2022

ZAKLJUČNA NALOGA (FINAL PROJECT PAPER)

DUKOVSKA

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA
(FINAL PROJECT PAPER)

PRESLIKAVE MED SLIKAMI Z UPORABO GENERATIVNIH NASPROTNIŠKIH MREŽ (IMAGE-TO-IMAGE TRANSLATION THROUGH GENERATIVE ADVERSARIAL NETWORKS)

IVANA DUKOVSKA

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga
(Final project paper)

Preslikave med slikami z uporabo generativnih nasprotniških mrež

(Image-to-Image Translation through Generative Adversarial Networks)

Ime in priimek: Ivana Dukovska
Študijski program: Računalništvo in informatika
Mentor: izr. prof. dr. Klen Čopič Pucihar
Somentor: asist. dr. Domen Šoberl

Koper, september 2022

Ključna dokumentacijska informacija

Ime in PRIIMEK: Ivana DUKOVSKA

Naslov zaključne naloge:

Preslikave med slikami z uporabo generativnih nasprotniških mrež

Kraj: Koper

Leto: 2022

Število listov: 38	Število slik: 18
Število tabel: 2	Število referenc: 23

Mentor: izr. prof. dr. Klen Čopič Pucihar

Somentor: asist. dr. Domen Šoberl

Ključne besede: globoko učenje, globoke nevronske mreže, konvolucija, konvolucijske nevronske mreže, generativne nasprotniške nevronske mreže

Izvleček:
V tej diplomski nalogi izvedemo temeljito študijo algoritma CycleGAN, ki se lahko nauči pretvarjati med dvema domenama slik, pri čemer učenje poteka na neurejenih parih slik. Najprej opišemo vse teoretične predpogoje, vključno z arhitekturami nevronskih mrež, konvolucijo slik, konvolucijske nevronske mreže in generativne nasprotniške mreže, ki so potrebni, da lahko razumemo, kako algoritem CycleGAN deluje. Pri tem predstavimo zanimive primere za pet vrst takšnih pretvorb slik — prenos sloga, preoblikovanje objekta, preoblikovanje letnih časov, generiranje fotografij iz slik ter izboljšave fotografij. Nato analiziramo uradno TensorFlow implementacijo algoritma CycleGAN, napisano v programskem jeziku Python in jo primerjamo s predlagano implementacijo v izvirnem članku. Algoritem testiramo na naboru 1068 slik konj in 1355 slik zeber, z namenom usposobiti sistem za pretvarjanje podob konjev v podobe zeber in obratno. Pri tem izpostavimo razlike med izvorno implementacijo ter TensorFlow implementacijo.

Key document information

Name and SURNAME: Ivana DUKOVSKA

Title of the final project paper:

Image-to-Image Translation through Generative Adversarial Networks

Place: Koper

Year: 2022

Number of pages: 38	Number of figures: 18
Number of tables: 2	Number of references: 23

Mentor: Assoc. Prof. Klen Čopič Pucihar, PhD

Co-Mentor: Assist. Domen Šoberl, PhD

Keywords: deep learning, deep neural networks, convolution, convolutional neural networks, generative adversarial networks

Abstract:
In this thesis we conduct a study of the state-of-the-art algorithm called CycleGAN, which is capable of learning a translation between two image domains, where learning is done on unpaired sets of images. We outline all the theoretical prerequisites, including neural network architectures, image convolution, convolutional neural networks, and generative adversary networks, which are necessary to understand how CycleGAN works. We present interesting examples for five different types of image translations — style transfer, object transfiguration, season transfer, generating photographies from paintings, and photographic enhancements. We analyze the official TensorFlow implementation of CycleGAN, written in the Python programming language and compare it with the implementation suggested by the original paper. We test the algorithm on a database of 1068 images of horses and 1355 images of zebras, with the goal to train the system how to translate the images of horses to the images of zebras, and vice versa. We outline the differences between the original proposal and the TensorFlow implementation.

Acknowledgments

I would like to thank my mentors for their professional advice, patience and encouragement in the creation of the thesis. Sincere thanks to my parents for all their support, enormous love and strongly believing in me. Thanks also to everyone else who has stood by me all these years.

Contents

List of Tables

List of Figures

1 Introduction

Image-to-image translation has attracted a lot interest and made significant progress in recent years, because of its numerous applications in a variety of computer vision and image processing issues, including image synthesis, segmentation, style transfer, restoration, and posture estimation[11]. Image-to-image translation is a process of transforming images from one domain to another, with the goal to learn the mapping between an input image and an output image. It can be seen as a generative problem in which the input image determines how the output image is generated.

For instance, we can take a certain picture from the source domain and "translate" it to an image with the same content, but drawn in a different artistic style, where some collection of images is taken as a reference for the target artistic style. Another known use is transformation between animals, e.g. horses to zebras.

The transformation function can be learned through methods of supervised machine learning, where the training sets are sets of aligned pairs of images, e.g. a particular image of a horse corresponds to a particular image of a zebra. However, in many domains, such matched pairs are not always available, typically in domains of artistic styles, where the collection of images of a particular artist is scarce[9]. So, knowing this, we can distinguish between two types of image-to-image translation — paired and unpaired. In paired image-to-image translation, the input and the ground-truth image domains are aligned, although the paired training samples may be challenging to get. On the contrary, in unpaired image-to-image translation we do not have explicit pairs of images. This distinction is depicted in Figure 1.

In this thesis we focus on unpaired image to image translation, more specifically on the algorithm called CycleGAN [23], which is a special type of Generative Adversarial Network(GAN), successfully used for image style transfer.

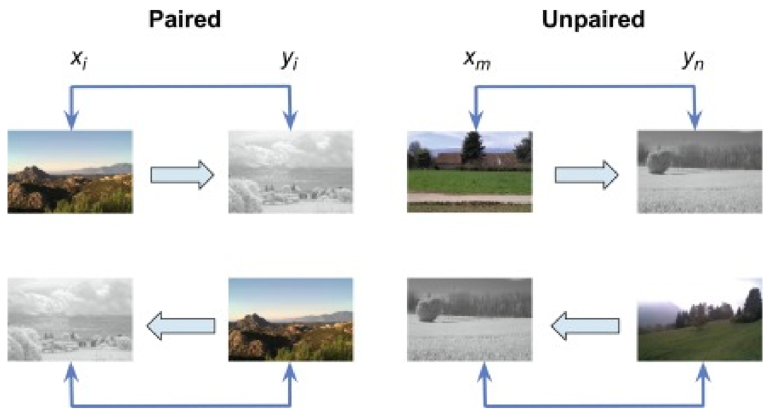

Figure 1: Paired and Unpaired Image to image translation. (Source: [18])

CycleGAN is a deep learning method where two neural networks compete against each other. One network, called the generator, learns to generate an artistic style, while the other one, called the discriminator, learns to discriminate between the real and the generated artistic styles. Once trained, such trained function is able to transform a given image from one artistic style to the other with a minimal effect on the content. Typical applications of this method include transforming between styles of known painters, transforming between horses and zebras, seasons, e.g. summer to winter, day and night [23], etc.

Image-to-image translation techniques address a wide range of issues in image processing, computer graphics, computer vision, and other fields. They have been extensively used in the creation of cartoons, object transformation, picture colorization, creation of semantic labeling from images, image segmentation, style transfer, picture in-painting, 3D posture estimation, image/video colorization, image super-resolution, and semantic image synthesis. Application-specific techniques may be used to address these issues, producing models with a high degree of specialization. However, these are all examples of the same issue, which may be solved by simply mapping one set of pixels to another. Studying a general-purpose approach that enables the use of the same architecture and loss function for a variety of specialized applications is helpful for this reason [11].

In this thesis we will analyze the theoretical and practical aspect of image-to-image translation with the emphasis on the CycleGAN method. We will cover the theoretical foundations of deep neural networks, convolutional networks and adversary generative methods. We will thoroughly analyze the implementation of the CycleGAN algorithm in Python, which is a part of the TensorFlow machine learning library. We will compare it with the implementation from the original paper on CycleGAN.

2 Deep Convolutional Neural Networks

To understand how the CycleGAN algorithm works, we must first consider the basics of deep learning — machine learning techniques that use deep neural network architectures. CycleGAN is based around a special type of neural network architecture called *Convolutional Neural Network* (CNN). In this chapter we outline the basics of artificial neural networks, image convolution and convolutional neural networks.

2.1 Neural Networks

In the machine learning community, one of the most widely used modeling paradigms are neural networks. In traditional methods of programming, when solving a domain specific problem, the programmer gives specific instructions to the computer what to do. The programmer must understand the domain in all its details and solve the problem by dividing into a large number of manageable, carefully defined tasks. On the other hand, using neutral networks, one may approach a problem with a limited domain knowledge and rely on the neural network to learn and gain knowledge from the given or observational data, without any particular instructions or specific commands on how to solve the problem. Neural networks come advantageous in the fields of speech recognition, natural language processing, image recognition, and many more [20].

A neural network is made up of thousands or even millions of tightly connected basic processing units (nodes) - neurons. The majority of neural networks used today are ”feed-forward,” meaning that the data only flows through them in one direction — from the input to the output nodes. These are arranged into layers. A single node may be linked to a number of nodes, from which it receives data, and to which it transmits data.

