







UNIVERSITÀ POLITECNICA DELLE MARCHE  
DEPARTMENT OF ENGINEERING  
MASTER'S DEGREE IN MECHANICAL ENGINEERING

---

**Sviluppo di un algoritmo di Deep-learning per il  
riconoscimento automatico dell'area inquadrata dalla  
telecamera di un dispositivo portatile per la  
rilevazione del Gap&Flush azionato da operatori  
umani**

**Deep-learning based automated recognition of the  
area framed by the camera of a hand-held gap and  
flush measurement device operated by human  
operators**

Candidate:  
**Samuele Calcabrini**

Advisor:  
**Prof. Paolo Castellini**

Coadvisor:  
**Ing. Nicola Giulietti**  
**Prof. Paolo Chiariotti**

Academic Year 2020-2021





UNIVERSITÀ POLITECNICA DELLE MARCHE  
DEPARTMENT OF ENGINEERING  
MASTER'S DEGREE IN MECHANICAL ENGINEERING

---

**Sviluppo di un algoritmo di Deep-learning per il  
riconoscimento automatico dell'area inquadrata dalla  
telecamera di un dispositivo portatile per la  
rilevazione del Gap&Flush azionato da operatori  
umani**

**Deep-learning based automated recognition of the  
area framed by the camera of a hand-held gap and  
flush measurement device operated by human  
operators**

Candidate:  
**Samuele Calcabrini**

Advisor:  
**Prof. Paolo Castellini**

Coadvisor:  
**Ing. Nicola Giulietti**  
**Prof. Paolo Chiariotti**

Academic Year 2020-2021

---

UNIVERSITÀ POLITECNICA DELLE MARCHE  
MASTER'S DEGREE IN MECHANICAL ENGINEERING  
DEPARTMENT OF ENGINEERING  
Via Brezze Bianche – 60131 Ancona (AN), Italy

*To my family, my girlfriend,  
my fellow students and my friends*





# Sommario

La *Quarta Rivoluzione Industriale* si basa sulla cosiddetta *Trasformazione Digitale*. La tecnologia ha fatto passi da gigante negli ultimi anni e questo ha portato alla creazione di vari dispositivi intelligenti. [1]

La spinta alla digitalizzazione arriva principalmente dal settore industriale; le aziende sono chiamate a competere in modi diversi, puntando non solo sulla qualità del prodotto, ma anche sull'innovazione e sui servizi.

L'impatto del processo digitale sui modelli di business è enorme.

Il modo in cui sia i clienti che i fornitori interagiscono tra loro sta cambiando rapidamente grazie alla sempre maggior implementazione di un processo produttivo articolato e automatizzato all'interno della catena di fornitura.

Prodotti e servizi stanno cambiando, rinnovandosi e riposizionandosi grazie all'inserimento di dispositivi al tempo stesso più funzionali e user-friendly messi a disposizione da una vasta gamma di applicazioni digitali.

Lo scenario competitivo e il modo in cui le imprese devono rispondere è in continua evoluzione; le aziende devono il loro successo principalmente all'efficienza della loro supply chain. Ciò ha portato ad un profondo cambiamento nella domanda in tutti i principali settori di utenza, dalla finanza all'industria, dalla distribuzione ai servizi di pubblica utilità.

Una domanda che ora è più che mai focalizzata sulle caratteristiche e sulle specifiche qualità di nuovi componenti, chiamati *Digital Enabler*, in quanto consentono all'utente di interagire e proiettarsi sempre più verso l'utilizzo di operazioni innovative nel campo della trasformazione digitale. [5]

In questo mondo pervasivo, dispositivi intelligenti come smartphone, veicoli di ultima generazione, smartwatch o un qualsiasi altro dispositivo *Internet of Thing (IoT)* stanno diventando onnipresenti e nella maggior parte dei casi riguardano la comunicazione e l'archiviazione di informazioni in specifici database chiamati *cloud*.

Con la proliferazione dei dispositivi *IoT* e l'avvento dei social media, viene generata un'enorme quantità di dati e la maggior parte di essi è non strutturata e multimodale. Questo improvviso aumento di scambio di una notevole quantità di informazioni ha comportato la necessità di un'accurata analisi dei dati generati, che a sua volta ha creato molte opportunità per i produttori e specifiche esigenze per gli utenti, che richiedono sempre più ampi spazi di archiviazione e un'accurata elaborazione dei dati.

Accanto alla proliferazione dei dati, un altro aspetto fondamentale e stimolante nel regno dell'*Industry 4.0* è la cosiddetta *Computer Vision*, che in gran parte rappresenta il futuro della controversa ricerca nel campo del *Machine Learning (ML)* e del *Deep Learning (DL)* - la base dell'apprendimento automatico che nel prossimo futuro potrebbe consentire alle macchine di prendere le proprie decisioni. Questo processo richiede tecniche computazionalmente sempre più efficienti per utilizzare questi dati in modo significativo. [1]

Nel campo del *ML* e del *DL*, un ruolo chiave è giocato dalle *Reti Neurali Artificiali*, le *Artificial Neural Networks*, modelli matematici che traggono ispirazione dalla funzione del cervello biologico. [14]

La maggior parte delle applicazioni di visione artificiale ruota attorno alle architetture delle *Reti Neurali Convolutionali*, le *Convolutional Neural Network (CNN)*, una tipologia di rete neurale artificiale basata sul sistema di visione degli esseri umani. [1] [14]

La visione artificiale gioca un ruolo fondamentale per i problemi di *Classificazione*, che compaiono quando la rete deve essere addestrata a riconoscere le varie classi di dati che vengono fornite in input. Un modello di classificazione si basa principalmente su tre set di dati: il *Training set*, in cui si trovano le immagini utilizzate per addestrare la rete; il *Validation set*, che contiene le immagini usate per valutare le prestazioni del modello e infine il *Test set*, con le immagini usate per testare il modello e valutarne la capacità di generalizzazione. In questo contesto, si inserisce il progetto *GOODMAN* (<http://goodman-project.eu>), un progetto europeo che consiste nello sviluppo di un dispositivo portatile intelligente, dotato di un laser scanner usato per la misurazione del *Gap & Flash*, il (*G3F*). Il dispositivo *G3F* è destinato all'uso in una linea di produzione automobilistica per il controllo di qualità. Puntando il laser sui punti di intersezione tra le diverse parti di un'auto, il *G3F* restituisce la misurazione del *Gap & Flash* come output, consentendo di valutare eventuali difetti presenti nei veicoli durante il processo produttivo.

Poiché il *G3F* si basa su un sensore laser a triangolazione, questo studio esamina e produce un'architettura di classificazione delle immagini basata sulle *Reti Neurali Convolutionali (CNN)*.

Il sistema permette di riconoscere diversi colori e i vari componenti di una determinata auto.

In questo modo è possibile regolare la potenza del laser in base alle diverse tipologie di colori e materiali per ottenere un miglior risultato di acquisizione. Lo studio è stato affrontato cercando, in primo luogo, di risolvere un problema di *classificazione binaria* connesso a due diversi tipi di materiali, che ha portato all'identificazione di due classi, denominate *lamiera-lamiera* e *lamiera-fanale*. E' stata eseguita l'ottimizzazione degli *iperparametri* della rete al fine di poter identificare i valori e i parametri fondamentali per ottenere una migliore accu-

ratezza della rete neurale.

Dopo aver valutato i risultati della *classificazione binaria*, si è andati quindi ad affrontare le problematiche legate al modello *multiclasse*, alimentando la rete neurale con un set di dati costituiti da 40.000 immagini che sono servite per addestrare la rete a riconoscere 10 diverse combinazioni di colori e parti di auto, che hanno portato alle seguenti 10 classi: *bianco-bianco*, *bianco-fanale*, *nero-nero*, *nero-fanale*, *rosso-rosso*, *rosso-fanale*, *grigio-grigio*, *grigio-fanale*, *grigio scuro-grigio scuro* e *grigio scuro-fanale*. L'ottimizzazione degli *iperparametri*, effettuata per la *classificazione binaria*, ha aiutato ad affrontare anche il problema della *classificazione multiclasse*.

I risultati sono stati analizzati valutando l'*accuracy* e la *loss*, grazie alle quali è stato possibile valutare le prestazioni del modello.

Il presente lavoro è strutturato come segue:

- *Capitolo 1*: in questo capitolo introduttivo viene fatta una panoramica dell'*Industria 4.0*, concentrandosi principalmente sul ruolo della misurazione; alla fine del capitolo viene presentato il dispositivo *G3F* insieme agli obiettivi e alle finalità di questo studio;
- *Capitolo 2*: questo secondo capitolo descrive in dettaglio le tecnologie all'avanguardia disponibili in letteratura, quali *Sistemi di visione*, *Machine Learning* e *Deep Learning* per la *Computer Vision*;
- *Capitoli 3*: il terzo capitolo spiega il lavoro svolto; elenca ed elabora integralmente le varie architetture che sono state implementate per la realizzazione di questo specifico progetto;
- *Capitolo 4*: il capitolo conclusivo riassume lo studio e il lavoro complessivo insieme ad una valutazione finale dei risultati raggiunti e una panoramica sui possibili futuri sviluppi del progetto in questione.

Una volta effettuate varie prove al fine di trovare la combinazione di *iperparametri* in grado di restituire il miglior risultato possibile, è stato ottenuto un classificatore capace di distinguere 10 differenti classi con un'*accuracy* prossima all'unità nella fase di *training* e pari a circa 0.94 per la *validation*. Per quanto riguarda la *loss* invece, si ha un valore prossimo allo zero per l'*addestramento*, e un valore attorno allo 0.3 durante la *validazione*.

Questa leggera perdita di accuratezza del modello tra la fase di *training* e di *validazione* è però in parte compensata dal fatto che il classificatore tende a confondere le auto bianche da quelle grigio chiare, così come ha difficoltà nel distinguere una macchina nera da una grigio scura (distinzione a volte impossibile anche all'occhio umano). Al fine dello scopo con cui è stato svolto il lavoro,

risulta comunque irrilevante per il *G3F* trovarsi di fronte ad un'auto bianca o grigio chiara, in quanto la potenza che dovrà essere emessa dal sensore laser a triangolazione al fine di avere un'acquisizione il più possibile accurata, è per lo più identica; discorso analogo può essere fatto per auto nere e grigio scure. Il lavoro eseguito è indice del valore aggiunto che gli algoritmi di *Intelligenza Artificiale* sono in grado di apportare al settore industriale, favorendo una maggiore velocità ed una miglior accuratezza delle analisi, con un guadagno in termini di tempistiche durante la fase produttiva e una importante riduzione degli sprechi, fattori chiave nel mondo dell'*Industry 4.0*.

# Abstract

The *Fourth Industrial Revolution* is based on the so called *Digital Transformation*. Technology has advanced by leaps and bounds over the past few years, and this has led to the creation of various smart devices.[1]

The push for digitalization comes mainly from the industrial sector; companies are called to compete in different ways, focusing not only on the quality of products, but also on services and innovation.

The impact of digital processing on business models has been huge.

The way in which both customers and suppliers interact is changing at a very fast pace thanks to the ever increasing implementation of articulated and automated process within the supply chain.

Products and services are changing, renewing and repositioning themselves thanks to the inclusion of features that are both more functional and user-friendly which are made available by a plethora of digital applications.

The competitive scenario and the way in which business have to respond is evolving constantly; companies owe their success predominantly to the efficiency of their supply chain. This brought about a profound change in demand in all major user sectors, from Finance to Industry, from Distribution to Utilities.

Demand is now more focused than ever on the specific features and qualities of innovative components, such as *Digital Enabler*, which enable the user to interact in a better way and perform innovative operations in the field of digital transformation.[5]

In this pervasive world, smart devices such as smartphones, last-generation vehicles, smartwatches, or any *Internet of Thing (IoT)* devices are becoming ubiquitous and in the majority of cases involve communicating and storing information with huge database centers called *cloud*.

With the proliferation of *IoT devices* and the advent of social medias, a huge amount of multimedia data is being generated and most of it is unstructured and multimodal. This sudden rise in the exchange of huge amounts of information has meant a sudden rise of demand for computation of multimedia data which has in turn created many opportunities for producers and special needs for the users who increasingly require larger storage spaces and accurate data processing.

