FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

# ASSIGNMENT OF BACHELOR'S THESIS

| | |
|---|---|
| **Title:** | Automatic Image Colorization |
| **Student:** | Justína Kušpálová |
| **Supervisor:** | Ing. Magda Friedjungová |
| **Study Programme:** | Informatics |
| **Study Branch:** | Knowledge Engineering |
| **Department:** | Department of Applied Mathematics |
| **Validity:** | Until the end of summer semester 2019/20 |

## Instructions

The problem of automatic image colorization (i.e., the coloring of black and white images) has been discussed a lot over the last decade. It is highly ill-posed due to the large degrees of freedom during the assignment of color information. We attempt to generalize the colorization procedure using state-of-the-art methods that are based on generative networks.

1) Review and theoretically describe the state-of-the-art approaches to image colorization.
2) Use or implement at least two of the reviewed approaches and experimentally compare their performance on a suitable data set. Use existing implementations as much as possible.
3) Propose a direction for further improvement of image colorization approaches.

## References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague November 25, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

# Automatic Image Colorization

***Justína Kušpálová***

Department of Applied Mathematics
Supervisor: Ing. Magda Friedjungová

August 13, 2020

# Acknowledgements

I would like to thank my supervisor Ing. Magda Friedjungová. She was always willing to help and give advice. I would also like to thank my family, which was supportive during my entire studies.

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on August 13, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology


### Citation of this thesis

Kušpálová, Justína. *Automatic Image Colorization*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

# Abstrakt

Množstvo rôznych metód bolo navrhnutých na ofarbovanie obrázkov. V mojej bakalárskej práci implementujem automatické kolorizovanie obrázkov pomocou generatívnych kontradiktórnych sietí – GANov. Ukázali sľubné výsledky pri generovaní rôznych dát, vrátane obrázkov. Používam dve modfikácie GANov – DCGAN a CycleGAN. Tieto dve metódy porovnávam a vyhodnocujem pomocou najpoužívanješích metrík, vhodných pre tento problém. V záverečnej časti práce sú zobrazené aj ukážkové obrázky, vygenerované jednotlivými modelmi.

**Klíčová slova** automatické ofarbovanie obrázkov, strojové učenie, generatívna kontradiktórna sieť, konvolučná neurónová sieť, image-to-image translation

# Abstract

Many different methods have been suggested to colorize images yet. In this thesis, I try to implement a fully automatic image colorization using generative adversarial networks – GANs. They have shown promising results in generating various kinds of data, including images. I adopt two different modifications of GANs – DCGAN and CycleGAN. These two methods are compared, and

results are evaluated using the most common metrics used for this problem. Example images are provided as well.

**Keywords** automatic image colorization, machine learning, generative adversarial network, convolutional neural network, image-to-image translation

# Contents

# List of Figures

# List of Tables

# Introduction

It is not a difficult task for a person to fill the black-and-white regions of a picture with colors – even children are capable of accomplishing a similar task in a coloring book. Some objects keep their color – the grass is green, or the sky is blue, others are dependent on the human imagination and can be assigned several plausible colors. But what is easy for a human can be quite challenging for the computer. And so is the image colorization.

A variety of techniques has been employed to produce credible, colorful images from their grayscale counterparts. With the arrival of convolutional neural networks, machine learning models have shown a huge potential in different image-to-image translation tasks. Automatic image colorization is not an exception. Many models have been trained to improve the results in this challenging area. These models can be used as a simple tool to bring old, black-and-white photos into the colorful world.

This work focuses on automatic image colorization using generative adversarial networks. In recent years, it has become one of the most popular deep learning techniques due to its ability to deal with numerous problems from various areas. The studies have also shown its robust results in the field of image colorization.

## Objectives

The goals of this thesis are following:

- review current state-of-the-art machine learning methods utilized for automatic image colorization
- implement selected methods
- evaluate their performance on publicly available datasets

Firstly, I will introduce image colorization in general and survey up-to-date machine learning methods that tackle this problem. After that, I will implement similar models keeping the best practice strategies, and I will evaluate their performance on the benchmark CIFAR-10 dataset. Finally, I will apply these models to another dataset to observe their behavior on higher-quality images.

Chapter 1

# Image colorization

Image colorization can be defined as a problem of taking a grayscale image and transforming it into a colorized version of the monochrome image. The output does not necessarily have to be the same as the original image – many times, there is no original colorful version of the image when we want to restore old photographs. The goal is to create a fidelitous, realistic output.

It began together with the first photographs when people tried to add colors to monochrome films and photographs by hand. Digital colorization started in the 1970s and has been widely improved since then.

Different approaches were used to achieve the best results. Some of them require human supervision like scribble-based [1, 2, 3], where the user needs to provide a few color strokes of the desired color to certain areas of the grayscale image. The algorithms then propagate these colors into the whole picture, assuming that the neighboring pixels with similar luminance should be colored in like manner.

Transfer-based colorization requires an example image, from which color information is transferred to the target grayscale image. The mapping can be established with the manual intervention [4, 5, 6] or partially automatically where reference image selection is chosen manually [7, 8].

Automatic colorization focuses on coloring grayscale images without manual intervention. These methods [9, 10, 11] are fully automatized, although trained only on datasets with smaller sizes. With the rise of deep learning, convolutional neural networks [12] and modified variational autoencoders [13] have shown favorable results in this field.

## 1.1 Color representation

Color representation is a fundamental problem in computer vision. Colors can be represented in many different ways, and color spaces give us information about the exact representation. Some of the most famous color spaces include YCrCb, YUV, RGB, HSV CIE-LAB, CIE-LUV, and many more.

### 1.1.1 RGB

RGB color space is defined by an additive color model in which the sum of color channels can express the perceived color. The RGB model is characterized by red, green, and blue light channels, together composing a vast range of colors. In the standard RGB model, the intensity of each channel can take on values in range 0-255. The full strength of every channel corresponds to black color, while zero strength represents the white color. The color information is not separated from the intensity, causing the channels to be highly correlated. Moreover, the channels are not proportional to the color difference, so just small perturbation can result in sudden variations in color and brightness. It is a device-dependent color space; different devices reproduce colors in a different manner.

### 1.1.2 CIE-Lab

Three channels also represent CIE Lab color space, but these channels denote something else than in the case of RGB. Lightness channel $L$ is defined in range 0-100, representing black at $L = 0$ and white at $L = 100$. It is kept separate from color channels, and it is equivalent to a grayscale image of the given color picture. Color information is encoded in two channels: $A$ – green to red and $B$ – blue to yellow. The color range for $A$ and $B$ channels is implementation-specific, but usually acquires values between -128 to +127. This color model is device-independent.

## 1.2 Generative Adversarial Network

The idea of a generative adversarial network is quite young; it was first introduced in [14] in 2014. It is capable of generating new data points of any kind, but it is especially popular for generating images. The architecture of GAN consists of two parts: the generator and the discriminator. It is often compared to the thief and police, where the thief (the generator) creates fake money, and the police (the discriminator) tries to expose him. The thief has to produce better and better counterfeits so that he is not exposed by the police, and the police also keeps improving its techniques to capture more thieves. In other words, they play a two-player game where both sides try to achieve equilibrium.

Initially, the generator is fed with a random noise sampled from normal or uniform distribution $z \sim p_z(z)$. It produces so-called fake output, which is similar, but not exactly the same as data points from the training set. This network learns the probability distribution of the training set and tunes its parameters $\theta_g$ to generate realistic samples.

The input to the discriminator is either a real or a fake sample. The discriminator's role is to distinguish between real and fake samples, returning 1 if it is real and 0 if not. In fact, the discriminator decides whether the sample is from $p_r$ or $p_g$ where $p_r$ is real distribution of the training dataset and $p_g$ is distribution generated by the generator. Ideally, the generator produces such realistic samples that the discriminator cannot tell whether they are real or fake, returning 0.5 probability for any input data point.

The generator tries to minimize the probability of fake sample being classified as fake by the discriminator:

$$\min_g L(D,G) = \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z;\theta_g);\theta_d))] \tag{1.1}$$

The discriminator maximizes:

$$\max_d L(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x;\theta_d)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z;\theta_g);\theta_d))] \tag{1.2}$$

where the first term $\mathbb{E}_{x\sim p_r(x)}[\log D(x)]$ is the probability of classifying real sample as real and second term $\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$ is the probability of classifying fake sample as fake.