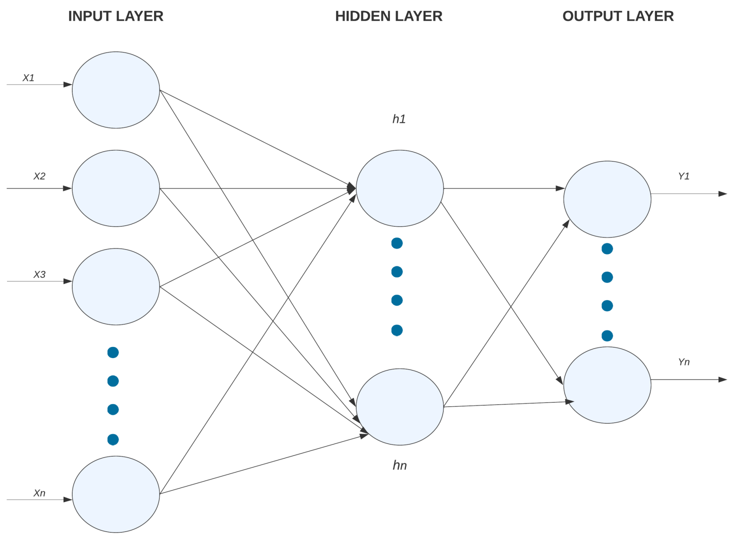

Figure 2: An architecture of a feed-forward neural network.

Let us consider the architecture of a neural network as shown in 2. Each line shows a synapse (a link) between two neurons and the arrow denotes the direction of information flow. Input neurons are the neurons that make up the input layer, which is the leftmost layer in this network. The output neurons, or in this example, just one output neuron, are found in the rightmost or output layer. Since the neurons in the middle layer are neither inputs nor outputs, it is known as a hidden layer. The neural network in the figure only has one hidden layer, but neural networks in general may contain more then one. This is especially the case in deep learning architectures.

The nodes work in parallel and communicate with each other through connections — synapses. Each synapse is assigned a weight. The weights are initially chosen at random, but get refined through neural network training. Depending on the current value of the weights and the type of activation functions in the neurons, certain output neurons get activated when certain input data is present. This way a neural network acts as a mapping between the input and the output data, which is encoded to comply to the input and output layer. In the case of image-to-image translation, input and output neurons would typically be associated with pixels.

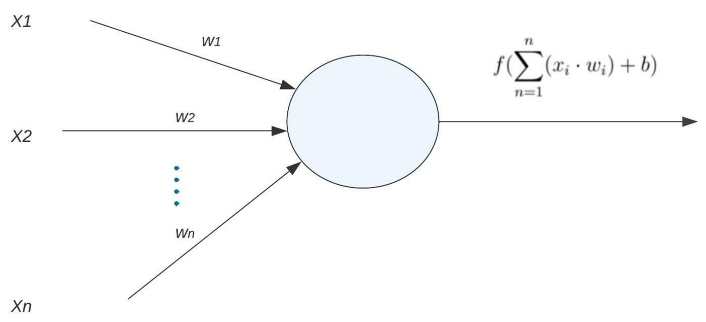

Figure 3: A neuron with n inputs and activation function f.

So, after understanding the architecture of neural networks, let us see how exactly neural networks work. The role of a neural network is to approximate a function, i.e. a mapping between the input and the output data. The mapping works in the following way. An input data is fed through the input layer of nodes. Passing through the synapses to the next layer in the network's architecture, these input values get multiplied by the weights assigned to these synapses and then summed up in the receiving node, where also a real-valued bias can be added. The node is then activated according to its activation function and the computed output value passed on to the nodes in the next layer. Each node therefore executes the following mathematical function:

$$y = f(\sum_{n=1}^{n}(x_i \cdot w_i) + b), \tag{2.1}$$

where x_i are the input node values, w_i the weights, b_i biases, and f the activation function. This process of node activation is depicted in Figure 3.

The input data activates the input neurons, which send the signals to the first hidden layer. The neurons of the first hidden layer sum their weighted inputs and activate their outputs according to their activation function. The process repeats to the next hidden layer. Finally, this forward propagation process reaches the last layer — the output layer. The activated output nodes represent the value of the function that the neural network approximates. Obviously, the output values depend on the weights of synapses (and also the biases assigned with each node). So in order to approximate a desired function, these values need to be adjusted accordingly. This is done through the process of training the neural network, which usually requires a large amount of training samples.

As the training data is being fed through the network, the output values would initially significantly deviate from the expected output values, which is considered a prediction error and determined by the loss function. The loss function is defined by the expert. The goal of training is to minimize the error for the given training samples. If the training data is representative for the learning domain, we may expect the loss to remain small even for instances that the network has not seen during training. The error is being minimized through the algorithm known as *back-propagation*, where the output error is being propagated backwards, from the output layer towards the input layer, during which the weights and biases are getting readjusted by a certain amount which is determined by the learning rate parameter. This way, the network gets slightly more fitted to a specific learning instance (or a batch of learning instances). To prevent overfitting to a small subset of samples, the learning rate is usually small and the process is iterated many times through a large training dataset.

We may summarize the process of training a neural network as follows:

1. Inputs x_i, arrive through the preconnected path.
2. The node's output is computed by Equation (2.1).
3. Outputs of each layer are applied to the inputs of the next layer, until the nodes of the output layer are being activated.
4. The output error is calculated by the loss function.
5. The error is propagated back, towards the input nodes. The weights and biases are being readjusted to reduce the output error.
6. The process is repeated until the desired prediction accuracy is achieved.

In our particular case the training data is a large collection of images. The machine learning algorithm, iterates through the training images many times, until a desired transformation effect is achieved.

2.2 Convolution

Image denoising and feature extraction challenges have received a lot of attention in recent years in the context of image processing [6]. Various methods of computer vision, such as object detection and recognition, motion tracking, identity recognition, and variety of other challenges connected to autonomous picture and video processing, rely on the process of feature extraction [23]. One of the fundamental methods of feature extraction is convolution.

A convolution, in pure mathematics, is the merging of two functions, $f(x)$ and $g(x)$, while one glides over the other. The corresponding points of the first function $f(x)$ and the mirror image of the second function $g(t-x)$ are multiplied together and added for each minute sliding displacement (dx). As we can see in the following formula, two functions are combined to get the outcome, which is denoted by the phrase $[f * g](t)$ [19]:

$$[f * g](t) := \int_{-\infty}^{\infty} f(x)g(t-x)dx \tag{2.2}$$

In image processing, a discrete version of convolution[4] is used as a technique to produce all-purpose filter effects like edge detection, embossing, sharpening, and more. It is also an effective method for extracting image features, which lowers data dimension and provides a less redundant data set, also known as a feature map.

To understand how convolution works on images, we must first understand that digital images are made up of pixels, which are represented numerically. Higher pixel counts result in higher-quality images. Each pixel is typically represented by three color components, and each color component has a brightness value between 0 and 255. Value 0 means a complete absence of this color, while 255 represents the most intense color. All other values in between represent a linear transition between both extremes.

Convolution is performed by sequentially moving a small two-dimensional array of numbers, often a matrix of size $[3 \times 3]$ or $[5 \times 5]$, over the image. This matrix is usually called *filter* or *kernel*. At each convolution step the filter values and the corresponding pixel values are multiplied and summed up. The resulting value then replaces the original value of the central pixel. This way, an output feature matrix is produced.

The mathematical expression for image convolution is:

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x-dx, y-dy) \tag{2.3}$$

where $g(x,y)$ is the filtered image, $f(x,y)$ is the original image, ω is the kernel and $(-a,a)$, $(-b,b)$ the dimensions of the filter (4).

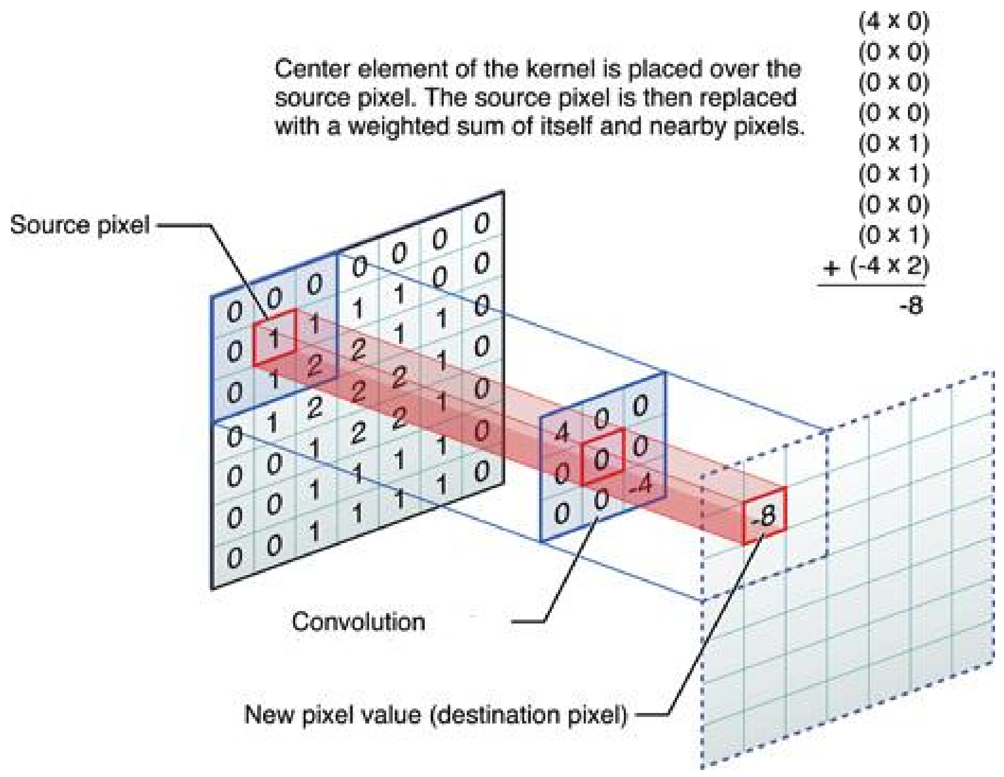
Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution
New pixel value (destination pixel)

Figure 4: Image convolution. (Source: [2])

An example of image convolution is given in Figure 4. A 3×3 kernel is layered on top of a 7×7 source image. In this two 3×3 matrices, the first one is the kernel, and the second one a part of the image. The kernel's central element is positioned above the source pixel. The output pixel is a weighted sum of the input pixels of the image matrix, where weights are determined by the kernel. The process is repeated for all output pixels [2].

