UKRAINIAN CATHOLIC UNIVERSITY

MASTER THESIS

Region-Selected Image Generation with Generative Adversarial Networks

Author:
Vadym KORSHUNOV

Supervisor:
Dmytro MISHKIN

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2020

Declaration of Authorship

I, Vadym KORSHUNOV, declare that this thesis titled, "Region-Selected Image Generation with Generative Adversarial Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

“A man must cling to the belief that the incomprehensible is comprehensible; otherwise he would not try to fathom it.”

Johann Wolfgang von Goethe

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Master of Science

Region-Selected Image Generation with Generative Adversarial Networks

by Vadym KORSHUNOV

Abstract

Generative adversarial networks (GANs) are one of the most popular models capable of producing high-quality images. However, most of the works generate images from the vector of random values, without explicit control of desired output properties. We study the ways of introducing such control for the user-selected region of interest (**RoI**). First, we overview and analyze the existing works in areas of image completion (inpainting) and controllable generation. Second, we propose our model based on GANs, which united approaches from the two mentioned areas, for the controllable local content generation. Third, we evaluate the controllability of our model on three accessible datasets – Celeba, Cats, and Cars – and give numerical and visual results of our method.

Acknowledgements

I want to thank the company Ciklum for their scholarship, due to which I was able to study for a Master's Degree at UCU and have the opportunity to write this thesis. Also, I want to thank my supervisor Dmytro Mishkin for advice, assistance in ideas, and instructions during the writing of the work. And finally, I want to thank the managers of the Master's Program that they are developing it and that there is an opportunity to study Data Science in Ukraine at the world level.

Contents

List of Figures

List of Tables

List of Abbreviations

GAN(s)	**G**enerative **A**dversarial **N**etwork(s)
RoI(s)	**R**egion(s) **o**f **I**nterest
VAE(s)	**V**ariational **A**uto **E**ncoder(s)
FID	**F**réchet **I**nception **D**istance

List of Symbols

$\mathbb{D}_{KL}$	Kullback-Leibler divergence
$\mathbb{P}$	Probability measure, or distribution
$\mathbb{E}(x)$	Expectation over r.v x
$\mathbb{D}(x)$	Variance over r.v x
$p_z(x)$	Probability density function of r.v $\vec{z}$
$x \sim p_z(x)$	Sampling x from the distribution
$\mathbb{H}(x)$	Information entropy of distribution x
$\mathbb{H}(x\|y)$	Conditional information entropy of x w.r.t y
$D(f)$	Domain of function f
$X \odot Y$	Per-element multiplication between X and Y
$\|\|x\|\|_p$	L_p–norm over x ($p \geq 1$)

Chapter 1

Introduction

In this chapter we define prior definitions, thesis goals, and deep learning theoretical basics.

1.1 Thesis objective

1.1.1 Image generation

Image generation is the process of creating artificial examples of images by specified algorithm – sampling from a *generative* model. Generative model before an inference is optimized on the existed real data (trainable model) or configured by expert knowledge for concrete tasks without optimization (non-trainable model). It has many variations depending on the context and goal, for example, popular ones (Figure 1.1):

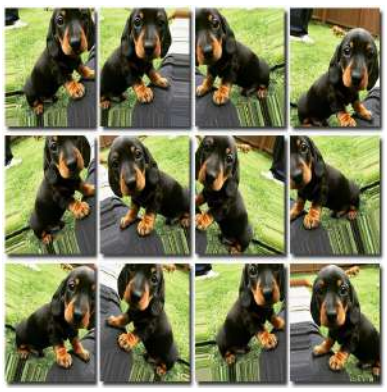
1. **Dataset augmentation**: expanding the current finite data with non-existed ones. For example, the most simple and widely used image augmentation: rotations with randomly selected angles, a saturation of the images with a random level, cropping in a random position, adding noise. Augmentation works without optimizable parameters and creates probabilistic distribution from a small amount of data. It allows training models with a high level of data diversity.

2. **Image-to-image translation**: translation of key elements from a source image, like style on structure, to a destination image. For example, draw clothes textures from sketches [1] or change the skin of horse to zebra [2], transfer style from one face image to another [3].

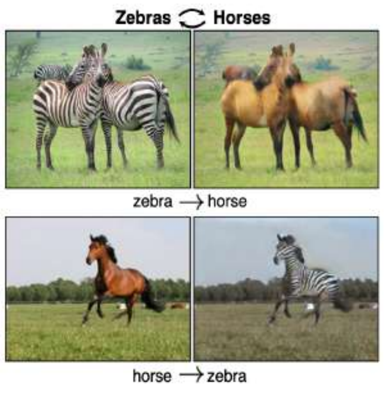
3. **Image editing and inpainting**: filling gaps in the images, videos, or merely changing existed content inside the specified mask. The key feature is that image inpainting uses spatial properties of the input data, so generation depends on the input Region of Interest (**RoI**).

4. **Deepfakes generation**: high-quality transfer of one face to another person in image or video sequence [4]. This technology causes intense debate in society and more often rejection, as it helps to spread fake news and fake non-appropriate content with celebrities or other persons. However, it stimulates the research community to create new tools for face anonymization and more sophisticated methods for fake detection.

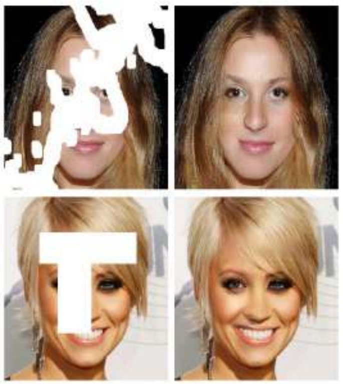
Through demonstrated examples, only the one type allows controlling generation in the sense of spatial control: image inpainting. Other methods sample data in



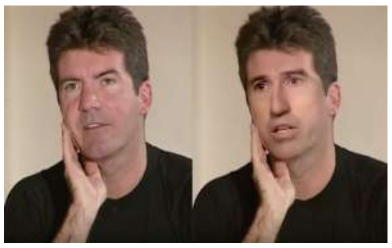
(a) Image Augmentations



(b) Image-to-image translation



(c) Image inpainting



(d) Deepfake generation

FIGURE 1.1: (a) Image augmentation process[1]: it randomly rotates and crops input image. The empty zones of the image (in case of rotations) are filled with mirrored values at the edges. (b) Image-to-image translation: e.g, color skin and texture from horses to zebras [2] (c) Image inpainting[2]: task: holes filling and generating new content inside empty parts. (d) Deepfake generation[3]: transferring one face to another person, preserving style, lighting conditions, and mouse position.

a non-control manner, i.e., the user can only take new images without changing the output properties by somehow changing the random input vector.

Formally, a generative model is a procedure of sampling from generated distribution $\mathbb{P}_g$:

$$\text{Artificial sample } x \sim p_g(\cdot|c), \tag{1.1}$$

where c – vector for control, i.e., features that controls generation process. It can be some predefined class that corresponds to the internal structure of the data (e.g., human pose for a dataset with humans, the emotion of faces), or class can be mixed with corresponding data. The general idea is that the vector for control can be different or even empty – non-conditional distribution, so the controlling over generation process will have different properties.

[1]Image source: https://rock-it.pl/images-augmentation-for-deep-learning-with-keras/

[2]Image source: https://www.eteknix.com/nvidia-shows-off-impressive-ai-photo-reconstruction-abilities/

[3]Image source: https://medium.com/@jsoverson/from-zero-to-deepfake-310551e59aa3

Within examples above, it can be a range of angles for dataset augmentation, different sketches for image translation, or part of the image for image inpainting task. Working with images, very important to properly use spatial information and introduce them to the model.

Generative model has several types depend on how we can obtain their density function $p_g(\cdot|c)$: via direct finding of the density function in *explicit* form or in the *implicit* form. Implicit models do not have standard density function, i.e., we cannot write its density function in a theoretical way, and we can only obtain samples from distribution without understanding which distribution has generated this process. Explicit distributions allow us to obtain distribution theoretically or at least approximately. The taxonomy of deep generative models explained in Figure 1.2.

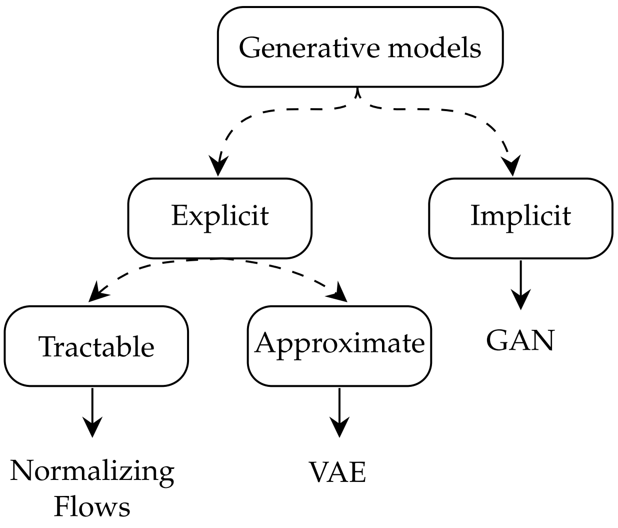


FIGURE 1.2: General taxonomy of deep generative models: depend on the task, generative models can be with implicit density (in the cases, if we need only samples from complex distribution) or with explicit density (we know theoretical or approximate density function); leaves demonstrate the members of each subtype: GAN is an implicit model, VAE – model with approximate density, and Normalizing Flows [5] compute density directly via log-likelihood estimation

In this work, we will consider and build only deep generative models, consists of neural networks. They have proven high-quality results in different tasks on arbitrary (image) data. Also, they have a simple training algorithm (backpropagation), and a transparent minimization objective procedure.

Nowadays, the dominant family of generative models for many tasks is *Generative Adversarial Networks* (GANs). We describe in detail this class in section 1.4 and will focus mainly on it.

1.1.2 Controllable generation

Typical deep generative models create new images via non-linear transformation of the prior random vector, e.g., multivariate standard normal or uniform distribution. Models forward one variable to another, but without the possibility of explicit control because the relationship between inputs of the generator and features for control is not apparent. For example, if the model generates human faces, we want to control the key features of a human face like eyes, nose, or lips. They have a particular location on the face, and therefore, we can assume that inside a specific region of the face, we can manipulate only with specific facial features. Historically, many types of generative models appeared, but there are several significant ways of how to add control to the generation process:

- *Conditional* models: generating images using prior conditional information about real data. In this class, we mean that the model has as additional input information about the class and generates entire content from scratch using this information. For example, depending on an emotion type, the model generates different faces. Nevertheless, during generation, the model does not use the spatial properties of the data and only transforms the class into a fake image.

- *Representation manipulation* models: changing in some way compressed latent vectors inside the generator network. Generative models often following autoencoder structure or use its ideas. So manipulation with internal representation significantly changes the output of the network and allows to control of some key elements in the taken images. For example, it can be averaging or subparts mixing of two internal representations, and their reconstructed images mush share properties of both originals.

- *Region-selected manipulation* models: manipulation with semantic masks for changing the desired content inside masks – inside **RoI** . Region-selected manipulation models similar to conditional models, but they introduce spatial properties in the generation process, so we separated it from each other.

The first two classes are models with global content manipulation, i.e., they create an entire picture from scratch. The last class is our topic of interest. We want to introduce a controllable generation inside the selected region of interest. Thus, we see our possible solution as an intersection between conditional, region-selected, and representations manipulation models.

1.1.3 Thesis problem statement

The goal of thesis work is the building of the deep generative architecture, which allows controlling content generation inside the defined region of interest (**RoI**) of the image. We see our research as two dependent stages:

1. Development of the model allows the generation of desired content in the specified **RoI** for the chosen image. For example, we expect that the model generates eyes inside the **RoI** defined around the eyes (Figure 1.3). This is image inpainting task, and there are a lot of research works and models trying to solve it. We analyze existing works in the area and define which ones appropriate for injecting controllability.

2. Find the way how to extract unsupervised features for controllable generation in the arbitrary specified region and check the quality of control. For example, extract features to control the color of eyes, size, or shape. We want to localize generation only inside **RoI** and add the possibility of control only inside this zone. Hence, at this stage, our model should perform image inpainting and controllable generation tasks simultaneously.

We will use GANs to solve this task. In Chapter 2, we will consider a lot of generative adversarial network types and provide cogent arguments, why we chose this type of network.

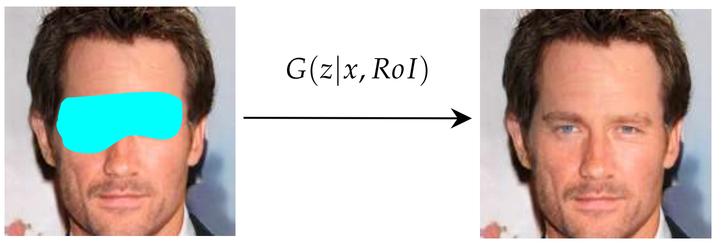


FIGURE 1.3: Region-selected generation: GAN receives the image, prior random variable $z \sim p_z(z)$ and the region, after that generates $G(z|x, \mathbf{RoI})$ with new content within **RoI**. The generator becomes conditional random model and allows controllable generation inside the **RoI** using random values from $p_z(z)$

1.1.4 Thesis structure

This work has following structure:

1. **Chapter 1**. We define prior definitions and goals of our work.
2. **Chapter 2**. We research the related work in the image inpainting and controllable generation fields, and at the end of the section, present our vision of how to inject controllable generation in the image inpainting process.
3. **Chapter 3**. We describe our solution: method concept, architectures, and loss functions.
4. **Chapters 4 and 5**. We present our results and discuss what we achieved, the advantages and limitations of our method, and possible applications.

1.2 Neural networks

We recall the basics of deep learning – neural networks. In 2020, they are default choice for solving complex tasks in image processing, natural language processing, reinforcement learning, and healthcare domains. These functions work on some ideas on how the brain works: a *neuron* accumulates the responses from all surrounding neurons a_l depend on the power of signal w_l from each neuron. On the mathematical language, it can be reformulated through the weighted summation of values from other L neurons to the concrete one:

$$n = \sigma\Big(\sum_{l=1}^{L} w_l \cdot a_l + b\Big), \tag{1.2}$$

where σ – threshold function, which modeled the size of the output signal depends on accumulated one and limited the input signal (the threshold passes only if the input signal is significant). b – bias in the neuron (analog of a constant signal).

If destination neuron is not alone, the same procedure can be repeated for all output neurons. Moreover, the entire formulas can be written in matrix notation:

$$n_j = \sigma\Big(\sum_{i=1}^{L} w_{il} \cdot a_l + b_j\Big),\ j = 1...M \quad \sim \quad \vec{n} = \sigma\big(W\vec{a} + \vec{b}\big) \tag{1.3}$$

where σ – point-wise applying function, and neurons, weights, and biases are in vector notations.

The simplest neural networks work using only fully-connected layers (affine transformation of the input vector, i.e., linear transformation with adding bias) with non-linear functions f_i. We can write neural network as a recurrent equation:

$$x_{i+1} = f_{i+1}(W_{i+1} \cdot x_i + b_{i+1}),\ i = 0...n, \tag{1.4}$$

where W_{i+1} – matrix with consistent dimensions for multiplying of the vector x_i, and x_0 – input vector

However, such class of neural networks is theoretically argumented in the case of $n = 2$ [6], but is not appropriate for fast processing of the images with a big resolution, and it suffers on gradient vanishing, slow convergence and an overwhelming number of parameters. Instead, the convolutional neural networks invented for image processing. They perform similar to the vision of living things: sliding view over high-dimensional *spatial* input. It consists of *parametrized convolutions*, that extracts spatial information from input. As a human, it iteratively processes spatial information and tries to capture simple patterns like lines or angles and then construct a more complex pattern using previous ones. The parameters of the convolution – kernels K – slices over input X and collects weighted response for all slice depends on the size of the kernel. Simultaneously, there can be few kernels, and the overall procedure can be formalized in complex summation (with bias B):

$$Y[C_{out},\ i,\ j] = \sum_{c=1}^{C_{in}} \sum_{k=1}^{K_h} \sum_{l=1}^{K_w} X[c,\ i+k-1,\ j+l-1] \cdot K[C_{out},\ c,\ i,\ j] + b[C_{out}] \tag{1.5}$$

This is affine operator, hence W – matrix, b – vector: $Y = WX + b$. So, this is almost the same as the fully-connected layer, but with strong spatial integrated properties, and many elements of matrix W will be zero. In modern programming, software computer convolution very quickly and in a parallel way, using few graphical processing units simultaneously. So they are prevalent in image processing and evolved a lot since the new dawn of the neural networks in 2012 [7].