Total loss function of GAN:

$$\min_G \max_D L(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log(D(x))] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{1.3}$$

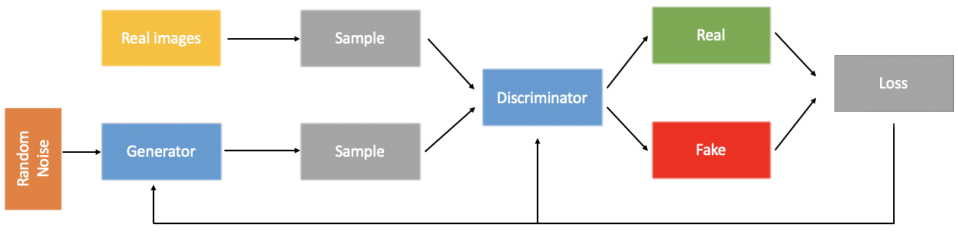


Figure 1.1: Basic GAN architecture.

Chapter 2

# State-of-the-art

Numerous machine learning techniques have been proposed to colorize images. Some of them are supervised and require a set of paired images, where one of the pictures is in grayscale, and the other is a colorized version of the same picture. But there are also methods which do not need matching images. In this chapter, I will summarize some of these techniques. In the next chapter, I will introduce two methods that I chose, in detail.

## 2.1 Image-to-Image Translation

Image-to-image translation includes problems such as reconstructing objects from the edges, creating realistic satellite photos from maps and vice versa, creating night-to-day and day-to-night photos, image inpainting, or automatic colorization of images. One approach is to create one universal method capable of solving all of these issues. The idea behind this common framework is simple: predict pixels from pixels.

Authors of this idea use a conditional generative adversarial network with the U-net encoder-decoder architecture. $L1$ distance between predicted and the original image has been added to the non-saturating objective function of conditional GAN since previous studies found this approach beneficial. No input noise is introduced to the generator because the authors have found the model ignoring the noise. The discriminator architecture uses PatchGAN.

The model is trained by alternating the training of generator and discriminator. Authors divide the loss function by 2 when training the discriminator, to avoid quick training of discriminator compared to the generator. Evaluation metric in case of colorization was qualitative, measured as plausibility to a human observer. Participants of the experiment were given a set of 40 trials where the real and fake image was presented for one second. After that, they had unlimited time to decided which of the pictures was fake.

A similar approach is used in [15]. This work is focused on automatic colorization in Japanese comics Manga. Two models were compared – a convolutional neural network ConvNet and conditional GAN. They were trained to predict how particular characters, animals, or objects should be colored since the model was trained on a specific animation. Two types of pictures were manufactured from original colorful pictures – grayscale and edge-only pictures.

The architecture of the GAN's generator is again a U-Net, consisting of 6 encoding and 6 decoding layers. Objective function with the $L1$ regularization term is used. Batch normalization [16] and LeakyReLU [17] activation function is applied after each layer. It is conditioned on a gray or edge-only image as the input and gives out an output with the same size as input but with 3 RGB color channels. No noise is introduced into the generator.

The discriminator is given a grayscale/edge-only image and either a real or generated image. It consists of 6 layers, just like the encoding part of the generator, with the sigmoid function applied on the last layer to decide whether the input is real or fake. The mini-batch size is set to 64. The model was trained on 20 epochs using Adam optimizer and learning rate 0.0002. Dataset was collected from 15 hours of Pokemon videos, where every 50 frames an image with a final size of 180x320 pixels was generated.

GAN's quality of images from the grayscale images is generally better than the images generated by the ConvNet, which are dimmer and less sharp. However, two often failures were experienced: in characters with similar appearances, GAN confused the colors of the thing in which they differ, and in large object coloring, GAN did not color the object as a whole. In a quantitative evaluation where mean absolute error and mean squared error per pixel-channel distances were measured, the results from GAN outperformed those generated by ConvNet. For both models, using the dataset with grayscale images had better results. This behavior was expected since grayscale images contain more information than edge-only ones.

## 2.2 Unsupervised Colorization

Another approach to colorization is unsupervised colorization. One study [18] proposes unsupervised diverse colorization, where conditional GAN tries to learn the real distribution of item colors. The goal is not to find an image which is as close as possible to the original one, but rather produce realistic and plausible images, that do not suffer from sepia-toned colorization.

The generator has fully convolutional architecture with stride 1, so it does

not downsize the data. Authors argue that keeping spatial information is essential to preserve item boundaries, in contrast with the encoder-decoder network, which tends to extract global features. The generator is introduced multi-layer noise on the first three layers to achieve diverse colorization. A conditional grayscale image is applied to every layer of the generator. Experimental comparison between one-layer noise and multi-layer noise, as well as the comparison between multi-layer condition and single-layer condition, is provided. The input to the discriminator is either original or generated color image. No conditional information in the form of the grayscale image is introduced to the discriminator.

The model was trained both on RGB and YUV color space to compare the results using both color representations. The authors also implemented Wasserstein GAN [19] to measure its performance. Models were trained on the TL-SUN bedroom dataset using 503900 images of size 64x64.

Multi-layer noise resulted in more color diversity in predicted images. Conditional input applied to multiple layers provided more information on the structure of the pictures; the edges of the items were preserved better than using single-layer condition. Wasserstein GAN achieved similar results to the model presented in this study; however, the training time was much longer. YUV model was more stable than the RGB one since it had to predict only two channels with a fixed $Y$ channel.

Turing test on 80 participants who were asked 20 questions was conducted to evaluate the results. Each question contained 5 images, among them one original and 4 generated. Participants were asked to determine whether any of them had disturbing quality. The results showed 62.6% positive feedback on generated images and 70% on the ground truth images. Moreover, the significance t-test showed that there is no notable difference between created and original pictures.

Chapter 3

# Implemented models

In this chapter, I will describe the two models I decided to implement in more detail. These two models are Deep Convolutional GAN and CycleGAN. I will characterize how they work, their objective function, and their architecture according to the original scientific paper that suggested them.

## 3.1 Deep Convolutional Conditional GAN

The first model is based on conditional DCGAN from [20]. Authors of this study try to create a stable model capable of automatic image colorization with good performance. The model can be described as a deep convolutional conditional GAN. It is built upon [21], where one GAN-based model is used for different image-to-image translation problems. In further text, I will refer to this model as DCGAN.

### 3.1.1 DCGAN

One of the prevalent GAN architecture is deep convolutional GAN (DCGAN). It eliminates fully connected layers, replaces all max-pooling layers with convolutional stride for downsampling, and utilizes transposed convolution (deconvolution) for upsampling. It uses Batch normalization on most of the layers, ReLU activation function in generator and LeakyReLU activation [17] in the discriminator.

### 3.1.2 Conditional GAN

In the classic GAN model, there is rather no control over generated data. When extra input information $y$ is provided to both generator and discriminator, the model is called conditional GAN. It can be any information, for example, class labels or some additional data, forcing the generator to produce a more specific output instead of a generic sample.

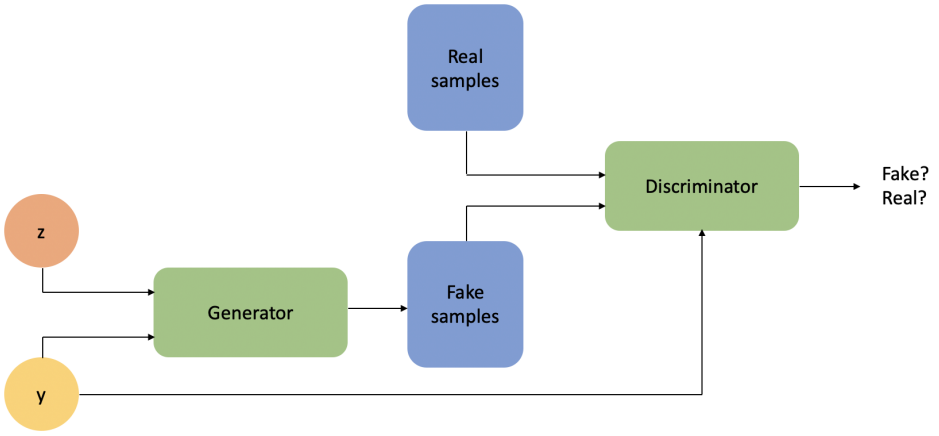


Figure 3.1: Simple conditional GAN architecture.