The size of the Kernel can be changed to influence the convolution's effect, just as the values of the Kernel can be adjusted for various levels of effects and changes. Figures 5, 6 and 7 show various possible effects by using different types of kernels, respectively image sharpening, blurring, and outlining. Many other effects are also possible, such as embossing, edge detection, noise reduction, etc.

As we can see in the examples, the sharpening kernel highlights differences in nearby pixel values, which results in the picture appearing bigger. The blur kernel minimizes the impact of variations in nearby pixel values. Large disparities in pixel values are brought to light using an outline kernel, often known as the "edge" kernel. In the new image, a pixel adjacent to a neighboring pixel with almost same intensity will seem black, while a pixel next to a neighboring pixel with significantly different intensity would appear white. These examples are taken from [15].

0	-1	0
-1	5	-1
0	-1	0

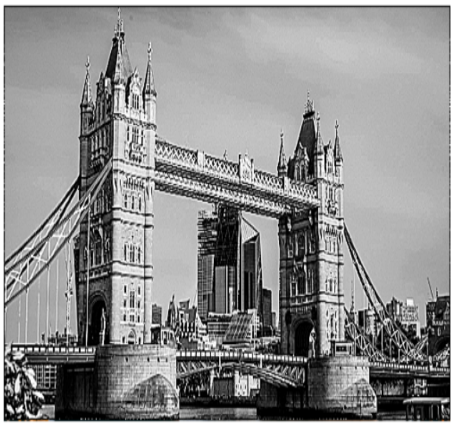

Figure 5: Kernel matrix for sharpening images.

0.0625	0.125	0.0625
0.125	0.25	0.125
0.0625	0.125	0.0625

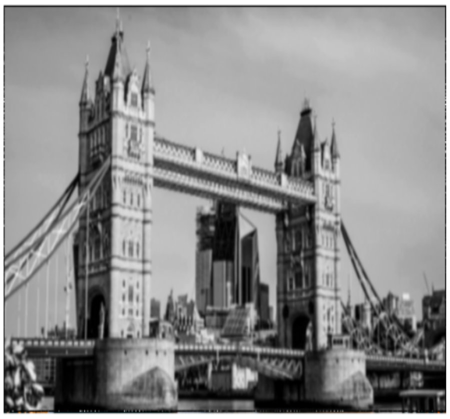

Figure 6: Kernel matrix for blurring images.

-1	-1	-1
-1	8	-1
-1	-1	-1

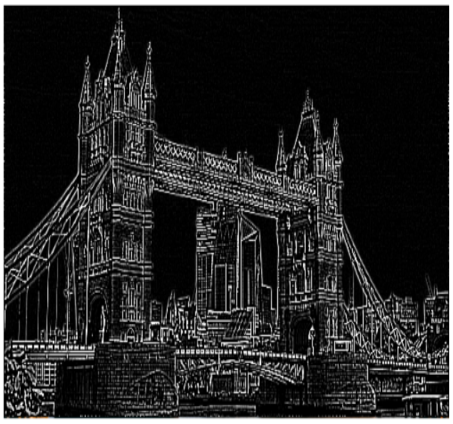

Figure 7: Kernel matrix for outlining images.

2.3 Convolutional Neural Network (CNN)

Convolutional neural networks (CNN) are the variety of deep neural networks that have been the subject of an extensive research in convolutional neural network has produced state-of-the-art results on a variety of tasks by taking use of the sharp increases in the amount of annotated data and the power of graphics processing units [10].

Convolutional neural network is a class of artificial neural network (ANN), designed to process the data coming in the form of multiple arrays, for example a color image composed of three 2D arrays which contain pixel intensity of the three-color channels. Convolutional networks have three types of layers, which are shown in Figure 8:

- Convolutional layer
- Pooling layer
- Fully-connected (FC) layer

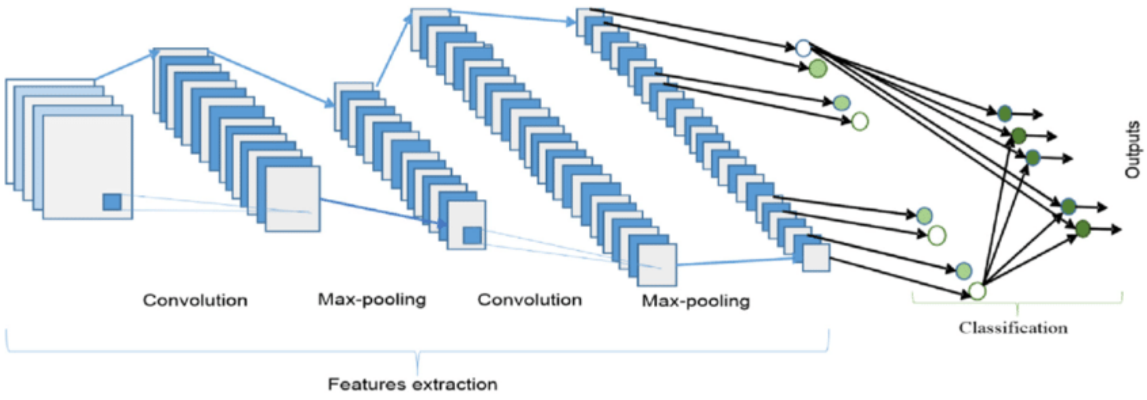

Figure 8: The overall architecture of the Convolutional Neural Network (CNN). (Source: [3])

A convolutional network's initial layer is the convolutional layer. A neural network may contain more than one convolutional layer. Typically, low-level characteristics like color, borders, gradient direction, etc. are captured by the first layer.With additional convolutional layers CNN becomes more capable, which allows it to detect more complex features. The architecture adjusts to high-level characteristics as well, giving us a network that comprehends the dataset's pictures holistically, much like we people do. So as a result, we can have two outcomes. The first is when the dimensionality of the convolved feature is decreased as compared to the input, and the other one is where it is either increased or stays the same [8]. The bigger features or forms are first recognized when the visual data moves through the CNN layers, and eventually the desired object is recognized by the later layers(pooling and fully-connected layer). We will briefly describe the three types of layers, in the reminder of this section.

2.3.1 Convolutional Layer

The purpose of this layer is to learn convolutional filters (or kernels) and their weights, so that they become capable of extracting the important features for the given problem. The architecture of the convolutional layer determines the following parameters: the number of filters (kernels), the size of the filters and the stride. Convolutional layer works by having each filter perform a convolution over the entire input image, while it moves horizontally and vertically with the given stride, shown in the Figure 9. The size of the filter and the stride determine the height and width of the output data, while the number of filters determines its depth.

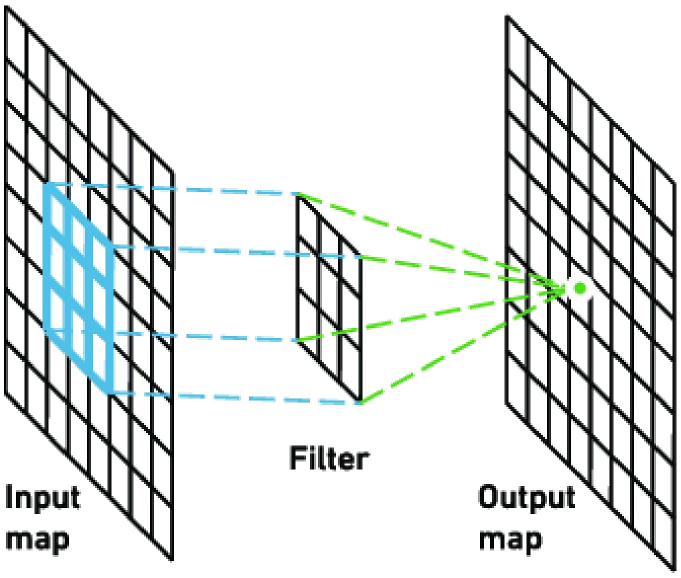

Figure 9: Outline of the convolutional layer. (Source: [21])

2.3.2 Pooling layer

The pooling layer is mainly used to reduce the size of the input image in order to give the neural network invariance and robustness. It is also known as a down-sampling layer. The pooling operation distributes a filter across the entire input, similarly as the convolutional layer, with the exception that this filter does not use or learn weights.