Important to say that, using convolutions and fully-connected layers, we can perform the continuous differentiable transformation from one image or vector with one resolution to another image or vector with *arbitary* resolution. Furthermore, this fact opened the door for neural networks to solve almost all possible tasks, like image reconstruction, super-resolution tasks, or classification. We should only take many blocks with different non-linear functions between them and apply them to our data to obtain the result.

1.3 Autoencoders

Autoencoder is neural network (can be fully-connected or fully-convolutional) with two different subnetworks: *an encoder* and *a decoder*. Autoencoders operates on the hypothesis that real data lies on low-dimensional manifolds, and we can find functional mappings onto and from this manifold. Hence, the encoder compresses input into underlying representation – the hidden state of the data. Depends on the architecture, convolutional, or entirely fully-connected, it can be tensor or vector correspondingly. The decoder performs the inverse operation: decompresses data into real representation (with the same size as input).

Exist generative version of autoencoders – Variational Autoencoders (VAEs). It interprets outputs of the encoder as parameters of some distribution: in the most

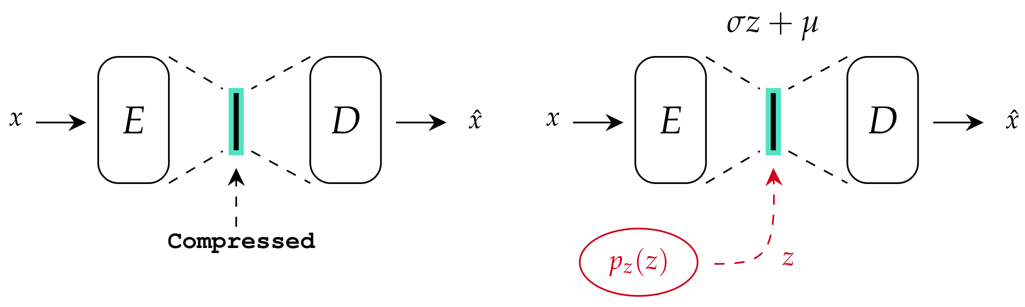


FIGURE 1.4: (a) Autoencoder forward propagation: encoder E receives input vector x and compresses it into new representation. Decoder restores $\hat{x}$ from the internal representation. (b) Variational autoencoder forward propagation: almost the same as (a), but encoder scales and biases input random vector with mean and variances, obtained from encoder

straightforward cases, mean and variance of gaussian distribution with independent components, and samples compressed vectors from distribution $Q(z|x) \approx \mathbb{P}\{z|x\}$. The log-likelihood principle for VAE's is very complex numerical task (integral optimization), so used evidence lower bound (ELBO):

$$\log \mathbb{P}\{x|\theta_E, \theta_D\} \geq \mathbb{E}_{z\sim Q(z|x)} \log \mathbb{P}\{x|z\} - \mathbb{D}_{KL}\{Q(z|x)||\mathbb{P}\{z\}\} \underset{\theta_E,\, \theta_D}{\longrightarrow} \max \tag{1.6}$$

If we maximize the lower bound of the log-likelihood function, we approximately will maximize the log-likelihood itself. This is the main principle of variational autoencoders. It used in many formulations in other research works on similar thematics. The detailed theory behind VAEs lighted in the paper [8].

1.4 Generative adversarial networks

The generative adversarial networks become a popular research topic in deep learning since their initial appearance in **[9]**. It is a new type of deep learning model consists of two networks: a *discriminator* and a *generator*. The working abilities of GANs is provided by competition between discriminator and generator. This competition has been demonstrating a perfect visual effect on the generated content, and these networks are recommended as one of the best methods for image generation. For example, GANs can generate high-quality human faces that are almost indistinguishable from real faces[4].

In the original definition, the goal of the discriminator with parameters θ_D is to determine if an input image is fake or not, and give the probability $D(\cdot) = \mathbb{P}_{real}(\cdot)$ that image is real. The discriminator should give high probability on the real samples and give low probability on the artificial samples. The goal of the generator with parameters θ_G is to create images as realistic as possible, i.e., generate images $G(z)$ from prior $z \sim p_z(z)$ that fool discriminator, and network makes a mistake in its predictions. This intuitive description of loss functions for generator and discriminator can be formalized using the maximum log-likelihood principle:

$$L_G = \mathbb{E}_{z\sim p_z(z)} \log(1 - D(G(z))) \underset{\theta_G}{\longrightarrow} \min \tag{1.7}$$

[4]This person is not exist: https://thispersondoesnotexist.com

$$L_D = \mathbb{E}_{x \sim p_d(x)} \log D(x) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z)) \xrightarrow[\theta_D]{} \max \tag{1.8}$$

Formally, networks play in the non-cooperative game, so the one loss function united two separate objectives for generator and discriminator:

$$L(D,\, G) = \mathbb{E}_{x \sim p_d(x)} \log D(x) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z)) \tag{1.9}$$

The optimal discriminator and generator is solution to minimax game:

$$D^*,\, G^* \leftarrow \min_{\theta_G} \max_{\theta_D} L(D,\, G) \tag{1.10}$$

The representation of generative adversarial networks from the game theory point of view gives critical differences between the training of standard deep learning models with the only one loss function and training GANs. The success of the only one agent does not define the overall success for the entire GAN because their optimization depends on each other, and their quality must increase slowly together. The generator's gradient flows via the discriminator network. Hence, the generator's updates strongly depend on the information from the discriminator. The discriminator's gradient depends on real and fake data and hence depends on the generator's outputs (Figure 1.5).

For example, if the discriminator converges in a few iterations, then the generator will not have enough information to generate realistic outputs. Hence the discriminator's "knowledge" is not based on real information. So the generator does not have appropriate output from the discriminator as well. In the same situation, if the generator converges very fast, the discriminator will not be able to classify correctly between real and non-real samples, and this information will ruin the generator's outputs in long-term training. Thus, the failure of one of the models will lead to the failure of both agents, so GAN training should be a careful and gradual process.

Nevertheless, researchers developed a lot of different losses for GAN to overcome existed problems. Losses differ from the classic interpretation and do not support clear minimax game formulation (Table 1.1). But, the GAN with other losses than original saves the competition properties, and their training is sophisticated as well.

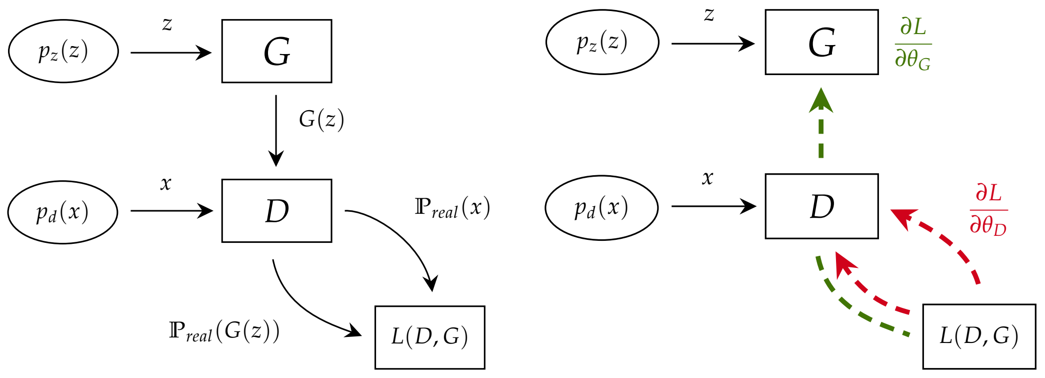
Hovever, Goodfellow et al in [9] admit that classic loss for generator (1.7) has convergence problems, so they recommended to use *non-saturating* version of standard generator's loss:

$$L_G = \mathbb{E}_{z \sim p_z(z)} \log D(G(z)) \xrightarrow[\theta_G]{} \max \tag{1.11}$$

The key difference between (1.7) and (1.11) is that their gradients w.r.t $D(G(z))$ have different behaviours during optimization: non-saturating loss has bigger norm of gradients on the domain $(0,\, 1)$ than classic one. Hence, optimization will be faster.

The next major steps in adversarial loss construction are *Least Squares GAN* [10], or LSGAN, and *Wasserstein GAN*, or WGAN [11]. Least Squares GAN changes classic loss with logarithm on the more smooth function – $f(x) = x^2$.

Wasserstein GAN derives a new vision on the adversarial loss function. Researches discovered that classic loss function, which based on the Kullback–Leibler divergence $\mathbb{D}_{KL}$, does not demonstrate continuous behavior in some specific cases. When the distribution of images $\mathbb{P}_g$, produced by a generator, has different support or lies on the other low-dimensional manifold than real distribution $\mathbb{P}_r$, the final


$p_z(z)$
z
G
$G(z)$
$p_d(x)$
x
D
$\mathbb{P}_{real}(x)$
$\mathbb{P}_{real}(G(z))$
$L(D,G)$
$p_z(z)$
z
G
$\frac{\partial L}{\partial \theta_G}$
$p_d(x)$
x
D
$\frac{\partial L}{\partial \theta_D}$
$L(D,G)$

(a) Forward pass (b) Backward pass

FIGURE 1.5: (a) GAN forward propagation: generator G receives prior random value $z \sim p_z(z)$ and transforms it to new distribution $G(z)$. Discriminator D receives a sample x from real distribution $p_d(x)$, fake sample $G(z)$ and determines if inputs are real with probabilities $\mathbb{P}_{real}(x)$ and $\mathbb{P}_{real}(G(z))$. Finally, loss function $L(D, G)$ receives the discriminator outputs. (b) GAN backward propagation: green and red dashed arrows show gradients flow w.r.t the generator's and the discriminator's parameters correspondingly. Gradient of the D accumulates from two directions; gradient of the G passes through the D in the one direction, and D serves as bottleneck for G

Loss type	**Discriminator**	**Generator**
Standard	$-\mathbb{E}_x \log D(x) - \mathbb{E}_z \log(1 - D(G(z))$	$\mathbb{E}_z \log(1 - D(G(z)))$
Non-saturating	$-\mathbb{E}_x \log D(x) - \mathbb{E}_z \log(1 - D(G(z))$	$-\mathbb{E}_z \log D(G(z))$
Hinge	$\mathbb{E}_x \max(0,\ 1 - D(x)) + \mathbb{E}_z \max(0,\ 1 + D(G(z)))$	$-\mathbb{E}_z D(G(z))$
LSGAN	$\mathbb{E}_x(D(x) - 1)^2 + \mathbb{E}_z D(G(z))^2$	$\mathbb{E}_z(D(G(z)) - 1)^2$
WGAN	$-\mathbb{E}_x D(x) + \mathbb{E}_z D(G(z))$	$-\mathbb{E}_z D(G(z))$
WGAN-GP	$-\mathbb{E}_x D(x) + \mathbb{E}_z D(G(z)) + \lambda \mathbb{E}_t(\|\|\nabla_t D(t)\|\| - 1)^2$	$-\mathbb{E}_z D(G(z))$

TABLE 1.1: The several popular GAN loss functions, based on different outputs of discriminator. The first two – *Standard* and *Non-saturating* – used discriminator's and generator's outputs limited to probabilistic range $(0,\ 1)$; other losses used outputs in real range $(-\infty,\ +\infty)$; the *WGAN-GP* loss used additional gradient penalty limited the discriminator's class function; loss functions are written in terms of minimization objective

properties of the $\mathbb{D}_{KL}(\mathbb{P}_r||\mathbb{P}_g)$ has undefined behavior, and infinite value of $\mathbb{D}_{KL}$ is possible. The solution which authors proposed – change the basis of measuring the distances between distributions, and set as primary measure *Earth-Mover distance* (EM):

$$W(\mathbb{P}_r,\ \mathbb{P}_g) = \inf_{\gamma \in \prod(\mathbb{P}_r,\ \mathbb{P}_g)} \mathbb{E}_{(x,\ y)\sim\gamma}\{||x - y||\}, \tag{1.12}$$

where $\prod(\mathbb{P}_r,\ \mathbb{P}_g)$ – the set of all joint distributions between marginal $\mathbb{P}_r$ and $\mathbb{P}_g$

Intuitively, it measures the mean distance between distributions for specific joint distribution for which this measure will be minimal. It also has physical interpretation: γ realizes transportation plan from mass $\mathbb{P}_r$ to $\mathbb{P}_g$, and EM-distance helps to find the optimal transportation plan with the lowest cost in the sense of distance $||\cdot||$.

However, the (1.12) has two significant problems. First, intractable $\mathbb{P}_r$ distribution – we do not know the explicit form of this value. Second, the equation depends on the γ in a non-differentiable way, and we know nothing about $\gamma \in \prod(\mathbb{P}_r,\ \mathbb{P}_g)$.

So, authors proposed transform equation (1.12) in more simple way. They used Kantorovich-Rubinstein duality [12] and simplified equation:

$$W(\mathbb{P}_r,\ \mathbb{P}_g) = \sup_{||f||_L \leq 1} \Big\{ \mathbb{E}_{x\sim\mathbb{P}_r} f(x) - \mathbb{E}_{x\sim\mathbb{P}_g} f(x) \Big\}, \tag{1.13}$$

where $||f||_L \leq 1$ defines the class of 1-Lipschitz (or K-Lipschitz) functions:

$$\forall x,\ y \in D(f) :\ ||f(x) - f(y)|| \leq 1 \cdot ||x - y||\ (\text{or } K \cdot ||x - y||) \tag{1.14}$$

The new equation (1.13) supports direct optimization if the supremum is reachable. Authors proposed to clip the domain of the function's weights – hence, it automatically will have maximum within the domain as a continuous function has maximum value on the compact set and function will be K-Lipschitz as well. During optimization, arguments clipping must be after every iteration (to ensure theoretical consistency).

Important to say, using this loss function, the discriminator does not give probability $\mathbb{P}_{real}(\cdot)$. It gives the continuous score $D(x) \in \mathbb{R}$, and discriminator sometimes recalled as *critic*. In this work, we call discriminator or critic only as discriminator – for simplification of terminology.

However, weight clipping slowed training process, so [13] proposed additional *Gradient penalty* to WGAN loss called *WGAN-GP*:

$$L_D = -\mathbb{E}_x D(x) + \mathbb{E}_z D(G(z)) + \lambda \mathbb{E}_t(||\nabla_t D(t)|| - 1)^2 \xrightarrow[\theta_D]{} \min \tag{1.15}$$

The gradient penalty limited the norm of the gradient to 1 and hence limited the functions to K-Lipschitz, because:

$$\forall x \in D(f) :\ ||f'(x)|| < A \implies ||f(x) - f(y)|| < A \cdot ||x - y|| \tag{1.16}$$

During optimization, $\nabla_t D(t)$ calculated for $t = \alpha \cdot x_r + (1-\alpha)x_g$, where $x_r \in \mathbb{P}_r$ and $x_g \in \mathbb{P}_g$. Non-trainable parameter α is sampled from uniform distribution $U(0,\ 1)$, and hence during optimization gradient is computed on the line between real and fake sample. In case of success training, i.e $\mathbb{P}_r \approx \mathbb{P}_g$, the $t = \alpha \cdot x_r + (1 - \alpha) \cdot x_g$ will cover all domain of discriminator and limit its gradient on all $D(f)$.

Also, researchers often used Hinge loss for image inpainting task and loss is demonstrated good visual results on restored image parts [14]:

$$\begin{cases} L_D = \mathbb{E}_x \max(0,\ 1 - D(x)) + \mathbb{E}_z \max(0,\ 1 + D(G(z)) \xrightarrow[\theta_D]{} \min \\ L_G = -\mathbb{E}_z D(G(z)) \xrightarrow[\theta_G]{} \min \end{cases} \tag{1.17}$$

It updates only discriminator's part of the loss and still the same, as in WGAN, the loss for generator.

Chapter 2

Related work

In this chapter, we describe research works in the image inpainting and controllable generation tasks domains.

2.1 Classic models

Classic computer vision image inpainting algorithms were built around the idea of iterative filling of empty pixels using non-masked parts of an image. Image inpainting technique based on the Fast Marching Method [15] and PatchMatch [16] researched in 2000th and used in the OpenCV library for fast image inpainting. The core idea of algorithms to fill **RoI** from the boundaries to the center of the region of interest, based on the values from neighbors pixels. Thus, in these approaches, subsequent pixels become dependent on the predicted ones, and a large error (in comparison with the real image) accumulates towards the center of the region, and the quality of filling significantly drops with increasing mask size. While the first algorithm used deterministic filling using the closest neighbors, the PatchMatch used specifically chosen patches of images to fill the desired region and enable quick processing of the image (in online or in the graphical-used interface of editing tools). PatchMatch allows us to generate new content inside **RoI**, but do not allow to control generation over image: the unambiguous mapping between input and output makes it impossible to control this generation using other variables than **RoI**.