### 3.1.3 Objective function

In classic GAN objective function, the convergence rate is rather slow. The gradients are small, and the generator produces similar samples in the early stages of the training. A new objective function called heuristic non-saturating game, was introduced [22] to overcome these issues. It can be interpreted as the maximization of the probability of discriminator being mistaken and rewritten to minimization problem:

$$\min_G L(D,G) = -\mathbb{E}_{z\sim p_z(z)}[\log(D(G(z)))] \tag{3.1}$$

Better properties of non-saturating heuristics can be also observed from Figure 3.2, where different cost functions of the generator are compared.

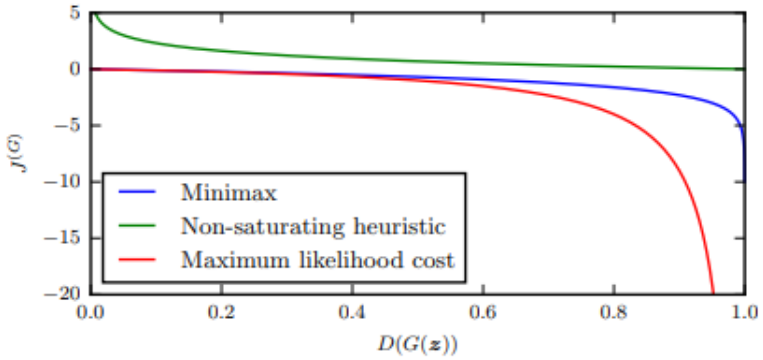


Figure 3.2: Different cost functions of the generator, image taken from [22].

Furthermore, $L1 = \lambda||G(z)-y||_1$ regularization term is added to the objective, to force the generator to produce images analogous to the ground truth. $\lambda$ is the regularization parameter set to 100, $G(z)$ is the generated image, and $y$ is the original color image. $L1$ norm was chosen rather than $L2$, which produces blurry images.

Final objective for the generator then is:

$$\min_G L(D,G) = -\mathbb{E}_{z\sim p_z(z)}[\log(D(G(z)))] + \lambda||G(z)-y||_1 \tag{3.2}$$

### 3.1.4 Network architecture

The generator architecture has a U-Net architecture [23], which is symmetric encoder-decoder based architecture. The encoder consists of $n = 7$ downsampling layers with an increasing number of filters. After that, $n$ decoding – upsampling layers are added to the network. The dimensionality reduction of the encoder shrinks the tensor shape and enables compact feature learning without large memory consumption. Skip connections are added between layer $i$ and $n - i$ to allow the flow of the low-level information in the network. One additional encoding unit without skip is inserted between the encoder and decoder.

The network is fully convolutional, without any fully connected layers. Kernel size is set to 4, stride to 2, except for the first layer, where the stride is set to 1. Batch normalization is applied on all layers except the first and last layers of the generator. It serves to avoid the mode-collapse when the generator learns to produce one output that can fool the discriminator all the time. LeakyReLU activation function is used on the encoding part, ReLU, on the decoding part. Hyperbolic tangent activation is applied on the last layer to produce output pixels between -1 and +1. The model for the dataset with the higher resolution has additional encoding and decoding block. The input noise is not given to the generator because the generator tends to ignore it. It is provided in the form of dropout on the first two decoding layers of the generator.

RGB pictures are converted into CIE-LAB space and normalized into range -1 to +1 beforehand. The $L$ channel of an image serves as the conditional input. The output of the generator are all three $L$, $A$, $B$ channels. There is an option of predicting only $A$ and $B$ channels, and the results of these two approaches I compare in Chapter 6.

The discriminator architecture consists of 5 encoding layers with kernel size set to 4 and stride set to 2 and 1 in the last two layers. Batch normalization

is applied, except for the first and last layer, and LeakyReLU with slope 0.2 is the activation function on all layers except the last one. The input to the discriminator is either real or fake, together with the same picture's grayscale equivalent. For both networks, Adam optimizer with the learning rate set 0.0002 is utilized.

### 3.1.5 Evaluation metrics

Two quantities were chosen to evaluate the experimental results:

- mean absolute error for every color channel for every pixel between original and generated image:

$$MAE(x, y) = \frac{1}{c * n} \sum_{p=1}^{n} \sum_{l=1}^{c} |h(x)^{(p,l)} - y^{(p,l)}| \tag{3.3}$$

  where $x$ is input grayscale image, $y$ is corresponding color image, $p$ is the index of pixel, $l$ is the index of a color channel, $n$ is the number of pixels of an image, $c$ is the number of color channels and $h$ is image translation function.

- accuracy, which "*is measured by the ratio between the number of pixels that have the same color information as the source and the total number of pixels. Any two pixels are considered to have the same color if their underlying color channels lie within some threshold distance* $\epsilon$" [20]:

$$acc(x,y) = \frac{1}{n} \sum_{p=1}^{n} \prod_{l=1}^{3} \mathbf{1}_{[0,\varepsilon_l]}(|h(x)^{(p,l)} - y^{(p,l)}|) \tag{3.4}$$

## 3.2 CycleGAN

The goal of image-to-image translation is to learn the mapping between input and output images. In a standard way, it requires paired images for the training procedure. The cost of getting paired images is often very high, and the images are challenging to obtain or do not exist at all. The authors of the study [24] suggest a method that is suitable for unpaired image-to-image translation. They do not use it for image colorization specifically, but among other experiments, they try to apply it to edge-to-image colorization, which can be classified as a similar problem. This method is able to use a collection of pictures from one domain, learn the style of the collection, and translate it to another domain.

Instead of paired images, the algorithm is fed with two datasets from different domains $X$ and $Y$. The goal is to learn the mapping $G : X \rightarrow Y$, such that the output $\hat{y} = G(x)$, $x \in X$, is indistinguishable from images $y \in Y$.

However, the classic adversarial loss itself does not have stable results, so cycle loss is introduced. Given another translation $F : Y \rightarrow X$, cycle loss forces $F(G(x)) \approx x$ and $G(F(y)) \approx y$. These two mappings are done using two generators:

- generator $G$ which translates images from domain $X$ to domain $Y$
- generator $F$, which translates images from domain $Y$ to domain $X$

The model also contains two discriminators:

- discriminator $D_y$ - distinguishing $G(x)$ from domain $Y$
- discriminator $D_x$ - distinguishing $F(y)$ from domain $X$

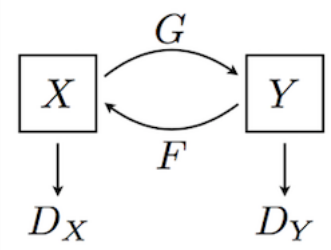


Figure 3.3: Basic structure of the CycleGAN [24].

### 3.2.1 Objective function

The total objective function consist of two parts:

- Adversarial loss
- Cycle loss

Adversarial loss is standard GAN loss function:

$$L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log(D_Y(y))] + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x)))] \tag{3.5}$$

The cycle loss consists of two parts: forward and backward cycle consistency. Forward cycle consistency encourages images $x$ from domain $X$ to be translated back to the original image in the cycle: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. The pictures should also satisfy backward cycle consistency: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$.

Total cycle consistency:

$$L_{CYC}(G,F) = \mathbb{E}_{x\sim p_{data}(x)}||F(G(x)) - x||_1 + \mathbb{E}_{y\sim p_{data}(y)}||G(F(y)) - y||_1 \quad (3.6)$$

The full objective function is then:

$$L(G,F,D_X,D_Y) = L_{GAN}(G,D_Y,X,Y) + L_{GAN}(F,D_X,Y,X) + \lambda L_{CYC}(G,F) \quad (3.7)$$

where $\lambda$ describes relative importance and it was set to 10.

In addition, identity loss is suggested to encourage training stability and the quality of images. The generator is given real pictures from the target domain. It should produce the same pictures without change. However, this cannot be applied for image colorization because of the different number of input channels to each generator.

### 3.2.2 Network architecture

Initially, the architecture for 128x128 pictures consists of convolution with kernel size 7, stride 1, and 64 number of filters. After that, two layers with kernel size 3, stride 1, and doubling number of filters follow. Then six residual blocks are applied. Two layers with kernel size 3, and stride 2 follow, halving the number of filters. Lastly, there is a convolution with kernel size 7 and stride 1, outputting 3 channels. Instance normalization and ReLU activation are applied after every standard convolution.

