

POLITECHNIKA WARSZAWSKA

WYDZIAŁ ...

Kierunek: ...

...

Adam Komorowski i Maciej Domagała

WARSZAWA, CZERWIEC 2021

Contents

Introduction	4
1 Theory	5
1.1 Generative Adversarial Networks	5
1.1.1 Overview	5
1.1.2 Architecture	6
1.2 CLIP model	9
1.2.1 Overview	9
1.3 Evolutionary Algorithms	14
1.3.1 Genetic Algorithm	14
1.3.2 Differential Evolution	16
2 Notable architectures	20
2.1 StyleGAN	20
2.1.1 Architecture overview	21
2.1.2 Mapping network	21
2.1.3 Adaptive Instance Normalization	22
2.1.4 Style mixing regularization	23
2.1.5 Additional noise inputs	24
2.2 StyleGAN2	24

2.2.1	Architecture overview	25
2.2.2	Revisiting Instance Normalization	26
2.2.3	Removing Progressive Growing	28
2.2.4	Other improvements	28
2.3	BigGAN	28
3	Framework description	29
3.1	Overview	29
4	Datasets and evaluation	31
5	Experiments	33
5.1	Evaluation	34
5.1.1	CIFAR10	34
5.1.2	ImageNet	38
6	Summary	43
7	Appendix 1	44
8	Appendix 2	48
Citation		52

Introduction

Chapter 1

Theory

1.1 Generative Adversarial Networks

1.1.1 Overview

Generative Adversarial Networks, or GANs for short, are an approach to generative modeling using deep learning methods, such as convolutional neural networks. They were proposed in a 2014 by Ian Goodfellow et al [7].

What applications of GANs are most popular today? They are most often used in the field of computer vision, they are a tool for e.g. data augmentation in relatively small datasets, for example, one can feed some text written in a particular handwriting as input to a generative model to get generate more text in the same handwriting. The following are results of the first facial images generation experiments with GANs from the work of I.Goodfellow et al [7].



Figure 1.1: Images generated by GAN - presented in [7].

Since then, GANs networks have come a long way, and their development is best summed up by the following image:



Figure 1.2: Example of Progression in the Capabilities of GANs From 2014 to 2017.

In our work, we will describe the architectures of two state of the art GAN models, StyleGAN and BigGAN.

1.1.2 Architecture

Generative adversarial networks are composed of two neural networks, one called the **generator** - G and the other called the **discriminator** - D.

The role of the generator is to estimate the probability distribution of the real samples in order to provide generated samples resembling real data. The discriminator, in turn, is trained to estimate the probability that a given sample came from the real data rather than being provided by the generator. These structures are called generative adversarial networks because the generator and discriminator are trained to compete with each other: the generator tries to get better at fooling the discriminator, while the discriminator tries to get better at identifying generated samples. GAN architecture can be described using schema below.

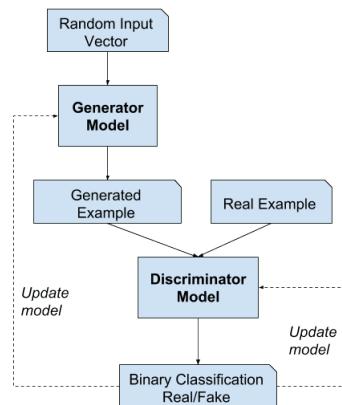


Figure 1.3: GAN architecture.

Generator can be any neural network - multilayer perceptron (MLP), as well as, a convolutional neural network (CNN).

Training

The GAN training process consists of a two-player minimax game. Although the dataset containing the real data isn't labeled, the training processes for D and G are performed in a supervised way.

To train D, at each iteration we pass some real samples taken from the training data labeled as 1 and some generated samples provided by G labeled as 0. G is generating "fake" images based on noise as an input derived e.g. from Gaussian distribution. This way, we can use a conventional supervised training framework to update the parameters of D. Going to equations - the discriminator outputs a value $D(x)$ indicating the chance that x is a real image. Model objective is to maximize the chance to recognize real images as real and generated images as fake i.e. the maximum likelihood of the observed data. To measure the loss, we use **cross-entropy** and therefore the objective function is as follows:

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

For each batch of training data containing labeled real and generated samples, model updates the parameters of D to optimize a loss function. After the parameters of D are updated, G is trained to produce better generated samples. The output of G is connected to D, whose parameters are kept frozen, for the G training time. On the generator side, it's objective function wants the model to generate images with the highest possible value of $D(x)$ to fool the discriminator.

$$\min_G V(G) = \mathbf{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

To sum up, we can define GAN as a minimax game which G wants to minimize V, while D wants to maximize it.

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

Once both objective functions are defined, they are learned jointly by the alternating gradient descent. We fix the generator model's parameters and perform a single

iteration of gradient descent on the discriminator using the real and the generated images. Then we switch sides. Fix the discriminator and train the generator for another single iteration. We train both networks in alternating steps until the generator produces good quality images. The training algorithm for GANs is summarized in the figure below, taken from the original 2014 paper by Goodfellow, et al. titled “Generative Adversarial Networks.”

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k , is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```

for number of training iterations do
    for  $k$  steps do
        • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
        • Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .
        • Update the discriminator by ascending its stochastic gradient:
            
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D(\mathbf{x}^{(i)}) + \log (1 - D(G(\mathbf{z}^{(i)})))]$$

    end for
    • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
    • Update the generator by descending its stochastic gradient:
        