Moreover, classic computer vision algorithms filled the empty zones based only on the statistics of the rest of the image. As a result, the filled zone is blurred and does not demonstrate real properties. They cannot create new smart content depending on the internal properties of underlying data.

2.2 Conditional generative models

Conditional GAN [17] has the same structure as classic GAN, but instead of distributions $p_z(z)$ and $p_d(x)$ used $p_z(z|c)$ and $p_d(x|c)$, where c – discrete label of data sample x. The entire process of training and inference remains the same, but along with random noise, the generator receives new information about the condition, concatenated with z. The conditional labels c define the prior information about data. Conditional distribution allows to control of the generation of predefined classes separately, and the generator creates new content much better because it has information that separates classes.

For example, image class "smiling face" introduced additional information in the generator, and it generates only smiling faces during the sampling procedure. However, the model strongly depends on the labeled data (images with different classes) and cannot control generation without labels, so it is the main disadvantage of the method.

Controllable Generative Adversarial Network [18] - **ControlGAN** - also manipulates with conditional deterministic variables, but introduces the additional *classifier* network to compress semantic information and use it in the generator as well. Hence, the entire GAN architecture consists of three blocks: generator competes with discriminator and classifier (Figure 2.1). The updated goal of the generator is to create samples fool discriminator and on which classifier will make a correct prediction. For example, for class "smiling faces" the generator must create not only smiling faces, but also classifier must predict "smiling" faces for artificial images from $G(\cdot|\text{"smiling faces"})$. Additional labels in the input for generator forced the network to make image instances as more similar to condition distribution $\mathbb{P}_r\{\cdot|c\}$ for all classes c, because the generator produces images with different, per-class features. The model also controls the trade-off between training of the classifier and training of the generator, using the smoothed ratio between classifier's losses on the real and generated data. In these settings, authors followed BEGAN [19], which used ideas from control theory to add a balance between discriminator and generator within the training process. As a result, ControlGAN has better quality than Conditional GAN, but the model also uses the human-labeled dataset with different conditions.

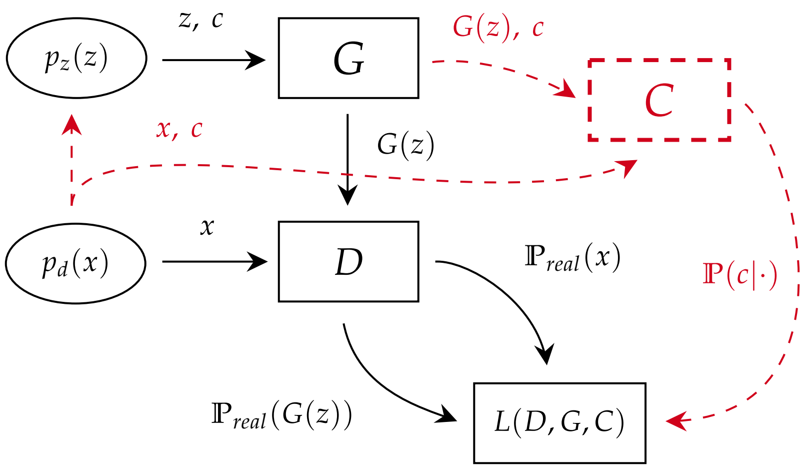


FIGURE 2.1: The forward propagation of ControlGAN: a generator G receives additional class label c and generates content concerning this condition; a discriminator D has connection only with G; a classifier C makes predictions on the real and generated data, which used in the loss $L(D, G, C)$

InfoGAN by [20] upgraded conditional model with features for control in an unsupervised way. The key idea is to use additional probabilistic distribution $p_c(c)$ for control together with initial $p_z(z)$ and create dependency between generated data and distribution for control via mutual information maximization:

$$I(G(z,c),\ c) = \mathbb{D}_{KL}(p_{G(z,\ c),\ c} - p_{G(z,\ c)} \cdot p_c) = \mathbb{H}(c) - \mathbb{H}(c|G(z,c)) \rightarrow \max \quad (2.1)$$

The objective increases the divergence between the product of marginals and mutual distribution, hence "increases" the level of dependency between random variables. This setting allows *disentangled learning* of the real data, i.e., features for control with high probability will correspond to specific visual aspects of the data (e.g., faces structures, size of the eyes or nose position). However, the direct optimization of $I(\cdot, \cdot)$ is not possible, because it depends on the distribution $\mathbb{P}(c|x)$, which is intractable. Authors tried to approximate $\mathbb{P}(c|x)$ via auxiliary distribution $Q(c|x)$ with parameters θ_Q, so they used Variational Lower Bound $L_I(\cdot, \cdot)$ instead:

$$I(G(z, c), c) \geq \mathbb{E}_{c\sim p_c(c),\, x\sim G(z,\, c)} \log Q(c|x) + \mathbb{H}(c) = L_I(\cdot,\ \cdot) \tag{2.2}$$

The new objective is added together with standard GAN loss objective:

$$D^*, G^* \leftarrow \min_{\theta_G,\, \theta_Q} \max_{\theta_D} \{L(D, G) - \alpha L_I(G(z, c), c)\} \tag{2.3}$$

Distribution for control $p_c(c)$ can be continuous vector, like multivariate gaussian distribution or few independently distributed uniform values, and can be discrete vector with finite number of values with probabilities p_i for every c_i, $i = 1...S$. In practice, discrete and continuous distributions perform different control functions: while continuous random variable corresponds to the continuous changing of the images (rotations or lighting), discrete variables significantly switches the data (a type of digits, smiling or not smiling face). Control with $p_c(c)$ is simple: after training, we can understand distribution as sliders with a defined minimum and maximum values for continuous variables (e.g., minimum and maximum values for uniform distribution), and set of discrete variables as a drop-down list.

Elastic-InfoGAN [21] extends InfoGAN with differentiable discrete $p_c(c)$ w.r.t generator's weights. InfoGAN has a limitation in case of training data that has imbalanced distribution, so the authors proposed directly estimate probabilities of discrete distribution for control using Gumbel-Softmax distribution. Also, authors constrained entropy of a posterior distribution $Q(c|x) \approx \mathbb{P}\{c|x\}$: it should be low, because created dependency between $\mathbb{P}_g$ and $\mathbb{P}_c$ creates an intractable mapping between real samples and conditional variables, reduces the level of uncertainty between random values:

$$L_{\mathbb{H}}(Q) = \mathbb{H}(Q(c|x)) + \mathbb{H}(Q(c|\delta(x))) \xrightarrow[\theta_G]{} \min \tag{2.4}$$

Authors used this loss both for original input $x \sim p_d(x)$ and also for augmented datasamples $\delta(x)$, $\delta \sim \mathbb{P}_\delta(\delta)$ - augmentation distribution. In this way, they want to force neural networks create representations invariant for specific class of augmentations (e.g rotations and scales), which can transfer control from this augmentations to internal properties of the data.

2.3 Representation manipulation models

StyleGAN [3] brings the idea of style transfer into generation process for better control of the output images. The style transfer is possible by **Adaptive Instance Normalization** layer [22], special version of Batch Normalization [23]. The Adaptive Instance Normalization adds direct scaling and biasing of the internal pictures representations – style modulation – and allows to control transfer style from one image to another:

$$\hat{x}(x, w) = \gamma(w) \cdot \frac{x - \mathbb{E}(x)}{\sqrt{\mathbb{D}(x) + \epsilon}} + \beta(w), \tag{2.5}$$

where w - vector for control, $\gamma(w)$ and $\beta(w)$ - scale and bias modulation factors.

Normalization allows us to control the output properties of the images only with scaling and biasing of the internal representations – it is an empirical fact, obtained in [24], and used for fast style transfer. The same convolution filters used with different coefficients in the normalization layer – and it allows us to generate completely different outputs.

Additional non-linear transform $W : z \to w$ of the input random variable $z \sim p_z(z)$ before generator stabilizes the style transfer process and allows to direct control of the generated images via complex random value. The authors suggest this technique helps to approximate $\mathbb{P}_r$ by the distribution of generator $\mathbb{P}_g$. The StyleGAN uses **Progressive GAN** [25] idea with image size increasing. Hence, the model tends to learn from small blurred images to high-quality pictures.

HoloGAN introduced by [26] is a novel GAN architecture that adds information about the volume in the model (Figure 2.2). HoloGAN shares architecture a lot with StyleGAN, but add two additional layers inside of the generator: 3D-Transformation with parameters θ and 2D-Projector. These layers perform a 3D-rotation of the internal representation and project it on some plane. The key idea is to introduce bias about 3D-world into the model, then manipulate with 3D-view of the data and finally perform volume-to-surface projection. Manipulation is possible by the 3D-Transformation layer, which has parameters θ defined outside the model, i.e., they were not used during optimization. Instead, parameters are randomly sampled, and the model learns to generate data in all possible 3D configurations, like to see the same object in 3D-space from different points of view. We consider this architecture very interesting for our research, so we use the idea about the 3D-Transformation layer and change it for our purposes.

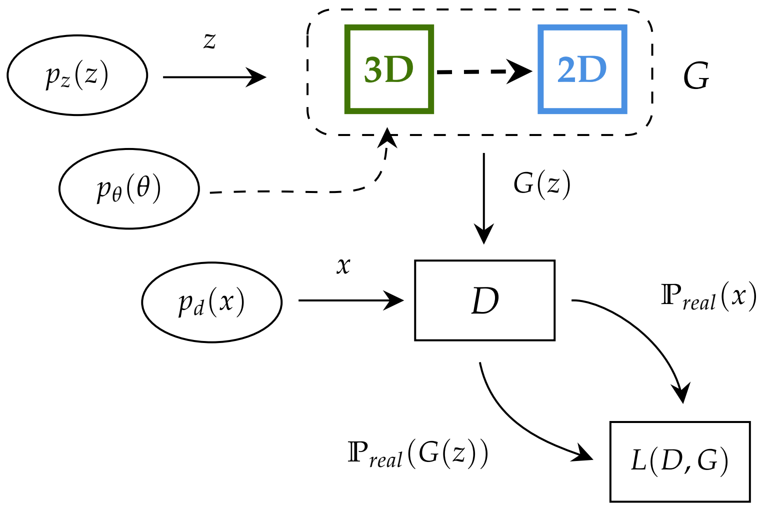


FIGURE 2.2: The forward propagation of HoloGAN: a generator G receives additional class label c and generates content concerning this condition; a discriminator D has connection only with G; a classifier C makes predictions on the real and generated data, which used in the loss $L(D, G)$

Research work **β-VAE: Learning Basic Visual Concepts With A Constrained Variational Framework** [27] introduces β-VAE. It allows to disentangle images representations inside autoencoder, using simple β-scaling of the $\mathbb{D}_{KL}$ term in the variational loss:

$$\log \mathbb{P}\{x|\theta_E,\, \theta_D\} \geq \mathbb{E}_{z\sim Q(z|x)} \log \mathbb{P}\{x|z\} - \beta \mathbb{D}_{KL}\{Q(z|x)||P(z)\} \underset{\theta_E,\, \theta_D}{\longrightarrow} \max \tag{2.6}$$

However, the authors report model has a trade-off between the accuracy of the reconstruction and the possibility of explicit control. In their next work **Understanding disentangling in β-VAE** [28] they tried to reduce this problem. They also argued (from a theoretical perspective) why controllability is possible only by changing β coefficient.

2.4 Region-selected manipulation models

The goal of **GAN Dissection: Visualizing and understanding generative adversarial networks** [29] work is to determine and control the internal representations for generating desired content in the chosen **RoI**. Authors developed an interactive online tool[1] based on this paper. They manipulated with generator's latent representations that correspond to specified **RoI** and semantic class (e.g., grass, door, or sky). They found the dependency between convolutional activation maps of different channels in the intermediate layer of the generator and the **RoI**. For these purposes, the authors used a two-stage approach:

1. <u>Dissection</u>: finding the best overlap with $\textbf{RoI}_r$ and zone with highest activations $\textbf{RoI}_{g,\,c}$ for each class c via Intersection over Union (IoU) measuring:

$$\textbf{IoU}(\textbf{RoI}_r, \textbf{RoI}_{g,\,c}) = \frac{\mu(\textbf{RoI}_r \cap \textbf{RoI}_{g,\,c})}{\mu(\textbf{RoI}_r \cup \textbf{RoI}_{g,\,c})} \tag{2.7}$$

   where μ - square of the **RoI** on the image.

2. <u>Intervention</u>: manipulate with the best match to maximize the square of chosen class in the RoI. It is achieved by direct changing features within $\textbf{RoI}_{g,\,c}$ to obtain highest scores for chosen class k.

The authors used custom loss function in the <u>Intervention</u> step to formalize the changes in the feature maps. They maximized the class appearance in the chosen region of interest with respect to activation. Though, their work strongly depends on the dataset with labeled classes and the segmentation mask of each class. As the result, they can control generation inside **RoI** using discrete classes (e.g sky $\rightarrow$ grass).

Model's two-stage manipulation procedure highly depends on the labeled data (both classes and segmentation mask), which limited application of this model to non-labeled datasets.

Patch-Based Image Inpainting with GANs [30] brings new ideas for adversarial convolutional models: authors extended discriminator from single-score to multi-score model. It has called *PatchGAN*. The discriminator now operates on the local and global levels and gives the probability that input is real not only for the entire image but for different parts of the image as well. The structure of discriminator allows reducing image resolution to some middle size between original and single-scalar output. It can be interpreted as scores for some part of the original image

[1]GAN Dissection painter: http://gandissect.res.ibm.com/ganpaint.html

(depends on the accumulated receptive field from the first convolution to the last one). The scores for different patches share information via a fully-connected layer, so it increases the quality of the discriminator. They also discussed the use of strided convolutions for image inpainting task – it covers the bigger receptive field, has a smaller number of parameters (if compare with non-strided convolution with the same receptive field). Model used L_1-loss for reconstruction:

$$L_1(G) = \mathbb{E}_{(z,\, x)\sim\, (\mathbb{P}_z,\, \mathbb{P}_r)} \left|\left| x - G(z|\mathbf{RoI}) \right|\right|_1 \xrightarrow[\theta_G]{} \min \tag{2.8}$$

Image Inpainting for Irregular Holes Using Partial Convolutions [31] tries to expand convolution operation on the case where part of the image is erased. They researched that typical convolution is not appropriate for this task, because it depends on the empty regions and can produce visual non-correct artifacts, so were proposed to modify convolution than it depends only on the non-empty part of the image – *partial convolution*. From first to the last layer, partial convolutions filled the empty parts of **RoI**, and changes mask as well. The model with such layer is not generative, i.e., model is not random, and it changes if mask changes. Authors designed specific loss functions for style and perceptual performance of the generated images, taken from style transfer literature by Gatys et al [32]:

Let $x_g = G(z)$, – generated image from noise and $x_c = \mathbf{RoI} \cdot x_g + (1 - \mathbf{RoI}) \cdot x_r$ – completed image with non-erased part. Then, perceptual loss defined as:

$$L_{per} = \mathbb{E}_{x,\, z\sim\mathbb{P}_r,\, \mathbb{P}_g} \sum_{i=1}^{L} \left|\left| Q_i(x_c) - Q_i(x_r) \right|\right|_1 + \left|\left| Q_i(x_g) - Q_i(x_r) \right|\right|_1 \tag{2.9}$$

where Q – pretrained network on the ImageNet dataset [33]. The main sense is Q has seen a lot of real data samples, so it can help transfer "real" style to generated images and concentrate generator to create a realistic filling of the chosen **RoI**. Authors also included Total-Variation loss L_{TV}, which is a penalty for increasing of the completed outputs continuity:

$$L_{TV} = \sum_{(i,\, j)\in\mathbf{RoI}} |x_c^{i,\, j+1} - x_c^{i,\, j}| + |x_c^{i,\, j} - x_c^{i+1,\, j}| \tag{2.10}$$

Free-Form Image Inpainting with Gated Convolution [34] paper introduces a new type of convolution with the gating mechanism depends on the input mask. It is a modification of partial convolution and allows the differentiable flow of masked image through a network:

$$\begin{cases} Y_f = \mathbf{Conv2d}(X,\, K_f) \\ Y_m = \mathbf{Conv2d}(X,\, K_m) \\ Y = f(Y_m) \odot \sigma(Y_m) \end{cases} \tag{2.11}$$

where $f(\cdot)$ and $\sigma(\cdot)$ – non-linear functions. Multiplication between two tensors realizes gating mechanism (authors used sigmoid for this purpose).