The residual block [25] contains convolution with kernel size set to 3, instance normalization, ReLU activation, followed by another convolution and instance normalization layer. The number of filters remains the same. The residual block output is concatenated with the input to the first layer of the residual block.

The discriminator consists of 4 convolutional layers, kernel size 4, stride 2, and doubling number of filters after every convolution. Leaky ReLU is applied after every layer, and instance normalization is applied on every layer except the last one. After the last layer, convolution is used to produce one-dimensional output.

The discriminator is a PatchGAN that, instead of classifying the whole image, classifies each 70 x 70 patch as real or fake. It returns the probability of the portion of the image being either real or fake. After running convolutionally across the whole picture, all responses are averaged to form the final decision. It can be applied convolutionally on images of any size, even on the larger

images than those on which it was trained. It requires fewer parameters, causing the algorithm to run faster.

### 3.2.3 Evaluation metrics

Perceptual study on Amazon Mechanical Turk served as the primary evaluation metric. Participants were shown one real and one fake image, and they were supposed to click on the one, they considered as authentic. The fool rate of the pictures created by the algorithm was counted during this evaluation. The second evaluation metric was the FCN score, which uses a fully convolutional network. It completes semantic segmentation and creates label maps. Afterward, the comparison of label maps between real and fake images is made.

Chapter 4

# Implementation

In this chapter, I will specify the technologies I used for the implementation. I will describe how I implemented the models, what changes I needed to make to suit my experiments, how I tuned the hyperparameters, and which training strategies I employed.

## 4.1 Technologies

I decided to pick Python for the implementation of the practical part. It was the first choice, and I did not even think about other possibilities. It offers vast library support for machine learning, and I was familiar with it from other school projects.

For image processing, I used scikit-image[1], an open-source Python package. It was useful for conversion between color spaces without writing the equations manually. Nevertheless, when I needed to do some numerical calculations, I used NumPy[2] library.

Another library I picked is Matplotlib[3]. I adopted it for visualization of the images and for plotting the training loss and evaluation metrics. Later, I decided to benefit from Tensorboard[4], which is Tensorflow's visualization toolkit for machine learning which is more interactive and offers better options for monitoring the training procedure.

Tensorflow 2.0[5] is an open-source platform the provides an API, which is excellent for numerical computing and working with tensors. It is one of the

[1] https://scikit-image.org/
[2] https://numpy.org/
[3] https://matplotlib.org/
[4] https://www.tensorflow.org/tensorboard
[5] https://www.tensorflow.org/

most famous libraries used for deep neural networks. Moreover, it provides a backend for Keras, which is a high-level Python-based framework built on the top of Tensorflow. It is designed for working with neural networks even more efficiently. It eliminates low-level computations, so the user does not have to deal with tensor algebra or optimization techniques.

Jupyter Notebook[6], an open-source web application has been, among other things, widely used for machine learning and data visualization. I used Google Colab[7], which is a free Jupyter notebook environment. It runs in the cloud and requires no setup, so it is straightforward to start working immediately. Moreover, it provides access to GPU units, that speed up the training process to a manageable period. Although it has some limitations on maximum RAM and running time without interruption, it was a convenient tool for this project.

## 4.2 DCGAN details and specifics

Since I worked with two datasets containing pictures of different sizes, I needed to create a slightly different neural network for every dataset.

### 4.2.1 DCGAN for CIFAR-10 dataset

I focused on the CIFAR-10 dataset, for which I created three different models that differ in the way I work with colors. These three models are:

- DCGAN AB
- DCGAN LAB
- DCGAN RGB

Although the baseline paper for my first model converts RGB to CIE LAB, it does not take advantage of predicting only two channels. During my research, I came across this idea, which is quite logical, since the model does not have to learn to predict all three channels, but instead predicts two and concatenates them with the grayscale image. The model has to learn fewer parameters, and most importantly, the structure of the image is preserved better. During the early training stages, the output resembles the original picture much more than using any other model. In all three models, the data were first processed into a given color space, and each channel was normalized to the range -1 to +1.

[6] `https://jupyter.org/`
[7] `https://colab.research.google.com/`

#### 4.2.1.1 DCGAN AB/LAB

These two models model work with CIE LAB color space. I used a function from the scikit-image library for conversion. The discriminator's architecture is the same as already described in Chapter 3 Section 3.1. The architecture of the generator is also similar, but instead of 7 encoding and decoding layers, I used only 4. I established this decision based on the original implementation of the paper that suggested this option. The three layers I removed did not bring any improvement in results, just caused a little longer training time. In the AB model, the output of the generator is only two channels, $A$ and $B$. The LAB network has the same architecture as the AB, but instead of 2 output channels, it generates all 3 $L$, $A$, and $B$ channels.

#### 4.2.1.2 DCGAN RGB

The architecture is exactly the same as in the DCGAN LAB model. I used this equation:

$$Grayscale = 0.299 * R + 0.587 * G + 0.114 * B \tag{4.1}$$

to convert a color image to the grayscale in RGB. It is based on human perception, it decreases the contribution of red color, increases the participation of the green color, and puts blue color contribution between these two, to create a grayscale image.

However, the RGB color space is not well suited for this problem because of its instability. The learning is much slower, it requires longer training time, and the results are slightly worse.

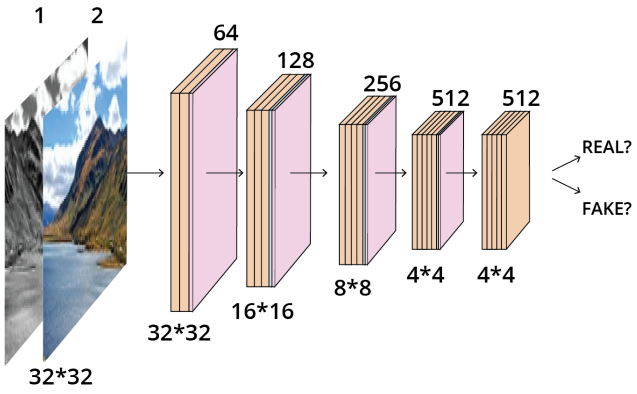


Figure 4.1: Architecture of the discriminator, DCGAN AB model.

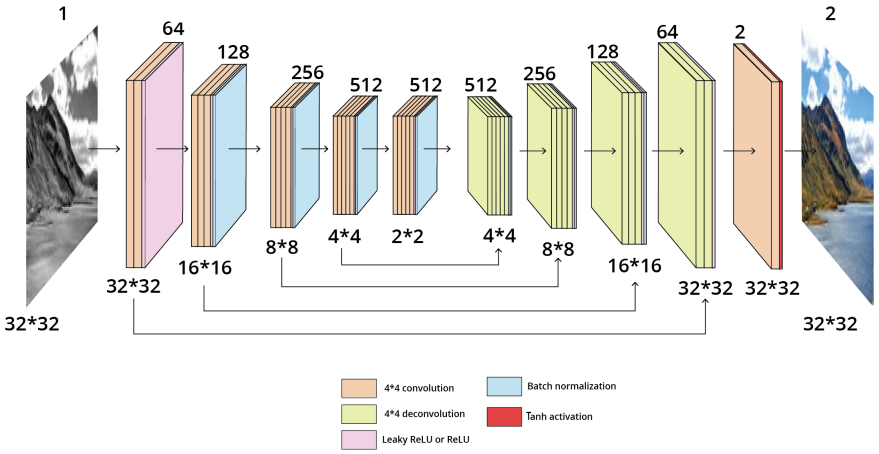


Figure 4.2: Architecture of the generator, DCGAN AB model.

### 4.2.2 Training strategies

The original number of epochs $e = 200$ showed up as the acceptable number of epochs. Longer training time did not bring much improvement. I kept the model running for 500 epochs as well, to observe the convergence of the model.

I used many implementation tips to stabilize training:

- scaling of the pictures to values between -1 and 1
- tanh as the output layer of the generator
- batch normalization
- weight initialization from the normal distribution
- convolutional stride instead of max-pooling layers
- Adam optimizer that is preferred optimizer for GAN training

The last thing that proved to be handy is label-smoothing. GAN discriminators tend to become overconfident and use only a few features to detect real or fake images. Smoothing the target label from 1 to 0.9 to penalize overconfident values greater than 0.9 enhances the training procedure.