There exist two types of pooling. One is *maximum pooling* and the other one is *average pooling*. The most used sub-sampling module in image processing is maximum pooling (max pooling) [16]. With this type of pooling, the image is first divided into blocks. As the filter goes through the input, the pixel with the highest value is taken as the pixel value of the output image. On the other hand, average pooling takes the average value of all the values as the pixel value in the output image. An example is given in Figure 10.

Max pooling is used to downsize the picture(if too big) and to extracts the most important features for e.g edges. While on the other hand average pooling extracts features more smoothly than maximum pooling.

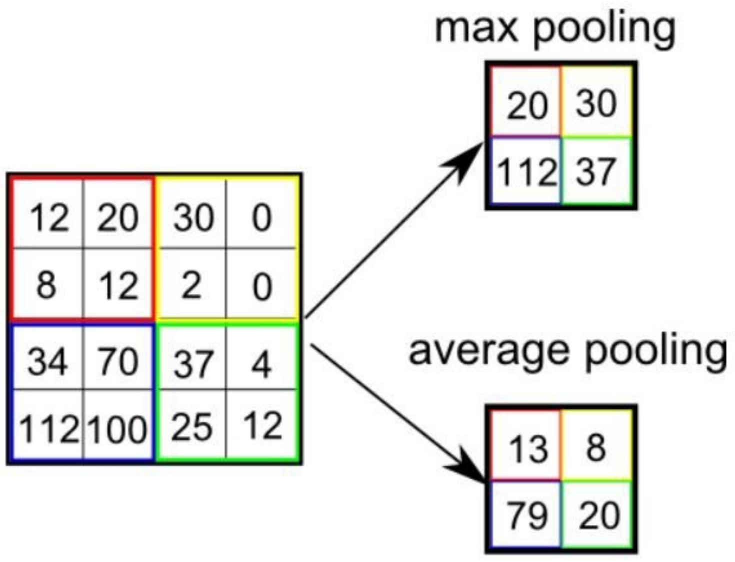

Figure 10: Types of pooling. (Source: [16])

2.3.3 Fully-connected layer

A fully connected layer is usually added for classification purposes and input it into a standard neural network. It serves as a link between a convolutional or pooling layer and the output layer. Every node in a fully-connected layer is directly connected to every node in the next layer. This way the output from the convolutional layers is flatten down, so it can be connected to the simple layout of the output neurons. In Figure 8, the fully connected layer is depicted under the “classification” tag.

3 Image-to-image translation

Image-to-Image Translation aims to translate images from one domain to another by learning a mapping between the input and output pictures using a training dataset of aligned or unaligned cross-domain image pairings. With aligned cross-domain image pairings, the input and the ground-truth image domains are pared,contrary, in unaligned image-to-image translation we do not have explicit pairs of images. Traditionally, this activity has been completed using a training set of aligned picture pairings. However, for many applications, paired training data is not available, and preparing them frequently requires a lot of labor from trained workers to gather thousands of paired picture data-sets, particularly with sophisticated image translations. CycleGAN is an architecture that can solve this problem, because it learns image translations without explicitly using pairings of images [18].

3.1 Generative Adversarial Networks (GAN)

To understand how CycleGAN works, we first have to undersand Generative adversarial networks (GANs). Those are algorithmic architectures that use two neural networks, competing one against the other (thus the "adversarial") in order to generate new, synthetic instances of data that can pass as real data. They are used widely in image generation, video generation and voice generation [23]. Given that they may be trained to replicate any data distribution, GANs have enormous potential for both good and bad. In other words, GANs may be trained to construct worlds that closely resemble our own in every area, including visuals, music, speech, and literature. In a way, they might be thought of as robot artists, and their work is often impressive. However, they may also be employed to create fake media material. [17]

GANs are an example of unsupervised machine learning. This means that the training data is not labeled with the expected output as in the case of supervised machine learning. In supervised machine learning the output (prediction) is compared with the expected results. This way it is determined how to improve the model to provide better outputs based on the expected output and the actual projected outcome. On the other hand, the architecture of a GAN is composed of two neural networks. The first one is called *generator* and the second one *discriminator*. The generator's task is to provide fake input (samples). The discriminator's task is to determine if a given sample is authentic or false. The adversarial nature of this lies in the fact that these two networks compete among each other. The generator is learning how to fool the discriminator and the discriminator is learning how to not be fooled by the generator. The final goal of this process is to make the generator good enough to eventually "fool" the human, which means to generate content that is as realistic as possible.

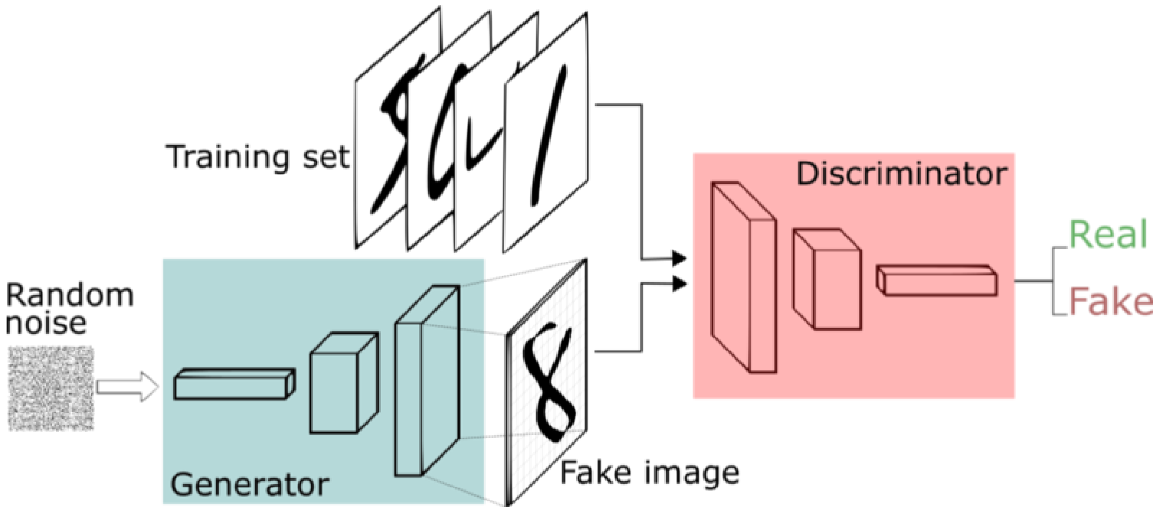

Figure 11: Generative Adversarial Network framework. (Source: [17])

The competition between the generator and the discriminator is a zero sum game. There is always a winner and a loser. The winner gets to stay the same, while the loser must update its model. Therefore, if the discriminator identified the item on the photo as a phony image, the discriminator is unaffected. But in order to produce better fakes, the generator will need to modify its model. The discriminator model will consequently need to be updated in order to be able to recognize the more realistic looking fake images. In contrast, if the opposite is true and the generator is producing something that deceives the discriminator, the discriminator model will need to be updated. Both networks are attempting to maximize an opposing and different objective function, or loss function. Their losses conflict with one another. [17]

As an example, let us assume we have a type of a low-quality image. By determining what each individual pixel is and then constructing a higher resolution version of that, we can use a GAN to build a much better resolution version of that image.

Some examples of GAN usage are: generating examples for image datasets, generating photographs of human faces, generating realistic photographs, generating cartoon characters, image-to-image translation, text-to-image translation, semantic-image-to-photo translation, face frontal view generation, generate new human poses, photos to emojis, photograph editing, face aging, photo blending, super resolution, clothing translation, video prediction, 3D object generation [5].

3.2 Cycle Consistency GAN (CycleGAN)

In this section we aim to understand what exactly CycleGAN is. We summarize the theory and present some examples from paper [23]. All the photos presented in this section are also taken from this paper.

In image-to-image translation, a new synthetic form of an existing image is created with a particular modification. A large dataset of matched samples is often needed when training a model for image-to-image translation. That is, a large collection of numerous instances of input photos X (for instance shown in Figure 12, summer landscapes) and the same image modified in the desired way that may be used as an anticipated output image Y (e.g. winter landscapes).

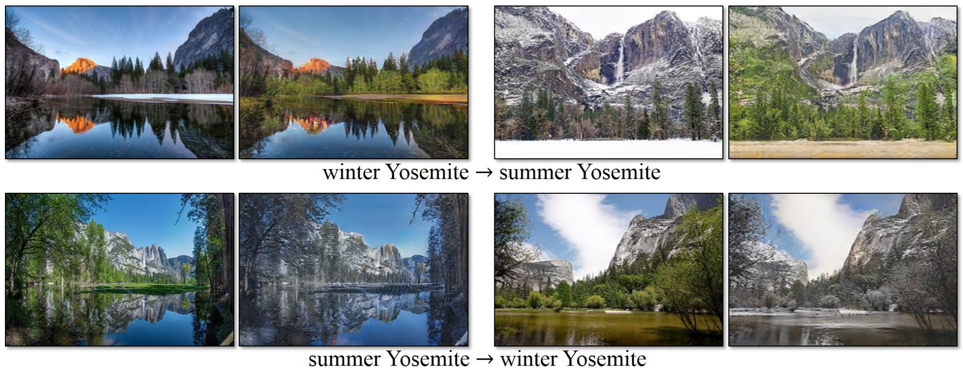
winter Yosemite → summer Yosemite
summer Yosemite → winter Yosemite

Figure 12: Image-to-image tanslaton, summer to winter.