They also used PatchGAN discriminator with Spectral Normalization [35] for convolutional layers: it stabilizes training process and doesn't allow to win competition in few epochs. Paper **SC-FEGAN** [36] used almost all improvements from the previous two works, but they added for control also color and shape into the mask. Both works use *coarse-to-refine* architecture type for neural networks, i.e., they defined generator as networks with two significantly different subparts: one generates possible corrupted by visual artifact images, and the last one refines generated image to a realistic one. **EdgeConnect** [37] works similarly to the coarse-and-refine network. However, it generates sketch and borders of the object inside **RoI** and then generates the entire object using the guidance of the previous output (like refine stage). **Globally and Locally Consistent Image Completion** [38] work builds local discriminator around **RoI** for better discrimination only the **RoI** part of an image, and generator structure also similar to the coarse-to-refine network.

Image-to-Image Translation with Conditional Adversarial Networks [1] is image translation method, which allows to generate content from defined multi-class mask. Authors used *PatchGAN* discriminator, updated generator with U-Net [39] based architecture and added to adversarial loss L_1 conditional reconstruction loss for generator:

$$D^*,\, G^* \leftarrow \min_{\theta_G} \max_{\theta_D} \Big\{ L_{GAN}(D,\, G) + \alpha L_1(G) \Big\} \tag{2.12}$$

However, their model is not appropriate for high-resolution image generation because it produces blurry images and artifacts on it. So, Pix2PixHD was introduced in the **High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs** [40] work. Authors defined the multi-scale structure of the generator and the discriminator, which operate on different scales and with different levels of details on the image. Discriminators work with few resolutions. Hence they specialized in different features depend on the image quality. Generators work together with discriminators and produce images with different resolutions, which united at the end of the architecture. Authors experimented with *Feature matching loss*: it helps to transfer properties of the real images onto generates ones using information from the intermediate layers of the discriminator:

$$L_{FM}(G) = \sum_{j=1}^{N_L} \mathbb{E}_{x,\, z \sim \mathbb{P}_x,\, \mathbb{P}_z} \big|\big| D_j(x) - D_j(G(z)) \big|\big|_1 \tag{2.13}$$

where N_L – number of layers in the discriminator (summation iterates over outputs of the discriminator's layers). Entire adversarial loss consists of three discriminators and one generator (few generators minimize the same part of the loss, to it can be formalized in the one model):

$$D_1^*,\, D_2^*,\, D_3^*,\, G^* \leftarrow \min_{\theta_G} \max_{\theta_{D_1},\, \theta_{D_2},\, \theta_{D_3}} \Big\{ \sum_{i=1}^{3} L_{GAN}(D_i,\, G) + \alpha L_{FM}(G) \Big\} \tag{2.14}$$

Semantic Image Synthesis with Spatially-Adaptive Normalization [41] work used only semantic information to generate photo-realistic pictures. Authors created **denormalization layer** – **Spa**tially-adaptive **de**normalization (SPADE). It converts the semantic map into a continuous tensor and hence allows the differentiable transformation of the discrete **RoI**-tensor (Figure 2.3). It possible with direct style changing of the normalized row tensor of data with **RoI** as the source of style. For increasing the diversity of the generated samples, authors used sampling inside the architecture with **VAE**-loss 1.6. The architecture with the SPADE block demonstrated excellent visual results in region-selected content generation, so we find this approach interesting for our purposes and uses some key elements of the work in our research.

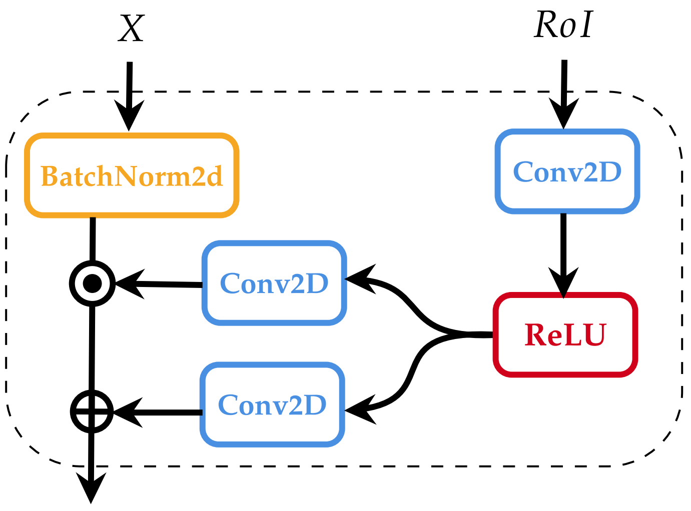


FIGURE 2.3: The schema of SPADE [41] layer. Block processes X input, normalizes it and modulates with scale and bias computed using **RoI** information.

2.5 Conclusion

The research works in the image inpainting, and controllable generation fields have a lot of great solutions. They mostly GAN-based, but have a lack of unsupervised control of the local generation inside defined **RoI**. Typical works on the image completion or inpainting mostly concentrate on the restoration process, ignores the generative properties of the obtained model, and also ignore adding a procedure to control the generation of the main visual aspects of the processed data. There are few works on the controllable generation, but they performed the global generation and ignores spatial dimensions to create new content. We see these facts as the bottleneck to the entire generation process. We hypothesize that, if a generator creates realistic filling of the **RoI**, it can be customized to *control* generation of the content within the region of interest.

We examined the best practices and decided to unite a few ideas in our solution: modified denormalization layers and **RoI**-block for masks sampling. We think that adding style control inside the denormalization layer for mask should allow to localize and control generation inside the defined area, and sampling random masks during training will cover almost all possible variations of filled and non-filled zones. It should force the generator to make good visual inpainting, and discriminator – to concentrate on the **RoI**s. At the same time, we want to tie random variables with local features. We plan to achieve it using mutual information maximization principle between random variable for control and prior random variable, like in [20].

Chapter 3

Proposed method

In this chapter, we describe proposed architecture of generative adversarial network for image inpainting with control within region of interest – **RoI**-GAN.

3.1 Method concept

We define our architecture concept in Figure 3.1. The generator output is conditioned on input image from real distribution $p_d(x)$ and selected **RoI**. During training, a **RoI**-block with non-trainable parameters θ samples the regions of interest. The generator receives the **RoI** into denormalization layers, which accumulates information abouts masks in the architecture and allows control of the generation inside **RoI**. The discriminator gets the true image x and changed image $G(z|x,\ \mathbf{RoI})$. Initial random vector $z \sim p_{z_0,\ c}(z)$ consists of incompressible noise z_0 and distribution $p_c(c)$ for control. They concatenated and fitted into network, and then, using optimization, distributions $p_{G(z_0,\ c)}$ and $p_c(c)$ have became entangled. For definitions simplification, we will call vector $(z_0,\ c)$ as vector z.

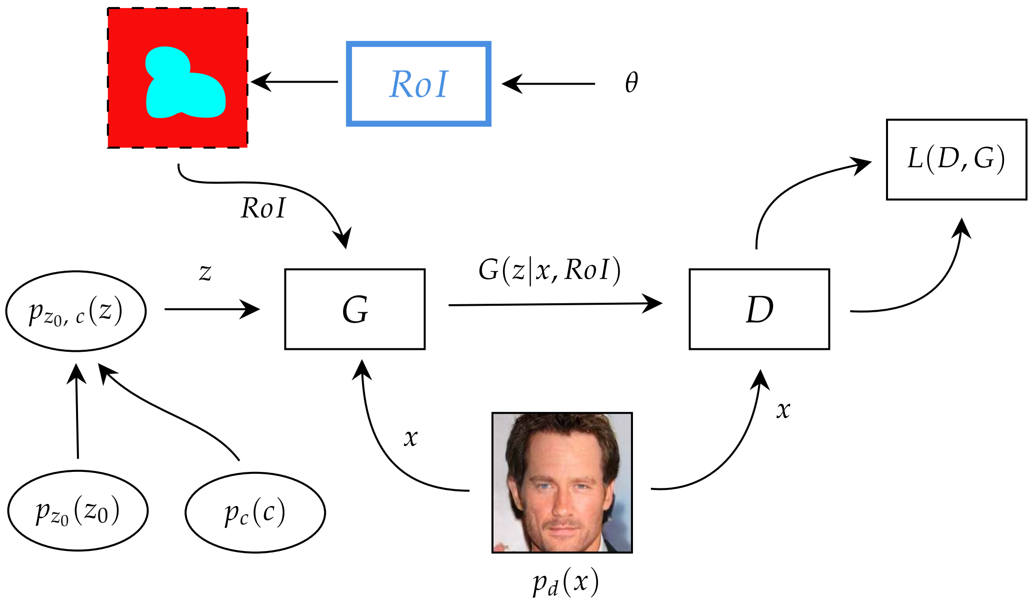


FIGURE 3.1: Own solution concept: the generator receives **RoI** mask, a sample from real data $x \sim p_d(x)$, a sample from random prior $z \sim p_z(z)$ and outputs $G(z|x,\mathbf{RoI})$. The discriminator gets the x and generated $G(z|x,\mathbf{RoI})$ and gives the probabilities that inputs are real, which used in the $L(D,G)$. The **RoI** is generated using parameters θ in the **RoI**-block. Prior distribution consists of incompressible noise $p_{z_0}(z_0)$ and distribution $p_c(c)$, which will be used for control after optimization process.

1. The **RoI**-block is the non-trainable layer. It generates random **RoI**s during training (elliptic, square, or complex areas of the interest). During the inference stage, the **RoI**-block is off and can be used a custom-defined region of interest.

2. Then the denormalization layer inside the generator receives and transforms **RoI**. It transforms the binary mask **RoI** into continuous tensor, modulates the input tensor with scale and bias, obtained from the mask. We designed two options: generative block with style modulation of mask features with a random variable, or without it. We used the first option for image generation inside **RoI** and the second for generation outside **RoI**.

3. The **generator** G is conditional generative model which receives additional real image $x \sim p_d(x)$ and **RoI** before creating synthetic images as prior information, and generates outputs depends on these values. It consists of encoder and decoder part. They were built in U-Net manner. The G consists of blocks with denormalization layers.

4. The **discriminator** D is a standard convolutional neural network, receives the input real/fake image, and its output $\mathbb{P}_{real}(\cdot)$ is proportional to the probability that samples are real. It consists of global and local parts, whose give score for the entire image or its parts correspondingly. Also, D outputs auxiliary network $Q(c|x) \approx \mathbb{P}(c|x)$. The Q network used to optimize $L_I(\cdot,\cdot)$ objective, which helps to entangle distribution for control $p_c(c)$ with prior noise $p_{z_0}(z_0)$

5. After training, we will use G as conditional generative model $x_g \sim G(z|x_r, \textbf{RoI})$ and generate new content inside **RoI** for defined **RoI**. We also will use the $p_c(c)$ distribution to control generation inside **RoI**: changing one coordinate of the random vector with high probability will change one of the major visual elements.

3.2 RoI-block

The **RoI**-block is a non-trainable generative part of the proposed GAN architecture. It samples different masks and used them inside the generator to create conditions in which generation occurs within **RoI**. The purpose of **RoI**-block to force generator creating content following the real distribution of training images. As well, we wanted to add like unsupervised feature extraction from autogenerated masks – during training, generator will learn to fill the **RoI** and control content, while discriminator will learn to discriminate more **RoI**, instead of other zones of AN image. There are three types of **RoI** (visual represenation in Figure 3.2):

1. **Squares RoIs**: using uniform two-dimensional distribution over image size, generate rectangles. We generate the positions of diagonal points within coordinates of the image, and delete part of the image in this rectangle:

$$\mathbb{P}_{\textbf{RoI}} \sim \begin{cases} x_1,\ x_2 \sim U(0,\ I_w) \\ y_1,\ y_2 \sim U(0,\ I_h) \end{cases} \tag{3.1}$$

where $\mathbb{P}_{\textbf{RoI}}$ – rectangles with coordinates of opposite diagonal points $(x_1,\ y_1)$ and $(x_2,\ y_2)$. We dropped cases when square measure of rectangle is zero (it happens when $x_1 = x_2$ or $y_1 = y_2$)

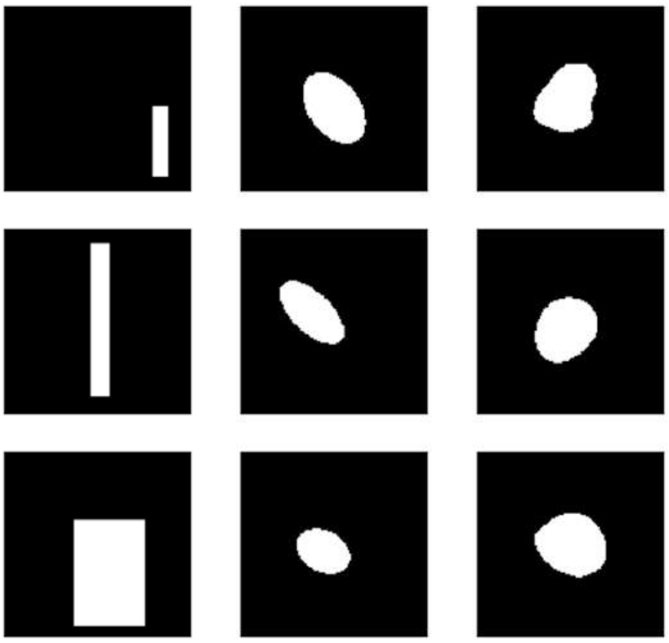


FIGURE 3.2: Different generated **RoI**: (a) Masks with rectangular form (b) Masks with elliptic form, with different orientation and center location (c) Masks, obtained from the level curve of gaussian mixture density function

2. **Elliptic RoIs**: using level curves of two-dimensional gaussian distribution, generate ellipses with different orientations. We generated density functions with different parameters, and then thresholded them with a predefined constant. The construction process begins with mean μ generation of gaussian distribution:

$$\mu \sim N\left(\begin{pmatrix} 0.5 I_w \\ 0.5 I_h \end{pmatrix}, \begin{pmatrix} I_w & 0 \\ 0 & I_h \end{pmatrix} \right) \tag{3.2}$$

Next generate covariance matrix Σ of gaussian distribution:

$$\begin{cases} \rho \sim U(-0.8,\, 0.8) \\ \sigma_x \sim U(\frac{I_w}{5}, \frac{I_w}{3}) \\ \sigma_y \sim U(\frac{I_h}{5}, \frac{I_h}{3}) \end{cases} \implies \Sigma^{-1} = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix} \tag{3.3}$$

Finally, generate probability density function with μ and Σ:

$$f(\vec{x}|\mu,\, \Sigma) = \frac{1}{2\pi\sqrt{|\det\Sigma|}} e^{-\frac{1}{2}(\vec{x}-\mu)^T\Sigma^{-1}(\vec{x}-\mu)} \tag{3.4}$$

The distribution of ellipses will be:

$$\mathbb{P}_{\textbf{RoI}} = \{\vec{x}|\ f(\vec{x}|\mu,\, \Sigma) > \frac{\max f(\vec{x}|\mu,\, \Sigma)}{2}\} \tag{3.5}$$

3. **Mixed RoIs**: using gaussian RoIs procedure, generate the level curves of gaussian mixture. Generate sequence of functions with different μ and Σ, and generate convex weighted sequence of real numbers $w_1,\ w_2\ ...\ w_n$, i.e $\sum_{i=1}^{n} w_i = 1$. The mixture probability density function will be:

$$f_n(\vec{x}) = \sum_{j=1}^{n} w_j f(\vec{x}|\mu_i,\ \Sigma_i) \tag{3.6}$$

The distribution of non-trivial region of interest will be:

$$\mathbb{P}_{\textbf{RoI}} = \{\vec{x}|\ f_n(\vec{x}) > \frac{\max f(\vec{x}|\mu,\ \Sigma)}{2}\} \tag{3.7}$$

RoI-block is defined as block only theoretically, while in implementation, it behaves like a random variable, and we sampled from it during the training procedure.