It was quite challenging to observe the direct impact of changes of some hyperparameters in a reasonable amount of time. The best option emerged in keeping the suggested: Adam optimizer with the learning rate of 0.0002 and $\beta = 0.5$. The $\lambda$ coefficient, which emphasizes the relative importance of the

L1 part of the objective function, I set to 100, as suggested. I used mini-batch mode, with the batch size set to 128.

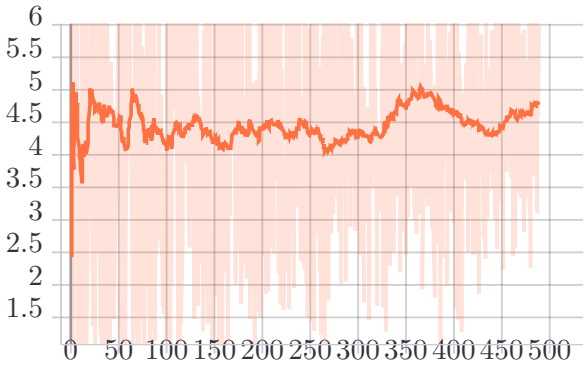


Figure 4.3: Generator adversarial loss over 500 epochs on CIFAR-10 dataset, DCGAN AB model

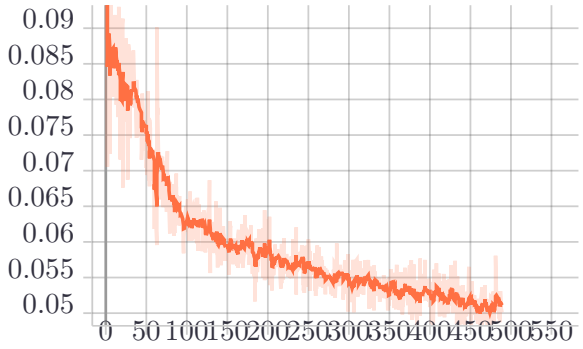


Figure 4.4: L1 loss over 500 epochs on CIFAR-10, DCGAN AB model

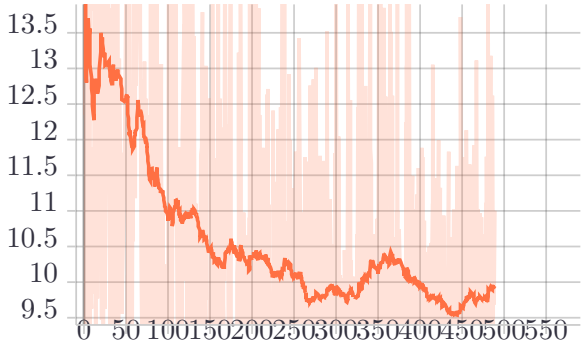


Figure 4.5: Total generator loss over 500 epochs on CIFAR-10 dataset, DCGAN AB model

### 4.2.3 DCGAN for Intel Image Classification dataset

Since the DCGAN AB model showed the best results on the CIFAR-10 dataset, I adopted a similar model for the second dataset. I kept the same discriminator and added two more encoding-decoding layers to the generator. I needed to decrease the number of epochs due to a higher number of trainable parameters in this network. I trained the model on 50 epochs, with a mini-batch size of the size of 16. I used the same training strategies and hyperparameters as before.

## 4.3 CycleGAN details and specifics

The CycleGAN model is implemented as two models. One converting from domain $X$ to domain $Y$ and the other from $Y$ to $X$. Firstly, the generator $G$ translates images from domain $X$ to domain $Y$, and the adversarial loss is counted. After that, the fake image is put into generator $F$, and forward cycle loss for is counted. Finally, the real picture from domain $Y$ is fed into the generator $F$ and then to $G$ to calculate the backward loss. These loss functions are then used to update both generators and discriminators. This is performed vice versa as well.

The dataset is unpaired, meaning the subset of images from domain $X$ are randomly chosen images from the whole dataset and converted to grayscale. Domain $Y$ is a random subset of color pictures taken from the same dataset. I use RGB conversion to grayscale.

### 4.3.1 CycleGAN for CIFAR-10 dataset

I experimented with the network's architecture, with the number of residual blocks, the number of layers and their characteristics (kernel size, stride), learning rate, the relative weight of loss function, and mini-batch size but even original architecture was not able to produce successful results on the CIFAR-10 dataset. Longer training time also did not bring meaningful improvement. I tried to reduce the size of the dataset or train the model only on images from one category but without any good outcome.

When I saw that the network was training several times longer than the DCGAN model and the results were still quite unsatisfactory, I decided to stop the execution early even if the generator loss was still slowly decreasing. I could not find a proper model, so I focused on the other dataset. I will show some examples of collapsing images in Chapter 6.

### 4.3.2 CycleGAN for Intel Image Classification dataset

Since the training time was much higher than in the other model, I could not use the whole dataset. I picked two categories: street and forest and created two separate models. The architecture of the generator and the discriminator remains the same as the original architecture I already described.

### 4.3.3 Training strategies

I used Adam optimizer with the learning rate of 0.0002 and $\beta = 0.5$, as suggested. The weight loss contribution of the discriminators is set to 0.5 to prevent the discriminators from learning too fast, giving no space for the generators to develop. I used the batch mode when the network updates after every single training point. I trained the model on 40 epochs for the Intel dataset.

Instead of feeding the discriminator with currently generated images, the model keeps a buffer that stores 50 previously created images. If the pool is full, there is a 50% probability that the current generated image will replace one of the pictures from the pool. The images given to the discriminator are then randomly chosen from this pool.

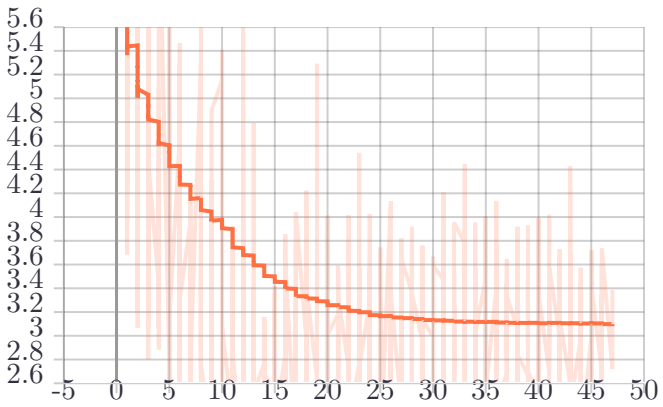


Figure 4.6: Generator loss over 40 epochs, CycleGAN model, Intel Image dataset. The loss of the second generator looks very similarly.

Chapter 5

# Experiments

In this chapter, I will introduce the datasets I picked and describe performed experiments.

## 5.1 Datasets

I used two different datasets on which I evaluated the models. I did not employ the whole dataset in some experiments, but I picked a smaller portion if needed.

The first one is a well-known CIFAR-10[8] dataset that has been widely utilized for deep learning evolving images. It contains 32x32 color images, split into 10 categories (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). The whole dataset consists of 50 000 training images and 10 000 test images.

Since CIFAR-10 has quite low resolution, for better visual classification, I chose publicly available dataset of natural scenes that was created for Intel Image Classification[9]. It contains 150x150 colorful images distributed under 6 categories (buildings, forest, glacier, mountain, sea, street). I resize the data to 128x128.

## 5.2 Design of experiments

I created a few models already described in the previous chapter. I ran each of them separately. For evaluation, I used suggested mean absolute error and custom accuracy with $\varepsilon = 2$ and $\varepsilon = 5$. I calculated these metrics on training and test dataset.

[8] `https://www.cs.toronto.edu/~kriz/cifar.html`
[9] `https://www.kaggle.com/puneet6060/intel-image-classification`

Figure 5.1: Example pictures from CIFAR-10 dataset.

