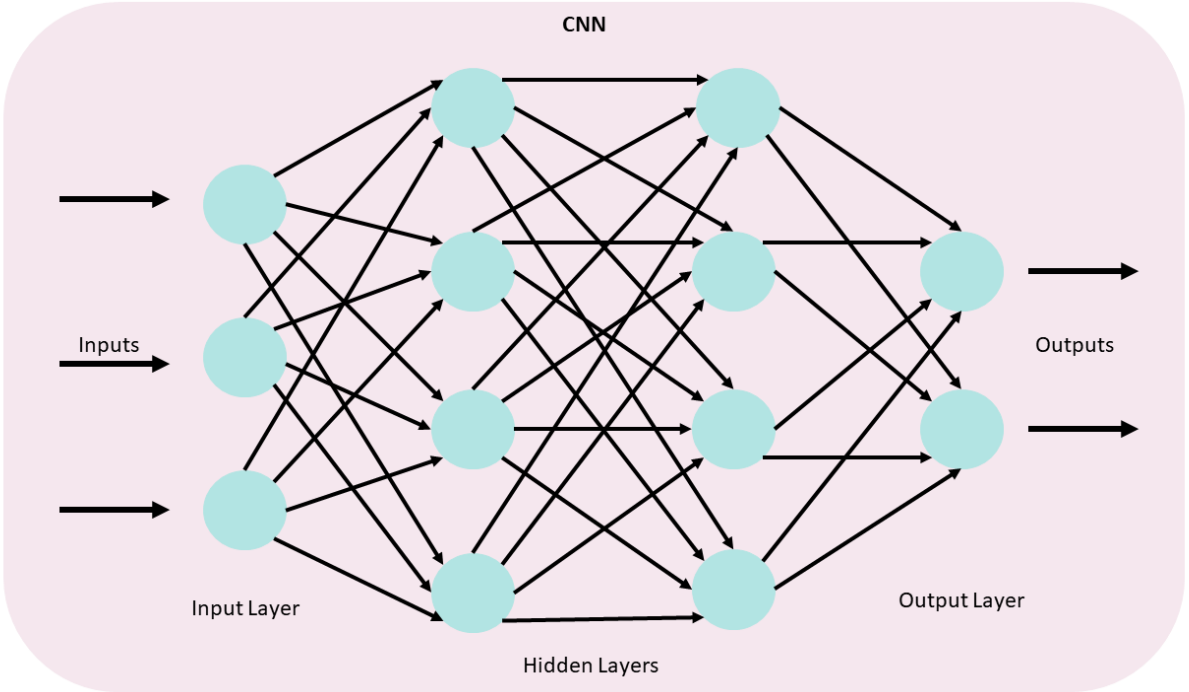
Table of contents

Abstract

This work describes the architectures of the Convolutional Neural Networks (CNN), Generative Adversarial Networks (GAN), Deep Convolutional Generative Adversarial Networks (DCGAN), the mathematics behind the GAN Loss function. Then we can see the results of the implementation of the equipment's own GAN. It can be seen that the training of the neural networks improves the quality of the artificial images obtained, however the final conclusion is that there is still room for improvement.

Convolutional Neural Networks (CNN)

