UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

NoGAN: Deblurring Images without Adversarial Training

Author:
Roman VEI

Supervisor:
Orest KUPYN

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

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2020

Declaration of Authorship

I, Roman VEI, declare that this thesis titled, "NoGAN: Deblurring Images without Adversarial Training" 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:

"Civilization transformed man from a food gatherer to a gatherer of pieces of paper: diplomas, employment contracts, money, etc."

Mokokoma Mokhonoana

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Computer Science

NoGAN: Deblurring Images without Adversarial Training

by Roman VEI

Abstract

In this paper, we systematically study generative adversarial networks (GANs) for single image motion deblurring. Firstly, we compare adversarial loss functions, discriminators, and other training configurations to find optimal setup. Secondly, we train the blurred image classification model and use it as a pretrained discriminator in the GAN setup. We train GANs in two ways: with frozen and unfrozen discriminator weights.

Acknowledgements

I want to thank my supervisor Orest Kypyn for directing and mentoring me during the research and all his advice given to me. I want to mention Oles Dobosevych for his support and every week's calls, which push me forward. Also, I would like to thank my parents for their support during the study and my classmates Volodymyr Zabulskyi and Vasyl Borsuk.

Contents

List of Figures

List of Tables

Dedicated to all long spring nights

Chapter 1

Introduction

Every smartphone nowadays has a camera. People want to make a nice-looking photo, and because of that, a lot of problems related to image degradation became very popular. Image blurring is one of the most commonly appeared types of artifacts when taking photos. It may arise due to camera shaking, low brightness, and items our of camera focus. Additionally to that, image deblurring can provide benefits for almost all other tasks because the majority of state-of-the-art solutions assume that as an input, we have a sharp image and don't deal with this problem, so improvement with that can help them to achieve better results.

FIGURE 1.1: Example of image deblurring

Image deblurring task can be solved as an optimization or learning problem. In the optimization approach [Gong et al., 2018], the blur kernel is optimized using gradient descent. The advantage of this approach that we don't need sharp data, but at the same time, these algorithms are usually computationally expensive and have worse results compared to the second one. After the massive success of neural networks, especially deep neural networks in classification and detection problems (Resnet [He et al., 2015], Yolo [Redmon et al., 2015]), they became popular in image restoring fields starting from super-resolution, denoising, dehazing and ending with the deblurring task. Currently, major SOTA implemented in a learning manner. For this formulation, we need pairs of blurry/sharp images.

One of the popular solutions is to use deep neural networks and train them in an adversarial manner - GANs [Kupyn et al., 2017; Kupyn et al., 2019; Zhang et al., 2020; Chen et al., 2019; Lenka, Pandey, and Mittal, 2019; Zhang, Zhen, and Stevenson, 2019]. In this work, we train GANs in not typical setup: in two stages with pretrained discriminator. Besides, we run experiments with frozen and unfrozen weights in discriminator.

Chapter 2

Related Works

2.1 Neural Networks

The idea of neural networks has more than 70 years. The first network was introduced in 1943 by Warren McCulloch, a neurophysiologist from the University of Illinois and Walter Pitts. The central intuition about it that they can emulate our brain's pattern-recognition skills [McCulloch and Pitts, 1943].

The easiest way to describe how neural networks works is that to imagine them as a black box, which as input has an image or another data and outputs some decision based on it. Each network has layers with a lot of units - neurons, which are connected to all adjacent layers neurons.

In the beginning, all these neurons have random weights, but after gradient descent optimization in a learning manner, they manage to predict the meaningful output.

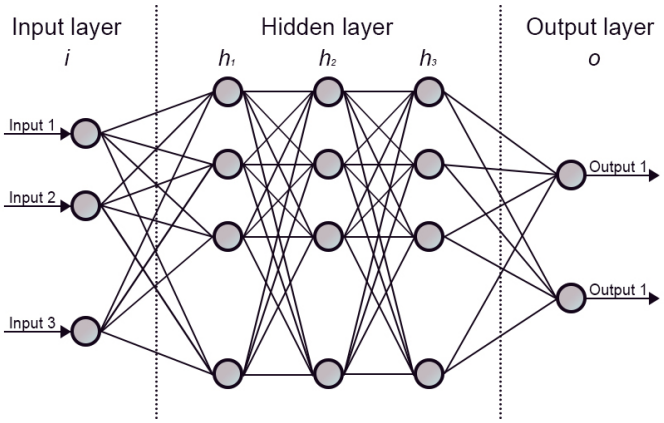

FIGURE 2.1: Neural Network example

There are two common problems with neural networks: **overfitting** and **underfitting**. **Underfitting** usually caused because of low model capacity. For example, when we train the model to classify 1000 classes and use only less than ten layers with a not large amount of neurons. **Overfitting** is a problem when our model only learns all patterns from training data and not scale well to new data. The main challenge with overfitting is that we cannot know how well the model will perform in the future on new samples until we test it in those examples. To address this issue, data usually divided into two parts: *training* and *test* subsets. We train our models on training data but calculate metrics only on test data (optionally, you can also calculate on training split). During training process you can use cross-validation: divide subset into n equal parts, use n - 1 parts for training itself and validate, fine-tune on n-th part.

FIGURE 2.2: Train/Test splits

First neural networks were slow because computers were not optimized for such operations, but at some point, people manage to utilize GPUs to speed up them in hundred times.

2.2 Convolution Neural Networks

Convolution Neural Networks (CNNs) were introduced in the 1980s by researcher Yann LeCun. His version of CNN's was called LeNet (after LeCun) and was able to recognize handwritten digits [Lecun et al., 1998].

After AlexNet [Krizhevsky, Sutskever, and Hinton, 2012] winning in the 2012 year in Imagenet classification challenge [Russakovsky et al., 2015], Convolution Neural Networks (CNNs) with multiple layers became widely used for different tasks. The key difference between them and standard Neural Networks is that they utilize convolution blocks to get useful data from the image. These blocks are square kernels with random weights at the beginning of the training. The huge advantage of this method is that we use information not from every pixel separately, but we aggregate them together using a sliding window. When you input a picture into a CNN, each layer generates activation (feature) maps. These maps highlight the relevant features: the first layer of the network usually detects basic primitives such as vertical, horizontal, or diagonal edges. As you move deeper, layers start detecting more complex objects, like faces.

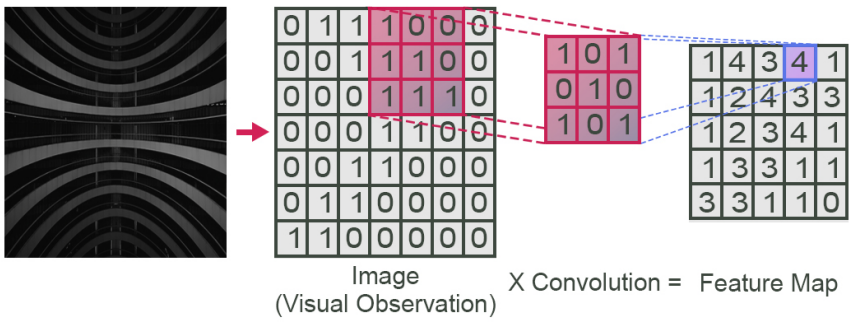

FIGURE 2.3: Convolution operation

2.3 Generative Adversarial Networks

2.3.1 GAN

The generative adversarial networks (GANs) [Goodfellow et al., 2014] were introduced in 2014 by Ian J. Goodfellow et al. They propose a framework for estimating generative models via an adversarial process. It was a simple setup which would be evolved in the future, but even then, they describe future conditional GANs in the conclusions.

