Politechnika Warszawska

Wydział ...

Kierunek: ...

...

Adam Komorowski i Maciej Domagała

Warszawa, czerwiec 2021

Contents

Introduction

Chapter 1

Theory

1.1 Generative Adversarial Networks

1.1.1 Metrics

Inception Score

The Inception Score is a metric used especially for measuring the quality of Generative Adversarial Network outputs. It was first proposed in [1] and was used ever since to compare benchmark results of new generative models. Usually, objective function used in generative-discriminator type of network is representing how well one part of the architecture is performing in comparison to the other part e.g. how well discriminator is detecting generated images. This does not provide a measurable information about the quality and diversity of images though.

Inception Score was defined to measure two important aspects of generative networks performance - quality of generated images and their diversity. If network produces diverse images of good quality - the score is designed to be high.

Authors used the Inception network [2] to calculate the conditional label distribution for every created image. It is denoted as $p(y|x)$, where y is bounded to label and x is a single image instance. For images of high quality this distribution should have a *low entropy* characteristic - meaning that one class (correct one) should have a high probability score, while the rest should have relatively small scores. This indicates that Inception classifier has a lot of certainty what is presented on given image. To measure

diversity, authors proposed to calculate the marginal distribution of y by combining label distributions for a large set of images (50 000 is a proposed value). More formally, if z is a latent vector and $G(z)$ is an image generated from the vector, then marginal distribution

$$p(y) = \int_z p(y \mid x = G(z)) \ dz \tag{1.1}$$

should have *high entropy* (should be close to the uniform distribution) if model has a strong diversifying power.

Combining two above conditions, we can observe that best performing model would have different distributions of $p(y|x)$ and $p(y)$. To link these two discrete distributions together authors decided to use a Kullback-Leibler Divergence formula:

$$D_{KL}(p\|q) = \sum_{i=1}^{N} p\left(x_i\right) \cdot \left(\log p\left(x_i\right) - \log q\left(x_i\right)\right), \tag{1.2}$$

where p and q are two distributions for which we want to calculate relative entropy. Combining this formula with exponent (for easier comparison between different results) we obtain final Inception Score definition:

$$\mathrm{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x}} D_{KL}(p(y|x)\|p(y))\right) \tag{1.3}$$

Inception Score has several limitations (pointed out in [3]):

- Scoring is irreversibly bounded to Inception model and its training data. There could be low IS scoring for generating objects not in primary training data or using labels not incorporated by original model.
- Convolutional Neural Networks (and hence Inception model) rely more heavily on the local image textures and local features for classification. Therefore there might be less penalization for unintuitive image generation on the coarse level (e.g image of person with many legs and arms) than on the local level.
- This score only estimates the produced outputs and does not incorporate information about overall training data. E.g. it does not penalize memorizing and replicating training data.

Fréchet Inception Distance

Fréchet Inception Distance is another metric based in Inception model proposed in [4] as an alternative for Inception Score. One of the drawbacks of Inception Score, as pointed out by authors, is the fact that it does not capture the comparison of generated images to the real images. It does not refer to real data statistics when calculating final value.

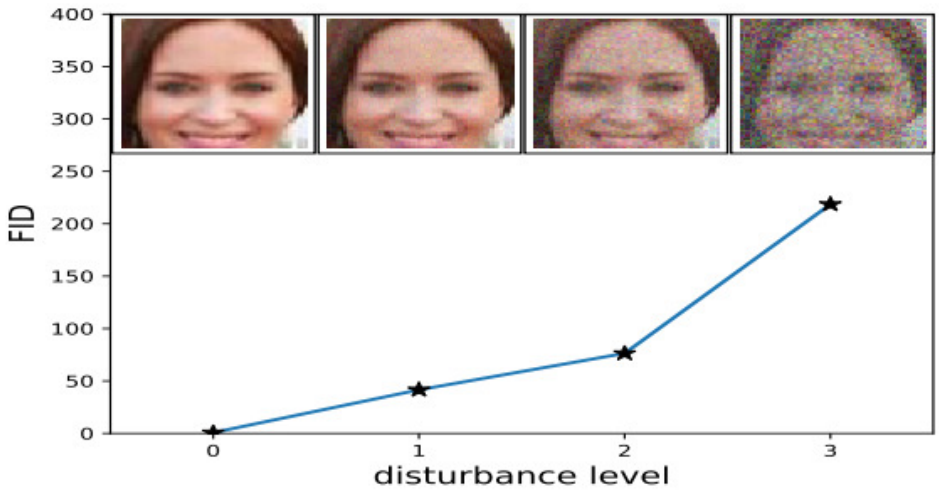

Figure 1.1: Plot of how FID scoring changes when Gaussian noise is added to the image.

