ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Style Transfer with Generative Adversarial Networks

Elaborato in
Machine Learning

Relatore
Prof. Davide Maltoni

Presentata da
Gabriele Graffieti

Seconda Sessione di Laurea
Anno Accademico 2017 – 2018

To all children who have ever felt different.

Abstract

In recent years, substantial advancements in machine learning have led us in the so-called *deep learning revolution* [1]. Tasks that were considered impossible to be done by machines are now the foundation of many intelligent objects that surround us. This revolution has been possible thanks to the increasing complexity of learning models, which have been developed to resolve many different tasks, from object detection to automatic translation.

One of the most fascinating models is *generative adversarial network* (GAN), proposed by Goodfellow *et al.* in 2014 [2]. This model, as its name suggests, is able to generate new data that closely resemble the examples used during training. The development of GANs has made possible one of the most challenging problems in computer vision: image-to-image translation. Just as language translation seeks to translate one sentence from one language to another without altering the meaning, image-to-image translation aims to map an image from one domain to another, without losing information about the content. As an example, a photograph taken at night can be translated to day conditions, even if a real image of the same scene during daytime does not exist.

Another task where GANs have demonstrated their potential is style transfer. Style transfer is the ability to merge two images together, keeping the content of the first picture applying the style of the second. Style transfer is nowadays mainly used for creating works of art, hence GANs are not only used by computer scientist, but also adopted by artists. For this reason, GANs used for artistic purposes are often renamed *creative adversarial networks* (CAN) [3].

This dissertation is focused on trying to use concepts from style transfer and image-to-image translation, not only to create art, but also to address some open problems in computer vision. In particular, the focus of experiments is concentrated on defogging. Defogging (or dehazing) is the ability to remove fog from an image, restoring it as if the photograph was taken during optimal weather conditions. The task of defogging is of particular interest in many

fields, such as surveillance or self driving cars.

In this thesis an unpaired approach to defogging is adopted, trying to translate a foggy image to the correspondent clear picture without having pairs of foggy and ground truth haze-free images during training. This approach is particularly significant, due to the difficult of gathering an image collection of exactly the same scenes with and without fog.

Many of the models and techniques used in this dissertation already existed in literature, but they are extremely difficult to train, and often it is highly problematic to obtain the desired behavior. Our contribute was a systematic implementative and experimental activity, conducted with the aim of attaining a comprehensive understanding of how these models work, and the role of datasets and training procedures in the final results. We also analyzed metrics and evaluation strategies, in order to seek to assess the quality of the presented model in the most correct and appropriate manner. In doing so, we focused on the aforementioned problem of defogging, trying to utilize the knowledge we have acquired to address a challenging research problem.

First, the feasibility of an unpaired approach to defogging was analyzed, using the cycleGAN model [4]. Then, the base model was enhanced with a cycle perceptual loss, inspired by style transfer techniques. Next, the role of the training set was investigated, showing that improving the quality of data is at least as important as the utilization of more powerful models. Finally, our approach is compared with state-of-the art defogging methods, showing that the quality of our results is in line with preexisting approaches, even if our model was trained using unpaired data.

Sommario (Italiano)

Negli ultimi anni, enormi avanzamenti nel campo dell'intelligenza artificiale, e in modo più specifico nel machine learning, ci hanno proiettato nella cosiddetta *rivoluzione del deep learning* [1], dove problemi che sembravano impossibili da risolvere in modo automatico, sono ora la base di molti degli oggetti intelligenti che ci circondano. Questa rivoluzione è stata possibile grazie alla complessità sempre maggiore dei modelli di apprendimento automatico, i quali sono stati sviluppati per risolvere compiti molto diversi, come il riconoscimento di oggetti o la traduzione automatica.

Uno dei modelli più interessanti è detto *generative adversarial network* (rete avversaria generativa, GAN) ed è stato proposto da Ian Goodfellow *et al.* nel 2014 [2]. Tale modello, come il nome suggerisce, ha la proprietà di poter generare nuovi dati, i quali sono indistinguibili dagli esempi usati durante il suo addestramento. GANs hanno reso possibile la soluzione di uno dei problemi più impegnativi nell'ambito della computer vision, ovvero la traduzione di un'immagine in un'altra. Allo stesso modo della traduzione tra due linguaggi, che cerca di tradurre una frase da un linguaggio ad un'altro senza alterarne il significato, la traduzione tra immagini cerca di mappare un'immagine da un certo dominio ad un'altro, senza perdere informazioni sul contenuto. Ad esempio, è possibile tradurre una fotografia scattata di notte, come se fosse stata scattata di giorno, anche se non esiste alcuna immagine della scena ripresa durante il giorno.

Un altro problema in cui i modelli avversari generativi hanno mostrato tutto il loro potenziale è lo *style transfer*. Esso consiste nel creare una nuova immagine, non esistente, che mantiene il contenuto di una certa immagine, e lo stile di un'altra. Oggigiorno lo style transfer è sopratutto usato per creare nuove forme d'arte, e per tale motivo, le reti GAN non sono utilizzate soltanto da informatici, ma anche da artisti. Per tale motivo, i modelli GAN usati per scopi artistici sono spesso chiamati *reti avversarie creative* (*creative adversarial networks, CAN*) [3].

L'elaborato si focalizza nel cercare di utilizzare concetti derivati dallo style transfer e dalla traduzione tra immagini, non solo per creare arte, ma per risolvere alcuni problemi tutt'ora aperti in computer vision. Da un punto di vista pratico, il problema sul quale è stato deciso di focalizzare l'attenzione è quello della rimozione della nebbia da immagini (*defogging*). Tale problema è di cruciale importanza in molte applicazioni pratiche, come la videosorveglianza o la guida autonoma.

Nel risolvere tale problema, è stato adottato un approccio non accoppiato, dove un'immagine annebbiata viene tradotta nella corrispondente immagine in assenza di nebbia, senza avere coppie di immagini della stessa scena con e senza nebbia durante l'addestramento del modello. Questo approccio è particolarmente significativo per il problema del defogging, poiché è estremamente complesso raccogliere un insieme di immagini della stessa esatta scena, con e senza nebbia.

Molti dei modelli utilizzati in questa tesi sono già noti in letteratura, tuttavia essi sono estremamente complessi da addestrare, e spesso è altamente problematico ottenere il comportamento desiderato. Il contributo di questa tesi di Laurea è una sistematica attività implementativa e sperimentale, atta ad ottenere una profonda conoscenza di come tali modelli funzionano, e il ruolo del dataset e della procedura di addestramento nel risultato finale. Oltre a ciò, diverse metriche sono state esaminate, in modo da valutare la qualità del nostro modello nel modo migliore e più corretto possibile. Facendo ciò, il focus è caduto sul già menzionato problema del defogging, scelto in modo da applicare le conoscenze acquisite in uno stimolante problema di ricerca.

Prima di tutto è stato analizzata la fattibilità di un approccio non accoppiato, utilizzando un modello denominato cycleGAN [4]. In seguito, tale prototipo è stato migliorato, attraverso una *cycle perceptual loss*, ispirata da diverse tecniche di style transfer. Dopo di che, è stato investigato il ruolo del dataset di addestramento del modello, mostrando come l'utilizzo di immagini migliori sia importante quanto l'uso di modelli più complessi. Infine, il modello risultate è stato confrontato con lo stato dell'arte, mostrando una forte competitività del nostro approccio, anche se esso è stato addestrato con dati disaccoppiati.

Acknowledgments

First of all I would like to thank my parents, for considering my education and my happiness the most important thing in their lives. Thank you for investing most of your time, energy and money in order to make me able to follow my dreams. I hope to be worthy of all your sacrifices. In particular, I want to praise my father, for instilling me the germ of curiosity when I was a child. Wherever you are, I know you are proud of me.

I would also thank my second family, which is composed of my dearest friends Alfredo, Luca and Manuel, for making me smile during difficult periods, and for the wonderful moments we passed together.

This thesis would not have been possible without the constant support of my supervisor, prof. Davide Maltoni, who guide me through the entire process of the dissertation development, answering innumerable questions, discussing ideas and useful insights, and introducing me into the huge field of machine learning.

Lastly, I would like to thank all the people who freely share their knowledge over the Internet, allowing me to spend many pleasant evenings enriching my mind.

Contents

Abstract **v**

Sommario (Italiano) **vii**

Acknowledgments **ix**

Introduction **1**

1 Background **3**
1.1 Machine Learning . . . 3
1.1.1 Machine learning tasks and applications . . . 4
1.2 Computer Vision . . . 7
1.2.1 Typical tasks . . . 7
1.3 Artificial Neural Networks . . . 8
1.3.1 Artificial Neurons . . . 8
1.3.2 Multilayer Perceptron . . . 11
1.3.3 Loss Function . . . 13
1.3.4 The Backpropagation Algorithm . . . 15
1.3.5 Optimization tricks . . . 20
1.4 Deep Learning . . . 20
1.4.1 Regularization for deep learning . . . 21
1.5 Convolutional Neural Networks . . . 24
1.5.1 Building blocks . . . 25
1.5.2 Architecture . . . 26
1.5.3 Applications . . . 28

2 Generative Adversarial Networks **31**
2.1 Generative Models . . . 31
2.1.1 Maximum likelihood estimation . . . 33
2.1.2 A taxonomy of generative models . . . 33
2.2 Adversarial networks . . . 35
2.2.1 Minimax games . . . 35
2.2.2 Model architecture . . . 36

2.3 Training . . . 37
2.3.1 Training algorithm . . . 38
2.3.2 Problems . . . 40
2.3.3 Possible solutions . . . 43
2.4 The problem of evaluation . . . 45
2.4.1 Why the loss function is not enough . . . 45
2.4.2 Assessment of result quality . . . 46
2.5 Examples and applications . . . 47
2.5.1 Examples . . . 47
2.5.2 Applications . . . 48

3 Cycle-Consistent Adversarial Networks 53
3.1 Image-to-image translation . . . 53
3.1.1 Style transfer . . . 54
3.1.2 Related work . . . 56
3.2 Cycle GAN . . . 58
3.2.1 The problem of paired data . . . 58
3.2.2 Cycle consistency . . . 59
3.2.3 General structure . . . 60
3.2.4 Details . . . 63
3.3 Training . . . 66
3.3.1 Training procedure . . . 67
3.3.2 Training improvements . . . 69

4 Experiments 71
4.1 PyTorch . . . 71
4.2 Defogging . . . 72
4.2.1 Related work . . . 74
4.3 Experiment design . . . 75
4.3.1 Dataset . . . 76
4.3.2 Metrics . . . 78
4.3.3 Set-up of experiments . . . 79
4.4 Getting started: cycle defogging . . . 80
4.5 Enhancing the model: cycle perceptual loss . . . 83
4.5.1 Cycle perceptual loss . . . 83
4.5.2 Evaluation . . . 85
4.6 The role of dataset . . . 88
4.7 Comparison with state-of-the-art methods . . . 92
4.7.1 Reference-based comparison . . . 92
4.7.2 Referenceless comparison . . . 93

5 Conclusions and future works **97**
5.1 Conclusions 97
5.2 Future works 99

Bibliography **101**

Introduction

What is a real image? How we can distinguish between a photograph or a computer generated picture? What are the essential features of what we call reality? These fairly philosophical questions have not been answered yet, though even a child can easily discern real and fake images. Before the advent of machine learning computers were programmed algorithmically: a procedure instructs the machine step-by-step, often in a deterministic and predetermined manner. However, not every aspect of our reasoning can be reduced to an algorithm, simply because we still do not know all the procedures involved. Nobody told us the exact algorithm to distinguish a cat from a dog, we simply have seen many cats and many dogs, and, during our infancy, other people told us which of them were cats, and which of them were dogs. We learned from data.

The same approach is the base of machine learning. Computers are not programmed, but they learn directly from data how to perform their tasks. This paradigm is the foundation of many intelligent systems that nowadays surround us, from vocal assistants to self driving cars. However, unlike us, these models will be extremely confused if something unexpected is presented to them. As an example, if we present a photograph of a dog with wings to a classificator, probably the output class will be dog, or bird, because the model cannot use its knowledge to create new classes. Unlike us, machine learning models are not creative, they usually do not aggregate their knowledge to produce new items.

In 2014, Goodfellow *et al.* introduced generative adversarial networks (GAN) [2], a machine learning model that can generate new data similar to the examples used during training. GANs learn the intrinsic features of the data and the data distribution, and, starting from a noise vector, they produce a totally new sample, which is indistinguishable from the real data. In order to perform well, GANs have to correctly answer the questions reported at the beginning of this chapter. GANs, like us, learn, somehow, the difference between real and fake data, and use this knowledge to produce realistic samples.

In the last few years, GANs have been enhanced in order to carry out increasingly complex tasks, such as style transfer and image-to-image translation. Style transfer is the ability to mimic the style of an image, keeping unchanged the content of the original picture. Given a photograph, machine learning models can apply the style of a portrait, producing works of art that never existed before. Computers have become creative.

Image-to-image translation is an even more complex task than style transfer. Images are translated from one domain to another, such as from summer to winter, or from day to night. A particular model derived from GAN and named cycleGAN [4] has shown impressive results in image-to-image translation, using an unpaired approach. The model is trained with two collections of images, representing the two domains that have to be translated. CycleGAN learns the most important features that distinguish the two domains, and it finds out a mapping directly from data, with the aim of correctly generating an image in one domain, given a photograph in the other.

The full potential of these creative models has not yet been explored; thus, the aim of this dissertation is to deeply understand these approaches and techniques, and apply them to an open research problem: removing fog from images. In chapter 1 a brief background about machine learning and neural networks will be covered. In chapter 2 and chapter 3 generative adversarial networks and cycleGAN are examined in details, with a particular focus on the learning procedure. In chapter 4 results of experiments on defogging are reported, and compared with the state-of-the-arts methods. Finally, in chapter 5, conclusions will be drawn, and future work directions suggested.

1

Background

> *Science is like a hungry furnace that must be fed logs from the forest of ignorance that surrounds us. In the process, the clearing we call knowledge expands, but the more it expands, the longer its perimeter and the more ignorance come to view.*
>
> – Matt Ridley, *Genome* (1999)

In this chapter a brief introduction to machine learning is provided. The goal of the following sections is to introduce the reader to the principal topics and techniques of machine learning, and present the recent developments in the field.

1.1 Machine Learning

Learning is commonly described as the acquisition or modification of knowledge and behaviors, as a result of interaction with the environment. This ability is possessed even by the simplest of animals and by some plants [5]. Learning can be achieved through education, experience, training or the processing of acquired data, in order for it to be organized it more generally or to infer new knowledge.

How learning works has been a mystery for a long time, and only in the last century, with the development of psychology, cognitive science and neuroscience, have some of the processes involved in learning come to light.

Since the advent of computers, producing machines capable of learning in a human-like manner has been one of the most compelling task in computer

science, and a great example of multidisciplinary research. It has also led to the creation of completely new subjects, such as *artificial intelligence* or *computational neuroscience*. The study of computational modeling of learning constitutes the foundation of machine learning.

Since the dawn of artificial intelligent research, many problems marked as difficult to solve by humans have been solved by machines, especially in the field of optimization. Surprisingly, many problems which are easy to solve by people, such as classifying or detecting objects in an image, proved to be extremely difficult for computers to achieve. This phenomenon can be explained by the lack of formalism for the aforementioned problems. In fact, it is nearly impossible to produce a classical algorithm that can detect components in an image when executed by a machine. One of the most effective way to produce such behavior is t train a computer in the same manner as children are instructed to recognize common object during their infancy.

Overall, machine learning can be described as a field of study aimed to give machines the ability to learn from data, without being explicitly programmed. In this sense, learning is formalized by the famous statement by Tom M. Mitchell "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E*" [6]. The importance of this definition is twofold: it describes machine learning in a formal way, and it emphasizes the role of performance evaluation.

1.1.1 Machine learning tasks and applications

The learning process of a *learning model* is commonly composed of two phases: the *training phase*, when the model actually learn from the given data, and the *evaluation phase*, when the performance of the model is assessed. From the definitions given in section 1.1 emerges that machine learning relies heavily on data. Data can be arranged in any form, from numbers to sequences. Usually the data is organized in a *dataset*, which is often split in three independent collections [7]:

- **Training set**: contains the data used for training the model in the training phase.
- **Test set**: contains items not present in the trainig set (the model does not see the elements of the test set during training) and it is used for evaluating the performance after the training phase.

- **Validation set**: contains data used for *hyper-parameters* validation and tuning.

Normally the training set contains a larger amount of data than the other two. A rule-of-thumb ratio of the sets dimensions is: 60% of the data go to the training set, and 20% to the test and validation set, respectively. However, if the number of samples is small, utilizing a subset of them for validation alone can cause unreliable estimations of the performance of the model. In these cases a common approach is to use *cross validation* [8].

Tasks

Historically, machine learning tasks are divided into three broad categories, based on the nature of the data, the presence of additional information, the nature of feedback given to the model and the nature of the task to accomplish [7].

- **Supervised learning**: in this approach the model infers a function from *supervised* or *labeled* data. The training set consists of input-output pairs, and the goal of the model is to infer the correct mapping between the inputs and the outputs. The inferred function has to be *general* in order to yield the correct output from an unseen input.
- **Unsupervised learning**: the data do not contain any additional information, and the goal of the model is to learn how to analyze the data, in order to find patterns or structures. An unsupervised approach can be the goal of the model (e.g. in the case of *clustering*), or an intermediate data analysis step.
- **Reinforcement learning**: in this case the learning model interacts directly with the environment, and learns from the consequences of its actions. The data consists of information about the environment and the set of available actions. When an action is performed, the RL-agent receives a reward, which indicates how positive was the action's outcome was. The goal of an RL-agent is to maximize the accumulated reward over time.

These categories are partially overlapped, i.e. between supervised and unsupervised learning lies the category of *semi-supervised learning*, where some (often many) of the entries in the training set are not labeled, or some labels are not correct.

Another category of machine learning tasks that is gaining interest in recent

years is *active learning*. In this approach, unlabeled data is abundant and labeling them is possible, but expensive in terms of time or resources. In this scenario, a learning model can actively query some external entity for the labels of a subset of data. The aim of the active learning agent is to analyze the data and seek the most suitable items that require labeling by the external entity, in order to infer the most general input-label map with the lowest amount of labeled data.

Applications

Another categorization of machine learning tasks arises when considering the desired output of the learning system [7]:

- **Classification**: the input is divided in two or more classes, and the goal of the learning-agent is to produce a model which assigns to an unseen input one or more of these classes. This is a typical application of supervised learning, when the training data is labeled with the correspondent class.
- **Regression**: the output of the model has to be continuous rather than discrete (e.g. the weight of a person given her/his height). As well as classification, regression is mainly accomplished through supervised learning.
- **Clustering**: the input data has to be divided into groups. The groups are usually not known beforehand, making clustering a typical example of unsupervised learning.
- **Density estimation**: finds the distribution of input data in a defined space. The inferred distribution can be used for statistical analysis of the data, or for generating artificial data that closely resemble the real one.
- **Dimensionality reduction**: simplify the data, learning a representation into a lower-dimensional space.
- **Representation learning**: in many machine learning models, raw data has to be pre-processed in order to extract significant features, which are provided as input to the model. Representation learning aims to learn how to automatically extract those features from raw data. Many of the machine learning models used nowadays use representation learning techniques to extract the necessary feature to solve the problem from raw data.

1.2 Computer Vision

Computer vision is an interdisciplinary field that deals with how machines can be made to acquire high-level understanding from digital images or videos. It also seeks to automate tasks that the human visual system can do [9]. This image understanding can be seen as the disentangling of symbolic information from image data, using contributions from physics, biology, geometry, statistic and machine learning.

The representation of visual data can widely vary widely according to different applications, ranging from simple RGB images, sequences of images, views from multiple cameras or multi-dimensional data from a medical scanner.

Since the majority of information that we process are acquired through the sight, it is a little surprise that many machine learning models are used for computer vision. Some of these models are directly related to their biological counterpart, such as *convolutional neural networks*, which will be discussed in section 1.5.

1.2.1 Typical tasks

Computer vision is a wide and active research area, and for this reason is commonly divided into subfields. Some of the most important fields are:

- **Recognition**: the classical problem in computer vision. The goal is to determine whether or not the the image contains some specific object, such as a face.
- **Motion analysis**: a sequence of images is processed, in order to produce an estimate of the velocity at each point in the image. A typical example of motion analysis is tracking, where an object is followed throughout the image sequence.
- **Scene reconstruction**: given one or more images (or a video) of the same scene, the objective is to reconstruct the 3D model of the scene.
- **Image restoration**: the aim is the removal of noise from images. Noise can be caused by movement, blur, occlusions or atmospheric phenomena, such as fog or rain.

1.3 Artificial Neural Networks

In machine learning, artificial neural networks (ANNs) are computing systems inspired by the *biological neural networks* that constitute the brains of animals [7]. Initially artificial neural networks were developed to solve problem in a general way, as the human brain does. However, over time, the attention shifted to more specific tasks, often departing from the initial biology-inspired systems [10]. Artificial neural networks have been used on a wide variety of tasks, from speech recognition to medical diagnosis.

ANNs are generally composed of interconnected units called *neurons*, which send signals to each other through their connections. Each connection has a numeric property, called *weight*, that can be tuned during the training phase, making ANNs capable of learning from data. Following this reason an artificial neural network can be seen as the function $f : X \to Y$, which maps some input $x \in X$ to some output $y \in Y$. f is characterized by learnable parameters θ_f, which represent the weights of the connections.

1.3.1 Artificial Neurons

Neurons used in artificial neural networks were initially developed taking inspiration from the biological neurons present in the brain of most animals. A neuron (or nerve cell) is a specialized cell that receives, processes and transmits information through electrical and chemical signals [11]. These signals between neurons occur via connections called *synapses*. Neurons can connect to each others to form neural circuits, or neural networks. Taking inspiration from biological neurons, Frank Rosenblatt developed the concept of the *perceptron* [12]. A perceptron is a mathematical abstraction of a biological neuron and, at the same time, a binary classifier. As actual neurons, the perceptron can be fed with more than one input source. It receives an input vector x consisting of n elements, and produces a single output y (see Figure 1.1). A perceptron is characterized by a set of n weights, one for each input, and a threshold. In the classical model, the output of the perceptron is binary, and is positive if the weighted sum of the inputs is greater than the threshold:

$$y = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} w_i x_i \leq threshold \\ 1 & \text{if } \sum_{i=1}^{n} w_i x_i > threshold \end{cases} \tag{1.1}$$

The learnable parameters are the weights and the threshold. By varying them, different models of decision-making can be obtained, according to the problem. Making a parallel with a biological neuron, the input vector x represents signals

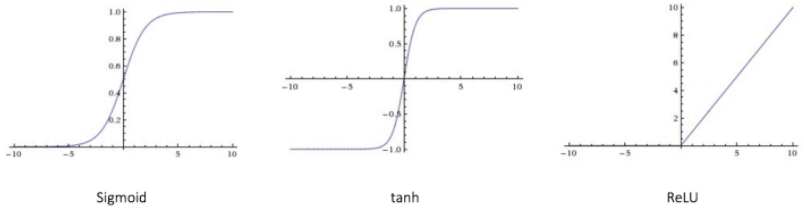


Figure 1.2: Some of the most used activation functions.

collected by the synapses of the neuron and the weights w represent the the modulation (e.g. amplification/attenuation) operated by each synapse to the corresponding input.

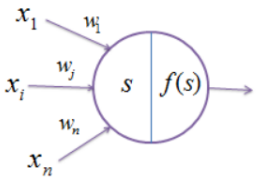


Figure 1.1: A perceptron.

The perceptron is also a binary classifier: suppose that the inputs can be distinguished in two classes, a value of y equal to zero can represent the first class, and a value equal to one, the second class.

The classic perceptron model is the easiest one, and starting from it many more neural models were developed using it as a starting point. The output of a classic perceptron is strictly binary, and this can be a disadvantage during learning. In fact, a small change in the weights or the threshold may switch the output of the neuron from zero to one or vice versa. In order to overcome this limitation, different activation functions were evolved. The most famous of these is the *sigmoid function*:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{1.2}$$

The sigmoid function is continuous, derivable, and its image vary from zero to one. This function is suitable during learning, because small changes in the perceptron's parameters are reflected in small changes in the output. A perceptron which uses the sigmoid function is commonly called a *sigmoid neuron*. Using a sigmoid function, the output of a neuron becomes:

$$y = \frac{1}{1 + \exp(-\sum_{i=0}^{n} w_i x_i - t)}. \tag{1.3}$$

where t is the threshold.

In some cases, the sigmoid function is replaced by the hyperbolic tangent (*tanh* or *softmax*), so the output of the neuron varies from -1 to +1.

Another widely used activation function is the *rectifier* (ReLU) [13]:

$$f(z) = z^+ = max(0, z). \tag{1.4}$$

The rectifier, apart from being biologically inspired, has shown better performance during the learning phase, especially in deep networks [14], and for this reason, is the most used activation function for deep models [15]. A graphical representation of the most common activation function is shown in Figure 1.2.

