

---

# TESI DI DOTTORATO

---

NICOLA FELICE CAPECE

## Image processing in 2D/3D Computer Graphics with Deep Supervised Learning

*Dottorato in Matematica ed Informatica*, Salento (2019).

[http://www.bdim.eu/item?id=tesi\\_2019\\_CapeceNicolaFelice\\_1](http://www.bdim.eu/item?id=tesi_2019_CapeceNicolaFelice_1)

L'utilizzo e la stampa di questo documento digitale è consentito liberamente per motivi di ricerca e studio. Non è consentito l'utilizzo dello stesso per motivi commerciali. Tutte le copie di questo documento devono riportare questo avvertimento.

---

*bdim* (Biblioteca Digitale Italiana di Matematica)  
SIMAI & UMI

<http://www.bdim.eu/>

# Image processing in 2D/3D Computer Graphics with Deep Supervised Learning



**Nicola Felice Capece**

University of Salento and University of Basilicata

This dissertation is submitted for the degree of  
*Doctor of Philosophy in Mathematics and Computer Science*

University of Salento and  
University of Basilicata

April 2019



Advisor:  
Dr Ugo Erra



## **Abstract**

Supervised learning is a learning paradigm derived from different Machine Learning algorithms. The idea of this approach is based on the mechanisms facilitating the functioning of the human brain. Human beings learn and recognize objects and their details from birth, through observation and association to specific categories to which they belong. In a similar way, supervised learning seeks to train intelligent systems using ideal examples, which consist of desired input and output coupled, through which the system learns to recognize and classify the objects that belong to the same domain. Today such learning paradigms are extensively used in Deep Learning contexts. This thesis presents a multi-layered study of the use of supervised learning, focusing particularly on Multilayer Perceptrons and Convolutional Neural Network, but due attention is also given to other approaches. The goal is to demonstrate how the use of this paradigm and deep learning can tackle and simplify different computer graphics problems, from the rendering field to computational photography. All this is represented from a unique common denominator, the Graphics Processing Unit (GPU), which is the basis of the success and the evaluations of deep learning and all its aspects.

Several studies and experiments performed at the University of Basilicata and the National Research Council of Pisa are presented. The obtained results suggest how this technology can play a leading role in the future progress of computer graphics and more generally, computer science.



# Table of contents

<b>List of figures</b>	<b>xi</b>
<b>List of tables</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivations . . . . .	6
<b>2 Background</b>	<b>9</b>
2.1 Deep Learning . . . . .	9
2.1.1 Artificial Neural Network . . . . .	10
2.1.2 Feed-Forward Neural Network . . . . .	10
2.1.3 Convolutional Neural Network (CNN) . . . . .	12
2.1.4 U-Net . . . . .	17
2.2 Learning Paradigms . . . . .	19
2.2.1 Supervised Learning Paradigm . . . . .	19
2.2.2 Unsupervised Learning Paradigm . . . . .	20
2.2.3 Reinforcement Learning Paradigm . . . . .	21
2.3 GPU Authority . . . . .	22
<b>3 Coin Recognition System</b>	<b>25</b>
3.1 Overview . . . . .	25
3.1.1 Implementation Details . . . . .	26
3.1.1.1 Dataset and Classification Model definition	29

---

3.1.1.2	Creation of the Classification Model . . .	30
3.1.1.3	Inferences . . . . .	31
3.2	Client Server REST Architecture . . . . .	31
3.3	Training Phase . . . . .	34
3.3.1	Results . . . . .	35
3.4	Discussion . . . . .	37
3.4.1	Details on AlexNet and Training Configuration . . .	38
<b>4</b>	<b>Ambient Occlusion Baking</b>	<b>41</b>
4.1	Overview . . . . .	41
4.2	Feed Forward Neural Network Approach . . . . .	45
4.2.1	Representation . . . . .	45
4.2.2	Training . . . . .	47
4.2.3	Rendering . . . . .	48
4.3	Experimental Results . . . . .	49
4.4	Discussion . . . . .	52
4.4.1	The Z-Buffer problem . . . . .	54
<b>5</b>	<b>Night time to Day time Approach</b>	<b>57</b>
5.1	Overview . . . . .	57
5.2	Fully Convolutional Neural Network . . . . .	58
5.3	Experimental Results . . . . .	64
5.4	Discussion . . . . .	70
5.4.1	Details on VGG Convolutional Network . . . . .	70
5.4.2	Details on Residual Learning Network . . . . .	71
<b>6</b>	<b>Deep Flash face photos</b>	<b>75</b>
6.1	Overview . . . . .	75
6.2	Turning a Flash Selfie into a Studio Portrait . . . . .	77
6.2.1	Deep Neural Network . . . . .	77
6.2.2	Training . . . . .	79

---

6.2.3	Problem Encoding . . . . .	81
6.2.4	Loss Function . . . . .	83
6.3	Experimental Setup - Dataset Creation . . . . .	85
6.4	Results . . . . .	86
6.4.1	Comparison with reconstructed Ground Truth . . . . .	87
6.4.2	Comparison with other approaches . . . . .	90
6.4.2.1	Comparison with HDRNet . . . . .	92
6.4.2.2	Comparison with Pix2Pix . . . . .	94
6.4.3	Comparisons with Style Transfer . . . . .	98
6.5	Discussion . . . . .	99
6.5.1	Limitations . . . . .	101
<b>7</b>	<b>Deep Chroma Key</b>	<b>103</b>
7.1	Overview . . . . .	103
7.2	U-Shape Convolutional Neural Network Architecture . . . . .	106
7.2.1	Decoder . . . . .	107
7.3	Deep Neural Network for the Chroma Key . . . . .	108
7.3.1	Dataset Collection . . . . .	111
7.4	Training Step . . . . .	114
7.5	Results . . . . .	115
7.5.1	Evaluation methods . . . . .	115
7.5.2	Unfiltered Dataset Results . . . . .	116
7.5.3	Filtered Dataset Indoor Results . . . . .	121
7.5.4	Filtered Dataset Outdoor Results . . . . .	125
7.5.5	Improvements . . . . .	127
7.6	Comparisons and Applications . . . . .	129
7.7	Discussion . . . . .	134
7.7.1	Advantages . . . . .	135
7.7.2	Limitations and Directions . . . . .	136

---

<b>8</b>	<b>Overall Discussion</b>	<b>137</b>
8.1	Generalization and Training Problems . . . . .	137
8.1.1	Rules of Dataset definition . . . . .	139
8.1.2	Overfitting and Underfitting . . . . .	141
8.1.3	Vanishing gradient problem . . . . .	143
8.2	Future Research Directions . . . . .	145
8.2.1	Generative Adversarial Networks . . . . .	145
<b>9</b>	<b>Related Work</b>	<b>147</b>
9.1	Deep Learning . . . . .	147
9.2	Coin Classification . . . . .	148
9.3	Real-Time Ambient Occlusion . . . . .	149
9.4	Global Illumination and Neural Networks . . . . .	150
9.5	Day Time to Night Time Translation . . . . .	150
9.6	Computational Photography and Deep Learning trend . . . . .	152
9.6.1	Flash Photography . . . . .	152
9.6.2	Deep Learning and Computational Photography . . . . .	152
9.7	Deep Chroma Key . . . . .	153
9.7.1	Chroma Key Effect . . . . .	153
9.7.2	Deep Learning and Image Semantic Segmentation . . . . .	154
<b>10</b>	<b>Conclusion</b>	<b>157</b>
	<b>Appendix A Comprehensibility of the Deep Learning Terminology</b>	<b>161</b>
	<b>References</b>	<b>171</b>
	<b>Index</b>	<b>193</b>

# List of figures

2.1	A basic example of feed-forward neural network and the comparison between the biological and artificial neuron. . . . .	11
2.2	A classifier based on Convolutional Neural Network. . . . .	12
2.3	Convolutional Neural Network concepts. . . . .	13
2.4	Typical non-linear activation functions used in convolutional neural network. . . . .	15
2.5	Logistic Sigmoid and Hyperbolic Sigmoid activation function	16
2.6	The max and average pooling operations. . . . .	17
2.7	An example of U-Net convolutional neural network. . . . .	18
2.8	Supervised Learning Paradigm structure and work flow. . . . .	19
2.9	Unsupervised Learning Paradigm work flow. . . . .	20
2.10	Reinforcement Learning schema of its behind mechanism. . . . .	21
2.11	Diagram and the graphic representation of a generic neural network through the Tensorflow framework. . . . .	22
3.1	The architecture of AlexNet, a convolutional neural network proposed by Alex Krizhevsky <i>et al.</i> [86]. . . . .	27
3.2	A coin recognition system proposed architecture. . . . .	32
3.3	The use of the mobile app. A very simple interface that enable the user to classify coins in real-time. . . . .	33
3.4	Loss training, accuracy validation and loss validation for the best obtained classification model. . . . .	36

3.5	The partial euro coins used to test the neural network. . . . .	37
4.1	Occlusion maps computed through Screen Space Ambient Occlusion and Offline Ambient Occlusion algorithms . . . . .	42
4.2	Raytraced Ambient Occlusion: a most used method to solve the Ambient Occlusion integral. . . . .	43
4.3	Real-Time Ambient Occlusion: a very used method is Screen Space Ambient Occlusion. . . . .	43
4.4	The acyclic feed-forward neural network, which defines a mapping from 16 sample normals in object space to the output Ambient Occlusion. . . . .	46
4.5	An example of normals and Ambient Occlusion values sampling operation. . . . .	47
4.6	Rendering phase by using a trained feed-forward neural network	49
4.7	A visual comparison of Happy Buddha among the different hidden neurons configuration . . . . .	50
4.8	A visual comparison of Stanford Bunny among the different hidden neurons configuration . . . . .	50
4.9	Visual comparison of Dragon 3D model among the different neural network configurations. . . . .	50
4.10	Stanford Lucy 3D model and the comparison among the three different neural network configurations . . . . .	51
4.11	Close-up of ambient occlusion results. . . . .	51
4.12	The effect of the introduction of the depth-buffer in the dataset and in the training of the neural network . . . . .	54
5.1	The architecture of our own Fully Convolutional Neural Network. . . . .	59
5.2	Details on the proposed residual learning net . . . . .	61
5.3	The plotted graph of truncated normal distribution with a classic bell curve shape. . . . .	64

---

5.4	Example of the night-to-day conversion on the real environment.	65
5.5	Comparison between 1,000 and 6.6 million of iterations. . .	66
5.6	Another example of inferences results obtained after 1,000 and 6.6 million of iterations. . . . .	66
5.7	Results of the Unreal Engine outdoor scene. . . . .	68
5.8	Other results obtained by using Unreal Engine outdoor scene.	68
5.9	Results obtained by using the Unreal Engine indoor scene. .	69
5.10	Other results obtained by using Unreal Engine on the same indoor scene. . . . .	69
5.11	The figure shows the residual block. . . . .	72
6.1	Two examples from our flash no flash approach. . . . .	76
6.2	Our neural network architecture for transforming a flash im- age into a non-flash image. . . . .	78
6.3	Each row of the table shows a possible (and tested) encoding of the problem. . . . .	81
6.4	Accuracy trend in the validation set at the end of training the convolutional neural network. . . . .	87
6.5	Samples of validation data: original flash, bilateral filtered flash, reconstructed and ground truth reconstructed images. .	90
6.6	Training set example: original input, image reconstructed and ground truth reconstructed are shown. . . . .	91
6.7	Test set example: original input, image reconstructed and ground truth reconstructed are shown. . . . .	92
6.8	Two example of a real images. . . . .	93
6.9	A comparison between HDRNet end-to-end training and our approach. . . . .	95
6.10	An example of comparisons between HDRNet (combined with our problem encoding) and our approach. . . . .	95
6.11	A comparison between Pix2pix end-to-end training and our approach. . . . .	97

---

6.12	An example of comparisons between Pix2Pix (with our problem encoding) and our approach. . . . .	98
6.13	An example of visual comparisons between the Portrait Style transfer and our approach. . . . .	100
7.1	Some examples of functioning and results obtained with our approach. . . . .	104
7.2	The used U-shape Net, developed from the SegNet [6] architecture. . . . .	107
7.3	A label overlay of a training image. The background pixels (light blue), are more frequent with respect to the foreground pixels (red). The frequency of foreground pixels in the training dataset is 11.16%, and the frequency of background pixels is 88.84%. . . . .	110
7.4	Example of pre-processing phase. . . . .	111
7.5	Comparison between an unfiltered image and the corresponding image filtered with the bilateral filter. . . . .	113
7.6	Results related to the test dataset after training using the unfiltered dataset. . . . .	119
7.7	Results of real images, taken directly with a camera, querying the network after training with the unfiltered dataset. . . . .	120
7.8	Results of the filtered test dataset for the chosen indoor area. . . . .	123
7.9	Results of real images, taken directly with a camera, querying the network after training with the filtered dataset related to the chosen indoor area. . . . .	124
7.10	Results of the filtered test dataset for the chosen outdoor area. . . . .	126
7.11	Results of real images, taken directly with a camera, querying the network after training with the filtered dataset for the chosen outdoor area. . . . .	127
7.12	Improvements in the neural network result. . . . .	128

---

7.13	The images show the results obtained with Adobe Photoshop Select Subject on a subset of our filtered test dataset for the indoor area. . . . .	131
7.14	The images show our results on the same test set by using Adobe Photoshop Select Subject 7.13 . . . . .	132
7.15	Results obtained with Select Subject on real images taken directly with a camera. . . . .	133
7.16	The trimap generated through our network output. . . . .	134
8.1	The graph represents: Underfitting, Good Model and Overfitting	142
8.2	A visual interpretation of the vanishing gradient problem. . .	144
8.3	The classical architecture of the Generative Adversarial Network. . . . .	146
A.1	The gradient descent algorithm. . . . .	162
A.2	The application of bilinear interpolation. . . . .	168
A.3	The bilinear interpolation. . . . .	169



# List of tables

3.1	Training dataset and classification models with their related parameters of our approach. . . . .	34
3.2	Test performed on the trained classification models. The best model is shown. . . . .	35
4.1	Structural Similarity Index between three neural network output	52
4.2	Comparison between the three neural networks, Mara <i>et al.</i> [100], and vertex attributes rendering . . . . .	52
6.1	Loss validation and loss test after 62 epochs. This table also shows the maximum accuracy achieved in both phases. . . .	89
6.2	The Structural Similarity Index of the reference images. . . .	89
6.3	Comparisons on samples of the training, validation, and test sets (HDRNet). . . . .	96
6.4	Comparisons on samples of the training, validation, and test sets (Pix2Pix). . . . .	99
7.1	The metric values of the whole test dataset for the network trained with the unfiltered dataset. . . . .	117
7.2	The metrics obtained by considering each class with respect to the unfiltered test dataset. . . . .	117
7.3	Results of the test carried out on the network after training with the unfiltered dataset. . . . .	117

---

7.4	The metric values of the whole test dataset for the network trained with our filtered dataset for an indoor area. . . . .	121
7.5	The metrics obtained by considering each class with respect to the filtered test dataset for an indoor area. . . . .	121
7.6	The confusion matrix related to the test carried out on the network after training with the filtered dataset for an indoor area. . . . .	121
7.7	The metric values of the whole test dataset for the network trained with our filtered dataset for the chosen outdoor area. .	125
7.8	The metrics obtained by considering each class with respect to the filtered test dataset for the chosen outdoor area. . . . .	125
7.9	The confusion matrix related to the test carried out on the network after training with the filtered dataset for the chosen outdoor area. . . . .	125
7.10	The first row shows the metric values obtained using the results of Photoshop Select Subject on a subset of our filtered test dataset for the indoor area. The second row shows the results of our network using the same test subset. . . . .	129
7.11	The metrics for each class obtained using the results of Photoshop Select Subject and our results on a subset of our filtered test dataset for the indoor area . . . . .	130
7.12	The confusion matrix related to the test carried out using the results of Photoshop Select Subject and our results on a subset of our filtered test dataset for the indoor area. . . . .	130

# Chapter 1

## Introduction

Machine learning is a set of methods that explores the study and construction of algorithms that can learn from input data and perform predictions on them. Deep Learning represents a subset of such methods containing learning models inspired by the structure and functioning of the human brain. As in the human brain, deep learning algorithms can process data at non-linear levels allowing computers to learn and perfect increasingly complex functionality. The goal of deep learning is to develop computational models, which consist of multiple processing layers used to learn data representation at multiple abstraction layers based on a hierarchy of concepts [56].

In recent years, deep learning has been used in more applicative contexts, *e.g.*, obstacle detection in the automotive field [125], [38]; in the medical field to identify cancer cells [49],[30],[32]; industrial automation to avoid accidents at work [49],[30],[32]; electronic field, auditory and vocal translations [37]. However, the use of deep learning has been growing more and more in all aspects of the Computer Graphics field, such as computational photography, real-time rendering, semantic segmentation and so on, where deep learning is taking on an increasingly crucial role [6], [53], [42], [45], [112].

Typically an Artificial Intelligence system needs to manually extract features from data. These features are then used in the next steps to create a

predictive model (*e.g.*, image objects classification). For many tasks, it is not easy to determine the best features to extract, especially in images analysis, because they are affected by highlights such as lighting variations, shadows, reflects, *etc.* In deep learning instead, the features are extracted automatically and an end-to-end learning is performed, through which the Artificial Intelligence learns how to process data and perform tasks automatically. However this mechanism suffers from a complex and difficult to eradicate problem *i.e.*, the need for significant amounts of training data.

Deep Learning is often approached to the Big Data concept because the main limitation of the deep learning algorithms is the requirement of a huge amount of training data which allows a correct convergence. In general, the Big Data term refers to a collection of so extensive data in terms of volume, speed and variety, requiring technology and specific analytic methods to extract values and knowledge [36]. Big Data life cycle is composed of principal processes that can be grouped into two main categories: (i) Big Data Management, which includes processes and technologies for acquisition and storage of Big Data and the preparation and recovery of themselves; (ii) Big Data Analytics, which includes the used processes for information analysis and acquisition useful from huge dataset to interpret and describe the past through a descriptive analytics process, predict the future through predictive analytics process or recommend actions through prescriptive analytics process [47]. Deep Learning plays an important role in the Big Data analytics solutions, in which deep and refined architectures are required to process large amount of data in real-time, with high accuracy and effectiveness. Traditional techniques have different limitations in a wide amount of data processing. In the last years, through the social media notoriety such as Facebook, Twitter and YouTube, a huge amount of data are collected, through the billions of users' contribution [72]. In the Big Data Analytics context, it is necessary a good data representation in order to allow high performance of traditional machine learning algorithms, even if they are simple algorithms. On the other hand, a

---

poor data representation can cause performance decreasing also with complex learning algorithms. The traditional machine learning algorithms are based on fundamental feature engineering concept [114], which represents the usage process of the knowledge domain in order to create features which allow the work of machine learning algorithms. This concept is very important but represents a very hard and expensive process and can be avoided through the feature learning concept [10]. Such concept describes a set of techniques which allow the system to discover automatically data representations and correlations. Deep learning fits perfectly in this context because it is able to learn and extract automatically data representation using supervised or non-supervised learning techniques. For this reason, it is used to deal with Big Data problems in an efficient way, such as the encoding and indexing of themselves, information retrieval, *etc.* Thanks to computational power and data size, deep learning finds considerable benefits in this area, indeed through the recent GPU-based framework, the training time for complex deep learning models is noticeably reduced, from several weeks to less than a day.

L.Y. Prat in 1993 [126] conceived of the concept of Transfer Learning, which permits the exploitation of the "knowledge" obtained from Artificial Intelligence in specific tasks and its reuse in similar but different tasks (*e.g.*, Artificial Intelligence knowledge that performs face classification can be applied to another Artificial Intelligence for face segmentation). More and more approaches based on deep learning use the transfer learning concept [150] [42] [117] [131] [83]. The advantage of this method is its capacity to train Artificial Intelligence systems using limited amounts of data, by exploiting the obtained knowledge as a training base [119].

All discussions of deep learning also indirectly refer to a neural network, called in more appropriate terms deep neural network. A deep neural network is a neural network that consists of several depth layers. A particular type of deep neural network often used in image recognition tasks is called convolutional neural network, whose spatial architecture is structured to take

advantages from the use of multidimensional data (tensors) such as images. Generally, neural networks consist of different layers, each of them composed of numerous artificial neurons. Each neuron is connected to the others through weighted arches called synapses, defining a fully connected neural network. Convolutional Neural Network's layers are composed of data volumes called tensors. In this type of neural network, each neuron is not connected to all those of the subsequent layer as in fully connected neural networks, but groups of them, called receptive fields are connected to the neurons of the next layer. In this case, neurons of one layer are connected to the neuron of the next layer through local connectivity. In general, neural networks analyse the input and recognize and classify it through automatic learning algorithms.

In this thesis, we investigate the use of deep learning through supervised learning paradigms in the 2D and 3D computer graphics field, addressing classification and regression problems. In particular, a first task concerns the deep learning application for coin recognition [21]. In this task, we show how a convolutional neural network can recognize and classify coins starting from image analysis. During the training phase, we determinate the optimum dimension of the training dataset needed to achieve high classification accuracy and low variance. Such a system is contextualized within the mobile world, where we propose a client-server architecture RESTful based that allows the users to identify the coins by taking pictures of them using smartphone cameras. The image provided by the user is processed through the neural network on a remote server and a prediction is performed based on the classification model obtained in the training phase. This application is a test that can be useful for numismatic experts in the complex coin classification tasks and it is useful to quantify the amount of coins needed to achieve optimal results.

A second supervised learning task concerns real-time rendering. In particular, we propose an Ambient Occlusion baking method via a feed-forward neural network [45]. The idea is based on the implementation of a multi-layer perceptron that allows a general encoding via regression and an efficient

---

decoding through a simple GPU fragment shader. The non-linear nature of multi-layer perceptrons makes them adaptable and efficient in capturing non-linearities described through ambient occlusion values. Moreover, a multi-layer perceptron is also random-accessible, it has a compact shape and can be evaluated efficiently on GPU. Our approach is shown on Screen-Space ambient occlusion based on a neural network taking into consideration its quality, dimension and speed.

Based on previous works, we investigate the ability of deep learning concerning variation in the lighting scheme on images. In particular, we show how a convolutional neural network enables the simulation of artificial and ambient light on images, addressing a case study that concerns the conversion of night-time images to day-time images [22]. In this work, we illustrate the architecture of the convolutional neural network and some preliminary results of a real indoor environment and two virtual environments (indoor and outdoor) rendered with a 3D engine. The experimental results confirmed that a convolutional neural network is an interesting approach in the image processing field.

Since the results were encouraging we deepened the study of the lighting scheme on images. In this case, the focus is shifted to a huge problem in the photographic field *i.e.*, conversion of a flash photo to no flash photo. In particular, the aim was to turn a flash selfie into a studio portrait [20]. The proposed method uses a convolutional encoder-decoder neural network (see U-Shape Network in Section 2.1.4) trained on a dataset composed of pairs of photos taken in an ad-hoc acquisition campaign. Each pair consists of one photo of a subject's face taken with a smartphone camera flash and another one of the same subject in the same pose illuminated using a studio-lighting setup. Our aim is to demonstrate that the proposed method can amend defects introduced by a close-up camera flash, such as specular highlights, shadows, skin, shine, and flattened images.

In the previous work, we encountered the problem of the extraction of the piece of the image that contains the shape of the actor in order to provide an input for the convolutional neural network which is as precise as possible. The idea is to identify the edges of the actor in images through the convolutional neural network, in order to perform the Semantic Segmentation and the Chroma Key effect. Although Chroma Key is a technique mainly used in Cinema and TV, it has limitations, because it is not achievable easily and quickly. In particular, it is necessary to maintain a separate illumination of the uniform background and of the subject and avoid as far as possible some shadows ending up in the frame. We propose a deep learning method to overcome these limits.

## 1.1 Motivations

One of the goals of this thesis was to demonstrate the use of deep learning in contexts where this technology had not been used yet. Thinking about the context of offline global illumination, in applications such as Ray Tracing [76], Photon Mapping [73], Ray-Traced Ambient Occlusion [88], [104] and others techniques [135] which require geometric information of the rendered scene and therefore a large amount of calculations and long computational times. Moreover, these algorithms also need a high availability of memory and computational resources to achieve high levels of photorealism. These limits increase the difficulty of applying such algorithms in the real-time field where fast calculations are necessary to obtain optimal results both in terms of performance and computational resources. In general, a compromise between quality and performance is often sought. In this context, deep learning can play a decisive role generating the quality and the photorealism of offline algorithms with real-time performance [45], [112]. In the last years there was a great increase in deep learning applications, especially in the context of image-to-image translation. The use of deep learning has

lead to obtain algorithms that allow to change the lighting scheme of images, improving their graphic quality. Thinking about removing flash effects from images, there are many applications developed in pre-deep learning era [123], [43] [4] and, although these applications have achieved excellent results, the rise of deep learning has clearly exceeded their quality and their limits [20], [149], [143], [68], [27][66], [176]. Remaining in the context of image-to-image translation, it is possible to focus on the translation from black and white images to colour images [28] or from night time images to day time images [22] or to change the lighting of an image based on sunlight at certain times of the day [95]. In this context the deep learning allowed to obtain results that were unimaginable in the pre-deep learning era. In addition to the purposes described above, our main motivations focus on the current limits of deep learning, among which the development of optimal datasets adapted to specific contexts (see Section 3). In fact, many of the works proposed in this thesis have the aim to find the correct dimensions of the dataset (see Section 3), to obtain optimal data quality and to use typology and data pre-processing methods contextualized to the application field (see Sections 6 and 7). Another motivation concerns the difficulty to use the deep neural network, in particular regarding the parameters and hyperparameters configuration of the deep neural networks and the optimization algorithms (see Appendix A) *etc.*

**Thesis Structure.** Chapter 2 provides an overview of the deep learning core concepts and the supervised learning paradigm. Starting from the deep learning definition in the Section 2.1 the focus will move to the different types of Neural Networks 2.1.1, the different type of Learning Paradigms 2.2 and the role of GPU through the more used deep learning frameworks.

Chapter 3 presents the coin classification problem, which helped us to study how to structure a good dataset and the basics of convolutional neural networks. Chapter 4 presents the application of deep learning for real-time

computer graphics. In this case, the focus is on neural network performance in terms of computational cost, memory space and the validity of deep learning for the approximation of global illumination and image or scene colours.

Chapters 5, 6 and 7 show deep learning application to image processing. In particular Chapters 5 and 6 show two approaches to solving image-to-image translation. The first one is based on generic indoor and outdoor images while the second one is used for specific images of faces.

An overall discussion on the unsolved or partially solved research questions is carried out in Chapter 8. In particular Section 8.1 presents the generalization and training problems, Section 8.1.1 shows some rules for a good dataset definition and the Sections 8.1.2 and 8.1.3 show the principal problems that afflict deep neural networks. The last Section of Chapter 8 shows future deep learning directions 8.2, and also offers a brief overview of the generative adversarial network. (see Section 8.2.1).

Chapter 9 shows the state of the art related to our presented applications and finally, Chapter 10 shows the conclusion and the research questions we tried to answer throughout this thesis.

# Chapter 2

## Background

In this chapter, we show some background information about the core concepts of deep learning and supervised learning paradigm. An overview is also shown on the other training paradigms and on the role of the GPU in the context of deep learning.

### 2.1 Deep Learning

Deep Learning refers to a set of technique which learns multiple levels of representation with a high level of features that represent multiple abstraction aspects of the data. A key concept of deep learning is definitely the one concerning the feature learning algorithms, who deal with identify the common patterns, to use them as the discriminant between the classes, both for classification and regression problems. In deep learning, the convolutional layers are excellent to extract automatically the features of images to the next layer in order to form the hierarchy of non-linear features that grow in complexity as it goes deep. To describe such process is possible to define two steps: the first step consists in the non-linear features extraction by using multiple levels, while the second step consists in passing these levels for example to a classifier, which combines all features to perform a prediction.

### 2.1.1 Artificial Neural Network

A basically neural network consists of connected nodes called artificial neurons that are modeled on biological neurons. These simple nodes are often called processing elements or units. Each neuron produces a sequence of real-value activation [142]. Within the neural network, artificial neurons are organized into layers. Artificial neurons in each layer are connected to upper and lower layers through weighted arcs called synapses. A standard neural network consists of three types of layer: input, hidden, and output. A layer is a container that receives a weighted sum over the inputs, transforms it with a set of non-linear functions, and passes these values as input to the next layer [11]. The first and last layers are called input and output layer respectively, while all layers in between are called hidden layers. Input neurons are activated by data provided through an external system. In contrast, output and hidden neurons are activated by data provided from already activated neurons [142]. Each neuron is activated by an activation function, which receives weighted data (matrix multiplication between input data and weights) and outputs a non-linear transformation of it. Through the repetition of these steps, the artificial neural network can learn multiple layers of non-linear features and combine them to the last layer in order to perform a prediction. The learning of the neural network is due to a loss signal, which is obtained from the difference between the prediction and the corresponding real value, often called **ground truth**. The loss is used to update the weights in order to get a more accurate prediction, for this reason, it is an adaptive system.

### 2.1.2 Feed-Forward Neural Network

A Feed-Forward Neural Network is an artificial neural network where the connections between the neurons of each layer do not create cycles. In this type of network, the information crosses through a unique direction, from the input layer forward to the output layer. It is able to resolve supervised

learning problem, in fact after the training step a mapping is established, that allows associating a set of possible input values to a set of output values, by creating a mathematical relationship that is stored inside the weights of the network. Such neural network is able to work in a continuous set of output values, by performing **regression** tasks, or in discrete set of output values, by performing **classification** tasks. As can be seen in the Figure 2.1, each branch of the network is composed of weighted arcs, that allow generating a priority relationship for the data computation inside the network. Typically the weights are initialized in a random way and their values are modified during the training step through the backpropagation algorithm [91].

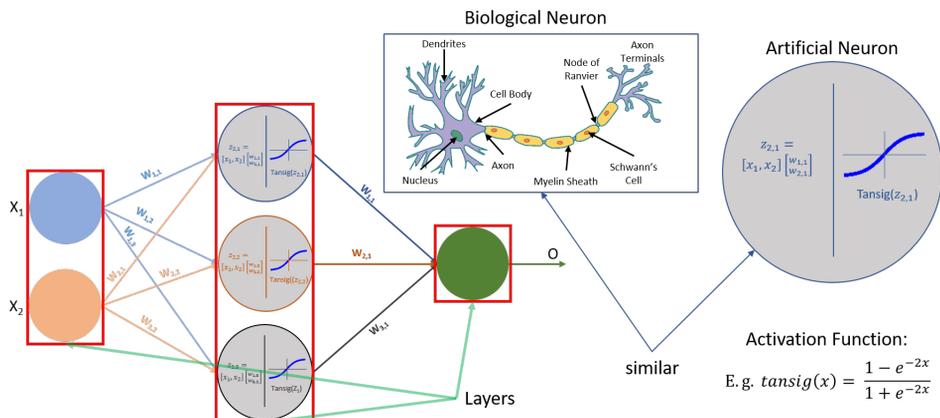


Fig. 2.1 A basic example of feed-forward neural network and the comparison between the biological neuron and artificial neuron. In the figure is represented a neural network with three layers, input, hidden and an output layer. Inside the artificial neuron is represented a *tansig* non-linear activation function.

Feed-forward neural network is also called Multilayer Perceptron which is defined as the quintessential example of a deep learning model by I. Goodfellow *et al.* [56]. Through the supervised learning paradigm it is able to learn a mathematical function  $f: \mathbb{R}^n \rightarrow \mathbb{R}^o$ , using a dataset which has  $n$  as input dimension and  $o$  as output dimension. This algorithm has to be provided

of a set of features  $X = x_1, x_2, x_3, \dots, x_n$  and of a ground truth  $t$ , and learns a non-linear function that binds both[54].

### 2.1.3 Convolutional Neural Network (CNN)

Feed Forward Neural Networks can be used also with a single hidden layer, but when they are composed of several hidden layers the discussion focuses on deep learning. However feed-forward neural networks do not take into account the spatial structure of the input, especially in the image processing context, where the spatial structure is crucial. Convolutional Neural Network is specifically designed to address tasks that involve the images although it can also be used in other contexts.

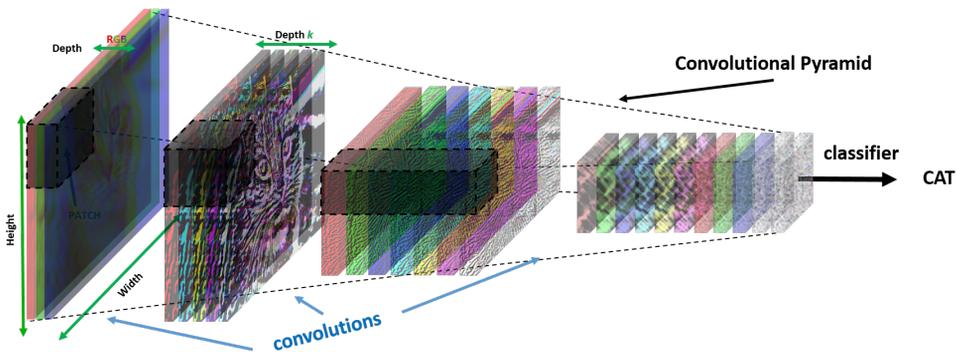


Fig. 2.2 A classifier based on Convolutional Neural Network. In this figure is represented an example of the classifier of animals, each group of blocks represent the feature maps and between one group and another, a convolutional operation is performed.

Unlike the feed-forward neural network, the neurons of convolutional neural network are organized in the width, height, and depth as dimensions. In this case, depth means the dimension of the activation volume (activation tensor) and not the depth of the network. Each layer of convolutional neural network transforms an input tensor in an activation tensor formed by neurons. In image processing task the input tensor is the image to be processed and the

depth is represented through the R G B channels while the width and height are the spatial dimensions of the Figure 2.2. Each neuron that belongs to the hidden layers is only connected to a small, localized region of the previous layer<sup>1</sup>, called local receptive field 2.3.

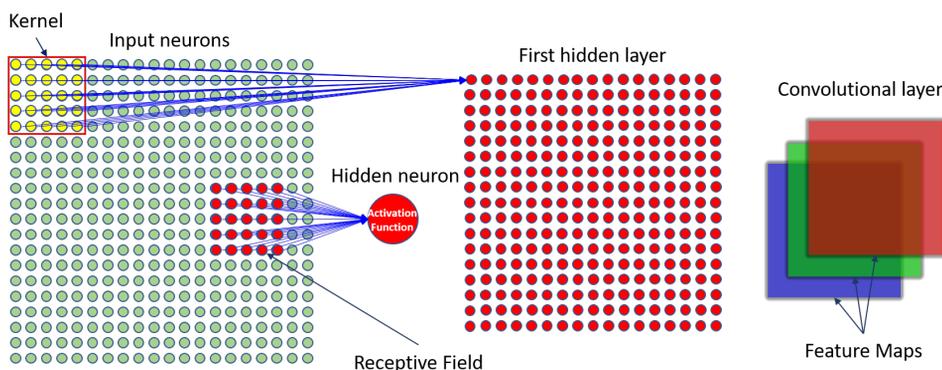


Fig. 2.3 Convolutional Neural Network concepts: green matrix represent the input tensor (*e.g.*, image), the red square is the kernel size and the yellow sub-tensor is a first local receptive field connected to the first hidden neuron. When the convolution was performed then the kernel moving toward by left to right until it reaches the last one. In each hidden neuron was performed the non-linear activation function (*e.g.*, Rectified Linear Unit). The right part of the figure shows the basic feature of an input image that is the R G B channels.

The local receptive fields are local in the width and height but complete on the entire depth of input tensor. As can be seen in the figure 2.3, the local receptive field is a small window of the previous layer that is moved by one or more neurons based on a parameter called stride length and the resulting local receptive field is connected to next hidden neuron of the current hidden layer. Such movement continues until the current hidden layer is built. Each connection learns a weight, while each hidden neuron learns an overall bias. To be more precise, for all hidden neurons, are used

<sup>1</sup> The previously layer can be the input layer, and in this case, the localized regions contain the pixels.

the same weights and biases, in order to allow the neurons to detect the same features, but in different positions of the previous layer. This property makes the convolutional neural network invariant to translation, allowing to recognize the same image even if it is translated. Convolutional neural network architecture can vary based on a type and number of layers, due to the application context. Typically [90] a convolutional neural network consists of a first part where the convolution<sup>2</sup> and pooling are interchanged. These operations gradually decrease the spatial dimension (width and height) and increase the depth of activation tensor. Finally, at the end of this pyramid, often some fully connected layers are stacked. Convolutional neural network is characterized by a predominant operation called convolution. The aim of this operation is to extract the principal features from the image provided as input. In particular, element-wise matrix multiplication between the input image and a kernel<sup>3</sup> and the sum of the multiplication output are performed. The kernel moves by as many pixels as indicated by the stride and the output of convolution is called **feature map** or activation map. If the kernel has smaller dimension than the input matrix, a sub-matrix which has the same dimension of the kernel is considered step-by-step. If *e.g.*, we consider an input with dimension  $N \times N$  and a kernel  $\mathbf{K}$  with dimension  $m \times m$ , the output dimension of convolution will be  $(N - m + 1) \times (N - m + 1)$  and the operation can be summarized as follows:

$$y_{i,j}^l = \sigma(x_{i,j}^l), \quad x_{i,j}^l = b_j + \sum_{s=1}^{m-1} \sum_{t=1}^{m-1} K_{s,t} y_{(i+t)(j+s)}^{l-1} \quad (2.1)$$

Where  $x_{i,j}^l$  is the input of the current layer, and  $y_{(i+t)(j+s)}^{l-1}$  is the output of the previous layer, due to the sum of its all elements, in particular,  $b_j$  is the shared bias and  $K_{s,t}$  is the shared weights that belong at the local receptive field and  $\sigma$  is the non-linear activation function.

<sup>2</sup> Convolution is often followed by activation function such as Rectified Linear Unit.

<sup>3</sup> Kernel is often called convolutional filter.

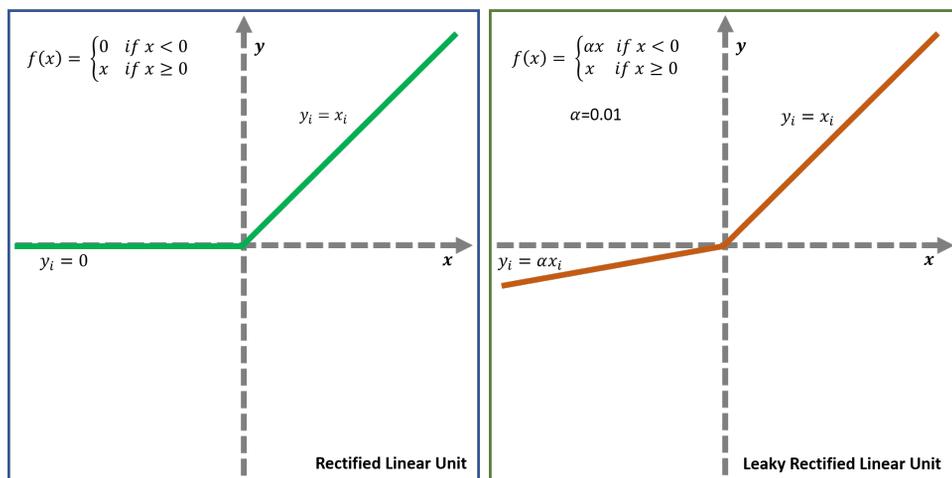


Fig. 2.4 Typical non-linear activation functions used in convolutional neural network. The left image shows a standard Rectified Linear Unit and the right image shows a Leaky Rectified Linear Unit.

A convolution is a linear operation, and since deep learning creates the increasingly complex function for each layer, it needs to use a non-linear activation function in order to create a new relationship between the elements. A most used non-linear activation function in the convolutional neural network is the rectified linear unit and its variants, which is applied to each non-activated element of the feature map and replace the negative values with zero values. In our applications we used the standard rectified linear unit and a variant called Leaky Rectified Linear Unit [55], which are shown in the Figure 2.4.

Although there are several type of non-linear activation functions(see Figure 2.5), the rectified linear unit and its variants are the best choice in more contexts, in particular for the image processing tasks, because they proved to be the most efficient [111].