Idea for the new metric is to use one of the last layers in the Inception model (pooling layer before classification output) to capture and encode specifics of an image. The output vector of this layer is of shape $(2048,)$ and is approximated by multivariate normal distribution. Set of these features is calculated for both real images and images generated by the model. We can denote the mean and covariance of embedded layer for generated images as, respectively, μ_r and Σ_r. Same properties for real images are denoted as μ_g and Σ_g. Final score is calculated as a distance between the two data distributions, formally defined as

$$\mathrm{FID}(r, g) = \|\mu_r - \mu_g\|_2^2 + \mathrm{tr}\left(\Sigma_r + \Sigma_g - 2\left(\Sigma_r \Sigma_g\right)^{\frac{1}{2}}\right), \tag{1.4}$$

where $tr(\dot{})$ stands for trace operator for matrix. Lower FID values indicate better generator performance, as the two distributions are *closer* to each other. This

score is more robust to noise than Inception Score - if generator only outputs small number of images per class (generated images are similar), the distance value between distributions will be high.

1.2 CLIP model

1.3 Evolutionary Algorithms

Evolutionary algorithms are a population-based heuristic methods of optimization. Algorithms from this family use computational implementations of processes related to natural selection, such as crossover, mutation and reproduction. Main idea of the algorithm is the *survival of the fittest* principle inspired by Darwinian evolution theory, where main objective is to generate more and more *fit* members of the population, while reducing the number of *weak* units. That *fitness* level is measured and described by *fitness function*, which determines the quality of each population sample.

In the following sections we will describe main algorithms of our interest that we will apply for latent search problem.

1.3.1 Genetic Algorithm

The Genetic Algorithm is one of the first methods used and defined as evolutionary algorithm. It is still used with success until today and the number of variations and different applications of the method still grows.

Standard genetic algorithm uses several operators for optimization purposes:

1. **Crossover** - process in which units from latest population are *mixed* together at random. In this way, offsprings that come from the parents have combined features from both parents. It works for intensification and diversification of search, although it depends on the type of crossover operator and the location of the parents in the search space.

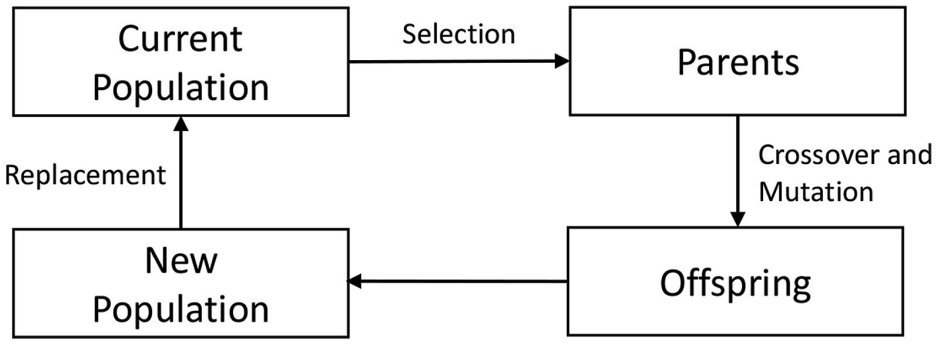

Figure 1.2: Genetic algorithm optimization loop.

2. **Mutation** - these operators are widely regarded as introducing some random disturbance for individual units in the population. Uniform mutation replaces the value of a single decision variable by a value that is randomly selected from a space within lower and upper bound for given variable. This mainly introduces diversification and also helps to escape local optima.
3. **Replacement** - after performing crossover and mutation for given population, newly generated offspring are evaluated by the fitness function. Several (this number is usually dependent on the algorithm hyperparameters) newly created members with best score are placed in the population, the same number of worst units is removed.
4. **Selection** - the most promising units from population (members with best results regarding the fitness function) are chosen for mating. This generally promises the best chance to generate better units in the next population.

The algorithm terminates based on several common criteria, most often when a solution is found that satisfies minimum criteria or when fixed number of generations is reached. Below is the pseudocode for the base genetic algorithm.

1.3.2 Differential Evolution

Differential evolution algorithm is a very efficient method most commonly used for optimization of function in continuous search space. It was proposed by Storn and Price in [5].

The main advantages over general genetic algorithm is more efficient memory usage, lower complexity and faster convergence.