Learning

As stated earlier, a perceptron, regardless of the activation function used, can be considered a simple binary classifier, and a learning strategy can be adopted to find the optimum combination of parameters. Given a binary labeled training set, the error of the perceptron can be expressed as:

$$E(\boldsymbol{w}, t) = \frac{1}{2} \sum_{j=1}^{m} \left(f \left(\sum_{i=1}^{n} (w_i \cdot x_i^{(j)}) - t \right) - l^{(j)} \right)^2 \tag{1.5}$$

where:

- $\boldsymbol{w}$ is a vector containing the weights of the perceptron.
- t is the threshold associated with the perceptron.
- m is the number of elements in the training set.
- f is the activation function.
- $\boldsymbol{x}^{(j)}$ is the j-th element of the training set.
- $l^{(j)}$ is the desired result (label) associated with the j-th element of the training set.
- n is the number of elements in the input vector $\boldsymbol{x}$.

To simplify the notation, the threshold t can be included in the weight vector ($w_0 = t$) and associated with an input $x_0 = 1$; thus Equation 1.5 becomes:

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{j=1}^{m} \left(f \left(\sum_{i=0}^{n} (w_i \cdot x_i^{(j)}) \right) - l^{(j)} \right)^2 \tag{1.6}$$

Hence, the objective is to find the optimal combination of values of w which minimize the error. Let

$$W_{x^{(j)}}(\boldsymbol{w}) = \left(f \left(\sum_{i=0}^{n} (w_i \cdot x_i^{(j)}) \right) - l^{(j)} \right)^2 \tag{1.7}$$

the error of the perceptron for the input vector $\boldsymbol{x}^{(j)}$. An approach based on gradient descent can be applied, in order to update w to minimize the error. The gradient of $W_{x^{(j)}}(\boldsymbol{w})$ ($\nabla W_{x^{(j)}}(\boldsymbol{w})$) indicates the direction of maximum growth of $W_{x^{(j)}}(w)$ with respect to w. The opposite direction ($-\nabla W_{x^{(j)}}(\boldsymbol{w})$), on the contrary, indicates the direction of faster decrease of the function, with regards to w. If the weights are updated following the opposite direction of the gradient, the function will be minimized. Using an iterative method, it is possible to get closer and closer to the minimum point, which correspond to the optimal solution of the optimization problem stated as:

$$s^* = \min_{w} \sum_{j=1}^{m} W_{x^{(j)}}(\boldsymbol{w}) \tag{1.8}$$

The update of the weighs occurs with the following procedure:

$$w^{j+1} = w^j - \eta \nabla W_{x^{(j)}}(\boldsymbol{w}) \qquad \forall j \in \{1, \ldots, m\} \tag{1.9}$$

where:

- w^j is the vector of the weights at step j
- η is the *learning rate*, which is the displacement step used to update w. If it is too large, the optimum may be impossible to reach, whereas if it is too small, it can greatly slow down convergence. The learning rate is a typical example of a hyper-parameter.

Thanks to Equation 1.9, a perceptron can learn how to classify data simply by being exposed to the data itself. However, the perceptron can only solve binary classification problems, and the classification is *linear*, which means that it may not perform well if the patterns are not linearly separable.

In spite of that, the perceptron is the fundamental building block of most of the neural network models, currently used to solve many extremely complex tasks in the field of machine learning.

1.3.2 Multilayer Perceptron

Following the biological inspiration that led to the development of artificial neurons, it seemed natural to try to connect them together, in order to form neural networks similar to those present in the brain, seemed natural. The first example of an artificial neural network was the *multilayer perceptron* (MLP). In a multilayer perceptron the neurons are organized in *layers*, and the outputs of the neurons of one layer are the input to the next layer. The minimum

number of layers is three, one input layer, one or more hidden layers and one output layer.

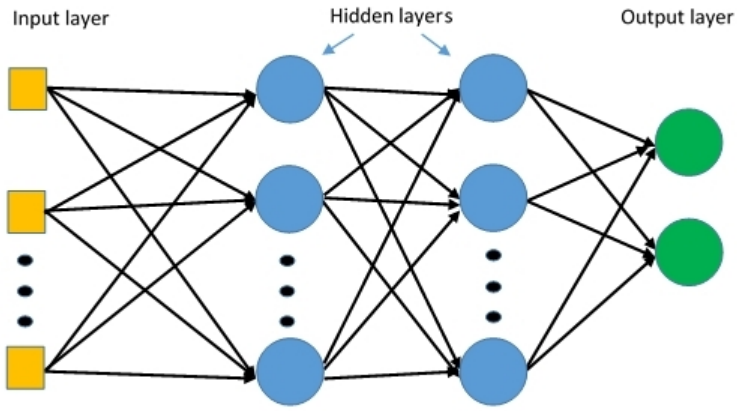


Figure 1.3: A multilayer perceptron with four layers.

A multilayer perceptron is an example of a *feedforward* neural network. These kinds of model can classify patterns in more than two classes, and separate them non-linearly. Moreover, a feedforward neural network with only one hidden layer and a finite number of neurons has been demonstrated to be a *universal approximator* [16]. This means that a simple neural network can represents a wide variety of interesting functions, when given appropriate parameters. This theoretical result is one of the reasons for the widespread adoption of neural networks for a wide range of different problems.
A multilayer perceptron usually has the following properties:

- In an MLP the number of input neurons should be equal to the number of dimensions of the input, i.e. if the input is a 3-dimensional vector, the network should have three neurons in the input layer.
- A MLP should have as many perceptrons in the output layer as the number of classes. The value produced by an output neuron represents the probability that the current input will be classified with the class associated to the neuron.
- A MLP should associate an input to the class represented by the output perceptron by computing the highest value among its peers.
- In a MLP all the neurons should have the same activation function.
- In a MLP every perceptron can only be connected to perceptrons in the next layer.

Figure 1.3 shows a common example of MLP architecture, composed of two hidden layers. In the example, the MLP acts as a binary classifier (there are only two neurons in the output layer). The perceptrons in the input layers usually do not compute any function, but simple propagate the input to the first hidden layer.

Besides MLP, many other types of neural networks exist, which differ from each other in terms of their architecture and the organization of neurons. One model worth mentioning here is the *recurrent neural network* (RNN), in which cycles are present between units of neurons. RNNs have gained popularity in recent years as a result of their extensive use with time sequences.

1.3.3 Loss Function

In section 1.3.1 it is stated that, in order to be able to learn, a perceptron should take classification error into consideration, and try to minimize it. Equation 1.5 is an example of a loss function.

Generally, a loss function (or cost function) is a function that maps an event or values onto a real number, which represents a certain *cost* associated with the event. In many optimization problems, the goal is to minimize the cost function, or, in a parallel way, maximize the objective function (which is the negative of the cost function). A loss function is often used in statistics as a parameter estimation of a distribution, computing the difference between real and estimated data. In classification, a loss function is usually a penalty for an incorrect classification, and it grows in accordance with the classification error.

The choice of the most suitable loss function is a problem of crucial importance, and the differences in performance and accuracy of the same model trained with different loss functions may be broad.

Application of loss function is not limited to neural networks, on the contrary, it is a common concept in various machine learning models. The most commonly used loss functions are described below.

Squared loss

The squared loss (or mean square error) is formulated as:

$$\mathcal{L}(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2. \tag{1.10}$$

where:

- $y^{(i)} \in Y$ are the observed data.
- $\hat{y}^{(i)} \in \hat{Y}$ are the predictions of the model.
- n is the number of elements in the training set.

The target of the squared loss function is to minimize the residual sum of squares. This loss function suffers from slow convergence when used with neurons that use the sigmoid function. It also tends to penalize outliers excessively, and if there are many of them present in the training data, the convergence may be difficult. Hence, squared loss is more commonly used for regression tasks, rather than classification.

Mean absolute error

The mean absolute error, or L1 loss, is formulated as:

$$\mathcal{L}(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} |y^{(i)} - \hat{y}^{(i)}|. \tag{1.11}$$

This loss is similar to squared loss, but it penalize outliers less, so it is preferred in this sense. However, the mean absolute loss has the same gradient for every point, regardless of the distance to the optimum, hence it is more difficult to find the optimal solution.

Cross entropy loss (log loss)

Cross entropy loss is commonly used in binary classifiers (labels are 0 or 1), and it is formulated as:

$$\mathcal{L}(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} [-y^{(i)} \log(\hat{y}^{(i)}) - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]. \tag{1.12}$$

Cross entropy loss is closely related to the *Kullback-Leibler divergence*, which measures the difference between two distributions. If the cross entropy loss is large, its means that the distribution of real data is different from what has been predicted. Cross entropy loss is usually preferred to squared loss, especially in deep models, due to its faster convergence. Cross entropy loss can be generalized for multiclass problems, in *categorical cross entropy* [8].

1.3.4 The Backpropagation Algorithm

section 1.3.1 describes how a single perceptron can learn the optimal set of parameters directly from data. Does a similar algorithm exists for training a neural network, potentially composed of milions of neurons? Ever since the 1960s, research had been carried out to provide possible answers to this question [17] and by 1970 the first version of the backpropagation algorithm was developed by Seppo Linnainmaa [18]. However, the algorithm remained mostly unknown, mainly due to the limited computing power of machines at the time. Backpropagation gained recognition after the publication by Rumelhart *et al.* in 1986 [19].

The backpropagation algorithm is an optimization algorithm, which minimizes the loss function by changing the parameters of the model. Since its introduction, the backpropagation algorithm has been the *de facto* standard for training neural networks, especially in recent years, when it benefits form cheap and powerful GPU-based computer systems.

The backpropagation algorithm repeats a two phases cycle: propagation and weight update. In the first phase, an input vector is propagated through the network, layer by layer, until it reaches the output layer. The output of the network is then compared to the desired output, using a derivable loss function. The resulting error is calculated for each neuron in the output layer, and then propagated from the output layer back to the input layer (as the name backpropagation suggests), until each neuron has an associated error value. After that, those errors are used to calculate the gradient of the loss function. In the second phase, the weights are modified by an optimization procedure, according to the gradient of the loss function.

Stochastic gradient descent

The backpropagation algorithm is an example of gradient descent optimization. The goal of the procedure is to find a minimum in a function, changing its parameters according to the opposite direction indicated by the gradient.

Let $F(\cdot)$ be a continuous function defined by parameters θ. Let θ_n be the parameters at iteration n. If we want to minimize $F(\theta)$ using gradient descent, the parameters of the next step are updated as follows:

$$\theta_{n+1} = \theta_n - \eta \nabla F(\theta_n) \tag{1.13}$$

In most machine learning models the function to be minimized is the loss function. As described in subsection 1.3.3, if the training set is composed of

m items, the loss function usually assumes the form:

$$\mathcal{L}(Q(X,\theta),Y) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}_i(Q(x^{(i)},\theta),y^{(i)}). \tag{1.14}$$

where

- $x^{(i)} \in X$ is an element of the training set.
- $y^{(i)} \in Y$ is the label associated to the i-th item of the training set.
- $Q(X,\theta)$ is the output of the neural network.
- $\mathcal{L}_i(Q(x^{(i)},\theta),y^{(i)})$ is the per-example loss.

Updating the parameters of such a model with gradient descent requires computing:

$$\theta_{n+1} = \theta_n - \eta\frac{1}{m}\sum_{i=1}^{m}\nabla_\theta\mathcal{L}_i(Q(x^{(i)},\theta),y^{(i)}). \tag{1.15}$$

which has a computational cost of $O(m)$, thus computing the gradient can be prohibitive if the training set contains billions of examples.

In order to overcome the problem, the gradient can be computed using a small subset of samples, so that the optimization is based on an estimation of the gradient rather than the actual gradient. This approach is called *stochastic gradient descent* (SGD). The parameters are updated considering a set of m' examples (*minibatch*), drawn uniformly from the training set. The number of sampled example is usually much smaller than the training set. The update to the parameters is performed as:

$$\theta_{n+1} = \theta_n - \eta\frac{1}{m'}\sum_{i=1}^{m'}\nabla_\theta\mathcal{L}_i(Q(x^{(i)},\theta),y^{(i)}). \tag{1.16}$$

One notable case is when $m' = 1$. In this particular case, the update to parameters is performed after every iteration.

SGD has made the training of models through gradient descent on very large datasets possible. Indeed, for a fixed model size, the cost per SDG update does not depend on the training set dimension. It can consequently be stated that the computational cost of one step of SGD is $O(1)$, as a function of m [10].

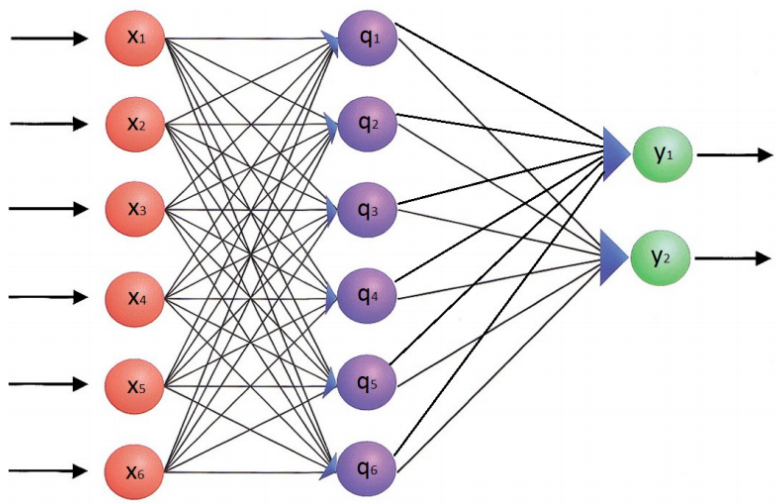


Figure 1.4: A multilayer perceptron

The algorithm

Consider the neural network displayed in Figure 1.4. Assuming that a mini-batch of dimension 1 is used, so for every iteration a new input is considered. Generalizing, the output of the hidden layer's neutrons are $q_1, q_2, \ldots, q_m$ and can be computed as:

$$q_i = f(v^i x) \qquad \forall i \in \{1, \ldots, m\} \tag{1.17}$$

Then, $y_1, y_2, \ldots, y_n$ are the output of the network, and can be computed as:

$$y_i = f(w^i q) \qquad \forall i \in \{1, \ldots, n\} \tag{1.18}$$

where:

- w^i is the weights vector associated with the $i-th$ perceptron in the output layer.
- v^i is the weights vector associated with the $i-th$ perceptron in the hidden layer.
- f is the activation function.

Let $\mathcal{L}_x(y)$ be the loss on the current input x. The loss can be seen as the error made by the network when it is fed with x. It is possible to begin decomposing this error by calculating the error of the output layer's neurons. This can be expressed as:

$$e_i = \frac{\partial \mathcal{L}_x(y)}{\partial y_i} \qquad \forall i \in \{1, \ldots, n\} \tag{1.19}$$

which represents how much the loss varies due to small changes in y_i. Since the only way to modify y_i is to change the weights w^i or the hidden layer output q, we can rewrite Equation 1.19 as:

$$e_i = \frac{\partial \mathcal{L}_x(y)}{\partial w^i q} \qquad \forall i \in \{1, \dots, n\} \tag{1.20}$$

which for the chain rule of calculus becomes:

$$e_i = \frac{\partial \mathcal{L}_x(y)}{\partial y_i} \cdot \frac{\partial f(w^i q)}{\partial w^i q} = \frac{\partial \mathcal{L}_x(y)}{\partial y_i} f'(w^i q) \qquad \forall i \in \{1, \dots, n\} \tag{1.21}$$

Now we want to calculate the error of the hidden layer, which is equal to:

$$r_i = \frac{\partial \mathcal{L}_x(y)}{\partial q_i} \cdot \frac{\partial f(v^i x)}{\partial v^i x} = \frac{\partial \mathcal{L}_x(y)}{\partial q_i} f'(v^i x) \qquad \forall i \in \{1, \dots, m\} \tag{1.22}$$

The first term of the equation can be rewritten, using the chain rule of calculus as:

$$\frac{\partial \mathcal{L}_x(y)}{\partial q_i} = \sum_{j=1}^{n} \left(\frac{\partial \mathcal{L}_x(y)}{\partial y_j} \cdot \frac{\partial f(w^j q)}{\partial w^j q} \cdot \frac{\partial w^j q}{\partial q_i} \right) = \sum_{j=1}^{n} \left(e_j \cdot \frac{\partial w^j q}{\partial q_i} \right) \tag{1.23}$$

And since $\frac{\partial w^j q}{\partial q_i} = {w_i}^j$ what is obtained is that:

$$r_i = f'(v^i x) \sum_{j=1}^{n} e_j {w_i}^j \qquad \forall i \in \{1, \dots, m\} \tag{1.24}$$

In case of a neural network with more than one hidden layer, the errors of the others layers are calculated by repeating Equation 1.24, treating the next hidden layer as if it were the output layer.

After the error backpropagation, the weights of the network are updated. Starting from the output layer, a small change in a weight produces the following variation on the loss:

$$\frac{\partial \mathcal{L}_x(y)}{\partial w^i} = \frac{\partial \mathcal{L}_x(y)}{\partial y_i} \cdot \frac{\partial f(w^i q)}{\partial w^i q} \cdot \frac{\partial w^i q}{\partial w^i} = e_i \cdot q \quad \forall i = 1, \dots, n \tag{1.25}$$

So the weights can be updated with the following criterion:

$$w^i = w^i + \eta \cdot e_i \cdot q \qquad \forall i \in \{1, \dots, n\} \tag{1.26}$$

And, similarly, the weighs of the hidden layer can be adjusted as follow:

$$v^i = v^i + \eta \cdot r_i \cdot x \qquad \forall i \in \{1, \dots, m\} \tag{1.27}$$

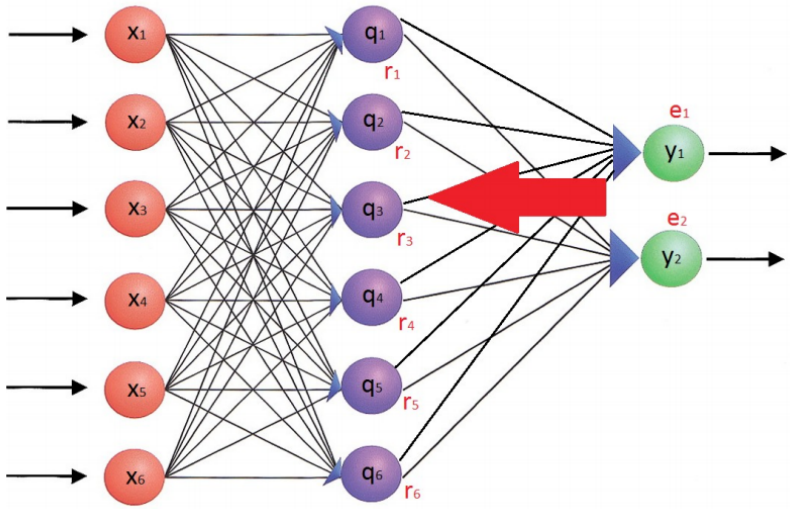


Figure 1.5: The error is back-propagated through the network, in order to update the weights accordingly.

At the end of the backpropagation operations, the weights of the network are updated in order to take a step towards the opposite direction of the gradient of $\mathcal{L}_x(y)$. Repeating this procedure for every pattern in the training set is the basic learning mechanism of neural networks.

Algorithm 1 Back-propagation algorithm

Require: H = number of hidden layers
for each input pattern x **do**
    forward propagation
    compute the error for output percetrons using Equation 1.21
    for $i = H$ to 1 **do**
        back-propagate error on the hidden layer i, using Equation 1.24
    end for
    update weight of the output perceptrons using Equation 1.26
    for $i = H$ to 1 **do**
        update weights of the hidden layer i using Equation 1.27
    end for
end for

1.3.5 Optimization tricks

Momentum

Stochastic gradient descent is the standard procedure for optimizing most machine learning models, but it can sometimes present problems. In fact, in the presence of a *ravine*, which are common around local optima, or if the function to be optimized presents many local minimums, the SGD oscillates across the slopes, making only hesitant progress towards the optimal solution [20]. The momentum method accumulates an average of the past gradients, and adds a fraction of that to the current updating vector. The name momentum derives from an analogy with physics: just as an object that rolls downhill gains momentum in the direction of the fall, so too are the parameters are pushed in the direction of previous steps. This prevents oscillation of the loss function around an optimal solution, and, moreover, momentum can move the cost function away from a local minimum.

Adaptive learning rates

In subsection 1.3.4, it is stated that the weight update occurs with a fixed learning rate. This means that every step downhill has approximately the same length. However, this behavior may not be desired. In fact, if the function to be optimized is far from the minimum, a larger step is preferable. On the other hand, when the optimal solution is near, a shorter step should be used. Indeed, the use of *learning rate schedulers* have become common in most neural network models. These schedulers decrease the learning rate as the training phase advances in order to make smaller steps when the model is reasonably near a minimum. Many methods, such as *Adagrad*, *Adadelta*, *AdaMax* or *Adam* [10], are based on using different learning rates for each parameter. The parameters which are infrequently modified are updated with larger steps, whereas parameters that are edited more frequently are updated with smaller steps. This procedure greatly improves the robustness of the SGD, and the methods cited above are used nowadays to train large-scale neural networks [20].

1.4 Deep Learning

Deep learning (DL) is a branch of machine learning, based on learning data representation using simple but non-linear hierarchical modules, which trans-

form the representation at one level into a representation at a higher, more abstract level [21].

Conventional machine learning techniques were limited in their ability to process raw natural data. In fact, constructing a pattern recognition machine required considerable domain expertise in order to design a feature extractor that transforms the raw data into a suitable feature vector. Deep learning is based on representation learning methods, which allow machines to be fed with raw data and to automatically discover the best representation needed for the actual task. The key concept of deep learning (and the reason is is *deep*), is that the representation is constructed in a hierarchical manner, often with several layers, enabling models to learn complicated concepts by building them out of simpler ones [10].

The quintessential model of deep learning is an MLP with more than one hidden layer. The first hidden layer can be seen as a feature extraction from the raw data, and each subsequent layer builds up more complex features from the output of the previous ones [10]. Nowadays, it is common to have networks with a large number of hidden layers (ten or more), which have been demonstrated to be greatly superior to classical one-hidden-layer MLPs [22].

The main reasons behind the widespread adoption of deep learning techniques in recent years are bigger datasets and fastest hardware, which supports deeper models [10]. The availability of big datasets (millions of items) have lightened the key burden of statistical estimation: generalizing satisfactorily after observing only a small amount of data. Advancements in hardware have made the training of huge models in a reasonable amount of time possible.

Various deep learning architectures such as *deep neural networks, deep belief networks, recurrent neural networks, convolutional neural networks*, etc., have been applied to a multitude of different fields, such as images, video or audio processing, bioinformatics, natural language processing and computer vision.

1.4.1 Regularization for deep learning

The training of deep models is obviously more difficult than training simpler ones. For this reason, many techniques have been developed in order to obtain better performance. The key point of regularization is reducing the generalization error of a model, while keeping its training error constant [10]. There are many regularization strategies, some of which are describe below.

Parameter penalties

Neural networks whose weights are close to zero[1] tend to be more stable and generalize better, even when they are fed with less data. In order to keep the parameters close to their initial value, a regularization term can be added to the loss:

$$\mathcal{L}(Y, \hat{Y}, \theta) = \mathcal{L}(Y, \hat{Y}, \theta) + \alpha\Omega(\theta). \tag{1.28}$$

where

- Y and $\hat{Y}$ are the expected output and the actual output of the network, respectively.
- $\alpha \in [0, +\infty)$ is a hyperparameter that weights the contribution of the regularization.
- $\Omega(\theta)$ is the penalty function, usually L1 or L2 norm.

Dataset augmentation

The more straightforward way to make a machine learning model generalize better is to train it on a larger dataset [10]. In practice, this is not always possible, even in the current age of *big data*. One way to get around this problem is to create artificial data and add it to the training set. This is particularly easy for tasks such as classification. In this case, the training set is composed of data in the form (x, y), where x is the actual data and y is the label. Applying some kind of transformation to x, (i.e. in the case of images: rotation, blurring, scaling, translation, etc.) obtaining x', and adding the pair (x', y) to the dataset is a widely adopted technique. For many other tasks, augmenting the dataset is not so straightforward. Recently, some *deep generative models* have been used to generate realistic data [23], making dataset augmentation for a wider range of problems possible.

Dropout

Dropout [24] provides a computationally inexpensive but powerful method for the regularization of a broad family of deep models. At every training step, a subset of the non-output neurons are ignored during the forward and backward

[1] In fact, we can regularize parameters to be near any point in space, and surprisingly still get a regularization effect. However, better results are obtained for a value near the optimal value. Since the optimal value is the objective of the training, zero is used as a default [10].

passes (their output is set to zero). This can be thought of as a method of *bagging* with many large models. At every step, some neurons are removed from the network, yielding a different model every time. Neurons can develop co-dependence during the training phase, leading to overfitting. Dropout offers a convenient way of preventing these dependencies from occurring, making the resulting model more general.