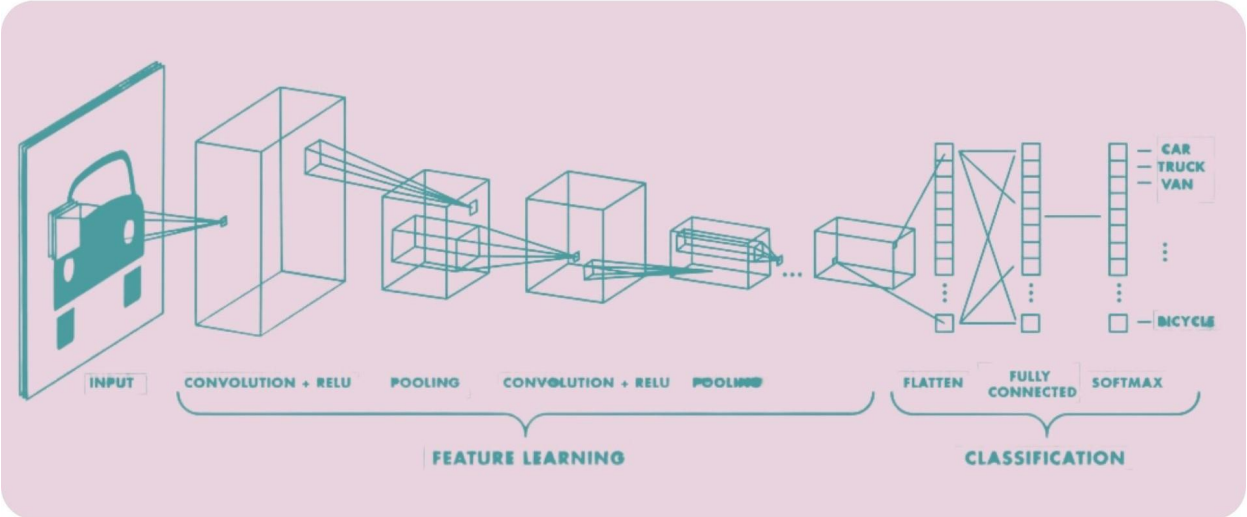
In this chapter we will present where Convolutional Neural Networks came from, what their building architecture looks like and how they work. The Convolutional Neural Networks have been used in image recognition since the 1980s. The increase in computational power, the amount of available training data and the optimized training mechanics for deep nets enabled it to achieve outrageous performance on some complex visual tasks.
The Convolutional Neural Network consists, like other neural networks, of an input layer, an output layer, and many hidden layers in between. They developed from the study of the neurons in the human brain, specifically the Visual Cortex. The analogous architecture is visualized below.



A Convolutional Neural Network works by receiving an image, defining it with some weightage based on different objects of the image, and then separating them from each

other. One benefit of the Convolutional Neural Networks is that it only requires little pre-process data as compared to other deep learning algorithms. Because it uses primitive methods for training the classifiers, which makes it good enough to learn the characteristics of the target object.

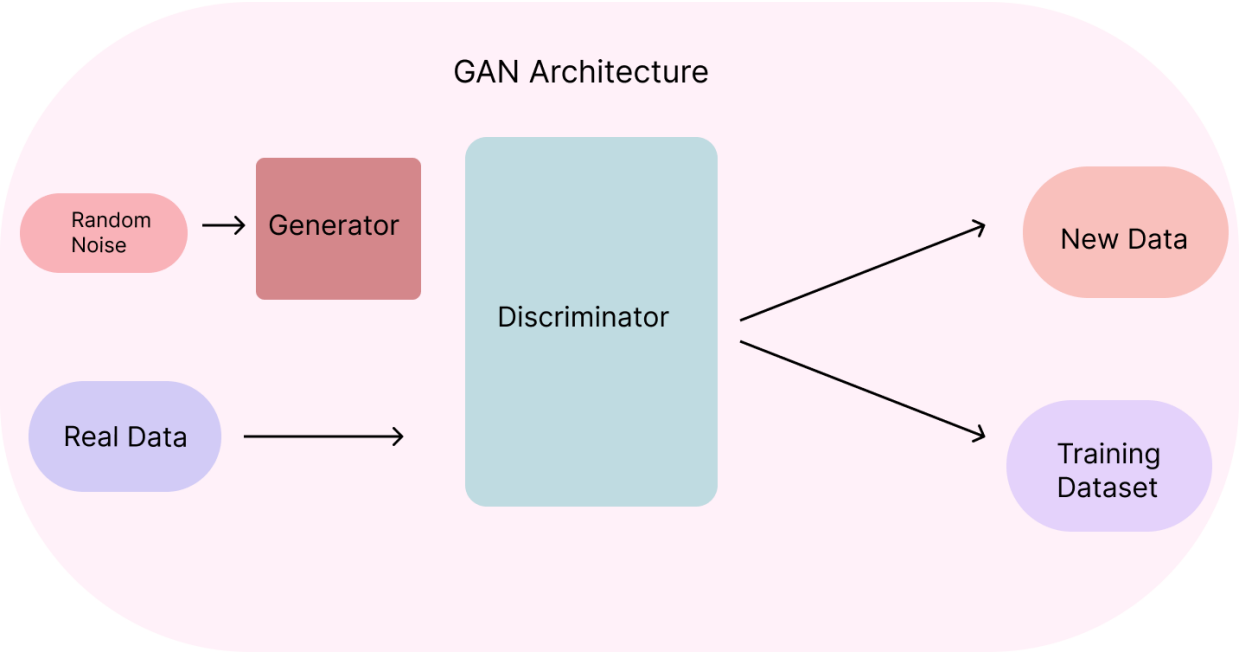
Also a Convolutional Neural Network can have hundreds of layers and each layer learns to detect different components of an image. Every training image will leach at different resolutions. The Output of each convolved image is also the input to the next layer. The filters start as simple components such as brightness and edges and will get to more complex features, which uniquely define the object. The process is visualized below.