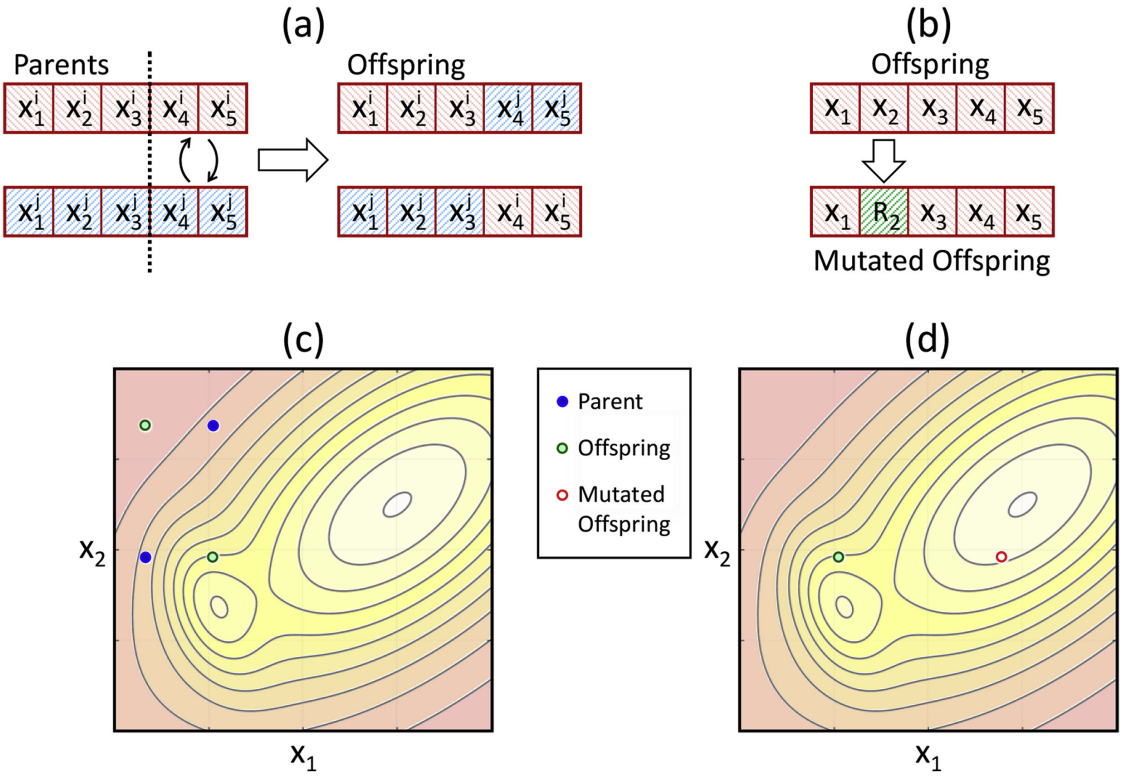

Figure 1.3: Example of crossover and mutation operations.

Algorithm 1 Genetic Algorithm

```
determine objective function (OF)
assign number of generation to 0 (t=0)
randomly create individuals in initial population P(t)
evaluate individuals in population P(t) using OF
while termination criterion is not satisfied do
    t=t+1
    select the individuals to population P(t) from P(t-1)
    change individuals of P(t) using crossover and mutation
    evaluate individuals in population P(t) using OF
end while
return the best individual found during the evolution
```

Figure 1.4: Genetic algorithm pseudocode.

Below we will present the mathematical formulation of the algorithm and discuss most popular versions.

Differential evolution is primarily described by three parameters: N_p - population size, C_r - crossover control parameter and F - scaling factor, also known as amplification parameter. Each member of every population is described as D-dimensional parameter vector. Each population in the algorithm can be understood as vector $\mathbf{x}_{i,g}$, where $i \in \{1, 2, ..., N_p\}$ and g stands for generation number.

Algorithm 2 Differential Evolution

```
1: determine objective function (OF)
2: assign number of generation to 0 (t=0)
3: randomly create individuals in initial population P(t)
4: while termination criterion is not satisfied do
5:     t=t+1
6:     for each i-th individual in the population P(t) do
7:         randomly generate three integer numbers:
8:         r1, r2, r3∈[1; population size], where r1≠r2≠r3≠i
9:         for each j-th gene in i-th individual (j∈[1; n]) do
10:            v_{i,j} = x_{r1,j} + F · (x_{r2,j} − x_{r3,j})
11:            randomly generate one real number rand_j ∈
    [0; 1)
12:            if rand_j < CR then u_{i,j} := v_{i,j}
13:            else
14:                u_{i,j} := x_{i,j}
15:            end if
16:        end for
17:        if individual u_i is better than individual x_i then
18:            replace individual x_i by child u_i individual
19:        end if
20:    end for
21: end while
22: return the best individual in population P(t)
```

Figure 1.5: Differential evolution DE/rand/1 pseudocode.

Method incorporates usage of three vectors, which we will name for simplicity:

- Donor vector, which is created in the mutation step,
- Trial vector, which is created in the crossover step,
- Target vector, which is the vector of current population.

In the **mutation** step donor vector $v_{i,g}$ is produced. It is calculated by adding the scaled difference of two vectors to the third vector from the population. There are two most popular variations of mutation used in differential algorithm, one is the *DE/rand/1* version, where all three vectors used in mutation are taken at random, which allows us to write a formula for donor vector $v_{i,g}$ as

$$\mathbf{v}_{i,g} = \mathbf{x}_{r_1,g} + F\left(\mathbf{x}_{r_2,g} - x_{\mathbf{r}_3,g}\right), \tag{1.5}$$

where $r_1, r_2, r_3 \in \{1, 2, ..., N_p\}$ are randomly selected indices and F is the aforemen-

tioned scaling factor.

The other choice is the *DE/best/1* version, which chooses two random vectors to calculate the scaled value which is used to change the **best** vector $\mathbf{x}_{best,g}$ in current population, which results in more greedy version of the method

$$\mathbf{v}_{i,g} = \mathbf{x}_{best,g} + F\left(\mathbf{x}_{r_2,g} - x_{\mathbf{r}_3,g}\right). \tag{1.6}$$

After the mutation algorithm performs **crossover** step. There are two most popular crossover schemes: binomial and exponential. Crossover step is performed using rate parameter $C_r \in (0,1)$, which determines size of perturbation of the target vector. This influences the population diversity.