Alongside the proliferation of data, another challenging aspect in the realm of the *4.0 industry* is the so called *Computer Vision* which in large part represents

the future of the controversial *Machine Learning (ML)* and *Deep Learning (DL)* research - the basis of automatized learning that could allow in the near future machines to take their own decisions. This process will require various computationally efficient techniques to make use of this data in a meaningful manner.[1]

In the field of *ML* and *DL* there are *Artificial Neural Networks*, mathematical models that draw inspiration from the function that make a biological brain work.[14]

Most of the *computer vision* applications revolve around *Convolutional Neural Network (CNN)* architectures, a typology of *artificial neural network* based on the vision system of human beings. [1] [14]

*Computer Vision* plays a fundamental role for *Classification* issues, which appear when the network is to be trained to recognize the various classes of data that are provided during the input process. A *classification model* is mainly based on three sets of data: the *Training set*, in which images are situated and used to train the network; the *Validation set*, which contains the images used to evaluate the model performance and finally the *Test set* where the images are located to test the model.

The *GOODMAN project*, a European project which regards the development of a smart portable laser scanner for *Gap&Flash* measurement (*G3F*), fits well into this context. The *G3F* device is intended to be used in an automotive production line for quality control. By aiming the laser at the intersection points between different parts of a vehicle, the *G3F* returns the measurement of the *Gap&Flash* as output, allowing to evaluate eventual defects present in the cars during the production process.

Since the *G3F* is based on a triangulation laser sensor, this study examines and produces an architecture of *image classification* based on the *Convolutional Neural Network (CNN)*.

The system allows to recognize different colours and the various components of a vehicle.

In this way it is possible to adjust the laser power according to the different typologies of colours and materials in order to obtain a better acquisition result. We begun our study by trying to address a *binary classification problem* connected with two different types of materials, which led to the identification of two classes, named *sheetmetal-sheetmetal* and *sheetmetal-headlight* . The optimization of the network *hyperparameters* was carried out to identify the values and parameters which are fundamental to obtain a better accuracy of the neural network.

After evaluating the results of the *binary classification*, we then proceeded to address issues connected to the *multiclass model*, by feeding the *neural network* with data made up of 40,000 images which served to train the network to rec-

ognize 10 different combinations of colours and parts: *white-white*, *white-light*, *black-black*, *black-light*, *red-red*, *red-light*, *grey-grey*, *grey-light*, *dark grey-dark grey* and *dark grey-light*, where the colours represented the sheet-metal colour scheme of the car and the *light* term indicated the headlight of the vehicle. The optimization of *hyperparameters*, carried out for the *binary classification* also helped to address the *multiclass problem*.

The results were analyzed by assessing the *accuracy* and the *loss* factors, by which it was possible to evaluate the performance of the model.

The present work is structured as follows:

- *Chapter 1*: in this introductory chapter, an overview of *Industry 4.0* is made, focusing mainly on the role of the measurement; at the end of the chapter the *G3F* device is presented together with the objectives and purposes of this study;
- *Chapter 2*: this second chapter describes in detail the state-of-the-art technologies currently available, such as the *Vision system*, *Machine Learning* and *Deep Learning* regarding the *Computer Vision*;
- *Chapters 3*: the third chapter explains the work done; it lists and fully elaborates the various architectures which have been implemented for this specific project;
- *Chapter 4*: the conclusive chapter sums up the project which has been carried out together with the achieve results and it ends with an overview on possible future developments.





# Contents

<b>1. Introducing issues connected to the Industry 4.0</b>	<b>1</b>
1.1. Introduction to Industry 4.0 . . . . .	1
1.2. The role of measurements and sensors in Industry 4.0 . . . . .	4
1.3. The G3F device and the GOODMAN project . . . . .	6
1.4. Thesis purpose . . . . .	8
<b>2. State of the Art technology</b>	<b>11</b>
2.1. Vision systems . . . . .	11
2.1.1. Optics designed for vision systems . . . . .	13
2.1.2. Formation of images . . . . .	17
2.1.3. Lighting . . . . .	21
2.1.4. Software tools for vision systems . . . . .	23
2.2. Machine Learning and Deep Learning for Computer Vision . .	27
2.2.1. Introduction to Artificial Intelligence (AI) and Machine Learning (ML) . . . . .	27
2.2.2. Introduction to Deep Learning for Computer Vision . .	32
2.2.3. Artificial Neural Networks . . . . .	33
2.2.4. Convolutional Neural Networks (CNN) . . . . .	50
<b>3. Materials and methods</b>	<b>57</b>
3.1. Introduction to the project . . . . .	57
3.2. Binary classification . . . . .	58
3.2.1. Multiple architectures with an online cats and dogs dataset	58
3.2.2. Binary classification with VGG16 architectures . . . . .	65
3.3. Multi-class classification . . . . .	69
3.3.1. VGG16 base model . . . . .	71
3.3.2. Inception models . . . . .	75
<b>4. Results and conclusion</b>	<b>79</b>
<b>A. Results tables</b>	<b>89</b>



# List of Figures

1.1. Gap & Flush definition. Source:[7]	7
2.1. Representation of a color image. Source:[13]	12
2.2. Formation of an object through a converging lens. Source:[13]	17
2.3. Example of artificial neural network. Source:[18]	34
2.4. Sigmoid Function. Source:[19]	45
2.5. Tanh Function. Source:[19]	46
2.6. Relu Function. Source:[19]	47
2.7. Leaky relu Function. Source:[20]	47
3.1. VGG-16 architecture. Source:[21]	58
3.2. VGG Nets of Various Depth. Source:[21]	59
3.3. Inception module. Source:[22]	60
3.4. Fully connected layer vs global average pooling. Source:[23]	61
3.5. GoogLeNet architecture. Source:[24]	61
3.6. Dataset images used for binary classification	66
3.7. Red car images dataset	70
3.8. Comparison of white and light gray car dataset images	70
3.9. Comparison of black and dark gray car dataset images	71
4.1. Model loss and accuracy	82



## List of Tables

2.1. Types of LUT. Source:[13] . . . . .	24
4.1. VGG16 results for cats and dogs dataset . . . . .	79
4.2. VGG16 results for binary classification . . . . .	80
4.3. VGG16 results for multiclass classification . . . . .	81
4.4. InceptionResNetV2 results for multiclass classification . . . . .	81
4.5. InceptionV3 results for multiclass classification . . . . .	82
4.6. Probability matrix for test set (Images from 96 to 101) . . . . .	84
4.7. Probability matrix for test set (Images from 445 to 450) . . . . .	84
A.1. Xception results for cats and dogs dataset . . . . .	89
A.2. InceptionResNetV2 results for cats and dogs dataset . . . . .	90
A.3. InceptionV3 results for cats and dogs dataset . . . . .	90
A.4. ResNet50 results for cats and dogs dataset . . . . .	91



# Chapter 1.

## Introducing issues connected to the Industry 4.0

### 1.1. Introduction to Industry 4.0

The *Digital Transformation* is taking center stage in the fourth industrial revolution. The term *Industry 4.0* refers to the industrial system transformed by digital technology. An ecosystem made of state-of-the-art factories, machines and objects capable of dialoguing with each others, aware of topical environmental issues and capable of interconnecting with the final consumer.

The *Digitalization* is radically changing the production and service sectors and the way business is carried out. This phenomena is also modifying the ways in which consumers and suppliers interact with each other.

In a relatively short period of time this new type of *digital transformation* has been almost universally embraced on a daily basis not just by those businesses which need to use state of the art technology but also by companies which operate in more traditional business sectors not normally associated with hi-tech gadgets, innovative machines and interconnected devices.

As a result an increasingly pressing demand for new skills is arising.

The *Digital Transformation* requires the use of more sophisticated technologies and higher throughput of services for *Information Technologies (IT)*. This has led to an increase of interest for tools that are extremely useful for the advanced analysis of *Big Data* within the predictive analytics optics. In recent time there has also been an considerable and rapid rise in *Cognitive Learning* and *Machine algorithms Learning*.

Advanced Manufacturing, the Internet and the Industrial Robotics sectors have now become integral paradigms in the *Iot* within the *Fourth Industrial Revolution*.

We are now witnessing a shift towards a "service economy" through the adoption of new flexible systems, capable of supplying valuable ethereal components dedicated to the improvement of commercial digital transactions during the buying and selling of goods regarding both the suppliers and the final consumers.

## Chapter 1. Introducing issues connected to the Industry 4.0

The *Digital Transformation* involves a number of fundamental changes that are taking place in the technological, cultural, organizational, social, creative and managerial areas.

The changes brought about by the use of innovative state-of-the-art technology by placing *IT* at the forefront of many businesses thanks to the *digital transformation* process are many and can be summarized as follows:

- improvement of business performances, thanks to the implementation of innovative systems conceived as strategic lever for cost containment leading to eventually better profit margins;
- increased visibility of businesses in the market place;
- greater operational efficiency between different departments;
- a more cost effective production process;
- better and safer conditions for the personnel working in the production process;
- greater efficiency in the way the industrial space is planned and used;
- a more rapid and efficient identification of new opportunities in collateral business segments;
- the development of a quicker and more effective way of finding new potential geographic markets thanks to the use of dedicated apps;
- a substantial improvement of customer relations and management.

To successfully build upon the many changes brought about this epochal transformation, shifts in cultural and technological attitudes will need to take place hand in hand. Those who are involved in the production process in organized structures particularly at managerial level will have to be willing to embrace the technological changes wholeheartedly throughout the entire structure, positively encouraging interaction and collaboration between different departments in order to obtain the highest level of technological integration throughout the entire system.

A fresh approach to innovation is therefore needed. This will mean a complete new way of thinking about technology and its implementation at all levels from the humble operator to the decision makers to allow *IT* to fully permeate the system in order to maximize efficiency.

Technology allows interconnection and cooperation between the workforce and all productive resources both within and outside the productive boundaries. The *fourth industrial revolution* is synonymous with connectivity between physical



### 1.1. Introduction to Industry 4.0

and digital systems and closely associated with complex analysis carried out by *Big Data* and in order to implement the necessary changes in real time. This revolution concerns the use of intelligent, interconnected and permanently on line machines.

Despite the obvious difficulties the contemporary world is experiencing in implementing the new *IT systems*, the unstoppable advent of the so-called *fourth industrial revolution*, is nevertheless bringing about a profound transformation in the mechanisms by which productive systems have managed to create up to now, at least in the western world, goods and services that have radically transformed our lives in the last couple of centuries.

The *fourth industrial revolution*, made possible by the availability of sensors and low-cost connections, is associated with an increasingly pervasive use of data and information, computational and data analysis technologies, new materials, automated, digitized and connected machines, components and systems (*internet of things* and machines).

Within the realm of the *4.0 revolution*, it is now possible to manage real networks that incorporate, integrate and connect machinery, installations and production facilities, logistics, warehousing system and distribution channels.

Thanks to this *digital transformation* and the use of cyber physical production systems, manufacturing sites are able to react virtually in real time to changes in demand, product specifications and flows procurement of raw materials, optimizing the processes of adaptation to new requirements. Margins of errors can also be considerably reduced while improving delivery lead time through engineering data control. Increased flexibility, higher speed and improved accuracy are fundamental gains normally associated with the introduction of *4.0 technology* within the operational procedures.

The scope of a systematic introduction of *Industry 4.0* is a conceptual one and goes well beyond the individual company just as it goes beyond the individual machine. *Innovation 4.0* is not just about putting state-of-the-art devices and machines at the forefront of the production process, but to properly integrate and combine people, information systems and operational procedures harmoniously through a number of dedicated technologies so that the entire system can benefit from it. An intelligent approach to *4.0 technology* should eventually lead to better services, safer and more pleasant working environments and ultimately to the manufacturing of smarter and cheaper products as well.

Actors within the supply chain who until recently have played a relatively marginal role in the determining the nature, the characteristics and prices of certain services or products have suddenly taken centre stage in the entire process: the accurate analysis and the response to individual, specific needs of the discerning customer has now become fundamental in the manufacturing process in order to satisfy demand in the best and fastest way possible; thanks to the

application of technological programmes, it is now possible to shift production rapidly in order to implement mass customization rapidly.