The GAN training strategy is a competition between two models. First *generator* model tries to predict the meaningful output from noise and second - *discriminator* tries to to distinguish real image from the generated one. Typically, inside these models, we use multilayer NNs, like ResNet [He et al., 2015], ResNext [Xie et al., 2016], MobileNet [Howard et al., 2017], EfficientNet [Tan and Le, 2019], etc. More formally, we can describe this process in such way:

$$\min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[\log(D(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(1 - D(\tilde{\boldsymbol{x}}))] \tag{2.1}$$

where $\mathbb{P}_r$ is, the data distribution and $\mathbb{P}_g$ is the model distribution.

This simple training setup became one of the most fundamental works in the recent decade. Not surprising, that a lot of state-of-the-art approaches used GANs for different tasks, including single-shot image deblurring.

2.3.2 WGAN and WGAN-GP

The section about GANs can create an illusion that everything is easy. In reality, GAN trainingis very unstable process. Discriminator can start significantly outperform generator, and gradient changes for second one would be too small, or you choose just not proper coefficients in loss terms, and training also fails. In 2017 was published paper called "Improved Training of Wasserstein GANs" [Gulrajani et al., 2017] and introduced new value function with better convergence and called them WGAN and WGAN-GP. They used another measure - Wasserstein distance. This measure is the minimum cost of transporting mass in converting the data distribution q to the data distribution p. Using Kantorovich-Rubinstein duality they obtain new value function:

$$\min_G \max_{D\in\mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[\boldsymbol{D}(\boldsymbol{x})] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] \tag{2.2}$$

where D is all possible of 1-Lipschitz functions, $\mathbb{P}_r$ is, the data distribution and $\mathbb{P}_g$ is the model distribution. For now, we don't use word discriminator because Wasserstein distance doesn't classify an image as 'fake' or 'real.' It approximates score so that we would call it 'critic' function instead, but in GANs it plays discriminator role. Minimizing generator with optimal critic function would lead us to minimize $W(\mathbb{P}_g, \mathbb{P}_r)$ - our measure, so that is why paper called **W**GAN (Wasserstein GAN).

One of the Lipschitz function constraints, is that weights must be in the range $[-c, c]$. The most obvious way to do it is just weight clipping, but this approach has a lot of disadvantages, like optimization problems, etc. So, researchers proposed to use the additional gradient-penalty term (WGAN-**GP**), it is simple L2-difference between weights norm and 1:

$$\underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_\omega}\left[\left(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1\right)^2\right]}_{\text{gradicat penaliy}} \tag{2.3}$$

where $\mathbb{P}_r$ is, the data distribution and $\mathbb{P}_g$ is the model distribution.

In general, these methods are the iterative improvement of default GAN setup, which leads to better results. Even more, this proposed value function could correlate with generator quality.

2.3.3 LSGAN

The paper "Least Squares Generative Adversarial Networks" [Mao et al., 2016] proposed another improvement. This approach, also, tries to address an issue with vanishing gradients. Standard GANs use as a discriminator, a classifier with the sigmoid cross-entropy loss function. The main problem occurs when two distributions far away from each other, so the distance measure near 0 and the generator has slow convergence.

Original GANs use Jensen Shannon Divergence value function for minimizing the distance between distributions:

$$JSD(\mathbb{P}_g, \mathbb{P}_r) = \frac{1}{2}KL(\mathbb{P}_g \| M) + \frac{1}{2}KL(\mathbb{P}_r \| M) \tag{2.4}$$

$$M = \frac{1}{2}(\mathbb{P}_g + \mathbb{P}_r) \tag{2.5}$$

where $\mathbb{P}_r$ is, the data distribution, $\mathbb{P}_g$ is the model distribution. KL is Kullbach-Liebler Divergence with following formula:

$$KL(A \| B) = \sum_{i}^{N} a_i \ln \frac{a_i}{b_i} \tag{2.6}$$

where a_i, b_i - variables from A, B distributions respectively.

LSGAN has a similar setup as in the WGAN, but instead of learning a critic function, it learns a loss function. For regularisation in paper used weight decay to make weights inside some range. But, the critical difference between normal GANs and LSGAN, that LSGAN uses L2 loss, but not log loss. L2 loss takes more care about how far distributions are and penalize it more proportionally what leads to better convergence. In the formula, it looks like this:

$$\begin{aligned} \min_D V_{\text{LSGAN}}(D) &= \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r(\boldsymbol{x})}\left[(D(\boldsymbol{x})-1)^2\right] + \frac{1}{2}\mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_g(\hat{\boldsymbol{x}})}\left[(D(G(\hat{\boldsymbol{x}}))+1)^2\right] \\ \min_G V_{\text{LSGAN}}(G) &= \frac{1}{2}\mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_g(\hat{\boldsymbol{x}})}\left[(D(G(\hat{\boldsymbol{x}})))^2\right] \end{aligned} \tag{2.7}$$

Intuitively, LSGAN wants the target discriminator label for real images to be 1 and generated images to be 0. And for the generator, it wants the target label for generated images to be 1. We can see the intersection between this approach and standard GANs, the difference only in distance measure selections. In practice, they show close to a similar performance.

2.3.4 RSGAN and RaGAN

The next big improvement to GAN performance was new Relativistic GAN (**RSGAN**) [Jolicoeur-Martineau, 2018] proposed by Alexia Jolicoeur-Martineau in 2018. It is not a new loss function; it is a general approach in creating new loss functions from the existing one. In practice, discriminator performs much better than a generator and produces correct high-confident predictions, which lead to a

small gradient and slow training process. In this paper, the researcher decided not to classify real and fake image independently, but to predict how similar these images are. For now, we need to define a transformation from our original discriminator measure to a new one:

$$\begin{aligned} D(x) &= \text{sigmoid}(C(x)) \\ \rightarrow D\left(x_r, x_f\right) &= \text{sigmoid}\left(C\left(x_r\right) - C\left(x_f\right)\right) \end{aligned} \tag{2.8}$$

where $C(x)$ is critic value for given input. How you can see, we take the difference between the two of them and, after that, only optimize.

$$\begin{aligned} L_D^{RSGAN} &= -\mathbb{E}_{\left(x_r, x_f\right) \sim \left(\mathbb{P}_r, \mathbb{P}_g\right)} \left[\log\left(\text{sigmoid}\left(C\left(x_r\right) - C\left(x_f\right)\right)\right)\right] \\ L_G^{RSGAN} &= -\mathbb{E}_{\left(x_r, x_f\right) \sim \left(\mathbb{P}_r, \mathbb{P}_g\right)} \left[\log\left(\text{sigmoid}\left(C\left(x_f\right) - C\left(x_r\right)\right)\right)\right] \end{aligned} \tag{2.9}$$

where x_r is real image, x_f is fake image, $\mathbb{P}_r$ is, the data distribution and $\mathbb{P}_g$ is the model distribution.