In binomial crossover, the trial vector $\mathbf{u}_{i,g} = (u_{i,1,g}, u_{i,2,g}, \ldots, u_{i,D,g})$ for $i \in \{1,2,...,N_p\}$ and $j \in \{1,2,...,D\}$ is created using formula

$$u_{i,j,g} = \begin{cases} v_{i,j,g} & \text{if } rand_j \leqslant C_r \text{ or } j = j_{\text{rd}}, \\ x_{i,j,g} & \text{otherwise}, \end{cases} \tag{1.7}$$

where $rand_j$ is a random number from $(0,1)$ which determines the probability that the j-th parameter will crossover and $\mathbf{v}_{i,g}$ is the donor vector calculated in the mutation step. Also, j_{rd} is a randomly chosen integer in the range $[1,D]$, which ensures that at least one parameter will be chosen for crossover. That ensures that the new population will be always different from previous one.

For exponential crossover scheme, let us define the indices a and b to be chosen independently and at random from $[1,D]$. These parameters are chosen for each trial vector separately. Let's denote $()_{MOD_D}$ as modulo function with modulus D. Using these we can define a set of indices

$$I = \left\{a, (a+1)_{MOD_D}, \ldots, (a+b-1)_{MOD_D}\right\}. \tag{1.8}$$

The trial vector $\mathbf{u}_{i,g} = (u_{i,1,g}, u_{i,2,g}, \ldots, u_{i,D,g})$ for $i \in \{1,2,...,N_p\}$ and $j \in \{1,2,...,D\}$ is created as

$$u_{i,j,g} = \begin{cases} v_{i,j,g} & \text{if } j \in I \text{ and } rand_j \leqslant C_r, \\ x_{i,j,g} & \text{otherwise.} \end{cases} \tag{1.9}$$

Essentially, above formula is focusing on the crossover between neighbouring vector

features instead of fully randomizing the choice.
Having produced a trial vector we can proceed to the selection step. In this phase, algorithm is greedily choosing new members of the population using the *fitness function* f.

$$\mathbf{x}_{i+1,g} = \begin{cases} \mathbf{u}_{i,g} & \text{if } f\left(\mathbf{u}_{i,g}\right) < f\left(\mathbf{x}_{i,g}\right), \\ \mathbf{x}_{i,g} & \text{otherwise} \end{cases} \tag{1.10}$$

The algorithm terminates on similar basis as genetic algorithm discussed previously, when a optimal solution below certain treshold is found or when algorithm reaches some defined number of generations.

Chapter 2

Notable architectures

2.1 StyleGAN

StyleGAN is an architecture proposed by NVIDIA team in [6]. It is considered to be one of the most important publications regarding image generation and it introduces several novelties comparing to models used so far. It used several mechanisms (such as adaptive instance normalization and merging regularization) to generate highly realistic images with great resolution. It is greatly inspired by direct predecessor - Progressive Growing GAN architecture (called ProGAN), also published by NVIDIA.

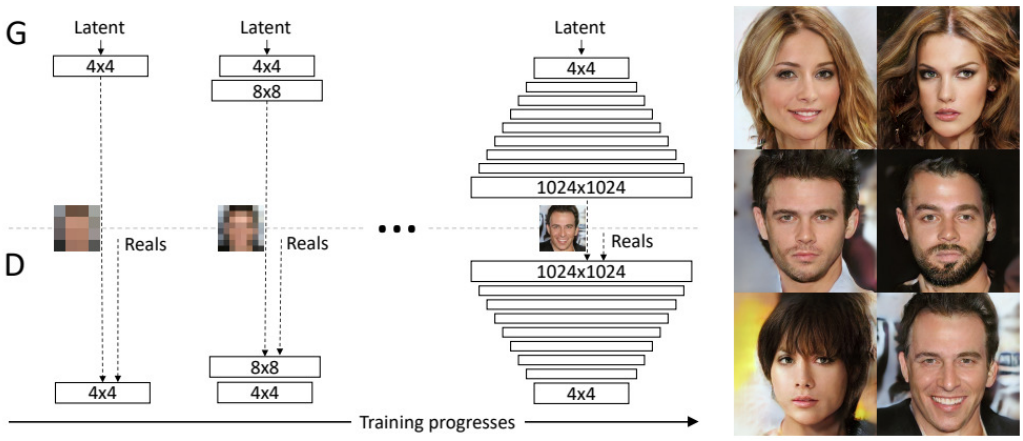

Figure 2.1: Progressive growing technique used in ProGAN.

Most important idea presented in the ProGAN architecture, utilized also in StyleGAN, is the progressive structure of learning. Volume of generated by network images is growing from small resolution (starting at 4x4 pixels) to high resolution (up to 1024x1024 pixels) by upsampling. This training principle helps the network to solve simpler task before attending to generate a full-resolution image. It has been proven

to help with stability of the training and reduce drastic abnormalities in final image.