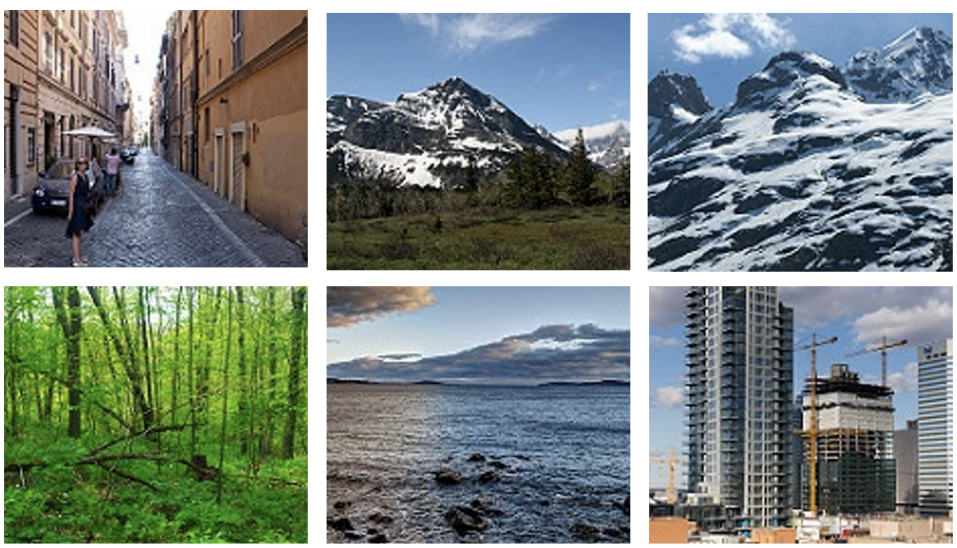

Figure 5.2: Example pictures from Intel Image Classification dataset.

Chapter 6

# Results and evaluation

In this chapter, I will evaluate the performance of my models quantitatively, I will provide some example images to visually compare the images, and I will discuss the results.

## 6.1 DCGAN

As the evaluation metrics, I used the suggested accuracy and mean absolute error. The original paper provides results only on the training set, but I evaluated the model on the test dataset as well. Quantitative results are shown in the Table 6.1 and 6.2. In Figure 6.1, some of the wrongly colorized images are displayed. Figure 6.5 shows example images that were colorized credibly. Figures 6.2, 6.3 and 6.4 show plots of evaluation metrics during the training procedure.

Visually, it can be observed that the models performed quite well. A slight problem with miscolorization occurred; for example, the models tend to put more green into the background, confusing it with the grass. This error was expected; it is a common mistake in image colorization, and authors of other studies experienced it as well. They believe this is caused by the considerable amount of images containing grass. The second issue happens when the picture is colored unusually and involves colors that emerge only rarely in the dataset images. In that case, the picture is painted with more natural colors.

Since the two models, AB and LAB, had similar performance, but AB variant performed slightly better on the test dataset, I decided to implement AB variant for the other dataset. The quantitative results on training and test dataset can be seen in the Table 6.3.

Qualitatively, in my subjective opinion, the images look very realistic. Generally, the colors are bright, not suffering from the hue. Again, the model cannot

capture some specific situations that are uncommon; for example, when the water in the picture is brown, the model colors it in blue. In some cases, when the original image is not very colorful, the model tends to colorize it more, so the generated images look even better than the original ones. Some of the images are shown in Figure 6.6 and Figure 6.7.

| | **Original** | | | **Train** | | |
|---|---|---|---|---|---|---|
| **Dataset** | **MAE** | $\varepsilon = 5$ | $\varepsilon = 2$ | **MAE** | $\varepsilon = 5$ | $\varepsilon = 2$ |
| CIFAR AB | - | - | - | 5.8 | 62.5% | 24.0% |
| CIFAR LAB | 5.1 | 65.5% | 24.1% | 5.4 | 70.2% | 27.2% |
| CIFAR RGB | - | - | - | 7.7 | 63.2% | 20.2% |

Table 6.1: MAE and accuracy of DCGAN models measured during training on CIFAR-10 dataset on 200 epochs. Comparison to benchmark results.

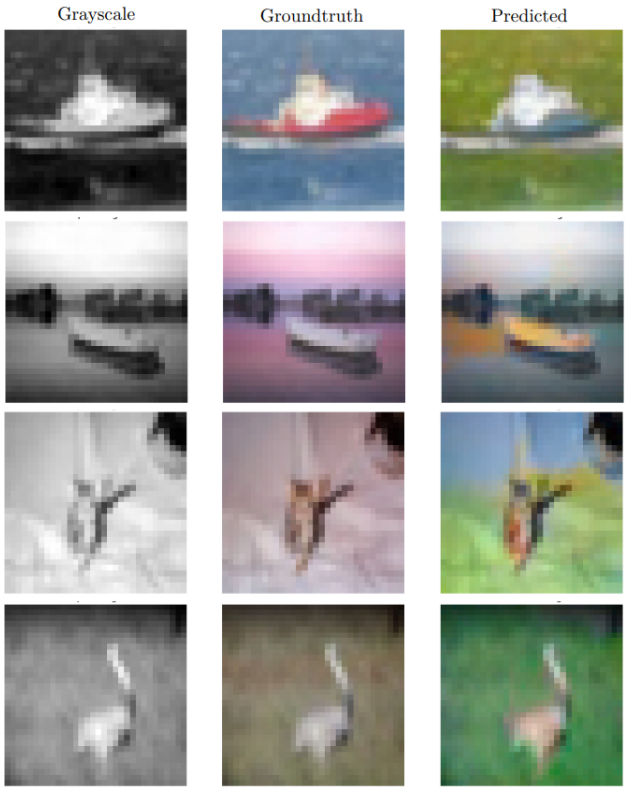


Figure 6.1: Miscolorized pictures from CIFAR-10 dataset, colorized by DCGAN AB model over 200 epochs.

| | Test | | |
|---|---|---|---|
| **Dataset** | **MAE** | $\varepsilon = 5$ | $\varepsilon = 2$ |
| CIFAR AB | 7.0 | 51.7% | 19.2% |
| CIFAR LAB | 7.9 | 47% | 13.3% |
| CIFAR RGB | 13.6 | 33.6% | 9.6% |

Table 6.2: MAE and accuracy of DCGAN models measured on CIFAR-10 test dataset. The test dataset consisted of 10 000 images.

| | Train | | | Test | | |
|---|---|---|---|---|---|---|
| **Dataset** | **MAE** | $\varepsilon = 5$ | $\varepsilon = 2$ | **MAE** | $\varepsilon = 5$ | $\varepsilon = 2$ |
| INTEL | 5.4 | 64.3% | 24.7% | 5.7 | 61.1% | 25.7% |

Table 6.3: MAE and accuracy of AB DCGAN model measured on Intel Image Classification dataset on 40 epochs.

## 6.2 CycleGAN

From the evaluation plots, Figure 6.8, Figure 6.9 and Figure 6.10 it can be observed that the accuracy is much lower, and the mean absolute error much higher than in the previous model. For example, in DCGAN, the accuracy $\varepsilon = 5$ was reaching more than 75% at the end, while here, the maximum it reaches is less than the 8%.

Visually, the images can be judged as credible in most cases, but they suffer from more errors and mislocorization than before. In later stages, after the mean absolute error and accuracy hit their peak (about epoch 17), I would say that the model tends to make the images too colorful. Example images are provided in Figure 6.11 and Figure 6.12.

As I already mentioned, I could not create a suitable CycleGAN model for the CIFAR-10 dataset. It would be meaningless to evaluate the model in any way, while it can be seen that the images are not realistic nor plausible. The model learned to create pictures in something like a negative tone – it colors the lightest areas as dark and vice versa. However, it can still perform a good translation the other way – from color to grayscale, even though the fake colorized image does not look well. Pictures are shown in Figure 6.13

## 6.3 Comparison

If I should compare both models, the results from the CycleGAN cannot be even compared to those from DCGAN. In my opinion, it is because of the fact that the first model was designed exactly for image colorization, while in the case of the second model, there is a broader range of different image translations it can be applied to.

The training time of the CycleGAN is much longer, resulting in worse results. Moreover, the CycleGAN has to be trained on a smaller dataset to accomplish the training in a reasonable period. The results of the CycleGAN are better if trained only on one category of images, which comes from the fact, that input to the original CycleGAN contained images with related features (day photos, night photos, zebras, horses, and so). However, this creates a model that is narrowly specified and cannot generalize well.