This value function has much better convergence. For standard GANs, when discriminator is optimal, we have a situation where $1 - D(x_r) \rightarrow 0$, so our generator learns only from fake images. At that point, standard GAN is not learning how to make images more natural, but RSGAN does not have this issue and learns from both of them.

RAGAN computes the probability that sampled real data is more realistic than fake one for the discriminator and vice versa for generator. These values have very high variance and depend on samples. To overcome this issue author proposed the next improvement to compare not exact values, but their averages to reduce variance:

$$\begin{aligned} L_D^{RaSGAN} &= \mathbb{E}_{x_r \sim \mathbb{P}_r}\left[\log\left(\bar{D}\left(x_r\right)\right)\right] + \mathbb{E}_{x_f \sim \mathbb{P}_g}\left[\log\left(\bar{D}\left(x_f\right)\right)\right] \\ \bar{D}(x) &= \begin{cases} \text{sigmoid}\left(C(x) - \mathbb{E}_{x_f \sim \mathbb{P}_g} C\left(x_f\right)\right) & \text{if } x \text{ is real} \\ \text{sigmoid}\left(C(x) - \mathbb{E}_{x_r \sim \mathbb{P}_r} C\left(x_r\right)\right) & \text{if } x \text{ is fake} \end{cases} \end{aligned} \tag{2.10}$$

The last point to mention that this approach can be used for other value functions to make them "relativistic" and make the training process more stable and faster. For example, existed modifications like RaLSGAN, RSGAN-GP, and more.

2.4 Task Setup

FIGURE 2.4: Deblurring in action

Blind image deblurring is the image restoration task that aims to restore a sharp image from blurry one. This blur can be caused by out-of-focus, object motion, camera blur, or poor visibility in the night. Also, this problem can be interpreted as a subtask from the image-to-image translation domain. Mathematically, it can be expressed as the following statement:

$$I^{\text{blur}} = I^{\text{sharp}} * k + n \tag{2.11}$$

where * denotes 2D convolution .I^{blur} is blurred image, I^{sharp} is corresponding sharp image, k is a blur kernel, n - some random noise.

2.5 Overview of Previous Methods

A lot of methods have been proposed in recent years to accomplish this task. We divide them into categories and briefly describe their approaches. The first three categories try to estimate kernel coefficients. The next three - use end-to-end approach.

2.5.1 Motion Vector Estimation

Jian Sun et al. [Sun et al., 2015] propose per-patch motion vector estimation. For each patch used neural network to find probability distribution of motion kernels:

$$P\left(m = (l, o)|\Psi_p\right) \tag{2.12}$$

where Ψ_p is patch centred at pixel p and m is motion kernel with length l and angle o. m and l are discretized sets and they solve it like multiclass classification.

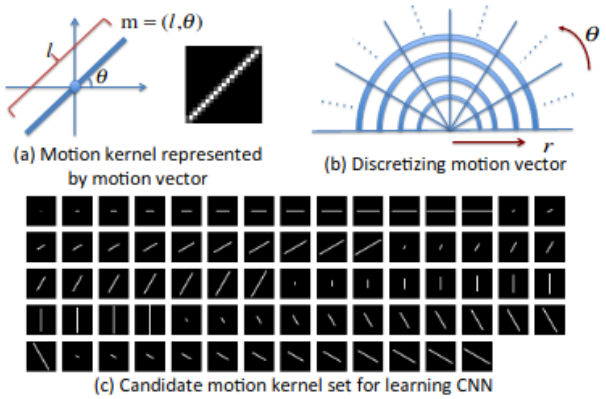

FIGURE 2.5: Visualization of motion vector [Sun et al., 2015]

Since the many patches have overlapping the author propose to use weighted average defined as a confidence of motion kernel:

$$C\left(m_p = (l, o)\right) = \frac{1}{Z} \sum_{q:p \in \Psi_q} G_\sigma \left(\left\|x_p - x_q\right\|^2\right) P\left(m = (l, o)|\Psi_q\right) \tag{2.13}$$

where q is center pixel, p is neighbour pixel, G_σ is Gaussian function and Z is normalization constant defined as $\sum_{q:p \in \Psi_q} G_\sigma \left(\left\|x_p - x_q\right\|\right)$ After estimating kernels used

Markov Random Function (MRF) [Kinderman and Snell, 1980] to smooth the transition between neighboring pixels and generate dense motion field:

$$\sum_{p \in \Omega} \left[-C\left(m_p = (l_p, o_p)\right) \right] + \sum_{q \in N(p)} \lambda \left[\left(u_p - u_q\right)^2 + \left(v_p - v_q\right)^2 \right] \tag{2.14}$$

where Ω is all image patches. u_p, u_q, v_p, v_q are defined as $u_i = l_i cos(o_i)$ and $v_i = l_i sin(o_i)$ for $i = p, q$. $N(p)$ is all neighbour pixels of p. The first term is a main and used earlier predicted confidence of motion kernel, where the second term had regularization purposes. After predicting the motion field, Sun et al. deconvolve the blurred image to get a sharp one.

2.5.2 Image Dense Motion Flow Estimation

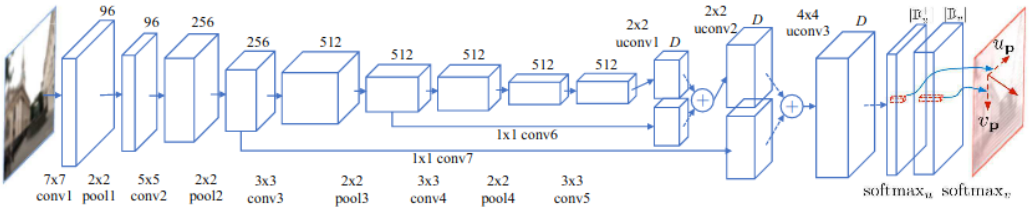

FIGURE 2.6: Architecture of Dong Gong approach [Gong et al., 2016]

Predicting motion flow for an image also used as one of the approaches. The difference between the previous paper is that now we predict a motion flow map for the whole image without using any post-processing algorithms like Markov Random Field (MRF) [Kinderman and Snell, 1980]. One of the purposes is Dong Gong et al. paper [Gong et al., 2016]. Inside neural networks, they used: 7 convolutional, four max-pooling, and three up-sample convolutional layers. In the output, they predict motion maps. Their map is formed in such manner:

$$\mathcal{M} = (\mathbf{U}, \mathbf{V}) \tag{2.15}$$

where $\mathbf{U}(i,j) \in \mathbb{D}_u^+$ and $\mathbf{V}(i,j) \in \mathbb{D}_v, \forall i, j$, $\mathbb{D}_v, \mathbb{D}_u^+$ - all possible labels. $\mathbf{U}$ and $\mathbf{V}$ denote the motion maps in the horizontal and vertical directions.