2.1.1 Architecture overview

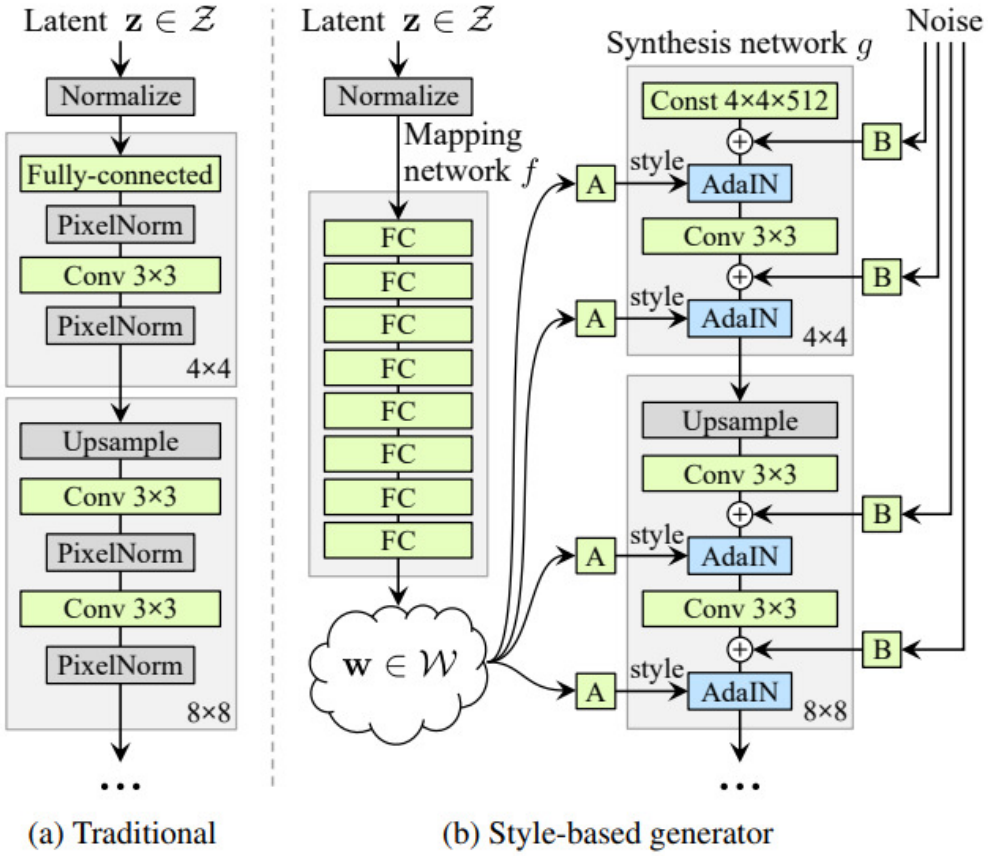

Figure 2.2: ProGAN and StyleGAN networks comparison.

FULL OVERVIEW OF ARCHITECTURE (FULL BUILD, LAYERS, PARAMETERS ITD.) HERE.ALSO DATASET IT WAS TRAINED ON.

TALK ABOUT CONVOLUTIONS MAYBE, SOME GENERAL STUFF NOT TO ATTACK WITH ALL THE FANCY SMART TALK.

SAME FOR OTHER ARCHITECTURES, stylegan2 and biggan.

2.1.2 Mapping network

In the StyleGAN architecture authors introduced a intermediate layer between an input (latent vector $z \in \mathbb{Z}$) and the network called mapping layer (fig...). It works by

applying 8 fully connected layers to the input and therefore encoding it as a new vector $w \in \mathbb{W}$. Main purpose of this procedure is to have better control over generative power of the model by separating elements of the vector that will be responsible for different image features.

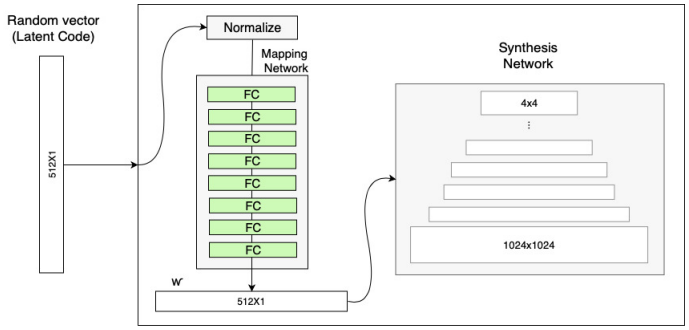

Figure 2.3: Mapping network of StyleGAN.

A common problem for generative models is a phenomenon called **feature entanglement**. We can consider a scenario where training is done on a dataset of human faces. It is very likely that most of the women in the dataset will have long hair and most men - short hair. This would result in the entanglement of *hair length* and *gender* features. Therefore, when latent vector is manipulated in order to change the hair length - the model will also change the gender of a person on the image, as it is associating these two occurencies as bounded together. The mapping layer is separating these features, allowing to change different elements of new vector w without losing the integrity of an image.