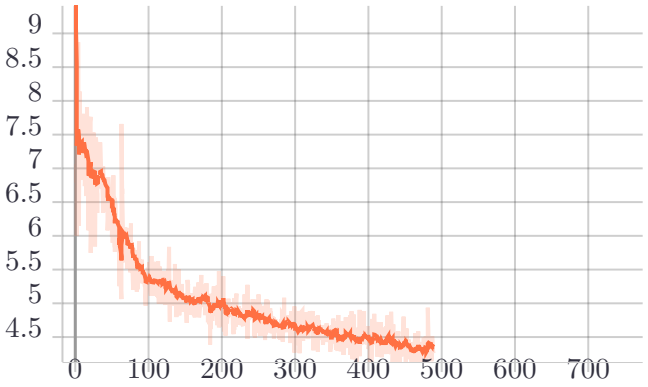


Figure 6.2: Mean absolute error during training, DCGAN AB model on CIFAR-10 dataset on 500 epochs.

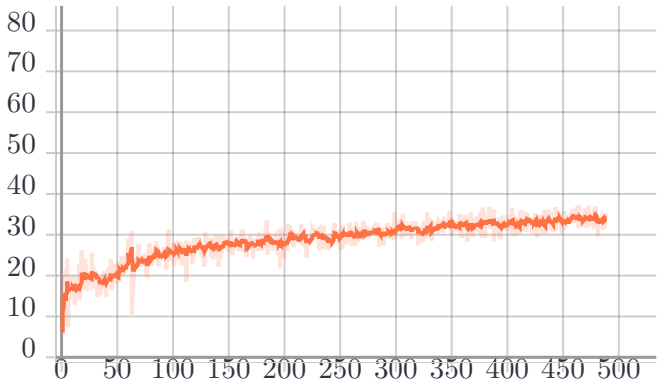


Figure 6.3: Accuracy $\varepsilon = 2$ during training, DCGAN AB model on CIFAR-10 dataset on 500 epochs.

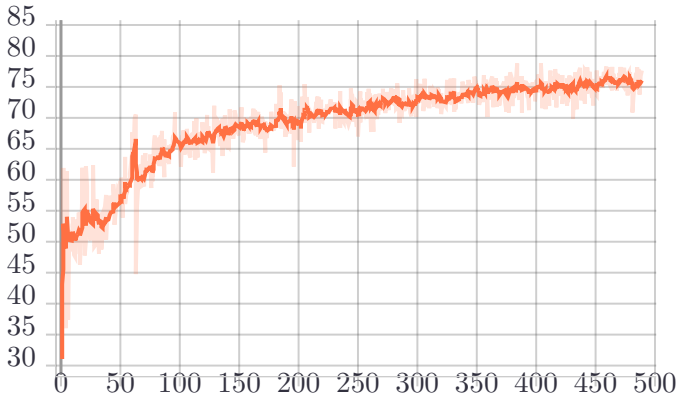


Figure 6.4: Accuracy $\varepsilon = 5$ during training, DCGAN AB model on CIFAR-10 dataset on 500 epochs.


Figure 6.5: CIFAR-10 dataset, colorized by DCGAN AB model on 200 epochs.

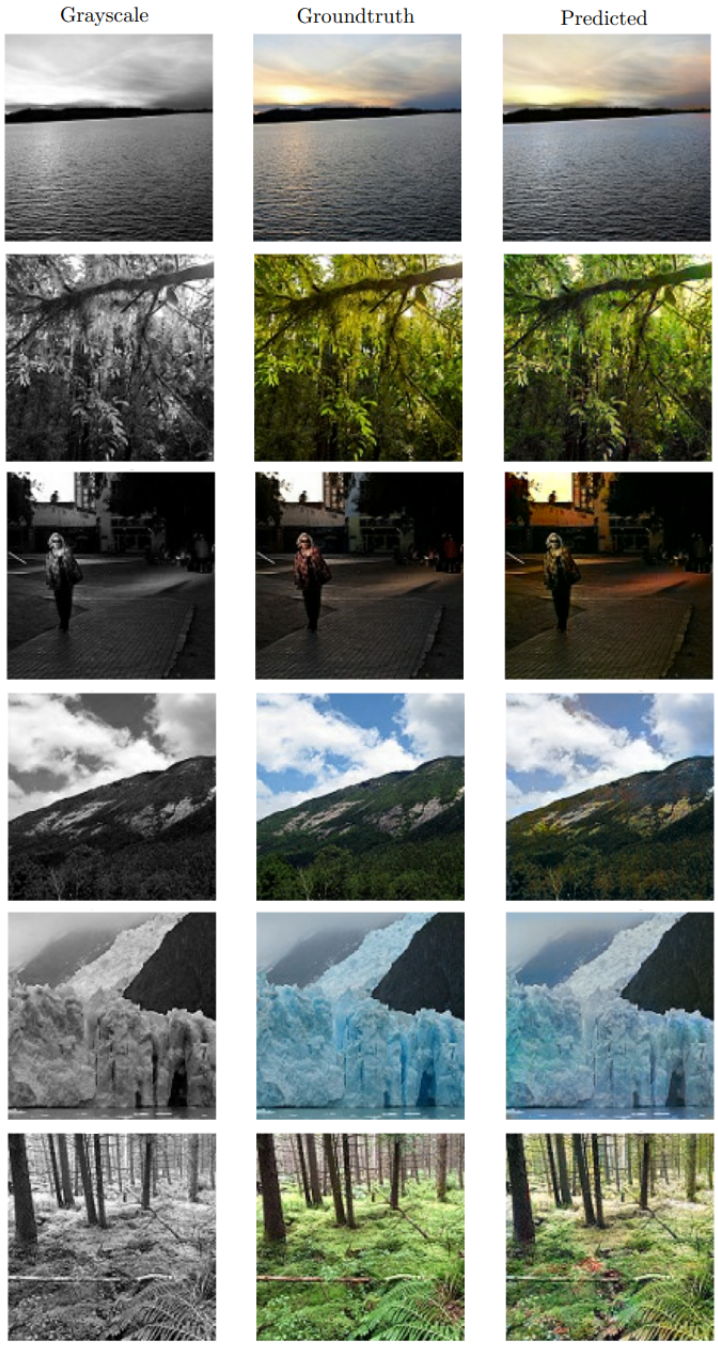


Figure 6.6: Example pictures from Intel Image Image Classification dataset, colorized by DCGAN AB model on 40 epochs.


Figure 6.7: Miscolorized images from Intel dataset, colorized by DCGAN AB model on 40 epochs

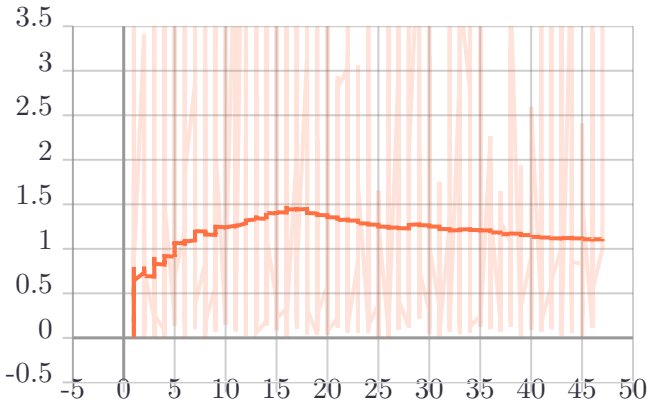


Figure 6.8: Accuracy $\varepsilon = 2$ during training CycleGAN model on Intel dataset over 40 epochs.

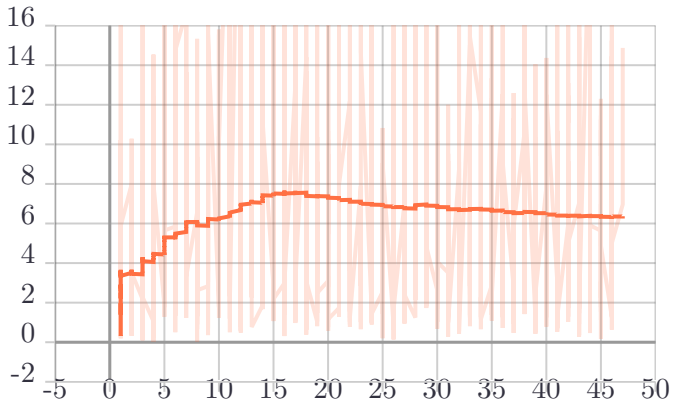


Figure 6.9: Accuracy $\varepsilon = 5$ during training CycleGAN model on Intel dataset over 40 epochs.