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(\mathbf{z}^{(i)})))$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

```

Figure 1.4: GAN training algorithm from [7].

At the very end, it is worth mentioning that a known problem of GAN networks is the fact that not always a smaller value of loss function means better results - in practice, a common solution is to empirically evaluate generated results every few iterations, rather than tracking loss. To cope with this problem metrics such as **Inception Score** or **Frechet Inception Distance** were presented. These will be described in detail in the following sections.

1.2 CLIP model

1.2.1 Overview

In January 2021 OpenAI released new multi-modal model called **CLIP - Contrastive Language-Image Pre-Training**. It is a neural network trained on a 400 000 000 (image, text) pairs - each pair is an image and its caption scrapped by OpenAI from the Internet. What this model does? Quoting OpenAI paper "It can be instructed in natural language to predict the most relevant text snippet, given an image, without directly optimizing for the task".

CLIP is an example of **zero-shot** model. That is, compared to commonly used

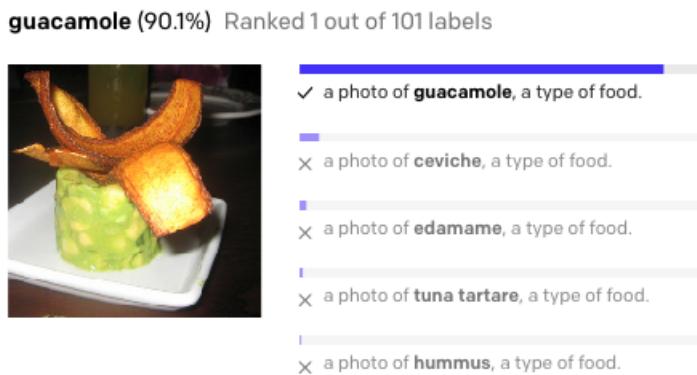
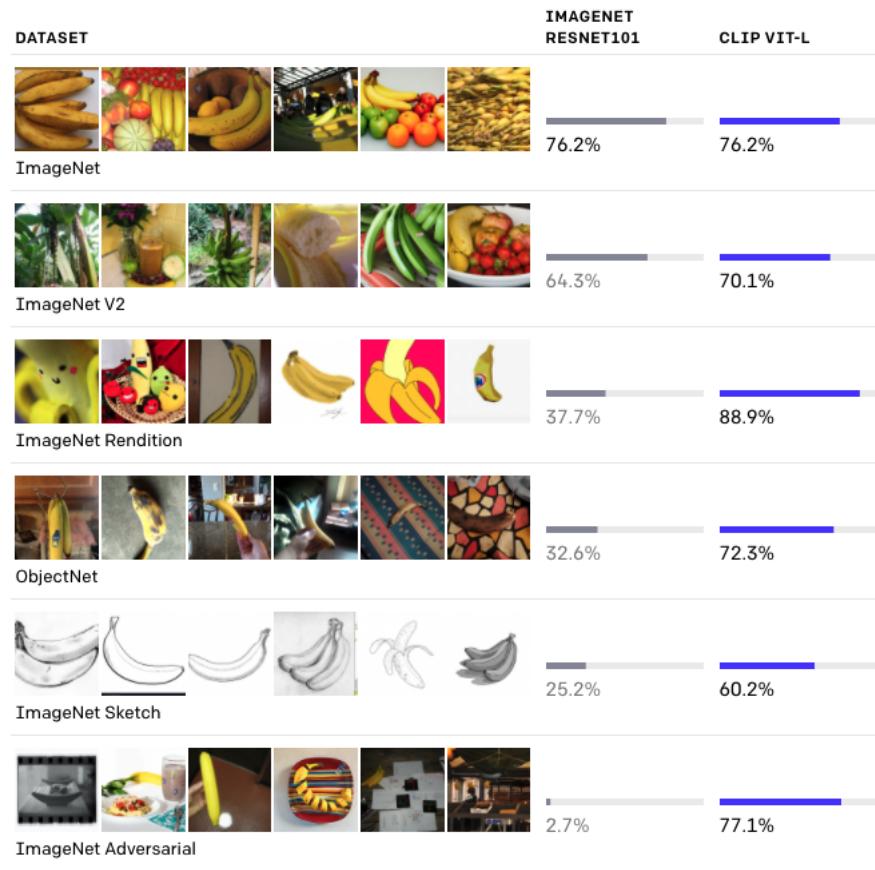


Figure 1.5: Example of CLIP prediction.

classifiers that require custom datasets that represent target classes and don't generalize very well, CLIP can identify an enormous range of things it has never seen before. OpenAI proves such behaviour using the following table



It compares ResNet101 model - trained on the ImageNet dataset with CLIP accuracy on different datasets. It can be seen that for the ImageNet dataset, the accuracy is identical, but for the other models, CLIP is the clear winner and therefore better generalizes the problem being modeled.

How does CLIP work?

In order for images and text to be compared to one another, they must both be embedded. Part of the CLIP model is responsible for embeddings - it contains two encoders - ImageEncoder and TextEncoder. So the first part of models is as follows:

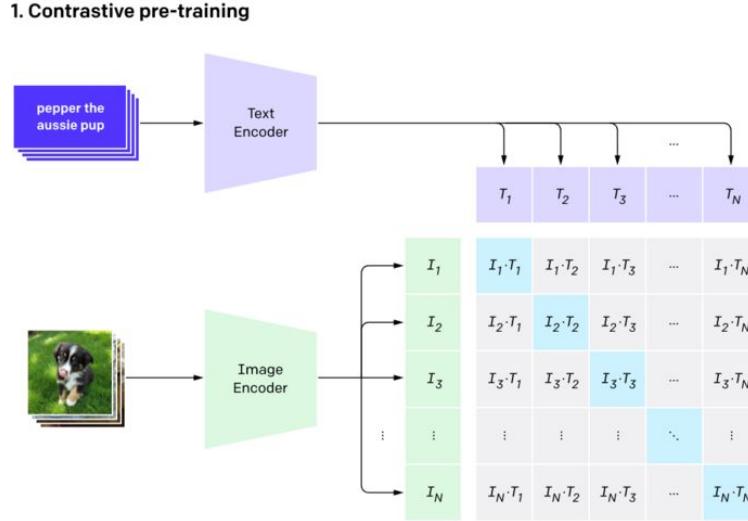


Figure 1.6: ...

Image and text embeddings are compared in the similarity matrix $I \times T$ using **cosine similarity**.

Definition 1. *The cosine of two non-zero vectors can be derived by using the Euclidean vector dot product formula:*

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \|\mathbf{B}\| \cos \theta$$

Given two vectors of attributes, A and B, the cosine similarity, $\cos(\theta)$, is represented using a dot product and magnitude as

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

where A_i and B_i are components of vector A and B respectively.