2.1.3 Adaptive Instance Normalization

In the traditional generator, latent code is introduced to the network only in the first layer. The authors of StyleGAN are using the vector w produced via mapping network to steer the style of an image at each convolutional layer by using **Adaptive Instance Normalization (AdaIN)** technique.

$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

Figure 2.4: Mapping network of StyleGAN.

The middle part of the equation is the **Instance Normalization (IN)**. Each channel of the convolution layer is normalized. Values $\mathbf{y}_{s,i}$ and $\mathbf{y}_{b,i}$ are calculated from the vector v using fully-connected layer and correspond to A on the figure-main one. These can be understood as scale and bias; these values are used to translate the information from vector w to a feature map generated by convolution.

Figure 2.5: Example of AdaIN application for style transfer.

Adding this method of including latent vector at every step of network computation resulted in drastic improvement of network's performance which can be see in the table below.

	Method	CelebA-HQ	FFHQ
A	Baseline Progressive GAN [30]	7.79	8.04
B	+ Tuning (incl. bilinear up/down)	6.11	5.25
C	+ Add mapping and styles	5.34	4.85
D	+ Remove traditional input	5.07	4.88
E	+ Add noise inputs	**5.06**	4.42
F	+ Mixing regularization	5.17	**4.40**

Figure 2.6: Mapping network of StyleGAN.

2.1.4 Style mixing regularization

As additional technique of regularization and method to differentiate the generated images further, a **style mixing** approach was introduced by the authors. The main idea is to use not one but two (or more) latent vectors w_1, w_2 (obtained from mapping of z_1, z_2) to generate the final image. The way in which mixing is introduced is fairly simple; up to certain point in the architecture vector w_1 is used for style control and from crossover point vector w_2 is used. Depending on the place of crossover, final image obtains different characteristics from each image.

DOPISAĆ DOKŁADNIE ROZMIARY.

2.1.5 Additional noise inputs

Authors provided the generator with ability to alternate very fine details on a picture by adding stochastic variance into the model. Single-channel images of uncorrelated Gaussian noise are fed into different places in the architecture. Noise is scaled per channel (which means that the amount of noise is a learnable parameter) and added after each convolution layer before AdaIN layer.

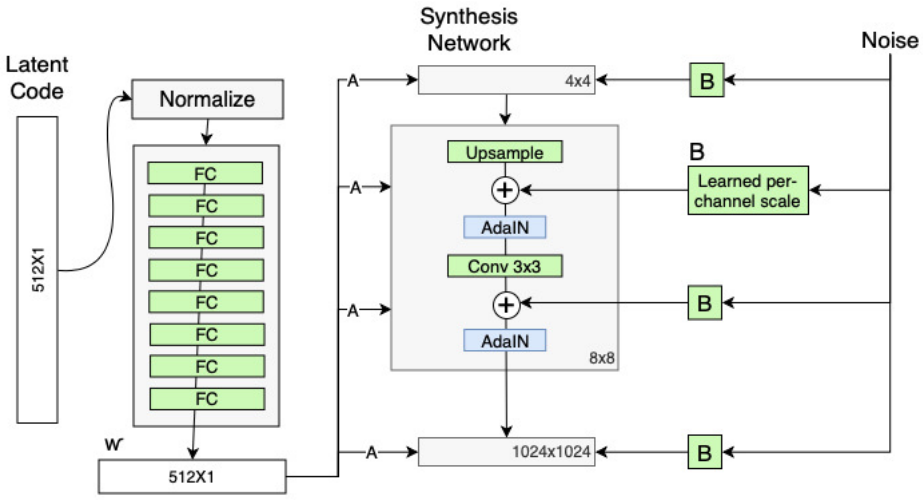

Figure 2.7: Places in the architecture where noise is added.

Additional noise allows to generate and alternate very detailed parts of the image (wrinkles, freckles, hair). Thanks to scalability and DOKONCZYC

(a) Generated image (b) Stochastic variation (c) Standard deviation

Figure 2.8: Visual inspection of where the gaussian noise tends to be the most active.

2.2 StyleGAN2

Significant improvements of the StyleGAN architecture were proposed in late 2019. Publication revolved mainly around the problems which emerged during StyleGAN generator. It has been noticed that several parts of the network cause imperfections in the final images, which are called *artifacts*. Two main types of artifacts were formulated in the paper:

- **Water-droplet artifacts** - the blob-shaped characteristics that resemble water-droplets are visible in various places in the final images. It might not be obvious while looking at the image, but it is very visible in the intermediate feature maps produced by generator. This artifacts starts to appear around 64x64 pixels resolution and gets stronger with higher resolutions.
- **Phase artifacts** - output images show strong location preference for several features e.g. facial features like teeth or eyes. This causes some of the features to remain unchanged when major components of the image such as pose or rotation are vastly different.

Figure 2.9: Water droplets artifact and its effect on feature maps.

Figure 2.10: Phase artifact. Blue line helps to visualize that teeth are staying in the same place even with changing a significant feature of image such as rotation of the face..