Another fundamental basic operation used in the convolutional neural network is the Pooling function (see Figure 2.6). This operation generally follows the convolutional layer and performs a spatial decrease of each feature

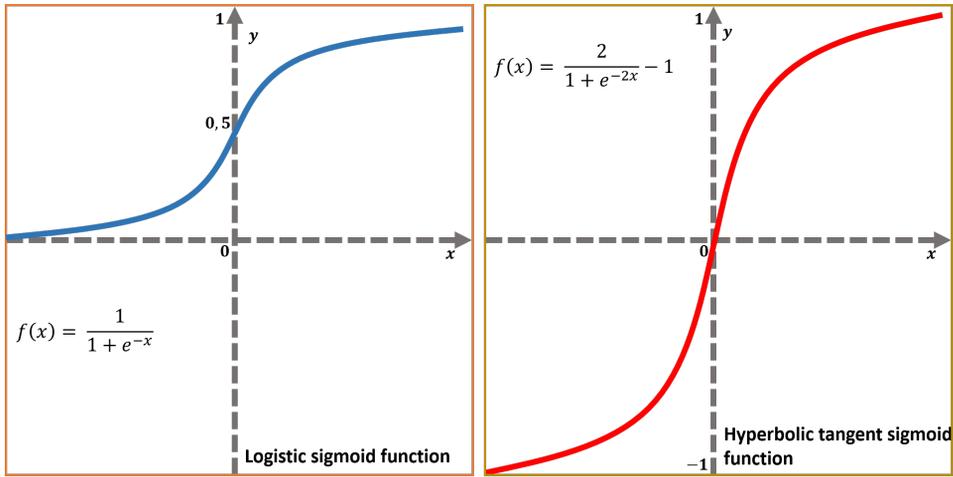


Fig. 2.5 The left image shows the logistic sigmoid activation function and the right image shows the hyperbolic tangent sigmoid activation function. Such type of activation functions are more used in the binary classification problems on the last layer and they can avoid exploding gradient problem.

map by keeping only the important information without decrease the depth of activation tensor. There are several types of pooling: max, average, sum, [175] *etc.* The most used is the max-pooling, which consists in the definition of a preset size moving window on the input matrix, in order to keep the maximum activation of each step of moving window. Among the advantages of max-pooling there are following: (i) the dimension of the feature maps are smaller and easier to manage; (ii) it decreases the amount of the parameters and the computations of the network by controlling the overfitting problem; (iii) the network is invariant to small transformations, distortions, translations of the input images; (iv) it help to get an invariant scaled representation of the images.

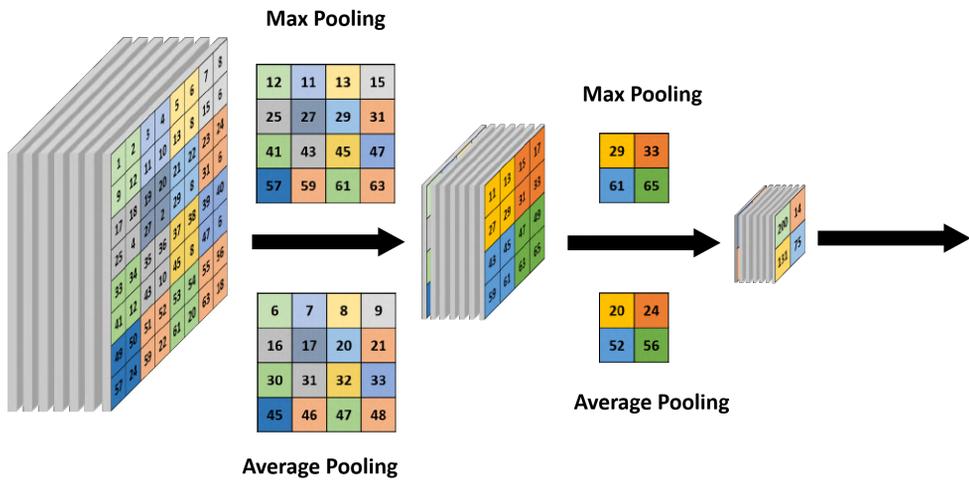


Fig. 2.6 As can be seen, the image shows two examples of Max-Pooling and Average-Pooling. The Max-Pooling is performed by sampling the maximum value for each  $2 \times 2$  window and the Average-Pooling performs the average for each window of same dimension.

### 2.1.4 U-Net

A particular type of fully convolutional neural network is called U-Net (Figure 2.7)<sup>4</sup>, which architecture was proposed from O. Ronneberger *et al.* [136] for bio-medical image segmentation. Convolutional neural network can be used in classification and regression tasks: in the first case, given an input image, the corresponding output is a label that represents the belonging class; in the second case, especially in the image processing tasks, the output results are localized and are applied on each pixel. In particular, the authors of the U-Net have extended the J. Long *et al.* [97] approach, by replacing pooling operations with up-sampling operations, in order to increase the resolution of the output. In this way, the convolutional neural network takes the U-shape based on the well known encoder-decoder architecture. As can be seen in the figure 2.7, the network is composed of two sub-networks, encoder and

<sup>4</sup> U is due to the U-shape of the network architecture.

decoder: the first sub-network consists of convolutional layers followed by max-pooling operations, in order to perform the encoding of the image into feature representations; the second sub-network, consists of up-sampling, which can include the unpooling and deconvolution [6],[115] operations followed by convolution operations.

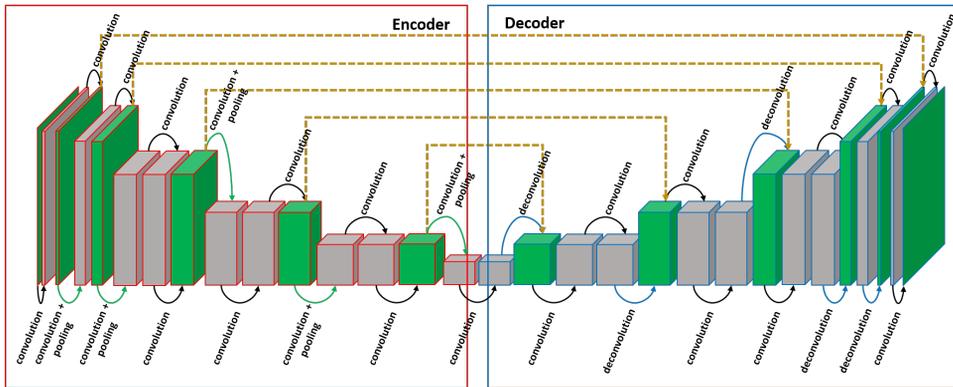


Fig. 2.7 An example of U-Net convolutional neural network. The figure shows the typical encoder-decoder structure: the left blocks framed in red, represent the encoder sub-network; the right blocks framed in blue, represent the decoder sub-network. The image shows also the operations performed inside each sub-network. The yellow arrows represent the shortcut connections, used to recover information from the encoder sub-network.

The main idea based on up-sampling operations can be explained as an expansion of the feature size in order to adapt them to the dimensions of the encoding blocks. In this way is also possible to recover the information lost during the encoding phase by concatenating the feature map obtained from each convolutional block in the encoding phase with the corresponding decoder block. The up-sampling operations issue was studied enough [170][172], in particular, another interesting method was proposed from V. Badrinarayanan *et al.* [6] which reuse the pooling indices in order to reduce the computational costs without losing accuracy and obtaining the high resolution output. In this thesis, for an easy understanding, we

refer to U-shape Net for all used convolutional neural networks based on encoder-decoder architecture due to their U shape.

## 2.2 Learning Paradigms

In this section will be shown an overview of the main learning paradigms which characterize the deep learning. There are three main categories of learning paradigms: supervised learning, unsupervised learning and reinforcement learning. Such paradigms are based on the type of experience obtained in the training phase using an appropriate dataset which contains many examples.

### 2.2.1 Supervised Learning Paradigm

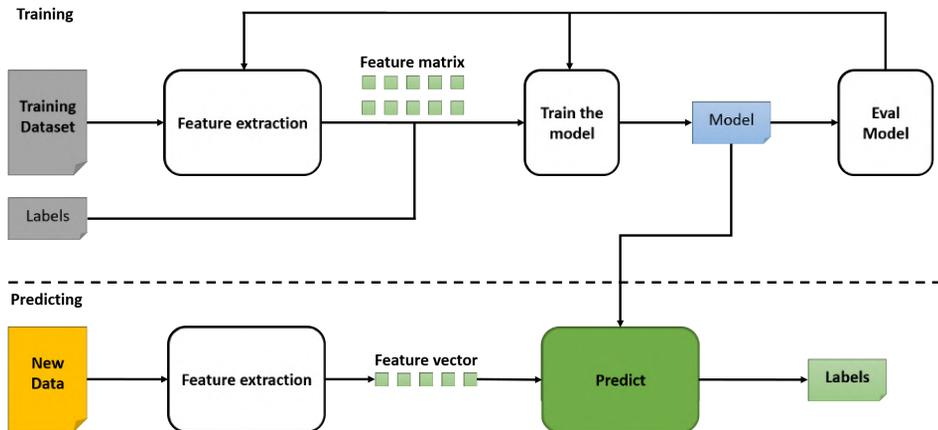


Fig. 2.8 The figure is divided into two parts: first part shows the training phase, in particular the training dataset and labels are provided to the algorithm that performs feature extraction, training and evaluation of the model; the second part shows the inference phase, where feature extraction is performed based on new data and the result is obtained from the prediction.

As humans learn to distinguish objects from the real world through the association of images, the supervised learning paradigm (Figure 2.8) allows

neural networks to learn a function that is able to predict the correct output value for each valid input provided to them, based on input-target pairs of examples that represent a dataset **experience**. In more specific terms, in the training phase, the neural network is provided with a series of inputs which the outputs called **targets** or **labels** are known. The aim of supervised learning is to develop a function also called **inductive hypothesis** that can replicate results obtained in the training phase using similar, but new, structured input [139].

### 2.2.2 Unsupervised Learning Paradigm

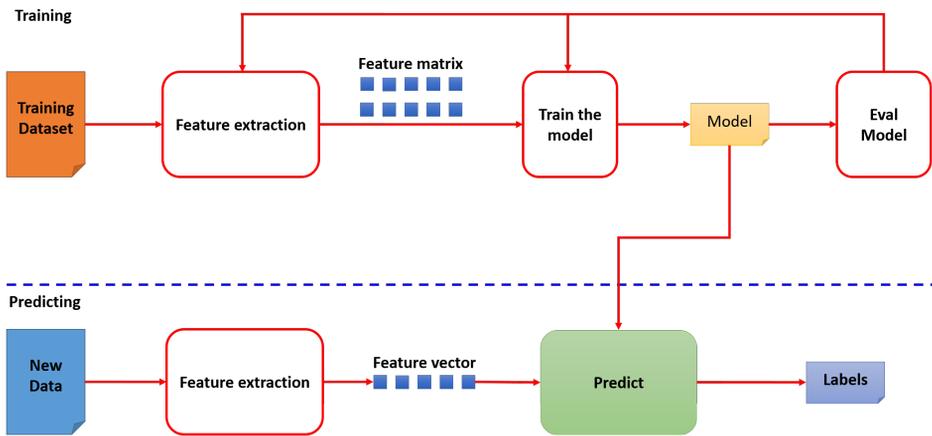


Fig. 2.9 An unsupervised learning algorithm flow. Training performed by using only training dataset without labels. The result is obtained in the inference phase through the model prediction.

Such paradigm consists of providing neural network an input whose target is not known a priori. In this case, the dataset consists of only input examples and the learning algorithm has the aim of finding hidden structures not labelled inside the input data. In more specific terms, the neural network take the inputs that are provided and reclassifies and organizes them according to a

set of shared features, in order to reason and make a prediction about any subsequent input [41].

### 2.2.3 Reinforcement Learning Paradigm

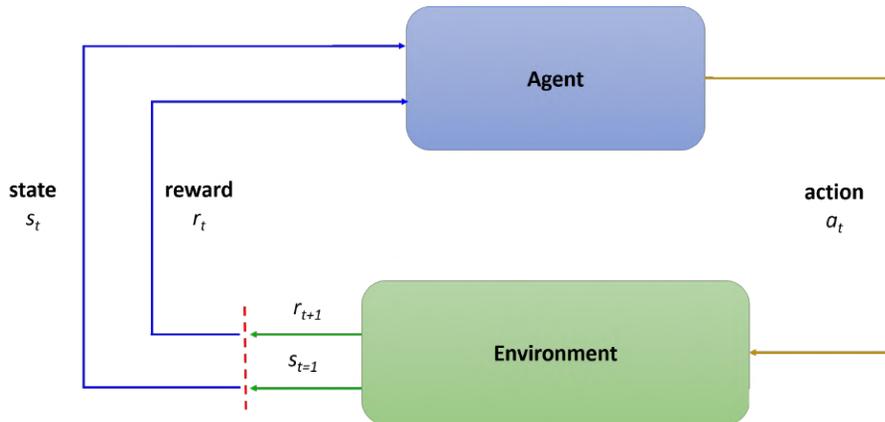


Fig. 2.10 The schema shows the mechanism behind the reinforcement learning paradigm. The agent performs an action  $a_t$  that influence the environment in which is immersed. The agent is able to learn the mutation in the environment due to the action through the reward  $r_t$ , which consists of its performance evaluations. The state  $s_t$  represents the current situation of the agent and the  $r_{t+1}$  and  $s_{t+1}$  represent the future reward whenever the agent performs an action in a particular state.

Reinforcement Learning is based on the premise to receive feedback from the external environment that influence the choices of the neural network. In more specific terms, the neural network aims to identify a certain modus operandi, starting with a process of observing the external environment. Each action has an impact on the ambient, and it produces a feedback that drives the algorithm itself in the learning process. Unsupervised learning algorithms attempt to determine a policy aimed at maximizing the cumulative incentives received during the training phase. These algorithms are different from supervised learning algorithms because no input-target pairs are used in the

training phase, but explicit interventions are performed during the training process [75] [155].

### 2.3 GPU Authority

Today, GPU computing leverages the architecture parallelism to perform mathematically compute-intensive operations that the CPU is not designed well to handle [46]. In particular, GPUs definitely can be used to speed up neural networks training. For this reason, several frameworks are developed for training and use of deep neural networks. Most of them exploit GPU acceleration and include Tensorflow [2], Caffe [74], Torch7 [31], Theano [12], and others<sup>5</sup>.

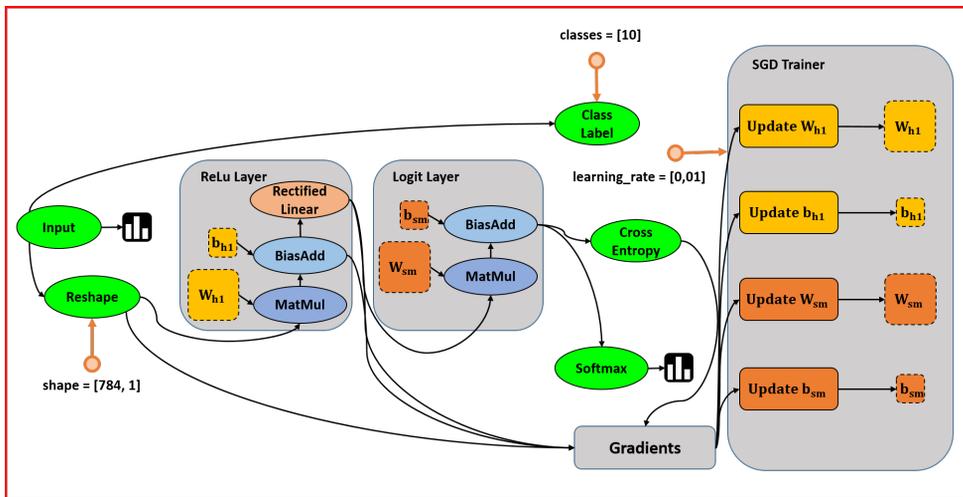


Fig. 2.11 Diagram and the graphic representation of a generic neural network through the Tensorflow framework. Rectangles represent the variables and arrows the tensors.

<sup>5</sup> Most popular Deep Learning Frameworks: <https://developer.nvidia.com/deep-learning-frameworks>

Tensorflow is a library written in C++ and CUDA<sup>6</sup> and provided APIs for Python, C++, Java and GO. For some of our applications, we used Python programming language. To use Tensorflow, there are two main phases: (i) A first phase, called construction phase, regarding the creation of the graph that defines the training model and the task that the neural network has to perform; (ii) A second phase, called run-time phase, in which the operations defined in the previous phase are performed. Figure 2.11 shows a simple graph of a classifier based on a feed-forward neural network. The ellipse-shaped icons represent the mathematical operations that are performed on the tensors. The rectangle-shaped icons represent the variables in the model and the arrows are the tensors and their values that cross the graph in calculations. Starting from the input tensor that can be seen from the first ellipsoid to the left of the Figure 2.11, which represents an image, there is a subsequent operation of reshaping that transforms the image into a vector. The larger grey rectangles represent the layers of the neural network in which the rectified linear unit activation functions is defined. The final part of the graph represents the computing of the loss function through the mean squared error, given from the prediction of the neural network and the expected data. Subsequently the optimization algorithm is performed, in the Figure 2.11 Stochastic Gradient Descent [14] is represented. The last layer of the graph is responsible for updating the weights using information provided by the gradient. This operation will be repeated until the error will not be sufficiently small, or it will reach a number of iterations defined a priori. Tensorflow is supported by front-end libraries with a higher level of abstraction, like Keras<sup>7</sup>, Tensorlayer<sup>8</sup> and others. In particular, we used Tensorlayer for the case study described in Chapter 6.

Another most used deep learning framework is Caffe (Convolutional Architecture for Fast Feature Embedding). Caffe introduces the blobs concept

---

<sup>6</sup> Compute Unified Device Architecture, an hardware architecture for parallel processing developed by NVIDIA: <https://developer.nvidia.com/cuda-zone>

<sup>7</sup> Keras: <https://keras.io/>

<sup>8</sup> Tensorlayer: <https://tensorlayer.readthedocs.io/en/stable/>

that is a 4-dimensional data array which provides holding batches of images and other data, unified memory interface, parameters or parameter update [74]. The main advantage of blobs is to hide the information exchange between CPU and GPU by synchronizing the host CPU and GPU when necessary. In this way, there is a considerable increase in the efficiency of memory management. Caffe support C++ and Python programming languages and use Google Protocol Buffer to store the models<sup>9</sup>. We use Caffe for the case study described in the Chapter 3, in particular as a module of the NVIDIA Deep Learning GPU Training System (see Section 3.1.1). Also, the MATLAB computing environment offers different solutions to build and train neural networks. In fact, we used the Deep Learning Toolbox™ provided by MATLAB to address the case studies proposed in Chapters 4 and 7. Such tool benefits of the parallel computing of single or clustered GPU and cloud.

Finally, other two important frameworks are Torch7 [31] and Theano [12]. The first one is a scientific framework, developed by using the Lua programming language. Torch7 supports two parallelization methods: OpenMP<sup>10</sup> and CUDA that allow to take advantage from the GPU parallel computing. Theano is an open source library based on Python programming language and also benefits of the GPU parallel computing by using CUDA and the NumPy syntax<sup>11</sup> to simplify the construction of mathematical expressions.

---

<sup>9</sup> Protobuf: <https://github.com/protocolbuffers/protobuf>

<sup>10</sup> OpenMP is a cross-platform API for creating parallel applications on shared memory systems: <https://www.openmp.org/>

<sup>11</sup> NumPy is an extension on Python programming language to support the vector, multi-dimensional matrix and mathematical expressions: <http://www.numpy.org/>

# Chapter 3

## Coin Recognition System

Coin Recognition is the first task we addressed [21] using a supervised learning paradigm and convolutional neural network. This chapter shows an overview of numismatic and image processing and as deep learning can be embedded within this relationship. The rest of the chapter shows our proposed approach to solve coin recognition task and an application that can support the numismatics experts in the coin classification operation. Finally, there is a discussion of the possible future works and the research trends.

### 3.1 Overview

Numismatics is the scientific study of all forms of currency. Various criteria are used to classify a coin, including its history, geography, and market value. In recent decades, image recognition techniques was investigated for the identification and classification of coins, as currently these procedures still rely on expensive human intervention. In [169], the authors survey numismatic research into the classification, identification, and segmentation of coins based on image recognition. Numismatics research can be divided into different historical periods: ancient, medieval, modern, and contemporary. Here, a contemporary numismatics was considered, which focuses on coins from the

17<sup>th</sup> century onward. In particular, we have addressing the classification of euro coins. The advantage of studying this category is that any engravings can still be clearly seen by the human eye, and there is a plentiful supply of coins, which is useful in machine learning approaches. Recent studies of coin classification are based on image recognition techniques. Several families of algorithms was developed, based on neural networks [51], decision trees [34], edge detection [116], gradient directions [132], and contour and texture features [158][168]. In this work an implementation of a system for automatic coin recognition based on deep learning was developed. In particular, the representation of coins is learned in a training phase, through a well-known convolutional neural network, proving that is a valid choice for coin identification. In particular, we determine the optimum size of the training dataset that is necessary to achieve high classification accuracy with low variance. Based on a set of 8320 images of euro coins, we trained the convolutional neural network using different-sized training samples and tested the resulting system. Using this data, we employ a learning curve approach to predict classification accuracy for a given training sample size. Furthermore, we propose a client-server architecture that makes it possible to query the classification model obtained from the neural network training set, and allows a user to identify a coin by photographing it with, for instance, a mobile device camera. A coin is identified when the image provided by the user can be matched with the neural network on a remote server.

### 3.1.1 Implementation Details

The coin recognition task is included in the wider category of the image recognition tasks. As mentioned in Chapter 1, in the context of deep learning, neural network are often called deep neural network, as they are distinguished from the more common neural networks by their depth. In deep neural network, each layer of neurons is trained on a distinct set of features based on the output of the previous layer. Deeper layers of the neural network can

recognize more complex features, as they reprocess features from the previous layer. This concept is known as feature hierarchy<sup>1</sup>. Image recognition requires a very deep neural network composed of multiple layers. It must be able to extract non-linear features and pass them to a classifier that can combine all of the features and make predictions. As mathematically shown by NVIDIA<sup>1</sup>, for image processing, the best features of a single layer are edges and blob. This is because they contain the most information that can be extracted from a single, non-linear transformation. It was shown that the human brain does the same thing. The first hierarchy of visual cortex neurons is sensitive to specific edges and blobs, while deeper regions of the brain that are further down the visual pipeline are sensitive to more complex structures, such as faces.

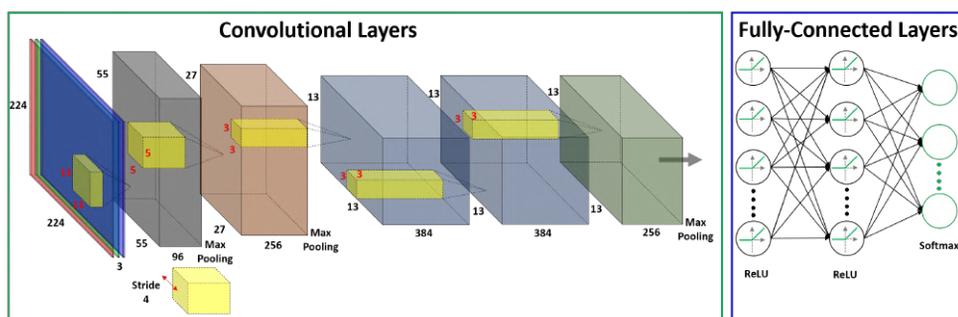


Fig. 3.1 AlexNet, a convolutional neural network proposed by Alex Krizhevsky *et al.* [86] that consists of eight layers where the first five are convolutional layers and the last three are fully-connected layers. To train the originally proposed convolutional neural network they were based on the use of two GPUs and outlined the responsibility of both, the communication of such GPUs occurred only at certain layers. The figure shows the original configuration of AlexNet, through the input dimensions, the kernels and strides size of each convolutional layer; where the pooling operations were performed and finally, the last two fully connected layers which consist of 4096 neurons each one and last output layer fed to a 1,000-way softmax.

<sup>1</sup> NVIDIA Deep Learning: <https://devblogs.nvidia.com/paralleforall/deep-learning-nutshell-core-concepts>

Convolutional Neural Network is often used in image recognition, also because a basic principle of such type of feed-forward neural network is that the connectivity pattern between neurons is inspired by the specific organization of the brain's visual cortex. The architecture of a convolutional neural network is designed to take advantage of the spatial bi-dimensional structure of an input image. Every image is a matrix of pixel values that describes intensity at that point. The range of values that can be encoded in each pixel is a function of its bit size. Convolutional Neural Networks are given an array of intensities as input and the output are numbers that describe the probability of the image belonging to a certain class. One benefit of the convolutional neural network is that they are easier to train and there are fewer parameters than fully-connected networks with the same number of hidden units. For coin recognition, we used the most popular convolutional neural network created by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, which is called AlexNet [86]. The input to the AlexNet is a resized image, while the output is a class-label probability. AlexNet includes object recognition steps such as local feature extraction, feature coding, and learning (see Figure 3.1). The advantage of such a convolutional neural network is that it can adaptively estimate optimal feature representations for datasets, which is a feature that is lacking in the conventional, hand-crafted approach. The effectiveness of the AlexNet was proven for large-scale object recognition at the ImageNet Large-Scale Visual Recognition Challenge 2012 (ILSVRC-2012). In addition, AlexNet has already been tested on image recognition in several contexts including, for instance, face detection [50], maritime vessel identification [33], food recognition [80], and playing video games [107].

As explained in Section 2.3, there are several framework for the training and use of deep neural networks. Here, we used the NVIDIA DIGITS version 4.0 <sup>2</sup>, the first interactive software for deep neural network training using

---

<sup>2</sup>Deep Learning GPU Training System: <https://developer.nvidia.com/digits>. Our used version included Caffe, Torch7, Theano, and CUDA-Convnet2 frameworks. Today the last version is 6.1.1, which include others frameworks such as Tensorflow.

the GPU. It makes possible to develop, train, and visualize the deep neural network. The interface is browser-based, and neural network behaviour can be viewed in real-time. For training, DIGITS uses the popular Caffé Deep Learning Framework and GPU capabilities to reduce training time. Using DIGITS requires two steps. The first step consists in the definition of the dataset and classification model and the second step concerns the creation of such classification model and the defining of the model parameters.

### 3.1.1.1 Dataset and Classification Model definition

Starting of first step, a set of folders are provided to the system: each contains a series of images of the same class. Then, a text file is defined that contains the labels belonging to each class. The overall dataset is divided into three subsets: training, validation, and testing. The training set is a collection of data for which both the input and the output are known. It is used in the neural network training phase and for the creation of the related classification model. The validation set is similar to the training set, but is used in the validation step, *i.e.*, a step useful to evaluate how much the neural network is generalized (see Section 8.1). In more specific terms, the validation set was designed to evaluate if the classification model is or not in overfitting through the comparison neural network performance on a different dataset from the training one. Training and validation set allow to obtain evaluation metrics such as prediction accuracy and loss training and loss validation. Accuracy is the reliability of the prediction based on validation data provided in the training phase. Loss training is the error on the training set computed on each batch size<sup>3</sup>. Loss validation is the error after running the validation set through the neural network. Generally as epochs<sup>4</sup> increase, both validation and training error fall. Training error can continue to fall even after many

---

<sup>3</sup> Batch Size: the number of examples used in each iteration (see Appendix A).

<sup>4</sup> The epoch is a forward and backward pass of all training data in the neural network (see Appendix A).

epochs, allowing the neural network to improve its learning. In this case, the validation error can increase, eventually leading to overfitting. Finally, the test set is a collection of data used to test and evaluate the performance of the classification model after the training phase.

### 3.1.1.2 Creation of the Classification Model

After the creation of dataset in the previous step, we need to define the training parameters such as the epochs<sup>4</sup>. In more specific terms, backpropagation learning algorithms (see Appendix A) involve two steps. The first step, the forward propagation, consists of passing the training data to the neural network in order to generate the activation output. The second step, the backward propagation, consists of the loss value computation obtained through the activation output of the neural network and the target data and its backpropagation in all hidden neurons. Loss value is provided using a loss function called also objective function, which represents the function to be minimized in order to obtain a good convergence of the neural network. The most used loss functions are the cross-entropy and mean squared error, the first one is often used in classification problems and the second one is often used in regression problems [137] (see Appendix A). In this work we used cross-entropy loss function.

DIGITS allows to set some useful parameters such as snapshot interval that represents the number of epochs of training between two snapshots (whenever a snapshot take place, the model is stored separately and can be used after training *e.g.*, to classify the image); validation interval, that represents the number of epochs of training, running through one pass of the validation data; batch size<sup>3</sup> and solver type, which represents the gradient descent optimization method and can be the stochastic gradient descent, the adaptive gradient, or Nesterov's accelerated gradient [90] [93] [113] [40]. DIGITS makes it possible to edit the pre-configured neural network; it is possible to change parameters, add layers, change the bias, etc. Once the dataset

was created, training can begin. In the training phase, DIGITS provides a visualization of the data used and the training state. It also provides an accuracy chart and loss value in real time.

### 3.1.1.3 Inferences

As DIGITS runs on a web server, the dataset and the network configuration can be easily shared with the client. Once the neural network was trained, its related model can be queried by providing an image in HTTP request form. DIGITS web service responds with predictions that take the form of couple labels (classes) and percentage accuracy. Messages can be exchanged between a client (*e.g.*, a web or mobile application) and DIGITS through the Representational State Transfer (REST) architecture.

## 3.2 Client Server REST Architecture

The architecture of our own system is structured as client-server. DIGITS is installed on the server side and provides services through the *Digits Rest Api*<sup>5</sup>. The interface makes it possible to query a classification or regression model that was trained on a neural network. Such API can be used to create dataset and models, or make predictions using a trained model. The interface is based on the REST architecture that consists of a set of components, connectors, and other data within a distributed system, where the focus is on the role of components rather than implementation details. REST uses the HTTP protocol [96] to transmit data, while the network allows components to exchange representations of resources [122] [19] [44]. Connection elements are connectors that handle communication between components. A component can be a client or a server, and it acts as a mediator, making it possible for the application to interact with a resource, given the identifier of the resource

---

<sup>5</sup> Digits Rest API: Digits Representational State Transfer Application Programming Interface.

itself, and the action to perform. The application must interpret the response, and be able to represent the information. DIGITS includes a server-side component that, once a request is accepted by a client, provides a response in JSON<sup>6</sup> format. In particular, it makes it possible to create a classification (or regression) model and a dataset, to classify an element or an inference using a previously-trained model, and to remove a classification or regression model.

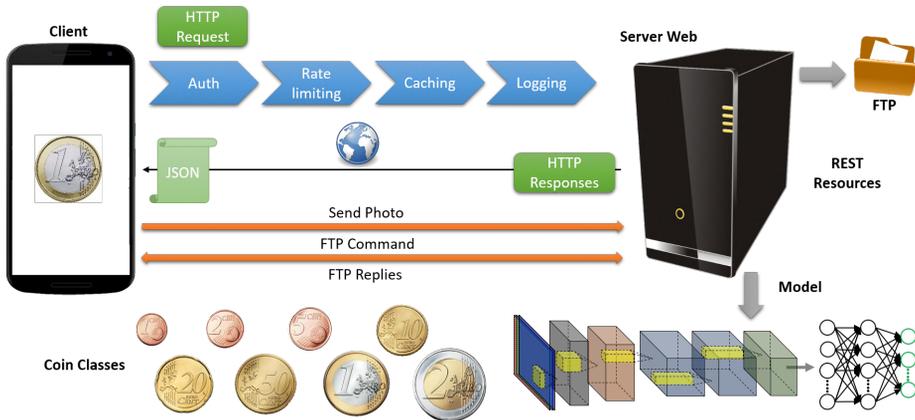


Fig. 3.2 The architecture of our system: the left side shows a client application and the photograph taken with the mobile camera; the right side shows a server application where the digits based web server is running and performs inferences on trained coin classification model; in the center is shown the HTTP request and FTP transaction and finally the bottom part in the figure shows the coin classes.

In our work, we focus on the classification of an image of a coin using a trained model. To exploit the remote classification model, we developed a client-side mobile application for Android devices (see Figure 3.2). This application uses the device camera to take a photograph of a coin to classify. The resulting image is sent by the client over FTP to the server, then an HTTP POST request that contains the query string is sent to the web service (DIGITS). This query contains the address of the server, the id of the trained

<sup>6</sup> JSON: JavaScript Object Notation.

classification model, and the location and name of the image to classify. Although it is possible to use a HTTP POST multipart request [101] to send the image directly to the server (together with the parameters specified above); FTP was chosen because the protocol is generally considered faster for sending individual files such as photographs. Afterwards, the web service response (in JSON format) is interpreted by the client.



Fig. 3.3 The mobile app in use: the interface is very simple and enables the user to take a photograph using the smartphone camera, or select an image from a local folder and send it to the server. Once the server-side model performs the recognition, the best five relevant results are highlighted.

The response consists of a list of predictions, each composed of a pair of labels (class), and a percentage accuracy (reliability) of the prediction. This response is displayed on the mobile device running the client application as shown in figure 3.3. In order to be as precise as possible, an example follows that will help to demonstrate the procedure: the user takes a photograph of the coin and the client software sends it to the server via FTP; it sends an HTTP request specifying the query string name and the location of the image, together with the id of the classification model that should be used. The server processes the image given in the query string, classifies it using the named model, and sends the client a list of predictions in JSON format. These predictions include the membership classes of the coin and prediction accuracy percentage sorted from the most to least accurate. The application presents the user with the results of the classification.

### 3.3 Training Phase

Input images consist of  $500 \times 500$  pixels, giving an input dimensionality of 250,000. There are eight classes that correspond to 0.01 €, 0.02 €, 0.05 €, 0.10 €, 0.20 €, 0.50 €, 1 €, and 2 € coins. It should be noted that here we only take into account the reverse side of the coin which contains the portraying map of Europe. For each of the eight classes, the training dataset consisted of 80 images taken with a camera at a distance of 10cm. To increase the number of coins images, we performed 13, 2D transformations in order to perform a very used operation in deep learning based applications called data augmentation. In particular, coins were rotated by  $60^\circ$  around their center, mirrored, then and rotated again by  $60^\circ$ . In general, as mentioned by I. Goodfellow *et al.* [56], classifiers can benefit from geometric operations such as random translation, flipping and rotation but also through other types of transformation such as the random colours perturbation and other types of image distortion [86][92]. The our own final dataset consisted of 1,040 images for each coin, and a total of 8,320 coin images.

DATASET	Euro1	Euro2	Euro3	Euro4	Euro5
MODEL	M1	M2	M3	M4	M5
TRAINING	60%	50%	55%	60%	59%
VALIDATION	35%	45%	37%	37%	38%
TEST	5%	5%	8%	3%	3%
EPOCH	10	10	10	10	10
ACCURACY	62.93%	42.65%	71.09%	75%	77.21%
LOSS TRAIN	0.63%	1.11%	0.51%	0.47%	0.30%
LOSS VAL	0.65%	1.10%	0.53%	0.45%	0.41%

Table 3.1 Training dataset and classification models with their related parameters. The first row shows the dataset denominations and the second row shows the related classification models. We performed numerous training tests with different configurations and we estimated that the M5 model configuration returns the lowest validation and test loss.

To obtain the best model, we performed several experiments on the input images using five different datasets. Each dataset had a different percentage of training, validation, and test data. The number of epochs was set to 10, and the batch size was set to 24. This approach is a useful way to establish the minimum number of input images needed to produce a useful model. Table 3.1 shows how the datasets and classification models were divided. The first and second row specify respectively the name of the dataset, while the other rows represent, in order, the percentage of training data, validation data, test data, the number of epochs, percentage accuracy, loss training, and loss validation.

### 3.3.1 Results

COIN	M1	M2	M3	M4	M5
0.01 €	0%	0%	100%	100%	100%
0.02 €	100%	100%	100%	100%	100%
0.05 €	100%	0%	100%	100%	90.3%
0.10 €	5.7%	21.2%	48.2%	32.3%	96.8%
0.20 €	50%	24.7%	89.2%	19.3%	77.4%
0.50 €	0%	53.8%	62.6%	22.6%	80.6%
1.00 €	100%	90.4%	90.4%	90.3%	100%
2.00 €	92.31%	90.4%	84.3%	100%	100%

Table 3.2 Test performed on the trained classification models. Model M5 performs best in the classification of test images according to the training results shown in table 3.1.

Results are shown in the table 3.2 through the tests that were performed on all of the classification models. These tests consisted of passing the test image folder to the respective classification model. Each test image folder is composed of a percentage of images defined in the dataset creation step as can be seen in Table 3.1. As Table 3.2 shows, each test is associated with a

class of coin that is defined in the first column. If the percentage for each coin is greater than 50% the test is passed, otherwise it is failed. Table 3.2 shows that the model M5 provided the most accurate classification. Therefore, for our input image dataset, this is the optimal recognition model.

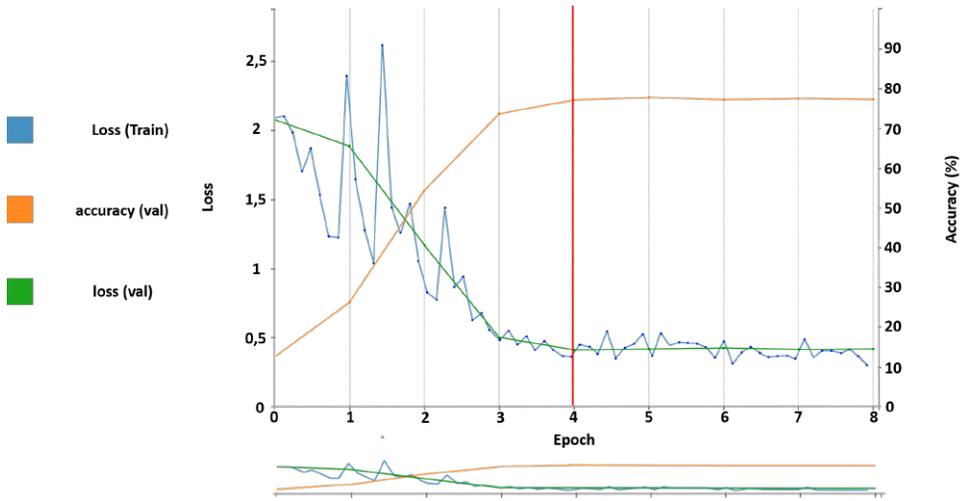


Fig. 3.4 Loss training, accuracy validation and loss validation for model M5. Figure shows the trend of such functions for each metrics, highlighting the convergence of the network and the stability of accuracy after the fourth epoch.

In more detail, the accuracy of model M5 is 72.21%. Figure 3.4 shows trends for each epoch in the training phase. The blue line indicates loss value trends, the orange line indicates accuracy trends, while the green line presents the loss value trend for the validation set. It should be noted that from the fourth epoch onward, accuracy percentage is stable at around 80%, and loss value remains low. Training and testing were performed on a NVIDIA 6GB TITAN GPU, which significantly improves response and training time of the deep learning classifier.

We also evaluated the response of the optimal model M5 on partial images of 1.0 €, 0.05 €, and 0.20 € coins as shown in Figure 3.5. This test was not exhaustive, but did give an indication of performance in the wide range

of production conditions associated with coin recognition. The percentage accuracy of the prediction was 77.25%, 100%, and 49.7% for 1.0 €, 0.05 €, and 0.20 € coins respectively. It should be noted that the accuracy of the test on 0.20 € coins was 49.7%, which is highest percentage compared to other accuracies obtained from the same image.

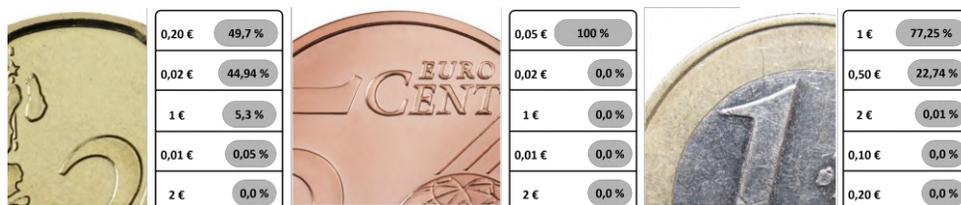


Fig. 3.5 The partial euro coins used to test the neural network. The results shown that the network is able to recognize with more accuracy pieces of coin images as can be seen on the percentage shown in the figure.