During training, it is known, that the values on the diagonal represent correct classifications, so their similarity must be higher than those in the same row/column. This approach contrasts what we know go together (diagonal values) to what we know doesn't go together (non-diagonal values). You can see that each row is a classification task: given an input image I, predict the text. Similarly, each column is a classification task: given an input text T, predict the image. During training, OpenAI used a very large size of mini-batches 32768 (N on the figure above). During inference one takes a set of labels, creates texts based on labels and runs these texts through the text encoder. Text embeddings are later matched to image representation.

Model objective is to maximize cosine similarities for successive training (image, text) pairs and minimizing it for "negative" pairs - which means not training pairs.

One detail that is worth mentioning is that CLIP is sensitive to words used for image descriptions. Texts "a photo of a bird", "a photo of a bird sitting near bird feeder", or "an image of a bird" all produce different probability paired with the same image:

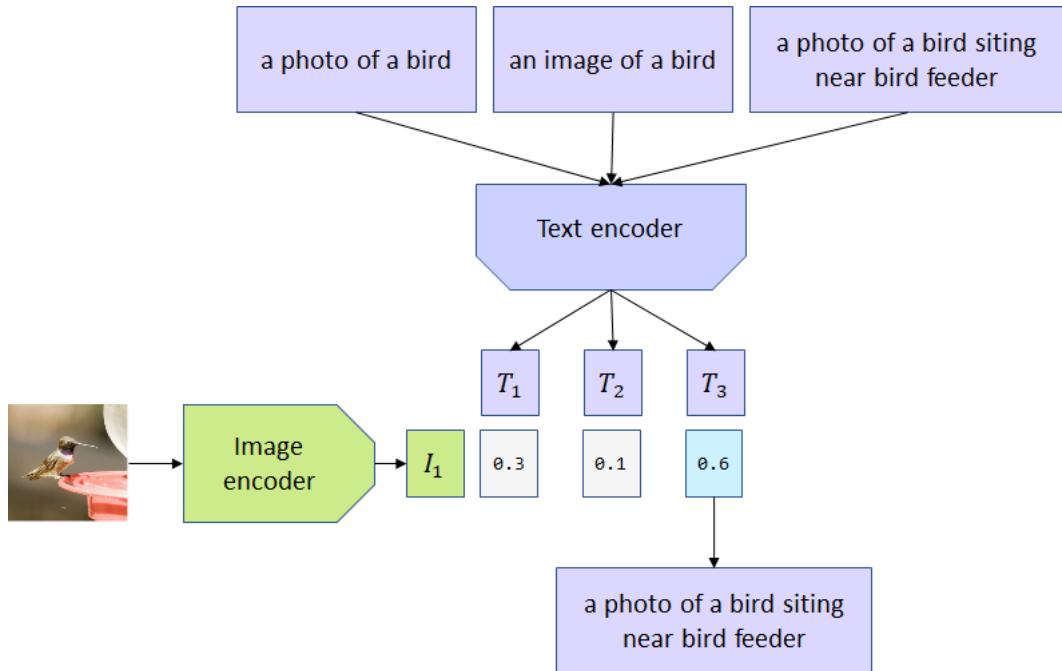


Figure 1.7: CLIP is producing different probabilities for different bird image descriptions.

CLIP current results and limitations

CLIP authors are open about its limitations. CLIP struggles on more abstract or systematic tasks such as counting the number of objects and on a more complex tasks such as estimating relative distances between objects. On such datasets, CLIP is only slightly better than random guessing. CLIP also struggles with very fine-grained classification, such as telling the difference between car models, variants of aircraft, or flower species.

What's more, CLIP model itself is data hungry and expensive to train. If pre-trained model doesn't work well for you, it may be not feasible to train your own version.

While zero-shot CLIP tries to reformulate classification task, the principles are still the same. And although CLIP generalizes well to many image distributions, it still generalizes poorly to data that is truly out-of-distribution. One example of this was CLIP's performance on MNIST dataset where CLIP zero-shot accuracy was 88%. Logistic regression on raw pixels outperforms CLIP.

Ability to adapt to new datasets and classes is related to text encoder. It is thus limited to choosing from only those concepts known to the encoder. CLIP model trained with English texts will be of little help if used with texts in other languages.

Finally, CLIP's classifiers can be sensitive to wording in label descriptions and may require trial and error to perform well.

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.

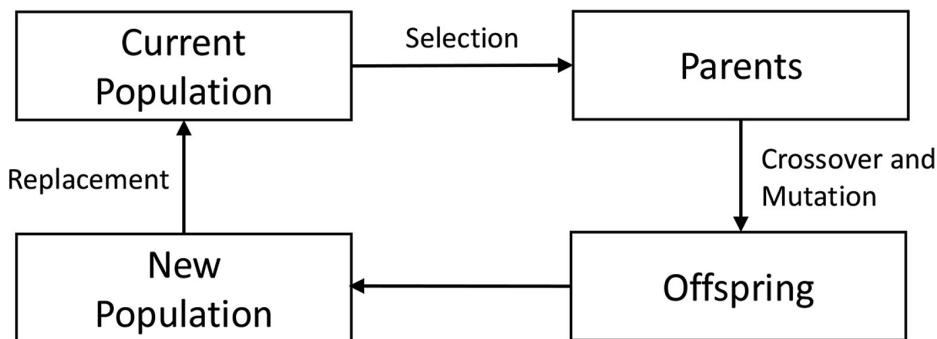


Figure 1.8: Genetic algorithm optimization loop.

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.

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.

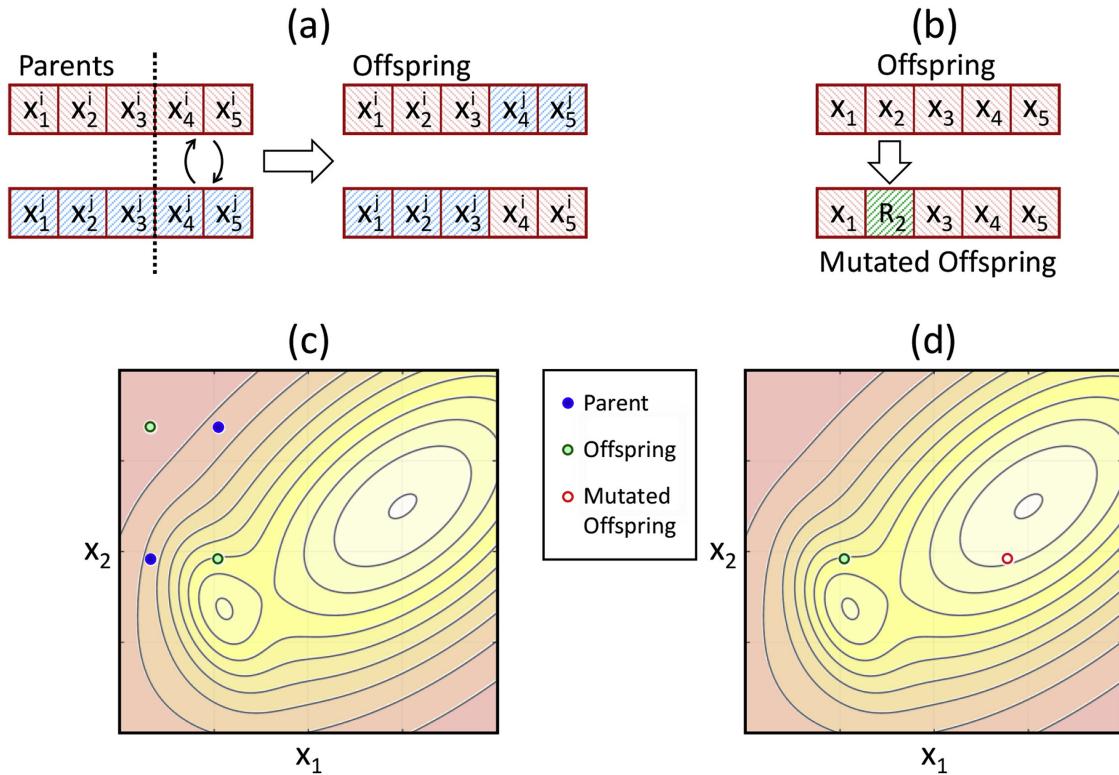


Figure 1.9: Example of crossover and mutation operations.

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.