Such datasets, like an images of artworks by long-dead painters, can be difficult, costly, and in some cases impossible to produce. Using the CycleGAN method, image-to-image translation models are automatically trained without the use of paired samples. Unsupervised learning is used to train the models using a set of images from the source and target domains that are not related in any way.

Before we start describing the architecture of CycleGAN, let's briefly recall the architecture of GAN. There are two models that create the GAN architecture, a generator model and a discriminator model. The discriminator takes a picture as input and determines if it is real (from a dataset) or fake, whereas the generator takes a point from a latent space as input and creates new plausible images from the domain (generated). In order to improve both the generator's ability to deceive the discriminator and the discriminator's ability to recognize created pictures, both models are trained against each other.

The extension of the GAN architecture is CycleGAN, which involves training two generator models and two discriminator models at the same time. Cycle consistency is an additional extension to the architecture that is used by the CycleGAN. If an image produced by the first generator is given as the input for the second generator, its output must resemble the original image. The opposite is also true, meaning that if the output from the second generator is fed into the first generator as input, the output should resemble the second generator's input. An interesting analogy from [23] is the following: if we have a sentence in English, and this sentence is translated from English to French, when translated back from French to English, the resulting sentence should be exactly the same. This is called *cycle consistency*. By including an extra loss to calculate the difference between the output of the second generator and the original picture, and vice versa, the CycleGAN promotes cycle consistency. As a result, the generator models are regularized, directing picture production in the new domain in the direction of image translation.

3.2.1 Applications of CycleGAN

There are numerous implementations of the CycleGAN. Paper [23] examines five such applications. So, let us briefly describe them.

1. **Style transfer**

   Style transfer is the process of taking an aesthetic approach from one domain — often paintings — and applying it to another domain, e.g. photographies of landscapes. To the images in Figure 13 are applied the aesthetics of Monet, Van Gogh, Cezanne, and Ukiyo-e.

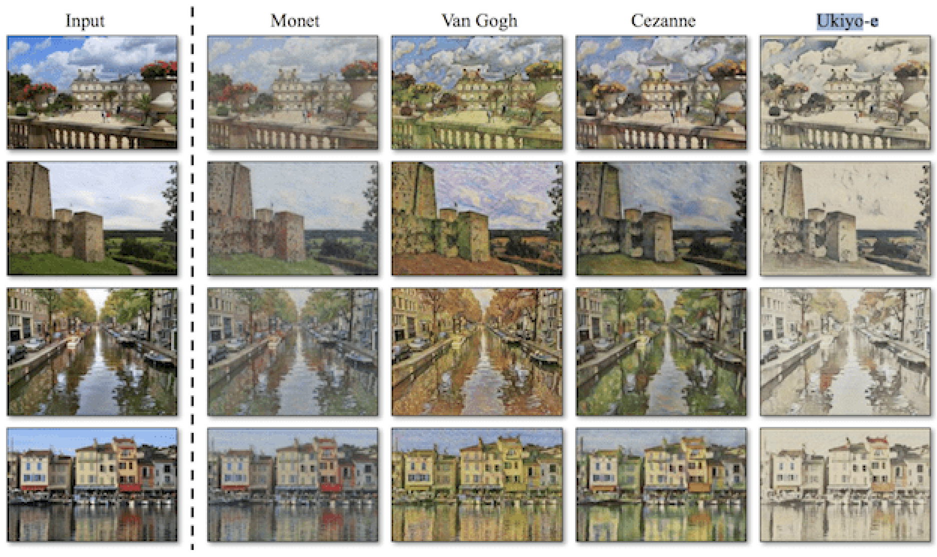

Figure 13: Example of style transfer.

2. **Object transfiguration**

   Transfiguration of objects from one class to another is referred to as object transfiguration. Figure 14 shows transformations of images of horses to images to zebras, and vice versa.

3. **Season transfer**

   Season transfer is the process of converting photos captured in one season, e.g. in the summer, to a different season, e.g. the winter. Some examples are shown in Figure 15.

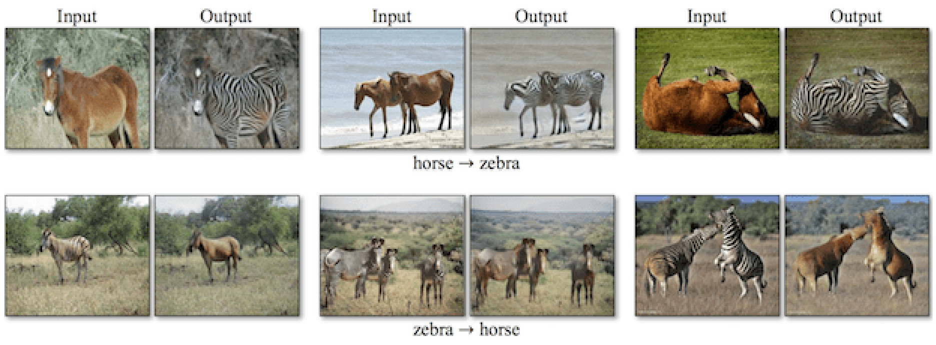

Figure 14: Example of object transfiguration.

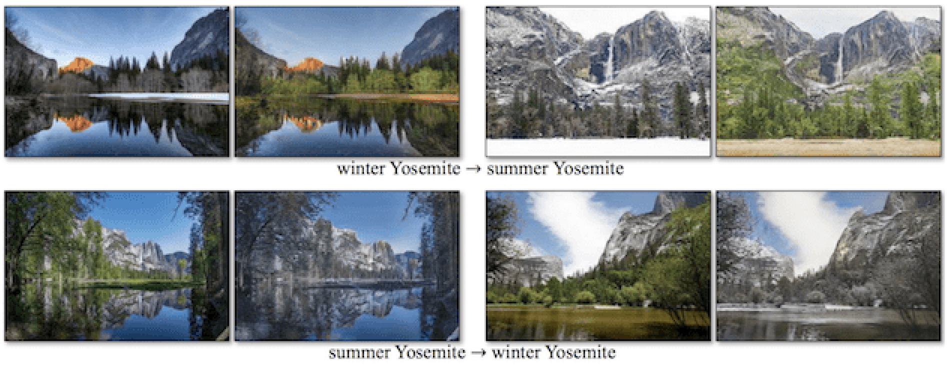
winter Yosemite → summer Yosemite
summer Yosemite → winter Yosemite

Figure 15: Example of season transfer.

4. **Generating photographies from paintings**

   Photography generation from paintings is a process of transforming a painting, generally by a well-known artist or an iconic scene, to a photo-realistic image. Figure 16 shows some paintings by Monet, translated to images of plausible photographic style.

5. **Photographic enhancement**

   The term "photographic enhancement" describes changes that make the original image better. Figure 17 shows examples of photos where the depth of field is enhanced.

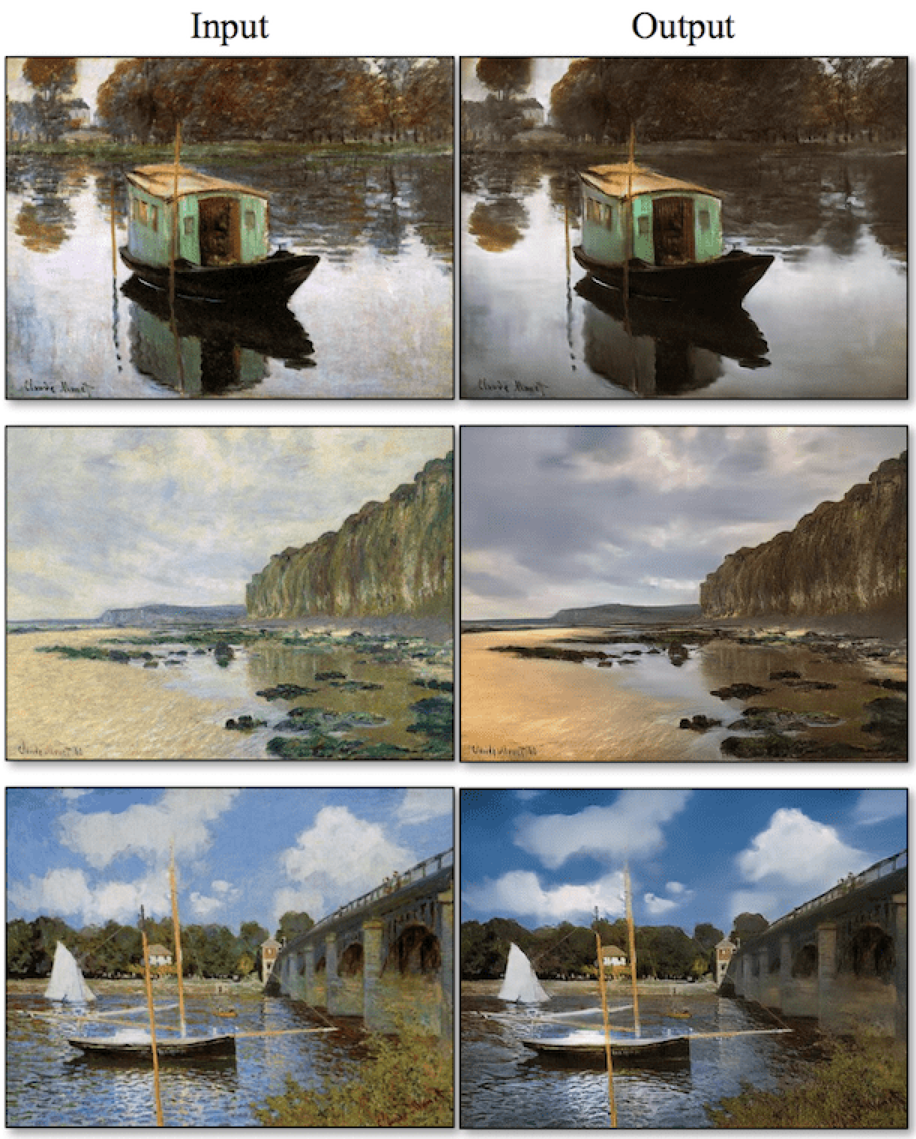

Figure 16: Example of photograph generation from paintings.