Batch normalization

Batch normalization [25] is considered one of the most important recent innovation in optimizing deep networks [10]. Its basic principle is really simple: in most models, the data is normalized (usually in the 0-1 range), before presentation to the model, to improve its invariance to input. Therefore, the concept is to normalize each layer's output, in addition to its input. In SGD, a mini-batch of m input vectors is used. Let o_i^j be the output of the i-th neuron of the j-th layer. The mean and the standard deviation of the output of the layer is calculated as:

$$\mu^j = \frac{1}{m}\sum_i o_i^j, \qquad \sigma^j = \sqrt{\delta + \frac{1}{m}\sum_i \left(o_i^j - \mu^j\right)^2} \tag{1.29}$$

where δ is a small positive value, imposed to avoid the indefinite gradient at $\sigma = \sqrt{z}$ with $z = 0$. Now the output of the neuron is substituted with:

$$\hat{o}_i^j = \frac{o_i^j - \mu^j}{\sigma^j} \tag{1.30}$$

Repeating this procedure for every layer of the network has the effect of normalizing the outputs of each layer, and reducing the internal covariate shift (ICS). ICS is the phenomenon wherein the distribution of inputs in a layer of the network changes due to an update in the parameters in the previous layers. This change leads to a constant shift in the underlying training problem, slowing down the convergence of the model. In addition to the normalization with batch mean and variance, batch normalization adds two more learnable parameters to every activation in order to maintain the expressiveness of the model. Thus, the output of the neuron is defined as:

$$y_i^j = \gamma_i^j \hat{o}_i^j + \beta_i^j. \tag{1.31}$$

where γ_i^j and β_i^j are the learnable parameters. Without them the model were not be able to learn even simple input transformations, such as the identity function.

1.5 Convolutional Neural Networks

Convolutional neural networks (CNN) are a class of deep, feedforward artificial neural networks, most commonly applied to images analysis. An image is usually defined in three dimensions: width (W), height (H) and number of channels (C). If we consider a fully-connected MLP, in which each input neuron is associated with one of the image's pixels, every neuron in the first hidden layer should have W×H×C weight associated to it. Even for small images (i.e 64×64×3), the first fully connected layers contains 12.288 weights for each neuron. For non-trivial architectures, the number of parameters rapidly explodes, leading to intractable models.

CNNs are one of the greatest successes of biologically-inspired artificial intelligence [10]. Indeed, the intuition that influenced the development of CNNs arose from studies of the visual cortex of mammals, conducted by Hubel and Wiesel [26]. They found that the visual cortex is composed of layers, and the first layer respond strongly to very specific patterns, such as edges or oriented bars, but hardly respond at all to more complex structures. The output of the first layer is then passed to successive layers, where simple characteristics are combined in order to produce more complex features (see Figure 1.6). Layering and the composition of simple feature are the basic mechanisms used by convolutional neural networks.

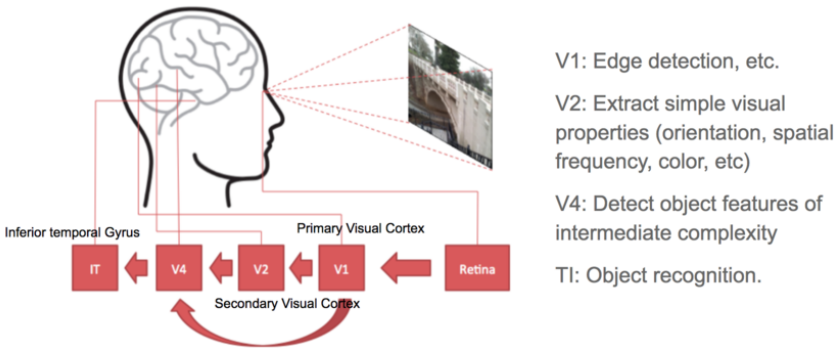


Figure 1.6: The human visual cortex system.

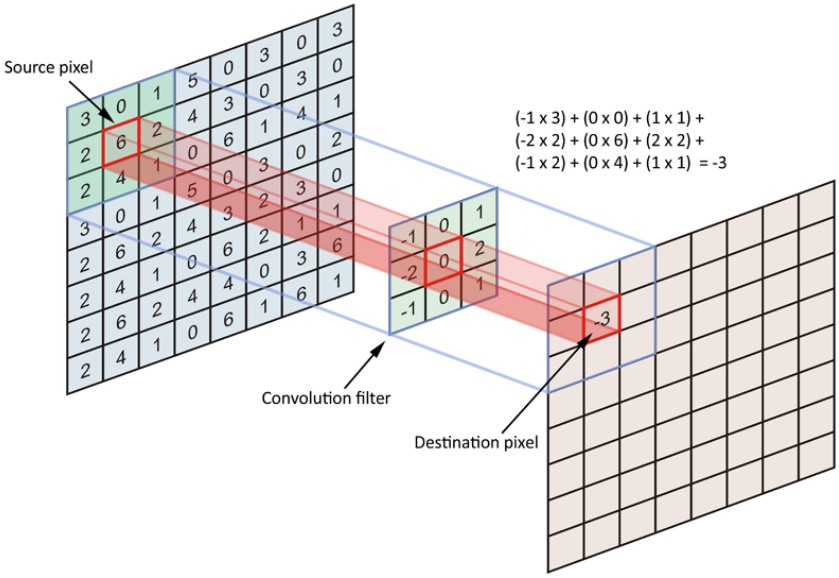


Figure 1.7: The result of a single convolution operation. The process is repeated by shifting the filter all-over the entire original image.

1.5.1 Building blocks

Convolution

The convolution operation is the core building block of CNNs. A convolution over an image requires a convolution matrix (or filter) which is moved across the image. The result of a single convolution operation is the sum of all products between the filter elements and the corresponding pixels.

Usually, the results are stored in a new image, which represents the response of the original image to the filter used. Changing the filter results in different features being extracted from the initial image. The resulting image's size depends on the size of the original image and the size of the filter. More formally:

$$w_o = w_i - w_f + 1 \qquad and \qquad h_o = h_i - h_f + 1 \tag{1.32}$$

where:

- w_o and h_o are the width and height of the image resulting from convolution.
- w_i and h_i are the width and height of the input image.
- w_f and h_f are the width and height of the filter used.

Often CNNs have to deal with 3D input, such as RGB images. In those cases, the filter is also 3D, with a depth equal to the input's number of input channels. The output of a single convolution operation is still a number. Generally, more than one filter is used in each convolution layer, with the aim of deriving different features from the same input. Thus, each convolution layer can be represented as a bank of filters, and its output is composed of k different output images (feature maps), with k = number of filters.

In a convolutional neural network the filters are automatically learned during the training phase. This means that the network learns what features are the most crucial, and how to extract them from an image.

The two most important characteristic of convolutional neural networks are the sharing of weights among neurons in the same layer, and the locality of the connection. Every neuron in a convolutional layer is connected with only a small portion of the perceptrons of the previous layers, as defined by the size of the filter. Moreover, the weights are shared among neurons in the same layer, drastically reducing the number of parameters, with respect to an MLP. As an example, in the first convolution layer, assuming 3×3 filters and an RGB input image ($C = 3$), the filter is characterized only by $3 \cdot 3 \cdot C = 27$ weights, so there are only 27 parameter to be learned, regardless the number of neurons in the layer.

Downsampling

The downsampling block is another important module of a CNN. It has the aim of reducing the dimension of the input, in order to lower the number of parameters and give the model more robustness to input changes [10]. The most commonly used downsamplig layer is *max pooling*. In the max pooling layer, the image is divided into small blocks and every block is condensed to its higher value (see Figure 1.8). Another commonly used downsamplig block is *average pooling*, when the output of a block is the average of all the values within it.

1.5.2 Architecture

Convolutional neural networks are powerful classifiers [27], mainly due to the automatic extraction of relevant feature, which had to be manually extracted before the advent of CNNs. They are extremely suitable when the dimensionality of the data is particularly high, as in images.

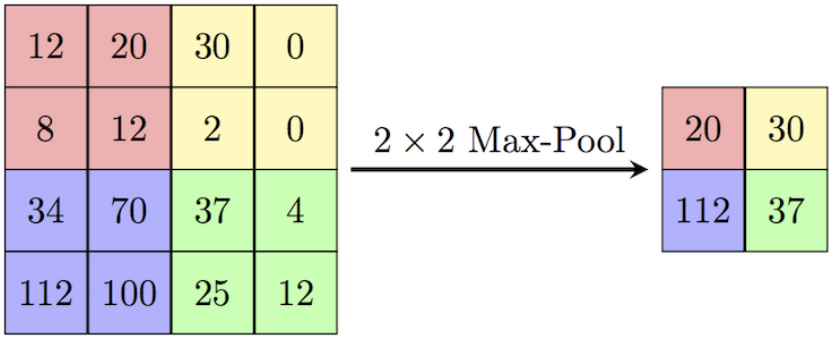


Figure 1.8: The result of a max pool operation using 2x2 blocks.

A CNN is typically composed of two modules: the feature extractor and the classifier. The input, which usually consists of RGB images, is given to the former module, which produces a feature vector of k elements. This array is then delivered to a fully connected neural network, which assigns a class to the input, relying on the extracted features.

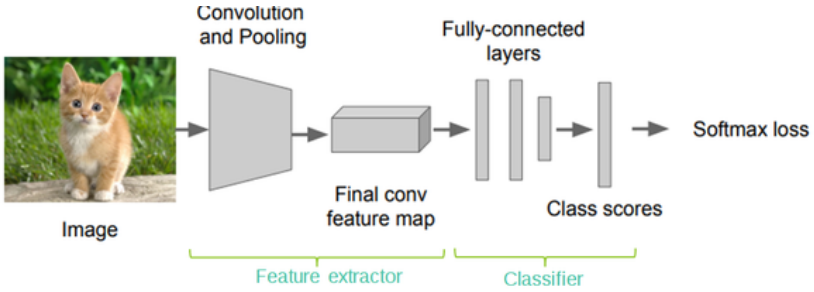


Figure 1.9: General CNN architecture, composed of a feature extractor and a classifier.

The classifier is usually a fully-connected multilayer perceptron, with as many neurons in the output layer as the number of classes. The feature extractor, by contrast, generally consists of many convolutional layers, each of which are usually followed by an activation layer (ReLU is commonly used as an activation function). Every s block of convolution plus activation, there is a downsampling layer, which reduces the dimensionality of the feature maps. The value of s depends on the network, and can even vary inside the network itself, but usually ranges from one to three. In some cases, only convolution plus activation is used [28]; on uch occasions, the CNN is commonly called a *fully convolutional network* (FCN).

The training phase of a convolutional neural network is similar to the one described in subsection 1.3.4. Backpropagation is used as in MLPs and the

weight are updated in a similar manner.

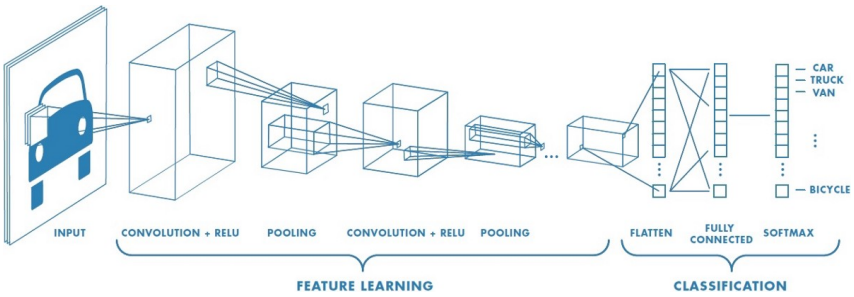


Figure 1.10: A typical CNN architecture, with a pooling layer after every convolution plus ReLU block. The total number of layers in the network is a design choice, and can widely vary from different models.

1.5.3 Applications

As stated at the beginning of this section, CNNs are widely used for image processing. This trend started in 2012, when *AlexNet* won the *ImageNet Large Scale Visual Recognition Challenge* [29]. Since then, convolutional neural networks have become the standard for image-related tasks, such as classification or object detection.

The feature extractor module can be used for *transfer learning*. A trained CNN is a powerful feature extractor: changing only the classifier and training it, keeping the former module fixed, is often much faster than training a CNN from scratch.

Another application of CNN is within a class of methods named *encoders*. In general terms, an encoder is a device that converts information from one format to another, and can be mathematically expressed as a function: $\phi : \mathcal{A} \to \mathcal{B}$ where $\mathcal{A}$ and $\mathcal{B}$ are two different spaces (e.g. image space and feature space). The inverse operation of encoding is decoding, which can be expressed as: $\psi : \mathcal{B} \to \mathcal{A}$. The purpose of encoding is to compress data into a short code in a *latent space*. If the encoding is carried out properly, the latent code should contain only the most important features of the data, ignoring noise or unimportant aspects. The latent space can also be used for data editing. For instance, if an encoder has learned how to represents concepts like a *cat* and *flying*, those codes can be mixed in the feature space in order to generate a *flying cat* after decoding, even if neither the encoder nor the decoder has ever seen a flying cat [30]. Often a single CNN acts both as encoder and as decoder. In this case, the model is called an *autoencoder*. Autoencoders are one of the

most active research topics in deep learning [10], and are predominantly used for representation learning.

2

Generative Adversarial Networks

The most interesting idea in the last 10 years in machine learning.

– Yann LeCun, *Director of AI research at Facebook*

In this chapter, a description of *Generative adversarial networks* (GANs) is provided, starting from the basic concepts that led to the development of the GAN model. Then, the procedure used to train such models is described, focusing on the differences and problems when compared to classical neural networks, and presenting some solutions to them. At the end of the chapter, the problem of evaluation is discussed , which is common to many generative models.

2.1 Generative Models

In statistical classification, as in machine learning, there are two main types of model: *discriminative* and *generative models* [8]. Given an observable variable X, and a target variable Y (i.e X represents the actual data, while Y are the labels), a discriminative model is a model of the target Y given an observation $x \in X$, symbolically $P(Y|X = x)$. A discriminative model, in brief, learns how to classify every pattern x to the correct label y. Methods like *support vector machines* (SVMs) and neural networks fall into this category.

On the other hand, generative models are statistical models of *joint probability distribution* $P(X, Y)$. A generative model learns the intrinsic distribution of

Figure 2.1: The operation of density estimation performed by a generative Gaussian model over mono-dimensional data. [31]

data, allowing it to *generate* new data according to the real data distribution (this is why these models are called *generative*). In short, a generative model, given a training set consisting of samples drawn from a distribution p_{data}, learns how to represent and estimate such a distribution, and the result is a probability distribution p_{model}, which will hopefully be similar to p_{data} [31].

It is important to report that, given the *Bayes theorem*:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \quad and \quad P(X|Y)P(Y) = P(X,Y) \tag{2.1}$$

the result is that:

$$P(Y|X) = \frac{P(X,Y)}{P(X)}. \tag{2.2}$$

Since

$$P(X) = \sum_{y} P(X, Y = y) \tag{2.3}$$

the discriminative model can be directly derived from the generative one:

$$P(Y|X) = \frac{P(X,Y)}{\sum_{y} P(X, Y = y)}. \tag{2.4}$$

Therefore, given a generative model, it is possible to derive a discriminative model. Moreover, the former contains more information than the latter, and it can be used to discover complex relationships between X and Y, such as in the case of *multi-modal* outputs (given an input $x \in X$, more than one element of Y is correct; i.e. x is a frame in a video and y is the predicted next frame). However, for classification purposes, discriminitavive models are usually preferred, due to their higher accuracy, if they are trained with large datasets [32].

2.1.1 Maximum likelihood estimation

In the previous section, it was stated that generative models produce a probability distribution p_{model} which should resemble the real distribution of data p_{data}. Likelihood provides a metric for comparing two probability distributions, and the maximum likelihood estimator can be used to find the optimal parameters of the generative model.

The likelihood of a model is given by $l = \prod_{i=1}^{n} p_{model}(x^{(i)}, \theta)$, where $x^{(i)}$ is the i-th element of the dataset, and θ represents the parameters of the model. To simplify computations, the log likelihood is often taken into consideration, and therefore the product is transformed into a sum of logarithms: $l = \sum_{i=1}^{n} \log p_{model}(x^{(i)}, \theta)$. The best model is the one with the maximum likelihood, so:

$$\begin{aligned} \theta^* &= \arg\max_{\theta} \sum_{i=1}^{n} \log p_{model}(x^{(i)}, \theta) \\ &= \arg\max_{\theta} \mathbb{E}_{x \sim \hat{p}_{data}} \log p_{model}(x, \theta) \end{aligned} \tag{2.5}$$

where $\hat{p}_{data}$ is the distribution defined by the training data. The maximum likelihood estimator can be interpreted as a method to minimize the dissimilarity between $\hat{p}_{data}$ and p_{model}, similarly to the *KL divergence*. Usually, the optimal parameters are estimated by minimizing the negative of the likelihood; thus:

$$\theta^* = \arg\min_{\theta} -\mathbb{E}_{x \sim \hat{p}_{data}} \log p_{model}(x, \theta) \tag{2.6}$$

Note that minimizing the KL divergence between two distributions corresponds precisely to minimizing the cross entropy between them. For this reason, the cross entropy loss defined in subsection 1.3.3 is widely used in many machine learning models.

2.1.2 A taxonomy of generative models

Some generative models do not use maximum likelihood in principle, but they can be examined using a maximum likelihood variant. The main distinction among generative models is between explicit and implicit density models. For a complete review of generative models in relation to GANs see [31].

Explicit density function

Models that are in this category *explicitly* define the density function p_{model}. For these kind of models, the maximization of the likelihood is straightforward: it is sufficient to use Equation 2.5 and follow the gradient uphill (or downhill if the divergence is used).

The main difficulty with explicit density models is the designing of a model that can capture all the complexity of the data and, at the same time, maintain computational tractability. To confront this challenge, there are two different strategies:

- **Tractable explicit models**: the density function p_{model} is chosen to be computationally tractable. Examples of such models are *fully-visible belief networks*, *nonlinear independent component analysis* and *pixelRNN*.
- **Approximated explicit models**: when the distribution of the data cannot be reduced to a tractable p_{model}, a intractable one is used, and approximated with a lower bound $L(x, \theta) \leq \log p_{model}(x, \theta)$. Maximizing $L(x, \theta)$ has almost the same effect as maximizing the likelihood if the two functions are not distant, especially in their optimal point. The most famous example of approximate explicit models is the *variational autoencoder* (VAE). Another kind of models uses the *Markov chain approximation* instead of defining a lower-bound function. The most prominent example of these models is the *Boltzmann machine* [33].

Implicit density function

Some models can be trained without the need to explicitly define a density function. These models interact only indirectly with p_{model}, usually sampling data from it in order to compare the generated data with the training set distribution $\hat{p}_{data}$. In brief, the goal of these types of models is not to learn how the data is distributed throughout space, but how to generate items that *seem* extracted from p_{data}. Some implicit density models use Markov chains to draw samples from the implicit p_{model}, as *generative stochastic networks*. Those models, however, do not scale well to high dimensional space, and they require many steps to generate data. Generative adversarial networks were designed to avoid those problems, so they are able to generate samples in a single step.

2.2 Adversarial networks

Generative adversarial networks (GANs), introduced by Goodfellow *et al.* in [2], are an emerging technique for both unsupervised and semi-supervised learning. They are implicit density generative models, and they are characterized by two main components: a *generator* G, and a *discriminator* D. The basic idea of GANs is to set up a game between the generator and the discriminator. The former tries to generate samples that are intended to come from the real data distribution, while the latter examines real and generated samples in order to distinguish between real or fake data. A common analogy is to think of the generator as an art forger, and the discriminator as an art expert [34]. The forger tries to create forgeries which are increasingly similar to real paintings, in order to deceive the art expert. The expert, at the same time, learns more and more sophisticated ways to discriminate between real and false artworks.

The generator can be seen as a function $G : Z \to X$ that transforms some latent variable $z \in Z$ into a sample in the data space $x \in X$. The discriminator can be formalized as a function: $D : X \to (0, 1)$, which maps an item to its probability of being real.

One of the most crucial points of GANs is that the generator has no direct access to the real data: the only manner for it to learn is through interaction with the discriminator. By contrast, the discriminator has access to both real and generated data. This behavior can be expressed via a *minimax* game, where the generator tries to minimize the gain of the discriminator, while the discriminator tries to do the opposite.

2.2.1 Minimax games

Minimax is a decision rule, used in many different fields, for minimizing the possible loss in a worst case scenario. Originally, it was formulated for *zero-sum* two-players games, where a player gains advantage at the expense of the opponent. Minimax has also been extended to general decision-making in the presence of uncertain.

In a two player game, the minimax value of a player is the smallest value that the opponent can force it to obtain, without knowing the actual action of the player. In other words, it is the largest benefit that the first player can be sure to obtain, when they know the action of the other player:

$$\bar{v_i} = \min_{a_{-i}} \max_{a_i} v_i(a_i, a_{-i}) \tag{2.7}$$

where:

- a_i and a_{-i} are the two players.
- v_i is the value function for player i.

In this case, player a_{-i} tries to minimize the gain of a_i, without knowing a_i's move. On the other hand, a_i tries to maximize its benefit, knowing a_{-i}'s move.

The reverse of minimax is maximin, where a player tries to maximize their gain without knowing the move of the opponent, who tries to minimize the same quantity. The solution of a minimax game is equivalent to a *Nash equilibrium* [35], in witch none of the players can achieve a higher benefit by unilaterally changing their strategy.

2.2.2 Model architecture

The first architecture proposed for GANs consisted of two multilayer perceptrons, one representing the generator and the other the discriminator. The generator is fed with a noise vector, and produces a sample x drawn from the implicit distribution p_{model}. The discriminator is fed with x and yields the probability that x is sampled from p_{data}. The output of the discriminator is used to update the parameters of both the models.

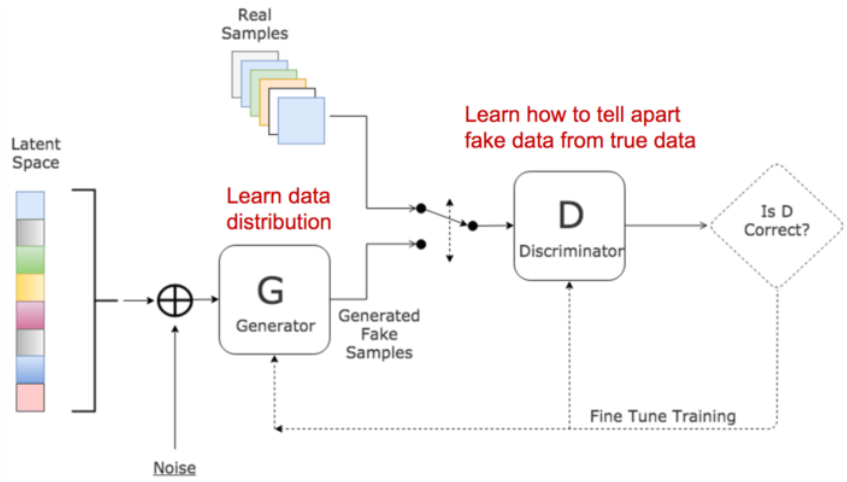


Figure 2.2: GAN general architecture.

More formally, the generator is a differentiable function $G(z, \theta^{(G)})$ and the discriminator a differentiable function $F(x, \theta^{(D)})$. The objective of the discriminator is to minimize a certain loss function $J^{(D)}(\theta^{(D)}, \theta^{(G)})$, and to have do so only by controlling $\theta^{(D)}$. At the same time, the generator wishes to minimize another loss function $J^{(G)}(\theta^{(G)}, \theta^{(D)})$ by only controlling $\theta^{(G)}$. Thus, the cost of each model depends on the other model's parameters. This scenario is simpler to describe as a game, rather than as an optimization problem. The theoretical optimum is when the generator's implicit model is able to perfectly match p_{data}, so that even an optimal discriminator will be maximally confused, predicting a probability of 0.5 for every input.
The GAN game is commonly played in the following way:

1. Some training examples $\boldsymbol{x}$ are randomly sampled from p_{data}.
2. The discriminator tries to make $D(\boldsymbol{x}) = 1$.
3. Some items $\boldsymbol{z}$ are sampled from the generator latent space.
4. The discriminator seeks to make $D(G(\boldsymbol{z})) = 0$, while the generator strives to make the same quantity approach 1.

In the last point, the reason GANs are adversarial models is made clear: both the generator and discriminator tries to maximize and minimize the same function respectively.

Neural networks have been used predominantly for both the discriminator and the generator since the work of Goodfellow *et al.* [2]. The main reasons for this trend are that NNs are able to perform nonlinear transformations of the input, potentially making the mapping between the latent space and the predicted distribution extremely complex [34]. Another reason is that neural networks are universal approximators, making GANs asymptotically consistent (i.e the estimation of optimal parameters grows if more data is provided). Nevertheless, any differentiable model can be used for both the components of a GAN, and even different models can also be adopted for the discriminator and the generator.

2.3 Training

The training of generative adversarial networks is often more problematic than the training of neural networks, due to the fact that it is not an optimization problem, but a minimax game. Ideally, the parameters of the two models should be updated at the same time, but this is unfeasible in practice. Since

the majority of GANs use neural networks, training is performed following the backpropagation algorithm described in subsection 1.3.4.