This approach can show better results than the previous paper because:

- larger spatial context given (we have access to the full image);
- it doesn't require additional post-processing (like MRF in the previous example);
- assume not uniform motion blur inside the image, which is better generalize to real-life data;

2.5.3 Fourier Coefficients Estimation

In this setup proposed by Ayan Chakrabarti [Chakrabarti, 2016]. Authors don't predict kernel coefficients directly but predicted complex Fourier coefficients of a deconvolution filter, which applied to the input. During training, they do per-patch overlapping blind deconvolution for predicting Fourier coefficients of a filter with

a neural network, take inverse discrete Fourier transform (IDFT) of it, aggregate results from patches and form a final estimate of a sharp image.

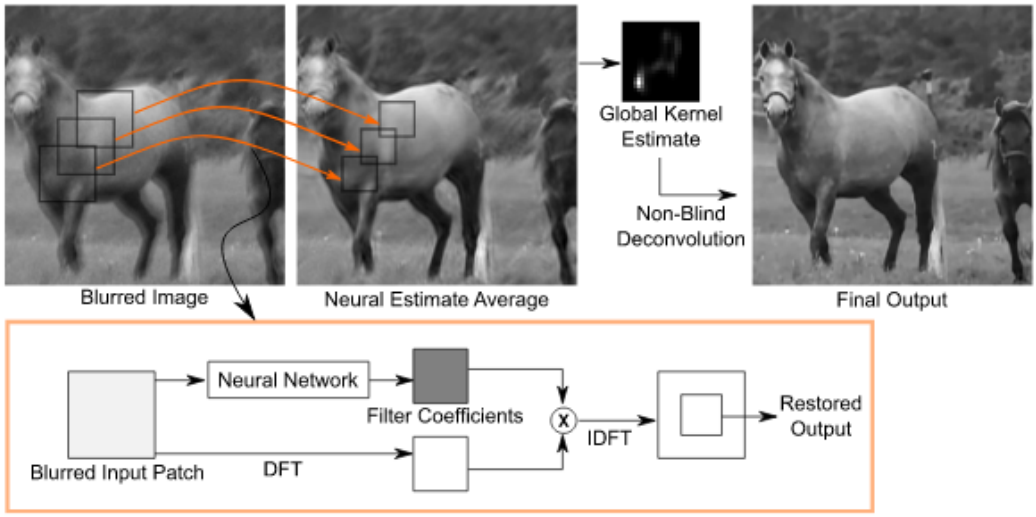

FIGURE 2.7: Architecture of Ayan Chakrabarti approach [Chakrabarti, 2016]

The main motivation for predicting Fourier coefficients is, first of all, to enforce specific parametric form.

2.5.4 GANs

Recent approaches use neural networks for predicting image deblurring in an end-to-end manner. The difference between standard neural networks and GANs, that second one has adversarial loss term inside their total loss and trains two competitive subnets simultaneously (see GAN section).

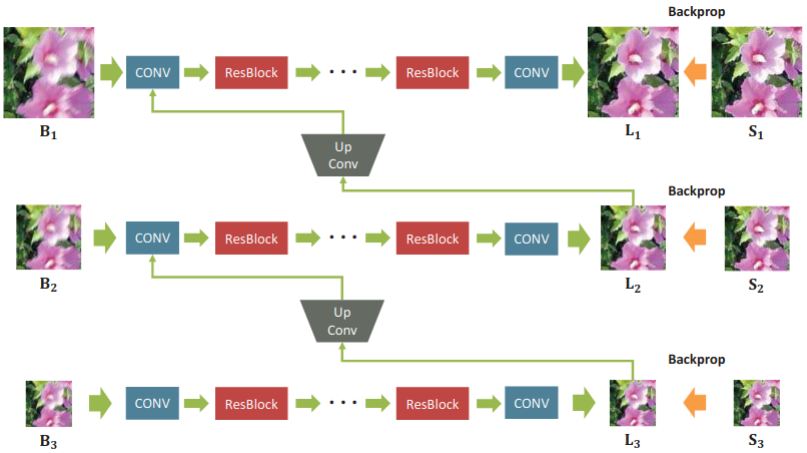

FIGURE 2.8: Nah et al. network architecture [Nah, Kim, and Lee, 2016]

Inside neural networks in most cases used convolutions, but one of the biggest problems of convolution neural networks is a small receptive field. Nah et al. [Nah, Kim, and Lee, 2016] uses multiple scales, and calculate the total loss for each size

separately to handle this issue. They use standard GAN loss as an adversarial one and L2 difference as a content loss. Total lass is a weighted sum of these two terms:

$$\mathcal{L}_{cont} = \frac{1}{2K} \sum_{k=1}^{K} \frac{1}{c_k w_k h_k} \|L_k - S_k\|^2 \tag{2.16}$$

$$\mathcal{L}_{adv} = \mathbb{E}_{\mathbf{x} \sim P_r}[\log D(\mathbf{x})] + \mathbb{E}_{\hat{\mathbf{x}} \sim P_g}[\log(1 - D(G(\hat{\mathbf{x}})))] \tag{2.17}$$

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{cont}} + \lambda \times \mathcal{L}_{\text{adv}} \tag{2.18}$$

where L_k, S_k denote model predictions and ground truth image in k scale. $\frac{1}{c_k w_k h_k}$ is normalization factor for every loss at each scale. c_k, w_k, h_k - number of channels, width, height for scale k. $\frac{1}{2K}$ is normalization factor for final content loss. The problem of this solution is a large amount of parameters, which leads to slow convergence and slow inference because of multi-scale approach.

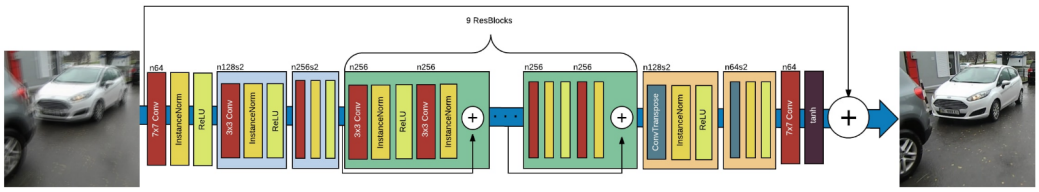

FIGURE 2.9: "DeblurGAN" architecture from Kupyn et al. [Kupyn et al., 2017]

In the same 2017 year, Kupyn et al. proposed their algorithm called "DeblurGAN" [Kupyn et al., 2017], which was an adaptation of detection algorithms for a deblurring task inside GANs. The main advantage of this method is a lot of problems with architecture modeling are solved, because they already use optimal and proven architectures, even in another domain. Another big step from this paper is generating motion blur using random trajectories. Also, it is one of the fastest and accurate algorithms in those years. Two years later, Kupyn et al. proposed the second version of their algorithm "DeblurGAN-v2" [Kupyn et al., 2019], which, also, showed SOTA results in the single image deblurring. More details about this project in the next chapter, Proposed Methods.