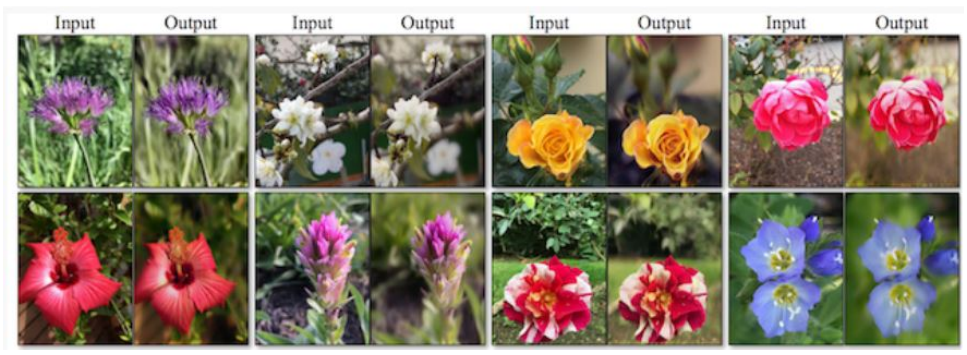

Figure 17: Example of photograph enhancement.

4 Analysis of the CycleGAN algorithm

In this chapter we analyze and describe the Python implementation of the CycleGAN algorithm, which can be found in the official repository [7] of the TensorFlow machine learning library [1]. As an example data the program code uses images of horses and zebras from the `tensorflow_datasets` package, which is accessible through the PIP Python package manager. The package contains 1068 images of horses and 1355 images of zebras. These two sets can easily be replaced with any images of choice, so instead of *horses* and *zebras* domains we will here refer to image domains A and B.

4.1 Algorithm overview

Training of the CycleGAN takes place over numerous epochs. An epoch refers to one complete cycle through all the testing instances, where each instance is processed once. The training process can be stopped after finishing any epoch, but waiting for more epoch yields better final results. After each epoch an example image of a horse being transformed to a zebra is output for the user to visually evaluate the training progress. This Tensorflow example trains the neural networks for 40 epochs, while the original paper [23] proposes 200 epochs.

Each epoch iterates through the same set of randomly generated pairs of images (a_T, b_T), where $a_T \in A$, $b_T \in B$ are considered *true* images, as opposed to a_F, b_F, which are *false* images, respectively generated by generators G_b and G_a. The sets of *true* images are limited, so to prevent overfitting to this specific database, training images are preprocessed — each image is slightly enlarged and then randomly cropped to size 256×256 pixels.

The most complex part of the algorithm is the implementation of the loss functions. This includes computation of 4 distinct loss components — the generator loss, the cycle loss, the identity loss, and the discriminator loss. After the loss values are computed, the Adam optimizer [13] is applied to train the generators and discriminators. The optimizer's learning parameters are the same as suggested in the original paper.

The complete algorithm can be summarized as follows:

```
1. Load the sets $\mathcal{A}$, $\mathcal{B}$ of training images for domains $A$ and $B$ respectively.
2. Shuffle and apply a random jitter to sets $\mathcal{A}$ and $\mathcal{B}$.
3. Construct the generators $G_a, G_b$ and the discriminators $D_a, D_b$.
4. Repeat for each epoch:
   4.1. For each $(a_T, b_T)$ from $(\mathcal{A}, \mathcal{B})$:
        4.1.1. Generate the cycle $\mathcal{A} \to \mathcal{B} \to \mathcal{A}$ for true image $a_T$:
               $b_F = G_a(a_T),\ \ a_C = G_b(b_F)$
```

```
    4.1.2. Generate the cycle $\mathcal{B} \to \mathcal{A} \to \mathcal{B}$ for true image $b_T$:
           $a_F = G_b(b_T),\ \ b_C = G_a(a_F)$
    4.1.3. Compute the loss values for generators $G_a$ and $G_b$:
           $\text{loss}(G_a) = \text{BCE}(\text{ONES}, D_b(b_F))$
           $\text{loss}(G_b) = \text{BCE}(\text{ONES}, D_a(a_F))$
    4.1.4. Compute the cycle loss value:
           $\text{cycle\_loss} = (\text{MAE}(a_T, a_C) + \text{MAE}(b_T, b_C))$
    4.1.5. Compute the identity loss values for generators $G_a$ and $G_b$:
           $\text{identity\_loss}(G_a) = \text{MAE}(b_T, G_a(b_T))$
           $\text{identity\_loss}(G_b) = \text{MAE}(a_T, G_b(a_T))$
    4.1.6. Compute the total loss values for generators $G_a$ and $G_b$:
           $\text{total\_loss}(G_a) = \text{loss}(G_a) + \lambda \cdot \text{cycle\_loss} + 0.5 \cdot \lambda \cdot \text{identity\_loss}(G_a)$
           $\text{total\_loss}(G_b) = \text{loss}(G_b) + \lambda \cdot \text{cycle\_loss} + 0.5 \cdot \lambda \cdot \text{identity\_loss}(G_b)$
    4.1.7. Compute the loss values for discriminators $D_a$ and $D_b$:
           $\text{loss}(D_a) = (\text{BCE}(\text{ONES}, D_a(a_T)) + \text{BCE}(\text{ZEROS}, D_a(a_F)))\,/2$
           $\text{loss}(D_b) = (\text{BCE}(\text{ONES}, D_b(b_T)) + \text{BCE}(\text{ZEROS}, D_b(b_F)))\,/2$
    4.1.8. Apply the Adam optimizer to $G_a, G_b, D_a, D_b$ :
           $\text{ADAM}(G_a, \text{total\_loss}(G_a), \text{lr} = 0.0002, \beta_1 = 0.5, \beta_2 = 0.999)$
           $\text{ADAM}(G_b, \text{total\_loss}(G_b), \text{lr} = 0.0002, \beta_1 = 0.5, \beta_2 = 0.999)$
           $\text{ADAM}(D_a, \text{loss}(D_a), \text{lr} = 0.0002, \beta_1 = 0.5, \beta_2 = 0.999)$
           $\text{ADAM}(D_b, \text{loss}(D_b), \text{lr} = 0.0002, \beta_1 = 0.5, \beta_2 = 0.999)$
  4.2. Output an example translation $(a_T, G_a(a_T))$.
```

4.2 Architecture of neural networks

The architecture of generators and discriminators somewhat differ from the one proposed by the original authors [23], who adopted it from an earlier work by Johnson et al. [12] on neural style transfer and super resolution. The implementation that we here consider rather adopts its neural network architecture from the work of Isola et al. [11], who proposed an algorithm for image-to-image translation with conditional GAN, that was implemented as the *pix2pix* program, which is available in the official TensorFlow examples repository [14].

The architecture of the generators is given in Table 1. The first 8 layers sample down the input image to a single pixel and from there on the image is upsampled back to 256×256 pixels. These layers that upscale data are called *deconvolutional layers* [22] and perform an operation, which is the reverse of convolution.

Layers are not connected in a strict linear order, but pairwise — the output of a downsampling layer is concatenated with the output of the upsampling layer of the same dimension, before input to the next layer in the sequence. So, the output of layer 1 with dimensions

Table 1: The architecture of the CycleGAN generator.

Layer	Input dimension	Num. of filters	Filter size	Stride	Activ. function
1	$(256, 256, 3)$	64	4×4	2×2	LeakyReLu
2	$(128, 128, 64)$	128	4×4	2×2	LeakyReLu
3	$(64, 64, 128)$	256	4×4	2×2	LeakyReLu
4	$(32, 32, 256)$	512	4×4	2×2	LeakyReLu
5	$(16, 16, 512)$	512	4×4	2×2	LeakyReLu
6	$(8, 8, 512)$	512	4×4	2×2	LeakyReLu
7	$(4, 4, 512)$	512	4×4	2×2	LeakyReLu
8	$(2, 2, 512)$	512	4×4	2×2	LeakyReLu
9	$(1, 1, 512)$	512	4×4	2×2	ReLu
10	$(2, 2, 1024)$	512	4×4	2×2	ReLu
11	$(4, 4, 1024)$	512	4×4	2×2	ReLu
12	$(8, 8, 1024)$	512	4×4	2×2	ReLu
13	$(16, 16, 1024)$	256	4×4	2×2	ReLu
14	$(32, 32, 512)$	128	4×4	2×2	ReLu
15	$(64, 64, 256)$	64	4×4	2×2	ReLu
16	$(128, 128, 128)$	64	4×4	2×2	Tanh
Output	$(256, 256, 3)$				

Table 2: The architecture of the CycleGAN descriminator.