### 3.4 Discussion

However, deep neural network cannot used in contexts in which there are few available data, especially in supervised learning based applications where the collection of sufficient data is tedious and difficult, and in some cases impossible to obtain. For example, if we consider a network that must recognize cancer from an X-ray, this requires a training dataset labeled by an expert and that can not be obtained immediately. Generally deep neural network needs a lot of data to achieve comparable or better performance than conventional, local feature-based methods. In our preliminary coin recognition experiments, we trained a deep neural network on an already prepared dataset of images, which confirmed that the amount of training data was enough. The performance of the best model shown that 70-80 coin images for each class are needed in the training phase. More specifically, coins had medium to light overall wear, and all details were visible. The software

architecture that we implemented proved to have substantial advantages: these include an immediate response to the client, and the ability to use GPUs for training, validation and testing of the neural network in order to classify models quickly. We developed an application for Android-based mobile devices that provide a visualization of the results of the prediction based on an image of a coin, obtained using the device camera. In the future, we will use the Android-based application and a serious game to create a large training dataset [29]. Our discussion supports that through the embedded devices such as the NVIDIA Jetson Embedded Systems<sup>7</sup> this approach can be used in other contexts. These could include, for instance, a currency detector for retail kiosks, self-checkout machines, or gaming machines to detect counterfeit coins. The basic principle of the application is to test the physical properties of the coin. With respect to future work, we plan to use the approach for recognizing ancient coins. The challenge in this field mainly concerns the reconstruction of worn coins. In this case, the neural network can be applied to the partial recognition of images, by isolating the best-preserved parts of an ancient image. As deep learning techniques continue to progress, we are confident that this technology could be applied to ancient numismatics [141] with important results.

### 3.4.1 Details on AlexNet and Training Configuration

AlexNet is a deep neural network proposed by Alex Krizhevsky *et al.* [86], to classify 1.2 million of images for the ImageNet Large-Scale Visual Recognition Challenge 2010 (and 2012 with a variant of the model) contest to perform optimal results on 1,000 different classes. The architecture of this neural network can be seen in the Figure 3.1 and provide a well-optimized implementation on GPU of the convolution and other operations useful to train a deep neural network. The original neural network was trained on down-

---

<sup>7</sup>NVIDIA Jetson: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/>

sampled ImageNet images aligned with a  $256 \times 256$  constant resolution. An interesting feature is the training on multiple GPUs through the proposal of a parallelization scheme in which the half of the kernels are processed of each GPU, allowing the GPUs to interact each other only in certain layers. In particular, the second, fourth and fifth convolutional layers have their kernels connected to the previous kernel maps, which share the same GPU, while the third convolutional layer has its kernels connected to all the kernel maps of the second convolutional layer. This feature is due to the ability of GPU to interact with each other directly without passing in the host machine's memory and in this way the time of the training is dramatically decreased. Finally, the fully connected layers are composed of neurons connected to all neurons of the previous layer.

AlexNet was trained using stochastic gradient descent with 0.9 as momentum, 128 as batch size dimension, 0.01 as learning rate (see Appendix A) and 0.0005 as weight decay as regularization method (see Chapter 8.1.1, Equation 8.6). The weights are initialized by using Gaussian distribution (see Equation 5.2) with zero mean and standard deviation of 0.01. The bias of the second, fourth, fifth convolutional and all fully connected layers are initialized using the constant value 1. For our training experiments we use the same configuration proposed by the AlexNet authors, however as explained in Section 3.3 we used 10 as batch size value that increase the number of backpropagation steps and, in our case, it gives us optimal results.



# Chapter 4

## Ambient Occlusion Baking

A different task, which we addressed, concerns the deep learning and the real-time computer graphics using the supervised learning paradigm. In particular we perform the ambient occlusion baking via feed-forward neural network. This chapter shows an overview of the state of the art of the shading and the use of deep learning in such context and our proposed method. Our proposal is based on implementing a multilayer-perceptron that allows a general encoding via regression and an efficient decoding via a simple GPU fragment shader [45]. We illustrate our approach of screen-space ambient occlusion based on neural network including its quality, size, and run-time speed.

### 4.1 Overview

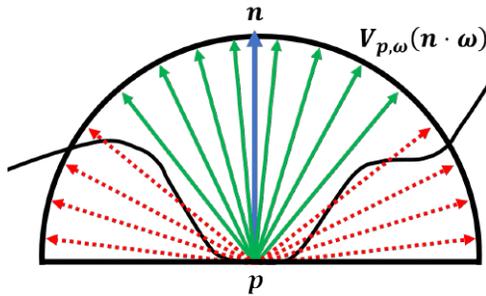
Shadowing of ambient light is called ambient occlusion. It was shown in [87] that ambient occlusion offers a better perception of the 3D shape of displayed objects, and its effectiveness is evident in its popularity in video-game engines [106]. Without the aid of shadows, such objects would appear flat, and in several cases suspended in the balance or both. The shadow understanding is directly connected to the number of shadows created by the light on a



Fig. 4.1 The left image shows the occlusion map computed in real-time which characterize the rendered portion of the scene. A most famous real-time ambient occlusion was proposed in Crysis [106] video game, called Screen Space Ambient Occlusion. The right image shows the occlusion map computed offline.

surface. In Computer Graphics, the shadow effect is computed and rendered as any contribution inside the scene. Computing of shadow is divided into two main categories: **large scale shadows** and **small scale shadows**. The first case concerns a hidden area from the light source, and the second one concerns *e.g.*, shadows due to folds in a cloth, between the wrinkles of the skin or between the cracks in a wall. Ambient Occlusion can increase the perception of a virtual world, by increasing the photo-realism degree of a virtual scene.

The mathematical definition of ambient occlusion is related to the concept of the solid angle. In fact, the occlusion  $A_{\mathbf{p}}$  at a point  $\mathbf{p}$  on a surface with normal  $\mathbf{n}$  can be computed by integrating the visibility function over the hemisphere  $\Omega$  with respect to the projected solid angle, as shown in equation (4.1). In the same equation  $V_{\mathbf{p},\omega}$  is the visibility function at  $\mathbf{p}$  along a direction  $\omega$ . There are several method to compute the Ambient Occlusion, which differ from each other for speed and accuracy: **Offline Ambient Occlusion** and **Real-time Ambient Occlusion**. A simple method to approximate the Ambient Occlusion integral (4.1) in practice, in off-line rendering (see figure 4.1b), is based on ray-tracing.



$$A_{\mathbf{p}} = \frac{1}{\pi} \int_{\Omega} V_{\mathbf{p},\omega}(\mathbf{n} \cdot \omega) d\omega \quad (4.1)$$

Fig. 4.2 Raytraced Ambient Occlusion: a most used method to solve the Ambient Occlusion integral. It is considered the simplest and most accurate possible, although due to its structure is also the slowest.

Rays are shot in a uniform pattern across the hemisphere over point  $\mathbf{p}$ , and an occlusion value can be calculated as the number of rays that hit the geometry divided by the total number of rays shot as shown in figure 4.2. Rays can be restricted to a certain length, avoiding the need to take into account distant geometry while calculating the occlusion value. This is fundamental in closed environments, which would otherwise result in total occlusion at every point and subsequently the complete removal of ambient light.

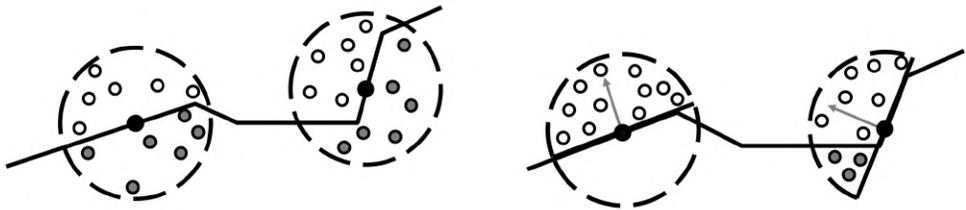


Fig. 4.3 In real time, a very used method is Screen Space Ambient Occlusion. This algorithm darkens a pixel treating adjacent pixels as potential occluders. Grey elements represent the samples with depth value is greater than an examined fragment. In the figure there are two examples with spheres and hemispheres.

A most used method to approximate the Ambient Occlusion in real-time rendering is called Screen Space Ambient Occlusion (see figure 4.1a). The

Screen Space Ambient Occlusion algorithms are used on the bi-dimensional space by darkening a pixel treating each of its adjacent pixels as potential occluders. This technique was introduced by M. Mittring [106] in 2007 implemented in the most popular video game *Crysis* of the Crytek. Respect to the offline raytraced Ambient Occlusion, Screen Space Ambient Occlusion is much more performing because it does not need of geometric information, but only the depth value of each pixel. In particular, Screen Space Ambient Occlusion will compute the depth value for each pixel rendered on the screen, by comparing such value with that of the neighbouring pixels, which are chosen at random way in a surrounding sphere. The number of neighbouring pixels, which depth value is greater than of the depth value of the current pixel provides its occlusion factor.

Video games often pre-compute Ambient Occlusion and bake out the results into vertex or texture data, which is later loaded into OpenGL or DirectX shaders. To implement such a baking pipeline, an implementation of ray-tracing is used as off-line rendering. When the ambient occlusion values was computed for a 3D shape, they must be written in a format that can be easily consumed during the rendering. The methods used in film and game rendering are point clouds, 2D textures, and vertex attributes. Point clouds are not efficient to access in hardware for real-time rendering. 2D textures retain the detail from the original ambient occlusion values, but require too many memory resources. Vertex attributes require occlusion values from the 3D surface shape to be mapped onto the vertices. The last solution yields significantly lower memory overhead and less expensive run-time reconstruction for real-time applications such as video games [78].

Our key idea is to represent ambient occlusion by using a multi-layer perceptron, allowing general encoding via regression and efficient decoding via a simple GPU fragment shader in real-time. Because of the non-linear nature of multi-layer perceptrons, they are suitable and effective for capturing non-linearities described by ambient occlusion values. In addition, multi-

layer perceptrons are random-accessible, have a compact size, and can be evaluated efficiently on the GPU. Our approach offers benefits in terms of quality, size, and run-time speed particularly for low-poly models used in real-time application. To evaluate the quality of results, we compare screen shots from our implementation with images rendered off-line in the ray-tracing engine NVIDIA OptiX[120], which is also used to train our neural network. For comparison, we use the Structural Similarity Index [163], which is a metric that measures similarity between images in a way that is consistent with human eye perception.

## 4.2 Feed Forward Neural Network Approach

As mentioned in Section 2, a feed-forward neural network is a weighted and directed graph whose nodes are organized into layers. The weights of the edges constitute the components of the weight vector  $w$ . The network we used is an acyclic feed-forward neural network with two hidden layers, as shown in Figure 4.4. Each node is connected to all nodes of the previous layer by directed edges, such that a node in the  $i^{th}$  layer receives inputs from all nodes in the  $(i - 1)^{th}$  layer. The graph takes inputs through the nodes in the first layer, and produces outputs through the nodes in the final layer.

### 4.2.1 Representation

Our proposed feed-forward neural network is composed of the 16 normals taken in the object space which represent the nodes of the input layer. The output layer consists of one node that is the value of the Ambient Occlusion. We use a  $16 \times X \times Y \times 1$  neural network, where  $X \times Y$  are the two hidden layers. In particular, we evaluated three hidden layers configurations:  $16 \times 8$ ,  $32 \times 16$ , and  $64 \times 32$ . Each neuron in a particular layer is connected with all neurons of the previous layer. The connection between a neuron  $k$  in the  $(i - 1)^{th}$  layer and a neuron  $j$  in the  $i^{th}$  layer is characterized by the weight

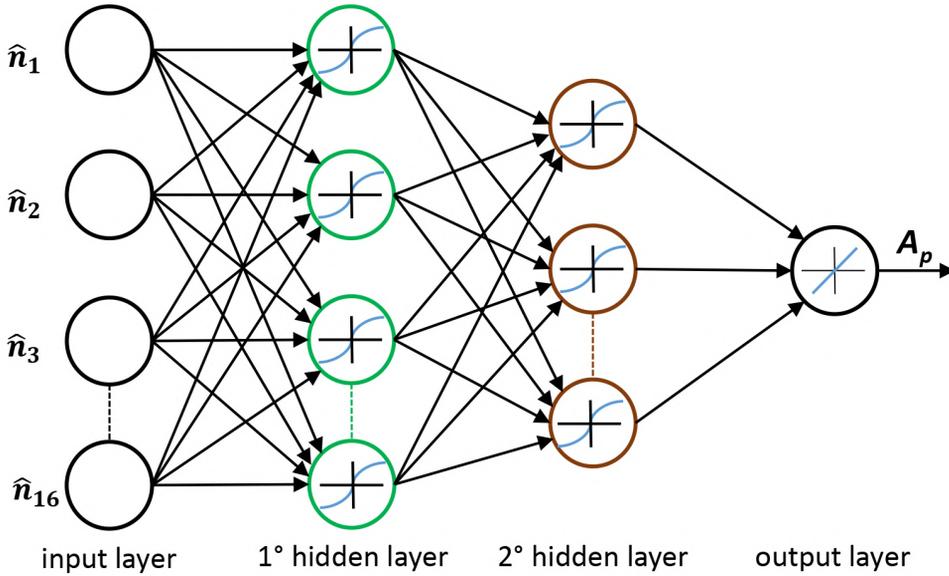


Fig. 4.4 The acyclic feed-forward neural network, which defines a mapping from 16 sample normals in object space to the output ambient occlusion. We evaluated three neural network with different hidden layers: neural network  $16 \times 8$ , neural network  $32 \times 16$ , and neural network  $64 \times 32$ .

coefficients  $w_{kj}^{i-1}$ . Let  $n_j^i$  be the output of the node  $j$  in the  $i^{th}$  layer and  $w_{j0}^i$  be its bias weight. Each node output of an hidden layer  $i$  is calculated from the outputs of all nodes in the  $(i - 1)^{th}$  layer as follows:

$$n_j^i = \sigma(z_j^i), \quad z_j^i = w_{j0}^i + \sum_{k>0} w_{kj}^{i-1} n_k^{i-1} \tag{4.2}$$

The summation is carried out over all neurons  $k$  transferring the signal to the  $i^{th}$  layer. The function  $\sigma(z_j^i)$  is the activation function and for all hidden layers is used the hyperbolic tangent function  $\sigma(z) = \tanh(z) = \frac{2}{(1+e^{-2z})-1}$ , while for the node in the output layer it is a linear function. The training process varies the bias weight  $w_{j0}^i$  and weight coefficients  $w_{kj}^{i-1}$  to minimize the sum of the squared differences between the computed and required output values.

### 4.2.2 Training

We use four<sup>1</sup> models and select 128 points of views by using a rotating camera around the bounding sphere containing the model. From each point of view, we use OptiX[120] to render the global Ambient Occlusion at a resolution of  $1920 \times 1080$ . From each image, we randomly pick 512 pixels from which we obtain the ambient occlusion values. We then take 16 normal samples in the object space located around the ambient occlusion value, in a circle of radius 10 as shown in Figure 4.5. These values represent the input data for our neural network, while the ambient occlusion value of the central pixel is the corresponding output data. When completed, the final dataset has 65,536 data points. The dataset extracted with this approach produces the best results experimentally.

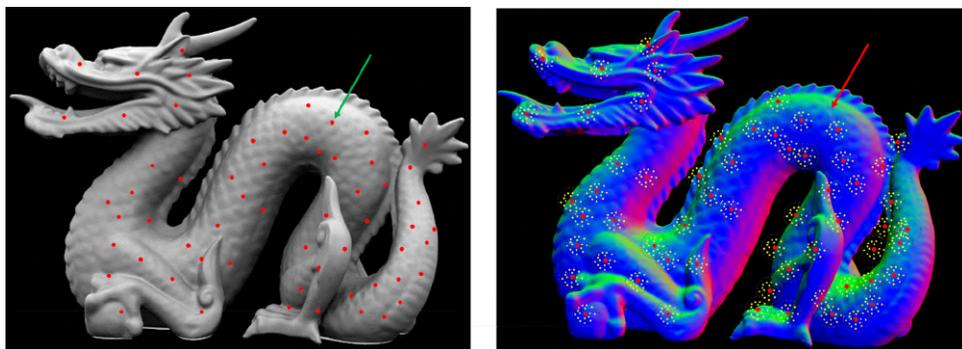


Fig. 4.5 Four 3D models were used to create a dataset for each of them. The Ambient Occlusion and normals values around the point were selected randomly. The normals components were saved in a texture in order to perform the sampling.

For the training, the dataset was split into three subsets: 50% as training set, 25% as validation set, and 25% as test set. Dataset splitting was done in a random way. The training was performed by using the scaled conjugate

<sup>1</sup> The used models were "Happy Buddha", "Dragon", "Stanford Lucy", "Stanford Bunny", which are freely downloadable: <https://graphics.stanford.edu/data/3Dscanrep/>

gradient backpropagation method [109], and performance was evaluated by using the mean squared normalized error performance function. The number of epochs and max fail were set to 20,000, while the other parameters were set to default. The neural network was trained by using a NVIDIA Tesla K40c installed on a machine equipped with an Intel Core i7-3820 and 16 GB RAM. The average training time for the neural network of  $16 \times 8$  hidden layers is about 4.50 minutes, for the network of  $32 \times 16$  about 5.55 minutes, and for  $64 \times 32$  about 7.53 minutes. Training was performed by using MATLAB Deep Learning Toolbox™.

### 4.2.3 Rendering

After training the feed-forward neural network, the ambient occlusion of each point of the 3D mesh can be rendered in real-time by using the trained model in a fragment shader. We developed a shader by using OpenGL Shading Language, which is a straightforward translation of the neural network. Weights and biases are provided to the fragment shader by using the uniform buffer for allocation of GPU memory. In particular, the memory occupation for neural network  $16 \times 8$ , neural network  $32 \times 16$ , and neural network  $64 \times 32$  are 3,716 bytes, 8,452 bytes, and 20,996 bytes, respectively.

In Figure 4.6 is shown the sampling of normals, which are computed in the object space by using a G-buffer and then projecting them in the screen space by using a texture. In more detail, for a given viewpoint we first compute the visible surface points of each screen pixel. For each pixel, we take its normal and then assign it as input the same value over all inputs of the neural network. This way of querying the neural network model is different in terms of the way we trained it, but the approach produces excellent ambient occlusion without either noise or blurring artifacts, thus avoiding the cost of a Gaussian blur pass. This corresponds to taking 16 normal samples located around a pixel in a circle of radius zero.

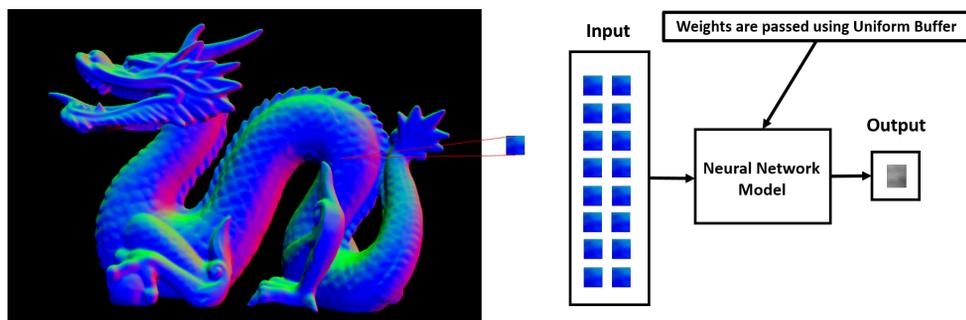


Fig. 4.6 Computing a visible surface is our first step. For each pixel, we got its normal and replicate this value in order to generate an input for the Neural Network. The weights and bias of the trained model are passed to the fragment shader using a uniform buffer.

## 4.3 Experimental Results

We implemented the rendering algorithm on a GPU via OpenGL API and GLSL shading language. All results and performance measures shown in this chapter were conducted on a PC with 3.50 GHz i7-5930K CPU, 16GB memory, and NVIDIA GTX 1070 graphics card.

In Figures 4.7, 4.8, 4.9, 4.10, we visually compare the results of our approach to the ground truth images rendered with OptiX, and Mara *et al.* (14 samples, 7.4 radius, 1 iteration, and 0.002 bias) [100]. These images are the final results of the fragment shader without any pass of Gaussian blur and other filters. Compared with the ground truth images, where there are geometric creases and contact shadows between triangle meshes, our approach can produce a result that is less dark because of the radius used to sample normals. Our approach produces good results in general, and visually there are slight differences between the three neural networks as shown in Figure 4.11.

In addition, because the training data are based only on the normals of the 3D model, depth resolution artifacts that occur in screen-space ambient occlusion techniques are not present regardless of the distance from the



Fig. 4.7 From left to right: neural network  $16 \times 8$ , neural network  $32 \times 16$ , neural network  $64 \times 32$ , ground truth with OptiX, and Mara *et al.* [100]. The 3D model shown in Figure is called Happy Buddha, which consists in 543,652 vertices and 1,087,716 triangles.



Fig. 4.8 From left to right: neural network  $16 \times 8$ , neural network  $32 \times 16$ , neural network  $64 \times 32$ , ground truth with OptiX, and Mara *et al.* [100]. Stanford Bunny is the 3D model shown in the Figure, it consists of 35,947 vertices and 69,451 triangles.



Fig. 4.9 From left to right: neural network  $16 \times 8$ , neural network  $32 \times 16$ , neural network  $64 \times 32$ , ground truth with OptiX, and Mara *et al.* [100]. The 3D model shown in Figure is called Dragon, which consists of 566,098 vertices and 1,132,830 triangles.



Fig. 4.10 From left to right: neural network  $16 \times 8$ , neural network  $32 \times 16$ , neural network  $64 \times 32$ , ground truth with OptiX, and Mara *et al.* [100]. The model shown in Figure is called Stanford Lucy and consists of 14,027,872 vertices and 28,055,742 triangles

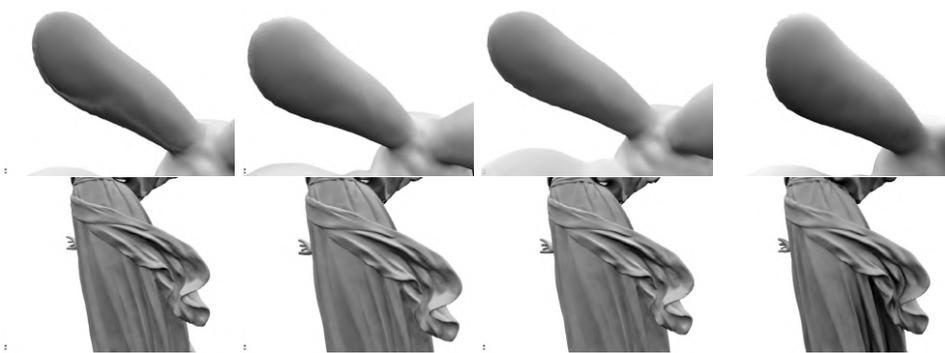


Fig. 4.11 Close-up of ambient occlusion results. From left to right: neural network  $16 \times 8$ , neural network  $32 \times 16$ , neural network  $64 \times 32$ , and ground truth with OptiX.

camera of the 3D model, however we tried to train our neural network by using the depth information as discussed in Section 4.4. To define the similarity between two images, we adopted the Structural Similarity Index [163] as a metric to measure similarity between images in a way that is consistent with human eye perception. In Table 4.1, we report the Structural Similarity Index between images generated with the three neural networks and those generated with OptiX.

In Table 4.2, we show the results of a numerical comparison between our method and that of Mara *et al.* All measurements are taken at  $1920 \times 1080$ . The neural network  $16 \times 8$  offers better performances. For reference only, in the same table we also report the performance of rendering the same models

MODEL	$16 \times 8$	$32 \times 16$	$64 \times 32$
Buddha	0.9646	0.9642	0.9655
Bunny	0.9814	0.9819	0.9793
Dragon	0.9495	0.9504	0.9489
Lucy	0.9743	0.9750	0.9754

Table 4.1 Structural Similarity Index between images generated with the three neural networks and those generated in Figures 4.7, 4.8, 4.9, 4.10 with OptiX.

MODEL	$16 \times 8$	$32 \times 16$	$64 \times 32$	MARA <i>et al.</i>	VERT. COL.
Bunny	1.96 ms	21.31 ms	29.29 ms	16.99 ms	0.19 ms
Buddha	2.74 ms	22.18 ms	29.97 ms	17.00 ms	0.67 ms
Dragon	2.17 ms	16.35 ms	22.76 ms	16.99 ms	0.55 ms
Lucy	2.25 ms	21.71 ms	29.54 ms	16.99 ms	0.32 ms

Table 4.2 Comparison between the three neural networks, Mara *et al.* [100], and vertex attributes rendering

by using vertex colours to store the baked ambient occlusion. We find that the neural network  $32 \times 16$  requires more computing resources, probably because of the loop performances, and neural network  $64 \times 32$  offers essentially the same performance. Nevertheless, we are confident that some optimization can be implemented to increase performance.

## 4.4 Discussion

In this chapter it was discussed an approach for performing screen-space ambient occlusion using a feed-forward neural network. The training phase is performed by using only normals of the 3D model in the object space and by using a rendering framework such as OptiX to create a high-quality ambient occlusion. We create a fragment shader that computes the values of the ambient occlusion in real-time efficiently and with accurate results. The

---

proposed approach offers a new way to use precomputed ambient occlusion during rendering for geometry from low to medium-grained high-frequency structures. The limit of this approach depends on the type of 3D model. Models with extreme variability and geometric complexity could cause network overfitting of the training data and fail to capture the distribution of the ambient occlusion over the model surface. Future works will focus on how to design dynamically the neural network based on geometric complexity. We will also study how to sample in a more efficient and effective way normals and the ambient occlusion values to avoid the selection of points of view when picking the pixels.

### 4.4.1 The Z-Buffer problem

Starting from the approach proposed by Martin Mittring [106] to use the depth values (or z-values) to compute the screen space ambient occlusion, we tried to use a similar approach by sampling the depth values into a depth buffer [103]. The idea was based on the combination of the normals and depth values of the scene, in order to create a more generalized dataset and optimize the performance of the neural network.



Fig. 4.12 The effect of the introduction of the depth-buffer in the dataset and in the training of the neural network. As shown in the figure, the rendering result is affected by the camera position.

After the training, we use the neural network in the fragment shader and we sampling the normals and the depth values into two respective buffers: normal and depth buffers. Although the training performance was promising, in the rendering phase we were faced with the typical problems of the use of the depth buffer. In more specific terms as shown in the Figure 4.12 the depth buffer changes as the observer's position changes and this behaviour was also found through the use of the neural network in the rendering phase demonstrating that the neural network is affected to the camera motion. In addition, this problem also occurred in the sampling phase because the camera has to be move on the z-axis to an indefinite number of times for proper neural

network training. For these reasons we did not use the depth buffer but we investigated the use of only normals, as described in the previous sections.



# Chapter 5

## Night time to Day time Approach

Starting from the Ambient Occlusion baking algorithm proposed and illustrated in the previous section, we have investigated the use of the deep learning to approximate the images lighting scheme. In particular, we propose an approach based on a fully convolutional neural network to convert night-time images to day-time images [22]. This chapter describes such approach by illustrating the design of the deep neural network and some preliminary results. The obtained results have led us to investigate further on the use of deep learning and image processing, as will be shown in Chapters 6 and 7.

### 5.1 Overview

As previously said, deep learning is often used in computer vision and image processing. Starting from this assumption, we develop an approach based on a supervised learning algorithm that through a deep neural network is able to process an image and simulate a filter that applies artificial or natural lighting. The idea is to transform an image or a scene with a low ambient light in a fully illuminated one as if it receives light from a lamp or the sun. The training dataset consists of pairs of input-output values where the input is composed of a set of images with low light and the corresponding outputs

are the same images with greater brightness. The goal of our work is to be able to transform an image obtained in night-time ambient to a day-time image. In order to achieve our goal, we built a neural network from a structure composed of 16 convolutional layers, of which 13 were convolutional and 3 were fully connected, developed by VGG-16 [150] team for ILSVRC-2014 competition. The first part performs a classification process, the second part is formed by a structure that is responsible for the decoding of the image. It uses an approach based on Deep Residual Learning [63]. To train our deep neural network, we used a tensorflow framework [2], described in section 2.3. To performing the experimentation, we created three datasets: the first two were synthetic, created through a software called Unreal Engine 4, and the third one was created in the real world by taking a lot of photos. Through their use, we developed three regression models, each of them can operate in a defined dataset domain. Each dataset represents a level of complexity where the neural network has to work and we evaluate the obtained results by performing a visual comparison. Others evaluations were performed through the neural network metrics obtained on validation dataset in the training phase and test dataset after the training phase. Due to the limited number of images we were able to collect, we can only demonstrate that our approach can effectively obtain good results. However, this work represents a first step of the deep neural network application on the image processing task, in fact in the next chapters we propose a more complexity task by using a similar deep neural network approach.

## 5.2 Fully Convolutional Neural Network

As mentioned in the previous section, our goal is to simulate natural or artificial illumination through deep learning with a supervised learning paradigm. The input of the neural network is an image with low brightness and the output will be the same image but with a sufficiently high illumination to

ensure a consistent view of it. We developed a neural network that consists of two main parts: a first part is structured as the well known VGG-16 [150] architecture, which identifies a pattern from the image and extract the features from it, in order to create a sufficiently complex function to approximate the lighting of the image itself.

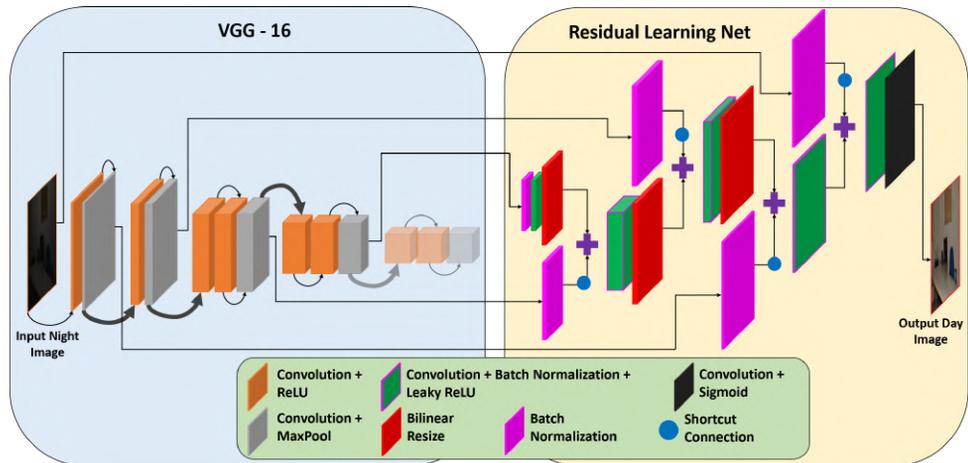


Fig. 5.1 The architecture of our own Fully Convolutional Neural Network. On the light blue background are represented only the 13 convolutional layers of the VGG-16 divided into 5 groups. We used the first four groups and concatenate them on the second part, which is shown on a light yellow background. Such part is based on residual learning net and represents our contribution.

As can be seen from the Figure 5.1, the first performed operation is the VGG-16 neural network import. Such neural network is composed by 16 layers of which we consider only the convolutional layers, divided into five groups. The first two groups are composed of two convolutional layers and the other three are composed of three convolutional layers. Among these groups is performed an operation called max-pool, which reduces the image size, represented by a tensor, through a non-linear operation of downsampling to adjust the tensor as the input of the next layer. As the image is represented using tensors, the output of each convolution operation is a tensor, called

activation tensor, which is passed as input to the Rectified Linear Unit [55] activation function of the next layer. We extracted the tensors from the last layer of the first four convolutional layers groups of the VGG-16. At such tensors is applied a normalization, called batch-norm, and they will serve us in the image decoding phase. During the training phase, there is a mutation of the neural network parameters, which involves also the distribution of the output of each layer, so subsequent layers have to adapt to this mutation. To solve this problem Ioffe *et al.* [70] proposed the batch normalization, which ensures that the input of the activation function has mean zero and variance equal to one:

$$BN(x_i) = \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} + \beta \quad (5.1)$$

Where  $\mu_B$  represents the mean and denominator is the standard deviation, the smaller constant  $\varepsilon$  is added to the variance in order to avoid dividing by zero.  $\gamma$  and  $\beta$  are learnable parameters.

The second part of our neural network is composed of so-called Residual Learning Net. Such an approach proposed by He *et al.* [63] describes a solution to the degradation problem that affects very deep neural network. In fact it is known that the number of features extracted from convolutional neural network increases in proportion to the number of layers of which it is composed. He *et al.* [63] in a Microsoft Research studio have demonstrated that in this case is noticeable a decrease of accuracy and an increase of error both in training and test phase. In more specific terms, considering the same dataset and iterations, a neural network with fewer layers achieves better result compared to a neural network with more layers, due to the degradation problem. As can be seen in the Figure 5.2, the solution to this problem is to allow the neural network to learn a residual function. In details, let  $H(x)$  a function that a layers group has to learn, if the problem is reposed so that the function to approximate is  $F(x) = H(x) + x$ , then a residual function will be obtained and the problem will be repeated in the form  $H(x) = F(x) + x$ ,

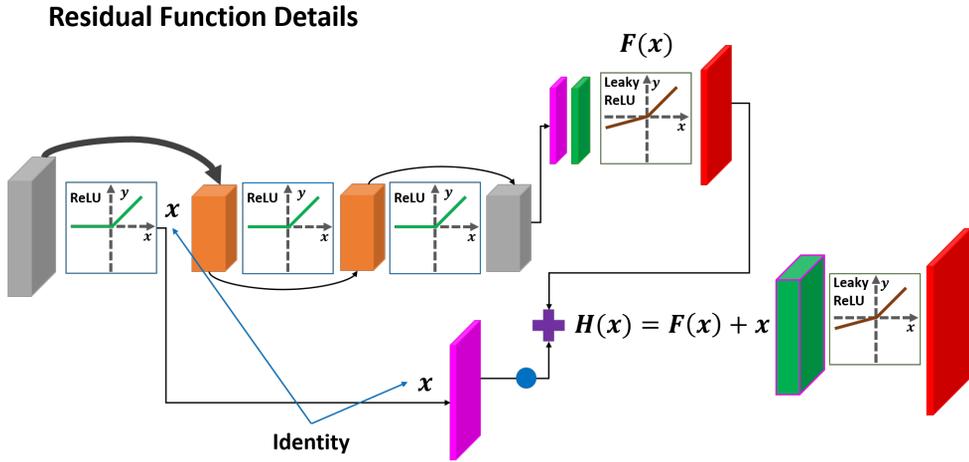


Fig. 5.2 Details on the proposed residual learning net. The figure shows a block of layers that implement  $H(x) = F(x) + x$ , and as can be seen the tensor  $x$  is extracted through the last layer of each group and is concatenated with the respective tensor, characterized of the same dimension and extracted from the residual learning net. For the VGG-16 parts we keep the Rectified Linear Unit activation function after each convolution, as proposed by the authors, while for our residual learning net component we used Leaky Rectified Linear Unit (See section 2 and Figure 2.4).

where  $F(x)$  represents the residual function and  $x$  represents the input of this function.

The residual function was learned by contiguous layers and the input of such group of layers is added to the output of itself through links called **shortcut connection** in order to attenuate the aforementioned degradation problem. The approximation of the residual function  $F(x)$  is performed through two or more layers as shown in the Figure 5.1. As the sum of the residual function and the input is performed element by element, it is needed to adapt these two dimensions. This operation is performed on the height and width dimensions (see Appendix A) through the bilinear interpolation [128]. The task of such operation is to move the pixel values, based on a parameter of scale, which defines how much the feature map must be enlarged or reduced.

In this way is possible to obtain empty pixels. To this pixels must be assigned a value through interpolation based on four known values around the considered pixel [128]. To obtain the same dimension even on depth we have performed a convolutional operation with a  $[1 \times 1]$  kernel and subsequently  $[3 \times 3]$  kernel.

The extracted tensor from the last VGG-16 convolutional layer of the fourth layers group has a 512 features map. Starting from this tensor, our residual learning net performs a normalization through a  $[1 \times 1]$  convolutional operation, which reduces the number of features map to 256, to sum this tensor and the output tensor of the last layer of the third layers group. This operation is performed after a bilinear interpolation that fits the width and height dimension of the tensor. The VGG-16 output tensor is an input of the residual learning net through a convolution operation using the  $[3 \times 3]$  kernel, stride 1 and "same" padding, which adds to the input convolutional tensor a border of zeros so that the output feature map has the same dimension. Such operation was performed through the following formula:  $\left(\frac{W-F+2P}{S} + 1\right)$  where  $W$  represents the tensor size,  $F$  is the local receptive field size,  $P$  represents the amount of zero padding on the border and, finally,  $S$  represents the stride. Continuing for the next convolution, bilinear interpolation is performed in order to double the width and height feature map dimensions. The subsequent operation consists of adding such resized feature map with the output tensor of the last layer of the second VGG-16 convolutional layers group after applying a batch normalization. The operations aforementioned are repeated until the input tensor and the last two convolution operations return a tensor that has only three features map, which represent the RGB channels, in addition to the width and height that characterize the output image.

At the end of the last convolution, a sigmoid activation function (see Figure 2.5) is used to normalize the output values between 0 and 1. To train our neural network, we use a well-known optimization method called Stochastic Gradient Descent [14]. The loss function is computed using the mean

squared error (see Appendix A):  $Error = \frac{1}{n} = \sum_{i=1}^n (y_i - t_i)^2$ , where  $t$  is the ground truth and  $y$  is the neural network prediction. Such loss function is computed on each input-output pair that is forwarded to the neural network. The weights are updated at each iteration by setting an hyper-parameter called batch size to 1. We also used a learning-rate equal to 0.0001 to adjust the learning speed to get better results. As our neural network consists of two parts, the values of the weights were initialized in two different ways: for VGG-16 (encoder component) we used a pre-trained model on more than a 1.2 million images (ILSVRC-2014), to make sure that what was learned in that given context is exploited to improve the level of generalization in our context [56] (see Appendix A); The residual learning net weights were initialized using a continuous distribution of probability called truncated normal distribution [17]. The values of the weights were generated by following the normal distribution in which the mean and standard deviation were specified. The normal distribution is expressed through the following formula:

$$F(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right) \quad (5.2)$$

where  $\mu$  is the mean of the distribution, computed on all possible values obtainable;  $\sigma$  is the standard deviation, which represents formally the dispersion of the points with respect to the mean and finally  $\sigma^2$  is the variance, which, given a set of points, represents the mean of the distance of such points from  $\mu$ . For our approach, we set  $\mu = 0$  and  $\sigma = 0.01$  (Figure 5.3) and, as our distribution is truncated, the values assumed by the weights vary between  $-0.02$  and  $0.02$ , while the values with greater magnitude were not used for the weights initialization. This approach can avoid that the gradient is amplified or completely dissolved, leading to critical errors during learning.

In many tasks, such as image processing, a normal distribution is a good choice to initialize the weights of own neural network. I. Goodfellow *et al.* [56] explain that the main benefits of using this distribution are two: (i) as established by **central limit theorem**, the sum of many independent random

variables is roughly normally distributed. Through such theorem is shown that very complex system as a neural network can be successfully modelled using a normal distribution; (ii) unlike other probability distributions, the normal distribution encodes the maximum amount of uncertainty on real numbers. In this way, it inserts the minimum amount of initial knowledge on the neural network and it can learn directly from the supplied data distribution (dataset).