Suddenly it is now becoming possible to vary the type of production in order to respond to the changes in demand or type of product, within a logic of modularity and continuous re-configurability. There has been a significant impact in terms of sustainability too, particularly with reference to aspects related to recycling, workplace safety, the optimization in the use of raw materials and consumption of energy. The implementation of the so called circular business models, are also useful to eliminate whenever possible emission leakage and reduce resource inputs, while minimizing waste during the operational procedures via the achievement of a *zero defect production*.

All of this has led to an increase of performance in the productive process and a better interaction between man and machine thanks to a dedicated technology which reduces considerably errors and accidents, while helping to improve safety within the working environment.

Production systems that support and assist operators while performing their duties, lead to a reduction in stress related work. They are also extremely useful in helping with the organization of the workforce and personnel by using dedicated programmes which optimize the duties that need to be assigned in specific areas. Other important advantages concern the assignment of less pleasant tasks to automatized machines and the integration of workers with disabilities.

Furthermore the *4.0 transformation* thanks to the application of digital technologies makes possible in most cases to implement a rapid shifts in the production of goods and in the offer of services to better satisfy the sudden changes in market demand. [5]

## 1.2. The role of measurements and sensors in Industry 4.0

A measure is a type of information which consists of a number, an *uncertainty* and a unit of measurement which is assigned to represent a parameter in a given system state.

To measure means to transduce a physical phenomenon into information.

A measurement process is a process where an object in input, named *measurand*, which is related to a physical quantity is measured at a point in time when the moment is enters the measurement system. It will be returned by the system as output data, a quantity that is correlated to the input quantity.

Measurements, particularly those of mechanicals quantities, have always been

## 1.2. The role of measurements and sensors in Industry 4.0

the basis of human knowledge and represented scientific and technical progress. In the engineering field measuring is fundamental for verifying the performance of buildings, structures, driving and operating machines, land, marine and air vehicles, and is also use for quality control checks during the production of industrial goods.

For this reason, in professional activities, as well as in everyday life, humans have to deal constantly with many measuring instruments.

In most cases, the most important aspects deriving from the technical measurements is what can be deduced and extrapolated from those figures. One fundamental aspect to obtain reliable information from the measurements, is to minimize the *uncertainty factor* which plays a big part in the result itself. [4] All the scenarios regarding *Industry 4.0* address the concepts of data and information. Quantitative data about physical quantities originate from measurement. This is why measurement plays a fundamental role.

Measuring instruments, due to their nature and to their complex interaction with the *measurand* and the measurement environment, produce data that is always intrinsically affected by *uncertainty*. Measurements are made to make decisions. *Uncertainty of measurement* affects the level of confidence in decisions made on the basis of uncertain data; minimizing *uncertainty* is therefore essential to guarantee the quality of the data; this implies the ability to understand and to manage the entire measurement process.

In the contemporary manufacturing sector, a large variety of production environments have now specific arrangements for the various stages of the manufacturing process. (Multi-Stage)

Highly automated sectors now use a number of dedicated sensors placed along the production line, which are used to ensure the correct implementation of the manufacturing process and to guarantee through quality checks the best possible results.[6]

According to the *Industry 4.0* paradigm, *Zero Defect Manufacturing (ZDM)* is one of the pillars of the modern digital production process. A *zero-defect strategy* requires accurate measuring of data during the process and from the product itself and a close integration of the manufacturing methods and quality control.

In particular, one of the fundamental strategies to approach *zero-defect manufacturing* is the application of in-line quality control covering the production process in its entirety. In multi-stage production systems, the earlier a defect is detected, the better: in fact, early identification of non-conformity, performed at single process level on parts and sub-assemblies, prevents further defects to propagate to down-stream processes. Quality control is a decision-making process, aimed to asses if a part or a product complies to the specifications required; this is done by measuring specific characteristics and comparing them

to the requirements set at the design phase. The outcome is a diagnostic judgment. In such process, measurement *uncertainty* plays a fundamental role; if data is uncertain, the following diagnosis will be uncertain too.

*Uncertainty* of measurement is therefore a central concept in the digitalized manufacturing process. In a real production scenario, however, *uncertainty* of measured data can be an issue, mainly due to the possible variation of the harsh environmental conditions in which measurements take place.

The various structures located in strategic areas which are dedicated to the manufacturing process are designed and implemented in multi-stage production lines, with the purpose to integrate process and quality control.

Such schemes include smart *Quality Control Stations (QCS)*, which are designed to exhibit real-time adaptive behaviours, in order to keep measurement *uncertainty* under control even in the presence of variations of process/product parameters; they can implement data pre-processing to collect synthetic quality indicators, self-diagnosis and self-calibration to maximize the confidence level of the sensors output. Even in industrial plants with a high level of automation, operators can still play a fundamental role. For a variety of technical reasons and in order to keep occupational levels at an ethically and economically acceptable level, highly specialized personnel, is often in charge of sometimes complex operations, that might include the measuring of certain parameters. Indeed, *humans in the loop (HITL)* not only perform production tasks but are involved in all aspects of associated decision making, essentially contributing with calibrated but autonomous decisions to the production process; when performing quality control tasks, human autonomous contribution can be considered a *QCS*. The roles and relevance of *humans-in-the-loop* are fundamental. One of the main roles of human agents is data acquisition, sensing and communication. When operators are "in the measurement loop process", specific problems arise, mainly related to human behaviours and dexterity, factors which may have an important impact on measurement *uncertainty*; indeed, in these conditions, measurement *uncertainty* depends upon the level of accuracy of the measuring instruments and the dexterity of the human operator. This raises questions like ergonomics, operator training, design of human-machine interface, etc. as well as operator-dependent measurement *uncertainty*. [7]

### 1.3. The G3F device and the GOODMAN project

In modern digital manufacturing plants, the implementation of 100% 'in-line quality control' during the production process is a strategic key employed to target a *zero-defect manufacturing (ZDM)*. This is particularly relevant in a multi-stage production, where the overall system performance depends on

### 1.3. The G3F device and the GOODMAN project

each single process and on the interaction between the various stages of the manufacturing process. In fact, actions performed in real time and the early identification of deviations and trends, performed at a single process level, considerably prevents the generation of defects and their propagation to the down-stream processes, enabling the entire manufacturing system also to self-adapt to different conditions. However, a *zero-defect strategy* requires reliable information to support any decision. Furthermore obtaining reliable information also requires accurate data gathered from the manufacturing process and the product itself, a close integration between various production stages and quality control and the capability to keep measurement *uncertainty* under control.

These concepts are the basis of a recent European project, the *GOODMAN project*, which implements a 'distributed system architecture' on agent-based *Cyber-Physical Systems (CPS)* and smart in-line quality control systems designed to exhibit real-time adaptive behaviours, in order to keep measurement *uncertainty* under control even in the case of variations of process/product parameters and in the presence of disturbances.[10]

The goal of the *GOODMAN project* was to develop a multi-stage production management methodology and system architecture that can guarantee high quality products without interfering, while improving the production efficiency of the entire system. The *GOODMAN project* constitutes a real implementation on a worldwide scale of the *Industry 4.0* paradigm, through the integration and convergence of technologies for measurement and quality control, for data analysis and knowledge management, at single process and at factory level.[9] The automotive industry is a relevant example of multistage manufacturing targeting zero-defect objective. Within the automotive industry, the car-body assembly represents a key sub-set of multi-stage process. In this process, for instance, checking *gaps and flushes* among adjacent surfaces is of uttermost importance for both aesthetic and functional purposes. Indeed, these quantities are measured through the whole car body assembly chain.

We define *Gap* the space between two opposite surfaces, measured along the tangent plane on the surfaces considered. *Flush* is the mismatch between two surfaces, i.e. the distance between two adjacent surfaces measured in the orthogonal direction to the tangent plane on the surfaces in exam (Figure 1.1).

Depending on the automation level of the assembly process, gap and flush

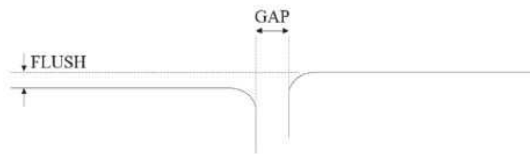


Figure 1.1.: Gap & Flush definition. Source:[7]

can be measured automatically - in dedicated measurement areas, exploiting non-contact methods based on optical triangulation - or manually by operators, who exploit feeler gauges and dial gauges. However, in the latter situation, no continuous data acquisition can be obtained, so no recorded data can be obtained at the end of the inspection process.

Contrarily, compliance to the *Industry 4.0* paradigm requires continuous data availability and the possibility to push information gathered from the line to the plant middleware. As a consequence, it becomes important to substitute conventional manual devices with new instrumentation, in the form of the *OT (operational technology)*, continuously connected to the *IT* level of the plant. In this situation, operators assume new responsibilities, becoming direct actors of the whole data acquisition chain. Indeed, when the measurements are performed by human operators, the *uncertainty* associated to the measurement system should be combined with the *uncertainty* resulting from the actions and behaviour carried out by the operators. Since the human *uncertainty* factor can at times be at higher than that associated to the automatized measurement system, it becomes clear of the need to develop dedicated measurement devices in order to implement associated strategies that would ease the operator during his tasks and would also reduce the overall *uncertainty* of the measurement.[10] The *GOODMAN PROJECT* has therefore designed, devised and implemented and built a hand-held, non-contact laser scanner, the *G3F*.

The *G3F* [7], [8], [9], [10] is a device that measures *gap and flush* during the car body assembly; thanks to its compact size, it fits in a smart phone cover: an InfraRed distance sensor, a laser-line projector and a camera, are assembled together in order to create a triangulation system. Data acquisition and additional sensor devices are performed and provided by a Raspberry Pi. A laser beam is projected orthogonally both at the surface and at the gap between two adjacent car parts. Then a picture of the projection is taken and, based on the laser profile, the *gap and flush* are measured.

## 1.4. Thesis purpose

A challenging fundamental aspect in the realm of the *4.0 industry* is the so called *Computer Vision* which in large part represents the future of controversial *Machine Learning (ML)* and *Deep Learning (DL)* research - the basis of automatized learning that could allow in the near future machines to take their own decisions. This process will require various computationally efficient techniques to make use of this data in a meaningful manner.[1]

In the field of *ML* and *DL* there are *Artificial Neural Networks*, mathematical models that draw inspiration from the function that make a biological brain work.[14]

#### 1.4. Thesis purpose

Most of the *computer vision* applications revolve around *Convolutional Neural Network (CNN)* architectures, a typology of artificial neural network based on the vision system of human beings. [1] [14]

*Computer Vision* plays a fundamental role in addressing *Classification* problems, which appear when the network is to be trained to recognize the various classes of data that are provided during the input process. A *classification model* is mainly based on three sets of data: the *Training set*, in which images are placed in order to train the network; the *Validation set*, which contains the images used to evaluate the model performance and finally the *Test set* which contains the images used to test the model.

The *GOODMAN project*, a European project that consists of a development of a smart portable laser scanner for *Gap&Flash* measurement (*G3F*), fits perfectly into this context. The *G3F* device is intended to be used in an automotive production line for quality control. By aiming the laser at the intersection points between different parts of a car, the *G3F* returns the measurement of the *Gap&Flash* as output, allowing to evaluate eventual defects present in the vehicles during the production process.

Since the *G3F* is based on a triangulation laser sensor, this study examines and produces an architecture of image classification based on the *Convolutional Neural Network (CNN)*.

The system allows to recognize different colours and the various components of a car.

In this way it is possible to adjust the laser power according to the different typologies of colours and materials in order to obtain a better acquisition result. We begun our study by trying to address a *binary classification problem* connected with two different types of materials, which led to the identification of two classes, named *sheetmetal-sheetmetal* and *sheetmetal-light*. The optimization of the network *hyperparameters* has been carried out in order to identify the values and parameters which are fundamental to obtain better accuracy of the neural network.