2.5.5 End-to-end Neural Networks

Tao et al. paper [Tao et al., 2018] uses a similar multi-scale structure as Nah et al. [Nah, Kim, and Lee, 2016] to increase receptive field, but have some key differences:

- Not used the adversarial term in loss;
- Weights between scales are shared;
- Significantly reduced the number of parameters by using smaller encoder-decoder block;
- Between encoder and decoder used LSTM;
- Used popular and optimal architectures inside backbone of the network;

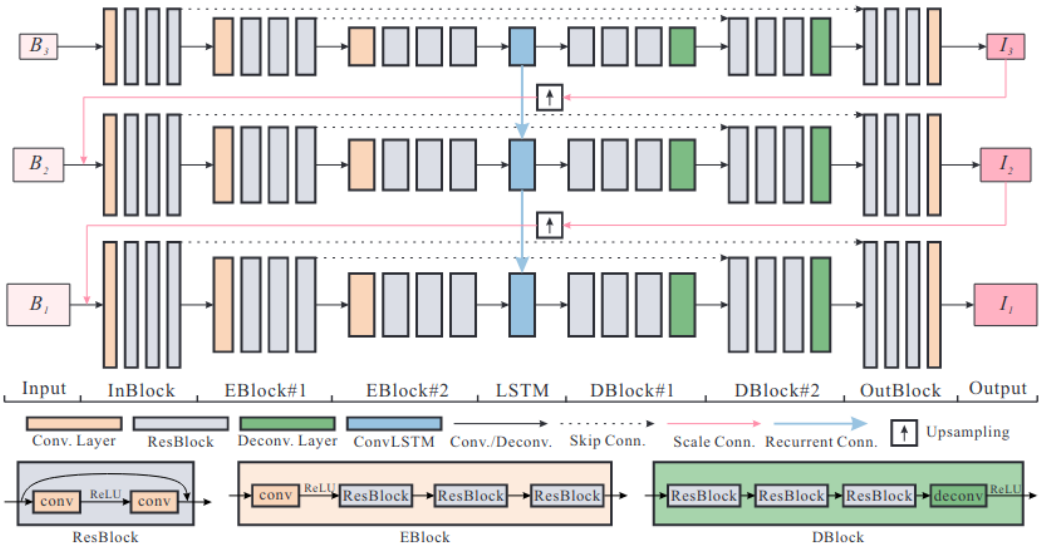

FIGURE 2.10: Tao et al. network architecture [Tao et al., 2018]

In general, end-to-end approach similar to GANs, but have better convergence, because GANs training process very unstable. From disadvantages, adversarial term adds additional accuracy and leads for better model performance.

Chapter 3

Proposed Methods

In our work, we use the GAN approach with some modifications. Standard GAN training setup has two dummy subnetworks in the beginning, and they simultaneously update their weights during training. The main difference in our approach is that we use pretrained discriminator. To achieve this, we need to train our model in two stages. Firstly, we train discriminator to estimate blurriness of the image from 0 (sharp) to 1. This number would be used as a penalty for the discriminator and forced it to generate more sharp images. Secondly, we train our generator with this pretrained discriminator. We explore two setups:

- with frozen discriminator weights - in that case, our model trained in a non-adversarial manner, because of only one subnetwork updates weights (generator);
- with unfrozen discriminator weights - it is standard GAN setup, where two subnetworks trained at the same time;

3.1 Network Architecture

We select Kupyn et al. algorithm "DeblurGAN-v2" [Kupyn et al., 2019] as a base of our project. There are several reasons for that:

- Extendable code structure;
- SOTA results in the deblurring field;
- Their custom algorithm of generating realistic blur;

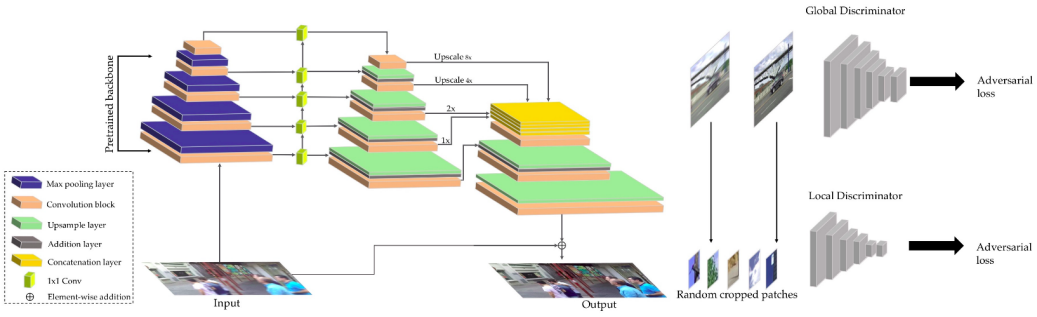

FIGURE 3.1: "DeblurGAN-v2" architecture from Kupyn et al. [Kupyn et al., 2019]

The network takes as the input 3-channel RGB image. This image is processed using one of standard backbones, it can be anything, like: ResNet [He et al., 2015],

ResNext [Xie et al., 2016], MobileNet [Howard et al., 2017], EfficientNet [Tan and Le, 2019], etc. After that comes into play Feature Pyramid Network (FPN) [Lin et al., 2016] - the main architecture improvement of "DeblurGAN-v2". The FPN is a feature extractor designed to replace simple inefficient image resizing for different scales followed with forwarding pass. Firstly, this idea was used in detection problems for better prediction of different objects in different scales. FPN composes of a bottom-up and top-down pathway. The bottom-up path is the regular backbone neural network. Every backbone has from 3 to 7 "unformal" feature maps - intermediate feature representations. You can think about it, like gradually increasing features, which describe the input photo: the first feature maps capture primitives, while last one - high-level structures, like people, on the image. As we go down-top way, the spatial resolution decreases, because our feature maps become smaller, but semantic value increases, because feature maps become more representative. These feature maps outputs connected with a top-down pathway of FPN. This FPN structure utilizes all benefits of feature maps.

3.1.1 Loss Function

As a loss function used composed function from "DeblurGAN-v2":

$$L_G = 0.5 * L_p + 0.006 * L_{cont} + 0.01 * L_{adv} \tag{3.1}$$

where L_p - perceptual term, which compare predicted and real image using VGG [Simonyan and Zisserman, 2014] feature maps, L_{cont} - L2 loss between actual and predicted pictures to compare the content of them, L_{adv} - the adversarial term for GAN training where we use RaGAN (see RSGAN and RaGAN section). Coefficients are taken from the original paper. As discriminator loss, we use a discriminator term from RaGAN loss.

3.1.2 Pretrained discriminator

The main difference between our setup and "DeblurGAN-v2" is that before the GAN training step, we train discriminator, and only after that, we train GAN with frozen/unfrozen weights. The discriminator is a simple classifier. Typically, inside, it used a neural network with few layers, but we decided to do experiments with a more complex one.

3.1.3 Pipeline

We can consider our neural network as end-to-end, so pipeline mainly consists of it and small pre-/post-processing steps. It has two parts: training discriminator and training GAN itself.

For discriminator training sampled batch of images from blurry and sharp image pools and trained to classify them: 0 - sharp, 1 - blurry. We can think about it, like a measure of blurriness. For classification we choose standard backbones and *NLayerDiscriminator* from "DeblurGAN-v2" paper - simple n layers neural network.