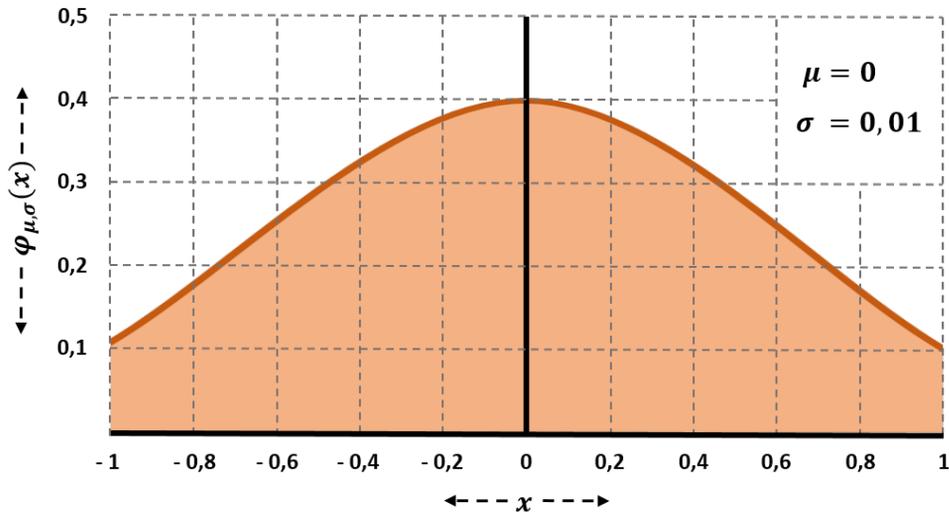


Fig. 5.3 The plotted graph of a truncated normal distribution with a classic bell curve shape, mean set to  $\mu = 0$  and the standard deviation is set to  $\sigma = 0.01$ .

Finally, another important consideration concerns the batch normalization operation used, which is done after every convolution operation. It reduces the probability that a bad initialization of weights leads to a non-optimal convergence.

### 5.3 Experimental Results

For the purpose of testing the performance and predicting the results of our deep neural network, we performed several experimentations. In order

to obtain accurate predictions, we defined three application domains, two indoor environments and an outdoor one. For each domain, we trained the deep neural network defining a regression model able to perform the correct prediction of images belonging to the same domain. The first domain is represented by a real indoor environment, whose dataset is composed of 523 pairs of input-output pictures, captured using a camera placed on a tripod to ensure a perfect matching between pictures.

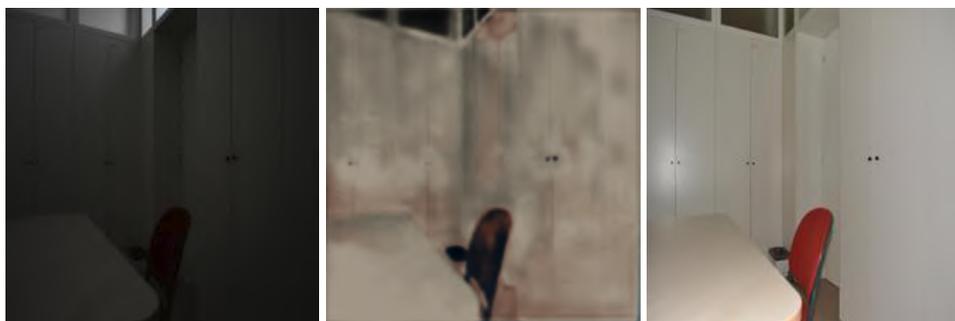


Fig. 5.4 Example of the night-to-day conversion on the real environment: The left image is the input of our deep neural network; the central image represents the prediction and finally the right image represents the Ground Truth.

Dataset images are taken at a resolution of  $6,000 \times 4,000$  and then cropped and resized to  $224 \times 224$ , which is the size accepted by our deep neural network as input. The picture with low illumination was obtained through the occlusion of all possible sources of light which illuminated the environment and the lighted image was obtained at the same angle but using the camera flash that sufficiently illuminated the small room (Figure 5.4).

The use of low resolution for the input and output images of the neural network is due to the aim of reducing neural network complexity and training time. We divided our datasets into training and validation set and, subsequently, we created the test set that contains only low-light images (Figures 5.5a and 5.6a). This dataset was used to test the accuracy of the neural network prediction. Our first dataset consisted of 473 images for the training

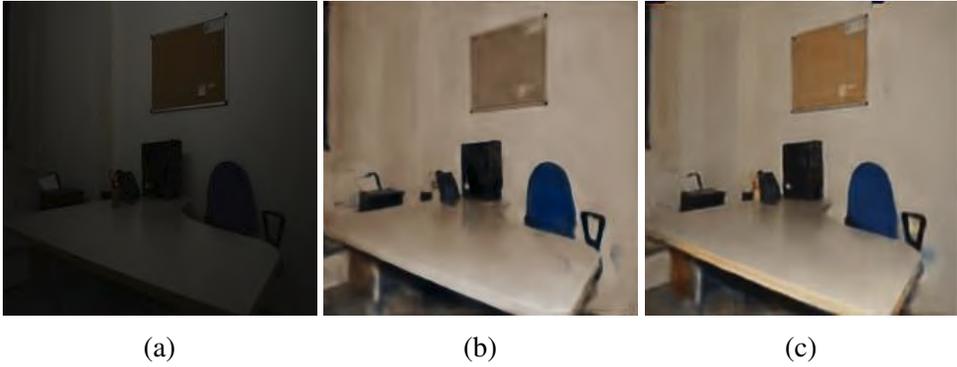


Fig. 5.5 Comparison between the results obtained after 1,000 and 6.6 million of iterations. The central image is an inference result performed after 1,000 iterations and the right image represent an inference result after 6.6 million of iterations.

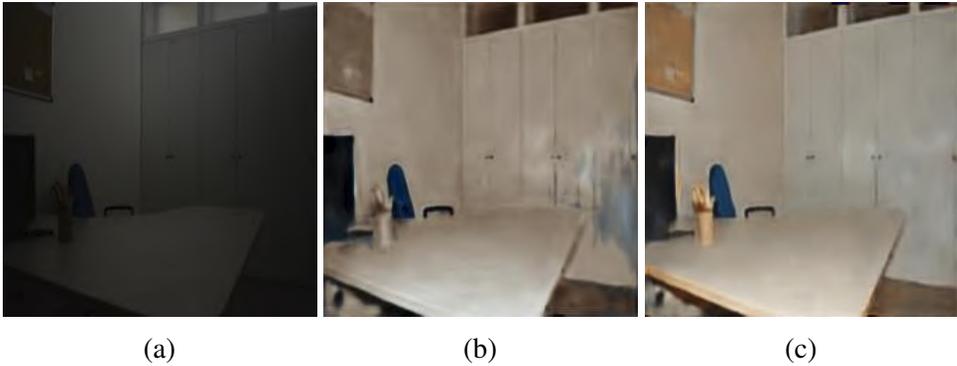


Fig. 5.6 Another example of inferences results obtained after 1,000 and 6.6 million of iterations. As can be seen, the right images obtained from our neural network trained for 6.6 million of iterations represents the best result.

set and 50 for the validation set. In this dataset, we added further images, using a technique called data augmentation that creates new images applying artificial noise to saturation and contrast of the dataset images, bringing the number of images to about 34,000. In the training phase, we used a system of dynamic change of the neural network input. In more specific terms, each iteration was characterized by a training operation and every two iterations a validation operation was computed. In the training iteration, was computed loss-training error using the training set of images, the later loss-validation error was computed using an image belonging to the validation set. For this dataset, the training time lasted around six days running about 6.6 million of iterations and showing a good grade of convergence. Figure 5.4 shows the result of an inference performed to the trained neural network after just 1,000 iterations by using an image that belongs to the indoor dataset. This results obtained after very few iterations has encouraged us to continue to test by training the neural network for more iterations and then for more time. In fact, the Figures 5.5 and 5.6 show the results of two inferences performed by using an image of the test set. The result of the first inference shown by the Figure 5.5a, was obtained after about 1.5 million of iterations, while the result (Figure 5.5c) of the second inference, performed on the same input image (Figure 5.5a) was obtained after 6.6 million of iteration in which it can be noted a much better improvement represented through a loss training value equal to 0.0005.

The second domain is a virtual environment created with the Unreal Engine 4. The virtual scene represents a landscape composed of trees and plants. We obtained a dataset consisting of 3,894 input-output pairs, of which 3,834 used for the training set and 60 used for the validation set. Data augmentation was applied, leading to an increase in the number of images to about 276,000. The images were obtained through a video in which a camera was moved along a default path within the virtual scene both in night-time and in the daytime. On this experiment, we needed about 0.6 million iterations to



Fig. 5.7 Results of the Unreal Engine outdoor scene obtained by performing inference on the neural network trained for 0.6 million of inferences: the left image is the input of the neural network; the prediction is represented from the central image and finally the right image represent the ground truth.



Fig. 5.8 Other results obtained by using Unreal Engine outdoor scene. As can be seen in the images, the neural network is able to well approximate the shadows and the colour of the corresponding ground truth image.

obtain good results. We extended the training until reaching 1.2 million of iterations but in this case, we found that the neural network was in overfitting by observing the trend of loss validation and verifying through the results of the inferences which were worse.

Through the Figures 5.7 and 5.8 we show the results of two inferences after 0.6 million of iterations with a loss training value of 0.0004. The last domain is a virtual environment also created with Unreal Engine 4. The scene, in this case, is an indoor environment and the images were taken in the



Fig. 5.9 Results of the Unreal Engine indoor scene obtained by performing inference on the neural network trained for 0.1 million of inferences. The images are arranged in the same way as the previous ones.



Fig. 5.10 Other results obtained by using Unreal Engine on the same indoor scene. As can be seen in the images, the neural network is able also to reduce the highlights like reflections, flashes, *etc.*

same outdoor mode. From the video images were extracted 2,028 images, of which 1,998 were used for the training set and 28 for the validation set. With data augmentation, images were increased to 143,000. In this case, we got a good convergence to 0.12 million of iterations. The Figures 5.9 and 5.10 show the results of two inferences after about 0.1 million of iterations. To obtain the better performance during the training phase, we exploited parallel computing allowed from the GPU through tensorflow functions. The tests were performed on a machine equipped with an NVidia Tesla K40 GPU, which allows optimal performance on the computation time.

## 5.4 Discussion

In this chapter we have investigated that the deep neural network is an useful solution to obtain a good simulation of the artificial and ambient light on the images. The main problem of our approach is the creation of a good dataset with a high amount of images that allows the neural network to achieve optimal results in generic environments. A supervised deep learning algorithm generally achieves acceptable performance with around 5,000 labeled examples per category and matches human performance when trained with a dataset containing at least 10 million labeled examples [56]. In our approach, we used datasets limited to isolated application contexts because, at the state of art, the use of heterogeneous environment does not lead to good results. Possible applications of this approach can be applied in the field of photo editing or digital post-processing of images taken in a low light environment. Our goal was to implement a proper filter that can be applied to the images. This is a work in progress and we are going to further investigate this neural network with a sufficiently large dataset, allowing to perform predictions regarding much larger and generic domains compared to those presented in this work. In fact in the next Chapter 6 we shows an approach based on this work to obtain a Flash / No Flash filter on images over-illuminated.

### 5.4.1 Details on VGG Convolutional Network

Karen Simonyac and Andrew Zisserman proposed a deep neural network called VGG [150] for image recognition. The proposed architecture is characterized from a high depth by using very small convolutional filters and performing several configurations among which, the most famous are with 16 and 19 layers respectively. Such neural network was trained by using images with  $224 \times 224$  as resolution. The used filter for each convolutional layer is  $3 \times 3$  as dimension and 1 as stride. After each convolutional layer, a

rectified linear unit was performed. VGG can be divided into 5 convolutional groups, each of them is followed by max-pooling operation with  $2 \times 2$  as pixel window and 2 as stride. After the convolutional layers, there are two fully connected layers with 4,096 neurons, one fully connected layer with 1,000 neurons and a softmax layer. The original neural network was trained using stochastic gradient descent with 0.9 as momentum, 256 as batch size and weight decay as 0.0005. The learning rate was 0.01 as the initial value, which was decreased 3 times by a factor of 10 and for the fully connected layers a dropout [151] regularization with 0.5 as the ratio was introduced. The training was performed at 74 epochs and the learning rate was decreased when the accuracy of the validation stopped improving. In the first neural network configuration, the weights are initialized by using a normal distribution with zero mean and 0.01 as variance, while in the other configurations the first four convolutional layers and all fully connected layers are initialized by using the weights of the first configuration. The VGG authors have shown that increasing the number of convolutional layers in the individual groups of the network improve its performance. Furthermore, the structure of the VGG is an optimal solution when used as an encoder in a U-shape net.

Since in our version we used the only first 4 convolutional groups, we have removed the dropout regularization because we used the VGG as the encoder of our network. Other parameters and configuration of our neural network has been described in Section 5.2.

### 5.4.2 Details on Residual Learning Network

As explained in Section 5.2, residual learning networks were designed to address the degradation problem. Such problem concerns the very deep neural network where was observed that as the depth increased, the training accuracy was reduced. To solve this problem Kaiming He *et al.* [63] propose a deep residual learning framework (more detail in Section 5.2) that also allows to avoid the vanishing gradient problem (see Section 8.1.3). In particular, starting

from a current layer and reusing the activations from the previous layer, by using the connections called skip connections (or shortcut connections) until the next layer have learned its weights, the degradation and vanishing gradient problems are greatly reduced.

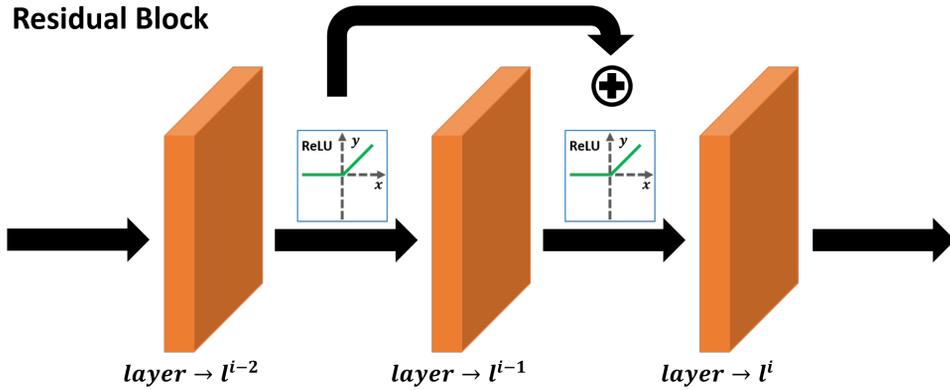


Fig. 5.11 The figure shows the residual block. The top arrow represents the skip (or shortcut) connection.

In more specific terms, let  $l^i$  the current layer,  $l^{i-1}$  and  $l^{i-2}$  the previous layers as shown in the Figure 5.11 and let  $w^{i-1,i}$  and  $w^{i-2,i}$  the weights for the connections of the previous layers with the current layer respectively, then in the activation function we have a forward propagation defined as

$$x^i = \sigma \left( w^{i-1,i} x^{i-1} + w^{i-2,i} x^{i-2} \right) = \sigma \left( z^i + w^{i-2,i} x^{i-2} \right), \quad (5.3)$$

Where  $x^i$  and  $\sigma$  are the output and the activation function of the current layer respectively,  $x^{i-1}$  and  $x^{i-2}$  are the outputs of the two previous layers and finally  $z^i = w^{i-1,i} x^{i-1}$  is used to simplify the notation, moreover, the biases have been omitted for simplicity. Previous equation is valid for one skipped layer, while for the several layers the equation can be rewritten as

$$x^i = \sigma \left( z^i + \sum_{k=2} w^{i-k,i} x^{i-k} \right). \quad (5.4)$$

In the backpropagation phase the gradient of the loss function without the residual block can be computed for each layer as

$$\nabla_{w^{i-1,i}} = -\varepsilon \frac{\partial L^i}{\partial w^{i-1,i}} \quad (5.5)$$

Where  $L^i$  is the error at the layer  $l^i$  and  $\varepsilon$  is the learning rate. If we include the residual block with several skip connections the equation can be rewritten as

$$\nabla_{w^{i-k,i}} = -\varepsilon \frac{\partial L^i}{\partial w^{i-k,i}} \quad (5.6)$$

Kaiming He *et al.* [63] have evaluate their method on the ImageNet 2012 (dataset) for the classification task by comparing plain networks and residual networks using 18 and 34 layers. The plain network with 34 layers obtained training and validation errors greater than the plain network with 18 layers, showing the degradation problem. The same comparison was performed by using the residual learning networks with 18 and 34 layers. In this case both configurations obtained better results than the plain networks, but the residual learning network with 34 layers obtained lower error values than the residual learning network with 18 layers. In this way the authors demonstrated that the degradation problem was avoided. As positive side effect, the residual learning network can regularize the model and consequently it can reduce overfitting problem.



# Chapter 6

## Deep Flash face photos

Through the knowledge obtained with the application of deep learning in the modification of the image lighting scheme, as explained in the previous chapter, we continued to investigate on this front. In this chapter we, show a method based on convolutional neural network for turning a flash selfie into a photograph as if it had been taken in a studio setting with uniform lighting [20]. We show how our method can amend defects introduced by a close-up camera flash, such as specular highlights, shadows, skin shine, and flattened images.

### 6.1 Overview

With the steady improvement of built-in digital cameras, pictures taken on smartphones and computer tablets are becoming increasingly predominant on the internet, even on web-based services dedicated to quality photography, such as Flickr, 500px, and Instagram. However, if it is true that in favorable light conditions, smartphones can take pictures that are comparable to those of digital reflex cameras, it is also true that they perform poorly in low light conditions. This is mainly due to the size of their sensors, which is a constraint difficult to overcome within the small space of a smartphone. It follows that,

often, taking pictures in low light triggers the camera flash, which is typically a low-power light-emitting diode (LED) flash mounted side by side with the camera lens, and results in images with high noise. One of the most common type of photograph taken with a smartphone is the so-called selfie, which is a picture of one’s face taken by holding the phone in the hand or by using a "selfie stick". Low-light flash photographs and selfies are an unfavorable combination that produces images with specular highlights, sharp shadows, and flat, unnatural skin tones. Therefore, researchers have recently started to develop several correction techniques: re-lighting and enhancement of images with non-uniform lighting [18, 161], some of them have focused mainly on the images of faces [146, 162, 164, 148].



Fig. 6.1 Two examples from our results. The split images show a comparison between the input and the output of our algorithm. In the central column the input, output, and ground truth images.

In this chapter we explore the possibility of taking smartphone flash selfies and employing a convolutional neural network to turn them into studio portraits. Convolutional neural network have already been extensively used to improve pictures, for example for creating high dynamic range images from single exposure [42], colorization [69], super-resolution [94], and so on, as will be discussed in Section 9.6.2. However, our problem is especially challenging for at least three reasons. Firstly, it concerns an effect that has both local and global discriminant features such as highlight and skin tone,

respectively. Secondly, we want to imitate a process that humans are very good at performing; *i.e.*, to picture what an image would look like if flash was not used. Finally, both previous points apply to the domain of human faces, on which humans are extremely good to spot any kind of inconsistencies. We leverage the fact that, by their nature, smartphone flash selfies share many common traits and make a fairly well-defined sub-domain of photographs: they are front or three-quarter single-face portraits, taken from less than one meter away with a single flash collocated with the camera lenses. Our approach consists of training a convolutional neural network with a series of pairs of portraits, where one is taken with the smartphone flash and one with photographic studio illumination. The two photographs of the same pair are taken as simultaneously as possible, so that the pose of the subject is the same.

## 6.2 Turning a Flash Selfie into a Studio Portrait

We designed a regression model targeted to the restricted domain of human faces. We adopted a supervised approach where a convolutional neural network is trained by feeding pair of flash and no flash portraits. Although the main idea is straightforward, there are several details that need to be addressed in the design of the network, the training procedure, the loss function and the encoding of the problem. All these aspects are discussed in the following sections.

### 6.2.1 Deep Neural Network

Our convolutional neural network is an U-shape Net that consists of two sub-networks: the first network takes as input a flash image and performs the encoding to create a deep feature map representation; the second network takes as input the encoder output and recreates the image without the flash

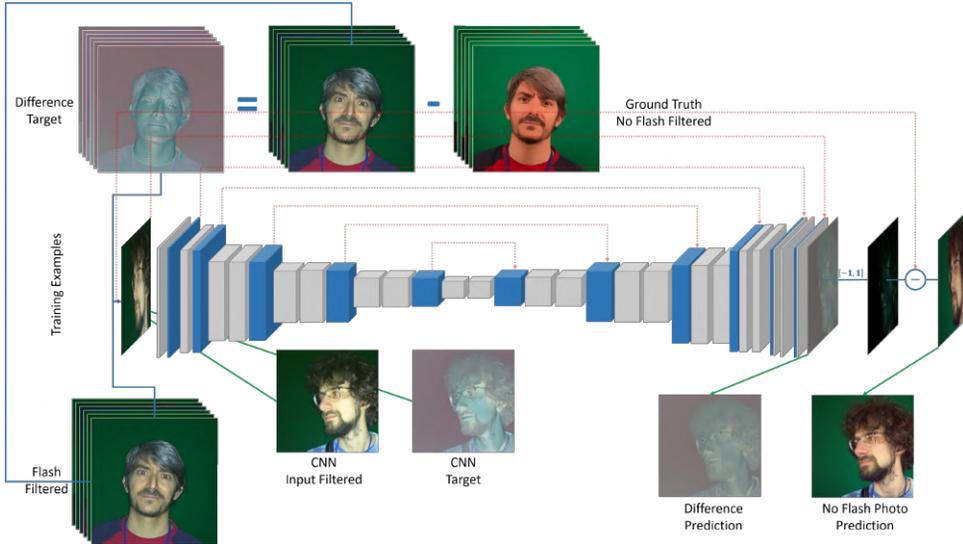


Fig. 6.2 Our neural network architecture for transforming a flash image into a non-flash image. The first 13 blocks represent the VGG-16 convolutional layers, which perform the image encoding. The second part reconstructs the output image and has several convolutional and deconvolutional layers. From the blue blocks of the VGG-16, the shortcut connections start, which are linked with their counterparts in the decoder. The convolutional neural network input is an image taken with the smartphone flash, and the ground truth is an image taken using simulated ambient light, on both of which the bilateral filter is applied. The target image is the difference between the input and ground truth image, normalized in a range between 0 and 1. The network prediction is the searched difference, which is denormalized in a range between  $-1$  and  $1$  and then subtracted from the non-filtered input. The final output prediction is an image without flash highlights.

defects. The input image encoding is performed by the VGG-16 [150] network. As in the previous work (see sections 5.2 and 5.4.1) we used only the convolutional layers but in this case of the all five groups. We developed the decoder component, which performs the decoding task, based on Eilertsen *et al.*'s approach [42]. In particular, the input of the network is the output from the last VGG-16 convolutional layer after a further convolution operation.

After each convolution a batch normalization [70] (see Equation (5.1)) is performed forcing the input of the activation function to have mean zero and unit variance. After each batch normalization, the obtained activation tensor, crosses the next activation function, Leaky Rectified Linear Unit [98], which introduce a non-zero gradient for the next inputs. To obtain optimal convolutional neural network performance in the training phase, the slope parameter  $\alpha$  is set to 0.2 [165].

The main features of the decoder layers are operations such as convolutions, batch normalization, deconvolutions, and concatenations. Because our network is composed of many layers, to avoid the vanishing gradient problem [64] also studied by He *et al.* [63], we used an approach based on a residual learning network. This problem concerns the weight update in the backpropagation phase, which is proportional to the gradient of the error function compared with the weight that has to be updated. Progressing in the backpropagation phase and inverse crossing the network to update the weight may mean that the gradient is so small as to make inefficient updates on the weights belonging to the first deep neural network layers, and consequently, their training is stopped. A simple way to solve this problem is to use a block division of the VGG-16 and concatenate in depth each block output with its counterpart in the decoder through a link called a shortcut connection. For this reason, we use concatenations layers[67].

The proposed U-shape Net works in the RGB (red, green, blue) domains only, and it is able to recreate similar input images of faces, but in a different light mode. Starting from the VGG-16 output tensor, to reconstruct the output image, we use deconvolutional layers [171], which transpose the convolutional layers.

### 6.2.2 Training

Our convolutional neural network was trained to minimize the loss function using an algorithm called the Adam Optimizer [84]. This algorithm is a

stochastic gradient descent with momentum variant [138], which manages in a different way the problem of setting the learning rate. The choice of the learning rate can influence the convolutional neural network training convergence because a high value can lead to a possible divergence, while a very low value can lead to a slow convergence. Stochastic gradient descent with momentum addresses this problem by updating weights through a linear combination of the gradient and previous updates. Adam is a Stochastic Gradient Descent with Momentum variant, which is based on two other well-known ones, called AdaGrad [40] and RMSProp [156]. These are classified in the category of adaptive algorithms because they adapt the learning rate for each of the parameters, leading to better convergence results. In particular, Adam combines the advantages of the methods mentioned above: It adapts the learning rate based on the first and second gradient moments. In our Adam configuration, the initial learning rate is set to  $10^{-5}$ , while  $\beta_1$  and  $\beta_2$ , called the forgetting factors, are left at the default values, 0.9 and 0.999, respectively. A parameter,  $\epsilon$ , used to avoid divisions by zero, is set to  $10^{-8}$ .

To increase the generalization level and to compensate for the amount of the training data available, we initialized the weights of the VGG-16 encoder using a pre-trained model, which is used for face recognition [121], exploiting the transfer learning concept [126]. The model was trained using a very large-scale dataset that consists of 2.6 million faces belonging to 2,600 identities (about 1,000 images for each identity), using four GPU Titan Blacks. The input images of the pre-trained model have a resolution of  $224 \times 224$  pixels, from which was subtracted the mean of the training set images. The weights of our decoder were initialized using truncated normal distribution [17], which ensures that the weights values have mean zero and unit standard deviation. The weights of the last decoder layer were initialized using Xavier Initialization [54] (see Appendix A), which ensures that the weights are neither too large nor too small and that the signal passing through the neural network is propagated accurately. This prevents the signal from

being amplified or reduced too much due to excessively large or small weight initialization.

### 6.2.3 Problem Encoding

	NN input	NN target	NN prediction	Output
A	 $x_i$	 $o_i$	 $y_{iA}$	 $y_{iA}$
B	 $x_i$	 $x_i - o_i$	 $y_{iB}$	 $x_i - y_{iB}$
C	 $B(x_i)$	 $B(x_i) - B(o_i)$	 $y_{iC}$	 $x_i - y_{iC}$
	 $B(o_i)$			 $x_i - (B(x_i) - B(o_i))$

Fig. 6.3 Each row of the table shows a possible (and tested) encoding of the problem. For each row, the neural network input, target, prediction, and algorithm output are shown. The last row shows the bilateral filter applied to the uniform lit image (left) and the best achievable result with approach C.

The network can be used in more than one way to achieve our goal. The table in Figure 6.3 illustrates three alternative encodings of the problem, showing the neural network input, target, prediction, and algorithm output for each one. We indicate with  $x_i$  the flash image,  $o_i$  the uniformly lit image, and  $y_{i[A|B|C]}$  the network prediction. The most straightforward solution, shown in the first row of Figure 6.3, consists of training the neural network by providing  $x_i$  as the input,  $o_i$  as the target, and, once trained, to take the network prediction  $y_{iA}$  as the final output. With this setting, the network converged and gave good results in terms of colours and chromaticism. On the downside, the predictions were blurred, and small misalignments of facial expressions between  $x_i$  and  $o_i$  (e.g., eyes closed/open, the position of the lips, and other facial landmarks) created very visible artifacts in the predicted images. To reduce the blur in the images, we can train the network, giving as input the flash image and as the target the difference between  $x_i$  and  $o_i$ . This way, the artifacts due to the alignment of facial expressions were greatly reduced and the training was simplified. Through this approach, shown in the second row of Figure 6.3, the results visibly improved, but the blur was not entirely removed. Inspired by these results, we chose an encoding that decouples high-frequency details such as hairs of facial features from low-frequency characteristics such as the global skin tone. We employed an accelerated version of the *Bilateral Filter* [8],[157], which is a nonlinear filter that is ubiquitously used to smooth images preserving the edge features. In practical terms, a weight is assigned to each pixel of the image to be filtered, depending both on spatial proximity (spatial domain and on photometric similarity (range domain or intensity domain). The idea behind many spatial filters is based on the requirement that neighboring pixels tend to have similar values. However, this idea turns out to be incorrect on the borders of objects in images because, in these points, the signal changes quickly. The bilateral filter considers this feature and replaces the intensity of each pixel with a weighted average of the intensity value of the neighboring pixels. Through this filter, we remove the input and the ground

truth image high frequencies, retaining the low frequencies and we compute the distance between two images, normalizing it to  $[0, 1]$ . The network input is the filtered flash image, and the target is the distance between the filtered input and the filtered ground truth:

$$t_i = \frac{[BL(x_i, \sigma_s, \sigma_r) - BL(o_i, \sigma_s, \sigma_r)] + 1}{2} \quad x_i, o_i, t_i \in [0, 1], \quad (6.1)$$

where  $BL$  is the bilateral filter operator,  $\sigma_s = 16$  is the spatial sigma value, and  $\sigma_r = 0.1$  is the range sigma value. The parameters of the filter were selected after an experimentation to determine the values that could create high quality results. The final reconstructed image is computed as:

$$pred_i = x_i - 2y_i + 1, \quad (6.2)$$

where  $y_i$  is the network output.

#### 6.2.4 Loss Function

As mentioned above, we trained our neural network with images filtered using the bilateral filter as the input, and the distance between input and ground truth filtered with the same filter as the target. The aim was to preserve the low frequencies and retrieve them in a subsequent step from the original non-filtered image. For this reason, we minimized the distance between the low frequencies of the input and ground truth. The objective function is therefore defined as follows:

$$L(y_d, t_d) = \frac{1}{3N} \sum_i \left( (y_{di} - \mathbb{E}[y_{di}]) - (t_{di} - \mathbb{E}[t_{di}]) \right)^2, \quad (6.3)$$

where

$$\begin{aligned} y_{di} &= BL(x_i, \sigma_s, \sigma_r) - 2y_i + 1 \\ t_{di} &= BL(x_i, \sigma_s, \sigma_r) - 2t_i + 1 \end{aligned} \quad (6.4)$$

In more specific terms,  $N$  is the number of pixels,  $BL(x_i, \sigma_s, \sigma_r)$  is the convolutional neural network input,  $x_i$  is the flash image,  $y_i$  is the predicted difference of the convolutional neural network, and  $t_i = BL(x_i, \sigma_s, \sigma_r) - BL(o_i, \sigma_s, \sigma_r)$ , where  $o_i$  is the ground truth. The arguments of the bilateral filter are the same in equation (6.1). Replacing equation (6.4) in equation (6.3), and by simplifying and exploiting the linear property of the mean, the objective function can be rewritten as

$$L(y, t) = \frac{4}{3N} \sum_i \left( (t_i - y_i) + \mathbb{E}[y_i - t_i] \right)^2. \quad (6.5)$$

To avoid negative values affecting the convolutional neural network convergence due to activation functions, we normalized the target difference image in the range  $[0 \dots 1]$  (6.2). In particular, since *Rectified Linear Unit* and *Leaky Rectified Linear Unit* are non-saturating nonlinear activation functions [86] [154], they tend to eliminate completely or partially the negative output values of each layer, leading to faster convergence than saturating nonlinear activation functions such as  $\tanh$ , which lead to longer training times and a slower convergence. Furthermore, it was shown that rectified units are much more efficient for tasks concerning images [111] [55]. The network, therefore, will perform predictions in the  $[0, 1]$  range; for this reason, the values are reported in the  $[-1, 1]$  range and subsequently subtracted from the input values. If we consider the bilateral filter function and perform a further substitution, we can insert the original non-filtered images into equation (6.4), and the objective function can be explicitly rewritten as

$$L(y, x, o) = \frac{4}{3N} \sum_i \left( (BL(x_i, \sigma_s, \sigma_r) - BL(o_i, \sigma_s, \sigma_r) - y_i) + \mathbb{E}[y_i - BL(x_i, \sigma_s, \sigma_r) + BL(o_i, \sigma_s, \sigma_r)] \right)^2 \quad (6.6)$$

Mean subtraction is performed for each channel of each image pixel by pixel only in the evaluation phase of the objective function, to centralize the data and to distribute the weights of each image across the training in a balanced manner, so that each image gives the same contribution to the training and does not have more or less important than the others. This normalization operation is performed differently for every single image, compared with the classic method of centralizing the data, which involves subtracting from each image the mean computed across the whole training dataset. Because our problem is confined to a specific domain, in which the data are stationary and the image lighting parameters are well defined and always the same both for the input and for the output, we subtracted the mean for each single image, which was computed on the same image to remove the average brightness or intensity from each pixel.

## 6.3 Experimental Setup - Dataset Creation

We performed neural network training using pairs of photos taken with a 13 Megapixel Nexus 6 smartphone camera. The photographs were taken in a studio equipped with four Lupoled 560 lamps to provide uniform illumination. For each pose, a photograph was taken with the lamps on, which were then immediately switched off, and a second photograph was taken with the smartphone flash only. Because the switching off the lamp imposes a significant delay between the two shots (about 400ms with our lamps), the pose of the face between the first and the second may change significantly. To reduce

the problem of face misalignment, we performed an *affine* alignment (*i.e.*, translation, rotation, scale, and shear) using the MATLAB Image Processing Toolbox™. In particular, we considered the non-flash image as the misaligned image ( $M$ ) and the flash image as our reference ( $R$ ). Because the images have different lighting conditions, we employed a *multimodal* metric [130] [102] and optimizer[153]. Once the geometric transformation was estimated, we applied it to  $M$ , obtaining  $M'$ , which is a better alignment to  $R$ . Note that a limitation of this approach is that misalignments remain between open and closed eyes. After affine registration, we identified the face of a subject in  $M$ ,  $M'$ , and  $R$  using the Face Recognition API <sup>1</sup>, which returns a bounding box for the photograph. Each bounding box is used to crop the image, and then the image is downsampled to  $512 \times 512$ . In this way, the convolutional neural network takes the *global information* on the images. We collected about 495 pair of photos of 101 of men and women in different poses. The dataset has then been augmented in three ways. First, through 5 rotations from  $-20$  to  $20$  degrees around the center of face bounding box, using a 10 degree step. Second, by cropping the image to the face bounding box and rescaling to original image size. Finally, images are flipped horizontally. Altogether, we augmented the initial set of examples by a factor 20, therefore our training is done with 9.900 images with  $3120 \times 4160$  resolution (13 Megapixel).

## 6.4 Results

We evaluated the results in validation and test sets. In particular, the convolutional neural network was trained using pairs of images with a resolution of  $512 \times 512$  in about five days using a NVIDIA Titan Xp GPU and by performing 62 epochs and about 458,000 backpropagation iterations. We interrupted the training when the value of the loss function computed on 1500 images reached a low level of approximately 0.0042. To evaluate the similarity ac-

---

<sup>1</sup> Face Recognition API: [https://github.com/ageitgey/face\\_recognition](https://github.com/ageitgey/face_recognition)

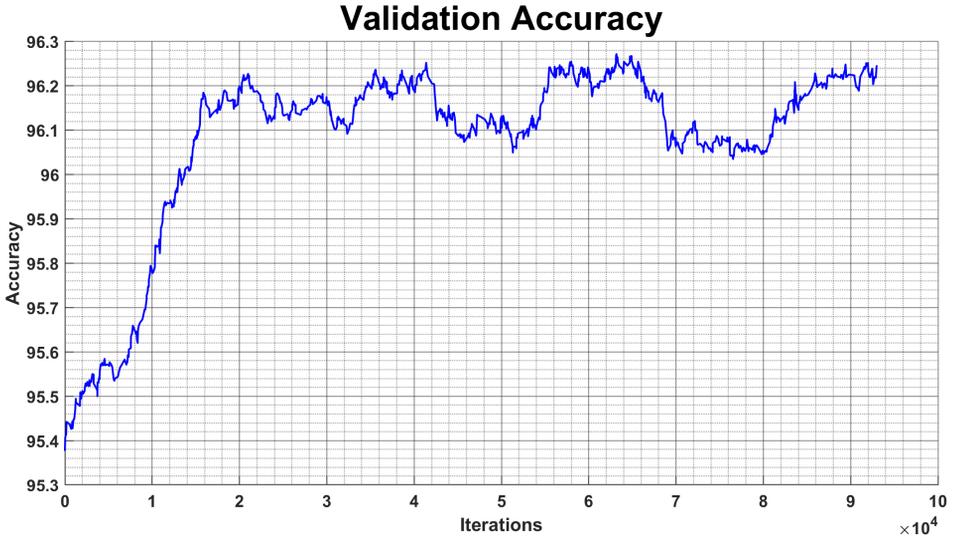


Fig. 6.4 Accuracy trend in the validation set at the end of training the convolutional neural network after 62 epochs and 458,000 iterations performed in 5 days.

accuracy between the current prediction and ground truth, we computed the percentage difference between the two images as follows:

$$acc = 100 - \left( \frac{100}{3w(I)h(I)} \sum_i \sum_c |I_c - \tilde{I}_c| \right) \quad (6.7)$$

where  $I = t_d$ ,  $\tilde{I} = y_d$ ,  $w(I) = width(I)$ , and  $h(I) = height(I)$ . After the training step, we obtained an accuracy value of 96.2% (Figure 6.4). In the test phase, we evaluated our approach using 740 test images, obtaining a loss-test value of 0.0045 and an accuracy value of 96.5% (Table 6.1).

### 6.4.1 Comparison with reconstructed Ground Truth

As explained in Section 6.2.3, the result provided by our pipeline is obtained by subtracting the convolutional neural network prediction from the original input image, allowing us to retrieve the high frequencies, which had been lost

due to the bilateral filter. It follows that even for a loss function  $L(y, x, o) = 0$  the exact ground truth can never be reconstructed. Furthermore, and more importantly, the misalignments due to pose changes between the flash and non flash photos would dominate when computing the input and output image differences. For these reasons we introduce a preconditioning operator on the ground truth as

$$\bar{o}_i = x_i - 2t_i + 1 \quad (6.8)$$

where  $t_i$  is the target difference, as explained in Section 6.2.4. Figure 6.5 shows the final result for some examples of the validation data, while Figures 6.6 and 6.7 show, similarly, the reconstruction of example images belonging to the training and test sets, respectively. In particular, it can be seen in Figures 6.5 and 6.7 that the information lost because of the flash, such as hairs, beard and skin colour, are retrieved through the convolutional neural network prediction. The shadow cast by the subject is also corrected by the convolutional neural network to reduce the highlights due to the camera flash. By reconstructing the final image through the non-filtered input, we retrieved the high frequencies resolving the blur problem. We evaluated the data using the Structural Similarity Index [163] for each subset of the dataset. In particular, as can be seen from Table 6.2, which shows a comparison between ground truth and the reconstructed predictions (see Figures 6.5, 6.6, and 6.7), the Structural Similarity Index average value is around 80% for the validation set, around 92% for the test set, and around 94.5% for the training set. As a final step, we run a Red Eye Removal filter with GIMP to present the final result. Please note that the metrics were calculated considering the images without removing the red-eye artifact.

	LOSS	ACCURACY
VALIDATION	0.0042	96.2%
TEST	0.0045	96.5%

Table 6.1 Loss validation and loss test after 62 epochs. This table also shows the maximum accuracy achieved in both phases.

SSIM	Left	Center	Right
Fig. 6.5: Validation	87.5%	76.0%	70.0%
	78.5%	81.1%	80.7%
	83.7%	73.8%	85.9%
Fig. 6.6: Train. Fig. 6.7: Test	First row	Second row	Third row
	94.8%	89.3%	79.5%
	91.0%	91.0%	92.3%

Table 6.2 The Structural Similarity Index (SSIM) of the reference images: The first three rows show the Structural Similarity Index of Figure 6.5 (starting from the top of the image) computed by comparing the central image of each group and the image at the top right. The subsequent rows show the Structural Similarity Index values related to Figures 6.6 and 6.7.