After evaluating the results of the binary classification, we then proceeded by addressing issues connected to the *multiclass* model, by feeding the neural network with a bank made up of 40,000 images which served to train the network to recognize 10 different combinations of colours and parts: *white-white*, *white-light*, *black-black*, *black-light*, *red-red*, *red-light*, *grey-grey*, *grey-light*, *dark grey-dark grey* and *dark grey-light*, where the colours represent the sheet-metal colouring of the car and the *light* term indicates the headlight of the vehicle. The optimization of the *hyperparameters*, carried out for the binary classification has also helped to address the *multiclass problem*.

The results were analyzed by assessing *accuracy* and *loss*, by which it was possible to evaluate the performance of the model.

*Chapter 1. Introducing issues connected to the Industry 4.0*



## Chapter 2.

# State of the Art technology

This chapter provides an overview of the tools used to achieve the project purposes set out in the first chapter.

Firstly, we are going to analyze in detail the vision systems currently in use in this kind of technological process, such as the *G3F*, an instrument that permits the acquisition, through a triangulation laser sensor, of images thus obtaining in output precise measurements of the *gap and flush*. A good knowledge of image acquisition and processing technologies is therefore fundamental.

Secondly, we will then proceed to analyze the technologies and tools that are available when interfacing with *machine learning* and *deep learning*. As stated in the first chapter, the work was carried out in order to create a *classification* architecture based on *artificial neural networks*. In order to have a global vision of the potential of these *artificial intelligence* tools, the various types of *machine learning* will be first described. We are then going to focus on *deep learning* for *computer vision*, based on the study and analysis of *artificial neural networks*. In particular, an important role in the project was played by *convolutional neural networks*, for which the main components of a general architecture will be described.

### 2.1. Vision systems

Images have been used for many years to collect data, but until recently there was no automated quantitative analysis, but only a qualitative analysis. Only after elements concerning image processing have been introduced then it has been possible for the first time to quantify observations. Today all this takes places automatically thanks to dedicated software.

*Vision systems* deal with images; these systems produce images from which the output quantity is obtained. The *vision systems* produces digital images of a scene via a process that involves several elements. Each single image contains vital information which is then extrapolated to collect data.

The extraction of information takes place through a process called image processing.

If you have distorted images or image that do not accurately represent a given scene, an *uncertainty* of measurement is produced. It is important to master the image detection. A good image acquisition process enables the user to acquire the correct information during the image processing phase.

An image is a source of multiple information; accuracy in the extraction of useful data during the acquisition phase is fundamental.

Images are, in other words, multidimensional data structures; a single image consists of a pixel matrix  $I_{ij}$ , when  $I$  is the intensity of the scene and  $ij$  is the pixel position.

A colour vision is obtained by superimposing the basic colors, red, green and blue (commonly known as the RGB colour model). Three different images are then superimposed to obtain a single color image. If we superimpose a time sequence on the image, we get a color video which is time dependent (Figure 2.1)

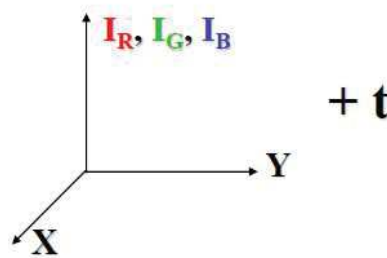


Figure 2.1.: Representation of a color image. Source:[13]

Each image is a scalar representation given by the sum of the intensity of the red, green and blue colours, which are represented in a two-dimensional domain to which time can possibly be associated. The data structure that is obtained when dealing with *vision systems* is therefore a two-dimensional representation of one or more scalars, depending on whether it is a colour or a black and white image. The scalars therefore represent the chromatic intensity of light.

As for the visual field concerning images, the  $I$  indicates the light intensity that occurs in that pixel. If, on the other hand, the images belong to the infrared range, the scalar band (in general 7-14  $\mu m$ ) still represents a luminous intensity but it isn't visible to the human eye.

The scalar size of the image can sometimes be something other than light, such as the rate of vibration of a particular point. The scalar size of the image depends on the sensor in use; in this case images may come from different tools (i.e. ultrasound, radar, etc.), that do not use light but acoustic or electromagnetic waves.

Images map a 3D object,  $O(x, y, z)$  into a two-dimensional domain, since the

image is the intensity  $I$  in space  $ij$ ,  $I_{ij}$ .

The elements that affect a given measurement system are normally optics, lighting, sensors, devices that convert analog data into digital and data processing. A calibration of the system in use is always required. Measurement uncertainty mainly depends on several factors, such as the geometry of the image, the light intensity and the frame acquisition, on which the uncertainty over time depends. When light interacts with objects or materials with different refractive indices, a series of events occur, such as reflection, diffusion, refraction and total internal reflection. [13]

### 2.1.1. Optics designed for vision systems

Light is an electromagnetic wave that travels in the form of a beam. In uniform refractive index materials, light travels in a straight line, creating a beam of light. When light interacts with objects or materials with different refractive indices, a series of events occur, such as reflection, diffusion, refraction and total internal reflection.

Reflection occurs when an incident beam that affects a specular surface at an angle  $\theta_i$  returns with an angle of  $\theta_r = \theta_i$  symmetrically with respect to the normal at the surface. When dealing with specular surfaces it is very difficult to acquire images as reflection phenomena normally occur.

Diffusion occurs when dealing with a rough surface; the radius incident on the surface is returned in space in all directions. The ideal diffuser model is a model that sees diffused light with a function:

$$I_{diff} \propto \cos \theta \quad (2.1)$$

when  $\theta$  is the angle that is formed with the normal at the surface.

The intensity of the scattered light, according to the ideal diffuser model, is maximum in the direction normal to the surface and is minimum in the direction tangent to the surface.

In reality, surfaces have characteristics that range from being mirrors to being diffusers. This behaviour varies as the roughness of the surface itself varies. In general, it is easier to obtain better images from diffusive rather than specular surfaces. If a surface has periodic roughness, such as parallel grooves, it is difficult to obtain good lighting. Roughness is a fundamental parameter. It is better to have a random roughness with a wavelength inferior to that of light to obtain a good diffuser.

Once a diffusive surface has been chosen, it is necessary to position the camera correctly in order to obtain the desired image.

Refraction, on the other hand, occurs when a material with refractive index  $n_i$  passes into a medium with refractive index  $n_t$ , different from  $n_i$ .

The phenomenon of total internal reflection occurs in prisms or optical fibers. [13]

As for the lenses, these are transparent optical elements consisting of refracting elements in which at least one surface is curved. A symmetrical lens might have spherical or aspherical surfaces with symmetry with respect to a common axis of revolution, called the optical axis.

Each lens has a beam with its own center; they are optical components that if crossed by light produce refraction. A lens is therefore characterized by refraction phenomena.

The optical axis is defined as the axis that unites the centers of the spheres with which the lenses are made; this axis corresponds to a reference line of the lens; typically a system with several lenses is made in such a way that all the lenses are aligned on the same optical axis.

There are different types of lenses; converging lenses (or positive lenses) are convex lenses, which surface is thicker in the center and thinner in the perimeter area. The beams of light which hit the surface parallel to the optical axis are deflected and made to pass through a single point which takes the name of *focal point*; the distance between the center of the lens and the focal point is called the focal distance,  $f$ .

Divergent lenses (or negative lenses) are lenses in which the thickness in the center is less than that of the edges. If I have a beam of light parallel to the optical axis, the beams are deflected and spread out in space. If the lens is well designed, the outgoing beams all appear to come from a point to the left of the lens and at a distance equal to the focal distance  $f$ .

To resume:

- A light parallel to the optical axis arriving in a converging lens is deflected onto the focus;
- A light that comes from a point of light that is on the focus, is transformed into a parallel beam of light;
- In a diverging lens, parallel beams of light are made to diverge from the focus itself;
- A light beam passing through the center of the lens is not deflected.

Real lenses behave differently from ideal lenses described above; what characterizes the real lenses from the ideal ones are the so called phenomenon of *aberration*, which is classified into spherical and chromatic. A spherical aberration is a phenomenon that occurs when, for example, we consider a spherical lens in which the light arrives from the left parallel to the optical axis and the beams focus in different points; the beams close to the optical beam have the

most distant focus, while the beams closer to the perimeter area of the lens end have the focus closest to the lens. The  $\Delta z$  interval between two focal points is called a *spherical aberration*. This phenomenon leads to blurry images. The issue aberration can be addressed with the use of complex mechanical devices. A reduction of the aperture might also reduce the aberration but causes also a reduction of the light exposure.

*Chromatic aberration* instead occurs with a variation of the refractive index of the wavelength of light. This aberration is evident when non-monochromatic light is used. This type of aberration can also be corrected with specific devices. Cylindrical lenses are lenses in which there is no rotational symmetry, i.e. they have a spherical profile in only one direction. They are characterized by the formation of a focal line. Cylindrical lenses can also be divergent or convergent and can be used to produce a flat beam of light, by adding a spherical lens before the cylindrical lens. Cylindrical lenses are also used to project laser lines (used in 3D scanners).

*Grin lenses* (graded index) are glass cylinders doped in order to have a parabolic refractive index higher in the center and lower at the edges. The light enters the lens in an axial direction and then bends, deviating its path. These type of lens as well as the cylindrical ones have in common a cylindrical shape which is one of the most common shapes obtained in modern mechanical process, making them fairly easy to produce.

Finally, in some applications it is necessary to obtain a separation between different types of lights (reflective and diffuse). In such cases, devices such *beam splitters* might be used; a thin layer of metal is applied into half of a surface of the splitter in order to obtain different reflective properties. In this way two different types of beams are obtained [13]

Light is commonly defined as a transverse electromagnetic wave, in which the electric field  $E$  and magnetic field  $B$  are orthogonal to the direction of propagation. The fundamental properties of light are:

- Wavelength,  $\lambda$ : period of oscillation of the electric field (or magnetic field);
- Phase,  $\phi$ ;
- Amplitude,  $E$ ;
- Polarization: position of the electric field vector.

The electromagnetic wave travels with velocity  $C = \lambda f$ , when  $f$  is the frequency.

A different wavelength is perceived by the human eye as a colour. The visible spectrum ranges from deep red ( $700nm$ ) to violet ( $400nm$ ); the human eyes are more sensitive to green ( $550nm$ ). The light intensity is characterized by the

equation  $I \propto |E^2|$ .

Light is not normally polarized, but there is also polarized light. Light is defined as polarized when the  $E$  field of the wave has a position which does not vary or which varies regularly over time. To better understand the polarization of light, it is useful to break down the electric field vector  $E$  in the two orthogonal directions to  $z$ :

$$E_x = |E_x| \sin(\omega t) \quad (2.2)$$

$$E_y = |E_y| \sin(\omega t + \phi) \quad (2.3)$$

$E_x$  and  $E_y$  are two sinusoids with different amplitude and phase.

- If the phase shift between  $E_x$  and  $E_y$  is zero, we have a horizontal polarization;
- If the phase shift is  $90^\circ$ , we have a circular polarization;
- If the phase shift is  $45^\circ$  we have an elliptical polarization.

As far as linear polarization is concerned, therefore, it occurs when the components of the electric field vector are in phase, that is when the two vectors move synchronously; the resultant is zero when both vectors are zero and the resultant is maximum when both vectors are maximum.

In the case of light with circular polarization, when one of the two components of the magnetic field is maximum the other is zero and vice versa; given that the magnetic field vector resulting from the two components  $E_x$  and  $E_y$  is a rotating vector, which can rotate clockwise or counterclockwise according to the signs of the phase shifts.

Regarding the elliptical polarization, when a non-polarized light hits a surface, the reflected light is generally polarized at a specific angle called *Brewster's angle*; at this angle there is a perfect polarization, which is however difficult to obtain in reality.

In general, an atomizer is an optical component that is transparent only at a certain position of the electric field, so it transmits only a component of the polarization. A polarizer is characterized by an axis along which light passes. It is known that two polarizers with crossed axes, that is two polarizers in which one is rotated  $90^\circ$  with respect to the other, do not allow light to pass.