3.3 Denormalization layer

The main functional part of architecture is block called *denormalization layer* (Figure 3.3). It normalizes input of block and denormalizes it, using scale and bias, computed using information from **RoI**:

$$\begin{cases} z^* = (z_1,\ z_2) = Az + b \\ \textbf{RoI}^* = \text{Activation}(\textbf{Conv2d}(\textbf{RoI}, K_{\text{internal}})) \\ \textbf{RoI}^* = \textbf{RoI}^* \cdot z_1 + z_2 \\ \hline \gamma(\textbf{RoI}^*) = \text{Activation}(\textbf{Conv2d}(\textbf{RoI}^*,\ K_\gamma)) \\ \beta(\textbf{RoI}^*) = \text{Activation}(\textbf{Conv2d}(\textbf{RoI}^*,\ K_\beta)) \\ \hat{x} = \gamma(\textbf{RoI}^*) \cdot \frac{x - \mathbb{E}(x)}{\sqrt{\mathbb{D}(x) + \epsilon}} + \beta(\textbf{RoI}^*) \end{cases} \tag{3.8}$$

where z – random vector, it used as vector for style control, $A,\ b,\ K_{\text{internal}},\ K_\gamma,\ K_\beta$ - trainable parameters of the layer.

The first three equations defined control over style and content (using transformed z^*) inside **RoI**, the three last ones – defined how to incorporate this style in the features x.

The denormalization block inspired a lot by SPADE [41], but we added a block for explicit control with random value. The block convolves input **RoI** (we increased resolution of **RoI** before convolution applying to remain the same dimensions between X, scale, and bias factors). We transformed input random variable Z with affine transform, and interpret new values as weights for channels of convolved mask's features. Then we obtained $\gamma(\textbf{RoI}^*)$ and $\beta(\textbf{RoI}^*)$ via additional convolutions.

Generally speaking, we performed two-stage style modulation of the normalized tensor. First, introducing mask in the network as the style modulation allows to generate new content only inside **RoI**. Second, and the style modulation of **RoI** with a random variable in the intermediate layers – allows to control generation process. As a result, the layer allows us to generate and control a part of the image within **RoI**.

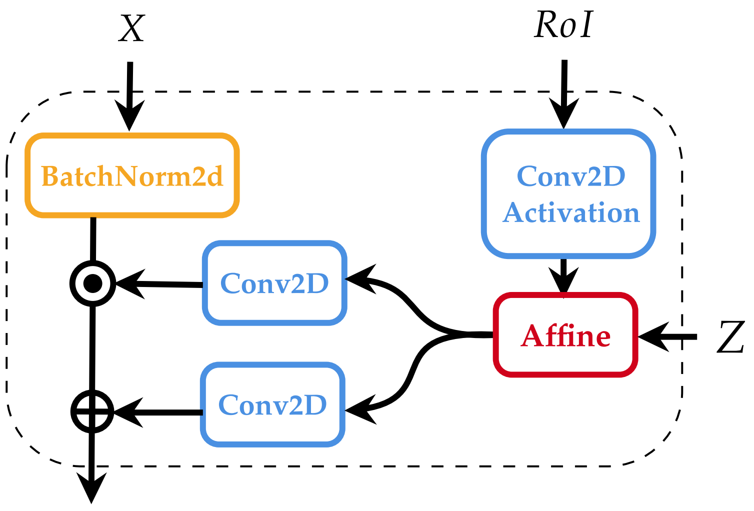


FIGURE 3.3: The denormalization layer. Block processes X input, normalizes it, and modulates normalized data with scale and bias computed with **RoI** information. Random vector Z performs the affine transformation of intermediate features

3.4 Upsample block

The generator mainly consists of blocks with possibility to control generation inside **RoI** or in $1 - \mathbf{RoI}$ (Figure 3.4). The block has two branches: left branch specializes on the generating content inside **RoI**, and for developing generative properties style modulation with random vector Z is used (Figure 3.3); the right branch does not use Z. Hence the denormalization layers of the right branch do not have affine layer and style modulation with random variable – output depends only on the mask $1 - \mathbf{RoI}$ and input tensor X. Both branches have convolutional layer with non-linearity after it. After merging, the block has another denormalization layer with style modulation, followed convolution.

Overall, the generation process exists only in the left branch and at the end of the block, while the right branch restores $1 - \mathbf{RoI}$ part of the image. All convolutions in the block have optimal padding to prevent image size decreasing and only upsample layer twice input resolution. We used the nearest mode for an upsampling algorithm. Leaky ReLU function was used with negative slope 0.2.

3.5 Generator

The generator is U-Net based (Figure 3.5): transforms images at different scales, and concatenate this information in the upscaling process (encoder's intermediate output saves and then used as additional decoder's inputs).

The encoder part used gated convolution layers: they help to transform and integrate mask information along with images. For simplification of the diagram, we do not plot activations (LeakyReLU with negative slope 0.2) and normalization (InstanceNormalization) layers in the encoder part. However, we used it in all gated convolution (gated convolution $\rightarrow$ normalization $\rightarrow$ activation). All convolutions in the encoder reduced input resolution by half. The last layers of encoder generated mean and variance vectors – they must be "global style" vectors.

Prior random variable Z goes through dense layers with activations and then modulates the global style of an image using extracted mean and variances from the image. Then, vector $\sigma Z + \mu$ reshaped using pixel shuffle [42] layer – it helps to

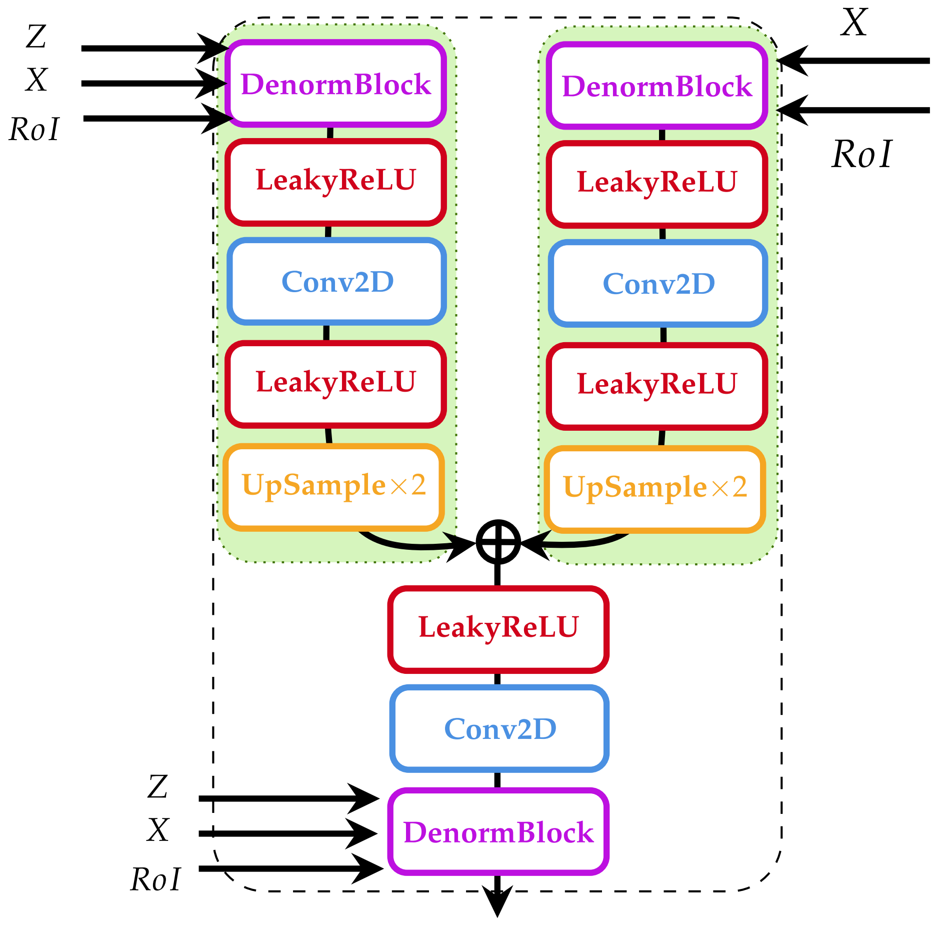


FIGURE 3.4: The upsample block: left branch specializes on the generating new content inside **RoI** (used information from random variable Z), while right branch restores content outside **RoI** (didn't use Z). Branches merged at the end, and final denormalization layer applyied to united branch.

increase the resolution of features and distribute channels information over spatial dimension in an optimal way. Then, it used in the generation process inside **RoI** and restoration process inside $1-$ **RoI**.

All upsample blocks in the generator double the spatial size of the input tensor. The last layer also has a convolutional layer that performs the mapping from input channels to an RGB-type image. We also perform InstanceNormalization after this convolution, but does not plot it, and then limited output with $\text{Tanh}(\cdot)$.

3.6 Discriminator

The discriminator is based on the DCGAN [43] architecture type, and entirely used convolutional layers (expect the final one). It has three parts: intermidiate part D_{main}, *local* discriminator D_{local} and *global* discriminator D_{global}. The local discriminator outputs scores for different parts of images that correspond to accumulated receptive field from convolution parts. The global discriminator scores entire image as fake or real. Then, their scores concatenated along feature dimension and released as multi-dimensional output. Other part of network, $Q(c|x)$, is part of discriminator by parameters, but optimized together with generator's parameters. The Q network

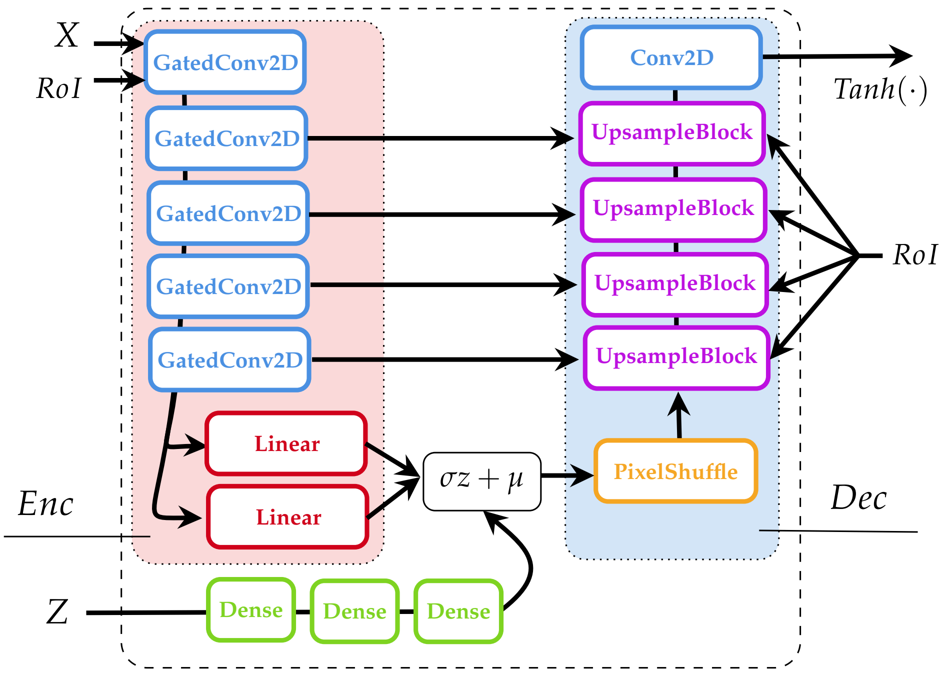


FIGURE 3.5: The generator network: image X and mask **RoI** goes through encoder part, consists of gated convolutions. Then encoder modulates mean μ and variance σ – the encoded global style of the image; random vector Z, sampled from prior distribution, pre-processed in the fully-connected layers with activations and then modulates with mean and variance from encoder part; decoder uses $\sigma Z + \mu$ and **RoI** to generate new content inside **RoI** and restore image in the $1 -$ **RoI**. **Attention!** Encoder part used instance normalization and LeakyReLU activations between blocks, Dense layers have LeakyReLU activations between blocks and decoder part have LeakyReLU activations between blocks. Input to Tanh$(\cdot)$ also goes through instance normalization layer. They aren't demonstrated on the scheme for simplification of the diagram.

approximates distribution $\mathbb{P}(c|x)$, which used in the lower bound of mutual information – $L_I(\cdot,\ \cdot)$ objective. For simplification, we interpret, as in original paper [20], distribution $\mathbb{P}(c|x) = N(\mu,\ \text{diag}(\sigma))$.

The only two first convolutions in the D_{main} decreases the size of the input in half ($64 \to 32 \to 16$).

3.7 Adversarial loss

We used Hinge adversarial loss, because it showed efficiency in the image inpainting domain [34] and in our experiments as well, with few additional penalties for discriminator and generator networks. Our goal is to enhance Hinge loss with additional properties, which help with training stabilization, help to generate proper content inside **RoI** – transfer style from original image to generated one (for good quality of the resulted image), and also to increase model generative abilities of the distribution $G(z|x,\ \textbf{RoI})$ – conditional distribution should have high entropy inside **RoI**.

Using the generated samples x_g and real samples x_r we constructed *completed* samples:

$$x_c = \textbf{RoI} \cdot x_g + (1 - \textbf{RoI}) \cdot x_r \tag{3.9}$$

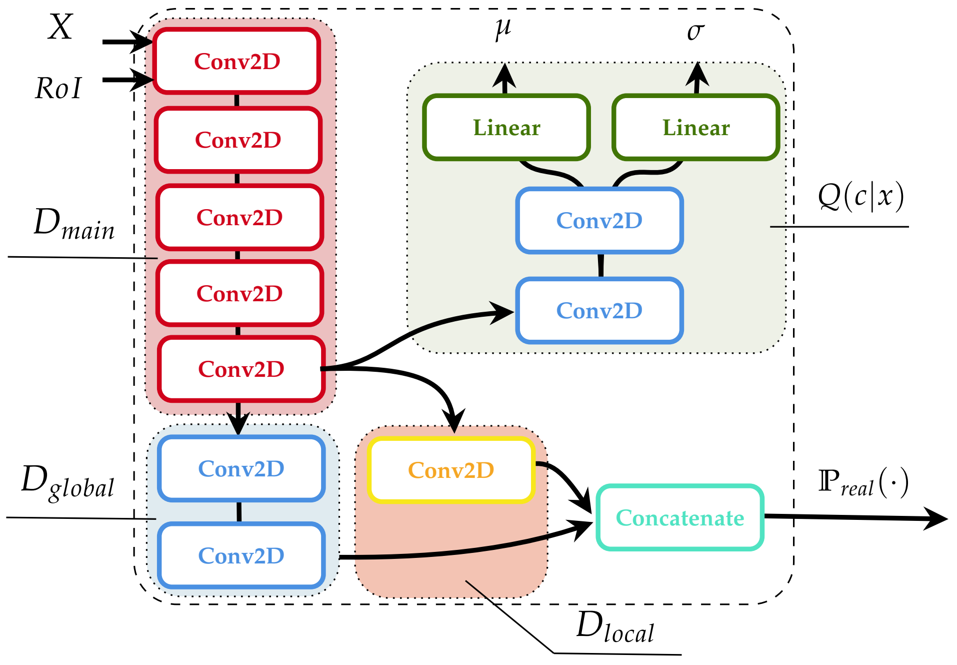


FIGURE 3.6: The discriminator network: D_{main} part computes intermidiate features, while D_{global} gives one score if image real or fake, and D_{local} gives vector of scores (one score per image part). They concatenated in the one vector scores $\mathbb{P}_{real}(\cdot)$; additional part of discriminator $Q(c|x)$ depends on D_{main} outputs and modulates a posterior distribution $\mathbb{P}(c|x)$. It used for optimization of mutual maximization information between variables for control $p_c(c)$ and distribution of generator $p_{G(x,\,c)}(\cdot)$; **Attention!** Instance normalizations and LeakyReLU activations used between all convolution layers (except the input and output), and we don't plot it for simplification of the diagrams.

This term defines edited image inside **RoI**.

1. **Gradient penalty**. During experiments, we obtained stable training only with this term:

$$\begin{cases} L_{\nabla} = \mathbb{E}_t(||(1-\mathbf{RoI})\cdot\nabla_t D(t)||_2 - 1)^2 \xrightarrow[\theta_D]{} \min \\ t = \alpha\cdot x_r + (1-\alpha)x_c,\ \alpha \in U(0,\,1) \end{cases} \tag{3.10}$$

   We limited norm of discriminator's gradient inside **RoI**. Motivation: our model is conditioned on the x_r and **RoI**; in this way, only part of the gradient that is inside **RoI** is important. Also it helps to stabilize discriminator's Wasserstein loss part.