The Convolutional Neural Networks are popular, because they don’t need manual feature extraction. They are learning the features directly. Also the recognition results are highly accurate and the convolutional neural networks can always be retrained for new recognition tasks.

Generative Adversarial Network’s (GAN)

The **GAN** (Generative Adversarial Network) was introduced for the first time in 2014 by Ian J. Goodfellow in the paper Generative Adversarial Nets. A GAN is a system of two neuronal networks. For the subject Predictive Analytics and



A GAN is a system of two neural networks, where each of them gets a name; the **Generator,** which is a generative model. It takes a probability distribution (random noise) and tries to generate a probabilistic output image. On the other side there is the **Discriminator**: Its job is to get data from two different inputs and it has to determine if the input consists of New Data generated with noise by the generator or if it consists of original data composed of the training dataset. Both neural networks are going to be trained as one system, however, they have opposite goals to achieve.

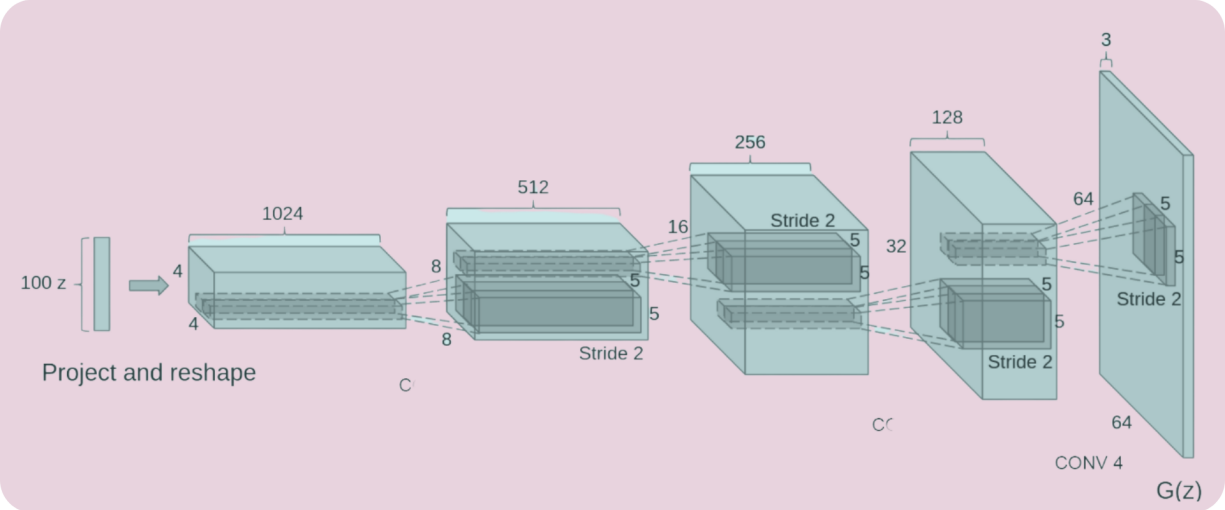
The architecture of GANS can be easily explained by taking the example of a banknote counterfeiter and a policeman, whose role is to detect which banknotes are genuine and which are counterfeit. The generator then plays the role of the counterfeiter, as it is the one who takes noise as input and transforms it into data. The generator is the policeman in charge of detecting which banknote is counterfeit and which is real. The ultimate goal of this neural network system is to train the generator well enough to be able to generate lifelike images. GANS can be trained for as many purposes as there is imagination, since their uses range from generating data for later use in a machine learning model, for which the data is not sufficient, to recreational and entertainment purposes.

Deep Convolutional Generative Adversarial Network (DCGAN)

A deep Convolutional Generative Adversarial Network is a direct extension of the GAN described above, except that it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator, which are described in the first chapter. The Convolutional Neural Networks was first described in the paper „Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks“.

The discriminator consists of strided convolution layers, batch norm layers and LeakyReLU activations. The input image has a ratio of 3x64x64 and the output is a scalar probability that the input is from the real data dispensation.