Loss functions

In the GAN game, the discriminator is the player that tries to maximize its gain, knowing the move of the opponent. Indeed, it seeks to classify the authentic data as real, and the generated samples as fake. This behavior can be correlated with a binary classifier, which aims to categorize data into two classes: real or fake. As reported in subsection 1.3.3, cross entropy loss is usually adopted for a binary classification. The GAN game can thus be expressed as:

$$\min_G \max_D \mathcal{L}(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}} [\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_z} [\log(1 - D(G(\boldsymbol{z})))] . \quad (2.8)$$

where $\mathcal{L}(D, G)$ represents the adversarial loss, which is actually a value function, since the discriminator aims to maximize it.

The value function of the discriminator, with regards to the parameters of the model, can be expressed as:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}} \log D(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z} \sim p_z} \log(1 - D(G(\boldsymbol{z}))). \quad (2.9)$$

In many models this is substituted with $\mathcal{L}_D(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)})/2$, in order to minimize instead of maximize. The only difference between adversarial discriminator loss and cross entropy loss is that the data comes from two different distributions, so the discriminator is trained with two minibatches of data: one coming from p_{data}, and the other from p_{model}. In [2], it is demonstrated that Equation 2.9 is the optimal discriminator strategy, and the equilibrium of the game is reached when the discriminator output is 0.5 for every input.

The cost function of the generator, since we are in a minimax game, is the same of the discriminator (one model tries to minimize it, while the other tries to maximize it). Thus, the generator has to minimize:

$$\mathcal{L}_G(\theta^{(G)}, \theta^{(D)}) = \mathbb{E}_{\boldsymbol{z} \sim p_z} \log(1 - D(G(\boldsymbol{z}))) \quad (2.10)$$

where the first term is omitted, since does not depend on the generator.

2.3.1 Training algorithm

Similarly to neural networks, SGD and backpropagation are used for parameter updating. In a training step, the discriminator is usually trained before

the generator. Some authors recommend running more training steps on the discriminator, before updating the generator[1], while others claim that the best strategy is to run only one step for both models [31]. This is an open research question, and a solution to this debate has not been found so far (November 2018). The procedure is described in Algorithm 2.

Algorithm 2 GAN Training procedure [2]

Require: k = number of steps the discriminator has to be trained before training the generator.

Require: m = minibatch size.

1: **for** number of training iteration **do**

2: **for** k steps **do**

3: • Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.

4: • Sample minibatch of m examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.

5: • Update the discriminator by **ascending** its stochastic gradient:

$$\nabla_{\theta^{(D)}} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$$

6: **end for**

7: • Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.

8: • Update the generator by **descending** its stochastic gradient:

$$\nabla_{\theta^{(G)}} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$

9: **end for**

One of the key feature of GANs is that the generator has no access to data, and it learns only by observing the output of the discriminator. This means that the generator is not directly related to its loss, i.e. changes in the perceptrons of the generator output layer cannot be converted in subsequent changes in the loss. Resuming the forgery example, the forger's ability to create counterfeits is not the only factor that determines the amount of money he or she can make by selling his or her works: the competence of the art expert is a part of the equation. More formally, this means that the gradient of the loss has to be propagated through the discriminator, before reaching the generator.

[1]Theoretically, the equilibrium of the game can be reached by starting from an optimal D and training only G; however, in practice, this approach is not feasible [36].

Moreover, since the discriminator is usually updated before the generator, the latter receives the gradient after the discriminator's weights are updated. This is the core of the minimax game played by GANs: the generator has to make a move without knowing the move of the discriminator. This, incidentally, makes GANs resistant to overfitting, since the generator has no possibility (even an implicit one) of copying training examples [31]. A more detailed GAN training algorithm is shown in Algorithm 3.

Summing up these properties, the training of the generator can be seen as a strange form of reinforcement learning [31]: the generator takes actions and gets a reward from them. It differs from traditional RL because the reward function is not stationary (the discriminator changes during training) and the generator is able to observe not just the reward, but its gradient. The last point states that, in spite of their names, the generator and the discriminator are not adversarial. In fact, the discriminator can be seen as a teacher, which evaluates the generator, and freely share knowledge with it about the real distribution in order to instruct the generator in how to improve [31].

2.3.2 Problems

As stated in the previous section, the training of GANs is more problematic than the training of most machine learning models, mainly because finding an equilibrium in a Nash game is more difficult than solving an optimization problem. In this section, three of the most important problems that are commonly encountered during learning are analyzed.

Instability

The instability of GANs is well documented in the literature [34, 36, 37] and it is caused by trying to find an equilibrium point with gradient descent. The equilibrium is a point $(\theta^{(D)}, \theta^{(G)})$ such that both the generator and the discriminator losses lie at a minimum with respect to their parameters. This has motivated the idea of using gradient descent to update the parameters of the two models. But, as discussed in [37], this idea does not hold, even for simple functions. If the discriminator changes its parameters, in order to minimize its loss, the generator is likely to shift away from its optimum point, and when the generator update its parameters, the same thing occurs with the discriminator. Equilibrium is not reached, because each player destroys the progress of the other, causing the value of the GAN function to oscillate. This oscillation is not easy to detect and to avoid, making the training of GANs instable. For

Algorithm 3 A more detailed GAN training algorithm.
For the discriminator cost function, cross entropy loss is used, instead of the objective function in Equation 2.9, in order to minimize instead of maximize.

Require: k = number of steps the discriminator has to be trained before training the generator.
Require: m = minibatch size.
for number of training iterations **do**
  for k steps **do**
    • Sample a minibatch $\boldsymbol{x}$ of size m from p_{data}.
    • Sample a minibatch $\boldsymbol{z}$ of size m from p_z (latent space).
    • Set the discriminator loss as:

$$\mathcal{L}_D = -\frac{1}{m}\sum_{i=1}^{m}\left[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))\right].$$

    • Back-propagate the loss **only** through the discriminator.
    • Update the parameters of the discriminator.
  end for
  • Sample a minibatch $\boldsymbol{z}$ of size m from p_z (latent space).
  • Set the generator loss as:

$$\mathcal{L}_G = \frac{1}{m}\sum_{i=1}^{m}\log(1 - D(G(z^{(i)}))).$$

  • Back-propagate the loss through the discriminator **and** the generator.
  • Update the parameters of the generator.
end for

a complete analysis of the various sources of instability during the training of GANs, see [36].

Even with those problems, SGD is still predominantly used during the training of adversarial networks, due to its popularity and its adoption by many machine learning frameworks.

Vanishing gradient

The vanishing gradient is a problem that is not limited to GANs, but common to many deep learning models. However, in generative adversarial networks, this problem is particularly difficult to overcome, for two main reasons.

The first reason is that the gradient of the loss has to flow through the discriminator, before reaching the generator. As can be seen in subsection 1.3.4, the error on one neuron is calculated by summing the errors of the following layer, and multiplying the result by the derivative of the activation function. For some activation functions, the derivative varies between zero and one, reducing the propagated error. Thus, when arrives at the generator, the error might be too small to change the parameters in a perceptible way, thereby stopping the learning.

Another cause of vanishing gradient is the loss function of the generator. In the classic model, the generator wants to minimize:

$$J^{(G)} = \log(1 - D(G(z))). \tag{2.11}$$

This function produces small values if the discriminator is able to reject the output of the generator with high confidence. This means that, if the discriminator overwhelms the generator, the latter does not have enough gradient to improve its parameters. This situation may seem difficult to encounter, but in [36] it is demonstrated that, if the space in which the p_{data} lies is high-dimensional, in the early stages of learning, even a trivial classifier is able to distinguish between real and fake data, with an accuracy of nearly 100%. This has led to the adoption of the following loss function for the generator:

$$J^{(G)} = -\log(D(G(z))). \tag{2.12}$$

This function has the advantage of providing an higher gradient during the initial steps of training. Moreover its minimum point is definite, as opposed to Equation 2.11, in which it is indefinite ($-\infty$).

Mode collapse

Mode collapse (also known as the *Helvetica* scenario) occurs when the generator maps many different vectors from the latent space to the same output. In practice, a complete mode collapse is rare, but partial mode collapse (the generator produces only similar images) is common. The problem arises when the generator finds some weaknesses in the discriminator, and, since the loss is low for that output, it continues to produce the same outcome. It is not easy for the discriminator to notice this problem, as it does not conserve a history of data.

Even if the discriminator detects the issue and starts to reject the generated data, the generator simply searches for another mode, and starts to generate

data that is close to the newly discovered vulnerability. Thus, the training becomes a cat-and-mouse game, without converging to the optimal point [38].

Mode collapse is still an open problem, and it is considered by many authors to be one of the most important issues of GANs [31].

2.3.3 Possible solutions

Since the advent of GANs, the problems described above have been analyzed, and some solutions have been proposed. Most of these contributions are merely practical, since a rigorous theoretical understanding of many phenomena related to training such models is still missing [37].

Virtual batch normalization

Batch normalization (see subsection 1.4.1) has been proved to be very helpful in many deep learning models, but for GANs it has a few unfortunate effects. The use of a different batch of data for computing the reference statistics (mean and variance) in every forward pass, results in fluctuations of these normalization parameters [31]. This is especially problematic when the minibatch sizes are small, because these fluctuations become large enough that they have a greater effect on the output than the input $\boldsymbol{z}$.

Virtual batch normalization [37] uses a *reference batch* of images extracted randomly from the training set at the beginning of the training phase. Statistics are computed on this batch and they are used in combination with batch statistics, calculated during forward passes. This procedure has the effect of reducing parameter fluctuations, and making the images in the same batch independent from each others.

One-sided label smoothing

Deep learning classification models are prone to produce classifications with very high confidence, especially when the data is artificially constructed. This is the case of the discriminator in GANs, which, in the first part of the training phase, usually greatly outperforms the generator.

One-sided label smoothing is a trivial technique from the 1980s, but one which has recently been rediscovered for the regularization of deep learning models [39]. The approach consists of *smoothing* the labels on the discriminator, i.e.

real data are not associated with the label 1, but with 0.9. Similarly, fake samples are associated with label 0.1 instead of 0. Training the discriminator with these labels results in less extreme gradients for the generator, speeding up the training. Moreover, it has been demonstrated that one-sided label smoothing can help the discriminator to resist attack by the generator (e.g. mode collapse).

Historical averaging

Historical averaging is a set of techniques derived from reinforcement learning, which consists of keeping a history of the training. In [37], is recommended that the parameters should be updated by taking the previous updates into consideration. Another approach is to maintain a pool of generated data, and sometimes feed the discriminator with an old sample, rather than the last one generated. Other authors recommend keeping the old parameters, and every k iterations substituting one of the two models with the old one.

Training with labels

If the training set is composed of data with labels, the use of a discriminator that classifies the data with those labels, instead of simply using real or fake classifications, often results in a dramatic improvement in the *subjective* quality of generated images [31, 37]. The reason of this behavior is not completely clear, but it seems that using labels results in a discriminator that is more similar to our own visual system. Thus, to trick the discriminator, the generator is forced to put more emphasis on the features we emphasize when we observe an image. Note that the improvement in image quality may not be objective. In fact, it is still not clear whether or not the images produced using labels are more realistic when they are viewed by humans, or if they actually have an intrinsic realism [31].

Noise adding

In [36], it has been demonstrated that, if the output space is high-dimensional, even a simple discriminator can easily overwhelm the generator. As an example, if the output space is the space of 64×64×3 images, it has 12,288 dimensions. Commonly, the target distribution (p_{data}) lies in a low-dimensional manifold, as does the model distribution (p_{model}). As demonstrated in [36], a

discriminator is not able to correctly classify data that comes from two different manifolds if they match perfectly in a large part of space. This is proven to be almost impossible if the space is high-dimensional (as in images).

One way to overcome this problem is by adding noise to the data before giving them to the discriminator. This procedure relaxes the bounds of the two manifolds, and penalizes the performance of the discriminator, in order to provide a better gradient to the generator during learning.

2.4 The problem of evaluation

The evaluation of generated data is an open issue, and it is not limited to GANs, but, in general, to all the generative models. As stated in section 2.1, a generative model tries to estimate the true distribution of data using some sample from a training set. Generative models are often built in order to generate new samples, so the evaluation has to be done with regards to the outputs.

A generative model can be evaluated using its likelihood, or, if it is impossible to calculate it, an approximation of the likelihood or some other metrics. However, as discussed in [40], these metrics are often unrelated to image quality. It is common that a generative model shows a good likelihood, but that it produces bad quality samples, or vice versa. In general, different metrics do not correlate well with each other, making an objective assessment of samples quality nearly impossible.

2.4.1 Why the loss function is not enough

Generative adversarial networks are particularly inclined to this problem, since the likelihood is not calculated: they lack an explicit representation of p_{model}, which is hidden in the generator's parameters. Moreover, GANs do not have a proper loss function: their goal is to find an equilibrium point, not a minimum.

The only cost function present in GANs is the discriminator loss (the generator loss is derived from it). It can be argued that this loss can be used for assessing the model quality, since a better generator is more likely to fool the discriminator. However, the discriminator loss should not be used as the only method to evaluate the quality of GANs for two main reasons: it is not stationary and it can suffer from adversarial attacks.

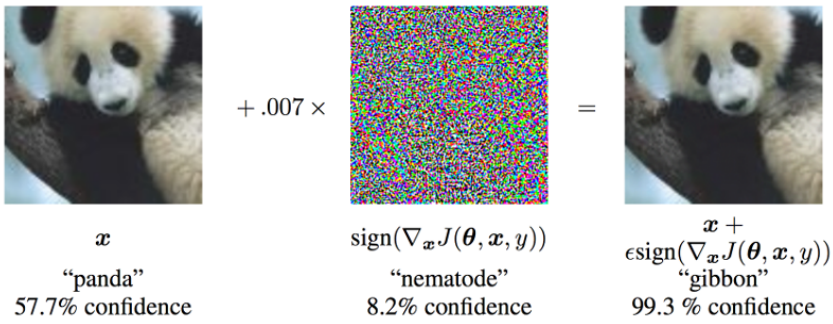


Figure 2.3: An example of adversarial attack. Suppose the objective of the generator is to generate images of gibbons, and the goal of the discriminator is to correctly classify gibbons. The image on the right is confidently classified as a gibbon, thus the generator loss is very low, even if it is improbable that the image belongs to the gibbon images distribution. Image from [41].

The discriminator is not a passive entity, it evolves, and its evolution is driven by both real and generated data. This means that, if, during training, the discriminator finds an effective way to separate real and fake samples, the generator loss increases, even if the quality of generated data remains the same. The discriminator's loss depends on two entities, so it may be unreliable to use it to evaluate the performance of the generator alone.

Moreover, the discriminator can be fooled if the generator is able to spot one weakness. A typical example is mode collapse: a mode collapsed generator produces very similar samples, which are believed to be real by the discriminator. In this case, p_{model} can be very different from p_{data}, but the generator's loss is usually small. An example of adversarial attack is shown in Figure 2.3.

2.4.2 Assessment of result quality

As stated in the above section, the assessment of result quality is particularly difficult for GANs. Indeed, many works still rely on human evaluation of generated data (especially if the results are images). One metric that has shown promising results is the *inception score* [37]. The inception score was initially proposed only for images, but can be used for every type of data. The score is based on the assumption that for an image to be recognized as real it must contain meaningful objects. At the same time, the generator has to produce a wide variety of different images, containing disparate objects. The metric applies a classifier to every image generated by the generator in order to obtain the conditional label distribution $p(y|x = G(z))$. Images that

contain meaningful objects should have a conditional label distribution with low entropy (the object is recognized with high confidence). On the other hand, the marginal $\int p(y|x = G(z))dz$ should have high entropy, since the classes have to be equally represented (note that $\int p(y|x = G(z))dz = p(y)$). The score is given by the divergence of the conditional and the marginal label distributions, so:

$$score = \exp(\mathbb{E}_x \mathrm{D}_{KL}(p(y|x)||p(y))). \tag{2.13}$$

It has been observed that the inception score correlates well with human judgment, and for this reason its adoption is increasing for a wide range of general models.

2.5 Examples and applications

In this section some examples of GAN models and some applications are provided. What follows is not a comprehensive list, since generative adversarial networks are a extraordinary active research field, and new models and applications are discovered and published every day. Those listed below are the most important, in particular, for the models used in the following chapters. For a more comprehensive view of models and applications see [31, 34].

2.5.1 Examples

The first GAN model dates back to the original paper by Goodfellow *et al.* [2] and it is commonly called *vanilla GAN*. This first version of GAN uses simple multilayer perceptrons for both the generator and the discriminator. This is the simplest model possible, and, in the training phase, it uses the exact algorithm described in Algorithm 3. With this model it is possible to generate simple images, such as fake digit, similar to those present in the MNIST dataset[2] or images of different objects using the CIFAR-10 dataset[3].

The reference for most of the GANs developed today is DCGAN [30]. DCGAN stands for *Deep Convolutional GAN*, and though GAN used convolution before DCGAN, it was the first model which learned how to generate high-resolution images in a single shot. Before this, another model called LAPGAN (*Laplacian* GAN) used multiple generators to generate different levels of detail in a

[2] `http://yann.lecun.com/exdb/mnist/`

[3] `https://www.cs.toronto.edu/~kriz/cifar.html`

Laplacian pyramid representation of an image. DCGAN is based on two key points, which are:

- The use of batch normalization in both the generator and the discriminator. During training, the discriminator uses different statistic for the two minibatches.
- No pooling or *unpooling* layers. If the spatial dimension of representation has to be increased, the model uses transpose convolution with a stride greater than one.

In the original architecture, the DCGAN was fed with a 100-dimensional vector from the latent space, and it produced a 64×64×3 image. Moreover, it has been clearly demonstrated that DCGAN uses the latent vector in a meaningful way, i.e. a simple operation in the latent code has a clear interpretation on the output [31].

Conditional GANs (CGAN) [42] use supplementary information, in addition to generator and discriminator input. The additional information can be the desired class of the output, or any type of data with the aim of conditioning the generator in producing whatever is required. The most important element of this model is that the discriminator also receives the conditioning information. The objective of the discriminitar is no longer to distinguish between real and fake data, but to evolve into a more sophisticated task, which is: given the conditional information, is the data produced by the generator conditioned in the proper manner? As an example, if the conditioning information is the desired class of the output, the discriminator's task becomes one of yielding the probability that, given the class, the output of the generator represents an object of that class ($p(x|y)$), which is different from classification ($p(y|x)$).

Another model worth mentioning is the *Wasserstein* GAN (WGAN) [43]. This model was developed with the precise idea of resolving most of the problems that usually arise when a GAN is trained. Wasserstein GANs use the notion of Wasserstein distance, or Earth Mover distance (EM) as a loss function for the discriminator. This approach was demonstrated to provide a more usable gradient for the generator, and it is less prone to instability problems.

2.5.2 Applications

Generative adversarial networks have been applied to a wide range of topics, predominantly related to computer vision. In this section some applications where GANs have shown particularly promising result are described, especially

when compared to classic machine learning approaches. For a broader list of present and potential future applications see [31, 34, 44].

Image synthesis

Image synthesis is the ability to produce images, starting from another type of information. This information can be random noise, as in DCGAN, a text describing the image, or a feature of the image. GANs, since their advent, have shown an extremely high level of photo-realism, even if the latent vector given to the generator is not as high-dimensional as the output result. One of the most impressing examples of image synthesis waws carried out by NVIDIA Research [45], which uses the technique of progressively growing the generator and discriminator, starting from only one layer in each model, and adding layers as the training advances. The model is able to generate 1024×1024 images that are practically indistinguishable from real ones, starting from a latent vector of only 512 values (see Figure 2.4).

Figure 2.4: An example of image synthesis using generative adversarial networks. None of the people represented above is a real person. The model was trained with celebrity faces, and maps a latent code of 512 values onto high resolution photographs. Image from [45]

Text-to-image is considered the *holy grail* of computer vision [44], since, if a model can generate realistic images from text, there is an high likelihood that the model actually understands what is in the image. Current test-to-image models based on GANs perform well if the image contains only one object, but lack precision and sharp details if the scene has to include more than one item. For a broad analysis of the use of GANs in text-to-image see [44].

Image-to-image translation

Image-to-image translation is defined as translating the possible representation of one scene into another, such as mapping greyscale images to RGB, or generating an image from only the edges. Problems related to image-to-image translation are often ill-posed, since for one input image, more than one correct mapping can exist, making the assessment of the result difficult. One task that is directly related to image-to-image translation is *style transfer*. In style transfer, the goal is to transfer the style of one target image to another, maintaining the content of the input. An example of this is the transformation of a photo into the style of a famous painter, or the change of the season, from winter to summer or vice versa. Style transfer is discussed in detail in the next chapter.

Another example of image-to-image translation is super-resolution, where the resolution of the input image is enhanced. One of the most successful models is SRGAN [46], where the adversarial loss is used in combination with other losses related directly to the resulting image. An example of super resolution is shown in Figure 2.5.

Image-to-image synthesis techniques can be used for video prediction. In this case, the generator takes a frame as input, and tries to produce the next frame of the video [47].

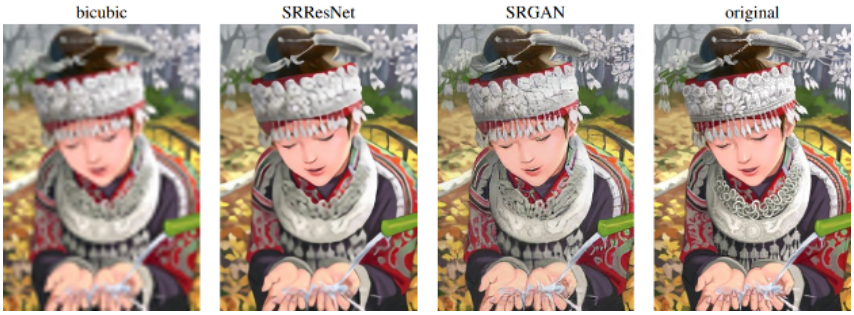


Figure 2.5: An example of super-resolution performed with a generative adversarial network, compared with a non-adversarial approach. The GAN result presents a higher sharpness, and more details are visible. Image from [46]

Feature learning

The generator, in GANs, learns a mapping between an arbitrary latent space and data space, in a completely unsupervised manner. The generator asso-

ciates the feature code's values to the actual semantic attributes of the output. It has been demonstrated that interpolations in the latent space produce smooth and plausible variations in the generated data, and certain directions in the latent space are associated with particular semantic values of the model distribution. So, the generator considers the latent code as a feature vector, and it learns in an unsupervised manner how to map data features onto latent space. If it were possible to invert the generator, a powerful feature extractor could be created. Unfortunately, most GAN models do not include this inverse mapping.

In [48], a model called BiGAN is proposed, where an additional encoder E is trained in order to invert the mapping performed by the generator. The encoder can be used as a feature extractor after the training phase, and the original image can be reconstructed using the generator.

Others models that are used in unsupervised feature learning are adversarial autoencoders [49]. In these models the generator is an autoencoder that maps the data onto a latent space, and back again to data space. So, the latent feature vector produced after the encoding phase must represent the most important feature of the data, and this mapping is learned in an unsupervised fashion.

Semi-supervised learning

In the previous applications, the goal of the model is to train the generator with the help of the discriminator, which acts like a teacher. Usually, after the learning phase, the discriminator is discarded, and only the generator is used. In semi-supervised learning, this paradigm is shifted, since the objective is to train the discriminator, with the help of the generator.

Having a vast dataset is one of the prerequisites for training deep models. However, a large amount of labeled data is not always available, even if unlabeled data is abundant. In these cases, a supervised approach is not feasible, but a deep classifier can be successfully trained with adversarial semi-supervised learning [37]. Suppose that the goal is to classify data in k classes, having many unlabeled samples and only few labeled ones. The approach consists in extending the discriminator, so that it will be able to classify data in $k + 1$ classes: the k original classes plus the class of fake data. Therefore, the discriminator is fed with labeled, unlabeled and fake data. The objective is different for each category of data. If the discriminator is fed with labeled data, the goal becomes to maximize the confidence of the right classification. If the given data is unlabeled, the discriminator's objective is to minimize the probability

of the data to be classified as fake, and when generated data is presented to the discriminator, its objective is to maximize the confidence of the fake class.

Using the procedure above, great improvements in classification tasks have been demonstrated, when there is only a small amount of labeled data [31], especially if the discriminator uses feature matching to analyze the input data.

3

Cycle-Consistent Adversarial Networks

> *It's difficult to be rigorous about whether a machine really 'knows', 'thinks', etc., because we're hard put to define these things. We understand human mental processes only slightly better than a fish understands swimming.*
>
> – John McCarthy, *The Little Thoughts of Thinking Machines* (1983)

In this chapter the focus will shift from more general concepts to a detailed description of one adversarial model. First, a discussion about image-to-image translation will be provided, with particular emphasis on the most recent developments in the field. Subsequently, the cycleGAN model will be presented and discussed in details, with an analysis of the training procedure, in order to give a comprehensive examination of the framework.