2. **Feature Matching**. We forces generator to create artificial content with similar statistics to the real samples:

$$L_{\text{FM}} = \sum_{i=1}^{M}\mathbb{E}_{x_r,\,x_c}||D_{l_i}(x_r) - D_{l_i}(x_c)||_1 \xrightarrow[\theta_G]{} \min \tag{3.11}$$

   where $l_1,\ \ldots l_M$ – layers with LeakyReLU activations inside the main part of discriminato. It helps to prevent mode collapse and makes distribution of artificial images more similar to the real data.

3. **Perceptual VGG-loss**. We used pretrained $VGG16$ [44] network for content style transfer from original to generated image and completed image:

$$\begin{cases} L_{\text{per}_1} = \sum_{i=1}^{M} \mathbb{E}_{x_r,\, x_c} \|V_{l_i}(x_r) - V_{l_i}(x_g)\|_1 \xrightarrow[\theta_G]{} \min \\ L_{\text{per}_2} = \sum_{i=1}^{M} \mathbb{E}_{x_r,\, x_c} \|V_{l_i}(x_r) - V_{l_i}(x_c)\|_1 \xrightarrow[\theta_G]{} \min \\ L_{\text{per}} = L_{\text{per}_1} + L_{\text{per}_2} \end{cases} \tag{3.12}$$

where $l_1, \dots, l_M$ – layers with ReLU activations inside pretrained network. The first one helps to restore the image in the $1 -$ **RoI** part and give information about differences (in the sense of features) between two disjoint areas, the second one – updates weights that are relevant to the **RoI**.

4. **Zero-concentrator loss**. We used ideas from Progressive GAN [25] and force discriminator's outputs on the real samples to be close to the zero:

$$L_0 = \mathbb{E}_{x_r \sim \mathbb{P}_r,\, \mathbf{RoI}} \frac{1}{N_{\text{out}}} \sum_{i=1}^{N_{\text{out}}} D^i(x_r, \mathbf{RoI})^2 \xrightarrow[\theta_D]{} \min \tag{3.13}$$

It should boost discriminator to learn realistic statistics from real samples quickly and hence give sufficient feedback to the generator.

5. **Info loss**. We used mutual information maximization part for creating a dependency between distribution for control and implicit distribution of the generator:

$$L_I = \mathbb{E}_{c \sim p_c(c),\, x_g \sim G(z_0,\, c|x_g,\, \mathbf{RoI})} \log Q(c|x_g) + \mathbb{H}(c) \xrightarrow[\theta_Q]{} \max \tag{3.14}$$

Network $Q(c|x_g)$ modulates a posterior distribution $\mathbb{P}(c|x_g)$, where c – random variable for control, dimension of the variable is L_c. It consists of continuous variable c_0 and discrete variable c_1. For simplification, we limit class of a posteriory distribution to multivariate gaussian with independent components, and network Q predicts μ and σ of gaussian density for continious variable. Also, it predicts log-probabilities for discrete variable $\log p_j = \log Q({c_1}^j|x_g)$ with dimension equal to L_d. Using this settings, loss function can be simplified and decomposed:

$$L_{\text{con}} = -\frac{1}{2}\mathbb{E}_{c_0 \sim p_{c_0}(c_0),\, x_g \sim G(z_0,\, c_0|x_g,\, \mathbf{RoI})} \sum_{i=1}^{L_c} \left(\frac{c_0 - \mu_i}{{\sigma_i}^2} - \log 2\pi\sigma_i \right) \xrightarrow[\theta_Q]{} \max \tag{3.15}$$

$$L_{\text{dis}} = \frac{1}{L_d} \sum_{j=1}^{L_d} \mathbb{E}_{x_g \sim G(z_0,\, {c_1}^j|x_g,\, \mathbf{RoI})} \log p_j \xrightarrow[\theta_Q]{} \max \tag{3.16}$$

Total mutual information maximization loss:

$$L_I = L_{\text{con}} + L_{\text{dis}} \xrightarrow[\theta_Q]{} \max \tag{3.17}$$

6. **VAE-loss**. We used VAE-loss 1.6 to manipulate with global style within **RoI**. We interpreted outputs of encoder as scaling factors μ and σ for input random

value (reparametrization trick). This loss also helps to increase diversity of the generated samples.

7. **Total variation loss**. We used 2.10 to enhance continuity inside region of interest.

Total generator's loss:

$$\begin{cases} L_{\text{style}}(\theta_G) = 0.5L_{\text{TV}}(\theta_G) + 5L_{\text{per}}(\theta_G) + L_{\text{FM}}(\theta_G) \\ L_{\text{add}}(\theta_G,\, \theta_Q) = -0.5L_{\text{VAE}}(\theta_G) - 0.5L_I(\theta_Q) \\ L_G(\theta_G,\, \theta_Q) = L_{\text{Hinge}}(\theta_G) + L_{\text{add}}(\theta_G, \theta_Q) + L_{\text{style}}(\theta_G) \underset{\theta_G,\, \theta_Q}{\longrightarrow} \min \end{cases} \tag{3.18}$$

Total discriminator's loss:

$$L_D(\theta_D) = L_{\text{Hinge}}(\theta_D) + 5L_{\nabla}(\theta_D) + 10^{-4}L_0(\theta_D) \underset{\theta_D}{\longrightarrow} \min \tag{3.19}$$

We used style transfer loss, like in the [32], but we didn't achieved improvements and refused to use it. We also used L_1-reconstruction loss, however, in the last experiments we disabled it and had more diverse results. All constants in the loss functions we took from the experiments via hyperparameter search: we run experiments and check if we have good generated quality. Concrete hyperparameters achieved the best visual result among all experiments.

3.8 Training details

For training stabilization, we used recommendations from [43] and [45]: feature matching loss, different learning rates for discriminator and generator – $L_D = 10^{-4}$ and $L_G = 3 \cdot 10^{-4}$ with Adam [46] optimizer, minimum number of dense layers, Leaky ReLU in the discriminator, etc. We also used the equalized learning rate, introduced in Progressive GAN [25]. It helps to create balanced competition between generator and discriminator, and it works by multiplication weights on the He et al initialization [47] constant before forward pass:

$$K_w \text{ with shape } [C_{out},\, C_{in},\, K_h,\, K_w] \implies \hat{K}_w = K_w \cdot \sqrt{\frac{2}{C_{in} \cdot K_h \cdot K_w}} \tag{3.20}$$

Using this technique, we defined initial weights for our model as from standard normal distribution $N(0,\, 1)$. We set $z_0 \sim N(0,\, 1)$ and sampled continuous part of vector for control from $c \sim U(-1,\, 1)$. The discrete part sampled from uniform discrete (categorical) distribution.

The discriminator has 6.04M parameters, and the generator has 6.52M parameters. We trained models together typically 100-400 epochs (depends on the dataset), and we do not provide loss plots or training details, because we provided enormous amounts of experiments for increasing quality of our model.

Implementation of our model, using PyTorch framework, locates in the GitHub[1].

[1]GitHub of the thesis: https://github.com/vaden4d/roi-gan

Chapter 4

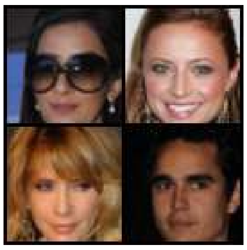
Experiments

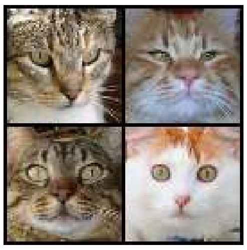
In this Chapter, we describe datasets and demonstrate conducted experiments, give visual and numerical results of our method.

4.1 Datasets

In experiments we used CelebA [48], Cats [49] and Cars [50, 51] datasets. The datasets details locate in Table 4.1 and Figure 4.2.

The first dataset is the celebrities' faces source. The faces are preprocessed and aligned in the one position – in the middle of the images. There are different types of faces (frontal, profile, faces with glasses, or other). We used dataset in the pictures 64×64 resolution. The second is dataset with 64×64 cats faces, which also are preprocessed and aligned in the middle of the image.


(a) CelebA


(b) Cats

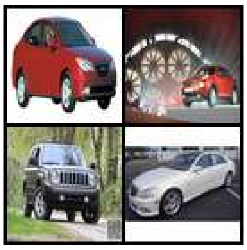

(c) Cars

FIGURE 4.1

FIGURE 4.2: The fours samples from each used image sources: (a) CelebA: dataset with human faces; (b) Cats: dataset with cats faces; (c) Cars: dataset with cars from different 3D-view points

Name	Train images N	Test images N	Total size	Resolution
CelebA 64	162 770	39 829	$\approx$ 2.0GB	64×64
Cats	15 748	—	$\approx$ 0.5GB	64×64
Cars	8 145	8 042	$\approx$ 2.1GB	64×64

TABLE 4.1: The characteristics of used datasets. We used CelebA, Cats and Cars datasets with resolutions 64×64. The resolution columns are not real resolution of the data, but what we used in experiments.

Method	FID (elliptic RoIs)	FID (rectangular RoIs)	FID (mixed RoIs)
Our	**0.323**	**0.074**	**0.106**
FMM [15]	0.539	0.111	0.197
GLCIC [38]	0.485	0.638	0.618

TABLE 4.2: The Fréchet Inception Distance (FID) [52] between generated samples from three models (rows) with different **RoI**s (columns) and test CelebA part. We used 1000 randomly sampled images from generated and real distributions to estimate FID in all cases.

The third one is Cars dataset with different vehicle orientations, used mostly in 3D-object manipulation and representations tasks. We used this dataset to study whether our model can control 3D-manipulation within **RoI**.

We tested our model mainly on the CelebA dataset, because it is the optimal dataset among size, the number of samples, images quality, and visually understandable data. We thought that control over human faces would be more comfortable than control over cars or cats, because we, as humans, can easily understand and perform an additional visual evaluation during training.

Other datasets – Cats and Cars – have a small amount of data, so we conducted experiments only to check generation properties. The results on the Cats and Cars should confirm or deny that our model can learn to fill **RoI**s: we thought our model could manipulate with facial features of cats face (e.g., type of wool) and 3D-view of the objects.

4.2 Models comparison

We compared our model with pretrained GLCIC [38] generative adversarial model on the CelebA using open-source implementation [53] on the PyTorch and with non-trainable Fast Marching Method (FMM) [15]. We compared models only by image inpainting quality task, because only our model allows to control generation inside **RoI**. We trained our model with elliptic **RoI**s, and validated the quality of the image inpainting on the other region types. We compared generation results on the standard test split of the CelebA dataset.

Both GANs used different loss functions and different image resolutions, so we compared Fréchet Inception Distance (FID) [52] between generated and real test dataset, using open implementation [54]. The FID measures the mean distance between the features of real and generated images. It computes means and covariances for generated and real distributions, and computes distance using the formula:

$$\mathbf{FID}(X,\ Y) = ||\mu_X - \mu_Y||^2 + \mathbf{Trace}(\Sigma_X + \Sigma_Y - 2(\Sigma_X\Sigma_Y)^{\frac{1}{2}}) \tag{4.1}$$

where X, Y – two datasets, and $\mu_X,\ \mu_Y, \Sigma_X,\ \Sigma_Y$ – their mean vectors and covariances matrices of taken features.

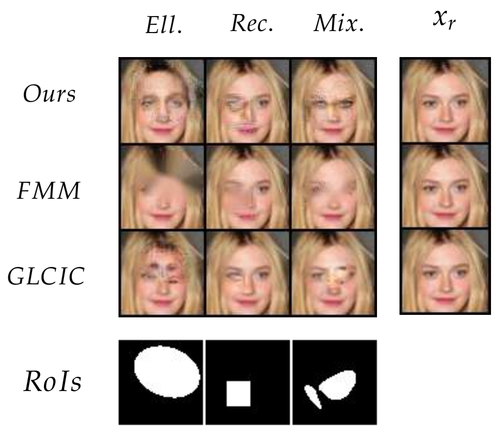
Features are obtained from pretrained model, e.g., a popular choice is Inception-v3 [55] network. The lower value of the metric talks that generated images closer to the real distribution.

Intuitively, features inside the pretrained model absorbate many properties of real data, so it can measure differences between generated and real images. Furthermore, it valid for numerical evaluation of generative models regardless of models used and loss functions.

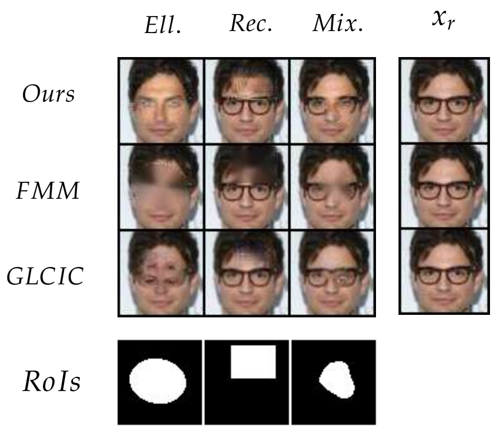
We computed FID on the reshaped to 64×64 generated and real images. The numerical results of the evaluation located in Table 4.2. The lowest results are in our model – hence, by this metric, distribution of generated images is more similar to the distribution of real data. Surprisingly, the results obtained by GLCIC model have higher results than the classic FMM algorithm.

The visual results of the evaluation located in Figure 4.3. We compare three models visually. Our results seem to be more diverse, in the sense of different faces per the same location of the mask. However, we can see many artifacts at the edges of the masks – GLCIC and FMM models behave in this sense more naturally and produced less noise.

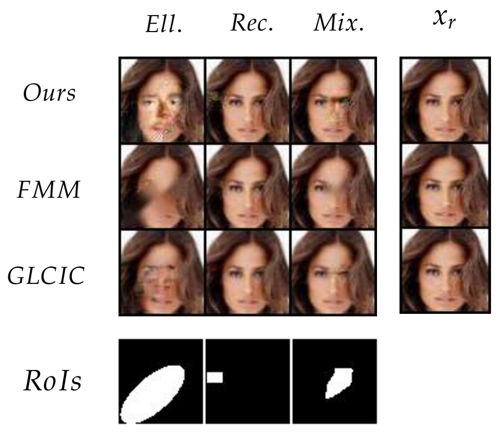
The samples from Cars and Cars are demonstrated in Figure 4.4. As we obtained in the experiments, the model more robust on the CelebA, rather than other datasets. Our model produces visual inconsistency at the borders of the selected zone in some cases (e.g, generated cats faces). This is due to the fact that human faces are simpler in terms of texture, so for Cats or Cars we need to use other hyperparameters for part of the loss function, which can take this into account.



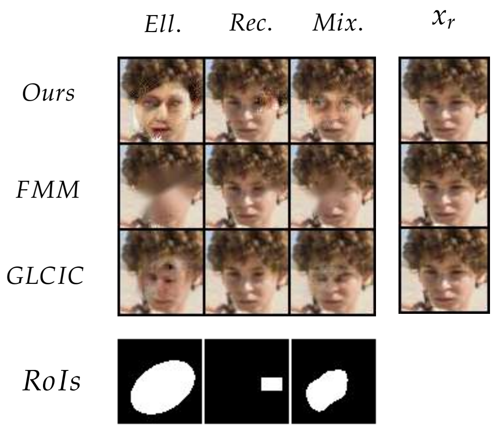
(a) Example one



(b) Example two



(c) Example three



(d) Example four

FIGURE 4.3: Visual comparing of the three models: ours, FMM [15] and GLCIC [38]. (a)-(d) Examples of generation for different **RoI**s types and models. Our model demonstrates more robust generative properties and creates almost realistic content. However, it struggles with visual artifacts. Other models are not generated proper content inside **RoI**: FMM blurred it, and GLCIC creates non-appropriate filling.