The generator contains convolutional-transpose layers, batch norm layers, and ReLU activations. The input is a latent vector, which is based on a standard normal dispensation, and the output is a 3x64x64 RGB image. The strided conv-transpose layers enable the latent vector to be converted into a volume with the same shape as an image. The network design for the generator is shown in the graphic below.



The deep Convolutional Generative Adversarial Network is so popular because it mainly consists of convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the downsampling and the upsampling.

Generative Adversarial Networks Losses

Binary Cross Entropy Loss/Log Loss

For binary classification, log loss is commonly used to evaluate how good the predictions are over two categories. High values are returned in cases of good predictions, and low values in cases of poorly predicted category. The mathematical equation for binary cross entropy is:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

The *y* is a category (0 or 1) and *p(y)* is the probability of an element being assigned to the given category. In total, the equation describes a mean of a sum of negative log probabilities over each element in the training sample. Negative log is needed to obtain a positive loss value overall, since log values between 0.0 and 1.0 are negative.

Minimax GAN Loss

To train two models of GAN we are using the minimax function presented by the equation:

$$min_G max_D V(D, G) = E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

The name indicates a two-player game, where competing parties are being simultaneously getting better. The generator tries to minimize the loss, while the discriminator tries to maximize the loss. E stands for expected value over all (real in the *x* case and fake in the *z* case) data instances, which also is called Nash equilibrium in this case.

Discriminator Loss:

Deriving from binary cross entropy, we achieve the following equation for real image $(y = 1)$ sample:

$$L(1) = -(1 \cdot log(D(x)) + (1 - 1) \cdot log(1 - D(x)) = -log(Dx)$$

and the following equation for fake image $(y = 0)$ sample:

$$L(0) = -(0 \cdot log(D(G(z))) + (1 - 0) \cdot log(1 - D(G(z))) = -log(1 - D(G(z)))$$

Combined, it gives us the formula for full loss, which we want to minimize:

$$L(Discriminator) = -[log(D(x)) + log(1 - D(G(z)))]$$

Finally, removing the negative sign changes the min function to max, so the complete discriminator formula (for a single data point) is as follows:

$$L(Discriminator) = max[log(D(x)) + log(1 - D(G(z)))]$$

Generator Loss:

Generator's loss is calculated based on the discriminator model. For the time of generator's training, the discriminator is frozen, therefore the input produced by generator is the only input being taken under consideration while calculating the loss.

The goal of this generator training is minimizing the chances of discriminator classifying generated images as “fake”. In the given formula, we can observe, that indeed generator’s loss consists only of the second term in the discriminator’s loss equation.

$$L(Generator) = min[log(1 - D(G(z)))]$$

Convergence

As the generator gets better in producing new data, the discriminator has more problems at distinguishing between real and fake data. This moment is described as convergence, and means that both models have reached Nash equilibrium. However, some cost functions might have issues with convergence using gradient descent.

It is important to mention, that convergence is not a stable state, as we will see in later documentation of our project.

Common complications in GAN training processes

Mode Collapse

Successful GAN is one, where the generator is reliable at generating data that confuses the discriminator, and generated data sample’s distributions are as diverse as the distribution of “real” data. The term mode refers to the output distribution.

Mode collapse is occurring while the second criteria is not met, which implies that generated data samples are not diverse. Too much generator training without updating the discriminator leads to collapsing/mapping many generator’s input values ‘z’ onto the discriminator output value x. In other words, Generator learns to associate a bunch of different inputs to similar output, which leads to diversity confusion.

Apart from comparing generated images between training cycle, one other way of spotting mode collapse is to plot model losses. In this case, rapid oscillations in the generator loss indicates a potential problem.

Non-convergence

Convergence failure occurs when model parameters are oscillating, destabilizing and never converging, meaning models losses do not find equilibrium. Depending on which model outperforms the other, we can distinguish between two cases.

Generator domination

This is the case, when the generator learns how to fool the discriminator fairly quickly in the training process, leading to poor image quality, although scoring high in terms of accuracy. This happens, because while the generator loss rises high, the discriminator loss is declining at the same pace, meaning the discriminator accepts most of the given data as “real”. There is a chance of recovery, but it is very unlikely, so it is most recommended stopping the training and introducing some shifts.
Example being, increasing the number of filters to enable discriminator to learn new features, or reducing the number of filters in the generator. Another way could be adding dropout layers to the generator.