3.1 Image-to-image translation

Image-to-image translation is a class of computer vision and graphic problems, where the goal is to learn how the mapping between two classes of images takes place, formally: $f : I_A \rightarrow I_B$, where I_A and I_B are the image classes. The concept of image translation can be compared with the translation between two different languages. Concepts have to remain the same, the only thing that changes is the *representation* of these concepts. Just as the same sentence can be translated in many different languages, so too can a scene be represented

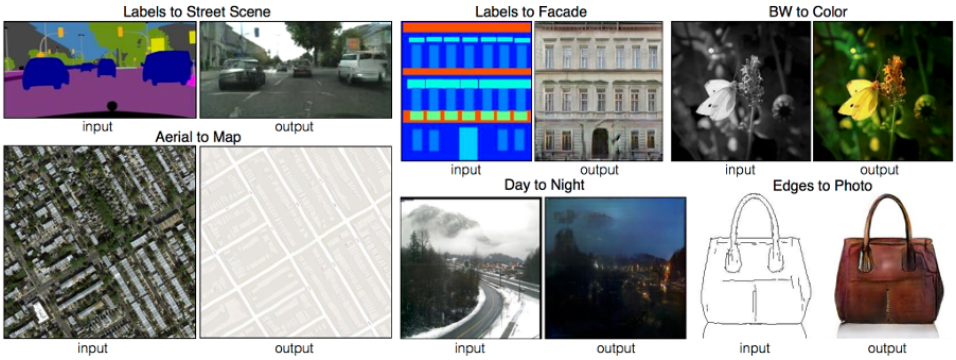


Figure 3.1: Some examples of image-to-image translation. Images from [50].

in many forms, such as RGB images, label maps, depth maps, edge maps, etc. The aim of image-to-image translation is to correctly transform an image from one representation into another. Many problems in image processing can be posed as a translation problem, such as image colorization, depth map estimation or semantic segmentation. Some examples of translations are shown in Figure 3.1.

Unfortunately, unlike sentences in different languages, images in distinct representations may not have the same information content, making the problem of image-to-image translation ill-posed, i.e. more than one output can be correct, given the same input. As an example, suppose we have a depth map of a scene, and we want to translate it into an RGB image. The depth map does not contain information about colors or small details, so many outputs can be considered valid. Moreover, a reference image is often missing, making the evaluation of the result even more difficult.

Image-to-image translation can be also used for images that have the same representation, but a different domain. For instance, an image of a winter scene can be translate into the same scene, but as if the photo was taken in summer. This form of image translation is particularly interesting for removing unwanted features, such as noise or bad weather conditions.

3.1.1 Style transfer

Style transfer is a particular application of image-to-image translation, where the goal is not translating from one image domain to another, but creating a new picture by the union of two different images. These images are named content and style images. The former defines the essence of the mixed image, while the latter determines its style. More formally, style transfer can be

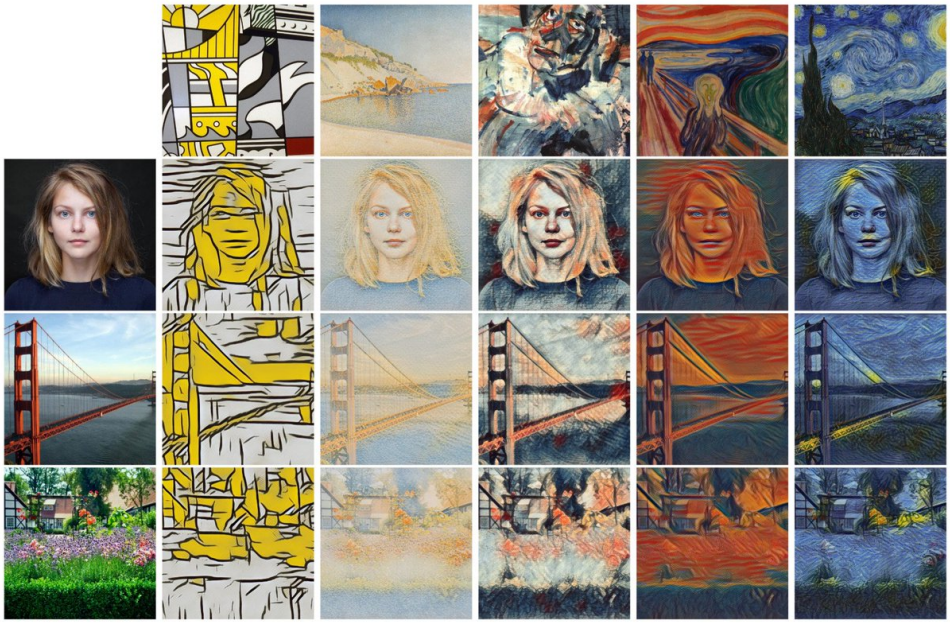


Figure 3.2: Some examples of style transfer. Image from [51].

formulated as $f : \{I_1(C_1, S_1), I_2(C_2, S_2)\} \rightarrow I_o(C_1, S_2)$, where $I(C, S)$ is an image with content C and style S. Style transfer is nowadays mainly used to create artistic artworks, therefore it is not only used by computer scientists, but by a wide range of artists and photographers. Some examples of style transfer are shown in Figure 3.2.

Apart from artistic creations, style transfer can be useful for understanding to what extent current machine learning models are able to correctly understand the semantics of images. Content and style are two different aspects of pictures, and combining them together in a meaningful way requires some sort of image comprehension.

The style of an image is usually defined as a texture; indeed, it is similar to different parts of the same picture, and it is repeated across the image. The content, however, is composed of local features, which represent the shapes of the represented objects. Mixing local and global feature is the goal of style transfer.

Nevertheless, the boundary between image-to-image translation and style transfer is not clearly defined, and often the two concepts are used as synonyms, especially if the translation is performed between two different domains, instead of diverse representations.

3.1.2 Related work

The revolution in computer vision that followed the introduction of deep convolutional networks dramatically improved the performance and the quality of image-to-image translation models, in particular for style transfer. On the other hand, generative adversarial networks have demonstrated exceptional results in image translation, due to adversarial training. In this section, a summary of the principal techniques used for image-to-image translation is provided.

Style Transfer

Style transfer is based on transferring the style of one image to another, while maintaining the same content. The work carried out by Gatys *et al.* [52] was the first example of style transfer that was able to produce impressive results. The proposed model consisted of a pre-trained classifier, used as a feature extractor, in order to match feature statistics in the convolutional layers of the network. VGG-19 [53] was used in the original paper as the feature extractor. The content images's features were initially extracted from a layer of the VGG-19 network, and, in a similar manner, the style features were extracted from more than one layer, and calculated using the Gram matrix. The output image was then edited by gradient descent during training, in order to minimize the loss defined to preserve the content of one image and the style of the other.

The presented algorithm is flexible, since it can work with any content or style image, but expensive, because it requires an optimization phase for any run. In [54], this problem is tackled by introducing an *image transformation network*, which is a CNN, so it can be trained in order to apply the style of one image to its input in one pass. The main drawback of this approach is that the image transformation network can learn only one style, so it is necessary to train more than one net to enforce different styles.

Some advancements are proposed in [51], where the image transformation network is trained with many styles, and the input is conditioned in order to inform the network about which style to apply. Another step forward is presented in [55], where the conditional information about the style is directly extracted from the style image, making the application of many more styles possible, even if the network has never been trained with the corresponding image.

Image-to-image translation

Generative adversarial networks have shown impressive performance in image-to-image translation problems, mainly because the same model can be used for many different tasks. Before the advent of GANs, each of these tasks was tackled by a different model, despite the fact that the fundamental problem is always the same: translating from pixel to pixel. One of the first works that took a step this direction was *pix2pix* [50], where a conditional GAN is used to translate an image in one domain to another. The model exploits the GAN's ability to automatically learn a loss that adapts to the data (the cost is given by the distance between real and fake data distribution), producing many different translations simply by training the model with different datasets. The generator receives an input image x, and it tries to translate it in the domain Y, producing $\hat{y}$. The discriminator is fed with the pair (x, $\hat{y}$), and yields the probability that, given x, $\hat{y}$ resembles a real mapping in the domain Y. This loss is added to the L1-loss between $\hat{y}$ and the real translation y, in order to diminish blurring.

The approach of pix2pix has demonstrated exceptional results, but it requires a ground truth image of the mapping during training. In other words, each image from the first domain must have a corresponding image in the second domain (i.e. image pairing). This is a limitation, since for several domains the ground truth image y may not be available (e.g. if we want to apply the style of a painter). *Cycle GAN* [4] surmounts this obstacle using two adversarial networks, one for the mapping $X \to Y$ and the other for the inverse operation. Cycle GANs are discussed in more detail in the next sections.

As stated before, the problem of image-to-image translation is ill-posed, since a single input image may correspond to multiple possible outputs. Both pix2pix and cycleGAN are deterministic: once trained, the same input image always produces the same mapping. In [56], an architecture named *BicycleGAN* is proposed, which is capable of producing many different results, given the same sample. Simply adding noise to condition the generator has no effect [50], so the idea consists in extracting a feature vector from the ground truth image, and using it to condition the generator. The same features are extracted from the resulting image, and compared with those of the ground truth image. Thus, the generator learns to produce translations that are conditioned by certain features. This, after training, can be used even in the absence of ground truth images, conditioning the generator with the features that are desired in the resulting image (e.g. a night-day translator can be conditioned with information about the weather condition, such as cloudy, sunny etc.).

Figure 3.3: An example of deep photo style transfer. Image from [57].

A model that lies in the indistinct boundary between style transfer and image-to-image translation is proposed by Luan *et al.* [57], where tasks typical of image-to-image translation are performed following a style transfer approach (see Figure 3.3).

3.2 Cycle GAN

Cycle-consistent adversarial network (CycleGAN) [4] is an adversarial model for image-to-image translation, which, unlike pix2pix, has the ability to learn mapping without requiring paired data. The model can capture the characteristics of one image collection, and figure out how these features could be translated to the other image domain, all in the absence of paired examples.

This is similar to how humans are able to translate an image from one domain to another, even if they have never seen the photo before. For instance, we can easily image how a winter landscape might appear in the summer. We can do this because we have seen many summer scenes over the course of our lifetime, hence we can infer their most important features, and use them to translate images to the summer domain.

The CycleGAN model tries to do the same, exploiting the adversarial training for learning a mapping from the domain X to Y so that $\hat{y} = map(x \in X)$ is a plausible element of the domain Y. Note that $\hat{y}$ cannot be compared to any image of Y, since the two domains are unpaired.

3.2.1 The problem of paired data

As illustrated in subsection 3.1.2, many powerful translation models have been produced, but the majority of them rely on paired datasets during the training phase. A paired dataset is a dataset which contains a collection of pairs $\{x, y\}$

representing the same scene rendered in two different domains. An example of a paired dataset is shown in Figure 3.4.

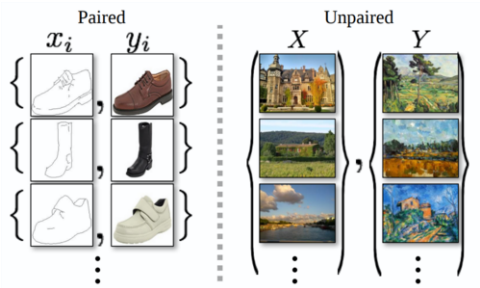


Figure 3.4: A paired and an unpaired dataset. Image from [4].

On the other hand, an unpaired dataset is composed of two distinct collections, a source set and a target set, with no information provided as to which element from the first set is associated to the second one. In general, an unpaired dataset consist of two uncoupled collections of data.

A paired dataset is often more difficult and expensive to obtain than an unpaired one, since for many domains, direct mapping is not possible. As an example, a paired dataset of summer-winter images is extremely challenging to collect, as small changes in camera position or the effect of the wind can easily result in misalignment between a pair of photographs. Furthermore, in many cases, a ground truth input-output pair is impossible to obtain, for example if the task is collection style transfer, where the goal is to apply the style of an artist to an input image. In this case, we have a collection of paintings and a collection of photographs, which are not linked in any way.

Working with paired data is usually simpler than using unpaired training sets, due to the fact that the output can be directly compared with the desired result. If the ground truth output is missing, there is no way to directly evaluate the quality of the mapping.

3.2.2 Cycle consistency

Trying to learn an image-to-image mapping with unpaired data is like learning to translate from one language to another without having a dictionary. What we have is just a collection of sentences in the two languages, without any direct relationship between them. Although ground truth input-output pairs are not present, an adversarial model can be exploited in order to learn a map $G : X \to Y$ such that the output $\hat{y} = G(x),\ x \in X$ is indistinguishable from images in Y for an adversarial discriminator. However, such a translation does not guarantee that the input x and the output $\hat{y}$ are paired in a meaningful way. In fact, the generator is only forced to produce images that resemble the ones present in Y, not to maintain the characteristics of the input x in $\hat{y}$.

In pix2pix [50], the mapping is forced to be meaningful using an L1-loss between the desired output and the generated image, but this is not possible if we work with unpaired data. Therefore, the property that a translation should be *cycle consistent* [58] can be exploited. This property, in the original formulation, states that a sentence, translated from one language to another, and then translated back to the original language, should be similar to the initial, original untranslated sentence. Formally, if we have a mapping $G : X \to Y$ and another mapping $F : Y \to X$, then G and F should be inverse mappings i.e. $F(G(x)) = x,\ \forall x \in X$ and $G(F(y)) = y,\ \forall y \in Y$.

CycleGAN uses this principle to obtain a meaningful mapping from X to Y. The generator G tries to translate an image $x \in X$ to an output $\hat{y}$, making it as similar as possible to images in Y. At the same time, another translator F is trained to learn the inverse mapping of G. The cycle consistency is calculated as the L1-loss between $F(\hat{y})$ and the original x:

$$\mathcal{L}_{cycle} = ||F(\hat{y}) - x||_1 = ||F(G(x)) - x||_1 \tag{3.1}$$

This loss forces $\hat{y}$ to be consistent with x, since the latter has to be reconstructed from the former. Combining the adversarial loss of GANs and the cycle consistency loss, the generator G can be forced to learn a mapping that translate an image x into an image $\hat{y}$ with two fundamental properties:

- $\hat{y}$ has to resemble images in the domain Y (adversarial loss).
- $\hat{y}$ has to be similar to x, since the inverse mapping should reconstruct x with high accuracy (cycle consistency loss).

These two properties together yield the final objective of cycleGAN, for unpaired image-to-image translation. It is worth noting that the cycle consistency loss is not domain specific, nor is the adversarial loss. For this reason, cycleGAN is a general purpose model, which can learn many mappings between different domains, from data alone.

3.2.3 General structure

The general model of CycleGAN is composed of two main components: a translator from domain X to Y, and the inverse translator, which maps the domain Y to X. The two components are modeled as generative adversarial networks, so they are composed of a generator, which performs the mapping, and a discriminator. Broadly speaking, two translations are learned in parallel: one defined by $G : X \to Y$ and the discriminator $D_Y : Y \to [0, 1]$, and the other formalized as $F : Y \to X$ and $D_X : X \to [0, 1]$.

The two models are held together by the cycle consistency loss, which states that: $F(G(x)) \approx x$, $\forall x \in X$ and $G(F(y)) \approx y$, $\forall y \in Y$. The general architecture of a cycleGAN is shown in Figure 3.5.

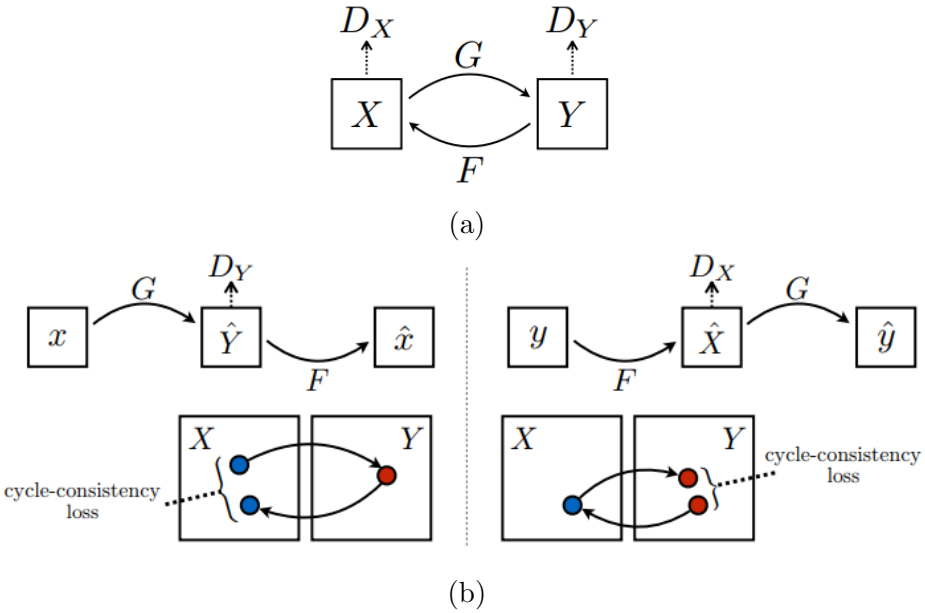


Figure 3.5: (a) The general model of a cycleGAN, with the two generators G and F and the two discriminators D_X and D_Y. (b) A visual representation of the cycle consistency. Note that the combination $G \circ F$ can be seen as an adversarial autoencoder, which maps x to $\hat{x}$, passing through an internal representation that is compelled to be similar to Y by the discriminator D_Y. The same is valid for $F \circ G$. Images from [4].

The cycleGAN model can be seen as the training of two autoencoders [4]: defined as $F \circ G : X \to X$ and $G \circ F : Y \to Y$. However, the internal representation is not defined in a feature space of lower dimensionality with respect to the input, but it is the translation of the input image into another domain. Such a set-up can also be seen as a special case of adversarial autoencoding [49], which uses the adversarial loss to train the bottleneck layer of an autoencoder to match an arbitrary target distribution. In this case, the target distribution for the $F \circ G$ autoencoder is the domain Y (see Figure 3.5b for a graphical explanation of this concept).

Losses

The cycleGAN model is composed of two losses: the adversarial loss and the cycle consistent loss. The adversarial loss is a typical GAN loss, presented in section 2.3. Since cycleGAN is composed of two GANs, there are two adversarial losses. The adversarial loss for the mapping $G : X \to Y$ and the discriminator D_Y can be expressed, using the classical GAN loss, as:

$$\begin{aligned}\mathcal{L}_{GAN}(G, D_Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] \\ + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x)))]\end{aligned} \tag{3.2}$$

where $p_{data}(x)$ and $p_{data}(y)$ are the two image distributions. This function has to be minimized by the generator, and maximized by the discriminator. For this reason, the name adversarial loss is misleading, since it is actually a value function. However, the name loss is maintained to keep coherence with the original publication [4].

In this case, G tries to generate images $G(x)$ that look similar to those from domain Y, while D_Y aims to distinguish between real and generated data. As in the GAN game, the generator and the discriminator play a minimax game; more precisely, G tries to minimize this objective, whereas D_Y tries to maximize it. This can be expressed as $\min_G \max_{D_Y} \mathcal{L}_{GAN}(G, D_Y)$. A similar adversarial loss is introduced for the inverse mapping $F : Y \to X$, with the discriminator D_X.

The cycle consistency loss is based on the assumption that, since G and F are one the inverse of one another, $F(G(x)) \approx x$ and $G(F(y)) \approx y$. The former is called *forward cycle consistency*, while the latter is *backward cycle consistency*. The cycle consistency loss is the union of these two components, and it is defined as:

$$\begin{aligned}\mathcal{L}_{cycle}(G, F) = \mathbb{E}_{x \sim p_{data}(x)} \|F(G(x)) - x\|_1 \\ + \mathbb{E}_{y \sim p_{data}(y)} \|G(F(y)) - y\|_1.\end{aligned} \tag{3.3}$$

The L1 norm was chosen because it produces less blurred results, compared to the L2 norm [50].

The cycle consistency loss does not include the discriminators, rather it is calculated using the generators alone. This loss binds the two GAN models together, because both the forward and the backward cycle losses require the two generator to be calculated. This loss in not adversarial, since both G and F have the goal of minimizing it.

The full objective of the cycleGAN model is:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y) + \mathcal{L}_{GAN}(F, D_X) + \lambda \mathcal{L}_{cycle}(G, F) \tag{3.4}$$

where λ controls the relative importance of the cycle consistency loss. The cycleGAN game is defined as:

$$G^*, F^* = \arg\min_{G,F} \max_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y). \tag{3.5}$$

In more details, the optimal discriminator D_Y and the optimal generator G are defined as:

$$D_Y^* = \arg\max_{D_Y} \mathcal{L}_{GAN}(G, D_Y) \tag{3.6}$$

$$G^* = \arg\min_{G} \mathcal{L}_{GAN}(G, D_Y) + \lambda \mathbb{E}_{x \sim p_{data}(x)} \|F(G(x)) - x\|_1. \tag{3.7}$$

It is important to notice that, in contrast to simple GANs, the optimal generator G depends on the discriminator D_Y and also on the other generator F. A similar conclusion can be derived for F and D_X. Thus, four different models have to be trained in parallel, introducing even more instability than in classical GANs.

In [4], some experiments are presented where only one loss is used, instead of the combination of adversarial training plus cycle consistency. Using only the adversarial or the cycle loss resulted in a degradation of the quality of the results. Moreover, using the cycle consistency loss in only one direction (forward or backward) is not sufficient to regularize the training.

3.2.4 Details

All four models are implemented as convolutional neural networks, since the primary goal of the models is to work with images.

Generator's network

The architecture of the generative networks G and F is an adaptation of the image transformation network, defined in [54], due to its impressive results for style transfer and super-resolution. CNNs, apart from classic blocks illustrated in subsection 1.5.1, use more sophisticated techniques, described below.

Instance normalization : this is an improvement of batch normalization, where the normalization parameters (mean and variance) are calculated for the single channel, rather than for the entire batch. As an example, if the batch size is 5, and every element of the batch is composed of 64 channels, and batch normalization is used, every activation in the k-th channel is normalized with mean and variance calculated among the five k-th channels of every element in the batch. With instance normalization the neurons' activations are normalized with regard to the k-th channel of the batch element. For a visual representation see Figure 3.6a.

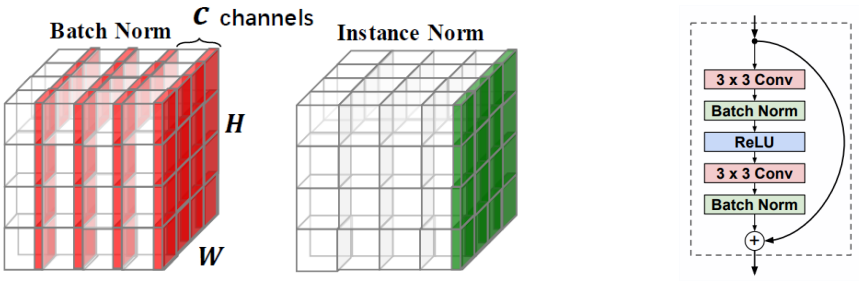
Residual block : this is the main component of residual networks [59], a very deep architecture that has shown impressive results on image recognition. A residual block is characterized by the *residual connection*, a shortcut connection between the input and the output, and it is often composed of only a few convolutional layers. In order to be able to combine the input and the output, a residual block does not produce any dimensionality transformation. The residual blocks used in cycleGAN's generators are composed of one 3x3 convolution layer with stride 1 and padding 1, followed by an instance normalization and a ReLU activation layer. After that, the convolution is replicated, followed by another instance normalization layer. The output is then combined with the original input. See Figure 3.6b.

Transpose convolution (or fractional-strided convolution): this is commonly used in the upsampling layers of autoencoders, to augment the dimensionality of the feature maps, using an operation which is the reverse of a convolution. The easiest way to think about a transposed convolution on a given input i is to imagine the convolution that had produced that input, starting from a feature map m. The transposed convolution can be seen as the operation that allows the shape of the initial feature map m to be reconstructed starting from i, maintaining the connectivity of the convolution operation [60].

The generators architecture is fully convolutional, this means that they are able to work with images of different dimensions, without changing the network structure. The detailed generator architecture is shown in Table 3.1.

Discriminator's network

The architecture of the discriminator is based on the *patchGAN* model [46]. This model aims to classify overlapping $n \times n$ image patches as real or fake, and the final result is the average of all the classifications. The output layer



(a) (b)

Figure 3.6: (a) The difference between batch normalization and instance normalization. In batch normalization, mean and variance are calculated batch-wise. In instance normalization, they are determined only with regards to the single channel. (b) The residual block used in the image transformation network [54]. In cycleGAN, the same architecture is used, but batch normalization is substituted by instance normalization.

of the network is a matrix of predictions, each of them related to a different patch. This is possible due to the locality property of CNN, that states that a convolutional neuron is connected only with a small set of neighboring neurons in the previous layer. For this reason, an output neuron's activation is only dependent on a small part of the input, as described in Figure 3.7.

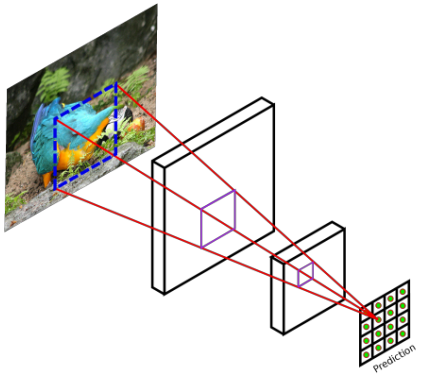


Figure 3.7: The patchGAN architecture.