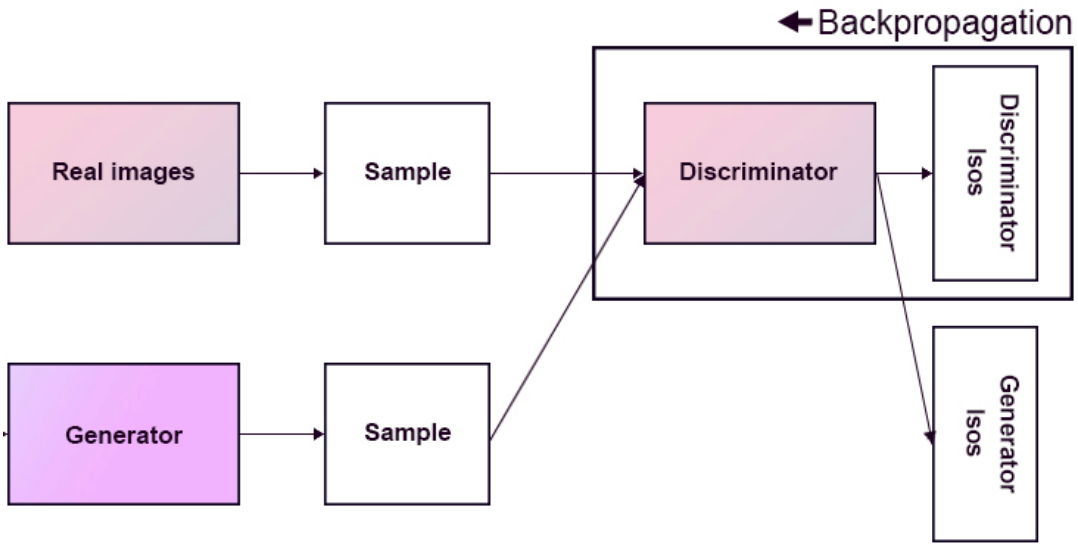

FIGURE 3.2: Pipeline

The second stage trained as standard GANs, but with one difference: pretrained discriminator. Firstly, we sample a pair of a sharp image and corresponding blurry one from the dataset. These two images we crop 256x256 pixels patch and augment with the following functions (order preserved, more in Implementation Details section):

- HorizontalFlip;
- ShiftScaleRotate;
- Transpose;
- OpticalDistortion;
- ElasticTransform;
- Crop;
- Pad;
- Normalize;

After that, the blurry one goes into a generator to generate a prediction. Finally, this prediction compared with a sharp image using composed loss from Loss Function subsection and backpropagated to update weights. We have to setups, where we update and don't update discriminator.

3.1.4 Datasets

For training, used the **GoPro Dataset** [Nah, Kim, and Lee, 2016]. Images for this dataset are captured by GoPro Hero 4 camera. The camera capture video sequences for sharp images. Blurry images are generated over averaging neighbor frames. In total, in dataset 3214, image pairs and 33 video sequences. The ratio of the train/test split (0.67/0.33) was preserved from the original paper. However, we prepare GoPro Dataset as described in [Kupyn et al., 2019]. They use the interpolation model to increase frame rate from 240 to 3840 and perform average pooling with the same window size.

FIGURE 3.3: Blurry GoPro dataset image examples

FIGURE 3.4: Sharp GoPro dataset image examples

Chapter 4

Experiments

4.1 Experiment Setup

For training we use images from GoPro dataset and make random 256x256 pixels crop. *batch_size* = 1 used to avoid problems with batch normalization ([Ioffe and Szegedy, 2015]). *num_epochs* selected to 200. As optimizer used standard Adam with 0.0001. As metrics we choose **SSIM** and **PSNR**. **SSIM** is a perceptual metric that calculates image degradation:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{\left(\mu_x^2 + \mu_y^2 + c_1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right)} \tag{4.1}$$

where μ_x the average over x, μ_y the average over y, σ_x the variance over x, σ_y the variance over y, σ_{xy} the covariance of x and y, c_1, c_2 - constants to prevent division by zero. The main difference between **MSE** and **PSNR** that it measures not *absolute error*, but tries to incorporate luminance and contrast terms inside. **PSNR** is older metric that indicates ratio between maximum pixel value and **M**ean **S**quare **E**rror (**MSE**) and maximum pixel value (usually 255):

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \tag{4.2}$$

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \tag{4.3}$$

where I, K are original and corrupted images and MAX_I - maximum value on image. For RGB images used sum of $PSNR$ across all channels.

These two metrics are very popular for this task setup, but recently more and more people say that we need another metrics to measure image degradation.

4.2 Baselines

First of all, we train baselines to see numbers. We select all possible combinations of adversarial losses and GAN setups: **PatchGAN**, **DoubleGAN** and **SingleGAN**. **PatchGAN** is a setup when we divide an image into N*N patches and calculate it separately. When N=*image size*, it is standard **SingleGAN** (**FullGAN**). **DoubleGAN** is a setup, when we use **SingleGAN** and **PatchGAN** simultaneously, to preserve local and global information for better results.

As shown in table, there is no much difference in loss function and GAN type. All results in range ∼0.81-0.83 for SSIM and ∼27.0-28.0 for PSNR.

	Single		Patch		Double-scale	
Adversarial loss	**SSIM**	**PSNR**	**SSIM**	**PSNR**	**SSIM**	**PSNR**
GAN	**0.8375**	28.65	0.8264	28.49	0.8257	28.29
LSGAN	0.8306	28.59	0.8271	28.69	0.8373	28.87
WGAN-GP	0.818	27.6	0.8408	28.84	0.8329	28.11
RaGAN	0.8372	**28.67**	0.8333	28.58	0.842	28.91

TABLE 4.1: Baseline results

4.3 Classification Problem

We train 4 models for a classification setup, where model predicts estimate of blurriness of the images from 0 (sharp) to 1 (blurry):

- standard NLayerDiscriminator with 5 layers from [Kupyn et al., 2019] repository;
- MobileNetV3 - large [Howard et al., 2019];
- EfficientNet B0 [Tan and Le, 2019];
- SE-ResNext50 [Xie et al., 2016, Hu, Shen, and Sun, 2017];

With this setup we have following result for classification task:

Model Name	Validation Loss	Accuracy
NLayerDiscriminator	0.284	73.19
MobileNetV3 - large	0.038	78.34
EfficientNet B0	0.083	80.89
SE-ResNext 50	**0.019**	**81.93**