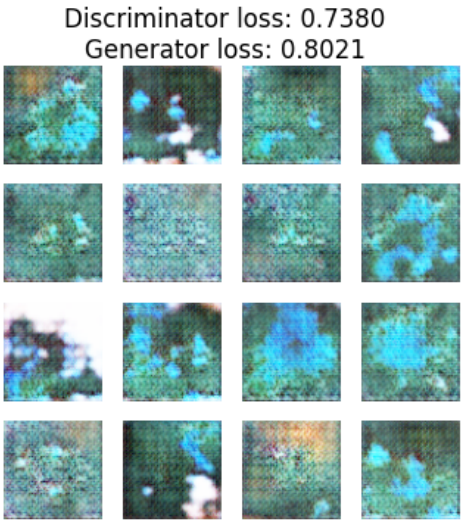
Discriminator domination

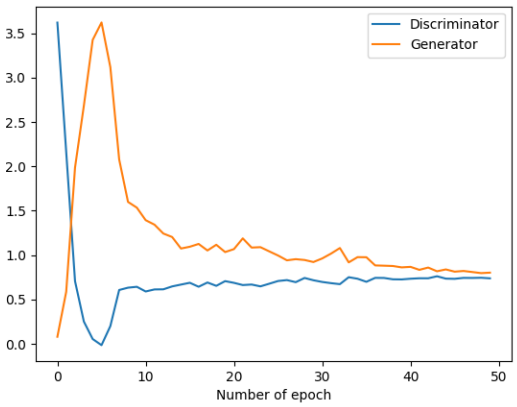
This happens when the discriminator reaches nearly one, making the generator fall back to near zero. Because the discriminator classifies all “real” images almost without a fail, the generator has no space to learn. To put it plainly, the discriminator does not provide enough information to train the generator, this case can be also called vanishing gradients problem. In backpropagation, as the gradient flows from the last layer to the first, it gets increasingly smaller. Small gradient influences learning rate, and slows down or stops the whole process from achieving better results.
The most common way of repairing such an event is one-sided label flipping, which means that we are on purpose feeding the discriminator some false labels to real images. Other than that, it could be useful to add dropout layers to the discriminator, or reduction of its filters.

Our Project

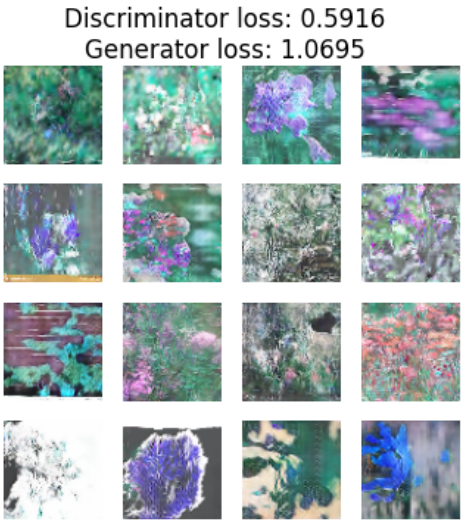
Model results

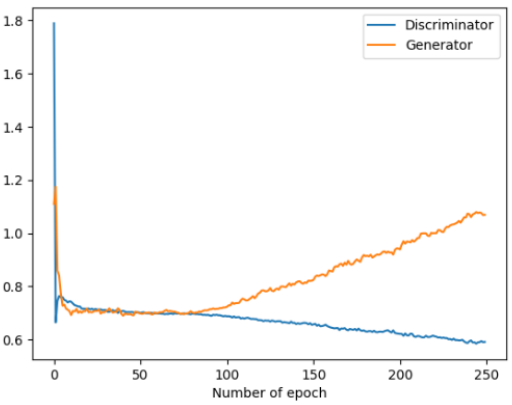
50 epochs, 500 real images





250 epochs, 3500 real images





Conclusions I

First try on 50 epochs run smoothly, we can see that models converge, and we achieve different outputs, that indicates that neither mode collapse, nor convergence failure has taken place.