Algorithm 1 Genetic Algorithm

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

Figure 1.10: Genetic algorithm pseudocode.

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.

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 $i \in \{1, 2, \dots, N_p\}$ and g stands for generation number.

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

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:      $r_1, r_2, r_3 \in [1; \text{population size}]$ , where  $r_1 \neq r_2 \neq r_3 \neq i$ 
9:     for each  $j$ -th gene in  $i$ -th individual ( $j \in [1; n]$ ) do
10:     $v_{i,j} = x_{r_1,j} + F \cdot (x_{r_2,j} - x_{r_3,j})$ 
11:    randomly generate one real number  $rand_j \in [0; 1)$ 
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 1.11: Differential evolution DE/rand/1 pseudocode.

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

$$\mathbf{v}_{i,g} = \mathbf{x}_{r_1,g} + F (\mathbf{x}_{r_2,g} - \mathbf{x}_{r_3,g}), \quad (1.1)$$

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

$$\mathbf{v}_{i,g} = \mathbf{x}_{best,g} + F (\mathbf{x}_{r_2,g} - \mathbf{x}_{r_3,g}). \quad (1.2)$$

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 $\mathbf{u}_{i,g} = (u_{i,1,g}, u_{i,2,g}, \dots, u_{i,D,g})$ for $i \in \{1, 2, \dots, N_p\}$ and $j \in \{1, 2, \dots, D\}$ is created using formula

$$u_{i,j,g} = \begin{cases} v_{i,j,g} & \text{if } rand_j \leq C_r \text{ or } j = j_{rd}, \\ x_{i,j,g} & \text{otherwise,} \end{cases} \quad (1.3)$$

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

$$I = \{a, (a + 1)_{MOD_D}, \dots, (a + b - 1)_{MOD_D}\}. \quad (1.4)$$

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

$$u_{i,j,g} = \begin{cases} v_{i,j,g} & \text{if } j \in I \text{ and } rand_j \leq C_r, \\ x_{i,j,g} & \text{otherwise.} \end{cases} \quad (1.5)$$

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 .

$$\mathbf{x}_{i+1,g} = \begin{cases} \mathbf{u}_{i,g} & \text{if } f(\mathbf{u}_{i,g}) < f(\mathbf{x}_{i,g}), \\ \mathbf{x}_{i,g} & \text{otherwise} \end{cases} \quad (1.6)$$

The algorithm terminates on similar basis as genetic algorithm discussed previously, when a optimal solution below certain threshold 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.

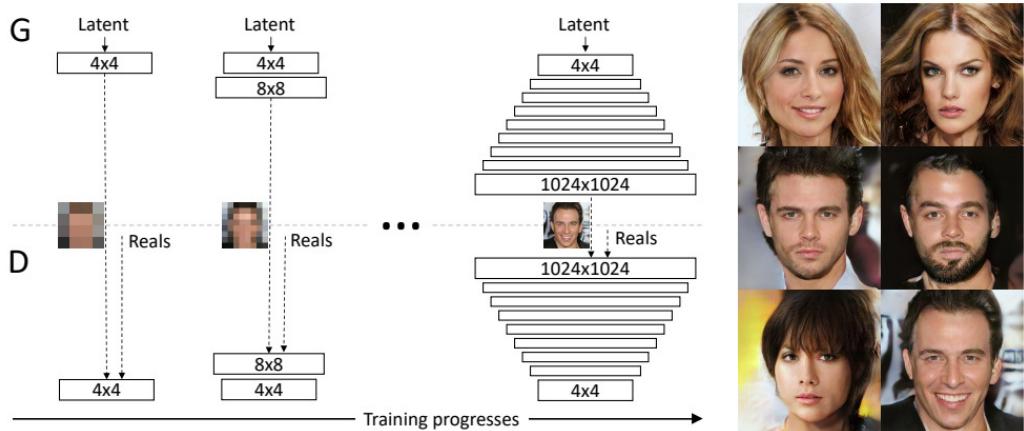


Figure 2.1: Progressive growing technique used in ProGAN.

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

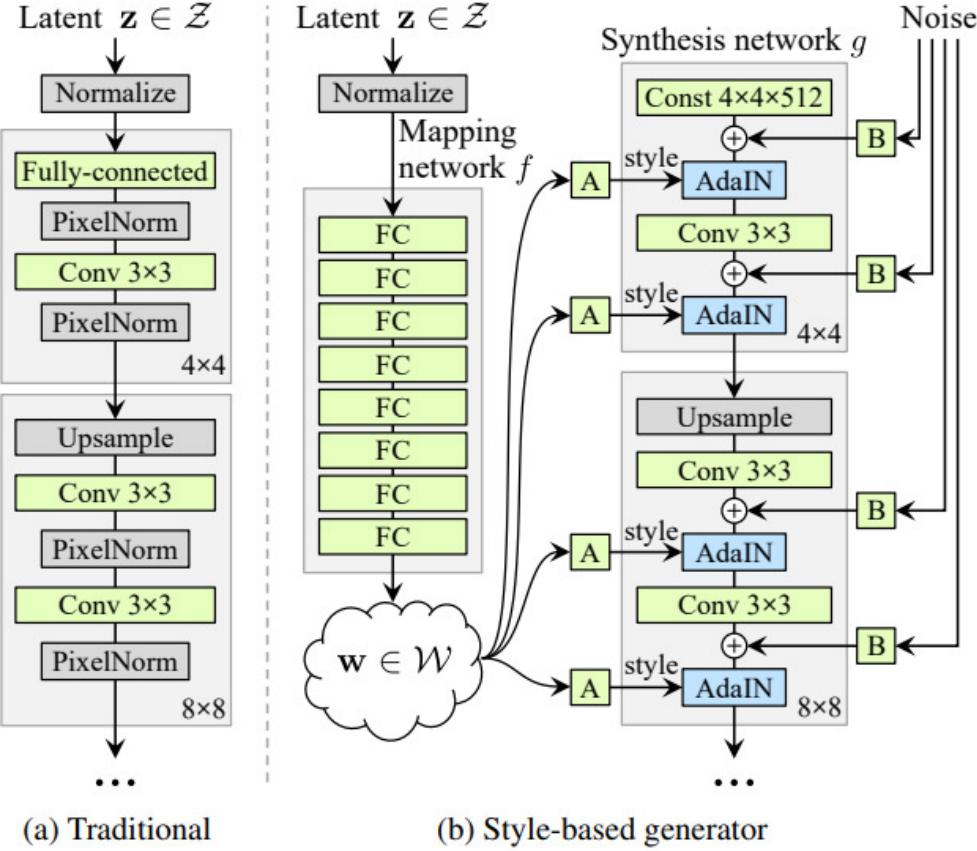


Figure 2.2: ProGAN and StyleGAN networks comparison.

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.

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

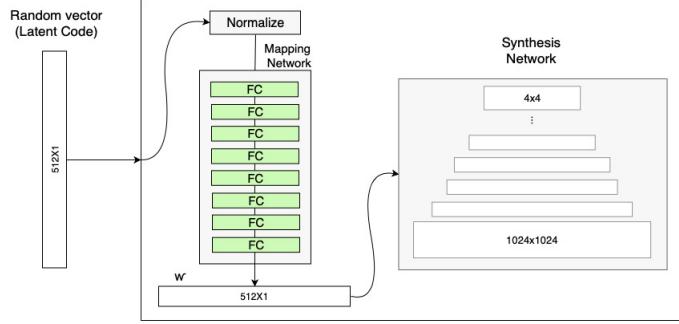


Figure 2.3: Mapping network of StyleGAN.

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 occurrences 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.

Adding this method of including latent vector at every step of network computation

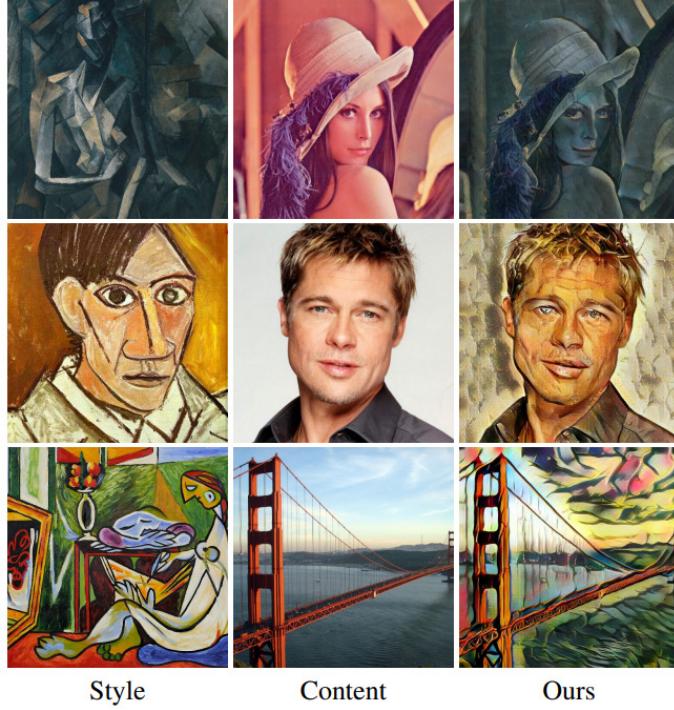