TABLE 4.2: Classification results

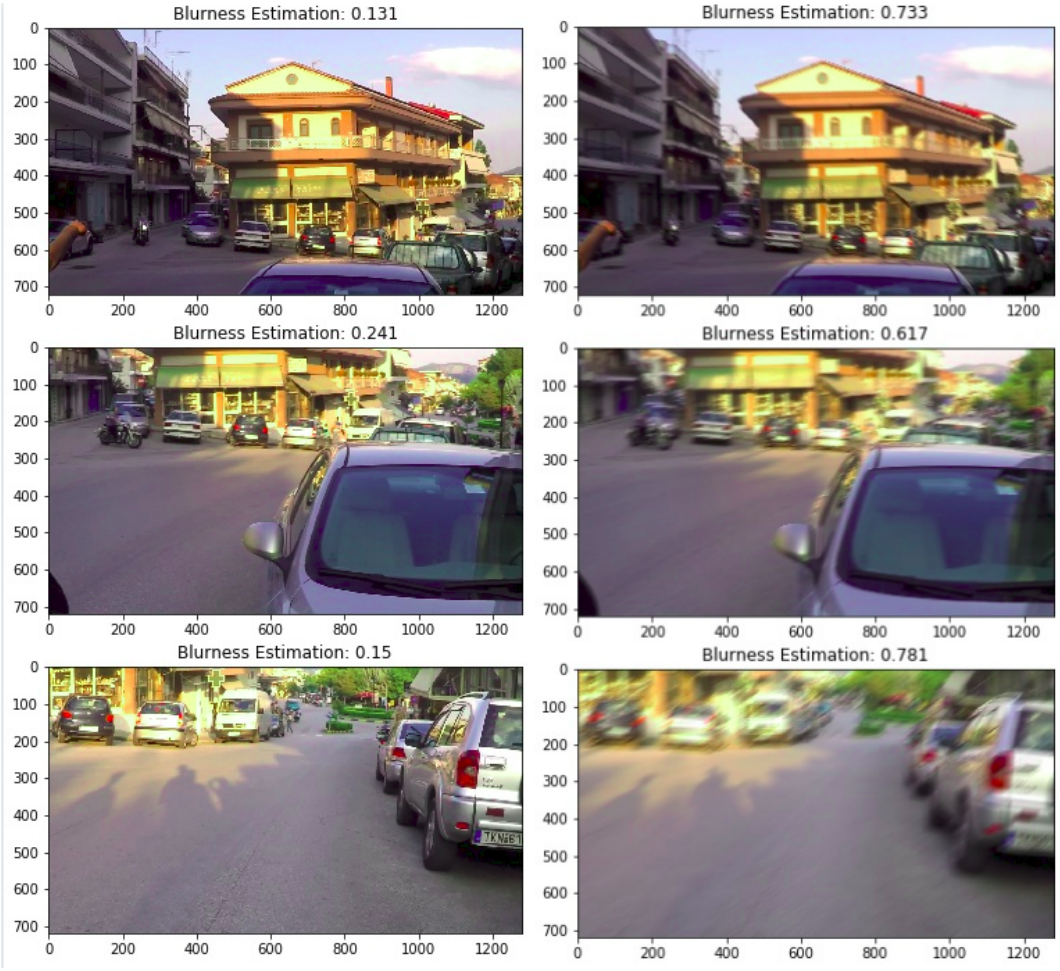

FIGURE 4.1: "MobileNetV3 - large" model blurriness estimations

From the table, we see that backbone doesn't play a big role in classification. It can be caused by a lack of training time. After accomplishing these experiments, we move to the next stage and train with already pretrained discriminator checkpoints.

4.4 GAN with Pretrained Discriminator

Model Name	Loss	Pretrained	SSIM	PSNR
NLayerDiscriminator	WGAN-GP	No	0.7766	26.03
NLayerDiscriminator	WGAN-GP	Yes	0.7773	26.12
NLayerDiscriminator	RaGAN	No	0.8194	28.25
NLayerDiscriminator	RaGAN	Yes	0.8225	28.33
MobileNetV3 - large	RaGAN	Yes	0.82	28.4
EfficientNet B0	RaGAN	Yes	0.8297	28.51
SE-ResNext 50	RaGAN	Yes	**0.8399**	**28.8**

TABLE 4.3: Results with Pretrained Discriminator

More complex backbones, in fact, increase the metrics. Also, we mention that pretrained discriminators add additional points to metrics. However, these changes

are not significant, so you need to make your trade-off between results and model complexity.

FIGURE 4.2: SE-ResNext 50 outputs

4.5 GAN with frozen Pretrained Discriminator

The next experiments are with frozen discriminator weights. During training, we update only generator weights. It slightly improves model training speed, but results are lower compare to standard GAN setup: How we can see from results,

Model Name	Loss	Pretrained	Frozen Weights	SSIM	PSNR
NLayerDiscriminator	RaGAN	Yes	No	**0.8225**	28.33
NLayerDiscriminator	RaGAN	Yes	Yes	0.7765	26.1
MobileNetV3 - large	RaGAN	Yes	No	0.82	**28.4**
MobileNetV3 - large	RaGAN	Yes	Yes	0.7812	26.41

TABLE 4.4: Results with Pretrained Discriminator

training generators, and discriminators simultaneously in GANs leads to better results.

Chapter 5

Implementation Details

As a starting point we select "DeblurGAN-v2" [Kupyn et al., 2019] repository. The most important parts from following repository we briefly discuss:

- **models/** - folder with different backbones;
- **adversarial_trainer.py** - file with PatchGAN, DoubleGAN, standard GAN wrappers;
- **train.py** - the starting point of the GAN training process. It contains **Trainer** with full training step code;
- **aug.py** - file where you define augumentations;
- **dataset.py** - file which contains **Dataset** subclass;
- **inference.py** - inference code;

Our code improvements of this project:

- added **wandb** logging support;
- **classifier.py** - file with **Classifier** wrapper for backbones to solve classification task;
- **train_classification.py** - the starting point of the classification training process;
- restructure code in some parts to make project more extendable;
- **Fire** library support for better usage from command line interface (CLI);

5.1 Frameworks

We use **PyTorch** [Paszke et al., 2019] deep learning framework with **Python 3.6** to run our experiments. It is Pythonic, easy to learn framework, which is mostly used in recent studies. Moreover, it mainly developed by Facebook and has large community. In addition, we use following libraries:

- **albunetations** [Buslaev et al., 2018] - augumentations library;
- **wandb** [Hobbs and Svetlichnaya, 2020] - library for logging;
- **tensorboard** - library for logging;
- **pytorchcv** - library with different backbones;
- **numpy** - library for input matrix preprocessing and postprocessing;

- **skimage** - library from which used **SSIM** function;
- **Fire** - library-wrapper for better experience using CLI;
- **cv2** - library for working with images;

5.2 Computational Resources

We run our experiments on the server with Nvidia GeForce GTX 1080 Ti with 11,178 MB GDDR5X memory on the card. One epoch takes near ∼8 minutes to train. One experiment has 200 epochs, so on average, it takes one day to get results.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.36       Driver Version: 440.36       CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 00000000:0B:00.0 Off |                  N/A |
| 21%   25C    P0    56W / 250W |      0MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

FIGURE 5.1: "nvidia-smi" command

Chapter 6

Conclusions

6.1 Results Summary

In current work, we explore another side of GAN training: discriminators and different setups of it. From the experiments, we see that for this task setup, different discriminator types (**PatchGAN**, **(Full)GAN**, and **DoubleGAN**) and various adversarial losses don't play a key role. Pretrained discriminators can slightly improve the situation with metrics. After that, we show that the best approach is to use pretrained discriminator with unfrozen weights and the ability to study during the GAN training process.