Layer	Input dimension	Num. of filters	Filter size	Stride	Activ. function
1	$(256, 256, 3)$	64	4×4	2×2	LeakyReLu
2	$(128, 128, 64)$	128	4×4	2×2	LeakyReLu
3	$(64, 64, 128)$	256	4×4	2×2	LeakyReLu
4	Zero padding $(32, 32, 256) \rightarrow (34, 34, 256)$				
5	$(34, 34, 256)$	512	4×4	1×1	LeakyReLu
6	Zero padding $(31, 31, 256) \rightarrow (33, 33, 256)$				
7	$(33, 33, 256)$	1	4×4	1×1	LeakyReLu
Output	$(30, 30, 1)$				

$(128, 128, 64)$ and the output of layer 15 with dimensions $(128, 28, 64)$ are concatenated to data with dimensions $(128, 128, 128)$ and input to layer 16. In the same way, layers 2 and 14 output to layer 15, layers 3 and 13 output to 14, etc. The last such pairing is performed with layers 7 and 9 that output to layer 10. The output of layer 8 (the single pixel data) in not paired with the output of any other layer.

The architecture of the discriminator is more straightforward. Its description is given in Table 2. The output of a discriminator is an image of dimension 30×30 pixels and a single color channel, normalized to real-value interval $[0, 1]$. Each pixel of the output image corresponds to a patch of size 70×70 pixels of the input image. The value of the pixel is the probabilistic estimate for that patch to be a part of the real image, i.e. the value 0 means that the patch is surely false, while 1 means that the patch is surely true. Two examples of discriminator's outputs are given in Figure 18.

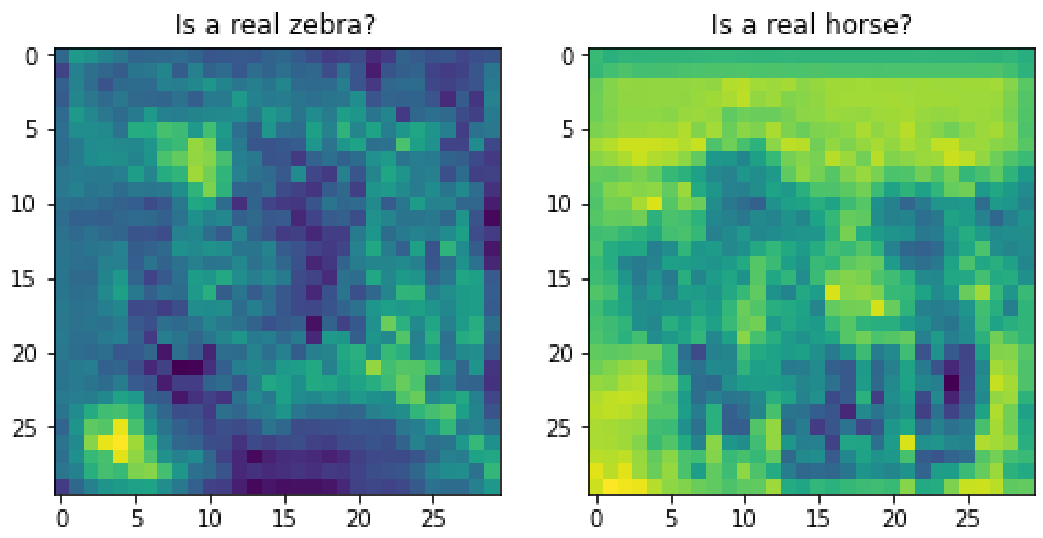

Figure 18: Examples of discriminator's output. Brighter pixels represent a higher probability that the corresponding patch belongs to a true image.

4.3 Loss functions

CycleGAN defines four distinct loss functions to train the neural networks:

- Generator loss function,
- Discriminator loss function,
- Cycle loss function,
- Identity loss function.

The *generator loss function* determines how well did the current generator 'fool' the discriminator. A false image generated by the generator is given to the discriminator and the more the discriminator finds this image to be true, the lower the loss value. The discriminator's 30×30 array output is compared with the array of 30×30 ones. Ideally for the generator is that there is no difference between the two, which would mean a zero-loss. On the other hand, an output of all zeros is the worst case for the generator, with the higher possible loss value. The *binary cross entropy* function for probability of class 1 is therefore used as the loss function for generators:

$$\text{loss}(G) = \text{BCE}(\text{ONES}, D(\text{false\_image})) = -\frac{1}{N}\sum_{i=1}^{N} \log(x_i), \tag{4.1}$$

where $x_1, \dots, x_N \in (0, 1]$ is the discriminator's output. A quick verification of the above equation would confirm that if $x_i = 1$ for all i, $\text{loss}(G_{a/b}) = 0$, and that $\text{loss}(G_{a/b})$ increases rapidly when all x_i are close to 0.

The discriminator must be successful in two ways — detecting false images as false and also detecting true images as true. The *discriminator loss function* is therefore defined as:

$$\text{loss}(D) = \left(\text{BCE}(\text{ONES}, D(\text{true\_image})) + \text{BCE}(\text{ZEROS}, D(\text{false\_image}))\right)/2, \tag{4.2}$$

where

$$\text{BCE}(\text{ZEROS}, D(\text{false\_image})) = -\frac{1}{N}\sum_{i=1}^{N}\log(1-x_i), \tag{4.3}$$

and $\text{BCE}(\text{ONES}, D(\text{true\_image}))$ as in (4.1).

The *cycle loss* represents the distortion of the original image when passed through both generators. Ideally, both generators work as inverters, one translating a horse to a zebra, and the other one the translated zebra back to the original horse, and vice-versa. The more the original image gets distorted in such a cycle, the higher the loss value. The *cycle loss function* is defined simply as the Mean Absolute Error (MAE) of the image difference:

$$\text{cycle\_loss} = \text{MAE}\left(\text{true\_image}, G_{a/b}(G_{b/a}(\text{true\_image}))\right) = \frac{1}{N}\sum_{i=1}^{n}|x_i - \hat{x}_i|, \tag{4.4}$$

where x_i are pixels of the original image and $\hat{x}_i$ the corresponding pixels of the cycled image.

Finally, the *identity loss function* helps train the generator in such a way, that when given a target image as an input, it changes it as little as possible. Suppose that we give to a zebra generator a picture with a horse and a zebra standing together. Ideally, the generator would turn the horse into a zebra, while leaving the zebra as is. Needless to say, a picture with just zebras on it should remain intact. The identity loss function is therefore defined as:

$$\text{identity\_loss}(G) = \text{MAE}\left(\text{target\_image}, G(\text{target\_image})\right) = \frac{1}{N}\sum_{i=1}^{n}|x_i - \hat{x}_i|, \tag{4.5}$$

where x_i are pixels of the original target image and $\hat{x}_i$ the corresponding pixels of the generated image.

The generators are trained using the weighted sum of the *generator loss function*, the *cycle loss function* and the *identity loss function*:

$$\text{total\_loss}(G) = \text{loss}(G) + \lambda \cdot \text{cycle\_loss} + 0.5 \cdot \lambda \cdot \text{identity\_loss}(G), \tag{4.6}$$

where the parameter λ was set to 10 in the implementation that we here consider. The discriminators are trained with the *discriminator loss function*.

5 Conclusion

In this thesis we made a thorough study of the recent advances in Generative Adversary Networks for image-to-image translation, which is a subfield of deep neural network learning — a machine learning discipline which became extremely popular in the last decade, due to the rise and accessibility of CPU and GPU computation power. We outlined all the theoretical prerequisites, including neural network architectures, image convolution, convolutional neural networks, and generative adversary networks, which are necessary to understand how machine learning is used to achieve translations between different image domains. We then summarized the original paper on the state-of-the-art algorithm called CycleGAN, which is able to learn such translations on the training sets of unpaired images. We presented interesting examples of such translations for five different types of applications — style transfer, object transfiguration, season transfer, generating photographies from paintings, and photographic enhancements. We then tested and analyzed the official TensorFlow implementation of CycleGAN, written in the Python programming language and compared it with the implementation suggested by the original paper. We analyzed the architecture of the *generator* and the *discriminator* neural networks, including the types, dimensions, activation functions, and connectedness of each layer; definitions of all 4 types of loss functions and the parameters used; and the overall training algorithm. We tested the algorithm on a database of 1068 images of horses and 1355 images of zebras, with the goal to train the system how to translate the images of horses to the images of zebras, and vice versa. We discovered that there are some slight differences between the original proposal and the TensorFlow implementation, which we studied, mainly in the architecture of neural networks. TensorFlow adopted the architectures from another project called *pix2pix*, which is also a part of the TensorFlow examples repository and implements a GAN approach for synthesizing and reconstructing photographies. However, the differences seem to be minor and should have little or no visible impact on the final results. Moreover, the TensorFlow implementation suggest training for 40 epochs, while the original paper suggests 200 epochs.