Figure 2.5: Example of AdaIN application for style transfer.

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.

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.

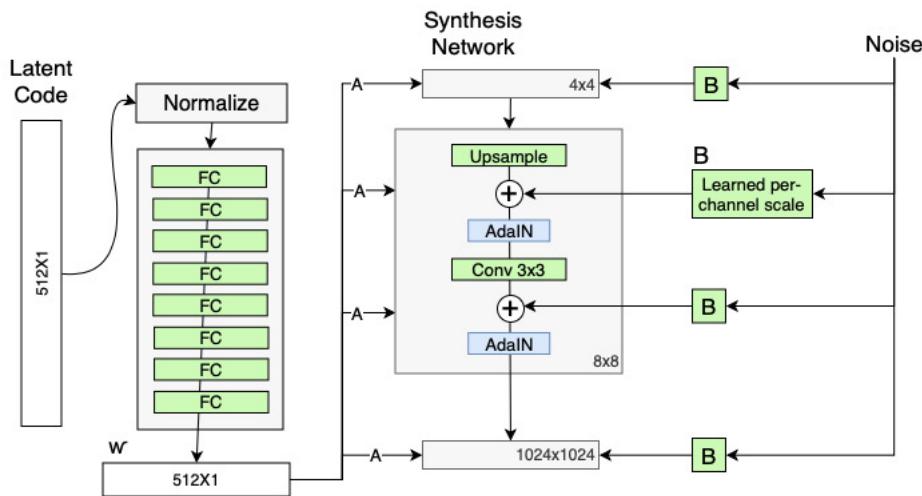
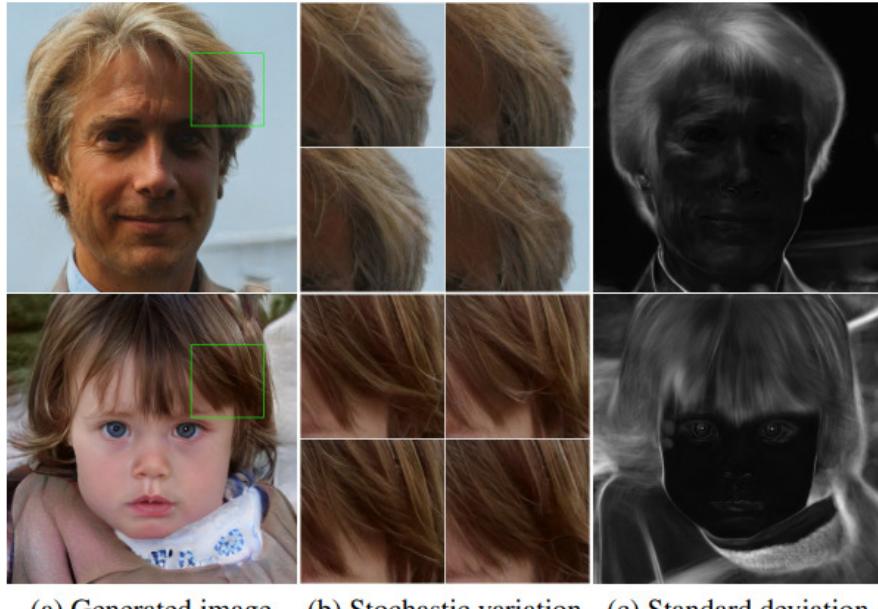


Figure 2.7: Places in the architecture where noise is added.

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

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:



(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.

- **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.

2.2.1 Architecture overview

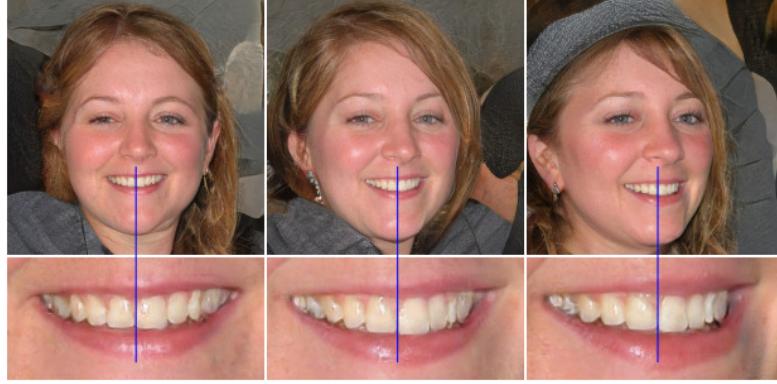


Figure 2.10: Phase artifact. Blue line helps to visualize that teeth are staying in the same place even with changing a significant feature of image such as rotation of the face..

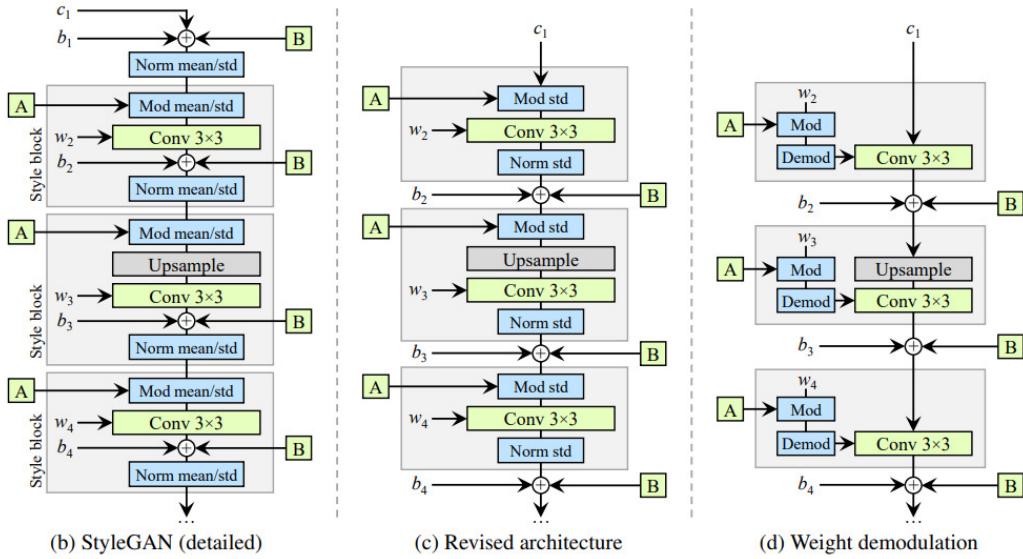


Figure 2.11: Visual inspection of where the gaussian noise tends to be the most active.

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.

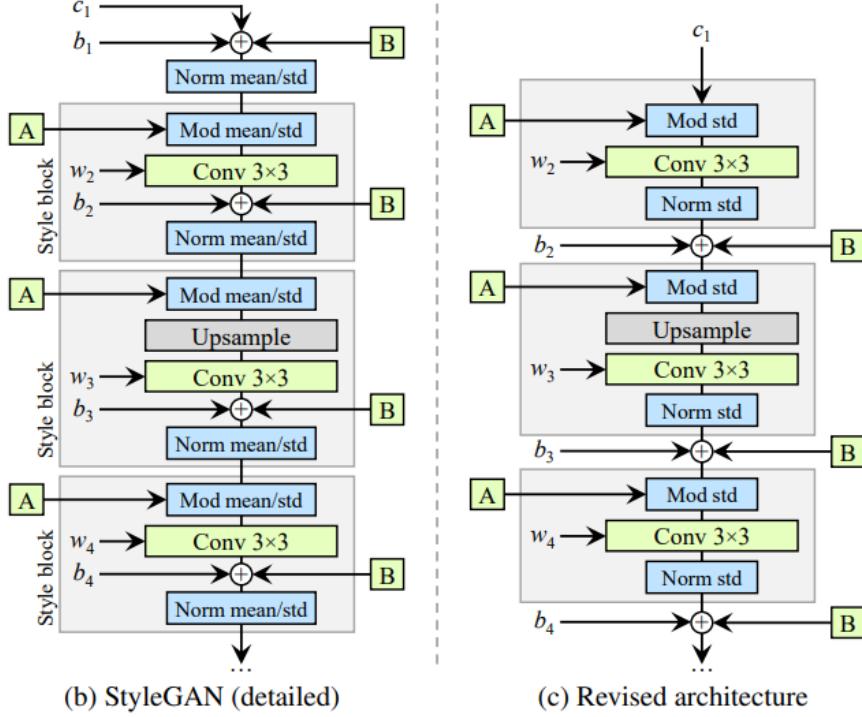


Figure 2.12: (b) shows the original StyleGAN architecture. The AdaIN function is shown as combination of normalization and modulation operations. (c) shows the revised model with changes in the network.

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**.

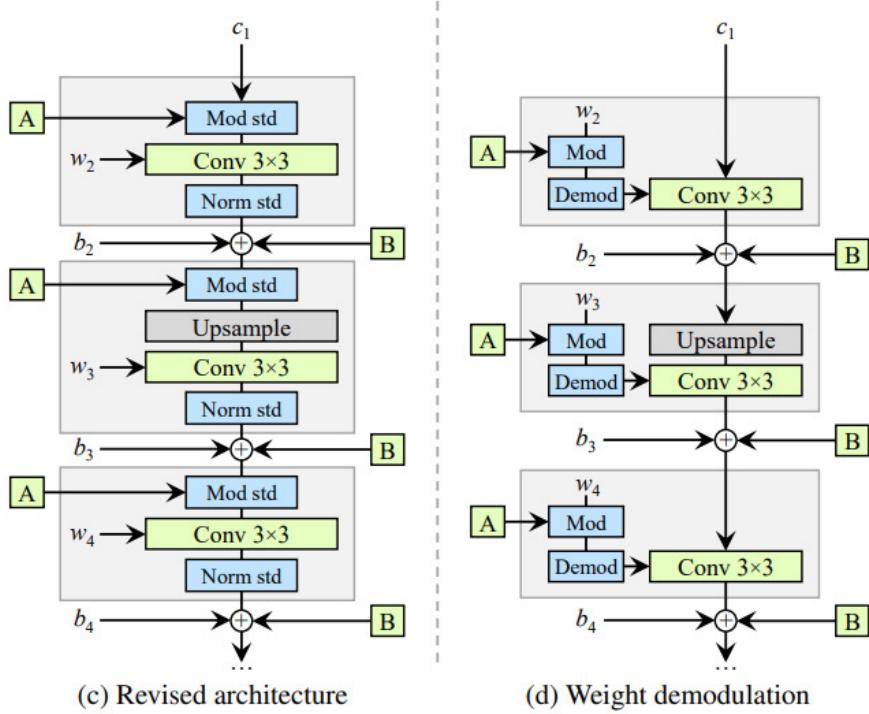


Figure 2.13: (b) shows the original StyleGAN architecture. The AdaIN function is shown as combination of normalization and modulation operations. (c) shows the revised model with changes in the network.

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.

2.2.4 Other improvements

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:

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*.

Chapter 4

Datasets and evaluation

In order to evaluate the model we described, we decided to see if the images we generated would be able to "fool" the classifiers learned on popular datasets. We took the following datasets as our benchmarks:

CIFAR10

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. Here are the classes in the dataset, as well as 10 random images from each: The classes are

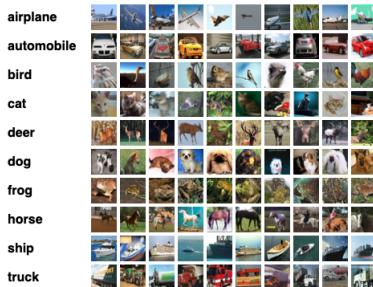


Figure 4.1: Examples from CIFAR10 dataset.

completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks. Dataset can be downloaded from [9].

ImageNet

The ImageNet dataset contains 14 197 122 hand-annotated images. Since 2010 the dataset is used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a benchmark in image classification and object detection. ImageNet contains more than 20000 categories with a typical category, such as "balloon" or "strawberry" and more detailed like dogs' breeds names "Rottweiler", "Old English sheepdog". Dataset can be downloaded from [10]. Having selected datasets, we chose classifiers trained on

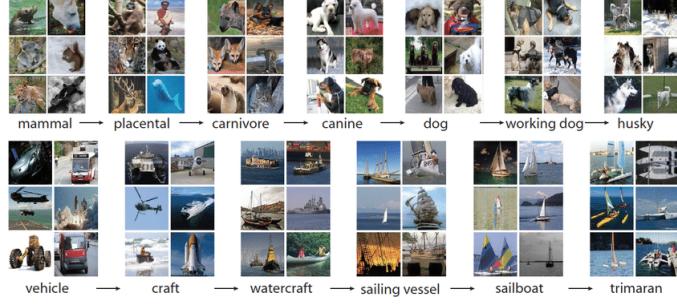


Figure 4.2: Examples from ImageNet dataset.

them with the highest possible accuracy. These are:

- CIFAR-10 CNN Classifier [link](#) with the $\sim 85\%$ accuracy.
- ImageNet - ResNet-50 Classifier - [link](#) with the $\sim 76\%$ accuracy.

To evaluate the model, we will generate a fixed number of images for the selected classes from both datasets (based on the class names), and then see what accuracy they achieved when passed to the corresponding classifiers.

For the CIFAR10 dataset we will generate images for each of the 10 available classes: **[airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck]**.

For the ImageNet dataset, we chose 10 classes - trying to have them come from different domains. The list looks as follows:

[llama, cash machine, hammer, miniskirt, pirate, shopping cart, wall clock, ice cream, banana, West Highland white terrier]

The parameters of the model we will use during the evaluation will be described in the experiments section.

Chapter 5

Experiments

5.1 Evaluation

5.1.1 CIFAR10

As an evaluation of our solution, we performed the experiment described in the Datasets and Evaluation section. Let us start by listing the most important parameters of the evaluation process for CIFAR10 dataset:

- GAN model used - **DeepMindBigGAN256**
- Evolutionary model used - **GA**
- Population size - **64**
- Number of model repetitions - **8**
- Number of images generated per class - **512 = 512 = 64**

We present results of our evaluation in the form of the following table.

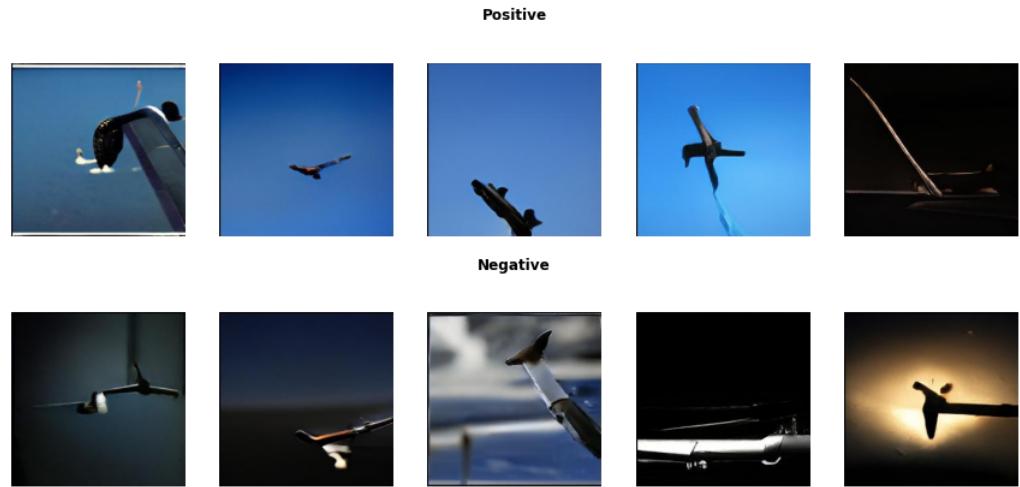
Class	Positive	Negative	Accuracy (%)
AIRPLANE	476	36	92.97
AUTOMOBILE	88	424	17.19
BIRD	12	500	2.34
CAT	177	335	34.57
DEER	0	512	0.0
DOG	255	257	49.8
FROG	39	473	7.62
HORSE	64	448	12.5
SHIP	60	452	11.72
TRUCK	97	415	18.95
TOTAL	1268	3852	24.77

In the table above, the positive column tells us the number of images generated by our model that were classified correctly, and negative tells us the number of incorrectly classified images. Starting from the end - we were able to fool the classifier learned on the CIFAR10 dataset in about 25% of the cases. This is not a very high result, but

let's take into account that the classifier itself has an accuracy of 85% and is therefore not equivalent to human, empirical experiences.

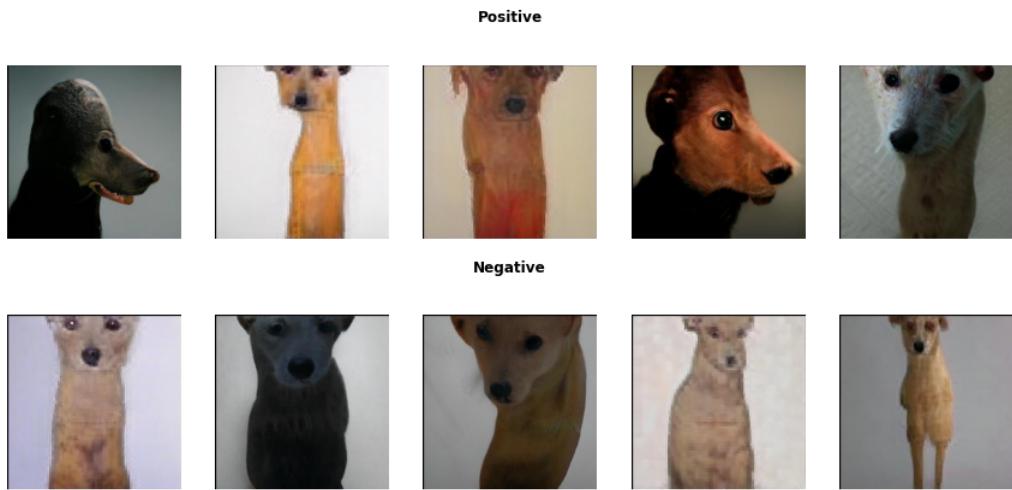
Moving on to the details - it's easy to see that for the most part accuracy is low - especially for classes like DEER, BIRD and FROG. The exception is the AIRPLANE class. Let's take a look at sample images from selected classes.

Airplane



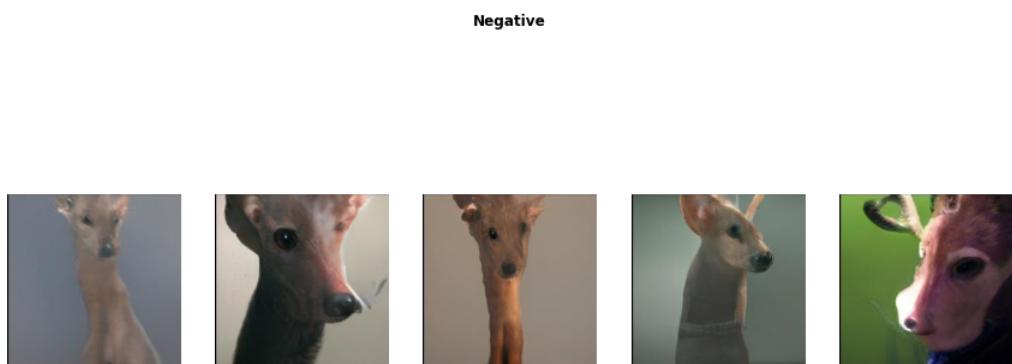
We see that the generated images of the AIRPLANE mostly show its wing against the sky. We can surmise that the classifier was wrong so often on the AIRPLANE images we generated, because they included the blue sky.

Dog



When it comes to DOG photos, it's hard to see a feature that distinguishes correctly and incorrectly classified images. Instead, it can be seen that the DOG images generated by the model are quite readable. Later in the chapter we will look at what other classes this class was confused with.