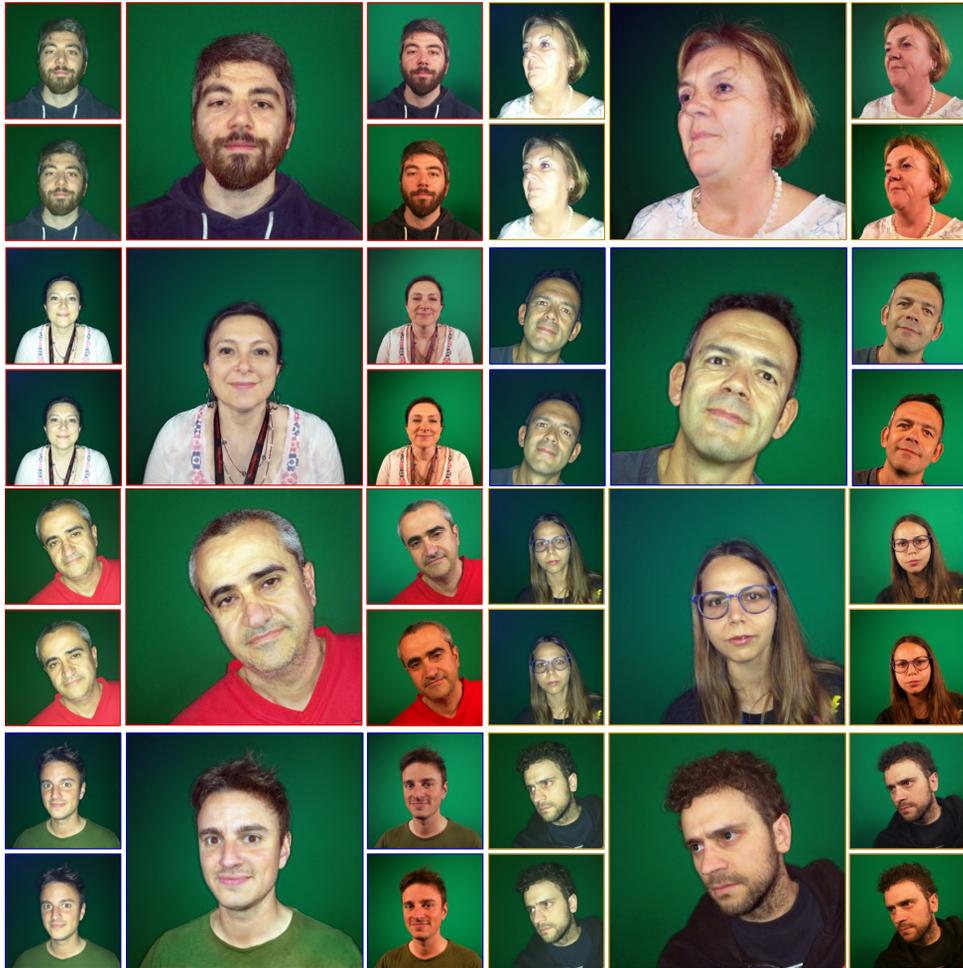


Fig. 6.5 Samples of validation data. Each group of images is composed of: the original image taken with the smartphone flash (top left); the flash image to which the bilateral filter was applied (bottom left); the image reconstructed by the difference prediction of the convolutional neural network (center); the ground truth reconstructed (top right); the ground truth (bottom right).

### 6.4.2 Comparison with other approaches

Our approach was compared with the similar approach of the state of the art. In particular, we perform three comparisons with three different approaches:



Fig. 6.6 Training set example. For each row: the original input (left); the image reconstructed by convolutional neural network prediction (center); the ground truth reconstructed (right).

the first approach is called HDRNet[53]; the second approach is based on Conditional Generative Adversarial Network [105] and is called Pix2Pix [71]; the final approach is non-deep learning based but on the direct style transfer between pairs of images [146].



Fig. 6.7 Test set example. For each row: the original input (left); the image reconstructed by convolutional neural network prediction (center); the ground truth reconstructed (right).

#### 6.4.2.1 Comparison with HDRNet

We compared our approach to HDRNet by Gharbi et. al [53]. This work is based on the use of a convolutional neural network inspired to the bilat-

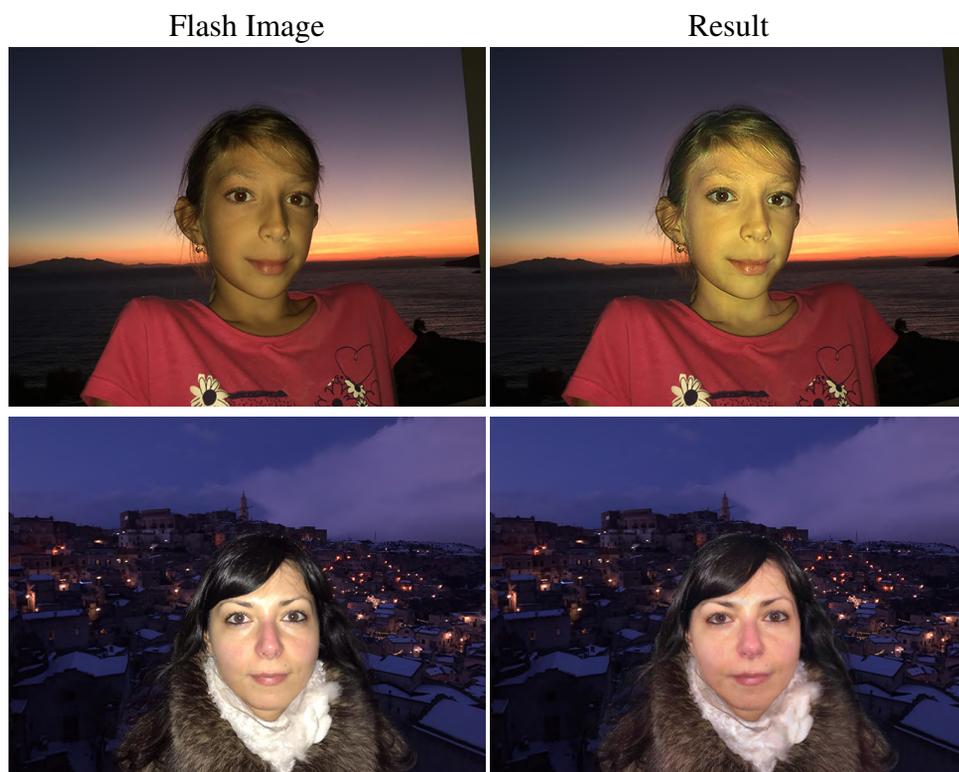


Fig. 6.8 Two example of a real images. The subjects were detected using Photoshop Subject Detection tool and were forwarded to our neural network with a green background. The results were blended with the original images.

eral grid processing [24] and to local affine colour transforms. HDRNet is designed to learn any image operator and hence is a suitable candidate to remove flash artifacts from photographs. In a first experiment, we trained HDRNet end-to-end with our dataset, by feeding the network with input and target images. We used the same parameters, times and training algorithm proposed by the authors and trained the network for 48 hours. We obtained a stable loss value of 0.0031. Figure 6.9 shows the comparison with our approach. It can be seen that, although HDRNet is capable of approximating the colours of the ground truth image, the flash highlights remain substantially

unchanged and, more important, the blurring is high to the point that the face is unrecognizable. Note that this is also a consequence of the input and output image misalignments for which we introduced the preconditioning operator explained at the beginning of Section 6.4.1. On the contrary, our result more closely matches the studio portrait preserving the high frequencies of the images such as hair and face traits. In a second experiment, we combined HDRNet with our encoding, that is, training HDRNet with the filtered images as input and the difference between the input and the filtered ground truth as target (see Section 6.2.3). Even in this experiment, we used the parameters proposed by the authors and trained the network for over 48 hours, obtaining an stable loss value of 0.0007. A few result samples of this experiment are shown in Figure 6.10. From this figure, we can notice that HDRNet performance has dramatically improved by using our encoding. However, our full approach (that is, our encoding on our network) does a better job at removing the flash artifacts (*i.e.*, highlight and shadows). We compared the two methods by using Structural Similarity Index and the Peak Signal to Noise Ratio between the prediction of networks and the target images obtained a random sample of 30 images from each sub-set of the dataset (training set, validation set, and test set). The results are reported in the Table 6.3. From this table, we can see that our approach results in higher Structural Similarity Index and Peak Signal to Noise Ratio values for all subsets. Summarizing, we can claim both that our approach outperforms HDRNet w.r.t. the specific image operator that removes the flash artifacts, and that HDRNet benefits from our encoding strategy by producing better results than its native end-to-end configuration.

### 6.4.2.2 Comparison with Pix2Pix

Another comparison of our approach and the state of the art was performed with Pix2Pix [71]. Such work is based on a particular type of generative adversarial network [57] in the conditional setting (conditional generative adversarial network) [105]. The use of such type of neural network was inves-



Fig. 6.9 A comparison between HDRNet end-to-end training and our approach.

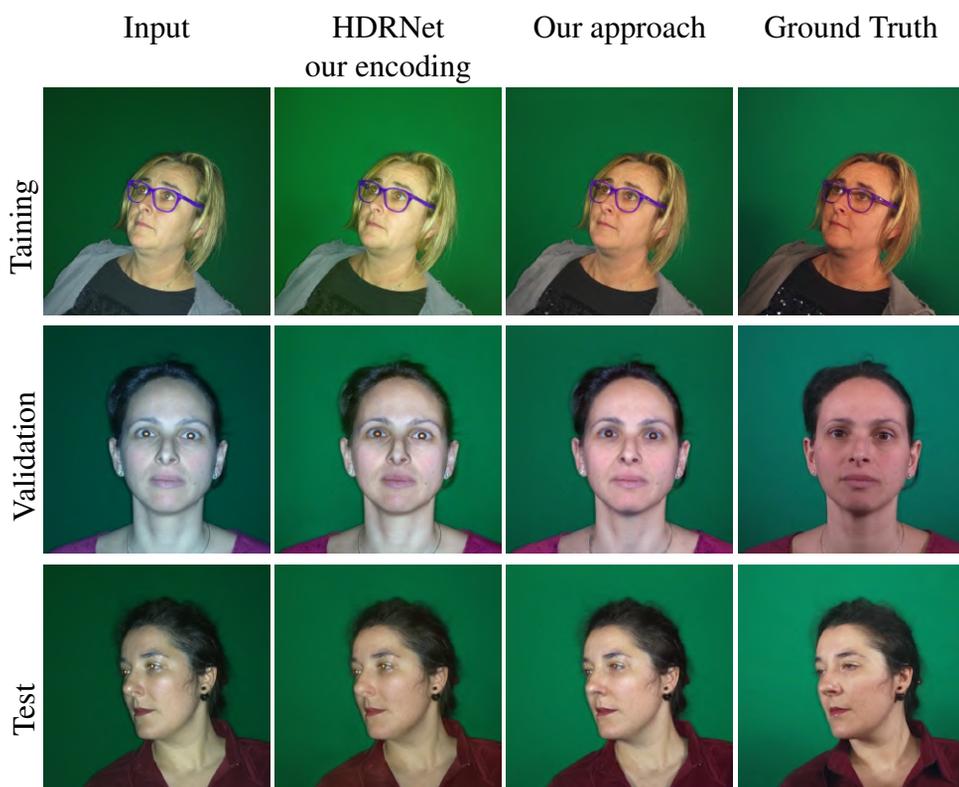


Fig. 6.10 An example of comparisons between HDRNet (combined with our problem encoding) and our approach. We show one example from the training set, the validation set and the test set, respectively.

	OUR SSIM	HDRNET SSIM	OUR PSNR	HDRNET PNSR
TRAINING	<b>91.55%</b>	73.93%	<b>24.05 dB</b>	19.71 dB
VALIDATION	<b>87.76%</b>	78.51%	<b>20.42 dB</b>	18.84 dB
TEST	<b>89.80%</b>	82.92%	<b>21.28 dB</b>	19.02 dB

Table 6.3 Comparisons on samples of the training, validation, and test sets. In particular, the Structural Similarity Index (SSIM) and Peak Signal to Noise Ratio (PSNR) values are computed on samples of 30 images extracted randomly from each dataset.

tigated as a general-purpose solution to image-to-image translation problems. The Pix2Pix authors tested their conditional generative adversarial network on several tasks such as photo generation and semantic segmentation. In the training details, they explained that the images were randomly jittered through resizing the  $256 \times 256$  (original size of images) to  $286 \times 286$  and then randomly cropped to come back to  $256 \times 256$ . Therefore, we trained Pix2Pix cGAN by using the training information provided for the *Day*  $\rightarrow$  *Night* task, which is performed by the authors. In particular, the network was trained using our dataset; *i.e.*, 4 as batch size and 80 as the number of epochs. We had to increase the number of epochs because 17 epochs, as in the case of the *Day*  $\rightarrow$  *Night* task proposed by authors of Pix2Pix, produced low quality results. The network was trained from scratch by using a Gaussian distribution to initialize the weights with 0 as mean and 0.02 as standard deviation, as suggested by the authors. We performed a mirroring and the random jitter starting from our image resolution  $512 \times 512$  and doubling the resizing to  $572 \times 572$ . As the previous comparison (see Section 6.4.2.1), we performed two experiments. In the first one, we trained Pix2Pix end-to-end using our dataset by feeding the network with input and target images. The network was trained for about 9 hours. Figure 6.11 shows the comparison with our approach. Although Pix2Pix better approximates the ground truth image

colours, it introduces notable artifacts by changing the image content significantly. Many of these artifacts can be seen on eyes and facial features. In the second experiment, we combined Pix2Pix with our encoding. We trained Pix2Pix with the filtered images as input and the difference between the input and ground truth filtered as a target; see Section 6.2.3. In this case, the network was trained for 9 hours with the same parameters used in the first case. Figure 6.12 shows some outcomes of this experiment. As before, it is possible to notice a dramatical improvement of the results obtained by combining Pix2Pix and our encoding. However, artifacts are still present on the geometry of the image (*i.e.*, facial features and around eyes), even though they are not as strong as in the end-to-end training. The results of the comparisons are visible in the Table 6.4, which reports the Structural Similarity Index and Peak Signal to Noise Ratio values. As before, these values are computed on a random sample of 30 images for each subset of the dataset (*i.e.*, training set, validation set, and test set). From this table, it can be seen that our approach obtains higher values of Structural Similarity Index and Peak Signal to Noise Ratio for all subsets. Finally, this comparisons elicits that our approach outperforms Pix2Pix w.r.t the image operator that remove the flash artifacts, and that Pix2Pix can benefit from our encoding strategy by generating high-quality results compared to end-to-end training.



Fig. 6.11 A comparison between Pix2pix end-to-end training and our approach.

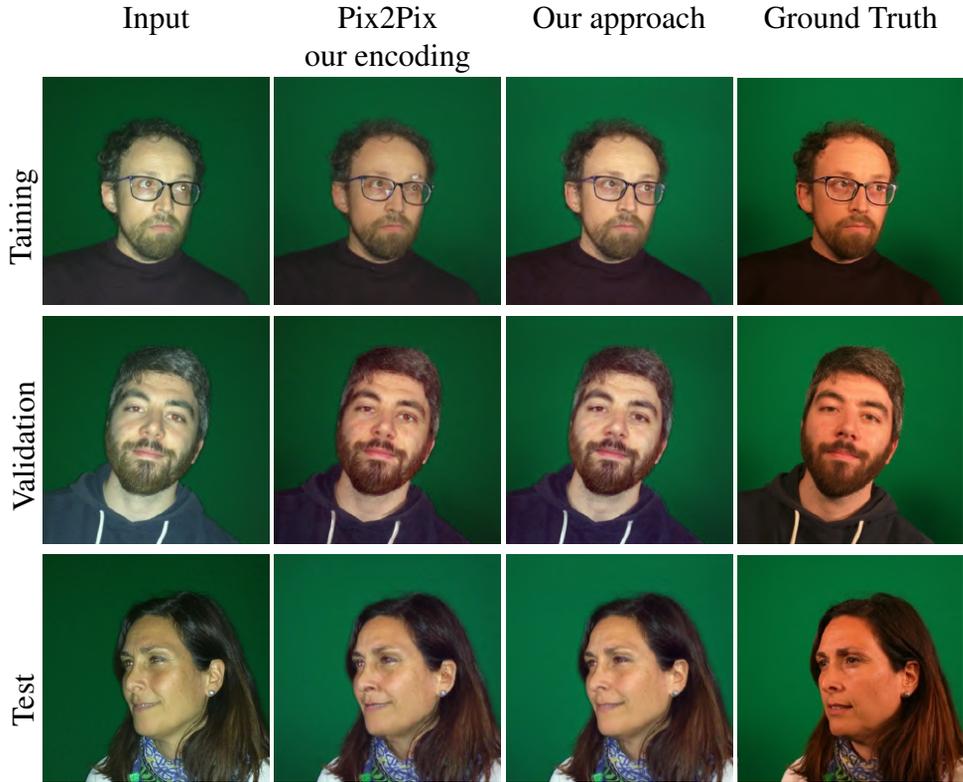


Fig. 6.12 An example of comparisons between Pix2Pix (with our problem encoding) and our approach. We show one example from the training set, the validation set and the test set, respectively.

### 6.4.3 Comparisons with Style Transfer

As our final comparison, we tested our method against the style transfer method by Shih *et al.* [146]. This method is specifically meant for portraits and based on a multi-scale local transfer approach. In our tests, we used the ground-truth image as target style to be transferred to the input image that is the ideal condition. Unfortunately, we could not try Shih *et al.*'s method on a large dataset because the generation of masks and landmarks for the original code is extremely cumbersome (*i.e.*, more than an hour per image). Therefore, we tested on a limited number of images that are displayed on Figure 6.13. As

	OUR SSIM	PIX2PIX SSIM	OUR PSNR	PIX2PIX PSNR
TRAINING	<b>90%</b>	71%	<b>24.8 dB</b>	17.82 dB
VALIDATION	<b>83.16%</b>	73.16%	<b>19.72 dB</b>	17.64 dB
TEST	<b>88.78%</b>	74.25%	<b>20.58 dB</b>	17.30 dB

Table 6.4 Comparisons on samples of the training, validation, and test sets. As before, the Structural Similarity Index (SSIM) and Peak Signal to Noise Ratio (PSNR) values are computed on samples of 30 images each, which were extracted randomly from each dataset.

can be seen from Figure 6.13, the method by Shih *et al.* can transfer colours correctly, but it fails to remove flash artifacts. Moreover, these are enhanced creating unnatural effects especially for eyes and hard shadows and lighting.

## 6.5 Discussion

We have proposed an unassisted pipeline to turn a smartphone flash selfie into a studio portrait by using a regression model based on supervised learning. We have defined a complete pipeline, starting from data collected by well-defined acquisition parameters, performing pre-processing by a bilateral filter, training the network, and finally, validating the results. We have made several comparisons using different metrics to validate our approach. Among these, we used the Structural Similarity Index that places more emphasis on the validity of the results because it measures the similarity between images in a way that is consistent with the perception of the human eye. Besides the obvious application of our method for correcting flash selfies, our results allow us to conjecture that a low-quality smartphone flash selfie contains enough information for reconstructing the actual appearance of a human face as one obtained with more uniform lighting. The most likely future work will be to

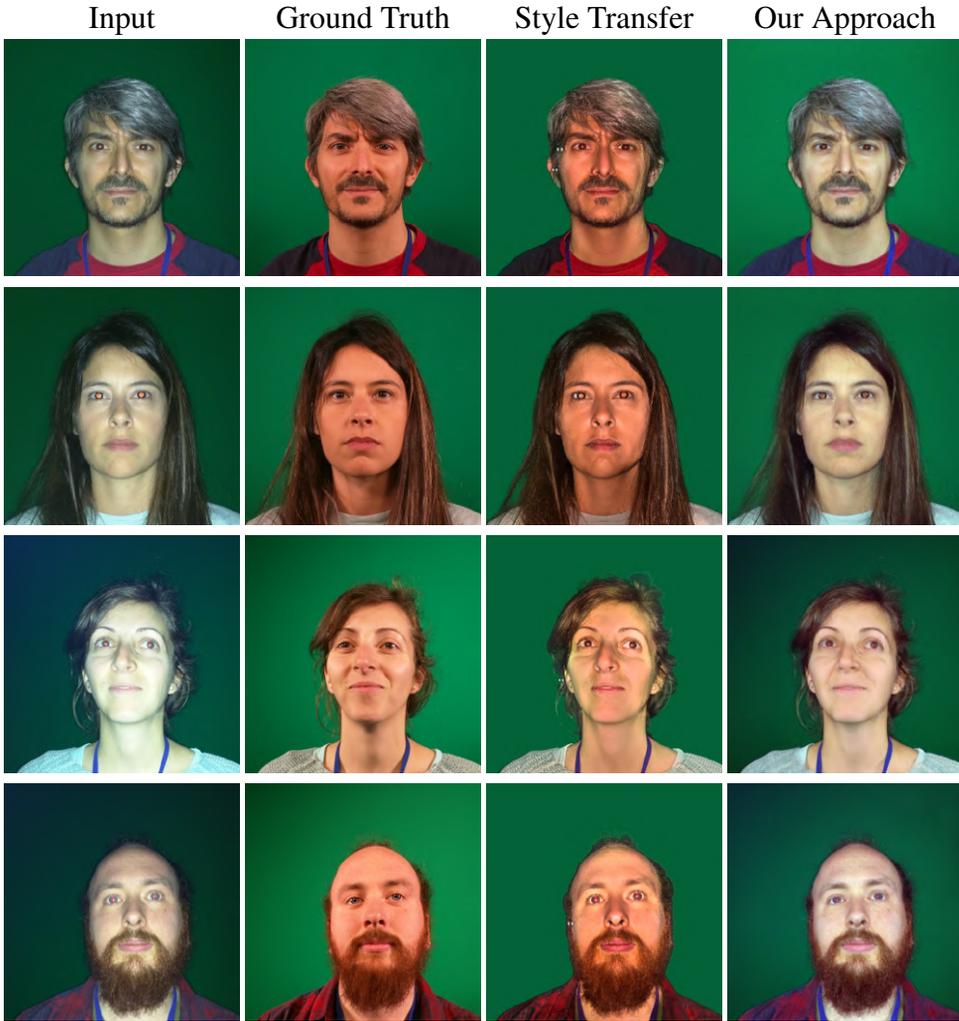


Fig. 6.13 An example of visual comparisons between the Portrait Style transfer [146] and our approach. Although there is a good match in colours several artifacts are presents such as flash hard shadows and around eyes.

widen the acquisition domain, both in terms of hardware and illumination settings and in terms of the age and ethnicity of photographed subjects. Then, we will build on our method by incorporating state-of-the-art solutions for multiple-face detection, red-eye removal, and background subtraction. Our

aim is to deploy a mobile app that can be used by any smartphone user. On the method itself, we are planning to explore the solution proposed by Larsen *et al.* [89] and to use our network as the generator component of a Generative Adversarial Network [57] where the discriminator is trained to recognize if a given image is a difference between the bilateral filtered version of a flash and no flash images.

### 6.5.1 Limitations

Thanks to the tightly bounded domain of the input, our system can provide convincing results even after being trained with a small dataset of 495 input images (prior to data augmentation). However, results would greatly improve if a more diverse training dataset was provided by relaxing some boundaries of the said domain. More specifically, all the subjects were white adults, the uniform-light setting was the same for all images, and all images were acquired with the same device.



# Chapter 7

## Deep Chroma Key

The shape and face of a person obtained to the selfie photos are typically in the foreground of the picture. To specialize the deep neural network algorithms on a well-defined domain such as the faces or shapes of people we need to decouple the background and foreground from the pictures. In this way, deep neural network learns only the intrinsic characteristics of the shapes and the faces, by focusing on the edges, colours and other details. To decouple the foreground and background we focused on the chroma key technique, by using convolutional neural network. In this context, the main advantage of deep learning is the ability to shoot a photo or video without equipment such as a green screen and numerous types of special lighting. In this way, video or photo editing times are decreased considerably, providing excellent results without the need for professional software.

### 7.1 Overview

Chroma Key is used mainly in cinema and TV, allowing two or more videos to be combined into one [124], [167]. This is achieved by signaling to a video mixer what source to use at a given time, using a background video and a foreground video with the actor moving against a uniform background (for



Fig. 7.1 Some examples of functioning and results obtained with our approach. The blue-bordered RGB sub-images, taken using an RGB camera, are passed through our network, which performs pixel classification by extracting the actor's shape from the picture. Such results can be viewed through the binary images in grey scale, that is, the green-bordered images. In the black border is shown a label overlay.

example, a green screen). The key colour is then interpreted by the video console as transparent, by requiring actors to avoid using objects and clothing that are the same colour as the background. However, chroma key presents several limits: it requires the use of separate lighting between the uniform background and the actor, and to avoid as much as possible any overlapping of shadows in the camera frame. Even under ideal lighting conditions, it is not always easy to identify image edges to a precise degree, and flat lighting often leads to the loss of thickness and three-dimensionality. Today, the most widespread way to obtain excellent results is to manage material in a completely digital process, through the use of computer graphics, and integrate it with the movements of the actor. Professional software and the knowledge and skills of industry experts are also required. In attempts to overcome these limitations, various techniques were studied, including image matting.

Image matting is the process of accurately estimating a foreground object in image-editing applications and film production. In the case of image segmentation, the goal is to segment an image into foreground and background objects by labelling the pixels. This method generates a binary image, in which a pixel belongs to either the foreground or the background. However,

the problem is that some pixels may belong to the foreground as well as to the background, and so the aim is to determine the combination of foreground and background intensity for each of these pixels. This is expressed as follows:

$$I_i = \alpha F_i + (1 - \alpha)B_i \quad \alpha_i \in [0, 1] \quad (7.1)$$

where  $I_i$  is the red, green, blue (RGB) colour at pixel  $i$ ,  $F_i$  is the foreground colour,  $B_i$  is the background colour, and  $\alpha$  is the matte estimation. In the image-matting problem, all quantities on the right-hand side of the equation (7.1) are unknown, which makes the problem ill-posed. It can be solved by adding more information to it. Many models and algorithms take an image and the corresponding *trimap* as inputs and predict the alpha matte of the image [166, 25, 61, 26]. A trimap is an image with three regions: known foreground in white, known background in black, and unknown regions in grey. A trimap allows a high degree of accuracy to be obtained, but user interaction is the most common form to generate it: In some trimap interfaces, the user manually partitions images into the three regions, requiring extensive computation, which narrows the possible applications. A useful idea is to use segmentation to obtain a good trimap initialization for image matting [144].

In this chapter, we propose an alternative method to simulate chroma key automatically and quickly, and attempt to overcome its limitations through the use of a deep convolutional neural network. It has an encoder–decoder structure composed of two subnetworks: an encoding and a decoding component, appropriately trained to perform automatic extraction of actors from images. The encoder is typical of a convolutional network and is topologically identical to the well-known VGG-16 architecture [150], but without the fully connected layers. The decoder has as many convolutional layers as those of the encoder, and converts the low-resolution encoder feature maps to full input resolution feature maps by using upsampling operations. In particular, this type of network classifies the pixels of the input images and produces an output image segmented appropriately into two classes: background and

foreground. To train our network, two specific datasets were created using the shapes of 35 different moving actors arranged on background frames, which were extracted from videos recorded in selected areas: the first dataset concerned an indoor area and the second an outdoor area. The main benefit of our approach is the possibility to shoot a video, in the same selected area, without the aid of alternative tools such as a green screen or particular types of lighting. In this way, it is possible to replace the background easily and faster for the same selected area, allowing the subject and the camera to move. Our network output can also be used to quickly generate an accurate trimap on real images with people in the foreground, by using morphological operations to derive unknown regions.

## 7.2 U-Shape Convolutional Neural Network Architecture

The proposed approach aims to extract foreground actors' shapes from the images, so that it is possible to add a different background, simulating the chroma key effect. We use a deep neural network, shown in Figure 7.2, which uses a technique called semantic segmentation. Increasingly, semantic segmentation techniques [118], [110], [145], [9], [81] are being used to divide an image into a set of non-overlapping regions. The pixels belonging to a specific region share some features such as colour. Segmentation algorithms are able to identify the areas that share similar features, but do not interpret the content. Semantic segmentation takes care of understanding the content of the image by classifying the pixels and relating them to certain classes, and is responsible for giving such content a closed and well-defined edge. Our neural network is based on the well-known SegNet architecture [6], that is, an U-shape Net, performing pixel classification through its final layer. The encoding component of the input consists of several convolutional layers [56], with batch normalization operations [70], rectified linear units as activation

functions [55], and max-pooling layers [175]. The component that performs the decoding of the encoder output is based on inverse operations such as deconvolutions [171] and unpoolings [170].

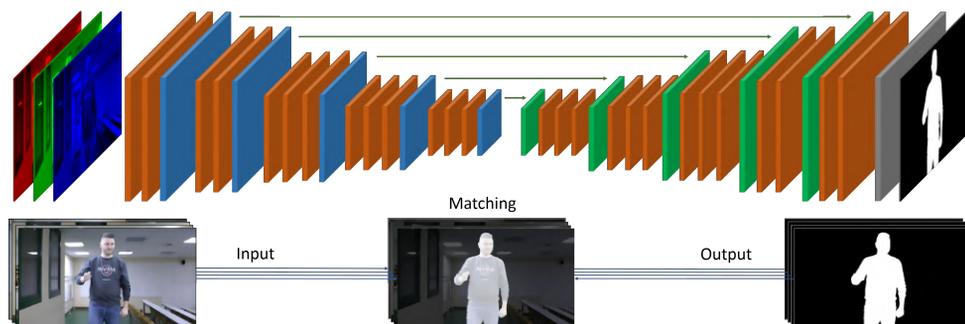


Fig. 7.2 The used U-shape Net, developed from the SegNet [6] architecture. The network structure is represented by coloured blocks: orange represents the convolutions, batch normalization, and Rectified Linear Unit operations; blue represents the pooling operations; green represents the up-sampling operations; and grey represents the pixel classification layer based on Soft-max [13] operation. The set of images to the left are the input set, while the set of images to the right are the corresponding outputs.

The used **encoder** sub-network is the same proposed in the previous Chapters 6.2.1 and 5.4.1, however, in this case it was introduced a stored operation of max-pooling indices. Such indices represent the positions of the maximum value and will be used in the decoding phase to perform the up-sampling operations.

### 7.2.1 Decoder

The remaining part of the network performs the decoding of the encoder output tensor. Such sub-network, called decoder, consists of many convolutional layers as those of the encoder. Decoding occurs through a set of operations such as deconvolution and unpooling. The unpooling operation performs the spatial up-sampling by using the stored indices in max-pooling operations. In

this way, the hardware memory is significantly reduced, avoiding the need to store all of the features map, which was obtained after the down-sampling operation in the encoding phase, and reducing the number of parameters without loss of accuracy. The features maps obtained in this phase are sparse, and they are then passed through convolutional layers to make them dense. After the convolution, a batch normalization is performed. The last layer of the decoder network classifies pixels by using Softmax as the activation function. The loss function uses cross-entropy [56], which is typically used in classification problems. Weights of the decoder network are initialized using the MSRA method, which was proposed by He *et al.* [62]. It is suitable for layers followed by a Rectified Linear Unit activation function on very deep neural networks.

### 7.3 Deep Neural Network for the Chroma Key

To train our deep neural network, we provided a pair of images as input. Each pair consists of an RGB element, which contains the shape of an actor on a background frame, and a binary image, which represents the label as shown in Figure 7.2. The background frames were extracted from a recorded video in defined areas. In this way, once the neural network was trained, it will be possible to use as input the frames of a video shot in the chosen area in which both the actor and the camera can move. The RGB element is represented through a tensor  $w \times h \times 3$ , with width  $w$ , height  $h$ , and depth 3. The label is represented through a binary matrix that uses only two values, 0 and 1, where 0 represents the background class and 1 represents the foreground class. Such distinction is useful for obtaining the chroma key effect. The background is the element to be replaced, while the foreground represents the shape of the actor, which is the element to be saved. More specifically, the label represents the ground truth for comparison with the output of the deep neural network by the training algorithm, to compute the loss value. Loss function was defined

using cross-entropy (see Appendix A), as indicated above. In particular, the error  $L(y_i, t_i)$  can be defined as follows:

$$L(y_i, t_i) = -t_i \log y_i \quad (7.2)$$

where  $y$  is the prediction and  $t$  is the target related to foreground and background classes.

Cross-entropy is useful for measuring the dissimilarity between the ground truth and the output predicted by our network. Equation (7.2) can be defined as

$$L(y_i, t_i) = -t_i^{(fg)} \log y_i^{(fg)} - t_i^{(bg)} \log y_i^{(bg)} \quad (7.3)$$

where

1.  $y^{(fg)}$  is the probability of classifying the output as foreground and  $y^{(bg)}$  is the complementary probability of classifying the output as background;
2.  $t^{(fg)}$  is the real probability of related to foreground and  $t^{(bg)}$  is the complementary real probability related to background.

Using Equations (7.2) and (7.3), the loss function  $L(y, t)$  computed on  $N$  samples and expressed according to the foreground terms, can be described as follows:

$$L(y, t) = \frac{1}{N} \sum_{i=1}^N [-t_i^{(fg)} \log y_i^{(fg)} - (1 - t_i^{(fg)}) \log(1 - y_i^{(fg)})] \quad (7.4)$$

If an image contains only one actor, in general, the background pixels will be more frequent with respect to the foreground ones, as shown in Figure 7.3. Such difference can be dangerous for the training process due to partial learning that favors the background class. Ideally, we would like each class to have the same number of observations in the training dataset to prevent a category from being underrepresented. Since the goal of the



Fig. 7.3 A label overlay of a training image. The background pixels (light blue), are more frequent with respect to the foreground pixels (red). The frequency of foreground pixels in the training dataset is 11.16%, and the frequency of background pixels is 88.84%.

network is to segment actors in the foreground, the inverse frequency is used to weigh the classes and give more importance to the foreground. Thus, during backpropagation, only the gradient from the maximally scoring instance is calculated and used for updating the weights. The inverse frequency can be expressed as follows:

$$F^{-1} = \frac{\sum_{i,j} I_{i,j}}{\sum_{i,j} I_{i,j}^{(k)}} \quad k = 0, 1 \quad (7.5)$$

where the numerator represents the sum of the pixels for each  $I$  image in the training dataset, and the denominator represents the sum of the pixels belonging to the  $k$ th class for each  $I$  image in the training dataset.  $F^{-1}$  is the inverse frequency for each class (background when  $k$  is equal to 0, foreground when  $k$  is equal to 1).

### 7.3.1 Dataset Collection

The training of our deep neural network was performed using two solutions to create the dataset. Our dataset was based on a number of requirements to make a good dataset for semantic segmentation: (i) There must be enough image pairs, composed of those to be segmented and the relative ground truth, and (ii) the labels must be as precise as possible, due to potentially critical issues such as the actor's hair is very jagged, clothes have folds, or there is little difference between background and actor.

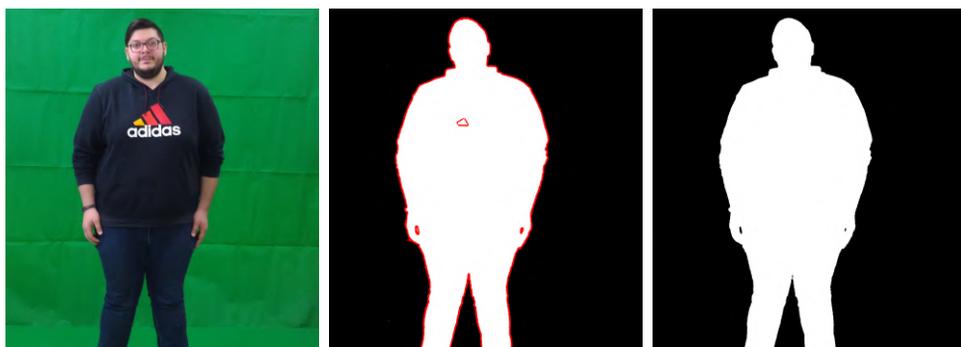


Fig. 7.4 Example of pre-processing phase. Left: The input taken using a camera and green screen. Center: The edge highlighted on the matte image obtained using Adobe After Effects. Right: The corresponding label.

The first step in creating the dataset was to recorder videos of moving actors using the green screen and a camera to obtain a large amount of data and very precise labeling. The green screen setup consisted of an opaque green drape and two lights, one of which illuminated the background, and the other illuminated the actor to remove the shadows as much as possible. The videos were recorded using a Panasonic HDC-SD800 camera with 14.2 megapixel and  $1,920 \times 1,080$  spatial resolution. The video pre-processing phase consisted of removing the green background (left image of Figure 7.4) using Adobe After Effects and extracting the shapes of the actors. Using the same software, it was also possible to obtain a matte version of each

shape (center image of Figure 7.4). This consists of a mask that defines the transparent or background areas as black and the matte or foreground areas, which contain the actor's shape, as white. The matte version contains three channels, between 0 and 255. To label (right image of Figure 7.4) each image, a binarization process was performed using a global threshold value, which creates a binary image in which all the values of the starting image over the threshold are set to 1 and all the values below it are set to 0. The threshold is a value that varies between 0 and 1, but we set this value to 0.3 because this represented a good compromise for maintaining, as much as possible, the quality of the matte version. Lower values of the threshold would have made the background areas white, and higher values would have cut out part of the shape, such as edges, hair, and clothes.

The second step was to extract background frames from a recorded video in an area that we chose. For the first solution, an indoor area was chosen. Then, the actors' shapes were arranged on the background frames to create the RGB elements, which, together with the labels, formed the first dataset solution. The obtained pairs were divided into 16,832 for the training set, 1,403 for the test set, and 900 for the validation set. The validation set was introduced to check the training performance of the network by using different data from the training set. The dataset images were reduced using  $640 \times 360$  spatial resolution for reasons related to the resources and computational times that were available.

The second solution was created to compensate for some critical issues that arose in the first one. In particular, shape overlapping on the background frame caused a clear distinction between the borders of the shape and the background compared with a real image. The network learned the shape from images created in the previous solution, and gave discrete results on real images where the variation between the background and the shape was more linear. To solve this problem, we introduced a bilateral filter and applied it to the images obtained in the previous solution. As explain in the



Fig. 7.5 Comparison between an unfiltered image and the corresponding image filtered with the bilateral filter. The filter parameters are  $w = 5$ ,  $\sigma_d = 16$ , and  $\sigma_r = 0.1$ . In the filtered image, an attenuation of the sharp edges can be seen, particularly at the neck and the hands.

previous Chapter 6.2.2, this is a nonlinear filter and is often used to reduce the noise of images while preserving the edges. In realistic photos, normally, there is no clear separation between the actor and the background. Since our photos resulted from the overlapping of background frames and actors' shapes clipped from the green screen, this separation was more evident in the left image of Figure 7.5. The bilateral filter was applied in the image in the right image of Figure 7.5, with the spatial parameter  $\sigma_d$  set to 16, the range parameter  $\sigma_r$  set to 0.1, and the dimension of a half-window of the Gaussian kernel  $w$  set to 5. Based on the knowledge acquired, we also used the second solution to create an outdoor dataset. In this case, the second step was performed by extracting the background frames from a recorded video in

an outdoor area. Data augmentation was applied to the datasets [39]. This operation generated perturbed images of the training dataset for each epoch, to avoid the problem of overfitting. The data augmentation was performed by applying several transformations on the fly in the training phase; therefore, the perturbed images were not stored, and the dataset dimension remained unchanged. We made spatial transformations mirroring right/left and rotation with a random angle between  $-30$  and  $30$  degrees.

## 7.4 Training Step

We developed, trained, and tested our neural network using MATLAB<sup>®</sup> and its toolboxes, in particular, the Neural Network Toolbox<sup>™</sup>. We used Nvidia GPU GTX 1080Ti, which has a Pascal architecture, 3584 CUDA core, 11 GB GDDR5X as a frame buffer and 11 Gbps speed memory. The training was performed by selecting Adam [84] (see Appendix A) as the optimization algorithm. It was demonstrated empirically that this algorithm achieves good results, in a short time, applied to large models and datasets. The initial learning rate was set to  $10^{-5}$ , and the validation set was introduced to check the level of generalization of the neural network for each epoch. The number of epochs was set to 40, but the training was stopped after 19 epochs since no further improvements were noted: neither an increase in accuracy nor a reduction in error. At the end of training, the accuracy was fixed at around 99.8%, and loss in training was around 0.01. Accuracy in the validation stage did not decrease with respect to that of training, reaching approximately 99.79%, and loss during validation fell throughout the training, reaching a final value of 0.008.

## 7.5 Results

This section begins with an overview of the methods of evaluation of the results, through analysis of the proposed neural network output. The remainder of the section will describe in detail the results obtained and their evaluation through analysis of the two dataset solutions, by using the described methods.