There were many ideas in which directions to continue this study, which have to be postponed to a later time, due to the lack of resources and the limited time to finish this thesis. The TensorFlow CycleGAN implementation could be tested out on various images domains, including the landscapes in different seasons, paintings of known painters, photos of different kind of animals, flowers, etc. Performance of the algorithm could be measured when executed on a CPU vs. GPU with different sizes of training sets and different sizes and qualities of images. The number of needed training epochs could be evaluated before the results are not anymore distinguishable by the human eye. Since the training is very time consuming — one epoch can take hours to finish — such en estimate for different types and quality of images could be useful. However, because training a CycleGAN is so time consuming, such a project should probably demand a few more months of work.

6 Povzetek naloge v slovenskem jeziku

Prevajanje slika-v-sliko (angl. *image-to-image*) je proces pretvorbe slik iz ene domene slik v drugo, npr. iz domene slik konjev v domeno slik zeber. Učenje pretvarjanja iz slike v sliko je v bistvu učenje preslikave med vhodnimi in izhodnimi slikami, kjer so nabori učnih primerov slik lahko urejeni ali neurjeni. Pri urejenih meddomenskih parih slik je izhodna slika dejanski rezultat preslikave vhodne slike, pri neurejenih parih pa sta vhodna in izhodna slika lahko poljubna elementa svoje domene. Tradicionalno je bilo učenje izvedeno nad nizi urejenih parov slik. Vendar pa v mnogih primerih takšni urejeni pari niso znani oz. ne obstajajo. Če na primer želimo pretvarjati med dvema slikarskima stiloma, zelo verjetno ne bomo imeli na voljo nabora parov slik, kjer je popolnoma enaka vsebina narisana v dveh različnih stilih. Algoritem CycleGAN lahko reši ta problem. To je oblika generativne nasprotniške nevronske mreže (angl. *Generative Adversial Network*, GAN), kjer dve nevronski mreži tekmujeta druga proti drugi. Ena nevronska mreža se imenuje *generator*, druga pa *diskriminator*. Naloga generatorja je, da pretvori dano sliko iz ene slikovne domene v drugo, naloga diskriminatorja pa, da razlikuje med pravimi in generiranimi slikami. Obe nevronski mreži med učenjem tekmujeta. Generator se uči preslepiti diskriminator, diskriminator pa se uči prepoznati ponaredke generatorja. Generator lahko sčasoma postane tako dovršen, da lahko preslepi človeško oko. Naučimo ga lahko preoblikovanja slik iz enega umetniškega sloga v drugega z minimalnim vplivom na samo vsebino, preoblikovanja med različnimi objekti, npr. med konji in zebrami, preoblikovanja med letnimi časi, npr. med poletjem in zimo, med dnevom in nočjo, itd.

V tej diplomski nalogi smo izvedli temeljito študijo nedavnega napredka v generativnih nasprotniških nevronskih mrežah za prevajanje slika-v-sliko, kar spada v področje globokega učenja nevronskih mrež — disciplino strojnega učenja, ki je postala izjemno priljubljena v zadnjih desetletjih, predvsem zaradi porasta in dostopnosti računalniške procesne moči CPE in GPE. V nalogi smo najprej povzeli vse potrebne teoretične osnove, vključno z arhitekturami nevronskih mrež, konvolucijo slik, konvolucijskimi nevronskimi mrežami ter teorijo generativnih nasprotniških nevronskih mrež. Nato smo predstavili vsebino izvornega članka o algoritmu CycleGAN ter predstavili zanimive primere slikovnih pretvorb za pet različnih načinov uporabe tega algoritma — prenos sloga, preoblikovanje objekta, pretvorba letnih časov, generiranje fotografij iz slik ter izboljšave fotografij. Preizkusili in analizirali smo uradno TensorFlow implementacijo algoritma CycleGAN, napisano v programskem jeziku Python ter jo primerjali s predlagano implementacijo izvornega članka. Analizirali smo arhitekturo generatorskih in diskriminatorskih nevronskih mrež, vključno z vrstami posameznih plasti, njihovimi dimenzijami, aktivacijskimi funkcijami in povezanostjo. Analizirali smo definicije vseh štirih tipov funkcij izgube (angl. *loss functions*) in uporabljene parametre. Preučili smo tudi algoritem treniranja nevronskih mrež. Algoritem smo nato testirali na podatkovni bazi 1068 slik konjev in 1355 slik zeber, z namenom usposobiti sistem za pretvarjanje podob konjev v podobe zeber in obratno. Ugotovili smo, da obstaja nekaj manjših razlik med prvotnim predlogom implementacije in izvedbo TensorFlow. Razlike so predvsem v arhitekturi nevronskih mrež, saj je TensorFlow prevzel arhitekturo po nekem drugem projektu, ki se imenuje *pix2pix* in je prav tako del repozitorija primerov knjižnice TensorFlow. Razlike so sicer majhne in menimo, da ne bi smele imeti vidnega vpliva na končne rezultate. Poleg tega implementacija TensorFlow predlaga, da treniranje izvajamo 40 epoh, medtem ko izvorni članek predlaga 200 epoh.

Diplomska naloga se osredotoča predvsem na teoretično plat študije algoritma CycleGAN, obstaja pa še veliko praktičnih možnosti za nadaljevanje dela. Algoritem CycleGAN bi lahko preizkusili na različnih domenah slik, npr. slik pokrajin v različnih letnih časih ali ob različnih delih dneva, slik znanih slikarjev, fotografij različnih vrst živali, rož, itd. Lahko bi izmerili hitrost učenja pri izvajanju algoritma na CPU v primerjavi s hitrostjo pri izvajanju na GPE. Pri tem bi lahko uporabili različne velikosti učnih množic, ob različnih velikostih in kakovostih slik. Ugotavljali bi tudi lahko, najmanj koliko epoh učenja je potrebnih, da človeško oko več ne zazna razlike med generiranimi slikami. Zaradi visoke računske zahtevnosti algoritma CycleGAN predvidevamo, da bi za izvedbo takšne praktične študije verjetno potrebovali nekaj mesecev.

7 Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Alexandros Agapitos, Michael O'Neill, Miguel Nicolau, David Fagan, Ahmed Kattan, Anthony Brabazon, and Kathleen Curran. Deep evolution of image representations for handwritten digit recognition. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 2452–2459, May 2015.

[3] Md. Zahangir Alom, Tarek Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Nasrin, Mahmudul Hasan, Brian Essen, Abdul Awwal, and Vijayan Asari. A State-of-the-Art Survey on Deep Learning Theory and Architectures. *Electronics*, 8, March 2019.

[4] Amidi S. Amidi A. Deep Learning: Convolutional Neural Networks. Stanford University, 2019. `https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks`.

[5] Jason Brownlee. 18 Impressive Applications of Generative Adversarial Networks (GANs), June 2019. `https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/`.

[6] Capobianco, Giovanni Cerrone, Carmine Di Placido, Andrea Durand, Daniel Pavone, Luigi Russo, Davide Donato Sebastiano, and Fabio. Image convolution: a linear programming approach for filters design. *Soft Computing*, 25(14):8941–8956, 2021.

[7] CycleGAN notebook at Google Colab. `https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/generative/cyclegan.ipynb`. Accessed: 2022-08-02.

[8] IBM Cloud Education. Convolutional neural networks. `https://www.ibm.com/cloud/learn/convolutional-neural-networks`. Accessed: 2022-07-19.

[9] Aashutosh Ganesh and Koshy George. Chapter 11 - generative adversarial networks for histopathology staining. In Arun Solanki, Anand Nayyar, and Mohd Naved, editors, *Generative Adversarial Networks for Image-to-Image Translation*, pages 263–287. Academic Press, 2021.

[10] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[11] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2016. doi: 10.48550/arXiv.1611.07004.

[12] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016. doi: 10.48550/arXiv.1603.08155.

[13] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, 12 2014.

[14] TensorFlow official examples repository. `https://github.com/tensorflow/examples`. Accessed: 2022-08-05.

[15] Victor Powell. Image kernels. `https://setosa.io/ev/image-kernels/`, 2019. Accessed: 2022-07-11.

[16] Sumit Saha. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`, 2018. Accessed: 2022-07-22.

[17] Thalles Silva. An intuitive introduction to generative adversarial networks (gans). `https://www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394`, 2018. Accessed: 2022-07-28.

[18] Patricia L. Suárez, Angel D. Sappa, and Boris X. Vintimilla. Chapter 9 - deep learning-based vegetation index estimation. In Arun Solanki, Anand Nayyar, and Mohd Naved, editors, *Generative Adversarial Networks for Image-to-Image Translation*, pages 205–234. Academic Press, 2021.

[19] Convolution — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Convolution&oldid=1099917634`. Accessed: 2022-08-19.

[20] Wikipedia contributors. Neural network — Wikipedia, the free encyclopedia, 2022. [Online; accessed 19-August-2022].

[21] Hiromu Yakura, Shinnosuke Shinozaki, Reon Nishimura, Yoshihiro Oyama, and Jun Sakuma. Malware Analysis of Imaged Binary Samples by Convolutional Neural Network with Attention Mechanism. In *CODASPY '18: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 127–134, March 2018.

[22] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, 2010.

[23] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. 10.48550/arXiv.1703.10593, 2017.