Wave sheets (or retarders) are birefringent optical components, i.e. in which the refractive index is different on two perpendicular axes; light which has  $E$  along  $x$  has a different refractive index than that which has  $E$  along  $y$  ( $n_x \neq n_y$ ). This difference gives rise to a phase shift, which can be calculated as:

$$\Gamma = \frac{2\pi\Delta nL}{\lambda} \quad (2.4)$$

Polarization can be manipulated to eliminate reflections. A  $\frac{\lambda}{2}$  foil leads to a phase shift of half a wavelength ( $\pi$ ); the effect is the rotation of the polarization of deg 90. A  $\frac{\lambda}{4}$  foil leads to a phase shift of  $\frac{\pi}{2}$ ; the linear polarizations are modified in circular polarizations and vice versa.[13]

### 2.1.2. Formation of images

A converging lens has its own optical axis perpendicular to the plane in which the lens itself lies. If we consider a two-dimensional object located on a plane, and we want to see how this is represented, we can see from the Figure 2.2 how the lens is characterized by two foci, one upstream and one downstream.  $d_0$  is de object dimension, that is situated at a distance  $o$  from the lens.

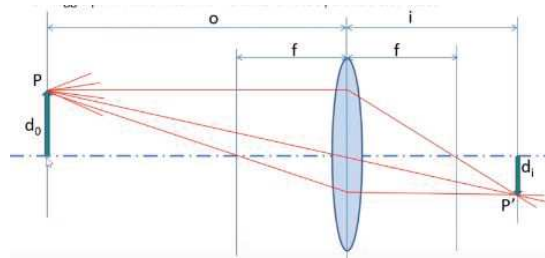


Figure 2.2.: Formation of an object through a converging lens. Source:[13]

If the object is illuminated, beams of light emanate from the object itself. There are also luminous objects that do not need to be illuminated. However, the light spreads in space and intercepts the lens. The hypotheses of a beam parallel to the lens, passing through the center and the front focus of the lens itself, are considered. The beams, starting from  $P$  and following different optical paths, all cross in  $P'$ .  $P'$  is the image of  $P$ . Similarly, by carrying out this operation for each point of the object  $d_0$ , the image  $d_i$  is obtained, which is formed at a distance  $i$  from the lens;  $d_i$  is smaller than  $d_0$  and is overturned. As the lens changes, the image observed also varies.

One of the most important choices that need to be made during this process to obtain the desired image is to use the most appropriate focal length. In general, a short focal length (wide angle) corresponds to a small image, as the lens is located near the sensor, in proximity to the back focus, so there is a high field of view and a high angle of view. A long focal length (telephoto) corresponds instead to a large image, due to the fact that the lens is placed far from the sensor, so that there is a reduced field of view and a reduced angle of view.

The choice of focal length is based on the equations of thin lenses:

$$\frac{1}{f} = \frac{1}{i} + \frac{1}{o} \quad (2.5)$$

where  $f$  is the focal,  $o$  is the distance between the lens and the object and  $i$  is the distance between the lens and the sensor.

$$M = \frac{d_i}{d_o} + \frac{1}{o} \quad (2.6)$$

where  $M$  is the transverse magnification, that represent the ratio between the image dimension,  $d_i$ , and the object dimension,  $d_o$ . With the equations 2.5 and 2.6 the first design of an optical system is realized. Typically  $d_i$  is fixed and depends on the sensor available, while  $d_o$  is chosen by the optical engineer instead.

Diverging (or negative) lenses also form images, but, unlike those formed with a converging lens, images formed with a diverging lens are generated in the front and are virtual images. Normally, negative lenses are not used individually for the realization of images, but are used by assembling more lenses together to form a single lens. In this type of lens there is a combination of lenses that is traced back to a focal length. [13]

With a real lens, however, a point does not become another real point due a number of issues that afflict a real lens such as the phenomena of diffraction. Light is a wave; if the wave starts from a point, the image that obtained won't be a real point but a so-called *Airy Disc*. In other words, there is a distribution of light intensity which has a non-point dimension, which, in optical terms, will be visible as a dot with various rings around it. The formation of the *Airy disk* is caused by diffraction, that is the anomalous behaviour of the light beam at the edges of the lens; when an electromagnetic wave passes through a slit, such as a lens, the light at the edges of the slit is affected by the diffraction phenomena, as light undergoes a curvature. The ray that starts from the starting point of the considered object, passing through the lens is diffracted, that is it branches out in several directions creating a series of constructive and destructive interferences. When dealing with a real lens, the image of a point can be calculated using the formula 2.7

$$r = \frac{1.22\lambda i}{D} \quad (2.7)$$

From the equation 2.7 it can be observed that the ability of a lens to focus is limited; indeed, being  $D$  the diameter of the lens, a larger lens has a better resolution, but considerable dimensions attract a number of issues such as practical use, storage and last but not least elevated costs of production. A lens is characterized by an impulsive response, which limits the optical resolution



and which corresponds to the spatial arrangement of the light that occurs in the image plane when the lens is stimulated by an object.

The depth of field is another fundamental parameter concerning the systems of vision, which serves to represent the distance interval  $\Delta_o$  within which it is possible to consider the object in focus. Each optical system finds a 3D scene in front of it.  $\Delta_o$  is acceptable when the blur caused by the geometric optics is less than, or equal to the blur that occurs during the diffraction phenomenon; in other words, the three-dimensionality of the image is tolerated as long as there is no greater blur than that obtained by diffraction.

The question is therefore to determine the distance interval  $\Delta_o$ , called depth of field (DOF), within which the objects appear sufficiently in focus. The dof changes by changing the aperture of the diaphragm, which is the useful diameter of the lens; the smaller is the aperture, the greater is the depth of field. If we consider the depth of focus  $\Delta_i$ , which represents the distance the sensor moves from the initial position, we have:

$$\Delta_i = 2.44\lambda\left(\frac{i}{D}\right)^2 \quad (2.8)$$

From the Formula 2.8 we can observe that with the same amount of  $\lambda$  and  $f$  the depth of focus varies as the diameter of the lens changes and in particular the smaller the diameter, the greater the depth of focus and the greater the ability to focus on a scene despite not having the sensor in the right position. Both the depth of focus and the depth of field depend on the ratio  $\frac{f}{D}$  where  $f$  is fixed while  $D$  is variable. The ratio  $\frac{f}{D}$  is the numerical aperture of the lens, a typical number of the lens,  $F\#$ . By decreasing the diameter of the lens, the depth of field and focus increase, the optical resolution decreases and the image becomes less bright. It is necessary to close the diaphragm just enough to obtain the right focus. If the objects are small you need a smaller depth of field.

There are lenses, called telecentric lenses, which deviate from the laws of optics of thin lenses and have a constant  $M$  magnification as the distance from the object varies. These lenses realize a parallel vision with zero angle of view, without perspective. With these lenses there is no need to correct dimensional errors. They are lenses made with two lenses separated by a distance equal to the sum of their focal lengths. [13]

Resolution is a property of the lens system that determines the ability to form images with small details. This leads to a certain degree of uncertainty for what concerns the accuracy of measurement. Resolution is limited by diffraction. In reality, diffraction can be seen and mathematically modeled as the limits that the optical system has in responding to incoming signals. There is a certain analogy with the impulse response.

Chapter 2. State of the Art technology

Being  $F\# = \frac{f}{D}$  and  $d = 2.44\lambda\frac{d}{D}$  you have:

$$d = 2.44\lambda F\# \tag{2.9}$$

From the equation 2.9 we deduce that the diameter of the dot image  $d$  grows by reducing  $D$  and then closing the aperture. If you want to have images made up of points of smaller diameter, in order to obtain more accurate details, it is necessary to use large diameter lenses; for practical reasons, however, it is impossible to have excessively large diameter lenses. To have a better resolution you have to open the diaphragm, but this leads to a penalty of the depth of field; it is therefore necessary to use the best combination obtainable taking into consideration the application in used in order to obtain the best result.

The *MTF* (*Modulation Transfer Function*) arises from the following analogy; a scene is a distribution of light intensity in a plane  $(x, y)$ . The spatial signal, intended as the distribution of light in space represented by a signal of two variables, is transferred into the image system. The image system has a response function that characterizes it and transfers the input signal  $o(x, y)$  to the output, in the form of  $i(x, y)$ . A perfect system would produce  $i(x, y) = kor(x, y)$ , with  $k$  scaling factor. Real systems, however, are far from perfect. If we could do an analogy with a signal in time, we could affirm that lenses might have a similar function of the response in frequency characteristic of the so called pass-base systems of low frequencies that can be found in time keeping devices. As the name might suggest, a pass-bass filter allows only the passage of low frequencies. In the so called ‘space dominion’ we talk about space frequency which is given by the inverse of space. A lens is therefore a pass-bass filter for space frequencies. This means that every lens gives a specific response in space frequency: frequency is low when the luminous signal changes slowly through space and it ‘s instead high when,- going through space -, the intensity of light changes rapidly from one point to the next.

The MTF therefore stands as a function of transferability from the object to the image and describes the impossibility of an optical system to respond to high space frequencies.

The attenuation of a certain frequency is therefore represented by the MTF function. The resolution power can be defined by  $[\frac{lp}{mm}]$ ; when we are in the presence of two lines that get higher in intensity or frequency, there is a treshold after which the modulation becomes equivalent to zero: this limit is called the resolution power. In this case the MTF worsens by two or three magnitudes moving from the center to the side of the edge of the lens. It is therefore necessary that the resolution lens has to be in line with the magnitude of the space resolution of the sensor. There would be point no point in using an expensive, high resolution lens if the resolution was higher than that of the sensor.[13]

### 2.1.3. Lighting

The illumination of an image depends on the lens, sensor and illumination. Illumination represents the light that is sent into an object. When light is sent to a surface, it interacts with the object and part of this light returns back to the camera. The light-surface interaction depends on the colour and optical characteristics of the surface: some materials are translucent, others opaque and others complete transparent.

The optical characteristics of a surface materials affect how an incident surface becomes light as it goes towards the camera. The system influences the incident light. What arrives at the camera can be reflected or scattered light. If you have translucent or transparent surfaces, part of the light will never be returned to the camera, but it enters the material, thus diminishing the amount of light that returns to the sensor.

Also a phenomenon called fluorescence takes place when you send light with a certain wavelength towards an object. The light is then returned with a different wavelength.

The problems caused by incorrect lighting can be many, so it is necessary to use the most appropriate lighting according to the task that needs to be carried out. When choosing a particular type of lighting, it is necessary to establish first whether you are dealing with colour or black and white images. In the presence of a black and white image, for example, one needs to deal predominately with the intensity of the radiation. Instead, when dealing with colour images, the operator needs to deal the spectrum of the illuminating source in relation to the colours that need to be seen.

Other fundamental factors have to be taken into consideration when we want to illuminate a particular surface, for example choosing the right exposure according to the characteristics and textures of what needs to be illuminated, its size and shape, whether the object(s) is in motion, and whether specific details are required. Furthermore, the more multifaceted a three dimensional object is, the larger of the amount of shadows produced by the illumination process, therefore even great accuracy is required when setting the parameters. There are a number of different types of lighting such as laser lighting. Laser light technology emits light with particular characteristics, i.e. monochromatic, coherent (the waves are all in phase), collimated (the light emits a beam in a single direction) and sometimes polarized (although not always). The word "laser" stands for Light Amplification Through Stimulated Emission of Radiation. We speak of the emission of an electromagnetic field in the band of light waves; the electromagnetic field propagates in the  $z$  direction and is characterized by a

## Chapter 2. State of the Art technology

certain wavelength and is generally described by its electric field.

The emission is coherent, that is, it maintains the same wavelength in space and time.

If we consider an atom, electrons are generally found in the lowest energy state; if electrification occurs, these electrons move to the excited state. The excitation of an atom occurs through the absorption of energy which in the form of the interaction with a photon or alternatively by creating a strong electric field. This phenomenon is unstable; the electrons do not remain in the excited state, but return to the lowest energy state and begins to emit light. There is a spontaneous emission which is neither monochromatic, nor coherent or collimated.

The emission of a laser is a stimulated emission, because the electron is in an excited state. Then an incident photon is sent to destabilize the electron, causing it to fall to the basic energy state. A new photon is then emitted which copies the incident photon in wavelength, phase and direction. The incident photon is transformed into two identical photons; this is called the light amplification process.