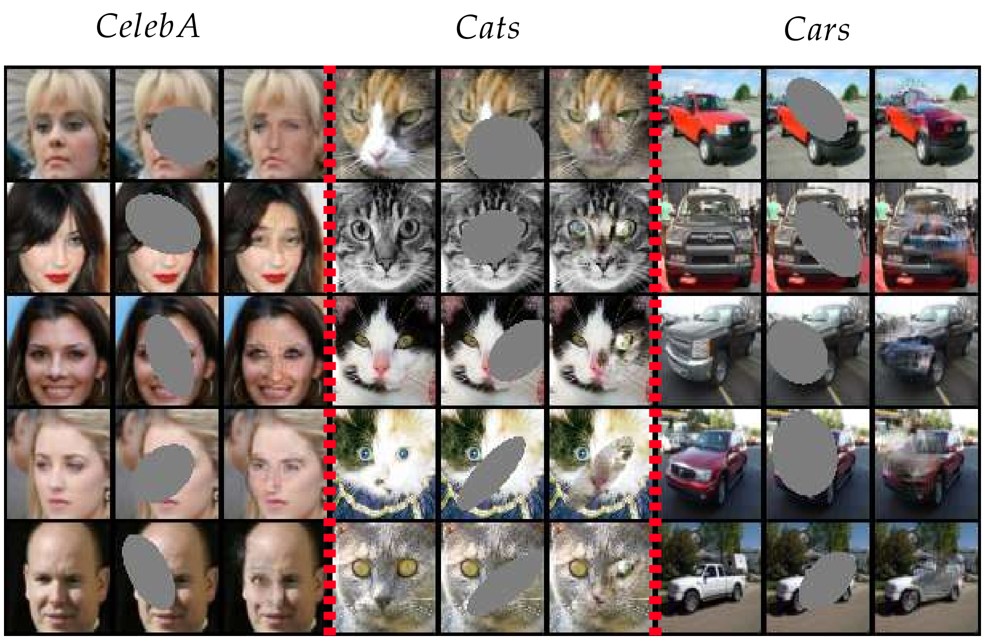


FIGURE 4.4: The examples of generation, using our model on the different datasets (but trained separately on each datasets). For each dataset we visualized three columns: the left images are taken from real data; middle images demonstrate which parts of the image must be filled; the right pictures are outputs of the generator w.r.t correspond **RoI**s and real images. It's easy to notice that the top results model demonstrates on the CelebA dataset.

4.3 Controllability inspection

During the inference stage, we used part $c \sim p_c(c)$ of the random noise to control the generation process. From this moment, we are interpreting this vector not as a random variable, but as a set of sliders: we can vary each coordinate from -1 to 1, and in an unsupervised way change some main aspect of the image inside . After training, mutual information between $p_c(c)$ and $p_{G(z_0,c)}$ is high, so changing one coordinate of the c will change something real on the image. This process is shown in the Figure 4.5. Using a few coordinates of the control vector, we can manipulate with main facial features: change nose form, the form of eyes, create a mustache, manipulate with shadows, or other aspects.

Important to say that this is unsupervised control, i.e., we do not know in advance what slider corresponds to eyes size, and we should find it via experiment. We must investigate by experiment what each coordinate is responsible for. Nevertheless, even now, the generation process has become more clear: changing the control vector changes the outputs more intuitive than a fully random generation. Moreover, using vector for control narrows the scope of the random variable, so we can cover most of the possible generation by using a small part of the random input (vector for control is part of the random noise).

However, the same values of control vector lead to almost the same output regardless of the face we used: changes the only style of the inserted mask, depending on the face style, color, face type. This proves the connection between vector for control, limitations w.r.t conditional arguments – **RoI** and image, and generated distribution.

We also can see that few variables for control edited a few key elements of the faces together, like the size of the nose and size of the eyes. We attribute this to our

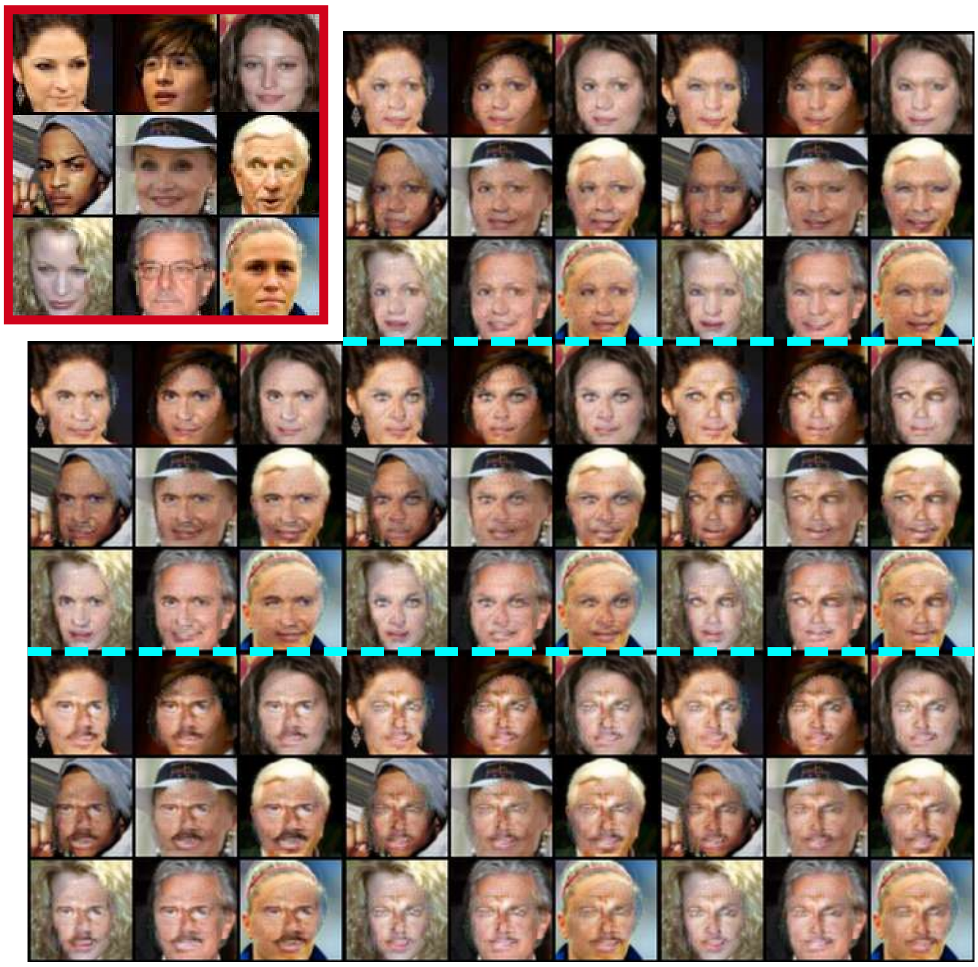


FIGURE 4.5: Controllable generation example with test CelebA faces. Upper left image with red border – real samples. We used $c \sim p_c(c)$ vector as set of sliders, and fitted this value together with incompressible noise part $z_0 \sim p_{z_0}(z_0)$ to create new samples. Changing vector c per coordinate gives desired control properties: the upper images demonstrated changing of the nose form and size eyes; the middle images correspond to the direction of a person's gaze; the down images demonstrated changing shadows and the presence of a mustache. However, these pictures show the cumulative effect of controllability (when we gradually add new effects to each other using per-coordinate control).

limitation about a posterior distribution $\mathbb{P}(c|x)$ to diagonal multivariate distribution.

The quality of control for human faces are much diverse than for cats and cars. We can control local generation, but at the same time this is not so obvious as in the case of human faces. Perhaps this is due to the fact that CelebA is the biggest dataset among chosen, so model highlighted more spatial information in order to create realistic outputs. And for smaller dataset this effect didn't become so significant.

We added controllability results on Cats and Cars datasets to the appendix in Figure A.1 and Figure A.3.

We additionally demonstrated per slider controllability in Figure A.2 on the one sample from CelebA, and also checked properties of our model at the border of the image (like hair zone for faces images), and give example (for CelebA) in Figure A.4. At the borders of the image our model shows lower quality than at the center, but we still can manipulate with content inside **RoI**.

Chapter 5

Conclusions

5.1 Conclusion

In this thesis work, we proposed, implemented, and validated the approach for controllable image generation inside the region of interest and with minimal influence over the rest of the image, using the generative adversarial network.

In Chapter 1, we defined prior definitions about the domain and objectives of our work, deep learning basics, and theory introduction in the GANs. We defined goals – create an image completion tool allows to control generation inside defined zone in the image.

In Chapter 2, we overviewed research works in the two areas – controllable generation and image inpainting. We discovered the absence or lack of popularity of methods that can combine two approaches.

We united image inpainting with controllable generation tasks, using existed top pipelines and our ideas, and created a model that allows localizing generation inside specified **RoI**. We described in detail our architecture in Chapter 3.

We gave an evaluation of our model on the three datasets – Celeba, Cars, and Cats in Chapter 4. We described the controllability properties of the resulted model depends on data type and describe properties of generated distribution.

5.2 Limitations and further work

During experiments and model validation, we observed few limitations:

1. **The appearance of visual artifacts**. Generated images produce artifacts near the border of the **RoI**. We tried to overcome this problem with content style transfer, as recommended in the literature [32], but it did not help.

2. **Inefficiency of discrete variables for control**. On the inference stage, we had found out discrete variables almost didn't change the output of the generation and hence didn't allow to control generation process. Currently, we don't know why this happened, but [20] used it for significantly changing the output. We think that information from these variables is lost somewhere in the middle of the generator. However, continuous variables took on the controllability abilities, so we don't loose in the quality.

3. **Size of the architecture**. We experimented a lot with one big architecture for different datasets. But we want to squeeze architecture and create for in mechanism like in MobileNet [56] family: control the width, depth, and a number of channels, and find the optimal size of the network depend on the dataset (hyperparameter search). It wasn't our goal of the research, so we did not focus on this task.

4. **3D-inconsistency**. We observed that in complex cases, like the different positions of cars or faces in profile, our model doesn't generate well. It creates content without 3D-understanding of the data, e.g., inserts frontal face in the profile position.

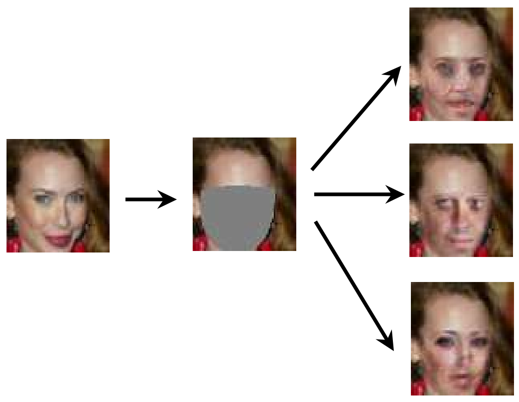


FIGURE 5.1: The process of deepfake anonymization: generate per frame new face and maximaly save facial expressions.

We would also like to test our model on the Celeba-HQ (1024×1024) dataset [57]. During writing the thesis, we didn't train our model on this dataset due to hardware and time limitations – state-of-the-art GAN [25] was trained on Celeba-HQ more than 10 days.

5.3 Possible applications

We see few possible applications of our work:

1. **Clever image edition tools**. The main purpose of the created model is smart image editing within the selected area. The user draws the desired area, and using predefined sliders changes the output inside the selected region.

2. **Deepfake anonymization and detection**. The goal is to retain the person anonymous in the photo or video, but at the same time, keep his emotions and facial expressions. Thus, the model creates a new, artificial face with the same facial expressions instead of the original. As our model cam generate diverse faces in the **RoI** of the face, we think we can perform additional procedures to tie the control vector with the facial features of the concrete person. Using the zone inside facial keypoints, our model could change the face per frame and create new content only inside the desired part of the face (Figure 5.1). Our example isn't follow defined setting (isn't prevent smile), but we think it has potential and should take our idea as proof of concept.

   Also, we see the possible application in the deep fakes detection: if we can use the generative model to create content, we can use the discriminator for frame-by-frame scoring – whether a face is real or not.

Appendix A

Additional visualizations

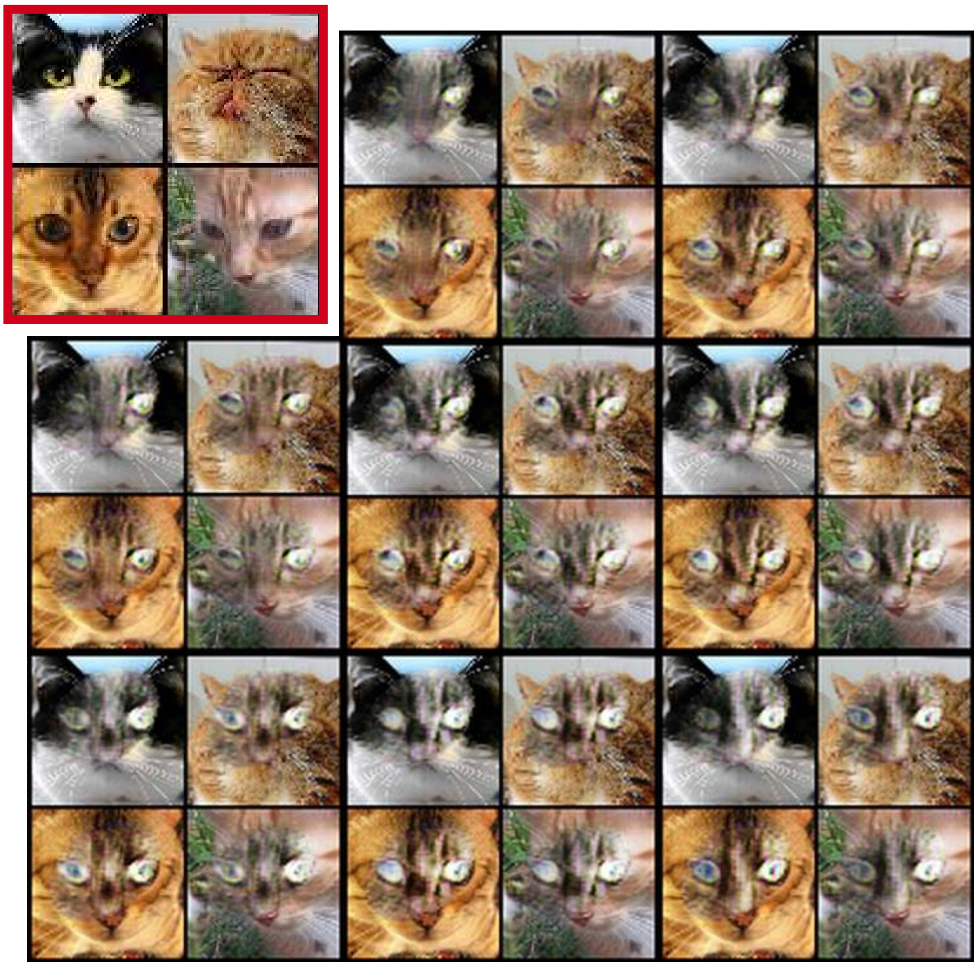


FIGURE A.1: Controllable generation example with cats dataset. Upper left image with red border – real samples. We used $c \sim p_c(c)$ vector as set of sliders, and fitted this value together with incompressible noise part $z_0 \sim p_{z_0}(z_0)$ to create new samples. Changing vector c per coordinate gives control properties: the controllability properties have shown themselves in the control over, e.g., type of wool or glare of the eyes. But, the control turned out not so explicit as in the case of CelebA faces.

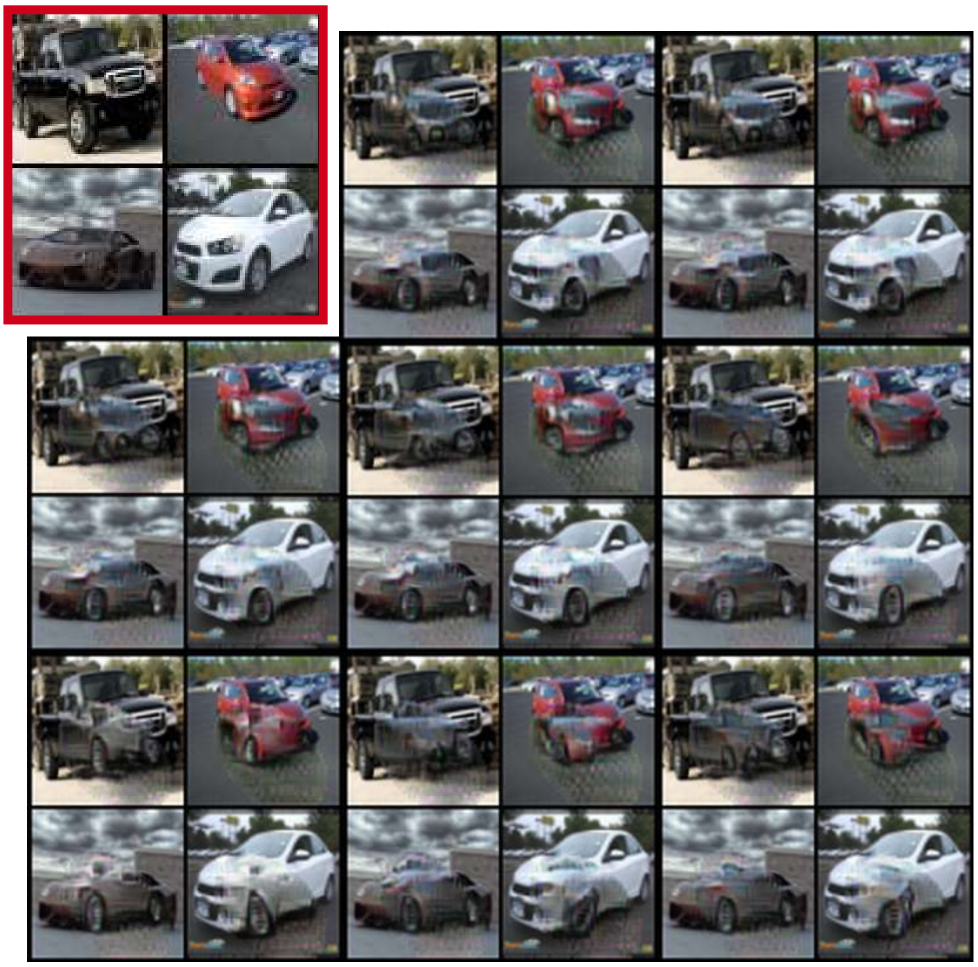


FIGURE A.2: Controllable generation example with test cars. Upper left image with red border – real samples. We used $c \sim p_c(c)$ vector as set of sliders, and fitted this value together with incompressible noise part $z_0 \sim p_{z_0}(z_0)$ to create new samples. Changing vector c per coordinate gives control properties: the controllability properties have not shown themselves as clearly as on other data – but we were able to bring the generation to the most realistic look among possible, using control (upper right images).