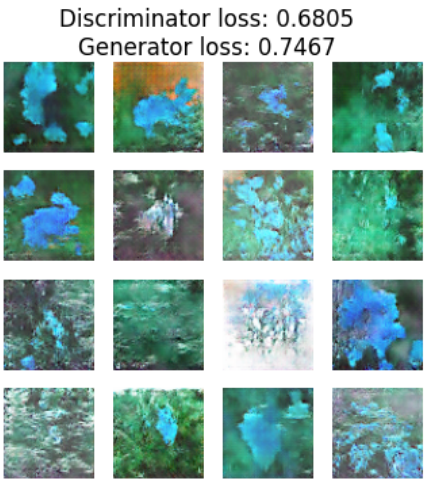
However, this is not the case in the 250 epochs try. Although the images from the second try look quite reliable, we can clearly see that there is no convergence between generator and discriminator.

After the 80th epoch things start going sideways, so we decided to rerun the program, however this time changing the value of latent space from 10 to 50. The latent space is simply a hypersphere and during learning process generator grasps how to efficiently map points onto the space.

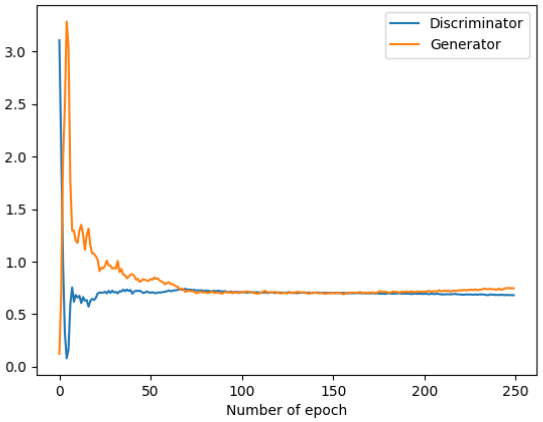
The results are much better, losses wise, although there still is room for improvement since, we can observe a slight change in convergence at the couple of last epochs. Presumably, larger number of epochs would unfold the nature of our model, but that is not available for us due to technical limitations. Nevertheless, we would argue that the image quality is far better from the previous run, since images are more precise.

250 epochs, 3500 real images, latent space 50

Model results



Model losses



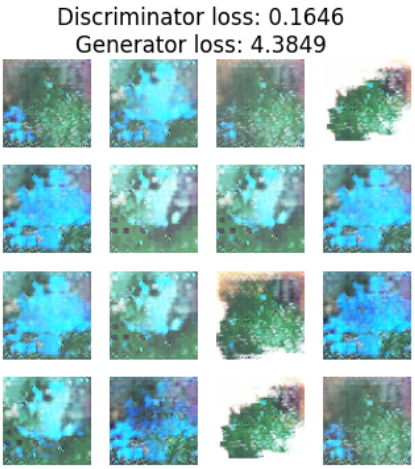
Imposed Mode Collapse

Finally, we wanted to perform one more check on our model, this time changing the latent space to the value of just 1.

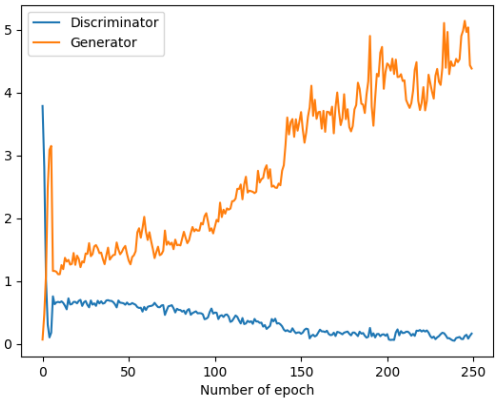
What we should achieve from this slight shift is imposed mode collapse, meaning generated data would be very similar to each other. This is not by any means a desirable result, since typically one would have to avoid such a mode failure, but we wanted to check if our suspicions on latent space value matched the reality.

250 epochs, 3500 real images, latent space 1

Model result



Model losses



Conclusions II

The results are as expected. Generator very quickly outrun discriminator and as the graph shows, there is no convergence in sight. That lead to creation of very low quality and similar images.

Literature

- Géron, A. “Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems Second Edition” O’Reilly
- Vasilev, I., Slater D., Spacagna, G. Roelants, P., Zocca V. “Python Deep Learning” Packt
- Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. “Generative Adversarial Nets”, Universite de Montreal,2014
- Ian Goodfellow, Yoshua Bengio, Aaron Courville, “Deep Learning”, The MIT Press, Cambridge, 2016
- https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf
- https://jonathan-hui.medium.com/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f
- https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
- https://arxiv.org/abs/1711.10337