### 7.5.1 Evaluation methods

Evaluation of the results after several training steps was performed through two types of test: A first test was performed by using the test set, which contained the images to be segmented and the labels to be used as a comparison with the output obtained from the network; the other test was performed on real and unlabeled photos. Semantic segmentation quality was evaluated through three metrics: Accuracy, Intersection over Union (IoU) [52], and Mean Boundary F1 Score (BF) [140]. The Accuracy metric measures the amount of correctly classified pixels with respect to the total amount of pixels. It can represent the ratio of correctly classified pixels to total pixels, regardless of class (Global Accuracy), the ratio of correctly classified pixels in each class to total pixels, averaged over all classes (Mean Accuracy), or the ratio of correctly classified pixels in each class to the total number of pixels belonging to that class according to the ground truth. This last definition can be expressed as

$$Accuracy = \frac{TP}{TP + FN} \quad (7.6)$$

where  $TP$  indicates the true positive and  $FN$  the false negative. Intersection over Union is a statistical measure of accuracy that penalizes false positives. This parameter shows the quality of the pixels correctly classified with respect to the total amount of pixels assigned to a certain class by ground truth and by the network output. Intersection over Union can be expressed by

the following formula:

$$IoU = \frac{TP}{TP + FP + FN} \quad (7.7)$$

where  $FP$  indicates the false positive. Intersection over Union can also be computed as an average value (Mean Intersection over Union).

Boundary F1 Score is a measure of the accuracy used in the statistical analysis and is calculated for each class. The measure takes into account the precision and recovery of the test, where the precision is the number of true positives divided by the number of all positive results, and the recovery is the number of true positives divided by the number of all the tests that should have been positive (*i.e.*, the sum of true positives and false negatives). This parameter is defined as the harmonic mean of precision  $p$  and recovery  $r$ .

$$score = 2 \cdot \frac{p \cdot r}{p + r} \quad (7.8)$$

In addition to these metrics, a further method of viewing the performance data from the tested network is the normalized confusion matrix [152]. It returns a representation of the accuracy of the classification. Each column represents the predicted values, and each row represents the real values. Each element  $(i, j)$  is given from the amount of pixels that belong to the true class  $i$ , but associated with the predicted class  $j$ . A normalization is performed by dividing by the total number of predicted pixels in  $j$ .

## 7.5.2 Unfiltered Dataset Results

After training with the unfiltered dataset, the performances of our neural network were evaluated with respect to the test dataset and the real photos. Table 7.1 shows the metrics aggregated over the test dataset, and Table 7.2 shows the metrics for each class. By observing the normalized confusion

GLOBAL ACCURACY	MEAN ACC.	MEAN IOU	MEAN BF SCO.
0.99737	0.99828	0.99132	0.9942

Table 7.1 The metric values of the whole test dataset for the network trained with the unfiltered dataset: Global Accuracy; Mean Accuracy; Mean Intersection over Union; Mean Boundary F1 Score.

matrix (Table 7.3), it is possible to obtain rapid feedback on the performed test.

	ACCURACY	INT. OVER UNION	MEAN BF SCO.
FOREGROUND	0.99974	0.98587	0.99168
BACKGROUND	0.99683	0.99677	0.99672

Table 7.2 The metrics obtained by considering each class with respect to the unfiltered test dataset.

	PRED. FOREGROUND CLASS	PRED. BACKGROUND CLASS
TRUE FOREGROUND CLASS	99.97	0.02647
TRUE BACKGROUND CLASS	0.3166	99.68

Table 7.3 Results of the test carried out on the network after training with the unfiltered dataset. A normalized confusion matrix returns a representation of the accuracy of the classification. Each column represents the predicted values, while each row shows the real values. Each element  $(i, j)$  is given by the count of the pixels belonging to the real class  $i$  but associated with the predicted class  $j$ . Finally, a normalization is performed by dividing by the total number of predicted pixels of the  $j$  class.

Figure 7.6 shows the results related to the test dataset. This dataset was created as the training dataset, with actors' shapes arranged on background

frames. These images highlight the best and the worst case for Mean Accuracy (left images) and Mean Intersection over Union (right images).

Other tests were performed on real photos 7.7, which were taken in the same indoor area where the background videos used to create the dataset were shot. These photos were resized appropriately to fit the aspect ratio to the size of the neural network input. Note that the subject in these photos is partially undivided by the network. This is because, since the network was trained on a dataset in which the actors presented a well-defined edge with respect to the background, there is an uncertainty in real cases above the boundary zones. This involves an error in the classification of foreground pixels.

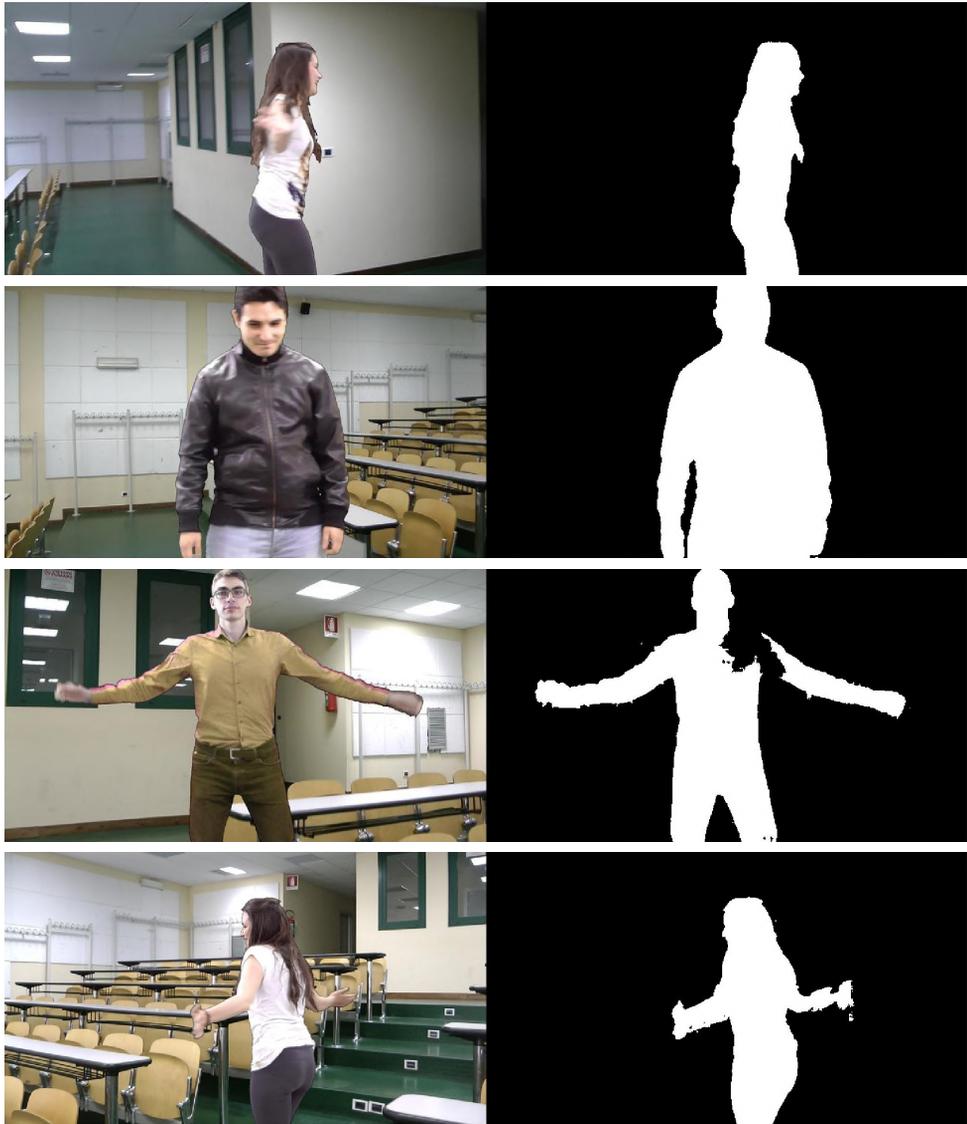


Fig. 7.6 Results related to the test dataset after training using the unfiltered dataset. The test set was created as the training set and contains unfiltered images. The first image shows the best result for Mean Accuracy (99.9%), and the second image shows the best result for Mean Intersection over Union (99.6%). A classification error related to foreground can be seen in the third image, which shows the worst result for Mean Accuracy (96%), and a classification error related to background can be seen in the last image, which shows the worst result for Mean Intersection over Union (96.6%).

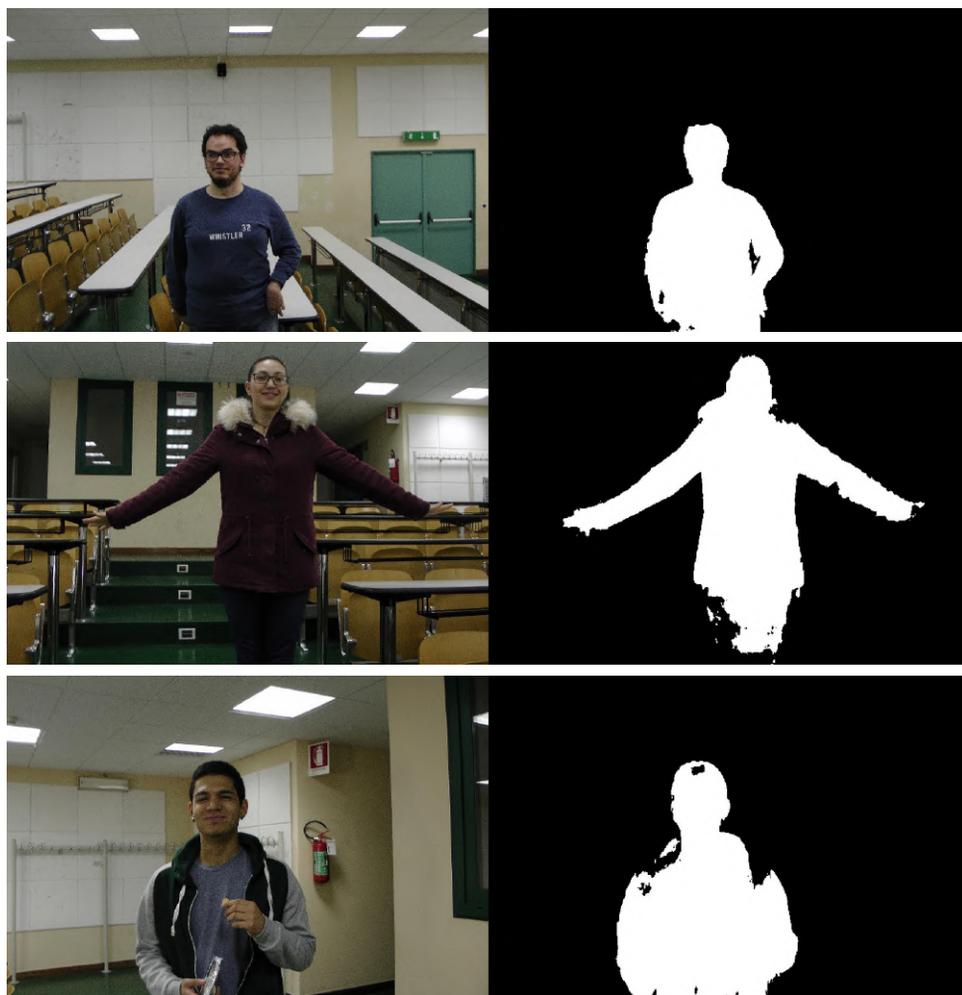


Fig. 7.7 Results of real images, taken directly with a camera, querying the network after training with the unfiltered dataset. A classification error related to foreground pixels can be seen in these three images. In particular, parts of the actor in the top and center images have a colour similar to the background.

### 7.5.3 Filtered Dataset Indoor Results

The next training step was performed on the filtered dataset with background frames of an indoor area, and the neural network performances were evaluated with respect to the test set and the real photos. Table 7.4 shows the metrics aggregated over the test dataset, Table 7.5 shows the metrics for each class, and Table 7.6 shows the normalized confusion matrix.

GLOBAL ACCURACY	MEAN ACC.	MEAN IOU	MEAN BF SCO.
0.99778	0.99829	0.99267	0.9955

Table 7.4 The metric values of the whole test dataset for the network trained with our filtered dataset for an indoor area.

	ACCURACY	INT. OVER UNION	MEAN BF SCO.
FOREGROUND	0.99909	0.98805	0.99351
BACKGROUND	0.99749	0.99728	0.9975

Table 7.5 The metrics obtained by considering each class with respect to the filtered test dataset for an indoor area.

	PRED. FOREGROUND CLASS	PRED. BACKGROUND CLASS
TRUE FOREGROUND CLASS	99.91	0.09141
TRUE BACKGROUND CLASS	0.2514	99.75

Table 7.6 The confusion matrix related to the test carried out on the network after training with the filtered dataset for an indoor area.

Figure 7.8 shows the neural network output that was obtained on the test set in a similar way to the results of the first solution. No improvements

were found in the results for the dataset test, unlike the real cases, as can be observed by comparing Figures 7.7 and 7.9. By testing the real photos, we found a marked improvement in the classification of foreground pixels. In particular, in the top image of Figure 7.9, the problem occurs on the left hand of the shape because the colour of the hand is similar to the background colour. In contrast, in the center image of Figure 7.9, there is excellent classification of pixels, due to a clear difference in colour between the background and the foreground.



Fig. 7.8 Results of the filtered test dataset for the chosen indoor area. The first image shows the best results for Mean Accuracy (99.9%), and the second image shows the best results for Intersection over Union (99.6%). A classification error related to foreground pixels can be seen in the third image, which shows the worst result for Mean Accuracy (96.4%), and a classification error related to background pixels can be seen in the last image, which shows the worst result for Mean Intersection over Union (94%).



Fig. 7.9 Results of real images, taken directly with a camera, querying the network after training with the filtered dataset related to the chosen indoor area. An improvement in the classification of foreground pixels can be seen, especially in the bottom image.

### 7.5.4 Filtered Dataset Outdoor Results

The final training step was performed on the filtered dataset with background frames of an outdoor area, and the neural network performances were evaluated as in the previous case. Table 7.7 shows the metrics aggregated over the test dataset, Table 7.8 shows the metrics for each class, and Table 7.9 shows the normalized confusion matrix.

GLOBAL ACCURACY	MEAN ACC.	MEAN IOU	MEAN BF SCO.
0.99771	0.99843	0.99245	0.99602

Table 7.7 The metric values of the whole test dataset for the network trained with our filtered dataset for the chosen outdoor area.

	ACCURACY	INT. OVER UNION	MEAN BF SCO.
FOREGROUND	0.99957	0.9877	0.99432
BACKGROUND	0.99729	0.9972	0.99771

Table 7.8 The metrics obtained by considering each class with respect to the filtered test dataset for the chosen outdoor area.

	PRED. FOREGROUND CLASS	PRED. BACKGROUND CLASS
TRUE FOREGROUND CLASS	99.96	0.04345
TRUE BACKGROUND CLASS	0.2705	99.73

Table 7.9 The confusion matrix related to the test carried out on the network after training with the filtered dataset for the chosen outdoor area.

Figure 7.10 shows the neural network output that was obtained on the test set in a similar way to the results of the previous case. Figure 7.11 shows some of the results obtained from the real photos taken in the same outdoor area selected for the network training.

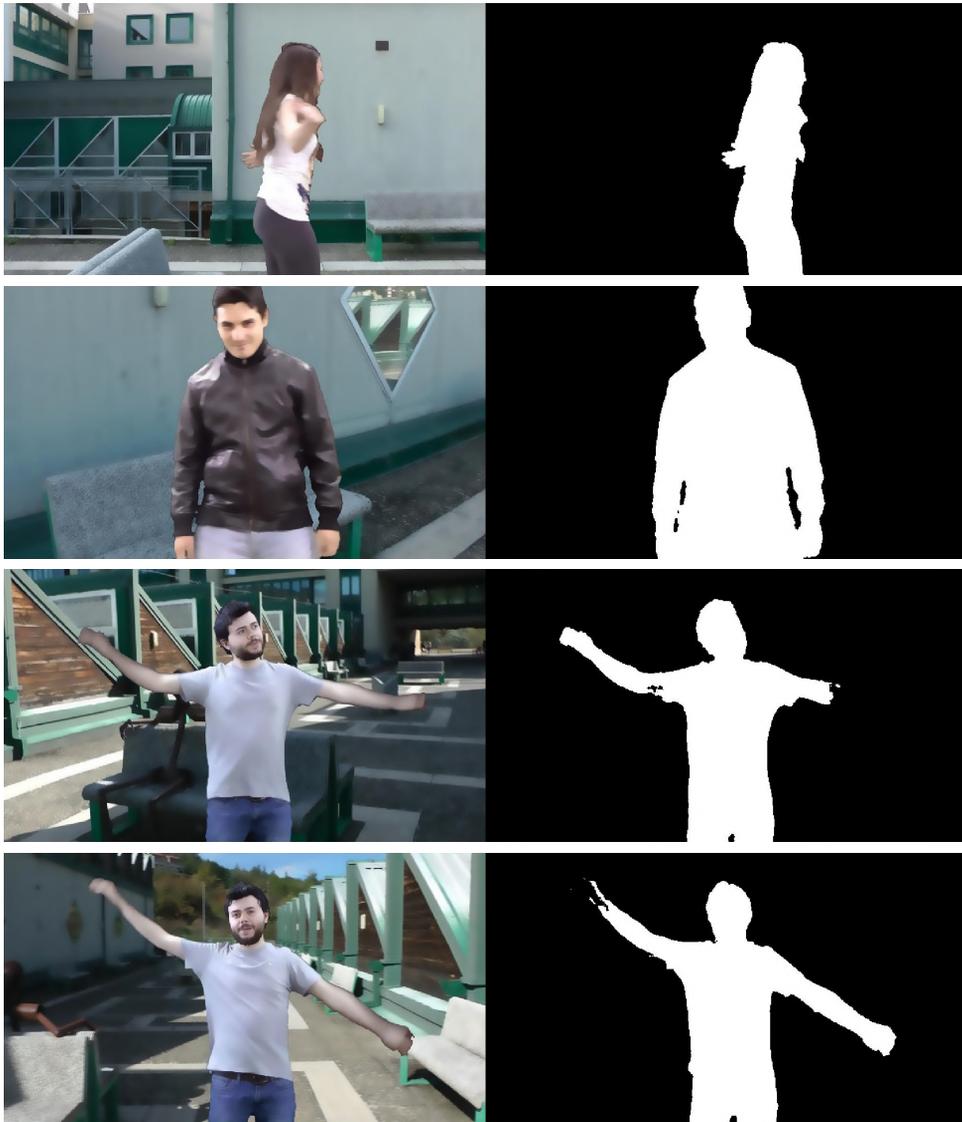


Fig. 7.10 Results of the filtered test dataset for the chosen outdoor area. The first image shows the best results for Mean Accuracy (99.9%), and the second image shows the best results for Intersection over Union (99.66%). The third image shows the worst result for Mean Accuracy (96.6%), and the last image shows the worst result for Mean Intersection over Union (97.8%). In these last two images, a classification error related to foreground pixels can be seen.

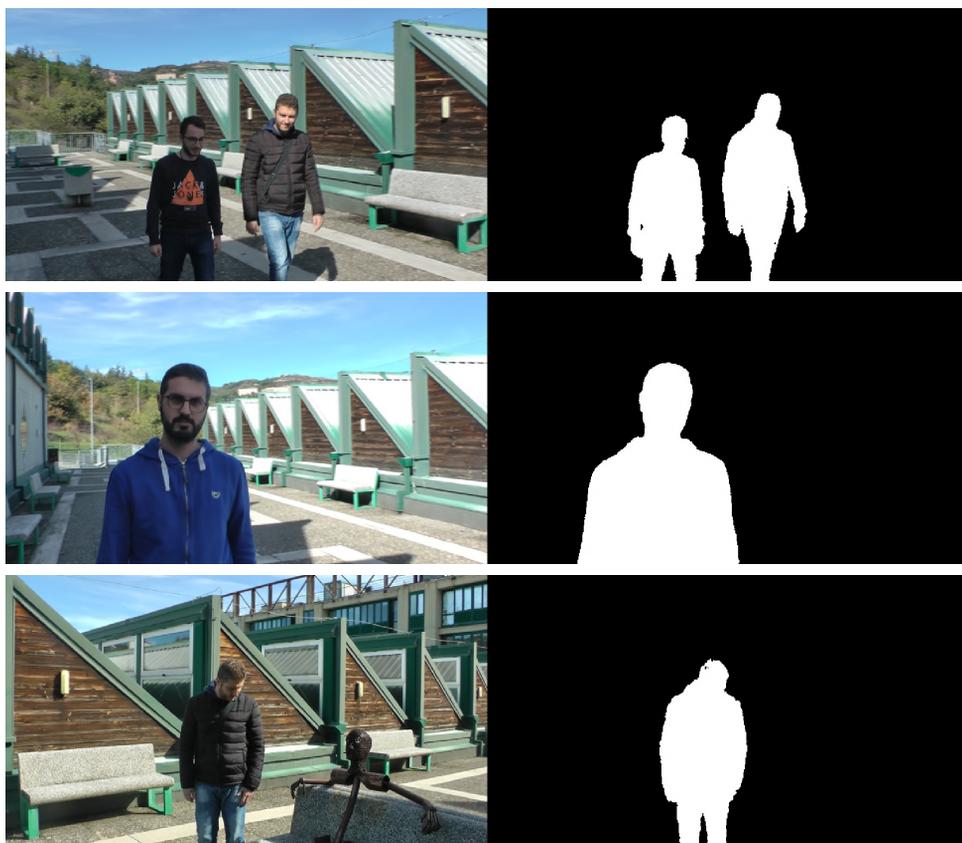


Fig. 7.11 Results of real images, taken directly with a camera, querying the network after training with the filtered dataset for the chosen outdoor area. In these three cases, there are no notable classification errors. The system is efficient even when there are more people in the image, as shown in the top image.

### 7.5.5 Improvements

In some cases, the obtained result has small imperfections that can be corrected, for example, isolated pixels wrongly classified as foreground and/or small holes. Then, it is possible to perform a post-processing phase. In the first case, an area opening operation can be used. This is a morphological

operation that removes all connected components that have fewer than a given number of pixels from a binary image. In the second case, a closing operation can be used, which is a dilation followed by an erosion, using the same structuring element for both operations. Figure 7.12 shows the advantages of the described operations. The number of pixels was set to 30 for the area opening operation, and a disk-shaped structuring element with a radius of 2 pixels was used for the closing operation.

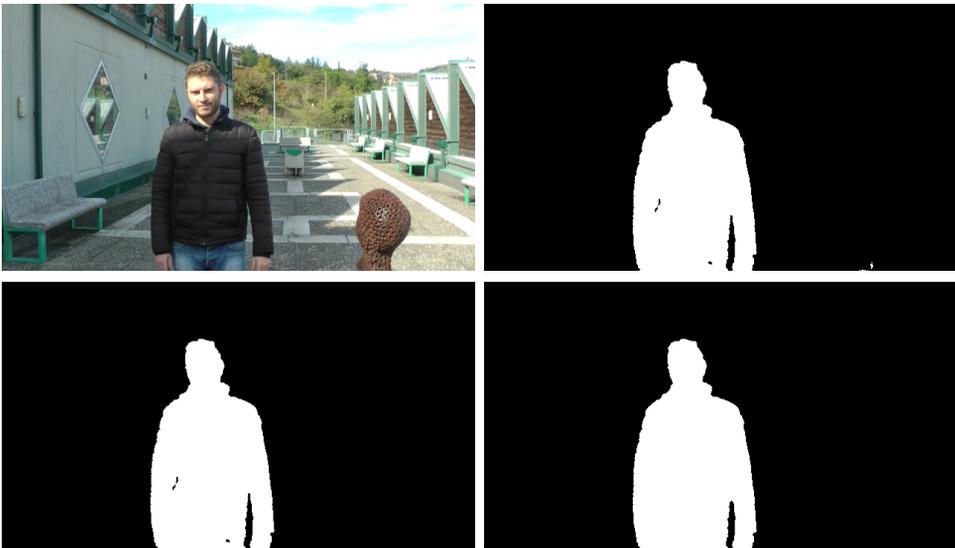


Fig. 7.12 Improvements in the neural network result. Top: The original image passed as input to the network (left) and the obtained result (right). Bottom: The output of the area opening operation, which removed all connected components with fewer than 30 pixels (left), and the output of the closing operation considering a disk-shaped structuring element with a radius of 2 pixels (right).

## 7.6 Comparisons and Applications

Starting with the results obtained through our approach, we made a visual comparison using an Adobe Photoshop tool called Select Subject. Select Subject was developed through Adobe Sensei, a framework that uses artificial intelligence to support image-processing tasks to enhance and simplify complex user image-editing operations. According to Adobe, this tool is useful for quickly selecting prominent subjects in pictures. Select Subject provides a basic method to select the subject and allows the selection to be refined through other tools or user actions. Adobe Sensei is an advanced machine-learning technology trained to identify a wide variety of objects in an image, such as people, animals, vehicles, and toys. We tested a subset of our filtered test dataset for an indoor area. By comparing our network with the Select Subject tool, we obtained the metrics aggregated over the dataset (Table 7.10), the metrics for each class (Table 7.11), and the normalized confusion matrix (Table 7.12).

	GLO. ACC.	ME. ACC.	ME. IOU	ME. BF SCO.
PHOTOSHOP	0.98942	0.99242	0.99	0.9248
OUR NN	0.99799	0.9983	0.99	0.99481

Table 7.10 The first row shows the metric values obtained using the results of Photoshop Select Subject on a subset of our filtered test dataset for the indoor area. The second row shows the results of our network using the same test subset.

Figures 7.13 and 7.14 show the best and worst cases for Mean Accuracy and Intersection over Union, comparing the Adobe Photoshop results with ours. It is evident that our network provides better results. Figure 7.15 shows some of the results obtained with Select Subject from real images, taken directly with a camera. In comparison with the images in Figure 7.7, greater accuracy was obtained with our approach, particularly in the subject's

		ACC.	IoU	MEAN BF SCORE
PHOTOSHOP	Foreground	0.99715	0.94511	0.893
	Background	0.98769	0.98706	0.9566
OUR NN	Foreground	0.99878	0.99754	0.99252
	Background	0.99781	0.99754	0.9971

Table 7.11 The metrics for each class obtained using the results of Photoshop Select Subject and our results on a subset of our filtered test dataset for the indoor area. A deterioration when using Select Subject compared with the results of our approach can be observed.

		PREDICTED FOREGROUND CLASS	PREDICTED BACKGROUND CLASS
PHOTOSHOP	True Foreground Class	99.72	0.2845
	True Background Class	1.231	98.77
OUR NN	True Foreground Class	99.88	0.1216
	True Background Class	0.2185	99.78

Table 7.12 The confusion matrix related to the test carried out using the results of Photoshop Select Subject and our results on a subset of our filtered test dataset for the indoor area. A deterioration when using Select Subject compared with the results of our approach can be observed.

contours in the center image of Figure 7.15. These tests show the effectiveness of our work.



Fig. 7.13 From top to bottom: The images show the results obtained with Select Subject on a subset of our filtered test dataset for the indoor area: The first and second images show the best results for Mean Accuracy (99.5%) and Intersection over Union (97.9%); in the third image, which shows the worst result for Mean Accuracy (98%), a classification error related to foreground pixels can be seen; in the last image, which shows the worst result for Mean Intersection over Union (94%), a classification error related to background pixels can be seen.

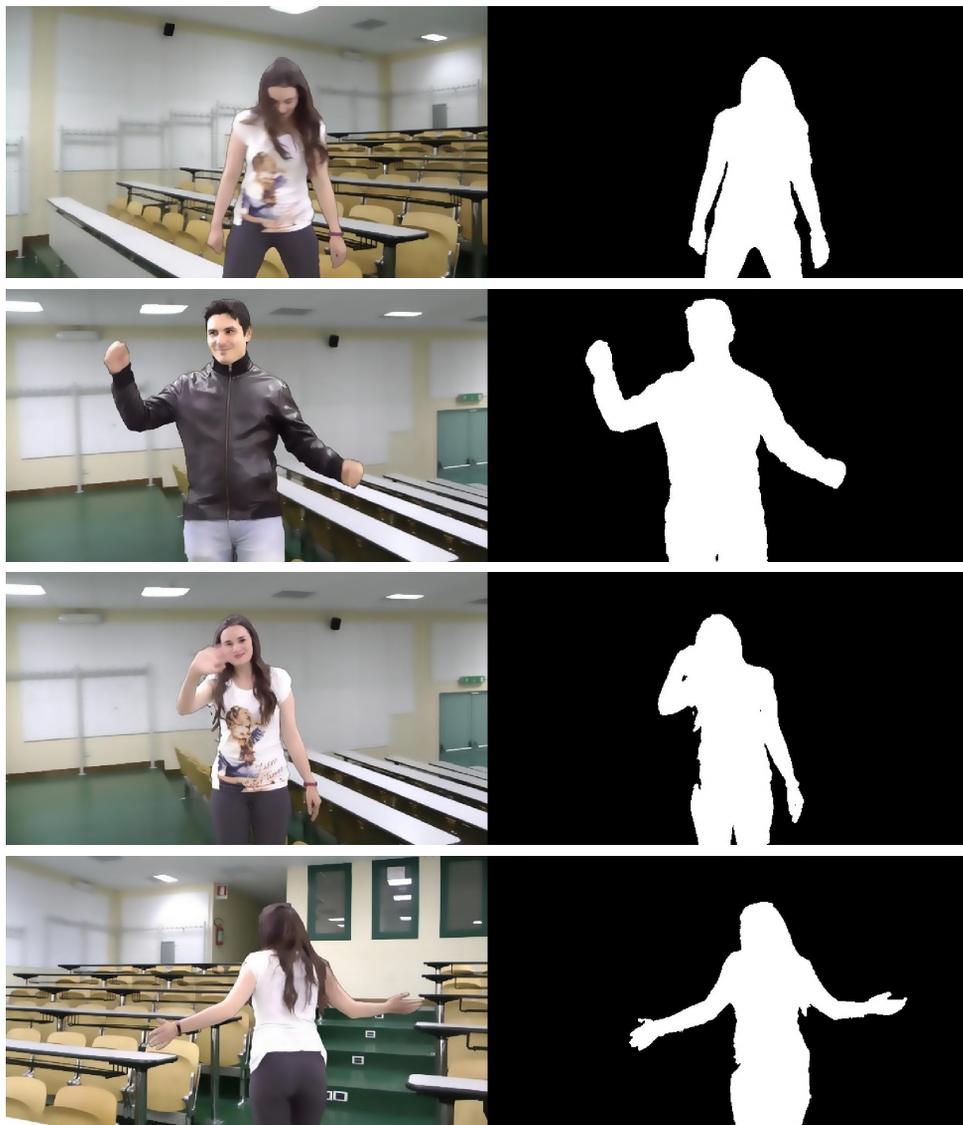


Fig. 7.14 The images show our results on the same test subset by using Adobe PhotoSp Select Subject 7.13: Better outcomes were obtained. In particular, the first image is our best result for Mean Accuracy (99.9%), the second is our best result for Mean Intersection over Union (99.5%), the third is our worst result for Mean Accuracy (99.4%), and the last is our worst result for Mean Intersection over Union (99.46%).

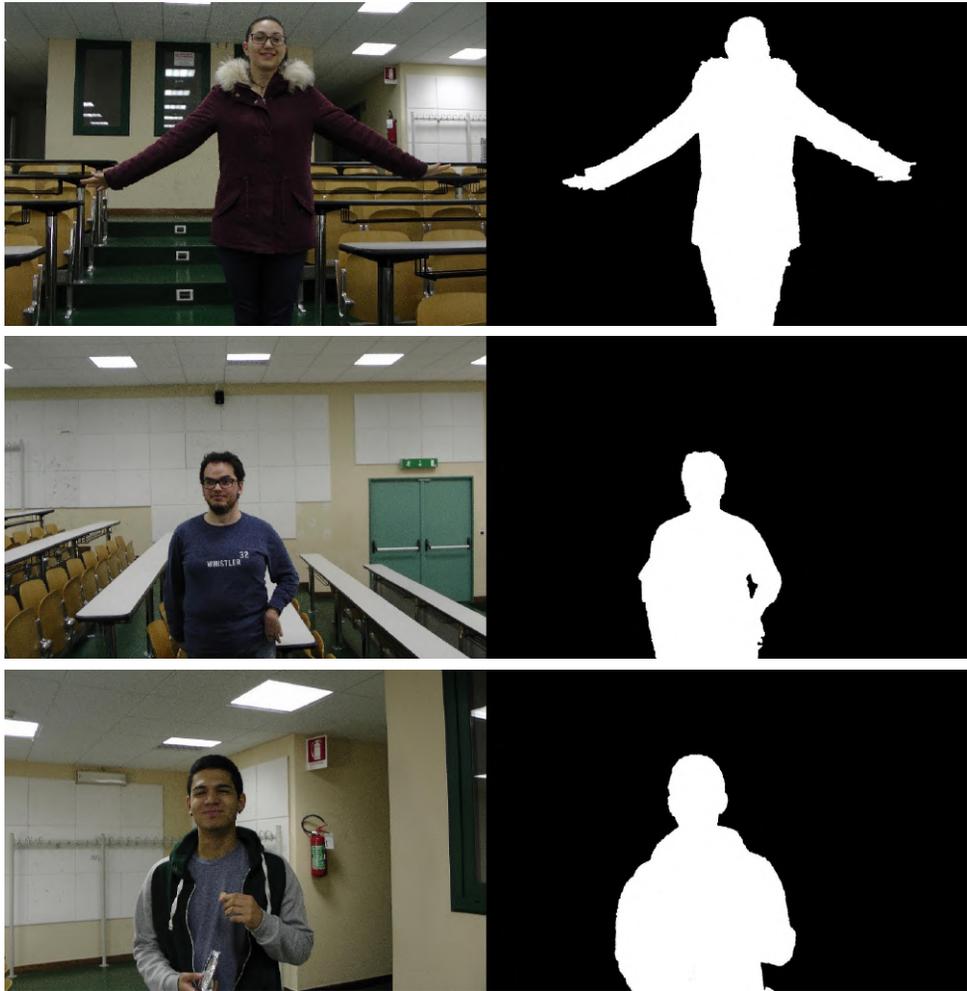


Fig. 7.15 Results obtained with Select Subject on real images taken directly with a camera. Comparing these images with the same three in Figure 7.9, less accuracy can be observed, particularly in the subject's contours in the center image.

The output of our neural network can also be used to provide a good trimap for image-matting problems, as shown in Figure 7.16. The trimap (top right image) was generated using morphological operations: The unknown region, in grey, was obtained with erosion and dilation using a disk-shaped

structuring element with a radius of 20 pixels. The bottom images show the KNN (K-Nearest Neighbors) [25] and the KL-divergence (Kullback-Leibler) [77] output using our trimap.

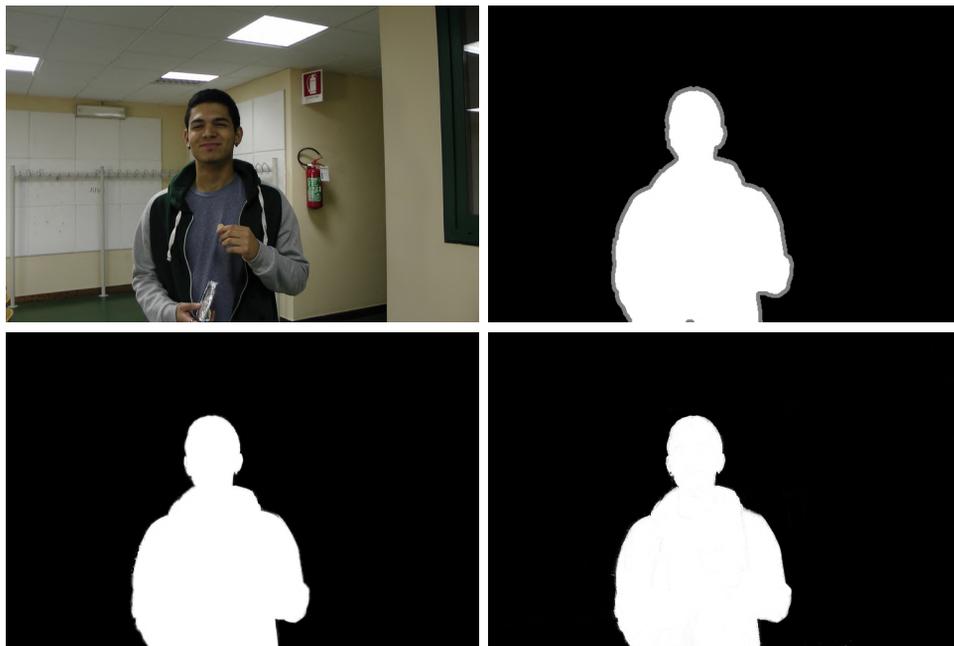


Fig. 7.16 The trimap generated through our network output. The top left image shows the image given as input to the network. The neural network output was used to obtain the trimap (top right image), in which the unknown region was obtained through morphological operations. The image and the trimap can be used as the input for well-known algorithms, such as KNN, resulting in the image shown in the bottom left image, and KL-divergence, resulting in the image shown in the bottom right image.

## 7.7 Discussion

Our aim was to simulate the chroma key effect through deep-learning approaches. The chroma key effect is usually obtained by using a uniform green

or blue background and by replacing it in the post-processing phase with a different background. Our goal was to provide an alternative to this classical method.

### 7.7.1 Advantages

The main advantage is the possibility of shooting a video with a person in the foreground without the use of a green screen or particular types of lighting, to obtain the chroma key effect in an automatic, simple, and fast way. Two case studies were defined, and the semantic segmentation problem was tackled by classifying pixels and assigning them a specific meaning using the U-shape Net. Our system identifies pixels belonging to the background class and those belonging to the foreground class, where the foreground is represented by the shape of a person in the image. The system also proved to be efficient in the semantic segmentation of images composed of more shapes, as shown in the top image of Figure 7.11. A critical issue that was tackled is the creation of a good training dataset, one that is representative of the system and allows the network to generalize appropriately, avoiding underfitting or overfitting. A green screen was used as a determinant to create datasets quickly and to easily extract labels by performing a binarization of the matte versions. These input images were built by blending the extracted shape with a selected background and applying a nonlinear filter to smooth the edges of the shape and make the images as uniform and real as possible. Most state-of-the-art matting algorithms [25], [61], [26] require human intervention to generate the alpha matte from the input image. The most common form of user interaction is the trimap interface, where the user manually partitions the image into foreground, background, and unknown regions [134]. Natural-image matting is usually problematic. To make it tractable, user-specified strokes or a trimap are used to sample foreground and background colours. Our approach uses semantic segmentation that takes only an RGB image as input and generates a binary output quickly, often with a very accurate boundary. Our estimated

segmentation result could be used as a good initial trimap for image matting. The unknown regions could be created with morphological operations such as erosion and dilation of foreground regions.

### **7.7.2 Limitations and Directions**

Despite the numerous advantages, there are some questions concerning the limitations of our approach. The main problem is due to its use of deep learning, which requires a large amount of data to obtain good results. This data must be as precise as possible to allow the neural network to obtain a good level of generalization for a certain behaviour. Another problem concerns the precision of the segmentation when the foreground colour is chromatically similar to the background colour, which makes it difficult to detect the edges of a shape. Other limitations are due to reflections within a scene (for example, glasses, windows) and motion blur (partially solvable by recording slow-motion video, from 120 fps) because the neural network fails to classify pixels in such areas. We intend to continue the development of our approach by increasing the dataset to obtain more accurate classification and by performing the segmentation on high-resolution images [174], [99]. Furthermore, our work is related to the chosen background area. A possible future development is to generalize the dataset in such a way that the network is independent of the type of background. Other future works include the resolution of the limitations exposed in this section.

# Chapter 8

## Overall Discussion

This chapter tries to answer on the unresolved or partially resolved research questions. Are delineated also the implications from the results and future research directions.

### 8.1 Generalization and Training Problems

The capacity of the deep learning algorithms to perform optimal and consistent predictions on unknown data is called generalization [56]. Such ability is particularly highlighted in the supervised learning paradigm because the deep neural network has to perform valid prediction on data that not part of the **training set** itself, but belong to the same domain. For example, if we want to classify cats from images, we have to train our neural network by using a large dataset of images of cats. After the training, we expect that the network knows how to recognize a cat also from unseen images such as a cat in other positions, a cat with different colours and other images which contain a cat. The unseen images with a cat, belong to the same domain of the training dataset but not directly to the dataset itself (**test set** is an example). A good response to such images establishes a good level of generalization of the neural network.