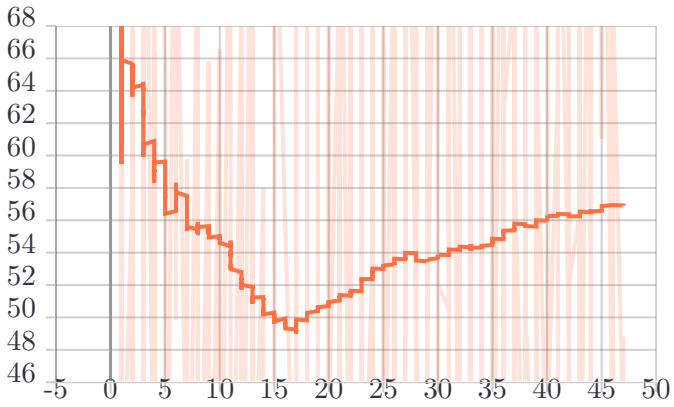


Figure 6.10: Mean absolute error during training CycleGAN model on Intel dataset over 40 epochs.


Figure 6.11: Example pictures from Intel Image Image Classification dataset, colorized by CycleGAN on 40 epochs.


Figure 6.12: Example pictures from Intel Image Image Classification dataset, colorized by CycleGAN on 40 epochs.


Figure 6.13: Example pictures from CIFAR-10 dataset, colorized by CycleGAN.

# Conclusion

Firstly, I presented the automatic image colorization problem and reviewed the state-of-the-art methods. Based on the survey, I picked two different methods – DCGAN and CycleGAN. I theoretically described how they work. After that, I mentioned some of the implementation details, changes I needed to make, training strategies, and technologies used. Based on the research, I picked two appropriate publicly available datasets – the CIFAR-10 and Intel Image Classification dataset, and I described them. Finally, I evaluated the results using the surveyed metrics – custom accuracy and mean absolute error. Moreover, I compared how color space influences the training procedure. I fulfilled the objectives for this work, and some extended or more detailed work can be built upon this project.

Using the first model, I achieved similar or, in some cases, even slightly better results than the original paper. Using the CycleGAN, the results could not be compared to the original article, since it does not work exactly with image colorization. However, I can compare the two models I executed, clearly declaring the first model as faster, more suitable, and more accurate for automatic image colorization.

## Future work

One of the most significant unresolved possibilities that can be conducted in the future is retraining the DCGAN model on pictures with a bigger size. For much larger images, some changes to the network will be necessary as well. The biggest challenge remains one universal project capable of colorizing images of any size and from any domains.

I would suggest training smaller GAN models similar to DCGAN. I would train each one on the dataset with few classes, so each model becomes confident in colorizing pictures from its domain. When the input image is given, image

recognition will be applied at first, and then the correct model will colorize the photo according to the picture parameters. I have not tried anything similar since it is far beyond the requirements for this work, but it might be a nice experiment.

# Bibliography


1. LEVIN, Anat; LISCHINSKI, Dani; WEISS, Yair. Colorization using optimization. In: *ACM SIGGRAPH 2004 Papers*. 2004, pp. 689–694.
2. YATZIV, Liron; SAPIRO, Guillermo. Fast image and video colorization using chrominance blending. *IEEE transactions on image processing*. 2006, vol. 15, no. 5, pp. 1120–1129.
3. MUSIALSKI, Przemyslaw; CUI, Ming; YE, Jieping; RAZDAN, Anshuman; WONKA, Peter. A framework for interactive image color editing. *The Visual Computer*. 2013, vol. 29, no. 11, pp. 1173–1186.
4. IRONI, Revital; COHEN-OR, Daniel; LISCHINSKI, Dani. Colorization by Example. In: *Rendering Techniques*. 2005, pp. 201–210.
5. GUPTA, Raj Kumar; CHIA, Alex Yong-Sang; RAJAN, Deepu; NG, Ee Sin; ZHIYONG, Huang. Image colorization using similar images. In: *Proceedings of the 20th ACM international conference on Multimedia*. 2012, pp. 369–378.
6. CHIA, Alex Yong-Sang; ZHUO, Shaojie; GUPTA, Raj Kumar; TAI, Yu-Wing; CHO, Siu-Yeung; TAN, Ping; LIN, Stephen. Semantic colorization with internet images. *ACM Transactions on Graphics (TOG)*. 2011, vol. 30, no. 6, pp. 1–8.
7. CHARPIAT, Guillaume; HOFMANN, Matthias; SCHÖLKOPF, Bernhard. Automatic image colorization via multimodal predictions. In: *European conference on computer vision*. 2008, pp. 126–139.
8. WELSH, Tomihisa; ASHIKHMIN, Michael; MUELLER, Klaus. Transferring color to greyscale images. In: *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 2002, pp. 277–280.
9. MORIMOTO, Yuji; TAGUCHI, Yuichi; NAEMURA, Takeshi. Automatic colorization of grayscale images using multiple images on the web. In: *SIGGRAPH 2009: Talks*. 2009, pp. 1–1.

10. CHENG, Zezhou; YANG, Qingxiong; SHENG, Bin. Deep colorization. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 415–423.

11. DESHPANDE, Aditya; ROCK, Jason; FORSYTH, David. Learning large-scale automatic image colorization. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 567–575.

12. ZHANG, Richard; ISOLA, Phillip; EFROS, Alexei A. Colorful image colorization. In: *European conference on computer vision*. 2016, pp. 649–666.

13. DESHPANDE, Aditya; LU, Jiajun; YEH, Mao-Chuang; JIN CHONG, Min; FORSYTH, David. Learning diverse image colorization. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6837–6845.

14. GOODFELLOW, Ian; POUGET-ABADIE, Jean; MIRZA, Mehdi; XU, Bing; WARDE-FARLEY, David; OZAIR, Sherjil; COURVILLE, Aaron; BENGIO, Yoshua. Generative adversarial nets. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

15. FU, Qiwen; HSU, Wei-Ting; YANG, Mu-Heng. Colorization using convnet and gan. In: *Stanford University*. 2017, pp. 1–8.

16. IOFFE, Sergey; SZEGEDY, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*. 2015.

17. MAAS, Andrew L; HANNUN, Awni Y; NG, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In: *Proc. icml*. 2013, vol. 30, p. 3. No. 1.

18. CAO, Yun; ZHOU, Zhiming; ZHANG, Weinan; YU, Yong. Unsupervised diverse colorization via generative adversarial networks. In: *Joint European conference on machine learning and knowledge discovery in databases*. 2017, pp. 151–166.

19. ARJOVSKY, Martin; CHINTALA, Soumith; BOTTOU, Léon. Wasserstein gan. *arXiv preprint arXiv:1701.07875*. 2017.

20. NAZERI, Kamyar; NG, Eric; EBRAHIMI, Mehran. Image colorization using generative adversarial networks. In: *International conference on articulated motion and deformable objects*. 2018, pp. 85–94.

21. ISOLA, Phillip; ZHU, Jun-Yan; ZHOU, Tinghui; EFROS, Alexei A. Image-to-image translation with conditional adversarial networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

22. GOODFELLOW, Ian. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*. 2016.

23. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. U-net: Convolutional networks for biomedical image segmentation. In: *International Conference on Medical image computing and computer-assisted intervention*. 2015, pp. 234–241.

24. ZHU, Jun-Yan; PARK, Taesung; ISOLA, Phillip; EFROS, Alexei A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2223–2232.

25. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

Appendix A

# Acronyms

**AB** Channel A and Channel B, from CIE LAB color space

**CNN** Convolutional Neural Network

**CycleGAN** Cycle Generative Adversarial Network

**DCGAN** Deep Convolutional Generative Adversarial Network

**GAN** Generative Adversarial Network

**LAB** Lightness, Channel A and Channel B, from CIE LAB color space

**MAE** Mean Absolute Error

**ReLU** Rectified Linear Unit

**RGB** Red Green Blue

Appendix B

# Contents of enclosed SD card

```
readme.txt ..... the file with contents description
src ..... the directory with source codes
    training ..... the directory with training files
    evaluation ..... the directory with evaluating files
text ..... the thesis text directory
    thesis.pdf ..... the thesis text in PDF format
    assignment.pdf ..... the assignment in PDF format
```