2.2.1 Architecture overview

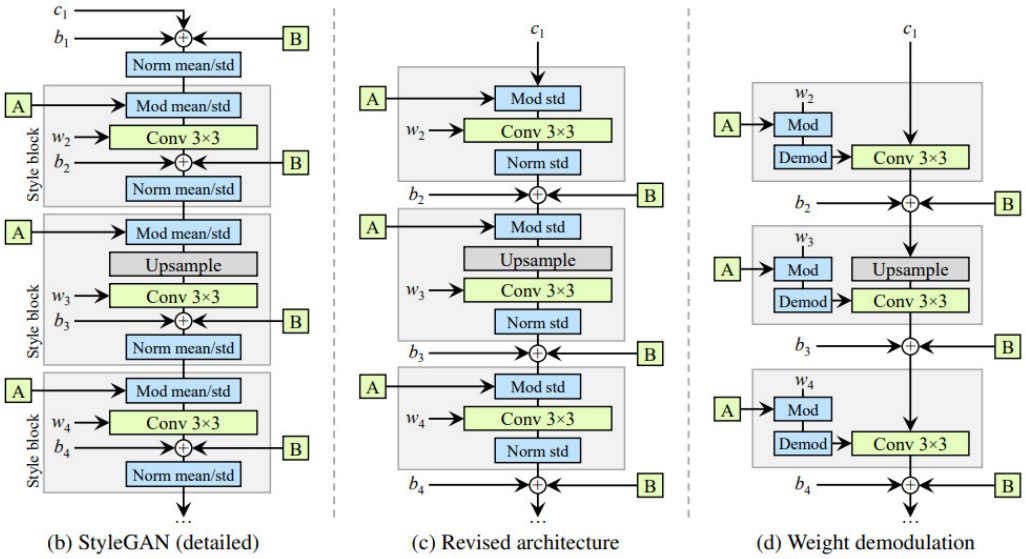

Figure 2.11: Visual inspection of where the gaussian noise tends to be the most active.

2.2.2 Revisiting Instance Normalization

The water-droplet effect was speculated to be a side effect of AdaIN method applied in the style blocks. Authors pointed out that this type of normalization affects each feature map separately (works independently for every channel) and potentially destroys meaningful information that is included in the differences between feature maps values.

To address this problem, significant changes to the architecture were proposed.

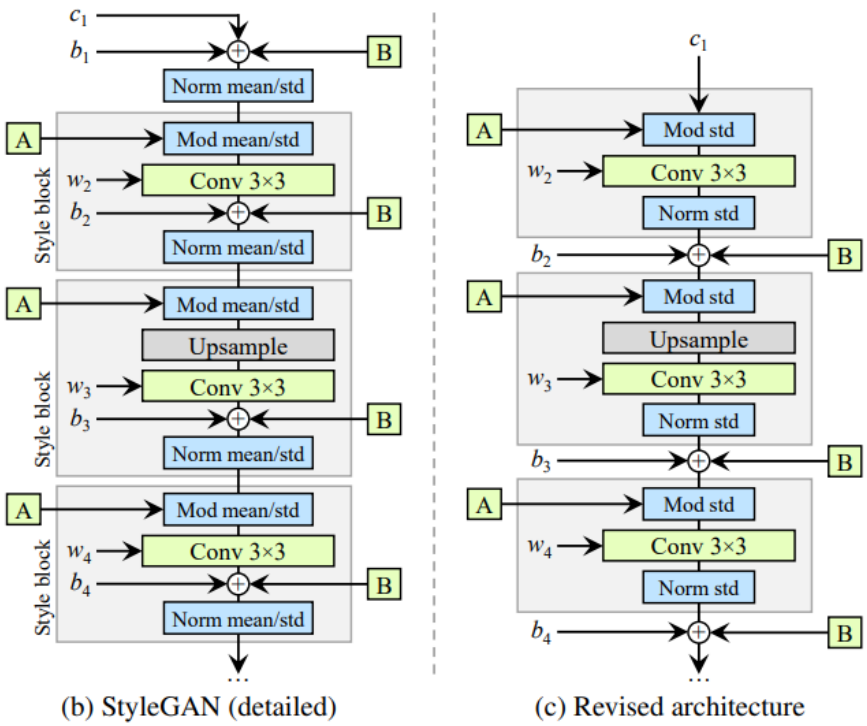

Figure 2.12: (b) shows the original StyleGAN architecture. The AdaIN fucntion is shown as combination of normalization and modulation operations. (c) shows the revised model with changes in the network.

As seen on the image above, several things changed in the network:

- Only standard deviation is modified for each feature map, modification of mean was deemed redundant as the effects of this operations were not meaningful for the network. Removing this part made the model simpler,
- Noise addition was removed from the style block and instead applied inbetween the blocks. This was mainly done to prepare the network for weight modulation,
- Input vector c is fed to the network directly, without normalization and applying

noise.

All of these steps were also necessary to introduce the main idea for handling water-droplets artifact problem - **weight demodulation**.