Let our neural network has to build a function  $f(x)$ , which is able to perform a prediction  $y$  starting from an input  $x$ . The risk of a particular function  $R(f_n)$  on all possible values of  $y$  and  $x$  is:

$$R(f_n) = \int_{X \times Y} L(f_n(x), y) P(x, y) dx dy, \quad (8.1)$$

where  $L$  is the loss function,  $P(x, y)$  is the true distribution and  $n$  is a number of samples. Since the true probability is unknown, is possible to compute the empirical risk of  $f_n$  as follow:

$$R_s(f_n) = \frac{1}{n} = \sum_i L(f_n(x_i), y_i). \quad (8.2)$$

The generalization error [79], also called **test error** is given by

$$\text{generalization error} = R(f_n) - R_s(f_n). \quad (8.3)$$

If this difference tends to zero, it means that our neural network generalizes well. Generalization level can be measured through the test set after the training process, however, the trend of the generalization level of the network in the training phase can be identified through the **validation set**. The use of training-validation paradigm is usually adopted in the deep learning context, where the algorithms have several **hyperparameters** (see Appendix A), which influence the behaviour of the algorithm itself. Although more of the hyperparameters can be manual tuned, such operation requires more experience from the user. For this reason is possible to optimize the hyperparameters (*i.e.*, learning rate, number of hidden units, *etc.*) in an automatic way, by trying to reduce an objective function, called also validation function or loss validation. The main problem due to the neural network inability of generalization is called overfitting, discussed in Section 8.1.2.

### 8.1.1 Rules of Dataset definition

Although there are no strictly specific rules of a good dataset construction, in image processing tasks there are several strategies to improve the performance of algorithms based on deep learning. Some of these strategies have been investigated and extended in this thesis. As explained in Section 3.3 we have collected a coins training dataset that consists of 80 images taken with a camera at a distance of 10cm. Such number of images is very low to train a deep neural network if we consider large dataset for machine learning research such as ImageNet<sup>1</sup> [86], CIFAR-10<sup>2</sup>, PASCAL VOC<sup>3</sup> [48] and others, because a small dataset can easily bring a deep neural network to overfitting. However, in our work described in Chapter 3, we considered only 80 images and we tried to exploit a data augmentation concept, through which the deep neural network can improve its generalization level. In particular, as mentioned by I. Goodfellow *et al.* [56], the geometric operations such as random translation, flipping, and rotation, but also through another type of transformations such as the random colours perturbation and another type of image distortions, can be beneficial for deep neural network, especially for the classifiers. In fact, our coins images were rotated by 60° around their center, mirrored, and then rotated again by 60°, to obtain a dataset of 1,040 images for each coin and a total of 8,320 coin images.

In the image processing tasks, there are several techniques used to improve the deep neural network performance and a very used used pre-processing step is called **contrast normalization**. Since the contrast amount represents a clear variation in the images, it can be removed dividing the image by the standard deviation of its pixels. In fact, the image contrast is defined through the ratio between the brightest and the darkest pixels and can be quantified

---

<sup>1</sup> ImageNet: a very large dataset, consisting of more than 14 million images and more than 20,000 categories.

<sup>2</sup> CIFAR-10: a dataset consisting of 60,000 images and 10 different classes.

<sup>3</sup> PASCAL VOC: a dataset consisting of 500,000 images which include also the Flickr images.

through the standard deviation of the image pixels. In particular, let the pixels of the image  $I$  is normalized in the range  $[0, 1]$  and let the  $w$  and  $h$  represent the image dimensions, the contrast of the images can be defined through the following formula:

$$C = \sqrt{\frac{1}{wh} \sum_i \sum_j (I_{i,j} - \mathbb{E}[I])^2}, \quad (8.4)$$

where  $\mathbb{E}[I]$  is the mean of the pixels value.

In Chapter 6 we use a variant of global contrast normalization, whose goal is to centralize the data distribution so that each image gives the same contribution to the training and does not have more or less important than the others. We perform such operation by subtracting the mean of the pixels of an image, from the image itself. The mean subtraction is a typical operation performed in the classification problem, in fact, K. Simonyan and A. Zisserman [150] remove the mean of the entire training dataset from each image of all dataset to reduce the variation of the amount of contrast. In a more generic global contrast normalization, this operation is performed together with the division for standard deviation, in order to rescale the image so that the standard deviation of its pixels is equal to a certain constant  $s$  [56]. In the global contrast normalization, to avoid divisions by zero due to zero-contrast images, a small positive regularization parameter called  $\lambda$  is introduced and a pixel  $p'_{i,j}$  of regularized contrast normalization is given from the formula

$$p'_{i,j} = s \frac{p_{i,j} - \mathbb{E}[I]}{\max \left\{ \varepsilon, \sqrt{\lambda + \frac{1}{wh} \sum_i \sum_j (I_{i,j} - \mathbb{E}[I])^2} \right\}}. \quad (8.5)$$

The last rule of dataset creation investigated by us concerning the influence of the high and low frequencies of the images. In particular, as shown in Chapter 6, the used deep neural network for image processing tasks suffers from the blur problem, especially when the images and ground truth are slightly

misaligned. To avoid this problem we introduce the **bilateral filter** [8], [157], which, by smoothing the images, preserves the low frequencies. To explore more depth this detail, we refer to Section 6.2.3. The variation of high and low frequencies was a problem also investigated in our work on chroma key (Section 7). In such work, the deep neural network learned well the edges of the shapes from the artificial images, created by overlapping actor's shape extracted using Adobe After Effect with the selected background. In this way, the deep neural network did not well recognize the edges in real images. For this reason, we use, the bilateral filter to smooth the artificial images. Also in this case, to explore more depth details, we refer to Section 7.3.1.

### 8.1.2 Overfitting and Underfitting

When a deep neural network is unable to generalize, the network can be found in an overfitting situation. In this situation, the neural network will not be able to recognize examples that it has never seen. To recognize the overfitting is possible to analyze the validation error, as can be seen in Figure 8.1.

There are several approaches to avoid the overfitting, *e.g.*, data augmentation described in the previous section can reduce this problem when a dataset with a few samples is available. A high number of well-distributed samples in the training set can reduce the risk of overfitting, however, there are other techniques to avoid such problem that involve the regularization. A first simple form of regularization is called early stopping. Such method consists of updating the weights at each iteration to better fit the training data. When the validation error starts to increase, the training must be stopped [127]. There are other techniques based on the use of the validation set such as the cross-validation [85], but this type of techniques are useful when dataset with limited samples is available. We do not use such type of techniques because our datasets are widely large and well generalized. Other types of techniques, used to avoid the overfitting and increase the generalization level, keeping training error low are based on regularization. Such techniques do not involve

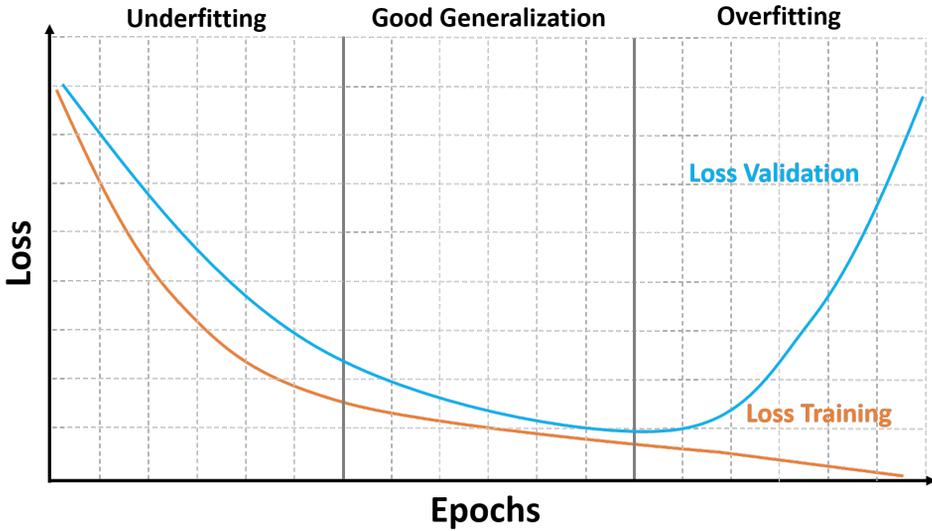


Fig. 8.1 The left block represents a state of underfitting; The central block represents a good model and the last block represents a state of overfitting.

the dataset rules and the most used are  $L^2$  [13] also called *weight decay* or *ridge regression*,  $L^1$  regularization and Dropout [151]. In general, we did not use  $L^1$  regularization and Dropout techniques because the problem of overfitting did not appear in our work.

However starting from the AlexNet [86] that we used for the coin recognition system proposed in Chapter 3, the weight decay was introduced to reduce the model training error. More generally, such method consists in adding of penalty called regularizer to the loss function:

$$L_{L^2}(x, y, w_t) = L(x, y, w_t) + \frac{\lambda}{2} w_t^2, \tag{8.6}$$

where  $L(x, y, w_t)$  is the loss function,  $L_{L^2}(x, y, w_t)$  is the regularized loss function and  $\lambda$  is a term that controls the weight of the penalty term contribution. After calculating the gradient the function becomes

$$\nabla_w L_{L^2}(x, y, w_t) = \nabla_w L(x, y, w_t) + \lambda w_t. \quad (8.7)$$

In the backpropagation phase the weights update are performed as

$$w_{t+1} = w_t - \varepsilon (\nabla_w L(x, y, w_t) + \lambda w_t), \quad (8.8)$$

where  $\varepsilon$  is the learning rate and  $\lambda w_t$  is the penalty term. The update of the weights can be rewritten as

$$w_{t+1} = (1 - \varepsilon \lambda) w_t - \varepsilon (\nabla_w L(x, y, w_t)). \quad (8.9)$$

When a learning algorithm is not able to fit well to the data, the problem opposed to overfitting occurs, called underfitting. This situation can be captured through the analysis of the loss training as can be seen in the left block of the Figure 8.1. In this case, the loss training is not minimized and continues to remain very high during the training process. The underfitting problem is typically due to the very simple model, where the data have low variance and high bias.

### 8.1.3 Vanishing gradient problem

Very deep neural network can suffer from a problem called "vanishing gradient", when the optimization algorithms are gradient-based. In the backpropagation phase, the partial derivative of the error function is used to proportionally update the weights of the network. As the weights are updated by moving the network levels backwards, the gradient will become smaller and smaller, until in some case it disappears. In this way, the first neurons of the network may not be trained or suffer a trivial influence from the gradient.

Typically this problem is due to the presence of saturated activation functions such as *tanh* or *sigmoid*, which have gradients in  $(0, 1)$  range. When the gradient of the activation functions takes high values, the problem

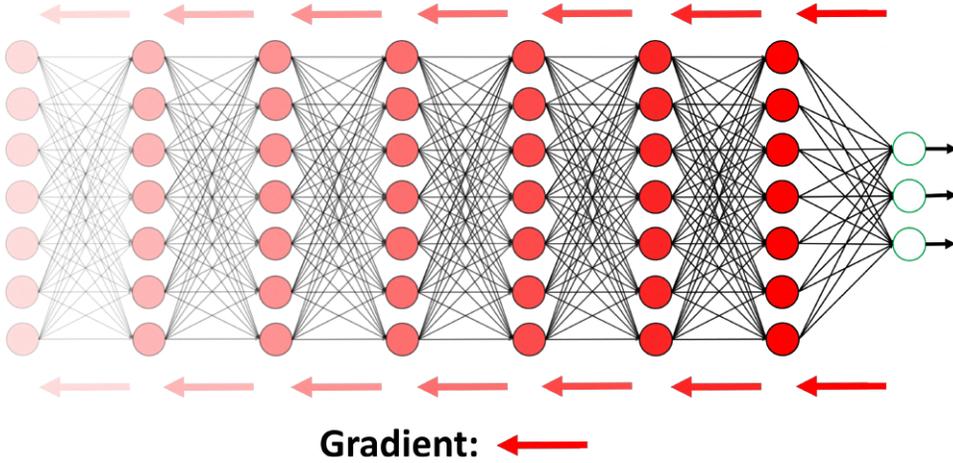


Fig. 8.2 The image shows a visual interpretation of the vanishing gradient problem. As can be seen, as the gradient is backpropagated through the network, its effect will be increasingly shallow.

is called "exploding gradient". There are several methods used to reduce such problem: in our work we used techniques such as batch normalization (see Equation (5.1)) and residual learning net (see Section 5.4.2). A simple solution for the vanishing gradient problem is to use the rectified function (see Figure 2.4), that we used in the works described in Chapters 3, 5, 6, and 7. Finally, another technique, similar to the Residual Learning Net, was proposed by G. Huang *et al.* [67], called DenseNet. Starting from the residual learning intuition in which, from the use of the skip-connection, the non-linear transformation is bypassed through  $x_l = H_l(x_{l-1}) + x_{l-1}$ , where  $x_l$  is the output of the  $l^{th}$  layer and  $H_l$  is the non-linear transformation of the layer, DenseNet proposes a different connectivity pattern. In particular, the idea is to remove the addition and replacing it with a concatenation. In this way the  $l^{th}$  layer can receive feature maps of all previous layers  $x_l = H_l(x_0, x_1, x_2, \dots, x_{l-1})$  are not lost in the sum.

We want to point out that batch normalization, residual learning, and DenseNet have as a positive side effect the regularization of the model and, consequently, the reduction of the generalization error and avoidance of overfitting problem.

## 8.2 Future Research Directions

From the obtained results we can assert that the application of the supervised learning paradigm in computer vision tasks, such as image-to-image translation, image segmentation and image classification, is a generally good solution. However, recently new deep learning techniques are affirming and, among the most important, there are certainly the GANs [57]. Although this type of techniques born from the unsupervised learning paradigm, in this section we briefly discuss their functioning.

### 8.2.1 Generative Adversarial Networks

A generative adversarial network is a type of network whose functioning is based on the competition between two neural networks. A first neural network, called **generator**, takes as input the random noise and generates image samples. The other network, called **discriminator**, takes as input two set of images: the first one coming from the training set and the second one coming from the generator sampling, as shown in the Figure 8.3.

These two networks are continually challenging each other: the generator has to learn how to produce an always more realistic image, while the discriminator has to become increasingly better to distinguish real data from those generated. The training is performed concurrently and the expectation is that the competition will lead the generated samples to be more and more indistinguishable from the real samples. Although it seems there may be similarities with reinforcement learning paradigm, instead in the generative adversarial network is present a backpropagation phase, in which the gradient

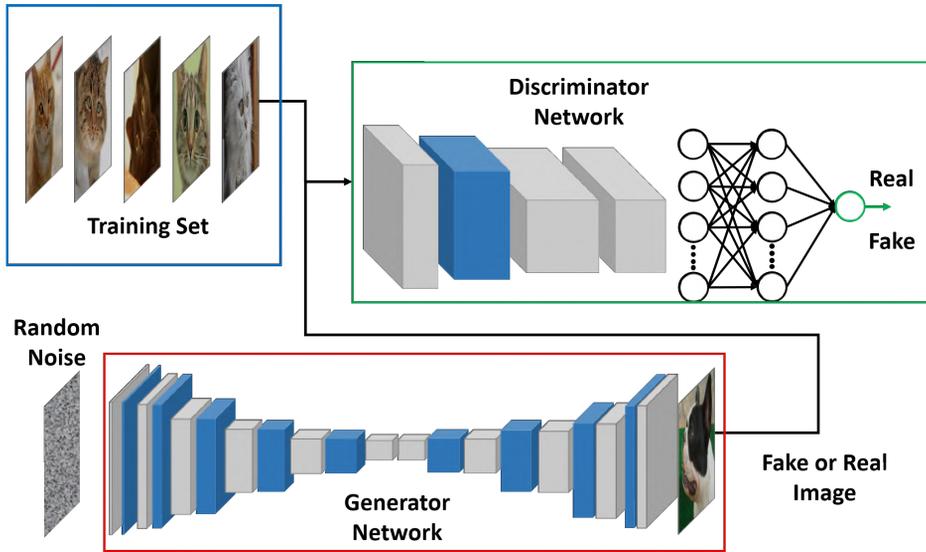


Fig. 8.3 The image shows the classical architecture of the Generative Adversarial Network. As can be seen, the generator takes the random noise as input and the discriminator takes samples from the training set and the output of the generator. The output of the discriminator is the truthfulness of the sample.

information is also propagated from the discriminator towards the generator, in order to allow the generator to adapt its parameters and produce data that can mislead the discriminator.

In the last period, the study of the generative adversarial network has led to several its variants. Among the most important, there are Deep Convolutional Generative Adversarial Networks [129], whose main characteristic is only based on convolutions without pooling and unpooling, and Conditional Generative Adversarial Networks [105], whose main characteristic is to be based on a semi-supervised type of learning. In particular, in this case, the generator takes both the random noise and the label as input, obtaining a considerable improvement of the generated samples.

# Chapter 9

## Related Work

This chapter shows the state of the art relative to the deep learning field, its application in computer graphics and image processing tasks related to our studies. Moreover, some works related to the addressed problems will be presented.

### 9.1 Deep Learning

As explained in Chapter 1, deep learning [90] is a framework inspired by biology that enables computational models (made of nonlinear projection) to learn high-level representation from data, through back-propagating errors with respect to the model parameters. The most important advantage of deep learning is it avoids feature engineering that is automatically inferred from raw data. One class of deep learning architecture of particular interest is the convolutional neural network class, namely, deep learning models that mimic how the visual cortex works. Krizhevsky *et al.*'s work [86] on Convolutional Neural Networks has sparked a renaissance in the field of deep learning. Furthermore, this work has pushed forward the state of the art in many imaging tasks. This was made possible because of three key factors: the increasing computing power of modern graphics processing units (GPUs), the avail-

ability of huge datasets, and novel and powerful algorithms and frameworks for Convolutional Neural Networks. To improve their learning invariance, Convolutional Neural Networks are typically trained on augmented data [39]. As shown in the previous chapters, data augmentation (see Section 8.1.1) was found to be essential in our experimentations.

## 9.2 Coin Classification

Starting from our proposed coin recognition approach, the research trend is focused on several studies. Among the different coin recognition methods, Bremananth [15] proposes a system that focuses on numerals rather than images on the front or back. The idea is to capture an image of the coin and extract the numeral using a technique based on pattern matching. The approach uses several techniques, such as statistical colour, threshold method, Gabor filtering, and backpropagation networks for accurate recognition of these numbers. Kim [82] proposes an automatic recognition method for Imperial Roman coins using convolutional neural network. The study implements a hierarchical framework that employs convolutional neural network models for coin classification tasks. The aim is to find a landmark on coin images. An optimization problem is formulated as the selection of a set of the elements of the coin that represents the class. The selected parts are considered to be elements that can be used to discriminate the coin. These landmarks can be critical for the analysis of the features of coins for numismatic experts. Modi [108] developed an artificial convolutional neural network based on the automatic recognition of Indian coins. The system is able to recognize both sides of coins. Information is extracted from images using methods such as the Hough Transformation, Pattern Averaging, etc. The extracted information is used as input to train a neural network.

The main difference between these methods and our proposed approach (see Section 3) is that we consider the entire coin, and classify it according to

membership classes. The information is extracted directly from the coin and is passed to the neural network in the training phase. In this way, we obtain a classification model that can predict the classification of other coins.

### 9.3 Real-Time Ambient Occlusion

As real-time global illumination is a hot topic in computer graphics research, and an impressive number of related works were published, we restrict the scope to techniques most closely related to our own: ambient occlusion suitable to real-time rendering that operates in image space (also called screen-space ambient occlusion). The technique called "screen-space ambient occlusion" appears for the first time in [106]. In its CryEngine, CryTek implements this technique, which works by sampling the surroundings of a pixel and, on the basis of the z-buffer, performs depth comparisons. Sample positions are distributed in a sphere around the pixel, and some randomness is introduced by reflecting position vectors on a random plane passing through the sphere origin. The occlusion factor depends only on the depth difference between the sampled points and the current point. This, combined with the simple distribution of samples (around a sphere and not a hemisphere), causes some over-darkening: even flat, non-occluded areas result in some samples being considered as occluders. Even in only this category of algorithms, a variety of approaches exist. Some sources attempt to correlate, compare, and evaluate such techniques. The interested reader may like to consult [1] and [58], which are two recent theses that agree that the Alchemy algorithm [103] was state of the art at that time. More recently, the state of the art in research is Mara *et al.* [100], which was demonstrated to be an order of magnitude faster and more correct.

## 9.4 Global Illumination and Neural Networks

Neural networks were used in real-time global illumination in [133]. In that work, the authors model a radiance regression function for fast rendering of global illumination in scenes with dynamic local light sources. The regression function is implemented by using a multi-layer acyclic feed-forward neural network, which provides a close functional approximation of the indirect illumination and can be efficiently evaluated at run time. A similar work based on a feed-forward neural network is [65]. In this work, the authors train a neural network to learn a mapping from the depth and normals surrounding the pixel to the ambient occlusion. In this thesis, we use only normals surrounding the pixel as the input of the neural network. The idea is to train the neural network to learn the shape of the surface to guess the occlusion value for a given surface point (see Section 4).

## 9.5 Day Time to Night Time Translation

Image processing is a very hot topic in deep learning research, in fact, in the last years were developed more and more approaches. Zezhou *et al.* [28] developed a method for colorizing images from a grey-scale input image. The problem is formulated as a regression problem and deep neural network is used to solve it. They used a huge training dataset, which contains several types of images such as trees, animals, buildings, sea and mountains.

The used neural network has a number of input neurons equal to the size of descriptive features extracted from every single pixel in the grey-scale image and two neurons in the output layer, which output are the  $U$  and  $V$  channels representing colour value in the chrominance space.

J. Lee and S. Lee[95] developed a system that allows applying image illumination in order to simulate that the same picture was taken repeatedly in different hours of the day. They use an approach based on the adversarial network[57] to define a convolutional neural network as a function that

changes the input image colour in the one it should have in the selected hour of the day. A function is trained to separate the samples day-time and night-time images. Because of the approach is based on the adversarial network, they use two Convolutional Neural Networks (see Section 8.2.1): The generator is composed by five convolutional layers, the discriminator one is composed by six convolutional layers. They use features like pooling, rectified linear unit and dropout layers. Last layers placed at the end of discriminator are fully connected; the activation function for each neuron is a sigmoid. T. Shih *et al.* [147] developed a system that creates an image in a particular hour of the day using a single input image. Their approach is data-driven and allows to automatically create a plausible-looking picture. This approach is based on the use of a database of time-lapse movies of several scenes. Such movies provide information on the colour variation of scenes during the day. This method transfers colour from movies with similar scenes to the input picture. Used movies cover several outdoor scenes like a cityscape, buildings, and street views. The transfer performs a matching of the input image with time-lapse movies that contains a similar image and is used a Markov Random Field based algorithm to find a match. Subsequently, they used an example based on a locally affine model that transfers the local variation colour between two frames of the movie in two different times of the input image.

The main difference between these methods and our presented work (Section 5) is that we consider only two classes of images, daytime and night-time. We define the problem as a regression problem: the model was trained using as input the image with applied artificial illumination and as output the same image with the desired real-world illumination. The key of our work is the perfect matching between images content representing input and output, in such a way the neural network was trained only to change colours scale of the images and not to translate or deform them.

## 9.6 Computational Photography and Deep Learning trend

In this section, we review works related to our and photography per se, without a focus on acquisition for computer vision/graphics tasks (*e.g.*, colour acquisition for three-dimensional (3D) meshes).

### 9.6.1 Flash Photography

In two contemporary works, Petschnigg *et al.* [123] and Eisemann and Durand [43] proposed the idea of using a flash photograph with low ISO (*i.e.*, low noise) to transfer the ambiance of the available lighting into a non-flash photograph of the same subjects/scene, to reduce noise. This is because non-flash photographs taken in dark environments suffer from high noise to avoid blur. Other works [4] have developed this idea further by removing over/under-illumination at a given flash intensity, reflections, highlights, and attenuation over depth.

### 9.6.2 Deep Learning and Computational Photography

Recently, Convolutional Neural Networks were applied to numerous computational photography tasks [149], [143], [68], [27][66], [176].

To improve the editing of selfies, Shen *et al.* [144] extended the FCN-8s framework [97] to automatically segment portraits. This allows a user to automatically edit/augment portraits. For example, users can change the background, stylize the selfie, enhance the depth-of-field, etc. Zhang *et al.* [173] proposed an end-to-end learning approach for single-image reflection separation with perceptual losses and a customized exclusion loss. Their method can be used to remove or reduce unwanted reflections in pictures. Eilertsen *et al.* [42] introduced a U-Net architecture for expanding the dynamic range of low dynamic range images to obtain high dynamic range images. Similarly,

Chen *et al.* [23] shown that U-Nets can be used successfully to debayer images captured at low-light conditions and high ISO, which typically exhibit noise. They extensively studied different approaches on real-world low light with real noise. For example, they tested a variety of architectures, loss functions (e.g., L1 (least absolute deviations), L2 (least square errors), and the structural similarity index), and colour inputs. Aksoy *et al.* [5] presented large-scale collection of pairs of images with ambient light and flashlight of the same scene. The images were obtained with the aid of casual photographers by using their smartphone cameras, and consequently, the dataset covers a wide variety of scenes. The dataset was provided in order to be studied in future works for high-level tasks such as semantic segmentation or depth estimation. Unlike their dataset, whose objective is to provide matching between two images under uncontrolled lighting, our dataset aims to change the lighting scheme by turning from flash lighting to a controlled photograph studio light.

To the best of our knowledge, there have not been proposed approaches for removing flash artifacts such as hard shadows and highlights from selfies as the work presented in this thesis (see Section 6).

## 9.7 Deep Chroma Key

In this section, we summarize the current state of the art used to obtain the chroma key effect by means of methods based on deep learning and Convolutional Neural Networks useful for semantic segmentation.

### 9.7.1 Chroma Key Effect

There are several techniques to obtain the chroma key effect [160]. Among the most recent ones, Yamashita *et al.* [167, 3] propose a method to extract a region using the chroma key effect with a two-tone checker pattern background. Their method prevents foreground objects that are the same colour as the background from becoming transparent, based on estimation of the

alpha channel. Bagiwa *et al.* [7] propose a method to identify the chroma key background in digital video using blurring artifact as a feature. They used the Wiener filter to extract the blurring artifact from the background of a video and the foreground objects, and chroma key detection was obtained through computing and analysis of the blurring cross-correlation between the video background and foreground blocks. Vidal [159] proposes a technique that provides immersive feedback between an actor and the background image, based on polarized non-directional reflections, retro-reflective screens, and polarization filters. The polarized filter allows projections of high-contrast colour images, while the camera captures the chroma key background, improving the immersive effect and maintaining good and uniform chromatic lighting. Gvili *et al.* [59] provide a solution to the problem of video keying in a natural environment, based on segmentation of foreground objects from background objects by using their relative distance from the observer through a depth camera, so that the colours of objects can be disregarded for this purpose.

### 9.7.2 Deep Learning and Image Semantic Segmentation

Deep learning is often used in the semantic segmentation field. For example, D'Avino *et al.* [35] propose a method to detect forged videos by using an autoencoder-based architecture and recurrent neural networks in a "long and short-term memory model". Long *et al.* [97] deal with semantic segmentation by using fully convolutional neural networks, which are trained end to end, pixel to pixel, and take an image of arbitrary size as input and produce an output of the same dimensions. Both the training and the reading are performed on complete images through computing steps such as feed-forward and backpropagation. Hariharan *et al.* [60] propose a method of detecting all instances of specified categories in each image and mark the pixels that belong to each category, in a method called simultaneous detection and segmentation, based on Convolutional Neural Networks and support vector

machines. Chroma key and semantic segmentation are strongly related to each other and, in the deep learning field, they are a very hot topic [16, 177]. Badrinarayanan *et al.* [6] use an U-shape Net structure to perform semantic segmentation, and our proposal is based on training a similar deep neural network structure through the creation of an ad-hoc dataset to obtain the chroma key effect and specific training and test parameters.

Today, the most widespread way to obtain excellent results with the chroma key effect is to use computer graphics to integrate a new background with the movements of the subject. However, these require professional software and expert knowledge, and is costly in terms of resources and computation times. The purpose of our work (see Section 7), therefore, is to try to overcome the limits of the classical approach to the chroma key effect and to simulate it in an easy and fast way on a recorded video in a defined area when both the camera and the actor are moving. After the network is trained, it will take just a few moments to obtain the output and use it to replace the background automatically.



# Chapter 10

## Conclusion

In this thesis, we have presented the results of supervised learning applications on several 2D and 3D computer graphics problems. In the first part, we focused on key concepts of deep learning investigating them through different proposed applications. This thesis aims to demonstrate the validity of deep learning methodologies through the answers it provides for different research questions. For example in the application of deep learning to Ambient Occlusion backing (see Section 4) the aim was to reduce the computational cost and optimize the memory space occupied because such problems affect the offline Ambient occlusion. Other examples concern the problems that affect image-to-image translation or other image processing technique such as the chroma key.

To study the deep learning techniques we started from a classification problem (see Chapter 3) and from this, we realized that one of the significant problems of deep learning concerns the rules of the construction of a dataset appropriate to the type of considered application context. In particular, the question that often arises during the dataset construction are: *"how should the dataset samples be constructed?"*, *"how much should the dataset samples be different from each other?"*, *"is it necessary to perform pre-processing on the data?"*, *"what type of pre-processing?"*, etc.

Investigating these aspects, we tried to answer these questions in the previous chapters, but the dataset is not the only topic addressed. Other questions concern the type and the configuration of neural networks for each context, by trying to bring their performance to the highest levels. When we applied deep learning to the ambient occlusion computing we proposed a multilayer perceptron as neural network in which the dataset consisted only of the "normals" belonging to the scene, and in this context, we evaluated the extent to which the neural network can be used for real-time rendering obtaining the results described in the Chapter 4. In this context, we tried to answer questions like *"how many artificial neurons are necessary to obtain neural network convergence and optimal performance?"*, *"how many layers are necessary to use?"*, *"what are the correct configuration parameters of the network and the training?"*, *"how is the validity of the network and training tested?"*, etc. To answer these questions we studied the correct solution in terms of the dataset and the neural network configuration to obtain the optimal results in terms of quality and performance, overcoming the state of the art and demonstrating it through the comparisons with other similar approaches.

Ambient occlusion is closely linked to the lighting scheme of the scene. A similar concept is at the basis of the image-to-image translation and from this perspective, we investigated the computational models based on the fully convolutional neural networks, in particular the encoder-decoder networks also called U-Shape Networks 2.1.4. Through the proposed approach we tried to answer the following questions: *"what optimization algorithm to use?"*, *"what are the hyperparameters to configure?"*, *"how to configure the U-Shape network to get the best result in a given context?"*, *"what are the regularization techniques that allow having a higher level of generalization?"*, etc. In order to change the lighting scheme of images such as faces (see Section 6) we opened other research scenarios. The removal of image background could be useful in specializing the neural network to work only on the characteristics of faces, decoupling it from the global scene. For this reason, we investigated

chroma key (see Section 7) techniques by also using deep learning in this context and proposing a method to apply this technique to video editing.

We aim in future developments to use the based convolutional neural network 2.1.3 and generative adversarial network algorithms 8.2.1 in real-time rendering and to improve the photorealism level bringing the offline rendering algorithms quality also to real-time performance. Also from an industrial point of view, big companies like NVIDIA are moving in this direction, and in fact, the proposed GPU Turing architecture <sup>1</sup> uses a mix of programmable shaders, tensor core <sup>2</sup> and much more to improve photorealism level details.

---

<sup>1</sup> NVIDIA Turing GPU Architecture:<https://www.nvidia.com/en-us/geforce/news/graphics-reinvented-new-technologies-in-rtx-graphics-cards>

<sup>2</sup> Tensor Core: process unit capable of processing operations on matrices and tensors that are at the basis of deep learning algorithms.



# Appendix A

## Comprehensibility of the Deep Learning Terminology

This appendix provides further information on details not explained in the chapters of the thesis. To obtain a neural network convergence is necessary to minimize a function called **Loss Function** or objective function. Typically, in the deep learning problems, two types of loss functions are used: **cross-entropy** for the classification problems and **mean squared error** for regression problem.

Let  $y$  the probability distribution of the neural network prediction values for a given input and  $t$  the true distribution probability for the label (ground truth) values. The cross-entropy for  $N$  samples is defined as

$$L_{c-e}(y, t) = -\frac{1}{N} \sum_i t_i \log y_i. \quad (\text{A.1})$$

Let us to consider a binary classifier with  $y \in \{0, 1\}$ , the cross-entropy can be used to compute the distance between  $t$  and  $y$  as

$$L_{c-e}(y, t) = -\frac{1}{N} \sum_i (t_i \log y_i + (1 - t_i) \log (1 - y_i)). \quad (\text{A.2})$$

In a similar way, the mean squared error is defined as

$$L_{mse}(y,t) = \frac{1}{N} \sum_i (y-t)^2, \tag{A.3}$$

we used cross-entropy for the applications described in Chapters 3 and 7, while the mean squared error is used for the applications described in the other chapters. When the loss is computed on training samples it is called loss training and when the loss is computed on validation samples it is called loss validation. Although the performance (loss function) of the neural network is often confused with accuracy, in some contexts such indicators can be measured separately. It is possible to compute the accuracy in training, validation or both phases as we shown in the equation (6.7) or by using Root-Mean-Square Error or other metrics. However, it is possible to compute accuracy after the training, *i.e.*, in the testing phase, *e.g.*, through Structural Similarity Index [163] or other metrics as described in Section 7.5.1.

The minimization of the loss function is performed through optimization algorithms. Such algorithms are based on the gradient descent, whose functioning is shown in Figure A.1.

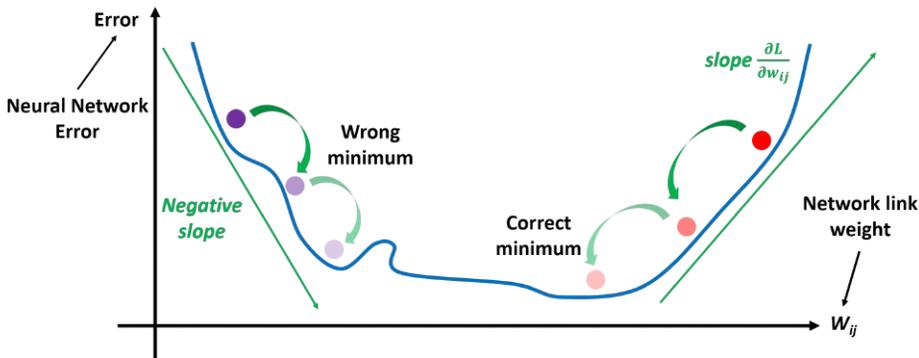


Fig. A.1 Starting from a point  $x_0$  the gradient  $\nabla f(x_0)$  is computed. Gradient provides the following direction to obtain minimum direction. At a certain distance, a new point  $x_1$  is found, on which the gradient  $\nabla f(x_1)$  is calculated again. The algorithm continues until the gradient reaches zero.

In particular, in the deep learning context, an extension of gradient descent, called **Stochastic Gradient Descent**(A.4) and its variants are used. The Stochastic Gradient Descent is characterized by a low computational cost because it is computed on a subset of random samples of the dataset, called **mini-batch**, on which the loss function is computed. In this algorithm a very important parameter is the learning rate, in fact, it is necessary to decrease gradually its value over the time because Stochastic Gradient Descent introduce a source of noise due to the mini-batch that remains even when it reaches the minimum value. Moreover, through the learning rate, it is possible to monitor the learning curve shown by the loss function over the time [56].

$$w_{t+1} = w_t - \varepsilon \frac{1}{m} \sum_i \nabla_w L(x, y, w_t) \quad (\text{A.4})$$

The equation (A.4) represents the updating of the weight through Stochastic Gradient Descent. The terms  $w_{t+1}$  and  $w_t$  represent respectively the new and the old value of the weight and the term  $m$  represent the mini-batch size. We use Stochastic Gradient Descent in the night to day application, as shown in Section 5.2.

For the coin recognition (see Section 3), we used a variant called Stochastic Gradient Descent with Momentum [138]. Momentum is a term dependent on previous interactions used to regularize the movement in the parameter space. The idea is to move in the direction of the exponentially decaying moving average that characterizes the past gradients. The algorithm introduces a variable  $v$  that represents the direction and speed of the parameter movements in order to accelerate the learning.

$$v_{t+1} = \alpha v_t - \varepsilon \frac{1}{m} \sum_i \nabla_w L(x, y, w_t) \quad (\text{A.5})$$

Where  $\alpha$  is the momentum parameter used to control the importance of past iteration. The weight is updated by adding  $v$  and the gradient of the current iteration:

$$w_{t+1} = w_t + \alpha v_t - \varepsilon \frac{1}{m} \sum_i \nabla_w L(x, y, w_t) \quad (\text{A.6})$$

For applications presented in Chapters 6 and 7, we used another optimized variant of the Stochastic Gradient Descent with Momentum called Adaptive Moment Estimation (Adam) [84]. Adam is based on other well-known optimization algorithms, AdaGrad [40] and RMSProp [156]. Such algorithms are called adaptive algorithms because they adapt the learning rate for each of the parameters, improving the learning trend. In particular, Adam takes into account the moving average of the first  $s$  and second  $r$  moments of the gradient, using the correct estimators  $\hat{s}$  and  $\hat{r}$  to counter the zero bias in the first iterations.

$$s_{t+1} = \beta_1 s_t + (1 - \beta_1) \frac{1}{m} \sum_i \nabla_w L(x, y, w_t) \quad (\text{A.7})$$

The equation (A.7) describes the partial update of the estimate of the first moment where  $\beta_1$  and  $\beta_2$  represent the exponential decay rates of the moment called also forgetting factors.

$$r_{t+1} = \beta_2 r_t + (1 - \beta_2) \left( \frac{1}{m} \sum_i \nabla_w L(x, y, w_t) \right)^2 \quad (\text{A.8})$$

The equation (A.8) describes the partial update of the estimate of the second moment.

$$\hat{s} = \frac{s_{t+1}}{\beta_1^{t+1}} \quad (\text{A.9}) \quad \hat{r} = \frac{r_{t+1}}{\beta_2^{t+1}} \quad (\text{A.10})$$

The equations (A.9) and (A.10) show the correct biases for the first  $\hat{s}$  and second moment  $\hat{r}$ . Finally the weight update is performed as

$$w_{t+1} = w_t - \varepsilon \frac{\hat{s}}{\sqrt{\hat{r} + \delta}} \quad (\text{A.11})$$

where  $\delta$  is a small smoothing constant used for numerical stabilization.

Another optimization method, used in the application described in Section 4, is the Scaled Conjugate Gradient, widely used in the multilayer perceptrons. This algorithm is too complex and very long to explain in a few lines, for this reason, we refer the original paper, proposed by Martin Fodsllette Møller [109], for more details. The main advantages of this algorithm concern the fact that the parameters do not depend critically on the user's choice and it is faster than other similar approaches.

The above-described optimization algorithms introduce new parameters to those that already belong to neural networks. Such parameters are called hyperparameters. Although the hyperparameters are defined as the parameters whose value is set before the learning process, it is possible to design a nested procedure to learn the best hyperparameters for another learning algorithm [56]. With reference to the algorithms described above, some of the hyperparameters are the learning rate, the moment control parameter, the size of mini-batch and others. Some hyperparameters belonging to neural networks are the number of hidden units, the size of the convolutional kernel, the dropout rate, the weight decay coefficient, the zero padding, *etc.* Hyperparameter can be useful to control the deep learning algorithm behaviour. Two important hyperparameters are epochs and batch size: the epoch represents the crossing of the entire dataset in the neural network both in the forward and backward propagation phase; the batch size is the number of the dataset samples used in one iteration. This latter hyperparameter can involve three options: batch-mode, when the batch size is equal to the dataset size (in this case, an iteration corresponds to an epoch); mini-batch mode, when the batch size is greater than one but less than the dataset size; the stochastic model, when the batch size is equal to one. In this final case, the neural network is updated after each sample. The use of the batch mode requires less memory and in general the neural network train faster with mini-batches.

In Chapter 6 we introduce the use of transfer learning. As described in Chapter 1, transfer learning [126] allows to exploit the "knowledge" obtained from a neural network in a specific task and reuse it in similar but different tasks. This operation can be performed by using pre-trained models. In particular, is possible to use the weights of the neural network already trained for a specific problem as initialization values of weights of the same or similar neural network for another problem. The weights of the neural network can be pre-initialized partially or totally. When neural network is partially pre-initialized, is possible to freeze such part and to train the uninitialized part. In this way, the old knowledge of the neural network is totally preserved and can be used for the new task.