Deer

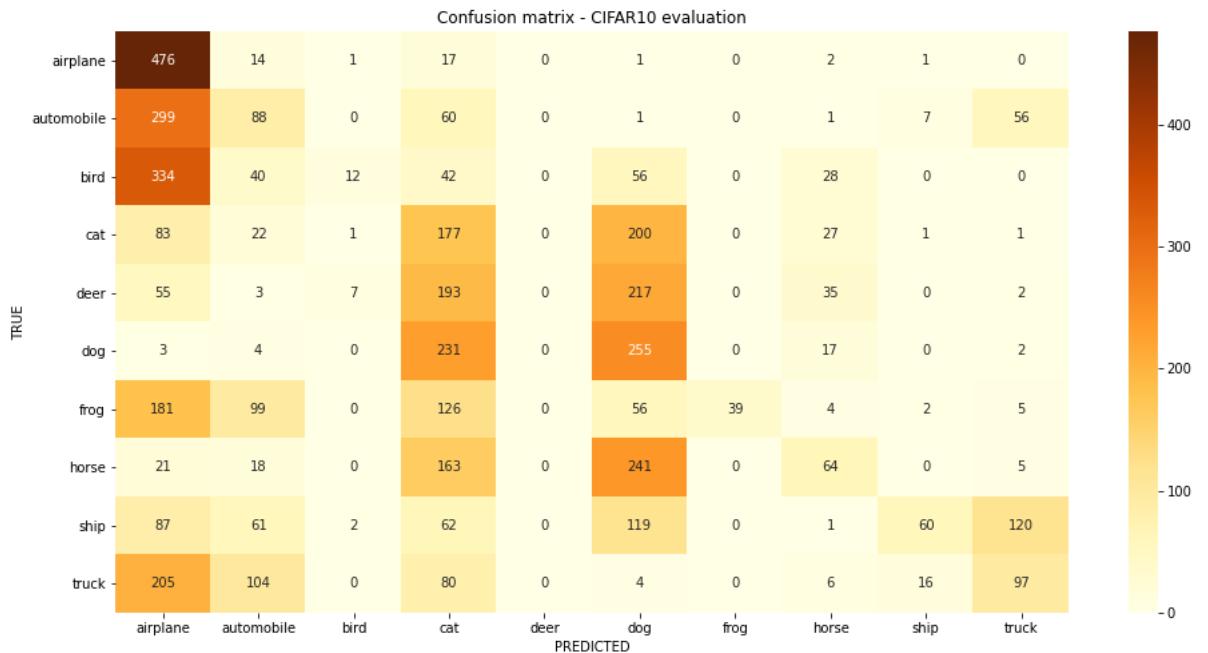


At the end, let's look at a class that has not been correctly ranked once - DEER. We can

see that the generated DEERS resemble DOGS with some element that may represent antlers. We suspect that the model failed this task because the GAN network we used was trained on the Imagenet dataset, which does not contain images containing deers. At this moment, we suspect that most DEER images will be mistaken for DOGS or CATS.

Examples from other classes are presented in Appendix 1 of this paper.

Let us now analyze which classes the images we generated were confused with. To do this let's look at the following confusion matrix. Of course, in order to fully "fool" the



classifier with our generated images - we would expect only zero values in the above matrix except for the diagonal starting at the upper left corner. Let's present what we noticed in the matrix.

1. Many classes were mistaken for the AIRPLANE class - including AUTOMOBILE, TRUCK, and BIRD. The mistakes in the classes AUTOMOBILE and TRUCK can be explained by the fact that all 3 classes depict some kind of machine, while the problem with the class BIRD is in our opinion due to the fact that the pictures of this class also often contain a generated sky (blue background).
 2. The matrix shows a clear concentration of darker color at the intersection of CAT, DEER, DOG and HORSE classes. This is probably due to the fact that, as we noted earlier - the ImageNet dataset that is the "template" for the BigGAN

network we are using contains many images of DOGS and CATS, and therefore biases the results towards them.

5.1.2 ImageNet

Let's move on to describe evaluation using the ImageNet dataset. Before conducting the experiment, we intuitively predicted that the results should be, at least, slightly better than the results from the evaluation of the CIFAR10 dataset, because ImageNet is the basis of the BigGAN network that we use in the evaluation. Therefore, let us start by presenting the parameters of the described experiment.

- GAN model used - **DeepMindBigGAN256**
- Evolutionary model used - **GA**
- Population size - **64**
- Number of model repetitions - **8**
- Number of images generated per class - **512 = 512 = 64**

As we can see, they do not differ in any way from the experiment setup in the previous section.

The results of our evaluation are as follows. We can see that we were able to fool

Class	Positive	Negative	Accuracy (%)
BANANA	111	401	21.68
CASH MACHINE	124	388	24.22
HAMMER	141	371	27.54
ICE CREAM	3	509	0.59
LLAMA	36	476	7.03
MINISKIRT	220	292	42.97
PIRATE	5	507	0.98
SHOPPING CART	125	387	24.41
WALL CLOCK	146	366	28.52
KERRY BLUE TERRIER	253	259	49.41
TOTAL	1164	3956	22.73

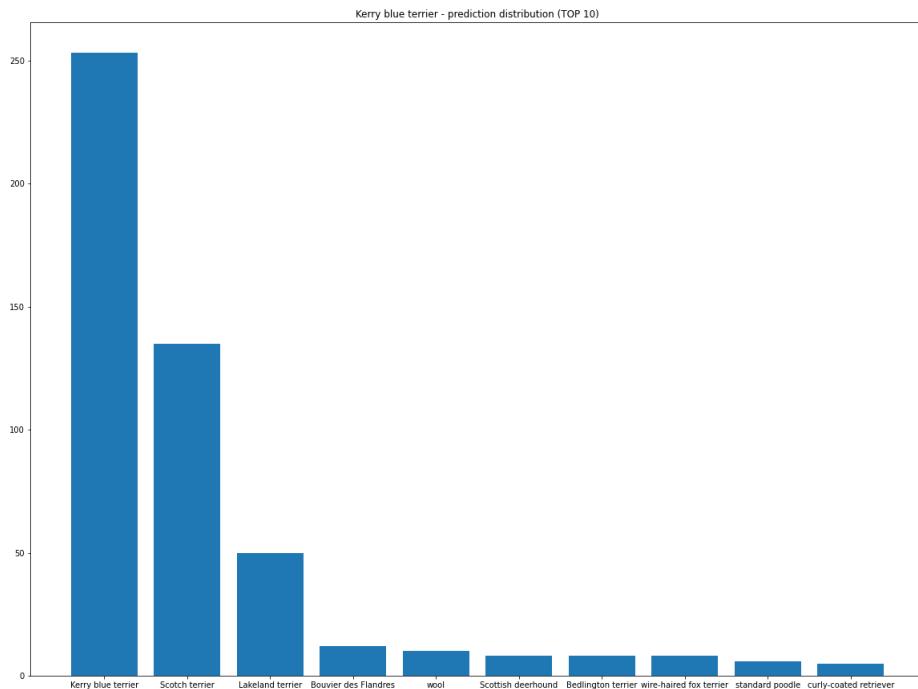
the ResNet50 classifier nearly 23% of the time, which is lower than for the CIFAR10