Just like the generators, the patchGAN discriminators are also fully convolutional, in order to process images of different sizes. The detailed architecture of the discriminators is shown in Table 3.2.

Layer	Activation size
Input	3×256×256
32×7×7 conv, stride 1, reflection padding 3	32×256×256
64×3×3 conv, stride 2, padding 1	64×128×128
128×3×3 conv, stride 2, padding 1	128×64×64
Residual block, 128 filters	128×64×64
. . .*	. . .
Residual block, 128 filters	128×64×64
64×3×3 conv, stride 1/2†, padding 1	64×128×128
32×3×3 conv, stride 1/2†, padding 1	32×256×256
3×7×7 conv‡, stride 1, reflection padding 3	3×256×256

* The networks use a total of nine residual blocks.
† Stride 1/2 represents a transposed convolution layer with stride 2.
‡ The last convolution is followed by a softmax activation layer.

Table 3.1: Architecture of generators. Every convolutional layer is followed by a an instance norm layer and a ReLU activation layer.

Layer	Activation size
Input	3×256×256
64×4×4 conv*, stride 2, padding 1	64×128×128
128×4×4 conv, stride 2, padding 1	128×64×64
256×4×4 conv, stride 2, padding 1	256×32×32
512×4×4 conv, stride 1, padding 1	512×31×31
1×4×4 conv†, stride 1, padding 1	1×30×30

* No instance normalization.
† Followed by a sigmoid activation layer.

Table 3.2: PatchGAN discriminator's architecture. Every convolutional layer is followed by an instance norm layer and a leaky ReLU activation layer, with slope 0.2.

3.3 Training

A cycleGAN is composed of two generative adversarial networks, which have to be trained simultaneously. As for GANs, the generator cost function depends on the discriminator, but furthermore, every generator depends on the other generator for the calculation of the cycle consistency loss.

3.3.1 Training procedure

Backpropagation is adopted as the default algorithm for training the four networks. In the GANs training algorithm illustrated in section 2.3, the discriminator is trained before the generator [4]. However, in the original implementation of cycleGAN, the two components are swapped, and the generators are trained before the discriminators. The order does not influence the quality of the system after the learning phase, so the original cycleGAN implementation is used.

Similarly to GANs, where the adversarial loss has to be propagated through the discriminator before reaching the generator, in cycleGAN the cycle consistency loss is also not directly related to the model that has produced it. For instance, the cycle loss for the generator G is:

$$\mathcal{L}_{cycle_G} = \mathbb{E}_{x \sim p_{data}(x)} \|F(G(x)) - x\|_1. \tag{3.8}$$

When the associated loss is back-propagated, it flows first through F, and then is propagated through G. Nevertheless, as opposed to classical GANs, the gradient that flowed in F is conserved and accumulated, and it is used during the update to F's parameters. A similar behavior is utilized for the backward consistency loss. Moreover, in the original cycleGAN implementation, the two generators are updated at the same time, since the cycle consistency loss affects both of them.

To improve the stability of the training and to produce better results [61], the adversarial loss is substituted with a squared loss: $\mathcal{L} = (\hat{y} - y)^2$, where y is the real label and $\hat{y}$ is the prediction of the model. The discriminator aims to yield a high probability when real data are presented to it, and a lower one when it is fed with fake data. Thus, real data should be labeled with 1, and fake data with 0. The adversarial loss of the discriminator D_Y becomes:

$$\mathcal{L}_{GAN_{D_Y}} = \mathbb{E}_{y \sim p_{data}(y)}(D_Y(y) - 1)^2 + \mathbb{E}_{x \sim p_{data}(x)}(D_Y(G(x)))^2 \tag{3.9}$$

In the same manner, the squared loss can be derived for the generator G. It tries to maximize the output of the discriminator, when generated data is presented to it. In other words, the adversarial objective of the generator is that $D_Y(G(x)) = 1$, for every generated image. Thus, 1 can be used as a label, and the generator adversarial loss written as:

$$\mathcal{L}_{GAN_G} = \mathbb{E}_{x \sim p_{data}(x)}(D_Y(G(x)) - 1)^2. \tag{3.10}$$

The same procedure can be performed for F and D_X.

Therefore, the losses of the four components of the model are:

$$\mathcal{L}_{D_Y} = \mathbb{E}_{y \sim p_{data}(y)}(D_Y(y) - 1)^2 + \mathbb{E}_{x \sim p_{data}(x)}(D_Y(G(x)))^2 \tag{3.11}$$

$$\mathcal{L}_{D_X} = \mathbb{E}_{x \sim p_{data}(x)}(D_X(x) - 1)^2 + \mathbb{E}_{y \sim p_{data}(y)}(D_X(F(y)))^2 \tag{3.12}$$

$$\mathcal{L}_G = \mathbb{E}_{x \sim p_{data}(x)}(D_Y(G(x)) - 1)^2 + \mathbb{E}_{x \sim p_{data}(x)}\|F(G(x)) - x\|_1 \tag{3.13}$$

$$\mathcal{L}_F = \mathbb{E}_{y \sim p_{data}(y)}(D_X(F(y)) - 1)^2 + \mathbb{E}_{y \sim p_{data}(y)}\|G(F(y)) - y\|_1 \tag{3.14}$$

The training procedure is shown in Algorithm 4.

Algorithm 4 CycleGAN training algorithm. The forward pass is performed only once for every sample (rows 6-7), and not recalculated in every loss function.

Require: m = minibatch size

1: **for** number of training iterations **do**
2: sample a minibatch $\mathbf{x}$ of size m from $p_{data}(x)$
3: sample a minibatch $\mathbf{y}$ of size m from $p_{data}(y)$
4: reset the accumulated gradient
5: **for** $i = 1$ to m **do**
6: $\hat{y} \leftarrow G(x^{(i)})$
7: $\hat{x} \leftarrow F(y^{(i)})$
8: calculate $\mathcal{L}_G$ using Equation 3.13
9: calculate $\mathcal{L}_F$ using Equation 3.14
10: back-propagate and accumulate $\mathcal{L}_G$ and $\mathcal{L}_F$ **only** in G and F
11: calculate $\mathcal{L}_{D_Y}$ using Equation 3.11
12: calculate $\mathcal{L}_{D_X}$ using Equation 3.12
13: back-propagate and accumulate $\mathcal{L}_{D_Y}$ and $\mathcal{L}_{D_X}$ **only** in D_Y and D_X
14: **end for**
15: update G and F parameters
16: update D_Y and D_X parameters
17: **end for**

3.3.2 Training improvements

The basic training algorithm presented in Algorithm 4 has been improved with some minor changes, especially to contrast instability and mode collapse issues.

The first enhancement tries to stabilize the training procedure, maintaining a pool of generated images, and updating the discriminator with one image in the pool, rather that the one produced in the last iteration [62]. One of the problems with adversarial learning is that the discriminator only focuses on the last images generated. This lack of memory may cause the training to become unstable, and the reintroduction of artifacts in the generated images that the discriminator had forgotten about. Indeed, the perfect discriminator should be able to correctly classify as fake every image that the generator has ever produced during the entire training phase. Algorithm 4 is slightly modified, adding an image pool of fixed dimension B. At each iteration, the discriminator loss is computed using $m/2$ images generated in this step, and $m/2$ images from the pool, with m being the batch size. After that, $m/2$ images are randomly replaced in the pool with newly generated ones. If $m = 1$, the use of a new or old image is random.

Another improvement is the decay of the learning rate in the second half of the training phase. The model uses the *Adam* optimizer by default, which computes a different learning rate for each parameter. However, the range of the update can be manually set, and is often maintained the same for the entire training process. In [4], the authors of the cycleGAN model used, for their experiments, the same learning rate for half of the epochs of training, and then it was linearly decayed to zero over the other half of the training. Such a mechanism ensures a more stable convergence, especially in the last epochs, where the two models are hopefully near to the equilibrium point, and large updates can cause a high fluctuation in the loss.

The last improvement is the addition of a further loss, to encourage the generators to preserve colors between input and output. In fact, especially for the task of photo generation from paintings, the generators are free to change the tint of input images, even if there is no need to do so. Indeed, as has been said before, image-to-image translation is an ill-posed problem, and more than one output can be valid for the same input. Identity loss is based on the assumption that if a generator translates from the domain X to Y, providing it with an image $y \in Y$ should not produce any transformation. Identity loss

can be formalized as:

$$\begin{aligned}\mathcal{L}_{identity}(G, F) = \mathbb{E}_{y \sim p_{data}(y)} \|G(y) - y\|_1 \\ + \mathbb{E}_{x \sim p_{data}(x)} \|F(x) - x\|_1\end{aligned} \tag{3.15}$$

This loss is added to Equation 3.13 and Equation 3.14 for the corresponding generator. The weight factor is normally 0.5λ, where λ is the weight of the cycle consistency loss.

4

Experiments

> *The principle of science, the definition, almost, is the following: The test of all knowledge is experiment. Experiment is the sole judge of scientific "truth".*
>
> – Richard Feynman, *The Feynman Lectures on Physics* (1964)

In this chapter, the results from several experiments regarding style transfer and image-to-image translation are reported. The main goal is to demonstrate the quality of the adversarial approach in such fields, especially with real-world examples. For this reason, an attempt was made to adopt the cycleGAN model to address the problem of defogging, principally with unpaired data. In the first section, *PyTorch* is briefly described, since it was the library used for developing and running the experiments. After that, an analysis of the problem of defogging is provided. Next, the experiments goals, setups and the procedures adopted for assessing the quality of our work are reported. Finally, the results of the experiments are revealed and described; and consequentially, some conclusions and reflection, based on the results obtained, are presented and discussed.

4.1 PyTorch

PyTorch [63] is an open source machine learning library, derived from *torch*, optimized to run on GPUs, especially with CUDA. PyTorch is based on *tensor* computation, which can be done either in the CPU or the GPU. A tensor is a multidimensional array, that abstracts the notion of a vector or matrix.

PyTorch is divided into modules, and one of the most important is the `autograd` module. This component is a tape-based automatic differentiation library, and it is one of the core features of the framework. Similarly from other machine learning libraries, such as *TensorFlow* or *Caffe*, PyTorch uses a technique called reverse-mode auto-differentiation, which automatically calculate the derivative of the operations, and backpropagated them through the computational graph. The computational graph is a graph that connects every operation performed to variables, linking the results to the sources. As an example, if two tensors are summed together, and the result multiplied with another tensor, the output can be differentiated with regards to a particular value, and then back-propagated through the computational graph, until the two initial tensors. However, differently from Tensorflow or caffe, that have a static view of the world, in PyTorch the computational graph is constructed dynamically, so adding another operation in the chain is possible even at runtime [64].

Another fundamental module of PyTorch is the `nn` module. Such a component is a library that contains utilities for the development of neural networks, and it is deeply integrated with the `autograd` module. The `nn` library contains useful implementation of neural network components, such as convolutional layers, in order to build complex networks in just a few lines of codes.

PyTorch is a complete library, that comprises data loaders and a multiprocessing library, which can be used to speed up the load of the datasets or to share tensors between threads, in order to train the model in parallel.

4.2 Defogging

Images captured under bad weather conditions such as fog, mist or haze, suffer from limited visibility, poor contrast, faded colors and loss of sharpness, which constitute a significant obstacle for computer vision applications, e.g. object detention, tracking, classification and segmentation. Most of the models used in these tasks are usually trained on clear images, so the presence of fog or haze may reduce their performance [65]. Defogging (or dehazing) is the task of removing the fog in a given image, reconstructing the same scene as if it were taken in good weather conditions.

A robust haze removal technique can be useful in many circumstances, both as a preprocessing step and as a stand-alone procedure. For instance, fog has ever constituted a major cause of road accidents, hence a strong defogging algorithm can be extremely helpful for drivers, informing them of hazards that are hidden

behind the mist. Even self-driving cars often are extremely confused in foggy weather, due to the limited visibility and the creation of reflections and glare in the cameras [66]. Moreover, image restoration can be used to improve the quality of surveillance systems that might have difficult in recognizing suspect movements with scarce visibility. Defogging can be used as an independent operation in image enhancement tasks, with the aim of improve the visual quality of photographs taken in bad weather conditions.

In presence of fog or haze particles, the original irradiance received by the camera gets attenuated, proportionally to the distance of the objects. This effect is combined with the scattering of atmospheric light, and the result is a hazy image. The mathematical model of a foggy image can be formulated as [65]:

$$I(x) = J(x)t(x) + \alpha(1 - t(x)) \tag{4.1}$$

where:

- $I(x)$ is the observed hazy image.
- $J(x)$ is the haze-free image.
- $t(x)$ is the scene transmission map.
- α is the atmospheric light.
- x is an individual pixel location in the image.

The scene transmission $t(x)$ can be expressed as function of the depth, as:

$$t(x) = e^{-\beta d(x)} \tag{4.2}$$

where:

- $d(x)$ is the depth of the pixel x with respect to the camera.
- β is the scattering coefficient of the atmosphere.

The goal of a defogging algorithm is to find $J(x)$, given $I(x)$. However, as can be seen from Equation 4.1, the problem is ill-posed, since the equation contains three unknown terms: $J(x)$, $t(x)$ and α.

The complexity of defogging is further increased by the difficulty of creating a large dataset of paired haze and haze-free images. Capturing the same photograph with and without haze is an extremely difficult task, due to the fact that little changes in the camera or an object's position can make the two images not perfectly aligned. Moreover, the level of illumination of the scene must be the same with and without fog. For these reasons, the majority of datasets used for training defogging models consist of synthetic foggy images, where

the fog is artificially added, starting from clear pictures. Nevertheless, synthetic haze does not perfectly resemble real fog, indeed, many models trained with synthetic dataset are not able to correctly reconstruct a clear image if they are fed with a real foggy image [67]. Therefore, an approach based on unpaired datasets, composed of real foggy images and real clear photographs is the preferred approach for resolving the dehaze problem in real conditions. CycleGAN has shown impressive results in many image-to-image translation tasks in different domains [4], so its adoption to tackle the defogging problem has been investigated here.

4.2.1 Related work

In this section, we only consider methods that use a single image to perform the defogging. Earlier algorithms based on multiple images are not considered, due to their limited practical applicability. For a complete review of state-of-the-art defogging methods see [65, 66, 68].

The first single image dehazing method was proposed by Narasimhan *et al.* [69], and it relied on supplied information about the scene structure. One of the most successful methods was developed by Fattal [70], who used an Independent Component Analysis-based method to estimate the albedo and transmission map of a scene. He *et al.* [71] observed an interesting property of outdoor nature scenes with clear visibility: most objects have at least one color channel that is significantly dark (the pixel value for that channel is near zero). Using this property, a method based on dark channel prior was used to estimate the transmission map $t(x)$. One of the drawbacks of the previous methods is computational time. The methods cannot be applied in a real time application, where the depth of the image change from frame to frame. Tarel *et al.* [72] proposed a method which complexity is linear in the number of image pixels. The method use a heuristic to estimate the *atmospheric veil*, which then is used to recover the haze-free image.

Most recent approaches are based on deep learning techniques, which are used for the estimation of atmospheric light and the transmission map. An approach based on convolutional neural networks was proposed by Ren *et al.* [73], where a coarse holistic transmission map is first produced using a CNN, and then refined using a different fine-scale network. Cai *at al.* use another CNN, which they called *DehazeNet* [74], with the goal of learning a direct map between foggy images and the transmission map, without using many learning levels.

Most of the models proposed for defogging are based on the estimation of the transmission map $t(x)$ (the atmospheric light α can be derived, with some

assumption, from $t(x)$ [65]). The fog-free image is then computed in a separate step, using Equation 4.1. None of the approaches described above consider the aesthetic quality of the resulting image in the parameter estimation phase. Only recently have end-to-end models emerged, which produce a clear image directly from an hazy one, starting from the *AOD-Net*, proposed by Li *et al.* [75].

The application of generative adversarial networks to dehazing is even more recent, indeed there are few works present in the literature on this topic. Most notably, the use of adversarial training for the estimation of the transmission map is proposed in [76]. Moreover, the use of GANs for end-to-end models which directly produce the haze-free image without estimating the transmission map, was first developed by Swami *et al.* [77]. In their model, the generator network consists of an autoencoder that transform the foggy image to the corresponding clear picture. In that model, the discriminator is conditioned: it receives a pair of images, the hazy image and the clear one, so its goal becomes to yield the probability that the given pair is ground truth, i.e. the haze-free image is the real one, given the foggy picture, and it was not produced by the generator.

All the machine learning approaches described above are based on the utilization of a paired dataset of foggy and clear images during training. As stated before, those datasets are often synthetic, and for this reason the quality of the model may decrease if it is tested with real hazy photographs. Engin *et al.* [78], recently proposed a model, based on cycleGAN, for unpaired image defogging. However, the dataset used by them is not really unpaired, since it was obtained by data augmentation of the O-Haze dataset [67].

4.3 Experiment design

The main goal of the following experiments was to evaluate the adoption of unpaired image-to-image translation methods, principally based on cycleGAN, to resolve the problem of defogging, especially in conditions of strong natural fog. Moreover, the investigation was conducted using a real unpaired dataset for hazy and clear photos, and not using image patches derived from a paired collection, as in [78]. The fist step was the evaluation of the cycleGAN model with regards to the defogging task, without any further improvement. That served as a baseline for following investigations. Next, the cycleGAN model was enhanced by the addition of perceptual loss, and the results are evaluated in comparison with the simple model. After that, the role of the dataset was

investigated, showing that the choice of the correct data might greatly improve the quality of the generated images. Finally, a comparison with state-of-the-art methods was conducted, followed by a discussion of the results and the critical issues of the proposed and existent defogging methods.

4.3.1 Dataset

The first difficulty was to find a dataset for training and testing the cycleGAN model, with the goal of translating between foggy and clear images. As stated in section 4.2, the majority of datasets used for defogging tasks are paired; and therefore, the fog is often artificially added to clear images, in order to produce paired data. Furthermore, the datasets are often small, including only a few dozens of images; thus they are not adequate for training complex machine learning models.

Below, a brief description of some of the most used haze datasets is provided, followed by an indication of those which were chosen in the following experiments.

D-Hazy [79]

Type	Loc.	Train	Test
Synthetic	Indoor	1449	-

The D-Hazy is a paired dataset composed of 1449 pairs of clear scenes and the corresponding synthetic hazy image. It is based on the NYU depth dataset[1], and contains indoor images of various rooms. Although interiors are not a common place to find fog, this collection is broadly used, mainly because the depth map used for haze generation is much more accurate than in outdoor datasets.

O-Haze [67]

Type	Loc.	Train	Test
Real	Outdoor	45	-

[1] https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html

The O-Haze dataset contains 45 pairs of high resolution images, with and without fog. It is one of the few non-synthetic paired datasets, despite the fact that the fog is not really natural. In fact, the haze was produced using fog machines, and even though it is artificial, the effect can be considered a good approximation of real foggy conditions.

RESIDE [66]

Type	Loc.	Train	Test
Synthetic	Outdoor	313.950	500

Reside is one of the biggest defogging datasets available, and is composed of many subsets. The outdoor training set (OTS) is made up of 8.970 clear images, mainly taken in urban environments. Each haze-free photograph was processed with 25 different levels of fog, producing the 313.950 training images. The synthetic objective testing set (SOTS) is composed of 500 clear pictures, and the corresponding 500 synthetically hazy images. The hybrid subjective testing set (HSTS) contains real and synthetic haze images, and it is used to evaluate state-of-the-art models. Reside contains other sets of haze and haze-free pictures, not used in this dissertation.

FRIDA2 [80]

Type	Loc.	Train	Test
Synthetic	Outdoor	330	-

The FRIDA2 dataset was specifically developed for defogging in urban road scenes, from the point of view of a car driver. The images are not real, but artificially rendered with computer graphics. FRIDA2 comprises 66 different road scene, each of which is associated with 4 different fog levels and the corresponding depth map.

LIVE [81, 82, 83, 84]

Type	Loc.	Train	Test
Real	Outdoor	1000	100

The LIVE database is one of the few unpaired datasets, comprising 500 real fog images and 500 clear photographs. A test set of a further 100 real hazy images is provided.

Since the goal of the experiments was to evaluate the possibility of exploiting unpaired image-to-image translation for removing haze from images, and the desire was to focus on real cases, our models were trained using the LIVE dataset. During testing, the test set of the same collection was used for referenceless metrics, and the SOTS dataset for full-reference ones. In order to compare our approach to the state-of-the-art models, we used the HSTS dataset.

4.3.2 Metrics

The evaluation of existent defogging methods, in most cases, relies on the presence of a ground truth real image, which can be directly compared with the dehazed result. Often, the preferred metrics are *structural similarity* (SSIM) and *peak signal-to-noise ratio* (PSNR), even if the quality of the defogging might not be correctly evaluated by them [66]. In particular, has been demonstrated that SSIM and PSNR do not correlate well with human judgment or with referenceless metrics. Moreover, since a ground truth image is required, the evaluation needs a paired dataset, and, as has been reported in the previous section, in these datasets, the fog is often artificial, and it can be quite different from natural mist. However, the majority of authors still use SSIM and PSNR to evaluate their model, so it was decided to adopt them here, in order to compare this approach to state-of-the-art methods.

In the case of referenceless data, i.e. when the ground truth clear image is not present, the majority of previous works have used visual comparison of images and human evaluation, usually performed via automatic tools such as *Amazon mechanical turk*[2]. Nevertheless, such techniques are subjective, and may be influenced by external factors, such as the competence or the motivation of assessors.

Since the goal of the presented approach is natural scene defogging, evaluating it using only synthetic images may not yield meaningful results, since artificial foggy scenes often present a low level of haze, whereas the goal in this case is to produce a model that can be used even in condition of severely scarce visibility. Unfortunately, a standard referenceless metric for evaluating the quality of a defogged image does not exist. The problem of assessing the

[2] `https://www.mturk.com`

quality of an image without a ground truth is not limited to defogging, but is an open research problem in computer vision [85]. Moreover, most of the methods proposed in recent years are general, and targeted to assess the quality of compressed images. For a wide-ranging review of no-reference metrics used for the evaluation of defogging results see [68].

One of those blind metrics was proposed by Hautière *et al.* [86], and consisted of three different indicators: e, σ and $\bar{r}$. The value of e is based on the number of visible edges in the defogged image, relative to the original foggy picture. An edge is considered visible if the local contrast is above 5%. A high value of e means that the defogging has revealed edges that were invisible in the original image. The value of σ represents the percentage of saturated (black or white) pixels in the defogged image. A good dehazing method should keep this value low, because the contrast has to be restored without over-saturating the image. $\bar{r}$ indicates the quality of the contrast restoration in the defogged image. For a more detailed description of the computation of e, σ and $\bar{r}$ see [86]. This metric was developed specifically for evaluating dehazing algorithms, thus it was chosen as a blind metric to evaluate the presented model.

4.3.3 Set-up of experiments

If not explicitly stated otherwise, for all the experiments the same set-up was used, which is similar to the one used by the authors of the cycleGAN model for their investigations. The model was trained for 200 epochs, using a batch size of one, in order to update the parameters after every sample. The learning rate was initially set to 0.0002, and kept unaltered during the first 100 epochs, and then linearly decayed to zero in the last 100 epochs. The objective function of the discriminators is divided by two, in order to slow down the rate at which they learn relative to generators. The parameters of the neural networks were initialized from a Gaussian distribution with mean 0 and standard deviation 0.02. The weight factor of the cycle consistency loss λ was set to 10, and the weight of the identity loss was set to 0.5λ. The images were scaled to 286×286 pixel, using a bicubic interpolation, and then a random crop of size 256×256 was taken and used as input to the networks.

The experiments were performed using a NVIDIA GeForce 1080 Ti[3] GPU, with 3.584 CUDA cores and 11GB of memory. The server were equipped with 64GB of RAM and two Intel Xeon E5-2650[4] with 16 core each.