In order for the laser to be functioning, an active material (a material that emits light) in a cylindrical shape is used, which is positioned orthogonally to two mirrors that are used to ensure that the photons emitted by the material along its axis are brought back inside. Paraxial photons travel  $n$  times in the material. Only the not perfect paraxial photons manage to escape from the cavity. A photon that encounters an atom duplicates itself, thus amplifying the source. There is therefore an intense radiation along the axis; as a result, a laser beam is formed which is projected because one of the two mirrors is semi-reflective.

In the active material there must be more excited atoms than the base atoms, that is, the so-called population inversion must occur. This occurs when energy is supplied to the material via electrical or optical pumping.

A laser can be characterized by a continuous or pulsed emission. In the case of continuous emission, the light intensity and therefore the power is constant over time. With a pulsed emission laser instead, the intensity goes from zero intensity to a maximum level over a certain time interval; in this case the power is not constant and takes the name of energy per pulse.

A typical laser beam follows a Gaussian intensity distribution, i.e. maximum intensity is found in the core, which diminishes as we move in the more peripheral areas; it also sports a high energy density, i.e. a high power per unit area (it can exceed  $3 \frac{kW}{m^2}$ ) which can present a serious challenge when used because of its intrinsic hazardous characteristics. [13]

#### 2.1.4. Software tools for vision systems

Regarding the analysis of the images, there are various tools for modifying the grey level obtained by overlaying the intensity of the three fundamental colours, i.e. the so called RGB standard, red, green and blue. Palettes, for example, are used to restore the colours that have to be shown on a screen or to change the colours of an image on screen. Palettes are born to represent a monochromatic section of the image itself. If the screen is in colour, it works with the three basic colours red, green, blue, that are overlaid to obtain the desired colour. There are three diagrams with the grey intensity display on the abscissa and the intensities of red, green and blue on the ordinate. *Look up tables* can be created in which the intensity of the three colours are related to the intensity of grey. These *look up tables* are used to map the different levels of grey into a triad of colours. If for example we want to obtain a black and white image on a colour monitor, we will need to have the same amount of red, green and blue on each pixel; a darker grey will be obtained if the intensities are low, while a lighter grey will be displayed if the intensities are greater. At a grey level of 255 we have the primary colours red, green and blue at their maximum intensity resulting in a white image. If for example the intensity of grey is instead lower, a darker image will be obtained. Based on the different distribution of the intensity of the red, green and blue colours, a large number of colours can be generated.

The image histogram is instead a function  $H$  that associates to each grey level  $k$  the number of pixels  $n_k$  of the image having that grey level:  $H(K) = n_k$

The grey intensity level for an 8-bit image ranges from 0 to 255. For a colour image, you have one histogram for red, one for green and one for blue. If the histogram is well centered and wide enough, the scene is well exposed and there is no saturation. An underexposed scene instead will need a greater aperture and different settings for lighting, exposure time and camera gain.

The *LUT (Look Up Table)* consists of algorithms that act on the gray level and allow the user to change the contrast and the brightness present in a scene. The LUT shows a more intelligible image on the monitor without modifying the information content but only the intensity of grey. A new image is created in which the intensity of the grey level is changed through the function  $I_{output} = f(I_{input})$ . The resulting image has a different contrast and a different brightness. If  $I_{output} = I_{input}$  the result doesn't change. In other cases, corrective actions can be undertaken to adjust various parameters according to the LUTs showed in the table 2.1.[13]





LUT	Transfer Function	Shading correction
<b>Equalize</b>	<i>Equalize histogram</i> 	Increase the intensity dynamic by evenly distributing a give grey-level interval $[min, max]$ over the full grey scale $[0, 255]$ . Min and max default values are 0 and 255 for an 8-bit image.
<b>Reverse</b>	<i>Reverse histogram</i> 	Reverses the pixel values, producing a photometric negative of the image.
<b>Logaritmik Square Root</b> Power $\frac{1}{Y}$	<i>Logaritmik histogram</i> 	Increases the brightness and contrast in dark regions. Decrease the contrast in bright regions.
<b>Exponential Square</b> Power $Y$	<i>Eponential histogram</i> 	Decreases the brightness and increases the contrast in bright regions. Decreases the contrast in the dark regions

Table 2.1.: Types of LUT. Source:[13]

An image is a pixel matrix; operators with images do operations on pixels. In general, the following operation applies:  $p_n = p_a(O_p)(p_b)$ , where  $O_p$  indicates the operator. As regard to images the following operations can be carried out:

- Multiplication:  $p_n = \min(p_a \cdot p_b, 255)$ ;
- Division:  $p_n = \max(\frac{p_a}{p_b}, 0)$ ;
- Addition:  $p_n = \min(p_a + p_b, 255)$ ;
- Subtraction:  $p_n = \max(p_a - p_b, 0)$ .

Spatial filters are meant to be working in space. They serve the purpose of letting through or stopping high or low spatial frequencies that represent the inverse of the space period.

A high pass filter does not pass spatial low frequencies. It does not affect the regions of the scene where the intensity is uniform, but enhances the parts with sudden changes in brightness. It is useful for highlighting the edges of an object. A low pass filter instead, favours regions with constant or little variable brightness in space.

Spatial filters are used to reduce noise, identify edges and enhance or eliminate details. The spatial filter algorithm is based on a matrix called the core or kernel of the algorithm, which is typically a 3 x 3 matrix, but can in some cases also be larger. The filter equation is:  $P_{i,j} = f(P_{i,j}; P_{i-1,j}; P_{i,j-1}; P_{i,j+1}; \dots)$ , where  $P_{i,j}$  is the native pixel, while other  $P_s$  are the pixels adjacent to the original pixel. A new image is generated pixel by pixel; at the edge of the image the adjacent pixels will be smaller than those positioned in a central pixel. Smaller kernels have a lower spatial resolution. The spatial filters then produce a new image; they operate on the central pixel, producing a new distribution of light intensity on it; this happens through a convolution algorithm given by:

$$P_{i,j} = \sum_{a=i-1}^{a=i+1} \sum_{b=j-1}^{b=j+1} k_{a,b} P_{a,b} \frac{1}{N} \quad (2.10)$$

The new image must be such as to ensure that  $0 \leq P_{i,j} \leq 255$

Spatial filters are divided into linear and non-linear filters and can be low pass or high pass filters.

Between the linear high-pass filters are placed the *Gradient Filter* and the *Laplacian Filter*. The *Gradient filter* is represented by the matrix below:

$$\begin{bmatrix} a & -d & -c \\ b & x & -b \\ c & d & -a \end{bmatrix}$$

The term  $x$  can be 0, if we consider only the edges so there is no trace of the original image in the final solution or 1 if we consider the edges and the central element. In this case the central image will still be present in the final result. The terms  $a, b, c$  and  $d$  are instead numbers that can all be identical. This filter highlights the variations in light intensity in certain directions which are the ones that make the value of the matrix elements change from negative to positive and viceversa.

The *Laplacian filter* instead works in all directions and is given by:

$$\begin{bmatrix} a & d & c \\ b & x & b \\ c & d & a \end{bmatrix}$$

where  $x$  is positive while  $a, b, c$  and  $d$  are negative.  $x$  can be  $= |2x(a + b + c + d)|$  if in the final result we want to obtain only the edges or  $> |2x(a + b + c + d)|$  if instead we want to obtain also the original image in the final result. This filter enhances the noise located in the regions where the brightness varies in space. As for the linear low pass filters, we can distinguish between the *Smoothing filter* and the *Gaussian filter*. The Smoothing filter acts as a linear operator. It

attenuates variations of bright light in the vicinity of a pixel. The matrix to be considered is the same as that seen for the Laplacian filter. If  $x$  is equal to zero, the original image in the final solution is not retained; if  $x$  is equal to 1, the original image is maintained. The larger the kernel, the bigger will be the space affected and the higher the number of pixels; consequentially a more accurate picture and a better resolution will be obtained.

As far as the Gaussian filter is concerned, the term  $x$  of the matrix is  $> 1$  and so the maximum value is therefore found in the center, as dictated in the Gaussian distribution function. This filter produces a similar effect to Smoothing filter, but since  $x > 1$ , the low pass effect is less pronounced.

Among the non-linear filters there is the *Prewitt filter*, which is a high pass filter that highlights the edges of an object and is given by the following matrix:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Morphology is the science of forms, aimed at making dimensional measurements in a two-dimensional domain. It concerns measurements involving the length or shape (2D) measurements of objects in the scene. There are several operations that need to be carried out during a morphological analysis of images. *Binarization* is a special Look up Table that divides an image into two regions, an object region and a background region. A black image corresponds to a bit equal to 0; a bright image corresponds to a bit equal to 1. A value is imposed below which a specific pixel is turned off, and above which the pixel is turned on. This allows objects that are not of interest to become optically invisible disappear. There are automated thresholding techniques that allow each time to choose the threshold level that gives the best results.

When an image is binarized, the problem of *Clustering* arises. Clasterization is used when classes greater than one need to be obtained. Sometimes different thresholds are used in various regions in an iterative way until the right representation is found. To prepare an object for measurement, some operations are necessary, such as erosion and dilation.

*Erosion* eliminates isolated pixels and erodes the spikes on the edges of the image. This process is therefore used to reduce or eliminate noise. Erosion eliminates bright spots and decreases the size of particles. The process of *Dilatation* can also be used to modify the shape of a given object within the image by adding virtual material; thus holes can be filled and shapes can be altered making it an effective tool ideal to be used in combination with erosion. Therefore, since dilatation and erosion change the shape of objects, they are frequently used in sequence. From the sequence of these two operations we obtain:



## 2.2. Machine Learning and Deep Learning for Computer Vision

- $Opening(I) = Dilatation[Erosion(I)]$ : to removes micro objects and smooth edges ;
- $Closing(I) = Erosion[Dilatation(I)]$ : to fill in holes before the erosion process is implemented.

[13]

## 2.2. Machine Learning and Deep Learning for Computer Vision

### 2.2.1. Introduction to Artificial Intelligence (AI) and Machine Learning (ML)

There is no formal definition of Intelligence. According to a cognitive approach it can be seen as the ability to solve a problem. The psychologist Howard Gardner has identified nine macro groups of intelligence by defining the following multi-intelligences:

1. Naturalist (nature smart);
2. Musical (sound smart);
3. Logical-mathematical (number/reasoning smart);
4. Existential (life smart);
5. Interpersonal (people smart);
6. Bodily-kinesthetic (body smart);
7. Linguistic (word smart);
8. Intra-personal (self smart);
9. Spatial (picture smart).

The concept of intelligence has changed a great deal in the last couple of centuries; for example reading skills, a privilege which only the very reach and powerful could access to until recently has now become common in large parts of the planet. It is therefore always necessary to contextualize intelligence. This also applies to *Artificial Intelligence*. The *Artificial Intelligence (AI)* is a branch of engineering that is mainly based on the process of being able to apply human learning methodologies to machines.

When it comes to AI, there are basically two schools of thought:

- *Strong AI* or *AGI, Artificial General Intelligence*: machines in the near future will be able to perform any task currently performed by human at the same or even higher level with self-awareness;

## Chapter 2. State of the Art technology

- *Weak AI*: this theory does not foresee a development self-awareness by machines, which will remain just devices only capable of performing certain given tasks.

To this date, the only form of AI that has produced appreciable results when implemented in practical applications is Weak AI.

According to the *Moravec Paradox*, from a computational point of view, it is much easier for a machine to solve problems that seem to be complex for the human mind, while it becomes extremely complex for a machine to deal with issues that the average human being finds easy to deal with.

Artificial Intelligence is typically divided into research algorithms (problem solving), issues connected to the representation of knowledge and Machine Learning (ML).

The idea behind ML is to feed data into the machine in order for the machine itself to build a model without any external interference or rules imposed from the outside. Whith a traditional AI, we define an algorithm and a set of rules that establish how a machine will behave according to a number of different settings and situations. With ML instead, we provide the machine with a number of different scenarios and settings; subsequently an algorithm will be used to choose the most appropriate setting thus creating a set of rules to be followed.

ML is therefore an AI field that deals with the development of machines capable of solving problems by choosing data previously imputed to the system in the most appropriate way; these sets of methods therefore create a system that is able to successfully assimilate data in its possession, a process which has a strong affinity with human learning.