dataset. However, two observations should be taken into account:

1. Imagenet dataset is significantly larger (more than 20,000 categories compared to 10 in CIFAR10)
2. ResNet50 classifier on the ImageNet dataset has an accuracy of 76%.

Let us now analyze the selected classes. For each of them we will present examples of correctly and incorrectly classified generated images, as well as a graph showing with which classes the given class was most often confused. Of course, examples of images for other classes can be found in Appendix 2.

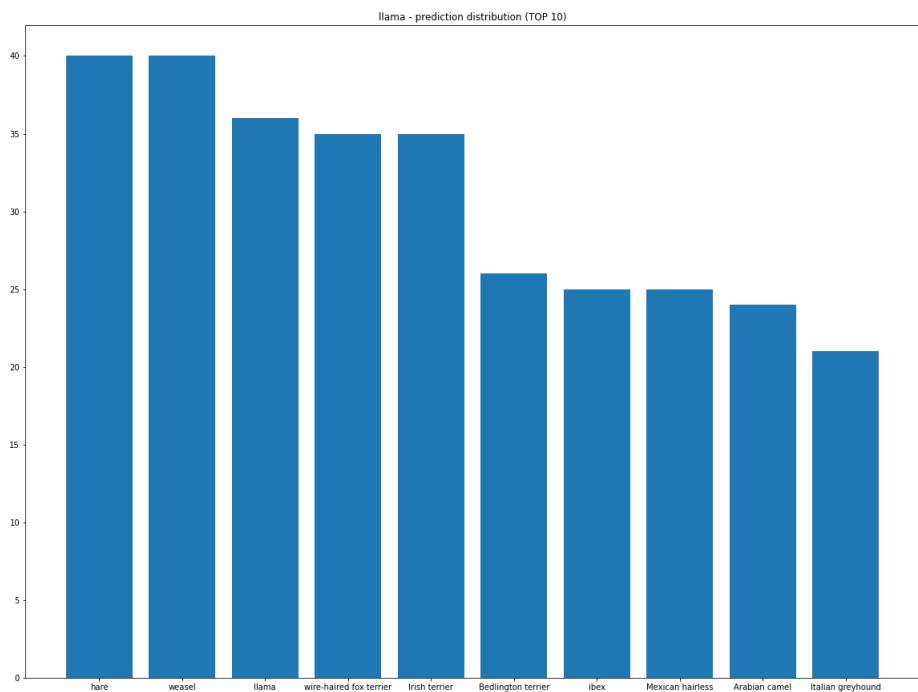
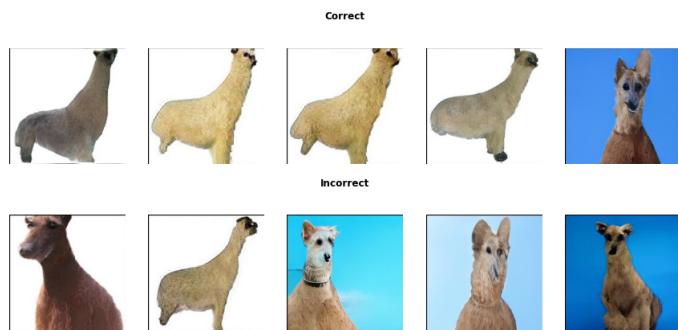
Kerry Blue Terrier



At first glance, it is clear that the images in both groups represent dogs - it is hard to judge empirically why the images in the bottom row were misclassified. However, we can make another observation - there is a noticeable problem with the context of the input query in this case - class name - KERRY BLUE TERRIER, which is encoded by the CLIP model. Specifically, the word "BLUE" is part of the dog's breed name, but when we embed the query in the model, we lose this information and experience a "leakage" of the color blue throughout the images creating, in this case, a blue dog.

Analyzing the barplot provided, we can see that in most cases our images generated for the query "Kerry Blue Terrier" were confused with other dog breeds including often other terriers. Therefore, we can conclude that if we had evaluated our model on groups of classes-for example - dogs - the accuracy of the model could have been much higher.

Llama



The LLAMA class has a very low accuracy - 7% - in our evaluation. Here we can see a noticeable difference between correct and incorrect images - while the top row resembles the shape of a LLAMA, in the bottom row we see, similarly to the section

on CIFAR10 evaluations - a bias resulting probably from too many dog images in the BigGAN model training set.

Chapter 6

Summary

Chapter 7

Appendix 1



Negative



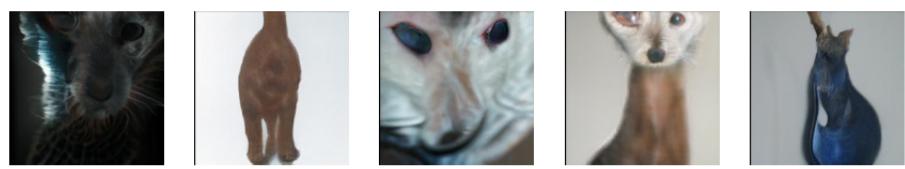
Positive



Negative



Positive



Negative



Positive



Negative



Negative



Positive



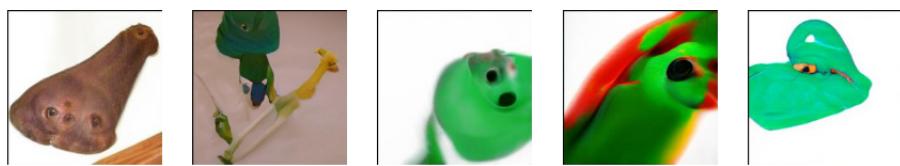
Negative



Positive



Negative



Positive



Negative



Positive



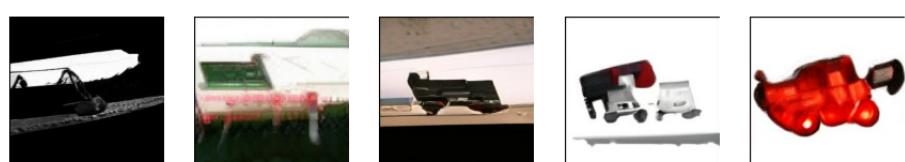
Negative



Positive



Negative

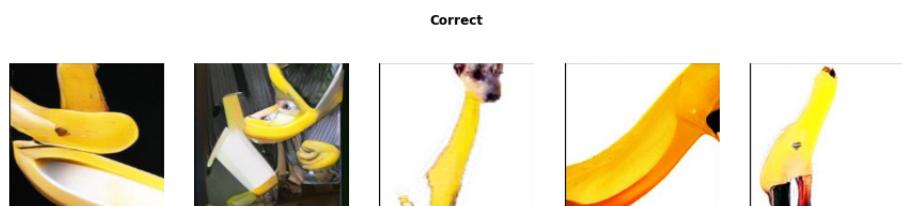


Chapter 8

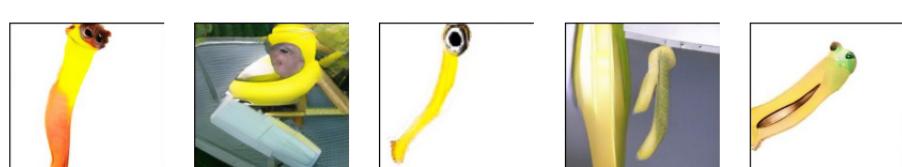
Appendix 2



Incorrect



Incorrect



Correct



Incorrect



Correct



Incorrect



Correct



Incorrect



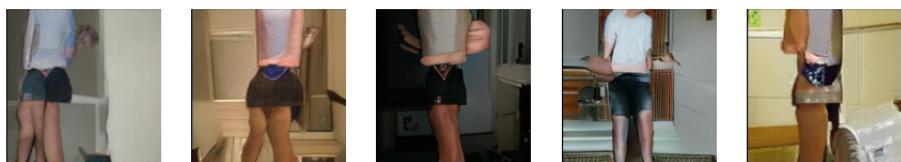
Correct



Incorrect



Correct



Incorrect



Correct



Incorrect



Correct



Incorrect



Correct



Incorrect



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.
- [7] I. Goodfellow et al.
- [8] OpenAI CLIP <https://arxiv.org/pdf/2103.00020.pdf>
- [9] <https://www.cs.toronto.edu/~kriz/cifar.html>
- [10] <https://www.image-net.org/>