[3] `https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti`

[4] `https://ark.intel.com/products/64590/Intel-Xeon-Processor-E5-2650`

4.4 Getting started: cycle defogging

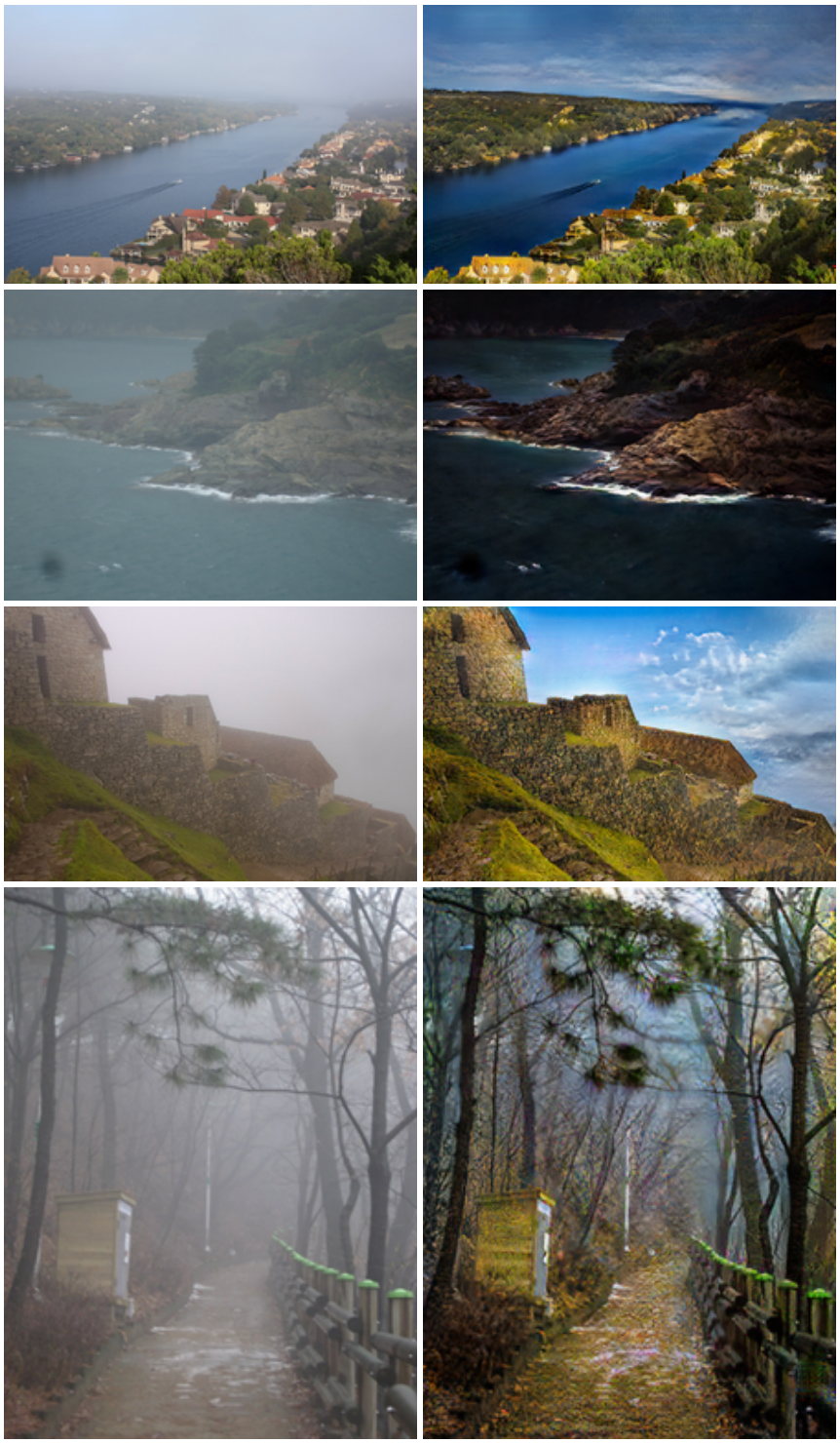
The first experiment was aimed to produce a baseline for evaluating subsequent improvements, in order to appraise the feasibility of defogging real images using a model trained on unpaired data. For this reason, the model consisted of a simple cycleGAN without any change, and it was trained with the LIVE dataset. This basic training configuration was called *training1*.

After the first experiment, however, the results were not satisfactory, since the images with thick fog were translated in a poor manner, almost resembling works of artistic style transfer (see Figure 4.1). After further investigation, the conclusion was reached that the two sets of foggy and clear images, were too different from each other. Indeed, the hazy image set mainly contains pictures of natural landscapes and city views, while the set of clear images includes many photographs of people or animals, and even some indoor scenes. Since cycleGAN tries to translate an image from one data distribution into another, if the two distributions are too diverse, the mapping will be meaningless. This is caused by the presence of both the adversarial loss and the cycle consistency loss at the same time. As an example, if a landscape photograph is translated into a domain composed mainly of photos of people, the former loss pushes the model to change the content of the image, while the latter enforce the generator to maintain the input's characteristic in the output.

Hence, an attempt was made to change the set of haze-free training images to one that was more similar to the collection of foggy pictures. The dataset used for collection style transfer in the original cycleGAN model [4] was chosen, which is composed of 6.287 RGB images downloaded from Flickr, using the combination of tags *landscape* and *landscapephotography*. We called this training configuration *training2*. The training was repeated using *training2* configuration, and a comparison between *training1* and *training2* configurations is shown in Figure 4.1. Particularly good images are shown in Figure 4.2.

(a) (b) (c)

Figure 4.1: The results obtained by training the model with different sets of clear images. (a) real hazy images from the LIVE test set, (b) the dehazing produced by training the model with *training1* configuration, (c) the results obtained using *training2* configuration.



Real foggy image Defogged image

Figure 4.2: Some good examples of defogging, using the cycleGAN model. These examples demonstrate the validity of an unpaired approach, even in case of fairly different collections of haze and haze-free images.

4.5 Enhancing the model: cycle perceptual loss

The example presented in Figure 4.2 demonstrate the validity of unpaired defogging, even if the model is the classical, general purpose, cycleGAN, without any domain specific improvement. However, some dehazed images manifested blurry borders, especially in the presence of thick fog. This phenomenon was particularly accentuated in city views, where the buildings were often confused with clouds, and faded into the background. Some examples of this phenomenon are shown in Figure 4.3.

Real foggy image Defogged image

Figure 4.3: Some example of poor defogging, where some elements of the image are transformed in clouds and blended into the background.

4.5.1 Cycle perceptual loss

Many models for style transfer do not evaluate the results exclusively in image space, but also in feature space [52, 54]. A pre-trained CNN, often trained for object detection, is utilized as a feature extractor, and a loss is computed between the features of the target and the output images. Such a comparison, performed in the feature space, is often called *perceptual* loss, since it is based on the fundamental characteristic of images, rather than their representation.

CycleGAN does not have a reference image, since it operates with an unpaired dataset; therefore, the only way to include a perceptual loss is to transform it into a *cycle perceptual loss*, calculated between the original image x and the recovered image $\hat{x} = F(G(x))$. A similar approach was developed by Engin *et al.* in [78], and has shown an improvement in the dehazing quality, when compared to cycle consistency loss alone. Their implementation uses VGG-16 [53] as a feature extractor, and the features of interest are extracted from the 2nd and the 5th pooling layers.

The cycle perceptual loss can be expressed as a mean squared error loss, utilizing the L2-norm, similarly to the content loss in the works of Gatys and Johnson for neural style transfer [52, 54]. The cycle perceptual loss is formulated as:

$$\begin{aligned}\mathcal{L}_{perceptual}(G, F) = \mathbb{E}_{p_{data}(x)}\|\phi(x) - \phi(F(G(x)))\|_2^2 \\ + \mathbb{E}_{p_{data}(y)}\|\phi(y) - \phi(G(F(y)))\|_2^2\end{aligned} \tag{4.3}$$

where $\phi(\cdot)$ represents features extracted from the 2nd and 5th pooling layers of VGG-16. As with the cycle consistency loss, both G and F aim to minimize this cost, but neither of them have full control over the loss, because the other generator takes part in its formulation.

The full objective of the cycle perceptual cycleGAN can be expressed as:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y) + \mathcal{L}_{GAN}(F, D_X) \\ + \lambda\mathcal{L}_{cycle}(G, F) + \gamma\mathcal{L}_{perceptual}(G, F)\end{aligned}$$

where γ controls the relative importance of the cycle perceptual loss in the full objective function.

Implementation details

The cycle consistency loss was added to the cycleGAN model as a separate module, which can be included or excluded depending on the experiment set-up. PyTorch, natively, makes the VGG pretrained on ImageNet models available, through the `torchvision` package. The networks had been trained using images normalized in the (0,1) range and with the mean and standard deviation of the *ImageNet* dataset[5]. However, the cycleGAN generators output images in the range (-1,1), so they have to be normalized before presentation to the feature extractor.

[5] `http://www.image-net.org`

In order to extract the features from the hidden layers of the VGG-16 network, an approach called `forward_hook` was used, which consists in dynamically attaching a custom function to a neural network layer, which will be called during the forward pass, when the data go through the desired layer. The custom forward hook simply stores the output feature map, in order to use it during the loss calculation.

4.5.2 Evaluation

First, the adoption of the cycle perceptual loss was evaluated in a objective manner, using a paired dataset and calculating the SSIM and the PSNR between the dehazed images and the corresponding ground truth. Since the models had been trained with nature and urban scenes, the test data should be composed of similar images, in order to evaluate the dehazing in a meaningful way. Thus the adoption of cycle perceptual loss was evaluated using the SOTS dataset. The images were scaled to 400×400 pixel with bicubic interpolation, and the SSIM and PSNR were calculated every ten epochs of learning, in order to track the progress of the two models. The relative weight of the perceptual cycle loss (γ) was set to 0.1, in order to make its value similar to the cycle consistency loss. The results of these evaluations are presented in Table 4.1.

As can be seen from Table 4.1, the addition of cycle perceptual loss has improved the performance of the model, especially in the fist epochs. Moreover, a more stable behavior is observed in the model augmented with cycle perceptual loss. In fact, the two metrics did not fluctuate as in the simple model, and the advancement of the training is more coupled with defogging improvements (see epochs 30-80). Some graphical comparisons between the results of the two models are shown in Figure 4.4.

The adoption of cycle perceptual loss was also evaluated using the blind metric outlined in subsection 4.3.2. Since the assessment does not require a paired dataset, the test set of LIVE dataset was used, which is composed of 100 real foggy images. However, only 99 of them were used, because one photograph does not present any edge with local contrast above 5%, so its value of e is infinite. The results of the evaluation are reported in Table 4.2.

	CycleGAN		CycleGAN + $\mathcal{L}_{perceptual}$	
Epoch	**SSIM**	**PSNR (dB)**	**SSIM**	**PSNR (dB)**
10	0.6074	14.54	0.7770	18.79
20	0.7586	18.32	0.8264	20.36
30	0.7616	17.29	0.7898	18.28
40	0.8110	19.56	0.8071	19.99
50	0.8101	19.53	0.8059	19.67
60	0.7668	17.11	0.8231	20.41
70	0.8018	19.35	0.8280	19.73
80	0.7671	17.39	0.8320	20.60
90	0.7610	16.88	0.8232	19.67
100	0.7683	17.92	0.8172	20.13
110	0.8254	19.65	0.7907	18.49
120	0.7936	18.37	0.8211	20.03
130	0.8157	19.10	0.8327	20.56
140	0.8171	19.48	0.8509	21.33
150	0.7895	17.24	0.8346	20.27
160	0.8041	18.40	0.8306	19.80
170	0.8068	18.55	0.8243	19.86
180	0.7992	18.29	0.8224	19.99
190	0.7976	18.25	0.8269	20.16
200	0.7975	18.33	0.8259	20.16

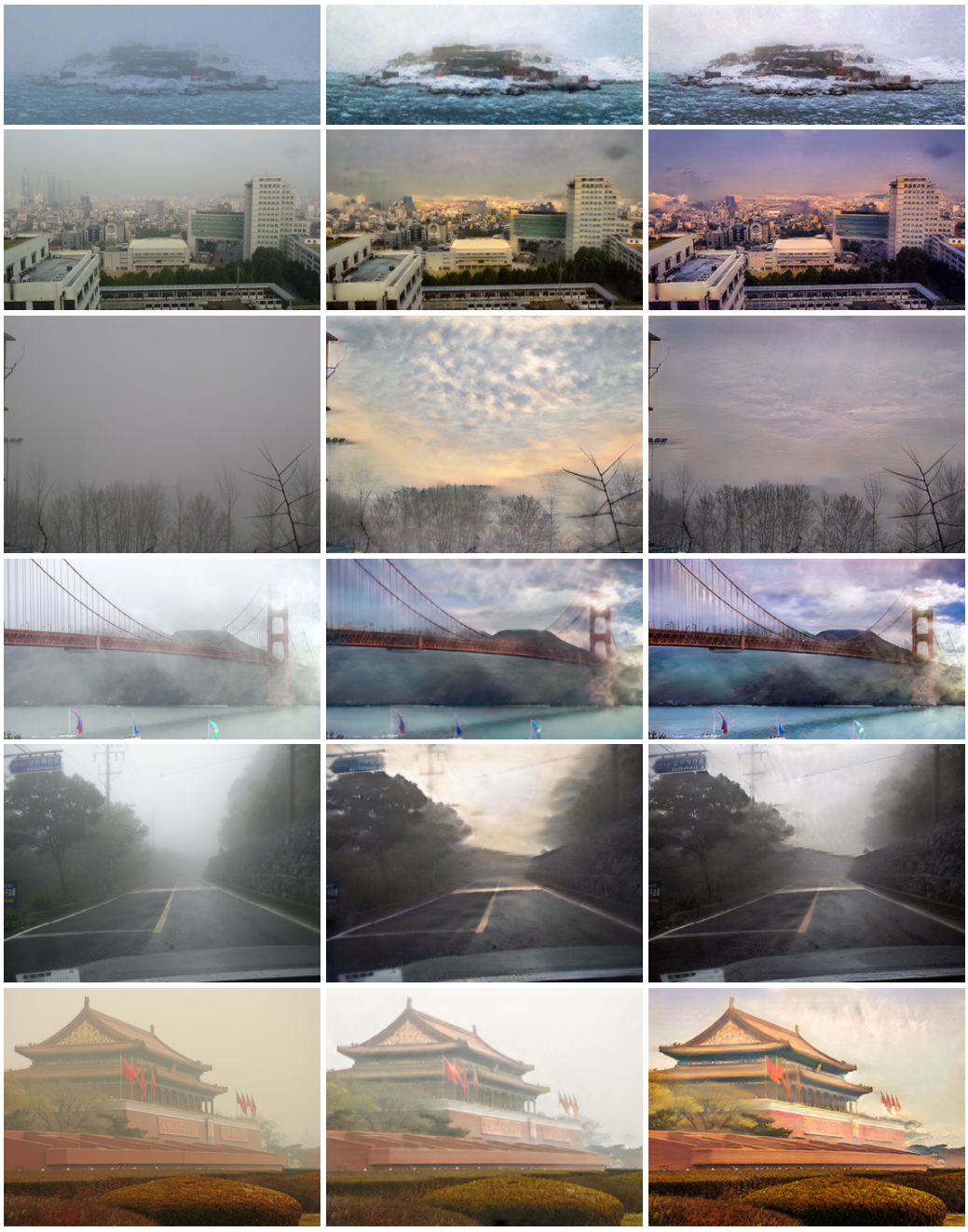
Table 4.1: The progress of SSIM and PSNR, calculated on the SOTS dataset, through the learning epochs. The best values of every model are highlighted in blue.

	CycleGAN			CycleGAN + $\mathcal{L}_{perceptual}$		
	$\boldsymbol{e}$	$\boldsymbol{\sigma}_{(\%)}$	$\bar{\boldsymbol{r}}$	$\boldsymbol{e}$	$\boldsymbol{\sigma}_{(\%)}$	$\bar{\boldsymbol{r}}$
Mean	5.546	0.0022	1.686	6.559	0.0011	2.094
Median	0.1182	0	1.463	0.3613	0	1.796

Table 4.2: Evaluation of dehazed images using the metrics from [86], using the LIVE test set of natural foggy images. The two models were evaluated using the best resulting parameters in Table 4.1.

In general, the cycle perceptual loss makes the images less blurred and less impressionistic, preserving the original colors more accurately. The objects represented in the photographs are more defined, and the contrast is intensified, especially for small details, such as tree branches. Overall, the quality of the dehazing was slightly enhanced by the addition of the cycle perceptual loss,

and this can be verified both from full reference and blind metrics.



Real foggy image | CycleGAN | CycleGAN + $\mathcal{L}_{perceptual}$

Figure 4.4: The effect of the perceptual cycle loss on defogging. The results of the two models were obtained using the best parameters that resulted from the objective study reported in Table 4.1.

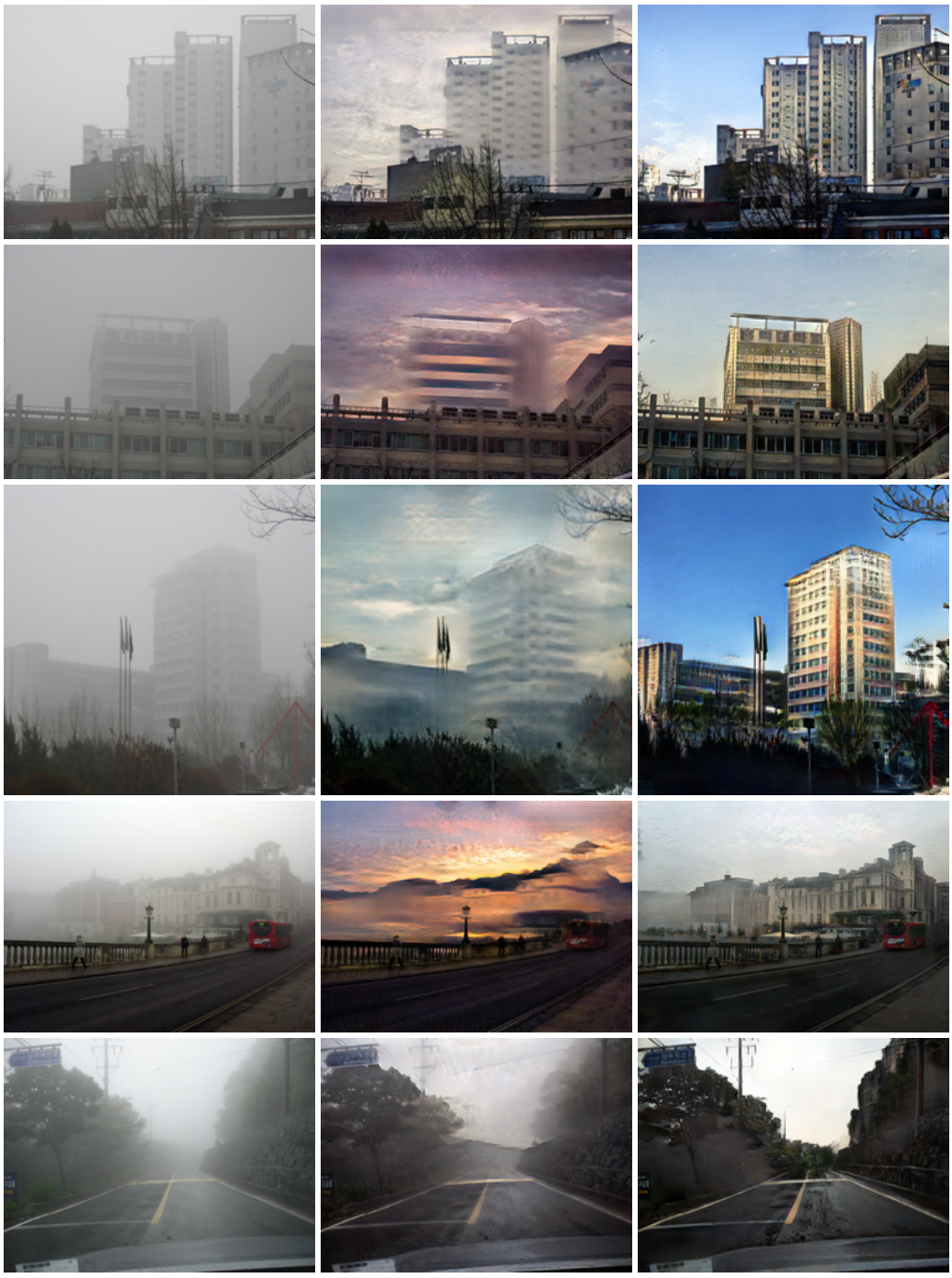
4.6 The role of dataset

The extension of the basic cycleGAN model with the perceptual cycle loss has resulted in an improved quality of the defogging, preserving the true colors more faithfully and enhancing the contrast. However, the problem of faded objects remained, especially for scenes captured in thick fog. This issue was particularly relevant in urban views, where buildings were often transformed in hybrid objects similar to clouds, and fused with the background. Moreover, some defogged images seemed impressionistic, as clouds and sunset colors were added during the dehazing. Apart from artistic purposes, that kind of images should be avoided, even if they are a plausible translation of a foggy image, due to the ill-posed nature of the problem.

Some techniques for reducing the problem of artistic translations were investigated, and following the reasoning reported in section 4.4 focus was primarily made on the importance of the training dataset. The models previously evaluated had used the *training2* configuration, which included a training set that was a combination of real foggy images from the LIVE dataset [81] and clear photograph from the real image set used for collection style transfer in [4]. The latter collection, as reported earlier, is composed mainly of landscape photographs, and most of the images are aesthetically appealing, containing colorful skies and interesting lighting.

To evaluate the importance of the training set, the clear image collection was substituted with a selection of clear images from the OTS (Outdoor Training Set) included in the RESIDE dataset [66]. This collection is composed of 2.651 images, manually selected in order to include only daytime photographs, with clear skies and good lighting. The images consist mostly of city view, so it was expected that the defogging would be more effectively for urban scene, and consequently less efficacious for landscapes. This new training configuration was called *training3*.

The same training parameters of the cycle perceptual model were maintained, the only change being the training set. Some results are shown in Figure 4.5 and Figure 4.6.



(a) (b) (c)

Figure 4.5: The effect of the training dataset on the quality of the defogging. (a) real foggy images, (b) results of the model trained with *training2* configuration, (c) results of the model trained with *training3* configuration.

(a) (b) (c)

Figure 4.6: The effect of the training dataset on the quality of the defogging II. (a) real foggy images, (b) results of the model trained with *training2* configuration (c) results of the model trained with *training3* configuration.

As can be seen from the above examples, the role of the training dataset is fundamental for obtaining valuable results. The cycleGAN model, in fact, is based on the estimation of two distribution, and how to transform a sample from one distribution to another. For instance, if the set of clear images does

not include buildings, the model does not know how to correctly translate them if they are present in the hazy set. Thus, it substitutes them with something known, in the case of *training2* configuration, with clouds. Using a training set as similar as possible to the target domain has proved to be at least as important as the addition of perceptual cycle loss.

The model trained with the *training3* configuration was evaluated using the same metrics adopted before: SSIM and PSNR. The same test set was used: SOTS, which is composed mainly of city views. The results are reported in Table 4.3.

CycleGAN + $\mathcal{L}_{perceptual}$ (*training3* configuration)		
Epoch	**SSIM**	**PSNR (dB)**
10	0.7897	22.10
20	0.8318	22.38
30	0.8382	22.47
40	0.8281	22.24
50	0.8572	22.86
60	0.8560	22.91
70	0.8696	22.61
80	0.8709	23.17
90	0.8778	23.19
100	0.8785	23.00
110	0.8890	23.44
120	0.8903	23.36
130	0.8904	23.22
140	0.8842	23.41
150	0.8960	23.46
160	0.8930	23.55
170	0.8935	23.67
180	0.8889	23.28
190	0.8899	23.19
200	0.8900	23.19

Table 4.3: The progress of SSIM and PSNR, calculated on the SOTS dataset, through the learning epochs. The best values are highlighted in blue. The best values of the model trained with *training2* configuration are SSIM = 0.8509, PSNR = 21.33.

The results reported above prove a considerable improvement in the quality of the defogged images. However, the credit goes mainly to the utilization of a training set that was more similar to the test images collection. This im-

plication should be considered when an unpaired image-to-image translator is trained, due to its high dependency on the data distribution of the two domains. Such a consideration poses a question about the possibility of building a general defogger from unpaired image sets. This problem goes beyond the scope of this thesis, and will be investigated in future work.

4.7 Comparison with state-of-the-art methods

In this section a comparison is carried out between the unpaired model presented above and some state-of-the-art defogging algorithms and machine learning approaches.

4.7.1 Reference-based comparison

First, an objective comparison was performed, using the full reference metrics SSIM and PSNR. For their evaluation, ground truth images were required, so the HSTS dataset, included in the RESIDE release, was used. The choice of that test set was driven by the fact that it was adopted as a test set by Li *et al.* in their comprehensive study of defogging techniques [66], thus the method presented in this dissertation could be compared using the same baseline. The pix2pix model [50] has been included in order to assess differences in image-to-image translation between paired and unpaired approaches, using generative adversarial networks as a ground model. Pix2pix was trained with the SOTS dataset for 200 epochs. Table 4.4 displays the detailed score of each technique in terms of SSIM and PSNR.

	DCP [71]	FVR [72]	BCCR [87]	GRM [88]	CAP [89]	NLD [90]
PSNR	14.84	14.48	15.08	18.54	21.53	18.92
SSIM	0.7609	0.7624	0.7382	0.8184	0.8727	0.7411
	DehazeNet [74]	MSCNN [73]	AOD-Net [75]	Pix2Pix [50]	**Our**	
PSNR	24.48	18.64	20.55	24.22	23.48	
SSIM	0.9153	0.8168	0.8973	0.8891	0.8868	

Table 4.4: Average PSNR and SSIM of dehazing results on HTST. Data of other methods, except pix2pix, are taken from [66]. The above methods are based on natural or statistical prior, the methods below are learning-based. The top-3 performances are highlighted using red, blue and cyan, respectively.