There are two learning methods, incremental (or online) and static (or batch). With incremental learning, the so called trial by error, the process is refined gradually by learning from the mistakes previously made. To update the system the only thing that needs to be done is to feed it with new data. To upgrade a static training system instead the entire system must be retrained.

The types of learning in ML can be divided in *Supervised Learning*, in which a mathematical model is implemented connecting the input data with the output data, the so called *Unsupervised Learning*, in which we obtain by input data and *Reinforcement Learning*, a process which is similar to the learning activities experienced by human beings. [14]

We are living in an age of great abundance of data; using machine learning algorithms, it is possible to transform this data into knowledge.

Machine learning has evolved as a subfield of artificial intelligence and encompasses knowledge derived from data in order to being able to make reliable predictions. [3]

A Machine Learning model can be a mathematical expression or a complex data structure deriving from the theory of computer science or, in some cases,

## 2.2. Machine Learning and Deep Learning for Computer Vision

a combination of both. It is therefore an intersection between statistics, core computer science and software engineering. A ML model can learn from the actions of humans or of natural events and can simulate future behaviour regarding unknown situations. In simple terms, a model can predict future events based upon historical data. These actions are stored as records in the database. So, having a consistent and comprehensive dataset is essential for building a fully working useful model.

A ML model can learn from a certain dataset in order to give later predictions. It analyzes the value of predictor variables and builds a mathematical form or a data structure that can predict the values of the target variable. Predictor variables are often called features.

Variables are continuous and categorical. Continuous variables can have any value within a specific range. Categorical variables are generally classified as string data type, and these can only have a fixed number of different values.

Like any software development project, the ML model development projects also are typified by a number of different lifecycles. There are four major stages in a typical ML model development:

1. *Data exploration, analysis and research*: various kinds of slicing, dicing and visual operations are carried out at this stage by using a proper sample of the total dataset. A trial and error procedure concerning several models is also implemented at this stage. At the end, one particular model is chosen to proceed;
2. *Model training/learning*: this stage involves tuning parameters and hyperparameters of the chosen model in order to make the system learn from the provided dataset. Parameters essentially carry information requested by the model while hyperparameters are employed for fine-tuning (a better explanation of parameters and hyperparameters is given below);
3. *Model testing*: this stage deals with testing the developed model with an unknown dataset;
4. *Model deployment and scaling*: this stage involves converting the model into *Big Data technology*. At this stage a development of proper data engineering platform is carried out together with the optimization of scaling parameters and infrastructure. Scaling is required for handling huge volumes of data. The process cannot be carried out without the use of specific hardware and a dedicated supporting infrastructure.

These four stages are repeated cyclically after obtaining the user's feedback about the deployed model. [2]

Machine learning projects can be very different. As a ML projects involve a lot

of research-related activities, a team of people with the right skill is normally employed on the early stages. These projects go through several iterations, and it may happen that after a lot of time-consuming research, no output is actually obtained on the final stage. A flexible and open approach is therefore required as a precondition when embarking in similar projects.

Each machine learning model is backed up by an algorithm. From the concept of computer science and mathematical formulas, it can be said that an algorithm may need some externally supplied information to proceed. This information is called parameters. Hyperparameter instead are optional for a model. In the model definition, there is no existence of hyperparameters, which are instead normally required to tune the neural network and make it strong enough to produce an optimal result.

Machine learning practice can be divided into two types depending on the objective that need to be achieved:

1. *Predictive machine learning*: this type of machine learning practice is often used when predictions are needed but data is unknown;
2. *Descriptive machine learning*: ML models that are built to explain some kind of hidden behavior or a particular pattern within the data.

Before designing a model, it is crucial to know what kind of issues the designer is trying to address. It is therefore vital to obtain a detailed analysis of the dataset required to carry out the task. Generally the various models of ML are divided in the types:

- *Classification model*: this process is implemented when data has to be subdivided in different categories. Classification is a supervised learning process and comes under predictive ML. There are various techniques of classification models such as *Decision Tree*, the *Logistic Regression* and the *Naïve Bayes* just to name but a few;
- *Regression model*: if the target variable is continuous variable, a regression model is obtained. It also comes under the predictive ML and the supervised learning category. There are a number of regression techniques such as *linear regression*, the *decision tree regression* and others;
- *Clustering model*: if we have to look for groups or chunks inside the dataset without knowing the target's variables, we have a clustering problem. It comes under unsupervised learning and can be of both predictive and descriptive ML types. In a typical clustering model, tightly related data is grouped by analyzing its features. These groups are called *clusters*.

After designing the model, it is necessary to test its level of performance in order to get reliable data. This parameter is called *accuracy*. *Accuracy* is a

## 2.2. Machine Learning and Deep Learning for Computer Vision

factor that can be successfully employed when addressing issues connected with training processes as well as the validation or tests of datasets. An input dataset of a ML project can be split into three parts in the following way:

- Training set (50%): set of data with which the model is built;
- Validation set (25%): it is used to test the performance of the model and to estimate prediction errors for the selected model;
- Test set (25%): it is used to test the model and to assess the generalization error of the final chosen model.[14]

Typically the data set is initially divided only into the training set (70%) and the validation set (30%) in order to find the most appropriate model. Once the model has been chosen, with the new photos employed for the test set, the ability of the architecture to generalize new data is evaluated.

The accuracy will be calculated differently depending on whether you are dealing with classification, regression or a clustering issues. For a classification problem, for example, accuracy is measured by the ratio between the number of data instances correctly classified and the total number of records. This accuracy can assume values between zero and one. A larger value of this ratio indicates a better classification.

An other important aspect to consider when dealing with a machine learning problem is the so called error of the model. The overall error in a model is made up of two factors, *bias* and *variance*. [2]

In the context of machine learning models, *variance* measures the consistency or variability of the model prediction by classifying a serious of given examples when retraining the model over a number of times with different subsets of the training dataset. The result of a too high variance would be, in a very complex model, a low accuracy in the test set due to the difficulty for the model to generalize new data. High variance is therefore a barrier in designing a perfect model. The *bias*, instead, measures the general difference between the predictions and the correct values if we retrain the model several times on different training datasets; bias is the measure of systematic error which is not related to randomness. During the classification process there might be issues connected to the model when it classifies a certain type of data labeled in a specific way corresponding to a set of values or characteristics. Sometimes the presence of highly regular classes creates biases in the model. A very highly complex model will have high variance and a low bias. A mathematical expression for the overall error is:  $Error = Bias + Variance$ . Decreasing bias will increase variance and vice versa. A good model should have the optimum combination between variance and bias values. [2] [3]

The combination between bias and variance during the ML development model often create two side-effects commonly known as *underfitting* and *overfitting*:

- *Underfitting*: it is caused by shortage of training data, in what is known as an immature model thus causing high bias and low variance. An underfitted model fails to predict properly future developments or work accurately in all situations. Sometimes, even though we have sufficient training data, the number of iterations done for the training is low, which is again causing high bias;
- *Overfitting*: as opposed to underfitting this side effects occurs when the system experiences a high variance and a low bias. This happens when the data imputed is over abundant to finely tuned in order to achieve a high degree of precision in the model. Therefore the over-tuning of parameters or too much iteration on the training data can be in some cases counter-productive. There might be instances for example when a model, previously fed with comprehensive highly accurate and finely tuned data, might fail to perform well due to its inability to recognize a certain type of new data because it might contain parameters not present in its system .

[2]

One way to achieve a good compromise between bias and variance is to adjust the complexity of the model through *regularization*. The *Regularization* is a very useful method to obtain collinearity (a high degree of correlation between characteristics), eliminating noise from the data and preventing overfitting. The concept on which regularization is based is to introduce additional information (*bias*) to penalize the extreme values of the parameters (*weights*). The most common form of *regularization* is the *L2 Regularization* which in mathematical terms is expressed as:

$$\frac{\lambda}{2} \|w^2\| = \frac{\lambda}{2} \sum_{j=1}^m w_j^2 \quad (2.11)$$

where  $\lambda$  is the regularization parameter, through which the adherence of the model to the training data can be checked. By increasing the  $\lambda$  value, the regularization intensity is increased. [3]

### 2.2.2. Introduction to Deep Learning for Computer Vision

*Computer Vision* is at the forefront of AI research and a virtual intersection between state-of-the-art computation, storage and deep learning research.[1]

## 2.2. Machine Learning and Deep Learning for Computer Vision

*Deep Learning* is a process of supervised learning in successive layers of increasingly meaningful representations. [2]

Some important applications in computer vision include:

- Self-driving transportation;
- Fraud detection;
- Security systems;
- Public administration;
- Content analysis, management and retrieval.

Computer vision requires various computationally efficient techniques to use data in a meaningful manner. In recent times, the considerable growth in CPU (Central Processing Unit) speed has not matched the speed of data creation. This in turn has led to the development of many parallel processing architectures and the rise of GPUs (Graphic Processing Units), which previously employed in computer game technology, and now are increasingly used for the computational purposes.

Deep learning is a powerful combination of methods and techniques which provide the blueprint for flexible models to be used. It combines multilayer perceptron algorithms with various mathematical concepts. In deep learning, the model, when properly tuned, automatically finds the optimum combination of input features, enhancing the accuracy of a decision-making process.

the process of Deep learning can be scantily described as mapping/connecting data from A to B. A and B broadly represent the input and the output respectively. A more accurate and comprehensive interpretation of Deep Learning might be the performance of universal function approximations; in other words, with sufficient labeled data, the system should be able to approximate any function. The complexity of a function can be increased by extending its layer and by adding more units on the various layers i.e. to add more variables and parameters to a function. Deep learning has better learning abilities than those associated to traditional machine learning algorithms because it can select efficiently the most appropriate features from datasets. Thanks to the increased efficiency and speed of data processing and the relatively easy use of its applications, deep learning is quickly becoming a personal favourite of many researchers and one of the most widely-used algorithm in the field of AI. [1]

### 2.2.3. Artificial Neural Networks

Our brain can recognize features and objects swiftly even when those are perceived in a form that might not be familiar with what had been previously

memorized by our neural cells. Whenever we touch an object, we can feel it. This is possible because of the sensors in our fingers tips which are connected directly to our brain. All human body sensors work simultaneously to collect data which is then processed and passed on to our brain. Based on this information, the brain makes the final decision and sends the relevant instructions or information to the appropriate body part.

Neural networks are inspired from the human brain and its inner workings. Even today with all the modern technology at our own disposal it is still a quiet daunting task to fully comprehend how intricate the human brain is and how it works. Nevertheless its main functions and features could be briefly resumed as follow: the human brain is composed of approximately 100 billion nerve cells called neurons. Each neuron can relate to thousands of other neurons that are found in a large network. A neuron communicates with other neurons in the form of electrochemical signals. Every feeling, thought and emotion we experience comes from millions of small neurons communicating to each other. Without these neurons who work and carry messages throughout our body constantly, we wouldn't be able to feel or do anything.

A typical neural network consists of many artificial neurons called units that are arranged in a series of layers. A typical artificial neural network consists of different layers, as shown in Figure 2.3:

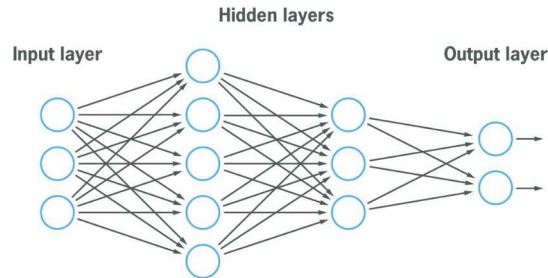


Figure 2.3.: Example of artificial neural network. Source:[18]

- *Input layer*: The input layer contains those units (artificial neurons) that receive input from the outside world on which the network will learn, recognize or otherwise process;
- *Output layer*: The output layer contains units that respond to the information about how it has learned some or all the tasks it was supposed to learn;
- *Hidden layers*: The hidden layers are units that are positioned between the input and output layers. The job of the hidden layers is to transform the input into a form that is comprehensible for the output layer.