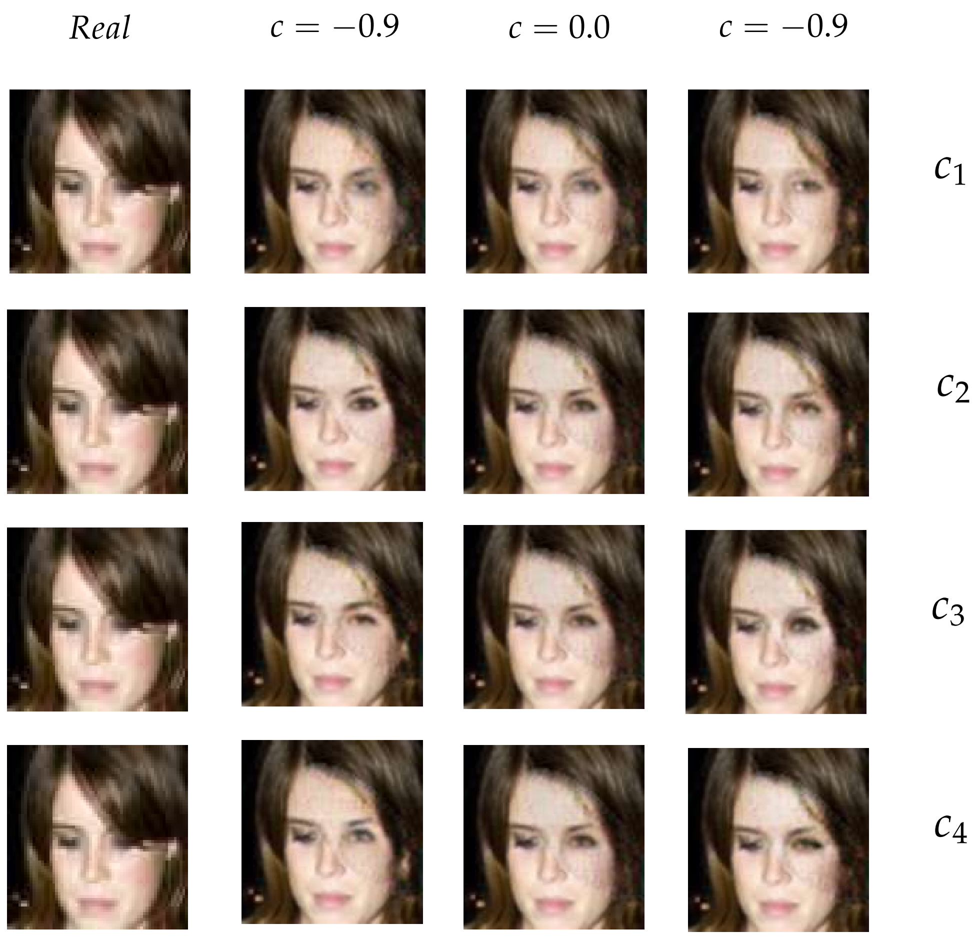


FIGURE A.3: Per slider content changing: we changed the slider's values while other sliders were fixed in the default ($c = 0.0$) position. We took four continuous sliders randomly and demonstrate behaviour of controllability. Our method removed hair within the **RoI** and replaced it with other content. Slider c_1 corresponds to the direction of view, slider c_2 – makeup and ear appearance, c_3 and c_4 – eyes shape.

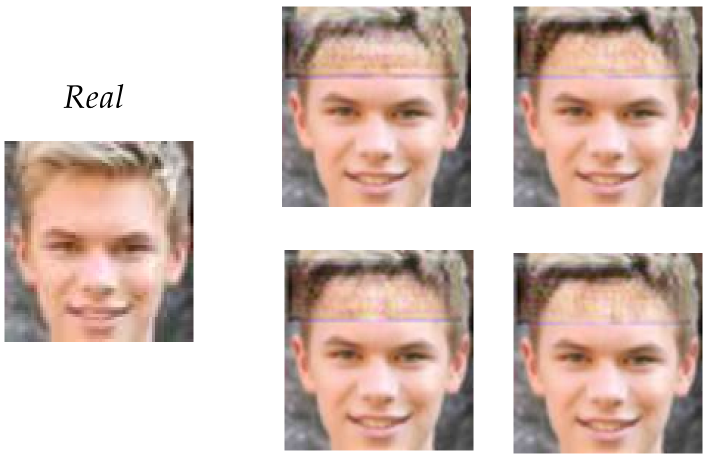


FIGURE A.4: An example of control in the hair area. Our model can manipulate with shadows and shade of hair, but at the same time, controllability is worse than in the area of the face and generates artifacts at the border of the **RoI**.

Bibliography

[1] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *arXiv e-prints*, arXiv:1611.07004 (2016), arXiv:1611.07004. arXiv: 1611.07004 [cs.CV].

[2] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation using Cycle Consistent Adversarial Networks”. In: *arXiv e-prints*, arXiv:1703.10593 (2017). arXiv: 1703.10593 [cs.CV].

[3] Tero Karras, Samuli Laine, and Timo Aila. “A Style-Based Generator Architecture for Generative Adversarial Networks”. In: *arXiv e-prints*, arXiv:1812.04948 (2018), arXiv:1812.04948. arXiv: 1812.04948 [cs.NE].

[4] Thanh Thi Nguyen et al. “Deep Learning for Deepfakes Creation and Detection”. In: *arXiv e-prints*, arXiv:1909.11573 (2019), arXiv:1909.11573. arXiv: 1909.11573 [cs.CV].

[5] Danilo Jimenez Rezende and Shakir Mohamed. “Variational Inference with Normalizing Flows”. In: *arXiv e-prints*, arXiv:1505.05770 (2015). arXiv: 1505.05770 [stat.ML].

[6] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals, and Systems* 2.4 (1989), 303–314. DOI: 10.1007/bf02551274. URL: http://dx.doi.org/10.1007/BF02551274.

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

[8] Carl Doersch. “Tutorial on Variational Autoencoders”. In: *arXiv e-prints* (2016). arXiv: 1606.05908 [stat.ML].

[9] Ian J. Goodfellow et al. “Generative Adversarial Networks”. In: *arXiv e-prints*, arXiv:1406.2661 (2014), arXiv:1406.2661. arXiv: 1406.2661 [stat.ML].

[10] Xudong Mao et al. “Least Squares Generative Adversarial Networks”. In: *arXiv e-prints*, arXiv:1611.04076 (2016), arXiv:1611.04076. arXiv: 1611.04076 [cs.CV].

[11] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein GAN”. In: *arXiv e-prints*, arXiv:1701.07875 (2017), arXiv:1701.07875. arXiv: 1701.07875 [stat.ML].

[12] G. P. Akilov L. V. Kantorovich. *Functional analysis - 3nd Edition*. Publishing house “Nauka”, USSR, 1984, pp. 294–311.

[13] Ishaan Gulrajani et al. “Improved Training of Wasserstein GANs”. In: *arXiv e-prints*, arXiv:1704.00028 (2017), arXiv:1704.00028. arXiv: 1704.00028 [cs.LG].

[14] Han Zhang et al. “Self-Attention Generative Adversarial Networks”. In: *arXiv e-prints*, arXiv:1805.08318 (2018). arXiv: 1805.08318 [stat.ML].

[15] Alexandru Telea. “An Image Inpainting Technique Based on the Fast Marching Method”. In: *J. Graphics, GPU, Game Tools* 9 (2004), pp. 23–34.

[16] Connelly Barnes et al. “PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing”. In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 28.3 (Aug. 2009).

[17] Mehdi Mirza and Simon Osindero. “Conditional Generative Adversarial Nets”. In: *arXiv e-prints*, arXiv:1411.1784 (2014), arXiv:1411.1784. arXiv: 1411.1784 [cs.LG].

[18] Minhyeok Lee and Junhee Seok. “Controllable Generative Adversarial Network”. In: *arXiv e-prints*, arXiv:1708.00598 (2017), arXiv:1708.00598. arXiv: 1708.00598 [cs.LG].

[19] David Berthelot, Thomas Schumm, and Luke Metz. “BEGAN: Boundary Equilibrium Generative Adversarial Networks”. In: *arXiv e-prints*, arXiv:1703.10717 (2017), arXiv:1703.10717. arXiv: 1703.10717 [cs.LG].

[20] Xi Chen et al. “InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets”. In: *arXiv e-prints* (2016). arXiv: 1606.03657 [cs.LG].

[21] Utkarsh Ojha et al. “Elastic-InfoGAN: Unsupervised Disentangled Representation Learning in Imbalanced Data”. In: *arXiv e-prints*, arXiv:1910.01112 (2019). arXiv: 1910.01112 [cs.LG].

[22] Xun Huang and Serge Belongie. “Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization”. In: *arXiv e-prints*, arXiv:1703.06868 (2017), arXiv:1703.06868. arXiv: 1703.06868 [cs.CV].

[23] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *arXiv e-prints*, arXiv:1502.03167 (2015), arXiv:1502.03167. arXiv: 1502.03167 [cs.LG].

[24] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. “A Learned Representation For Artistic Style”. In: *arXiv e-prints*, arXiv:1610.07629 (2016). arXiv: 1610.07629 [cs.CV].

[25] Tero Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. In: *arXiv e-prints*, arXiv:1710.10196 (2017), arXiv:1710.10196. arXiv: 1710.10196 [cs.NE].

[26] Thu Nguyen-Phuoc et al. “HoloGAN: Unsupervised learning of 3D representations from natural images”. In: *arXiv e-prints*, arXiv:1904.01326 (2019), arXiv:1904.01326. arXiv: 1904.01326 [cs.CV].

[27] Irina Higgins et al. “beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework”. In: *ICLR*. 2017.

[28] Christopher P. Burgess et al. “Understanding disentangling in β-VAE”. In: *arXiv e-prints*, arXiv:1804.03599 (2018), arXiv:1804.03599. arXiv: 1804.03599 [stat.ML].

[29] David Bau et al. “GAN Dissection: Visualizing and Understanding Generative Adversarial Networks”. In: *arXiv e-prints*, arXiv:1811.10597 (2018). arXiv: 1811.10597 [cs.CV].

[30] Ugur Demir and Gozde Unal. “Patch-Based Image Inpainting with Generative Adversarial Networks”. In: *arXiv e-prints*, arXiv:1803.07422 (2018). arXiv: 1803.07422 [cs.CV].

[31] Guilin Liu et al. “Image Inpainting for Irregular Holes Using Partial Convolutions”. In: *arXiv e-prints*, arXiv:1804.07723 (2018), arXiv:1804.07723. arXiv: 1804.07723 [cs.CV].

[32] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. "A Neural Algorithm of Artistic Style". In: *arXiv e-prints*, arXiv:1508.06576 (2015), arXiv:1508.06576. arXiv: 1508.06576 [cs.CV].

[33] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[34] Jiahui Yu et al. "Free-Form Image Inpainting with Gated Convolution". In: *arXiv e-prints*, arXiv:1806.03589 (2018), arXiv:1806.03589. arXiv: 1806.03589 [cs.CV].

[35] Takeru Miyato et al. "Spectral Normalization for Generative Adversarial Networks". In: *arXiv e-prints*, arXiv:1802.05957 (2018). arXiv: 1802.05957 [cs.LG].

[36] Youngjoo Jo and Jongyoul Park. "SC-FEGAN: Face Editing Generative Adversarial Network with User's Sketch and Color". In: *arXiv e-prints* (2019), arXiv:1902.06838. arXiv: 1902.06838 [cs.CV].

[37] Kamyar Nazeri et al. "EdgeConnect: Generative Image Inpainting with Adversarial Edge Learning". In: *arXiv e-prints*, arXiv:1901.00212 (2019). arXiv: 1901.00212 [cs.CV].

[38] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. "Globally and Locally Consistent Image Completion". In: *ACM Transactions on Graphics (Proc. of SIGGRAPH 2017)* 36.4 (2017), p. 107.

[39] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *arXiv e-prints* (2015). arXiv: 1505.04597 [cs.CV].

[40] Ting-Chun Wang et al. "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs". In: *arXiv e-prints*, arXiv:1711.11585 (2017), arXiv:1711.11585. arXiv: 1711.11585 [cs.CV].

[41] Taesung Park et al. "Semantic Image Synthesis with Spatially-Adaptive Normalization". In: *arXiv e-prints*, arXiv:1903.07291 (2019), arXiv:1903.07291. arXiv: 1903.07291 [cs.CV].

[42] Wenzhe Shi et al. "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network". In: *arXiv e-prints*, arXiv:1609.05158 (2016), arXiv:1609.05158. arXiv: 1609.05158 [cs.CV].

[43] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *arXiv e-prints*, arXiv:1511.06434 (2015), arXiv:1511.06434. arXiv: 1511.06434 [cs.LG].

[44] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv e-prints*, arXiv:1409.1556 (2014), arXiv:1409.1556. arXiv: 1409.1556 [cs.CV].

[45] Tim Salimans et al. "Improved Techniques for Training GANs". In: *arXiv e-prints*, arXiv:1606.03498 (2016), arXiv:1606.03498. arXiv: 1606.03498 [cs.LG].

[46] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv e-prints*, arXiv:1412.6980 (2014). arXiv: 1412.6980 [cs.LG].

[47] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *arXiv e-prints*, arXiv:1502.01852 (2015), arXiv:1502.01852. arXiv: 1502.01852 [cs.CV].

[48] Ziwei Liu et al. “Deep Learning Face Attributes in the Wild”. In: *Proceedings of International Conference on Computer Vision (ICCV)*. 2015.

[49] *Cat Dataset. Over 9,000 images of cats with annotated facial features*. https://www.kaggle.com/crawford/cat-dataset. Accessed: 2020-01-03.

[50] Jonathan Krause et al. “3D Object Representations for Fine-Grained Categorization”. In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013.

[51] *Cars Dataset*. https://ai.stanford.edu/~jkrause/cars/car_dataset.html. Accessed: 2020-01-03.

[52] Martin Heusel et al. “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”. In: *arXiv e-prints*, arXiv:1706.08500 (2017), arXiv:1706.08500. arXiv: 1706.08500 [cs.LG].

[53] *A High-Quality PyTorch Implementation of "Globally and Locally Consistent Image Completion"*. https://github.com/otenim/GLCIC-PyTorch. Accessed: 2020-01-03.

[54] *A Port of Fréchet Inception Distance (FID score) to PyTorch*. https://github.com/mseitzer/pytorch-fid. Accessed: 2020-01-03.

[55] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *arXiv e-prints*, arXiv:1512.00567 (2015). arXiv: 1512.00567 [cs.CV].

[56] Mark Sandler et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *arXiv e-prints*, arXiv:1801.04381 (2018), arXiv:1801.04381. arXiv: 1801.04381 [cs.CV].

[57] Cheng-Han Lee et al. “MaskGAN: Towards Diverse and Interactive Facial Image Manipulation”. In: *arXiv preprint arXiv:1907.11922* (2019).