Another method, introduced in Chapter 6, is the Xavier Initialization [54]. We used such method for the initialization of the last decoder layer (see Section 6.2.2) to avoid too small or too large weights initialization. It reduces or increases the signal as it crosses the neural network until it becomes too small or too large to be useful. In particular, suppose we have a linear classifier and the weights initialization with zero mean and specific variance, the output is given from

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots w_n x_n. \quad (\text{A.12})$$

The variance of the  $w_i x_i$  can be written as

$$\text{var}(w_i x_i) = \mathbb{E}[x_i]^2 \text{var}(w_i) + \mathbb{E}[w_i]^2 \text{var}(x_i) + \text{var}(w_i) \text{var}(x_i). \quad (\text{A.13})$$

Since we have initialized the mean as zero for the weights initialization distribution, the equation (A.13) can be simplified as

$$\text{var}(w_i x_i) = \text{var}(w_i) \text{var}(x_i). \quad (\text{A.14})$$

If we consider  $x_i$  and  $w_i$  independent and identically distributed, the variance of the neural network output value can be written as

$$\text{var}(w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots w_n x_n) = n \text{var}(w_i) \text{var}(x_i) = \text{var}(y) \quad (\text{A.15})$$

Since we want the same variance for the input and the output, then  $n \text{var}(w_i) \text{var}(x_i) = 1$  and  $\text{var}(w_i) = \frac{1}{n_f}$ , where  $n_f$  is the number of neurons fed by the forward propagation phase. The same operation is performed for the backward propagation phase, where  $\text{var}(w_i) = \frac{1}{n_b}$ . To satisfy these two properties, it is necessary to consider equal  $n_f$  and  $n_b$  and the variance is computed as

$$\text{var}(w_i) = \frac{2}{n_f + n_b} \quad (\text{A.16})$$

As shown in the previous equations, Xavier initialization is useful to obtain a better variance value for the weights initialization distribution.

For our classification problems (see Sections 3 and 7), we used as activation function for the last layer the **Softmax** function [13], called also normalized exponential function is often used as activation function for the last layer of the multi-class classifier, while the logistic function, such as sigmoid, is used for binary classification tasks. Softmax is defined as

$$\sigma(y_i) = \frac{e^{y_i}}{\sum_{k=1}^K e^{y_k}}, \quad (\text{A.17})$$

where  $y_i$  represent the  $i^{\text{th}}$  input element of the softmax that corresponds to class  $i$  and  $K$  is the number of classes. Such function acquires a vector of real values, which indicates the scores, and turns it into a probabilities vector, which contain the probability of sample  $x$  belong to each class. The sum of all probabilities is equal to one and the highest value of softmax indicates the greater probability of belonging to the class associated with that probability. Since the softmax output is used for the cross-entropy computation, its values have to be greater then zero.

Finally, the last topic to be clarified is the bilinear interpolation that we used in the application described in Chapter 5. Such technique is applied to resize the image or feature map dimension in 2D space (Figure A.2).

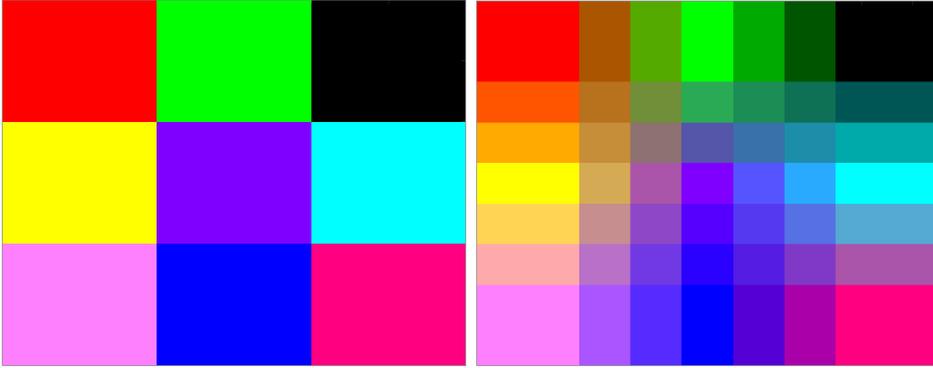


Fig. A.2 Left figure shows the original image and right figure shows the image after the application of the bilinear interpolation.

Bilinear interpolation allows to find the value of an unknown function  $f$  at the vertex  $x, y$ . We assumed that the value of the four vertices  $v_{1,1} = (x_1, y_1)$ ,  $v_{1,2} = (x_1, x_2)$ ,  $v_{2,1} = (x_2, y_2)$ ,  $v_{2,2} = (x_2, y_2)$  are known, as shown in Figure A.3. The algorithm works by interpolating first in the  $x$  direction

$$\begin{cases} f(x, y_1) = \frac{x_2 - x}{x_2 - x_1} f(v_{1,1}) + \frac{x - x_1}{x_2 - x_1} f(v_{2,1}) \\ f(x, y_2) = \frac{x_2 - x}{x_2 - x_1} f(v_{1,2}) + \frac{x - x_1}{x_2 - x_1} f(v_{2,2}), \end{cases} \quad (\text{A.18})$$

and then in the  $y$  direction by replacing (A.18) in the next equation:

$$f(x, y) = \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2). \quad (\text{A.19})$$

By explicating products and simplifying the equation above, will be obtained the following formula:

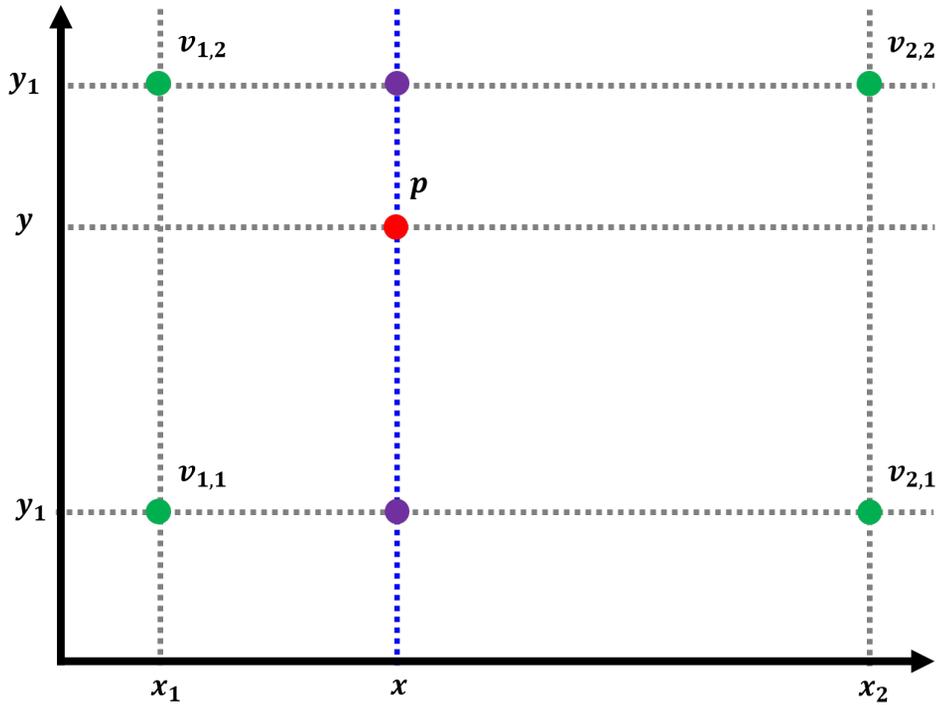


Fig. A.3 Four vertex shown with green dots represent the known points and the red dot represent the point we want to interpolate.

$$f(x,y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} f(v_{1,1}) & f(v_{1,2}) \\ f(v_{2,1}) & f(v_{2,2}) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}$$



# References

- [1] Aalund, F. P. and Bærentzen, J. A. (2013). A comparative study of screen-space ambient occlusion methods. Bachelor thesis, Technical University of Denmark, Informatics and Mathematical Modelling.
- [2] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.
- [3] Agata, H., Yamashita, A., and Kaneko, T. (2007). Chroma key using a checker pattern background. *IEICE TRANSACTIONS on Information and Systems*, 90(1):242–249.
- [4] Agrawal, A., Raskar, R., Nayar, S. K., and Li, Y. (2005). Removing photography artifacts using gradient projection and flash-exposure sampling. *ACM Trans. Graph.*, 24(3):828–835.
- [5] Aksoy, Y., Kim, C., Kellnhofer, P., Paris, S., Elgharib, M., Pollefeys, M., and Matusik, W. (2018). A dataset of flash and ambient illumination pairs from the crowd. In *Proc. ECCV*.
- [6] Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495.
- [7] Bagiwa, M. A., Wahab, A. W. A., Idris, M. Y. I., Khan, S., and Choo, K.-K. R. (2016). Chroma key background detection for digital video using statistical correlation of blurring artifact. *Digital Investigation*, 19:29 – 43.
- [8] Banterle, F., Corsini, M., Cignoni, P., and Scopigno, R. (2012). A Low-Memory, Straightforward and Fast Bilateral Filter Through Subsampling in Spatial Domain. *Computer Graphics Forum*, 31(1):19–32.

- [9] Batenburg, K. J. and Sijbers, J. (2009). Optimal threshold selection for tomogram segmentation by projection distance minimization. *IEEE Transactions on Medical Imaging*, 28(5):676–686.
- [10] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.
- [11] Bengio, Y. et al. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.
- [12] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, volume 1.
- [13] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
- [14] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.
- [15] Bremananth, R., Balaji, B., Sankari, M., and Chitra, A. (2005). A New Approach to Coin Recognition using Neural Pattern Analysis. In *2005 Annual IEEE India Conference - Indicon*, pages 366–370.
- [16] Brostow, G. J., Fauqueur, J., and Cipolla, R. (2009). Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97.
- [17] Burkardt, J. (2014). The truncated normal distribution. *Department of Scientific Computing Website*.
- [18] Bychkovsky, V., Paris, S., Chan, E., and Durand, F. (2011). Learning photographic global tonal adjustment with a database of input/output image pairs. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 97–104. IEEE.
- [19] Capece, N., Agatiello, R., and Erra, U. (2016a). A Client-Server Framework for the Design of Geo-Location Based Augmented Reality Applications. In *2016 20th International Conference Information Visualisation (IV)*, pages 130–135.

- [20] Capece, N., Banterle, F., Cignoni, P., Ganovelli, F., Scopigno, R., and Erra, U. (2019). Deepflash: Turning a flash selfie into a studio portrait. *arXiv preprint arXiv:1901.04252*.
- [21] Capece, N., Erra, U., and Ciliberto, A. V. (2016b). Implementation of a coin recognition system for mobile devices with deep learning. In *2016 12th International Conference on Signal-Image Technology Internet-Based Systems (SITIS)*, pages 186–192.
- [22] Capece, N., Erra, U., and Scolamiero, R. (2017). Converting night-time images to day-time images through a deep learning approach. In *2017 21st International Conference Information Visualisation (IV)*, pages 324–331.
- [23] Chen, C., Chen, Q., Xu, J., and Koltun, V. (2018a). Learning to See in the Dark.
- [24] Chen, J., Paris, S., and Durand, F. (2007). Real-time edge-aware image processing with the bilateral grid. *ACM Trans. Graph.*, 26(3).
- [25] Chen, Q., Li, D., and Tang, C.-K. (2013a). Knn matting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(9):2175–2188.
- [26] Chen, X., Zou, D., Zhiying Zhou, S., Zhao, Q., and Tan, P. (2013b). Image matting with local and nonlocal smooth priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1902–1907.
- [27] Chen, Y.-S., Wang, Y.-C., Kao, M.-H., and Chuang, Y.-Y. (2018b). Deep photo enhancer: Unpaired learning for image enhancement from photographs with gans. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6306–6314. IEEE.
- [28] Cheng, Z., Yang, Q., and Sheng, B. (2015). Deep colorization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 415–423.
- [29] Chiara, R. D., Santo, V. D., Erra, U., and Scarano, V. (2007). Real positioning in virtual environments using game engines. In *Eurographics Italian Chapter Conference 2007, Trento, Italy, 2007*, pages 203–208.
- [30] Cireşan, D. C., Giusti, A., Gambardella, L. M., and Schmidhuber, J. (2013). Mitosis detection in breast cancer histology images with deep neural networks. In Mori, K., Sakuma, I., Sato, Y., Barillot, C., and

- Navab, N., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, pages 411–418, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [31] Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning.
- [32] Danaee, P., Ghaeini, R., and Hendrix, D. A. (2017). A deep learning approach for cancer detection and relevant gene identification. In *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2017*, pages 219–229. World Scientific.
- [33] Dao-Duc, C., Xiaohui, H., and Morère, O. (2015). Maritime Vessel Images Classification Using Deep Convolutional Neural Networks. In *Proceedings of the Sixth International Symposium on Information and Communication Technology, SoICT 2015*, pages 276–281, New York, NY, USA. ACM.
- [34] Davidsson, P. (1996). Coin classification using a novel technique for learning characteristic decision trees by controlling the degree of generalization. In *Ninth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*. Gordon and Breach Science Publishers.
- [35] D’Avino, D., Cozzolino, D., Poggi, G., and Verdoliva, L. (2017). Autoencoder with recurrent neural networks for video forgery detection. *Electronic Imaging*, 2017(7):92–99.
- [36] De Mauro, A., Greco, M., and Grimaldi, M. (2016). A formal definition of big data based on its essential features. *Library Review*, 65(3):122–135.
- [37] Deng, L., Li, J., Huang, J.-T., Yao, K., Yu, D., Seide, F., Seltzer, M. L., Zweig, G., He, X., Williams, J. D., et al. (2013). Recent advances in deep learning for speech research at microsoft. In *ICASSP*, volume 26, page 64.
- [38] Dheekonda, R. S., Panda, S., Hasan, M., Anwar, S., et al. (2017). Object detection from a vehicle using deep learning network and future integration with multi-sensor fusion algorithm. Technical report, SAE Technical Paper.
- [39] Dosovitskiy, A., Springenberg, J. T., Riedmiller, M., and Brox, T. (2014). Discriminative unsupervised feature learning with convolutional neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1, NIPS’14*, pages 766–774, Cambridge, MA, USA. MIT Press.

- [40] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.
- [41] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). Unsupervised learning and clustering. *Pattern classification*, pages 517–601.
- [42] Eilertsen, G., Kronander, J., Denes, G., Mantiuk, R., and Unger, J. (2017). Hdr image reconstruction from a single exposure using deep cnns. *ACM Transactions on Graphics (TOG)*, 36(6).
- [43] Eisemann, E. and Durand, F. (2004). Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph.*, 23(3):673–678.
- [44] Erra, U. and Capece, N. (2017). Engineering an advanced geo-location augmented reality framework for smart mobile devices. *Journal of Ambient Intelligence and Humanized Computing*.
- [45] Erra, U., Capece, N. F., and Agatiello, R. (2017). Ambient Occlusion Baking via a Feed-Forward Neural Network. In Peytavie, A. and Bosch, C., editors, *EG 2017 - Short Papers*. The Eurographics Association.
- [46] Erra, U., Senatore, S., Minnella, F., and Caggianese, G. (2015). Approximate tf-idf based on topic extraction from massive message stream using the gpu. *Inf. Sci.*, 292(C):143–161.
- [47] Evans, J. R. and Lindner, C. H. (2012). Business analytics: the next frontier for decision sciences. *Decision Line*, 43(2):4–6.
- [48] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338.
- [49] Fakoor, R., Ladhak, F., Nazi, A., and Huber, M. (2013). Using deep learning to enhance cancer diagnosis and classification. In *Proceedings of the International Conference on Machine Learning*, volume 28.
- [50] Farfadi, S. S., Saberian, M. J., and Li, L.-J. (2015). Multi-view Face Detection Using Deep Convolutional Neural Networks. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval, ICMR '15*, pages 643–650, New York, NY, USA. ACM.

- [51] Fukumi, M., Omatu, S., Takeda, F., and Kosaka, T. (1992). Rotation-invariant neural pattern recognition system with application to coin recognition. *IEEE Transactions on Neural Networks*, 3(2):272–279.
- [52] Ge, F., Wang, S., and Liu, T. (2007). New benchmark for image segmentation evaluation. *Journal of Electronic Imaging*, 16(3):033011.
- [53] Gharbi, M., Chen, J., Barron, J. T., Hasinoff, S. W., and Durand, F. (2017). Deep bilateral learning for real-time image enhancement. *ACM Transactions on Graphics (TOG)*, 36(4):118.
- [54] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.
- [55] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.
- [56] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- [57] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.
- [58] Gravås, L. O. (2013). Image-space ambient obscurance in webgl. Technical report, Institutt for datateknikk og informasjonsvitenskap.
- [59] Gvili, R., Kaplan, A., Ofek, E., and Yahav, G. (2003). Depth keying. In *Stereoscopic Displays and Virtual Reality Systems X*, volume 5006, pages 564–575. International Society for Optics and Photonics.
- [60] Hariharan, B., Arbeláez, P., Girshick, R., and Malik, J. (2014). Simultaneous detection and segmentation. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision – ECCV 2014*, pages 297–312, Cham. Springer International Publishing.
- [61] He, K., Rhemann, C., Rother, C., Tang, X., and Sun, J. (2011). A global sampling method for alpha matting.

- [62] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA. IEEE Computer Society.
- [63] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.
- [64] Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.
- [65] Holden, D., Saito, J., and Komura, T. (2016). Neural network ambient occlusion. In *SIGGRAPH ASIA 2016 Technical Briefs*, SA '16, pages 9:1–9:4, New York, NY, USA. ACM.
- [66] Hu, Y., He, H., Xu, C., Wang, B., and Lin, S. (2018). Exposure: A white-box photo post-processing framework. *ACM Transactions on Graphics (TOG)*, 37(2):26.
- [67] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *CVPR*, volume 1, page 3.
- [68] Ignatov, A., Kobyshev, N., Timofte, R., Vanhoey, K., and Van Gool, L. (2017). Dslr-quality photos on mobile devices with deep convolutional networks. In *the IEEE Int. Conf. on Computer Vision (ICCV)*.
- [69] Iizuka, S., Simo-Serra, E., and Ishikawa, H. (2016). Let there be color!: Joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Trans. Graph.*, 35(4):110:1–110:11.
- [70] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 448–456.
- [71] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *CVPR*.

- [72] Jan, B., Farman, H., Khan, M., Imran, M., Islam, I. U., Ahmad, A., Ali, S., and Jeon, G. (2017). Deep learning in big data analytics: A comparative study. *Computers & Electrical Engineering*.
- [73] Jensen, H. W. (1996). Global illumination using photon maps. In *Rendering Techniques '96*, pages 21–30. Springer.
- [74] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia, MM '14*, pages 675–678, New York, NY, USA. ACM.
- [75] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.
- [76] Kajiya, J. T. (1986). The rendering equation. In *ACM SIGGRAPH computer graphics*, volume 20, pages 143–150. ACM.
- [77] Karacan, L., Erdem, A., and Erdem, E. (2015). Image matting with kl-divergence based sparse sampling. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 424–432.
- [78] Kavan, L., Bargaiteil, A. W., and Sloan, P.-P. (2011). Least squares vertex baking. In *Proc. of the Twenty-second Eurographics Conference on Rendering, EGSR '11*, pages 1319–1326, Aire-la-Ville, Switzerland. Eurographics Association.
- [79] Kawaguchi, K., Kaelbling, L. P., and Bengio, Y. (2017). Generalization in deep learning. *arXiv preprint arXiv:1710.05468*.
- [80] Kawano, Y. and Yanai, K. (2014). Food Image Recognition with Deep Convolutional Features. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication, UbiComp '14 Adjunct*, pages 589–593, New York, NY, USA. ACM.
- [81] Kenney, J., Buckley, T., and Brock, O. (2009). Interactive segmentation for manipulation in unstructured environments. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 1377–1382. IEEE.

- [82] Kim, J. and Pavlovic, V. (2015). Discovering Characteristic Landmarks on Ancient Coins using Convolutional Networks. *CoRR*, abs/1506.09174.
- [83] Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.
- [84] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [85] Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada.
- [86] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.
- [87] Langer, M. S. and Bülthoff, H. H. (2000). Depth discrimination from shading under diffuse lighting. *Perception*, 29(6):649–660.
- [88] Langer, M. S. and Zucker, S. W. (1994). Shape-from-shading on a cloudy day. *JOSA A*, 11(2):467–478.
- [89] Larsen, A. B. L., Sønderby, S. K., Larochelle, H., and Winther, O. (2016). Autoencoding beyond pixels using a learned similarity metric. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1558–1566. JMLR.org.
- [90] LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nature*, 521(7553):436–444.
- [91] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.
- [92] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- [93] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer.

- [94] Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2017). Photo-realistic single image super-resolution using a generative adversarial network. pages 105–114.
- [95] Lee, J. and Lee, S. (2016). Hallucination from noon to night images using cnn. In *SIGGRAPH ASIA 2016 Posters*, page 15. ACM.
- [96] Li, H. (2011). RESTful Web service frameworks in Java. In *Signal Processing, Communications and Computing (ICSPCC), 2011 IEEE International Conference on*, pages 1–4.
- [97] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.
- [98] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.
- [99] Maggiori, E., Tarabalka, Y., Charpiat, G., and Alliez, P. (2017). High-resolution image classification with convolutional networks. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 5157–5160.
- [100] Mara, M., McGuire, M., Nowrouzezahrai, D., and Luebke, D. (2016). Deep g-buffers for stable global illumination approximation. In *Proc. of High Performance Graphics, HPG '16*, pages 87–98, Aire-la-Ville, Switzerland. Eurographics Association.
- [101] Masinter, L. (2015). Returning values from forms: multipart/form-data. Technical report.
- [102] Mattes, D., Haynor, D. R., Vesselle, H., Lewellyn, T. K., and Eubank, W. (2001). Nonrigid multimodality image registration. In *Medical imaging 2001: image processing*, volume 4322, pages 1609–1621. International Society for Optics and Photonics.
- [103] McGuire, M., Osman, B., Bukowski, M., and Hennessy, P. (2011). The alchemy screen-space ambient obscurance algorithm. In *Proc. of the ACM SIGGRAPH Symposium on High Performance Graphics, HPG '11*, pages 25–32, New York, NY, USA. ACM.

- [104] Méndez-Feliu, À. and Sbert, M. (2009). From obscurances to ambient occlusion: A survey. *The Visual Computer*, 25(2):181–196.
- [105] Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.
- [106] Mittring, M. (2007). Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 Courses, SIGGRAPH '07*, pages 97–121, New York, NY, USA. ACM.
- [107] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari With Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*.
- [108] Modi, S. and Bawa, D. S. (2011). Article: Automated Coin Recognition System using ANN. *International Journal of Computer Applications*, 26(4):13–18. Full text available.
- [109] Møller, M. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533.
- [110] Morar, A., Moldoveanu, F., and Gröller, E. (2012). Image segmentation based on active contours without edges. In *2012 IEEE 8th International Conference on Intelligent Computer Communication and Processing*, pages 213–220. IEEE.
- [111] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10*, pages 807–814, USA. Omnipress.
- [112] Nalbach, O., Arabadziyska, E., Mehta, D., Seidel, H.-P., and Ritschel, T. (2017). Deep shading: Convolutional neural networks for screen space shading. *Comput. Graph. Forum*, 36(4):65–78.
- [113] Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate  $o(1/k^2)$ . In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547.
- [114] Ng, A. (2013). Machine learning and ai via brain simulations. *Andrew Ng*.

- [115] Noh, H., Hong, S., and Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528.
- [116] Nölle, M., Penz, H., Rubik, M., Mayer, K., Holländer, I., and Granec, R. (2003). Dagobert-a new coin recognition and sorting system.
- [117] Oquab, M., Bottou, L., Laptev, I., and Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '14*, pages 1717–1724, Washington, DC, USA. IEEE Computer Society.
- [118] Pal, N. R. and Pal, S. K. (1993). A review on image segmentation techniques. *Pattern recognition*, 26(9):1277–1294.
- [119] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.
- [120] Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M. (2010). Optix: A general purpose ray tracing engine. In *ACM SIGGRAPH 2010 Papers, SIGGRAPH '10*, pages 66:1–66:13, New York, NY, USA. ACM.
- [121] Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). Deep face recognition. In *British Machine Vision Conference*.
- [122] Pautasso, C., Wilde, E., and Alarcon, R. (2014). *REST: Advanced Research Topics and Practical Applications*. Springer Publishing Company, Incorporated.
- [123] Petschnigg, G., Szeliski, R., Agrawala, M., Cohen, M., Hoppe, H., and Toyama, K. (2004). Digital photography with flash and no-flash image pairs. *ACM Trans. Graph.*, 23(3):664–672.
- [124] Porter, T. and Duff, T. (1984). Compositing digital images. In *ACM Siggraph Computer Graphics*, volume 18, pages 253–259. ACM.
- [125] Prabhakar, G., Kailath, B., Natarajan, S., and Kumar, R. (2017). Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving. In *2017 IEEE Region 10 Symposium (TENSymp)*, pages 1–6.

- [126] Pratt, L. Y. (1993). Discriminability-based transfer between neural networks. In *Advances in neural information processing systems*, pages 204–211.
- [127] Prechelt, L. (2012). *Early Stopping — But When?*, pages 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [128] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1982). *Numerical recipes in C*, volume 2. Cambridge Univ Press.
- [129] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.
- [130] Raghunathan, S., Stredney, D., Schmalbrock, P., and Clymer, B. D. (2005). Image registration using rigid registration and maximization of mutual information. In *Poster presented at: MMVR13. The 13th Annual Medicine Meets Virtual Reality Conference*.
- [131] Ramachandran, P., Liu, P. J., and Le, Q. V. (2016). Unsupervised pretraining for sequence to sequence learning. *arXiv preprint arXiv:1611.02683*.
- [132] Reiser, M., Ronneberger, O., and Burkhardt, H. (2006). An efficient gradient based registration technique for coin recognition. In *Proc. of the Muscle CIS Coin Competition Workshop, Berlin, Germany*, pages 19–31.
- [133] Ren, P., Wang, J., Gong, M., Lin, S., Tong, X., and Guo, B. (2013). Global illumination with radiance regression functions. *ACM Trans. Graph.*, 32(4):130:1–130:12.
- [134] Rhemann, C., Rother, C., Wang, J., Gelautz, M., Kohli, P., and Rott, P. (2009). A perceptually motivated online benchmark for image matting. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1826–1833. IEEE.
- [135] Ritschel, T., Dachsbacher, C., Grosch, T., and Kautz, J. (2012). The state of the art in interactive global illumination. In *Computer Graphics Forum*, volume 31, pages 160–188. Wiley Online Library.
- [136] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

- [137] Rosasco, L., De Vito, E., Caponnetto, A., Piana, M., and Verri, A. (2004). Are loss functions all the same? *Neural Comput.*, 16(5):1063–1076.
- [138] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533.
- [139] Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.
- [140] Sasaki, Y. et al. (2007). The truth of the f-measure. *Teach Tutor mater*, 1(5):1–5.
- [141] Schlag, I. and Arandjelovic, O. (2017). Ancient roman coin recognition in the wild using deep learning based recognition of artistically depicted face profiles. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 2898–2906.
- [142] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.
- [143] Sengupta, S., Kanazawa, A., Castillo, C. D., and Jacobs, D. W. (2018). Sfsnet: Learning shape, reflectance and illuminance of faces in the wild. In *Computer Vision and Pattern Recognition (CVPR)*.
- [144] Shen, X., Hertzmann, A., Jia, J., Paris, S., Price, B., Shechtman, E., and Sachs, I. (2016). Automatic portrait segmentation for image stylization. *Comput. Graph. Forum*, 35(2):93–102.
- [145] Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905.
- [146] Shih, Y., Paris, S., Barnes, C., Freeman, W. T., and Durand, F. (2014). Style transfer for headshot portraits. *ACM Transactions on Graphics (TOG)*, 33(4):148.
- [147] Shih, Y., Paris, S., Durand, F., and Freeman, W. T. (2013). Data-driven hallucination of different times of day from a single outdoor photo. *ACM Transactions on Graphics (TOG)*, 32(6):200.
- [148] Shu, Z., Hadap, S., Shechtman, E., Sunkavalli, K., Paris, S., and Samarasinghe, D. (2018). Portrait lighting transfer using a mass transport approach. *ACM Transactions on Graphics (TOG)*, 37(1):2.

- [149] Shu, Z., Yumer, E., Hadap, S., Sunkavalli, K., Shechtman, E., and Samaras, D. (2017). Neural face editing with intrinsic image disentangling. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5444–5453. IEEE.
- [150] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- [151] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.
- [152] Stehman, S. V. (1997). Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89.
- [153] Styner, M., Brechbuhler, C., Szckely, G., and Gerig, G. (2000). Parametric estimate of intensity inhomogeneities applied to mri. *IEEE Transactions on Medical Imaging*, 19(3):153–165.
- [154] Sun, Y., Wang, X., and Tang, X. (2015). Deeply learned face representations are sparse, selective, and robust. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2892–2900.
- [155] Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.
- [156] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*.
- [157] Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846.
- [158] Van Der Maaten, L. and Postma, E. (2006). *Towards automatic coin classification*. na.
- [159] Vidal, B. (2012). Chroma key visual feedback based on non-retroreflective polarized reflection in retroreflective screens. *IEEE Transactions on Broadcasting*, 58(1):144–150.
- [160] Wang, J., Cohen, M. F., et al. (2008). Image and video matting: a survey. *Foundations and Trends® in Computer Graphics and Vision*, 3(2):97–175.

- [161] Wang, S., Zheng, J., Hu, H.-M., and Li, B. (2013). Naturalness preserved enhancement algorithm for non-uniform illumination images. *IEEE Transactions on Image Processing*, 22(9):3538–3548.
- [162] Wang, Y., Zhang, L., Liu, Z., Hua, G., Wen, Z., Zhang, Z., and Samaras, D. (2009). Face relighting from a single image under arbitrary unknown lighting conditions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(11):1968–1984.
- [163] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.
- [164] Wen, Z., Liu, Z., and Huang, T. S. (2003). Face relighting with radiance environment maps. In *null*, page 158. IEEE.
- [165] Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853.
- [166] Xu, N., Price, B. L., Cohen, S., and Huang, T. S. (2017). Deep image matting. In *CVPR*, volume 2, page 4.
- [167] Yamashita, A., Agata, H., and Kaneko, T. (2008). Every color chromakey. In *2008 19th International Conference on Pattern Recognition*, pages 1–4.
- [168] Zaharieva, M., Kampel, M., and Zambanini, S. (2007). Image based recognition of coins – an overview of the COINS project. In *Performance Evaluation for Computer Vision 31st AAPR/OAGM Workshop 2007*, pages 57–64.
- [169] Zambanini, S., Kampel, M., and Schlapke, M. (2008). On the use of computer vision for numismatic research. In *Proceedings of the 9th International Conference on Virtual Reality, Archaeology and Cultural Heritage*, pages 17–24. Eurographics Association.
- [170] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.
- [171] Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535.

- [172] Zeiler, M. D., Taylor, G. W., and Fergus, R. (2011). Adaptive deconvolutional networks for mid and high level feature learning. In *2011 International Conference on Computer Vision*, pages 2018–2025.
- [173] Zhang, X., Ng, R., and Chen, Q. (2018). Single image reflection separation with perceptual losses. In *IEEE Conference on Computer Vision and Pattern Recognition*.
- [174] Zhao, W., Du, S., and Emery, W. J. (2017). Object-based convolutional neural network for high-resolution imagery classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(7):3386–3396.
- [175] Zhou, Y. T. and Chellappa, R. (1988). Computation of optical flow using a neural network. In *IEEE 1988 International Conference on Neural Networks*, pages 71–78 vol.2.
- [176] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint*.
- [177] Zitnick, C. L. and Dollár, P. (2014). Edge boxes: Locating object proposals from edges. In *European conference on computer vision*, pages 391–405. Springer.



## Biography

Nicola Felice Capece was born in Potenza, Italy, on 23rd April 1987. He received his master's degree in Computer Engineering from the Università della Basilicata, Italy, in July 2015. Currently, he is a PhD student in Computer Science at the Università della Basilicata (in collaboration with the Università del Salento) under the supervision of Dr Ugo Erra. He is a member of the CGLab (Computer Graphic Lab) research group at the University della Basilicata. His research interests include: *(i)* Computer Graphics: Real Time and Offline Rendering; *(ii)* Deep Learning Application in Computer Graphic fields; *(iii)* Virtual, Mixed and Augmented Reality (VR/AR); *(iv)* Human Computer Interaction (HCI). From 1st November 2017 to 30 April 2018 (six months) he carried out a visiting period at National Council of Research (CNR) in Pisa, Italy. In such period he was member of Visual Computing Lab (VGC) at the Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" under the supervision of Dr Roberto Scopigno. He has attended in several international conferences at which he presented different works, among them: "Information Visualisation, Lisbon 2016", "Signal Image Technology & Internet Based System, Naples 2016", "Eurographics Lyon 2017" and "Information Visualisation, Salerno 2018".

## Publications

### Computer Graphics

- Erra, U., Capece, N. F., and Agatiello, R. (2017). Ambient Occlusion Baking via a Feed-Forward Neural Network. In Peytavie, A. and Bosch, C., editors, *EG 2017 - Short Papers*. The Eurographics Association.
- Erra, U., Scanniello, G., & Capece, N. (2012, July). Visualizing the evolution of software systems using the forest metaphor. In *2012 16th International Conference on Information Visualisation* (pp. 87-92). IEEE.

### Deep Learning

- Capece, N., Erra, U., and Ciliberto, A. V. (2016b). Implementation of a coin recognition system for mobile devices with deep learning. In *2016 12th International Conference on Signal-Image Technology Internet-Based Systems (SITIS)*, pages 186–192.
- Capece, N., Erra, U., and Scolamiero, R. (2017). Converting night-time images to day-time images through a deep learning approach. In *2017 21st International Conference Information Visualisation (IV)*, pages 324–331.
- Capece, N., Banterle, F., Cignoni, P., Ganovelli, F., Scopigno, R., and Erra, U. (2019). Deepflash: Turning a flash selfie into a studio portrait. *arXiv preprint arXiv:1901.04252*.

### Virtual and Augmented Reality

- Capece, N., Erra, U., Romano, S., & Scanniello, G. (2017, June). Visualising a software system as a city through virtual reality. In

- International Conference on Augmented Reality, Virtual Reality and Computer Graphics* (pp. 319-327). Springer, Cham.
- Erra, U., & Capece, N. (2019). Engineering an advanced geo-location augmented reality framework for smart mobile devices. *Journal of Ambient Intelligence and Humanized Computing*, 10(1), 255-265.
  - Capece, N., Erra, U., & Grippa, J. (2018, July). GraphVR: A Virtual Reality Tool for the Exploration of Graphs with HTC Vive System. In *2018 22nd International Conference Information Visualisation (IV)* (pp. 448-453). IEEE.
  - Capece, N., Erra, U., & Romaniello, G. (2018, June). A Low-Cost Full Body Tracking System in Virtual Reality Based on Microsoft Kinect. In *International Conference on Augmented Reality, Virtual Reality and Computer Graphics* (pp. 623-635). Springer, Cham.
  - Capece, N., Agatiello, R., & Erra, U. (2016, July). A client-server framework for the design of geo-location based augmented reality applications. In *2016 20th international conference information visualisation (IV)* (pp. 130-135). IEEE.



# Index

- Accuracy Training, 162
- Accuracy Validation, 162
- Activation Function, 10, 60
- AdaGrad, 80, 164
- Adam, 80, 114, 164
- AlexNet, 38
- Ambient Occlusion, 41
- Artificial Intelligence, 1
- Artificial Neuron, 10
  
- Backward Propagation, 73
- Batch Normalization, 60, 79, 145
- Batch Size, 29, 165
- Bilateral Filter, 82, 83, 112, 141
- Bilinear Interpolation, 62, 168
- Binarization Problem, 135
- Binarization Process, 112
- Blur Problem, 140
- Bounding Box, 86
  
- Caffe, 23
- Chroma Key, 103
- Classification Problem, 4, 25
  
- Coin Recognition, 25
- Colours Perturbation, 34, 139
- Concatenation, 79
- Confusion Matrix, 116
- Contrast Normalization, 139
- Convolution, 15, 79
- Convolutional Neural Network, 12, 28, 58, 76
- Cropping, 86
- Cross-Entropy, 30, 108, 161
  
- Data Augmentation, 34, 67, 86, 114
- Dataset, 139
- Deconvolution, 18, 79, 107
- Deep Learning, 1, 9
- Deep Learning Toolbox, 24
- Degradation Problem, 60, 72
- DenseNet, 144, 145
- Depth-Buffer, 54
- Discriminator Network, 145
- Downsampling, 59

- Dropout, 142, 165
- Encoder-Decoder, 19
- Epochs, 29, 165
- Exploding Gradient Problem, 144
- Face Recognition, 86
- Feed-Forward Neural Network, 10, 45, 48
- Flash Selfie, 75, 77
- Flipping, 34, 86, 139
- Forward Propagation, 72
- Fragment Shader, 54
- Generalization Level, 137
- Generative Adversarial Network, 94, 145
- Generator Network, 145
- Global Accuracy, 115
- GPU, 22
- Gradient Descent, 162
- Green Screen, 135
- HDRNet, 92
- Hidden Layers, 13
- Hyperbolic Sigmoid, 15
- Hyperparameters, 138, 165
- Image Distorsion, 139
- Image Processing Toolbox, 86
- Image Segmentation, 105
- Image-Matting Problem, 104, 133
- ImageNet, 38
- Intersection Over Union, 115
- JSON, 32
- Kernel, 14, 165
- $L^1$  Regularization, 142
- $L^2$ , 142
- Layer, 10
- Layer Group, 61
- Leaky Rectified Linear Unit, 15, 61
- Learning Rate, 71, 80, 114, 163, 165
- Lighting Scheme, 57, 75
- Local Receptive Fields, 13
- Loss Function, 83, 138, 162
- Loss Validation, 138
- Machine learning, 1
- MATLAB, 24, 114
- Max-Pooling, 18, 71, 107
- Mean Accuracy, 115, 123
- Mean Boundary F1 Score, 115
- Mean Squared Error, 63, 162
- Mean Subtraction, 85
- Mini-Batch, 163, 165
- Multilayer Perceptron, 11, 44
- Neural Network, 10
- Normal Distribution, 63, 80
- Normalized Exponential Function, 167

- NumPy, 24
- Nvidia DIGITS, 29
- Off-Line Rendering, 42
- Optimization Algorithms, 162
- Overfitting Problem, 141
  
- Peak Signal to Noise Ratio, 94
- Pix2Pix, 94
- Pooling, 18
- Pre-Trained Model, 63, 80, 166
  
- Real-Time Rendering, 43
- Rectified Linear Unit, 15, 61, 71, 84, 106
- Regression Problem, 4, 77
- Reinforcement Learning, 21
- Residual Function, 60
- Residual Learning Net, 60, 71, 144, 145
- REST, 31
- RMSProp, 80, 164
- Rotation, 34, 86, 139
  
- Scaled Conjugate Gradient, 165
- Screen Space Ambient Occlusion, 43
- Select Subject, 129
- Shortcut Connection, 72, 78, 144
- Sigmoid, 15, 62
- Skip-Connection, 72, 144
- Softmax, 108, 167
  
- Standard Deviation, 39, 60, 140
- Stochastic Gradient Descent, 39, 62, 163
- Stochastic Gradient Descent with Momentum, 163
- Stride, 14
- Structural Similarity Index, 88, 162
- Studio Portrait, 75, 77
- Style Transfer Method, 98
- Supervised Learning, 19, 58
  
- Tensorflow, 23
- Test Set, 29
- Theano, 24
- Torch7, 24
- Training Set, 29
- Transfer Learning, 80, 166
- Translation, 34, 139
- Trimap Image, 105, 133, 135
  
- U-Net, 17, 106
- Underfitting, 143
- Unpooling, 18, 107
- Unreal Engine 4, 67
- Unsupervised Learning, 20
  
- Validation Set, 29, 138
- Vanishing Gradient Problem, 72, 143
- VGG-16, 59, 62, 70, 80
  
- Weight Decay, 165

Weights, 10

Z-Buffer, 54

Xavier Initialization, 80, 166