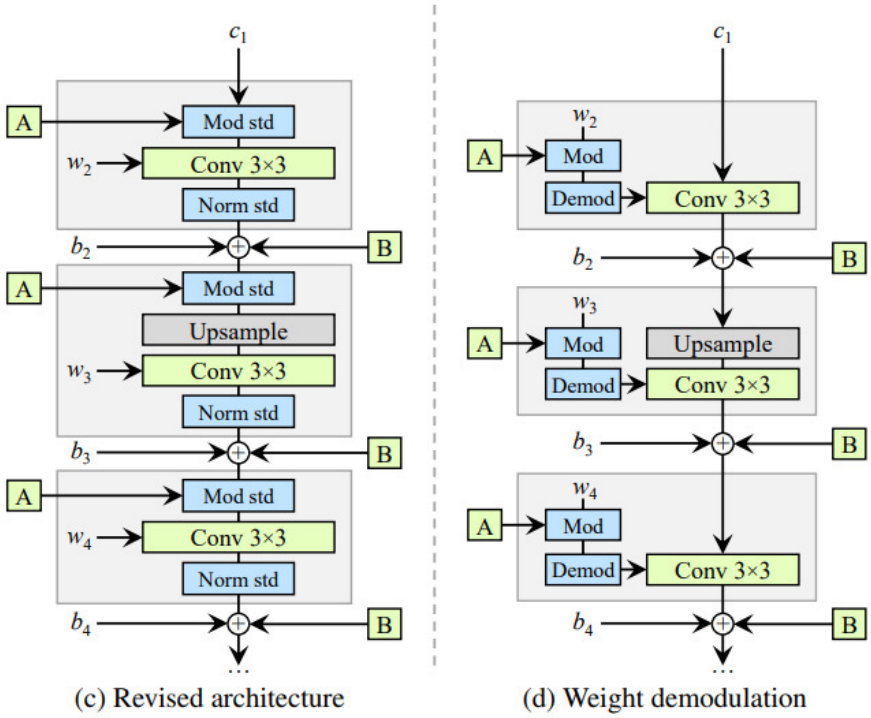

Figure 2.13: (b) shows the original StyleGAN architecture. The AdaIN fucntion is shown as combination of normalization and modulation operations. (c) shows the revised model with changes in the network.

HERE WEIGHT DEMODULATION

2.2.3 Removing Progressive Growing

StyleGAN2 uses different resolution feature maps that are produced in the base network and uses the ResNet like skip connections to incorporate lower resolutions maps to the final output. This change can be viewed as quite drastic - the whole concept of base network was changed, but it proved to be a necessary step in order to mitigate the *phase artifact* effect.

MORE HERE

2.2.4 Other improvements

LAZY REGULARIZATION

PATH LENGTH REGULARIZATION

2.3 BigGAN

Chapter 3

Framework description

3.1 Overview

The architecture of solution that we are exploring in this work consists of four main components:

1. Pre-trained generative model based on Generative Adversarial Network architecture,
2. CLIP model,
3. Evolutionary optimization algorithm,
4. Evaluation using classification network.

Graph below shows the flow of data and describes the overall framework architecture:

DOKŁADNIEJ ROZPISAĆ:

Description of the flowchart:

1. First, text is provided by the user,
2. Introduced text is encoded using pre-trained CLIP model.

Next steps can be considered as a 'optimization loop':

3a. Algorithm is performing optimization step,

3b. Optimized population of samples (vectors) is serving as an input for generator which produces images,

3c. Generated images are evaluated by fitness function (in our case, similarity function).

After completing this process, post-processing part takes place:

4. Output images are passed for an evaluation using classification network and similarity metric.

We consider StyleGAN2 architecture as the main choice for a generative model in this solution. Main advantages of that choice are:

- It is widely available as a pre-trained structure; it is public and open-sourced for both generative and discriminative parts of the architecture.
- It consists of several class-based dedicated models (such as StyleGAN2-car or StyleGAN2-horse). It allows for experimentations with different types of images.
- It is still considered a state-of-the-art model regarding the quality of produced images and was trained using huge datasets of images.

As of optimization algorithm, we used two algorithms described in previous chapters - *genetic algorithm* and *differential evolution*.

OPISAĆ ZASTOSOWANE TECHNOLOGIE I PACZKI, python pytorch hydra etc.

Chapter 4

Datasets and evaluation

Chapter 5

Experiments

Chapter 6

Summary

Bibliography

[1] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. *Improved techniques for training gans.* In NIPS, 2016.

[2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna. *Rethinking the inception architecture for computer vision.* arXiv preprint arXiv: 1512.00567, 2015.

[3] S. Barratt, R. Sharma. *A note on the Inception Score.* arXiv preprint arXiv: 1801.01973, 2018.

[4] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, S. Hochreiter. *GANs trained by a two time-scale update rule converge to a local Nash equilibrium.* In NIPS, 2017.

[5] R. Storn, K. Price. *Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces.* Springer, 1997.

[6] T. Karras, S. Laine, T. Aila. *A style-based generator architecture for generative adversarial networks.* In CVPR, 2019.