6.2 Future Work

To improve results, we need to make changes in architecture; another loss or discriminator type doesn't make a real change. Further steps to explore:

- use **BiFPN** [Tan, Pang, and Le, 2019] instead of **FPN** [Lin et al., 2016];
- explore model **quantization** introduced in PyTorch 1.5 to make model significantly smaller and faster with small quality degradation;
- try to generate more data and improve blur generation algorithm synthetically;

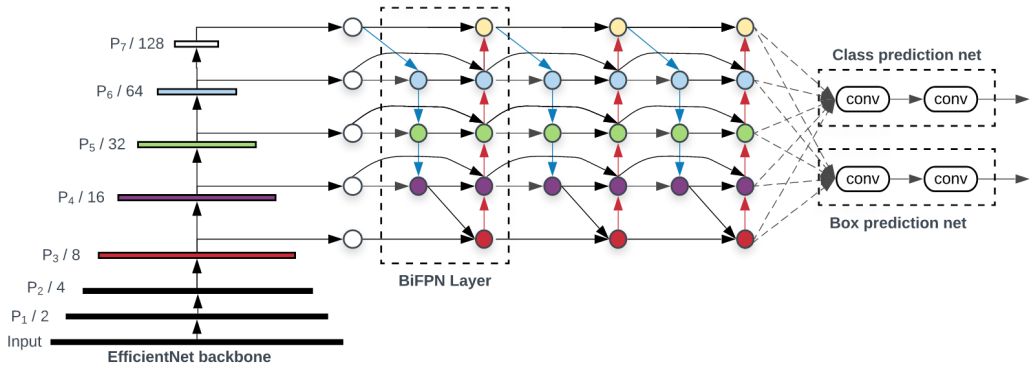

FIGURE 6.1: BiFPN architecure [Tan, Pang, and Le, 2019]

Bibliography

Buslaev, Alexander V. et al. (2018). "Albumentations: fast and flexible image augmentations". In: *CoRR* abs/1809.06839. arXiv: 1809.06839. URL: http://arxiv.org/abs/1809.06839.

Chakrabarti, Ayan (2016). "A Neural Approach to Blind Motion Deblurring". In: *CoRR* abs/1603.04771. arXiv: 1603.04771. URL: http://arxiv.org/abs/1603.04771.

Chen, Fuhai et al. (2019). *Semantic-aware Image Deblurring*. arXiv: 1910.03853 [cs.CV].

Gong, Dong et al. (2016). "From Motion Blur to Motion Flow: a Deep Learning Solution for Removing Heterogeneous Motion Blur". In: *CoRR* abs/1612.02583. arXiv: 1612.02583. URL: http://arxiv.org/abs/1612.02583.

Gong, Dong et al. (2018). "Learning an Optimizer for Image Deconvolution". In: *CoRR* abs/1804.03368. arXiv: 1804.03368. URL: http://arxiv.org/abs/1804.03368.

Goodfellow, Ian J. et al. (2014). *Generative Adversarial Networks*. arXiv: 1406.2661 [stat.ML].

Gulrajani, Ishaan et al. (2017). "Improved Training of Wasserstein GANs". In: *CoRR* abs/1704.00028. arXiv: 1704.00028. URL: http://arxiv.org/abs/1704.00028.

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

Hobbs, Andrew and Stacey Svetlichnaya (2020). *Satellite-based Prediction of Forage Conditions for Livestock in Northern Kenya*. arXiv: 2004.04081 [cs.CV].

Howard, Andrew et al. (2019). "Searching for MobileNetV3". In: *CoRR* abs/1905.02244. arXiv: 1905.02244. URL: http://arxiv.org/abs/1905.02244.

Howard, Andrew G. et al. (2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861. arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

Hu, Jie, Li Shen, and Gang Sun (2017). "Squeeze-and-Excitation Networks". In: *CoRR* abs/1709.01507. arXiv: 1709.01507. URL: http://arxiv.org/abs/1709.01507.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167. arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

Jolicoeur-Martineau, Alexia (2018). "The relativistic discriminator: a key element missing from standard GAN". In: *CoRR* abs/1807.00734. arXiv: 1807.00734. URL: http://arxiv.org/abs/1807.00734.

Kinderman, R. and S.L. Snell (1980). *Markov random fields and their applications*. American mathematical society.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Kupyn, Orest et al. (2017). "DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks". In: *CoRR* abs/1711.07064. arXiv: 1711.07064. URL: http://arxiv.org/abs/1711.07064.

Kupyn, Orest et al. (2019). *DeblurGAN-v2: Deblurring (Orders-of-Magnitude) Faster and Better*. arXiv: 1908.03826 [cs.CV].

Lecun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*, pp. 2278–2324.

Lenka, Manoj Kumar, Anubha Pandey, and Anurag Mittal (2019). *Blind Deblurring Using GANs*. arXiv: 1907.11880 [eess.IV].

Lin, Tsung-Yi et al. (2016). "Feature Pyramid Networks for Object Detection". In: *CoRR* abs/1612.03144. arXiv: 1612.03144. URL: http://arxiv.org/abs/1612.03144.

Mao, Xudong et al. (2016). "Multi-class Generative Adversarial Networks with the L2 Loss Function". In: *CoRR* abs/1611.04076. arXiv: 1611.04076. URL: http://arxiv.org/abs/1611.04076.

McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.

Nah, Seungjun, Tae Hyun Kim, and Kyoung Mu Lee (2016). "Deep Multi-scale Convolutional Neural Network for Dynamic Scene Deblurring". In: *CoRR* abs/1612.02177. arXiv: 1612.02177. URL: http://arxiv.org/abs/1612.02177.

Paszke, Adam et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. arXiv: 1912.01703 [cs.LG].

Redmon, Joseph et al. (2015). "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640. arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

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

Sun, Jian et al. (2015). "Learning a Convolutional Neural Network for Non-uniform Motion Blur Removal". In: *CoRR* abs/1503.00593. arXiv: 1503.00593. URL: http://arxiv.org/abs/1503.00593.

Tan, Mingxing and Quoc V. Le (2019). "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *CoRR* abs/1905.11946. arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946.

Tan, Mingxing, Ruoming Pang, and Quoc V. Le (2019). *EfficientDet: Scalable and Efficient Object Detection*. arXiv: 1911.09070 [cs.CV].

Tao, Xin et al. (2018). "Scale-recurrent Network for Deep Image Deblurring". In: *CoRR* abs/1802.01770. arXiv: 1802.01770. URL: http://arxiv.org/abs/1802.01770.

Xie, Saining et al. (2016). "Aggregated Residual Transformations for Deep Neural Networks". In: *CoRR* abs/1611.05431. arXiv: 1611.05431. URL: http://arxiv.org/abs/1611.05431.

Zhang, Kaihao et al. (2020). "Deblurring by Realistic Blurring". In: *ArXiv* abs/2004.01860.

Zhang, Shuang, Ada Zhen, and Robert L. Stevenson (2019). "GAN Based Image Deblurring Using Dark Channel Prior". In: *CoRR* abs/1903.00107. arXiv: 1903.00107. URL: http://arxiv.org/abs/1903.00107.