The method adopted in this dissertation exhibits results in line with the state-of-the-art learning-based approaches, achieving the third PSNR score and the fourth SSIM result. Indeed, the presented method is outperformed only by DehazeNet and Pix2Pix. However, all the other learning methods are based on paired data, and a ground truth clear image (or the true transmission map) is used during loss calculation. Moreover, the test was conducted on a synthetic fog dataset, while the presented model was trained using a set of natural foggy images, where, in many cases, the haze level is much higher than in testing images.

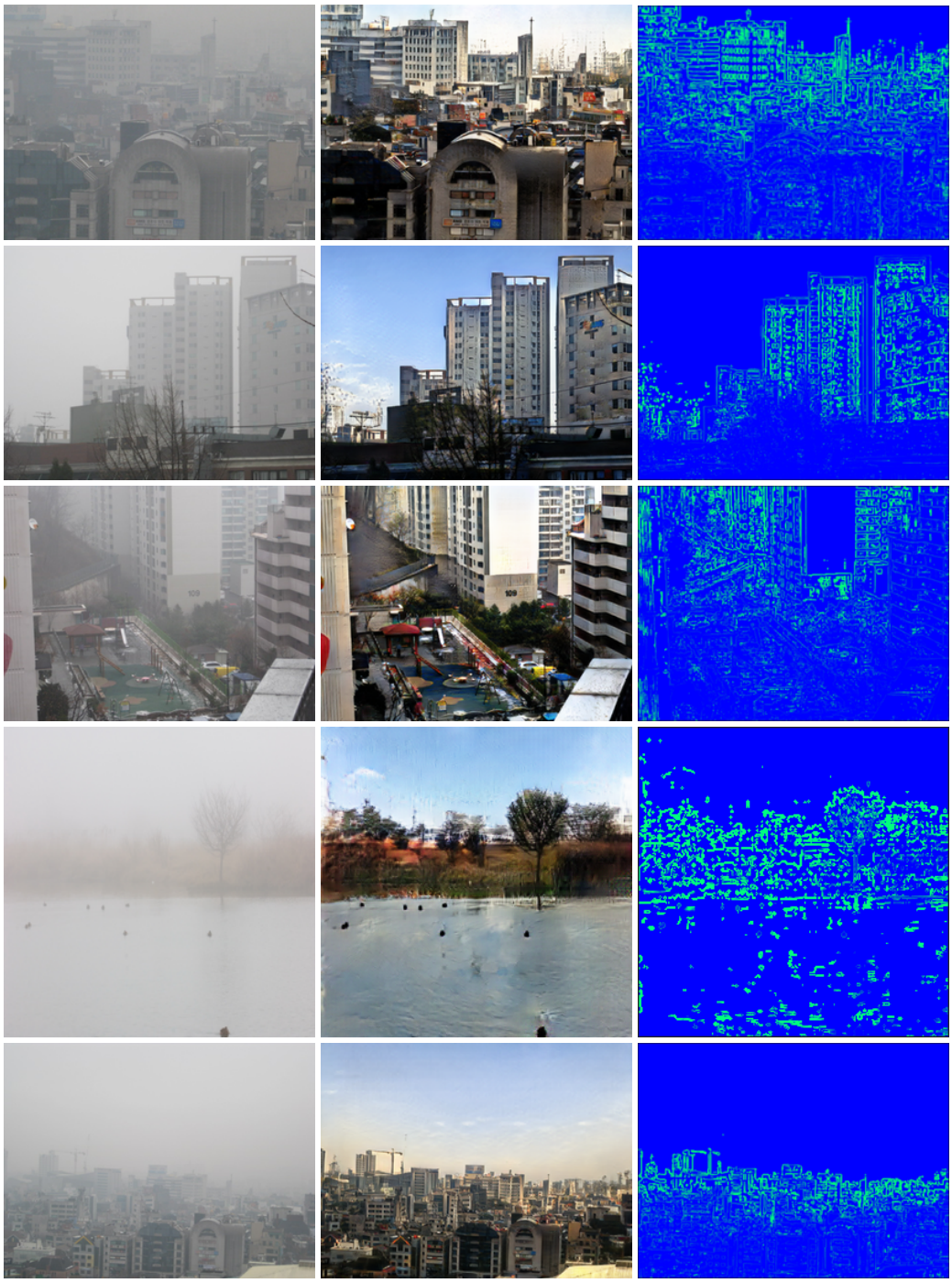
4.7.2 Referenceless comparison

The blind metrics proposed by Hautière *et al.* [86] and described in subsection 4.3.2 was used in the current study. The model trained with the *training3* configuration was evaluated, using the parameters of epoch 170. As for the comparison made in subsection 4.5.2, the test set of the LIVE dataset was adopted, using only 99 images, since one photograph was too foggy for being used with this metric. Many authors have adopted the Hautière *et al.* metrics in their work, but it was often used for evaluating single images. Even if this evaluation cannot be directly compared with state-of-the-art models, it was included here, in order to present a baseline for further research work. The results of the evaluation are reported in Table 4.5.

	e	$\sigma_{(\%)}$	$\bar{r}$
Mean	33.03	0.0214	3.271
Median	0.741	0	2.596

Table 4.5: Mean and median of the Hautière *et al.* metric [86], calculated on the LIVE test set of natural foggy images. The model used was trained with the *training3* configuration.

The results presented above demonstrate that the unpaired defogging method used in this dissertation is competitive, even in the presence of thick natural fog. The high average value of e can be explained by the fact that some images do not present many edges with local contrast above 5%, and therefore, even a small improvement in the quality might result in a high number of new edges in the recovered image. The median values are more significant, and they prove a contrast and quality enhancement of the defogged images. Some examples of dehazed images and the heat map of visibility enhancement, calculated following [86] method, are shown in Figure 4.7 and Figure 4.8.



(a) (b) (c)

Figure 4.7: Some example of defogging using the unpaired approach adopted in this dissertation. (a) real hazy image (b) the resulting defogged image (c) contrast enhancement map.

(a) (b) (c)

Figure 4.8: Some example of defogging using the unpaired approach adopted in this dissertation II. (a) real hazy image (b) the resulting defogged image (c) contrast enhancement map.

5

Conclusions and future works

Considerate la vostra semenza:
fatti non foste a viver come bruti,
ma per seguir virtute e canoscenza.

Consider well the seed that gave you birth:
you were not made to live your lives as brutes,
but to be followers of worth and knowledge.

– Dante Alighieri, *The Divine Comedy, Inferno, Canto XXVI*

In this section, some conclusions about the work carried out in this dissertation are drawn, highlighting the strengths and the weaknesses of the presented unpaired approach to defogging. Then, some future improvements, both theoretical and practical, are proposed.

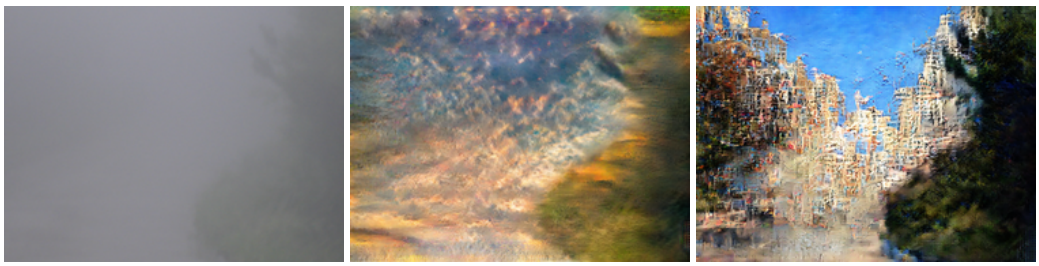
5.1 Conclusions

The goal of the experiments was to prove the effectiveness of an unpaired approach to defogging, using models and techniques from style transfer and image-to-image translation.

The qualitative results obtained using full-reference metrics, and the referenceless ones achieved using the Hautière *et al.* approach [86], demonstrate the validity of the unpaired method used in the present study, especially in the case of thick fog. The addition of cycle perceptual loss improved the quality of dehazed images, in particular, restoring sharpness and color vividness

of the scene. The high visual quality of the results can be attributed to the proven image generation capabilities of generative adversarial networks, especially in referenceless scenarios. The comparison with state-of-the-art methods has demonstrated that the unpaired approach developed in this thesis is in line with the most recent learning models, and can rival methods trained with paired couples of foggy and clear images.

On the other hand, our model is prone to produce artifacts on the generated results, particularly if the visibility on the hazy image is near zero. This can be explained by the fact that the cycleGAN model learns a mapping between two image distributions, which are estimated through the training data. The model is forced, by the adversarial loss, to produce images similar to those of the other distribution. If a generator maps images from domain X to Y, and a nearly uniform image is presented to it, the generator is forced to add the more probable objects present in Y images to the result, since a uniform image is not a probable item of Y. For this reason, in the presence of very thick fog, and almost uniform images, the model adopted produces unrealistic results, populated with the most probable characteristic of the clear images test set. An example of this phenomenon can be seen in Figure 5.1.



(a) (b) (c)

Figure 5.1: An example of the production of artifacts for extremely foggy images. In the absence of clear objects in the input, the model inserts the most common characteristics of the training data. (a) real foggy image (b) model trained with the *training2* configuration (c) model trained with the *training3* configuration.

This behavior may be controlled using a conditional approach, i.e. the generator does not receive only the foggy image, but some other data that can be used for producing better results, such as an estimated depth map. Another approach might be to present the same image many times to the generator, every time with a different noise vector added to the picture. After that, the resulted images are compared, and the artifacts that appear only in one of them will be removed.

5.2 Future works

In this section, a number of possible improvement to the work carried out in this dissertation are discussed. They are conveniently divided into the following four categories:

Use a more targeted dataset In section 4.6, was analyzed the importance of the training set for the quality of the defogging, and resulted that using a dataset as similar as possible to the target domain is effective to refine the model. Unfortunately, the majority of defogging datasets are quite general, and not targeted to a specific domain. The unpaired nature of the approach developed in this thesis simplify the acquisition of a dataset, even for specific domains. The gathering of such data collections may improve the performance of the presented model, and can assess in a more accurate manner the quality of the defogging. Apart from meliorate the model, the collection of a domain-specific dataset of real foggy images might be of interest for the entire "defogging" research community, since it can be used for the evaluation of existent models in real use cases. The creation of a general defogger can also be achieved using targeted datasets: instead of using a huge collection of different images, the model can be trained with one dataset at a time, and updated with continual learning techniques [91].

Use conditioning information In all the experiments, the presented model was trained without additional information, apart the input foggy image. Mirza *et al.* [42] demonstrated that the addition of information can help the generator to produce better results. Thus, the single image can be enriched with supplementary data, such as an estimated depth map or images from an infrared camera. These additional data are often available in vision systems of intelligent cars, and they may enhance the quality of the defogging, removing artifacts and producing images more similar to the real haze-free image. Recently, a system for recovering an approximated ground truth picture using hardware already present in self-driving cars has been presented [92]. This approach can be used to condition the generator with an estimated ground truth, or to collect datasets of hazy and approximated haze-free images.

Evaluate the model with task driven metrics The metrics used for assessing the quality of the defogging, and to compare the quality of the presented model against state-of-the-art methods, are not exempt of defects. In particular, SSIM and PSNR often do not align well with human evaluation or blind metrics, and the referenceless metric proposed by Hautière *et al.* [86] may be meaningless if in the result are present many artifacts. Thus, defogging models should be evaluated using *task-driven* metrics, rather than solely on the image quality, which is a fairly abstract concept. In a defogged image should be easier to detect objects, so a pretrained object detector can be used to evaluate the quality of the defogging. The presented model should be tested using the *Real-world Task-Driven Testing Set* (RTTS), included in the RESIDE dataset [66]. This dataset contains 4.322 foggy images annotated with objects categories and bounding boxes. Using an evaluation metrics based on a real task can yield more significant results, especially if the defogging is used as a preprocessing step in a computer vision pipeline.

Use videos instead of images The cycleGAN model was developed to perform image-to-image translation, thus the presented model works within the images domain. However, many tasks related to defogging use video stream as a data source, e.g. surveillance systems or intelligent cars. A model for high resolution video-to-video translation has been recently proposed by Wang *et al.* [93]. Thus, the present model can be adapted to work with videos. Apart from the obvious complications, working with videos may have some advantages, such as exploiting the data stream to remove artifacts that are present in only few frames, or regularizing the output using the previous results.

Bibliography

[1] Terrence J. Sejnowski. *The Deep Learning Revolution*. MIT Press, 2018.

[2] Ian J. Goodfellow et al. "Generative Adversarial Nets". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 2672–2680.

[3] Ahmed M. Elgammal et al. "CAN: Creative Adversarial Networks, Generating "Art" by Learning About Styles and Deviating from Style Norms". In: *ICCC*. 2017.

[4] J. Zhu et al. "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017, pp. 2242–2251.

[5] Charles I. Abramson and Ana M. Chicas-Mosier. "Learning in Plants: Lessons from Mimosa pudica". In: *Frontiers in Psychology* 7 (2016), p. 417. ISSN: 1664-1078.

[6] Thomas M. Mitchell. *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077, 9780070428072.

[7] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN: 0136042597, 9780136042594.

[8] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020, 9780262018029.

[9] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering, 2007. ISBN: 049508252X.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[11] E. R. Kandel, J. H. Schwartz, and T. M. Jessell. *Principles of neural science*. 6th. Upper Saddle River, NJ, USA: McGraw-Hill, 2000.

[12] F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books, 1962.

[13] Richard H. R. Hahnloser et al. “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit”. In: *Nature* 405 (2000), pp. 947–951.

[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323.

[15] P. Ramachandran, B. Zoph, and Q. V. Le. “Searching for Activation Functions”. In: *ArXiv pre-prints* (Oct. 2017). arXiv: 1710.05941.

[16] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 1435-568X.

[17] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural Networks* 61 (2015), pp. 85–117. ISSN: 0893-6080.

[18] Andreas Griewank. “Who Invented the Reverse Mode of Differentiation?” In: *Documenta Mathematica, Extra Volume: Optimization Stories* (2012), pp. 55–64.

[19] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning Representations by Back Propagating Errors”. In: *Nature* 323 (Oct. 1986), pp. 533–536.

[20] S. Ruder. “An overview of gradient descent optimization algorithms”. In: *ArXiv e-prints* (Sept. 2016). arXiv: 1609.04747.

[21] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 0028-0836.

[22] Yoshua Bengio. “Learning Deep Architectures for AI”. In: *Found. Trends Mach. Learn.* 2.1 (Jan. 2009), pp. 1–127. ISSN: 1935-8237.

[23] A. Antoniou, A. Storkey, and H. Edwards. “Data Augmentation Generative Adversarial Networks”. In: *ArXiv pre-prints* (Nov. 2017). arXiv: 1711.04340.

[24] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[25] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, 2015, pp. 448–456.

[26] David H Hubel and Torsten N Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of physiology* 160.1 (1962), pp. 106–154.

[27] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 0018-9219.

[28] J.T. Springenberg et al. "Striving for Simplicity: The All Convolutional Net". In: *ICLR (workshop track)*. 2015.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105.

[30] A. Radford, L. Metz, and S. Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *ICLR (workshop track)*. 2016.

[31] Ian Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *ArXiv e-prints* (Dec. 2017). arXiv: 1701.00160.

[32] Andrew Y. Ng and Michael I. Jordan. "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 841–848.

[33] G. E. Hinton and T. J. Sejnowski. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1". In: ed. by David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group. Cambridge, MA, USA: MIT Press, 1986. Chap. Learning and Relearning in Boltzmann Machines, pp. 282–317. ISBN: 0-262-68053-X.

[34] A. Creswell et al. "Generative Adversarial Networks: An Overview". In: *IEEE Signal Processing Magazine* 35 (Jan. 2018), pp. 53–65.

[35] John F. Nash. "Equilibrium points in n-person games". In: *Proceedings of the National Academy of Sciences* 36.1 (1950), pp. 48–49. ISSN: 0027-8424.

[36] M. Arjovsky and L. Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". In: *ArXiv e-prints* (Jan. 2017). arXiv: 1701.04862.

[37] Tim Salimans et al. "Improved Techniques for Training GANs". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 2234–2242. ISBN: 978-1-5108-3881-9.

[38] L. Metz et al. "Unrolled Generative Adversarial Networks". In: *ArXiv e-prints* (Nov. 2016). arXiv: 1611.02163.

[39] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 2818–2826.

[40] L. Theis, A. van den Oord, and M. Bethge. “A note on the evaluation of generative models”. In: *International Conference on Learning Representations*. Apr. 2016.

[41] I. J. Goodfellow, J. Shlens, and C. Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *ArXiv e-prints* (Dec. 2014). arXiv: 1412.6572 [stat.ML].

[42] M. Mirza and S. Osindero. “Conditional Generative Adversarial Nets”. In: *ArXiv e-prints* (Nov. 2014). arXiv: 1411.1784.

[43] M. Arjovsky, S. Chintala, and L. Bottou. “Wasserstein GAN”. In: *ArXiv e-prints* (Jan. 2017). arXiv: 1701.07875 [stat.ML].

[44] H. Huang, P. S. Yu, and C. Wang. “An Introduction to Image Synthesis with Generative Adversarial Nets”. In: *ArXiv e-prints* (Mar. 2018). arXiv: 1803.04469.

[45] T. Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variations”. In: *ICLR (workshop track)*. 2018.

[46] Christian Ledig et al. “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 105–114.

[47] Xiaodan Liang et al. “Dual Motion GAN for Future-Flow Embedded Video Prediction”. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 1762–1770.

[48] J. Donahue, P. Krähenbühl, and T. Darrell. “Adversarial Feature Learning”. In: *ArXiv e-prints* (May 2016). arXiv: 1605.09782.

[49] A. Makhzani et al. “Adversarial Autoencoders”. In: *ArXiv e-prints* (Nov. 2015).

[50] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 5967–5976.

[51] V. Dumoulin, J. Shlens, and M. Kudlur. “A Learned Representation For Artistic Style”. In: *International Conference on Learning Representations*. Apr. 2017.

[52] L. A. Gatys, A. S. Ecker, and M. Bethge. “Image Style Transfer Using Convolutional Neural Networks”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016, pp. 2414–2423.

[53] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *ArXiv e-prints* (Sept. 2014). arXiv: 1409.1556.

[54] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". In: *ECCV*. 2016.

[55] G. Ghiasi et al. "Exploring the structure of a real-time, arbitrary neural artistic stylization network". In: *ArXiv e-prints* (May 2017). Accepted as an oral presentation at *British Machine Vision Conference* (BMVC) 2017.

[56] Jun-Yan Zhu et al. "Toward multimodal image-to-image translation". In: *Advances in Neural Information Processing Systems*. 2017.

[57] Fujun Luan et al. "Deep Photo Style Transfer". In: *arXiv preprint* (2017). arXiv: `1703.07511`.

[58] Richard W Brislin. "Back-translation for cross-cultural research". In: *Journal of cross-cultural psychology* 1.3 (1970), pp. 185–216.

[59] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[60] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *CoRR* abs/1603.07285 (2016).

[61] Xudong Mao et al. "Least Squares Generative Adversarial Networks". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2813–2821.

[62] A. Shrivastava et al. "Learning from Simulated and Unsupervised Images through Adversarial Training". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 2242–2251.

[63] Adam Paszke et al. *Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration*. `https://pytorch.org`. Accessed: 2018-11-30.

[64] Adam Paszke, Sam Gross, and Adam Lerer. "Automatic differentiation in PyTorch". In: 2017.

[65] Yu Li et al. "Haze Visibility Enhancement: A Survey and Quantitative Benchmarking". In: *Computer Vision and Image Understanding* 165 (2017), pp. 1–16.

[66] Boyi Li et al. "Benchmarking Single-Image Dehazing and Beyond". In: *IEEE Transactions on Image Processing* 28 (2018), pp. 492–505.

[67] Codruta O. Ancuti et al. "O-HAZE: a dehazing benchmark with real hazy and haze-free outdoor images". In: *IEEE Conference on Computer Vision and Pattern Recognition, NTIRE Workshop*. NTIRE CVPR'18. Salt Lake City, Utah, USA, 2018.

[68] Y. Xu et al. "Review of Video and Image Defogging Algorithms and Related Studies on Image Restoration and Enhancement". In: *IEEE Access* 4 (2016), pp. 165–188.

[69] Srinivasa G. Narasimhan and Shree Nayar. "Interactive Deweathering of an Image using Physical Models". In: *IEEE IEEE Workshop on Color and Photometric Methods in Computer Vision, In Conjunction with ICCV*. Oct. 2003.

[70] Raanan Fattal. "Single Image Dehazing". In: *ACM Trans. Graph.* 27.3 (Aug. 2008), 72:1–72:9. ISSN: 0730-0301.

[71] K. He, J. Sun, and X. Tang. "Single Image Haze Removal Using Dark Channel Prior". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.12 (Dec. 2011), pp. 2341–2353. ISSN: 0162-8828.

[72] Jean-Philippe Tarel and Nicolas Hautière. "Fast visibility restoration from a single color or gray level image". In: *2009 IEEE 12th International Conference on Computer Vision* (2009), pp. 2201–2208.

[73] Wenqi Ren et al. "Single Image Dehazing via Multi-scale Convolutional Neural Networks". In: *ECCV*. 2016.

[74] Bolun Cai et al. "DehazeNet: An End-to-End System for Single Image Haze Removal". In: *IEEE Transactions on Image Processing* 25 (2016), pp. 5187–5198.

[75] Boyi Li et al. "AOD-Net: All-In-One Dehazing Network". In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[76] He Zhang, Vishwanath Sindagi, and Vishal M. Patel. "Joint Transmission Map Estimation and Dehazing using Deep Networks". In: *CoRR* abs/1708.00581 (2017).

[77] Kunal Swami and Saikat Kumar Das. "CANDY: Conditional Adversarial Networks based Fully End-to-End System for Single Image Haze Removal". In: *CoRR* abs/1801.02892 (2018).

[78] Deniz Engin, Anıl Genç, and Hazım Kemal Ekenel. "Cycle-Dehaze: Enhanced CycleGAN for Single Image Dehazing". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2018.

[79] Christophe De Vleeschouwer Cosmin Ancuti Codruta O. Ancuti. "D-HAZY: A Dataset to Evaluate Quantitatively Dehazing Algorithms". In: *IEEE International Conference on Image Processing (ICIP)*. ICIP'16. Pheonix, USA, 2016.

[80] J.-P. Tarel et al. "Vision Enhancement in Homogeneous and Heterogeneous Fog". In: *IEEE Intelligent Transportation Systems Magazine* 4.2 (Summer 2012), pp. 6–20.

[81] L. K. Choi, J. You, and A. C. Bovik. *LIVE Image Defogging Database*. `http://live.ece.utexas.edu/research/fog/fade_defade.html`. Accessed: 2018-11-30.

[82] L. K. Choi, J. You, and A. C. Bovik. "Referenceless Prediction of Perceptual Fog Density and Perceptual Image Defogging". In: *IEEE Trans-*

actions on Image Processing 24.11 (Nov. 2015), pp. 3888–3901. ISSN: 1057-7149.

[83] L. K. Choi, J. You, and A. C. Bovik. "Referenceless perceptual image defogging". In: *2014 Southwest Symposium on Image Analysis and Interpretation*. Apr. 2014, pp. 165–168.

[84] L. K. Choi, J. You, and A. C. Bovik. "Referenceless Prediction of Perceptual Fog Density and Perceptual Image Defogging". In: *IEEE Transactions on Image Processing* 24.11 (Nov. 2015), pp. 3888–3901. ISSN: 1057-7149.

[85] Anush Krishna Moorthy and Alan Conrad Bovik. "Blind image quality assessment: From natural scene statistics to perceptual quality". In: *IEEE transactions on Image Processing* 20.12 (2011), pp. 3350–3364.

[86] Nicolas Hautière et al. "Blind contrast enhancement assessment by gradient ratioing at visible edges". In: *Image Analysis & Stereology* 27.2 (2011), pp. 87–95.

[87] G. Meng et al. "Efficient Image Dehazing with Boundary Constraint and Contextual Regularization". In: *2013 IEEE International Conference on Computer Vision*. Dec. 2013, pp. 617–624.

[88] Chen Chen, Minh N Do, and Jue Wang. "Robust image and video dehazing with visual artifact suppression via gradient residual minimization". In: *European Conference on Computer Vision*. Springer. 2016, pp. 576–591.

[89] Qingsong Zhu, Jiaming Mai, and Ling Shao. "A Fast Single Image Haze Removal Algorithm Using Color Attenuation Prior". In: *IEEE Transactions on Image Processing* 24 (2015), pp. 3522–3533.

[90] Dana Berman, Tali Treibitz, and Shai Avidan. "Non-local Image Dehazing". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 1674–1682.

[91] Ari Seff et al. "Continual Learning in Generative Adversarial Nets". In: *CoRR* abs/1705.08395 (2017).

[92] G. Satat, M. Tancik, and R. Raskar. "Towards photography through realistic fog". In: *2018 IEEE International Conference on Computational Photography (ICCP)*. May 2018, pp. 1–10.

[93] Ting-Chun Wang et al. "Video-to-Video Synthesis". In: *Advances in Neural Information Processing Systems (NIPS)*. 2018.