## 2.2. Machine Learning and Deep Learning for Computer Vision

We can create different types of neural networks by connecting nodes in a different manner and changing the flow direction of data. Each type of neural network has a different level of complexity which is directly related to the tasks that need to be carried out for a specific application. [1]

### Feedforward Neural Network

The simplest of all neural networks, the feedforward neural network, moves information in one direction only. Data moves from the input nodes to the output nodes, passing through the hidden nodes. There is no feedback mechanism from the output to the input area; hence, the feedforward network can be simply expressed graphically with a directed acyclic graph.

It is possible to distinguish the single layer perceptron from the multilayer perceptron; the *single layer perceptron* is the most basic form of artificial neural network that consists of input and output layers with no hidden layers. It is a type of feedforward neural network with 0 hidden layers. A *multilayer perceptron (MLP)* instead, is a class of feedforward neural network. MLP consists of three layers; an input layer, an output layer and one (or more) hidden layer. Multilayer perceptrons or feedforward networks are the fundamental and classical form of Deep Learning models.

The hidden layers do not supply information in order to establish the true output value from the input data. Instead they are used by the learning algorithms in order to choose the optimum value of parameters to approximate function  $F$ , which maps input to output.

Each layer within network consists of units. Units of hidden layers are called hidden units. By adding more layers and hidden units, the dimensionality of the network increases and so does the complexity of the function. Mathematically speaking, each layer can be defined as a vector in which a number of units represent elements of the vector. A neural network therefore can be interpreted as connections between vector and scalar elements, where every single element of a given layer is connected, through a vector, to every other single element of the adjacent layers.

Focusing on deep learning, there are broadly two types of models: linear and non-linear. Linear models use simple mathematical properties, generally additivity and multiplicity, in order to obtain the desired solution. These models can fit on linearly separable data and they are simple and easy to use too.

In the real world, however, we almost exclusively encounter situations that present challenges that cannot be solved with linear functions but require instead complex and linear models.

The Deep learning technique introduces non-linearity into the network. In Deep Learning, we use an approximation function  $F$  with various parameters. During the mapping of input to output, this technique is useful to obtain

data concerning the parameters optimum value within the function itself. This process in turn optimizes the obtainment of a generic mapping function essential to achieve reliable results during data tests.[1]

### Cost/Loss Functions

Deep Learning algorithms use optimization techniques, i.e. minimization or maximization of a function to get the optimum solution. In the optimization process, a given function is minimalized to achieve a stable global minimum. Even during the process of maximization, should this be required, it is preferable to minimize the negative value of the specific function, a process virtually equivalent to the maximization of the original function. The function that needs to be maximized or minimized may be defined as *cost function*, *loss function* or *error function* (or more generally it might be called *objective function*).

The training of the neural network is quite similar to other machine learning algorithms, but the computation of the gradient is more difficult in nonlinear networks. In linear models, we use a convex function (for a function to be convex, the line connecting two points on the graph of the function must lie above the function) as an optimization function. The advantage of the convex function is that it converges to a very low value with any initial value of parameters, but due to nonlinearity in the neural network, the associated cost function becomes non-convex and hence, cost function does not converge to a very low value. In this case therefore iterative and gradient-based optimization techniques are employed to converge the cost function to a possible low value.

The *cost/loss functions* are used to estimate parameters of the universal approximation function that maps input  $X$  to output  $Y$  and eventually quantifies the performance of the approximated model. To get the best approximate model, the parameter of the model need to be estimated in order to achieve a minimum loss.

Maximum likelihood algorithm is the most widely practiced algorithm for the estimation of parameters. As regards the maximum likelihood principle, let's take  $m$  sample points from any probability distribution  $P(X)$ , where  $X = \{x_1, \dots, x_m\}$ . Next, let's define the model as  $\hat{P}(X; w, b)$ , such that it determines the probability that  $x_i$  (where  $i \in \{1, 2, \dots, m\}$ ) belongs to distribution  $P(X)$  having parameters  $w$  and  $b$ . Now it is possible to define the maximum likelihood estimator for  $\theta$  - that is used as a condensed form of parameters  $w$  and  $b$  - as:

$$\hat{\theta} = \theta_{MLE} = \operatorname{argmax} \hat{P}(X; w, b) = \operatorname{argmax} \prod_{i=1}^m \hat{P}(x^i; w, b) \quad (2.12)$$

With the help of the equation 2.12, the parameter values  $w$  and  $b$  are estimated

## 2.2. Machine Learning and Deep Learning for Computer Vision

so that probability of  $x^i$  belonging to the distribution  $P(X)$  is maximum. We can see that  $\hat{\theta}$  is a product of multiple probabilities, which leads to an error affecting numerical precision. To avoid any numerical precision error, we generally prefer to take the logarithm of  $\hat{\theta}$ , which changes the product of probability density to the sum of probability density. Logarithm is an increasing function that doesn't change the maximum point of the original function. After taking a log of  $\hat{\theta}$ , we can write  $\hat{\theta}$  as:

$$\hat{\theta} = \operatorname{argmax} \sum_i m \log(\hat{P}(x^i; w, b)) \quad (2.13)$$

The scaling of the function will not change the maximum point of the original function, so we can divide  $\hat{\theta}$  by  $m$  (total number of data point) and again define  $\hat{\theta}$  as expectation or average of  $\hat{\theta}$ .

$$\hat{\theta} = \operatorname{argmax} E\left(\sum_i m \log(\hat{P}(x^i; w, b))\right) \quad (2.14)$$

The *Maximum likelihood estimation* can also be interpreted as a technique that minimizes the dissimilarity between complex models and approximated models. *Kullback-Leiber divergence* or *KL divergence* is an algorithm that compares two probability distributions. *KL divergence* helps to measure the amount of loss of information when a complex distribution system is replaced with an approximated distribution one. Mathematically, we can write *KL divergence* as:

$$\begin{aligned} KL_D &= P(X) - \hat{P}(X) \\ &= \log(P(X)) - \log(\hat{P}(X)) \\ &= E(\log(P(X))) - E(\log(\hat{P}(X))) \end{aligned} \quad (2.15)$$

There is no condition on  $P(X)$  as it is an original distribution; so, to minimize dissimilarity, we will have to optimize the term containing  $\hat{P}$ . We can see that minimizing the above equation means maximizing  $E(\log(\hat{P}(X)))$ , which is the same as minimizing  $-E(\log(\hat{P}(X)))$ , which is the same as maximization of the equation 2.14.

The fundamental concept in supervised learning is the estimation of the conditional probability of model  $Y|X; w$ . We can define the relation between input  $X$  and output  $Y$  as follows:

$$Y = \hat{F}(X; w, b) + \epsilon = \theta^T X + \epsilon \quad (2.16)$$

For  $m$  data points, we can write:

$$y^i = \theta^T x^i + \epsilon = w^T x^i + b + \epsilon^i, i = 0, \dots, m \quad (2.17)$$

## Chapter 2. State of the Art technology

Where  $\widehat{F}$  is the approximated function,  $\theta(w, b)$  is the parameter of the function, and  $\epsilon$  is an error term or random noise in the model. Let's assume that this error term is independently and identically distributed and also normally distributed, i.e.  $\epsilon \sim N(0, \sigma^2)$ . From data points, we can write probability density of error term  $\epsilon^i$  as:

$$p(\epsilon^i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^i)^2}{2\sigma^2}\right) \quad (2.18)$$

From equation 2.18, we can write that probability distribution of output  $y^i$  will also be Gaussian, that is:

$$p(y^i|x^i; w, b) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^i - w^T x^i - b)^2}{2\sigma^2}\right) \quad (2.19)$$

In other words, we can say that  $y^i$ , will be a random variable that is distributed according to the Gaussian distribution, with mean  $w^T x^i + b$  and variance  $\sigma^2$ , that is:

$$y^i|x^i; w, b \sim N(w^T x^i + b, \sigma^2) \quad (2.20)$$

There are various *cost functions* such as the *least square function*, the *cross-entropy function* and the *softmax function*. [1]

The equation 2.19 is a function of  $y$  for fixed  $w$  and  $b$ , but it can also be interpreted as a function of  $w$  and  $b$ . In mathematical terms, it can be written as:

$$p(Y|X; w, b) = L(w, b) = L(w, b; X, Y) \quad (2.21)$$

$$L(w, b) = \prod_{i=1}^m p(y^i|x^i; w, b) \quad (2.22)$$

$$L(w, b) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^i - w^T x^i - b)^2}{2\sigma^2}\right) \quad (2.23)$$

We can define  $L(w, b)$  as the *likelihood function*. We can write  $l(w, b) = \log(L(w, b))$ . Now, we want to maximize the log likelihood function  $l(w, b)$  using the maximum likelihood principle.

## 2.2. Machine Learning and Deep Learning for Computer Vision

$$\begin{aligned}
 l(w, b) &= \log(L(w, b)) \\
 &= \log \left( \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp \left( -\frac{(y^i - w^T x^i + b)^2}{2\sigma^2} \right) \right) \\
 &= \sum_{i=1}^m \log \left( \frac{1}{\sqrt{2\pi}\sigma} \exp \left( -\frac{(y^i - w^T x^i + b)^2}{2\sigma^2} \right) \right) \quad (2.24) \\
 &= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^m (y^i - w^T x^i + b)^2
 \end{aligned}$$

In the preceding equation, we can see that the first term is constant, so maximizing  $l(w, b)$  is the same as minimizing:

$$\sum_{i=1}^m (y^i - w^T x^i + b)^2 \quad (2.25)$$

We define this minimizing function as the *least square loss function*, and in mathematica terms we write this as function of  $w$  and  $b$ ,

$$J(w, b) = J(w, b) = \frac{1}{2} \sum_{i=1}^m (y^i - w^T x^i + b)^2 \quad (2.26)$$

And we can define the mean square error as:

$$\frac{1}{m} J(w, b) = \frac{1}{m} \left( \frac{1}{2} \sum_{i=1}^m (y^i - w^T x^i + b)^2 \right) \quad (2.27)$$

From equation 2.27, it can be said that by applying probabilistic assumptions on the distribution of data, we can use the *least square cost function* to estimate the optimum value of parameters of the approximated model.

We can see that cost function  $J(w, b)$  will always be non-negative due to square factor. When the output from approximated model will be very close to the output of the true model, then the value of  $J(w, b)$  will be very low, i.e.  $J(w, b) \approx 0$ ; but, if  $J(w, b)$  is large, then parameters of the approximated model are not useful. So, our objective is to find the value of parameters  $w$  and  $b$  such that it makes the value of *cost function* as low as possible.[1]

In binary classification,  $y$  takes two values, either 0 or 1. Intuitively, it can be seen that the model should not give values larger than 1 and smaller than 0, because the output value  $y$  is always set between 0 and 1. In order to achieve this constraint, a function needs to be chosen which determines the output of the model between 0 and 1. The *sigmoid function* therefore is a function that keeps the output value between 0 and 1. The *Sigmoid function* is also known as

## Chapter 2. State of the Art technology

the *logistic function*. In mathematical terms, the *sigmoid function* is defined as:

$$g(z) = \frac{1}{1 + e^{-z}} \quad (2.28)$$

And we can define output  $Y$  of a model as:

$$Y = \frac{1}{1 + e^{-(w^T X + b)}} \quad (2.29)$$

Where,  $w$  and  $b$  are the parameters of the model. Any smooth function can be used as long as it keeps the output value between 0 and 1.

To better understand the *loss function* should have in order to estimate that can estimate the optimum values of parameters for a binary classification, the *quadratic loss function* should be employed to estimate the optimum value of the parameters. Let's define an actual and predicted output of model as  $y$  and  $\hat{y}$  respectively.

Then, the loss function should be  $J(w, b) = \frac{1}{2}(y - \hat{y})$ . It is preferable to minimize the convex function to obtain a stable global minima, but in the case of binary classification, the *optimization function* turns out to be non-convex with multiple local minima. The non-convexity arises because the presence of the nonlinear *sigmoid function*.

In this case certain probabilistic assumptions were used in order to obtain the *quadratic cost function*. Let's define a few probabilistic assumptions to derive the *cross-entropy cost function*.

Assumptions:

$$P(y = 1|x; \theta) = P(y = 1|x; w, b) = \hat{F}(x; w, b) \quad (2.30)$$

$$P(y = 0|x; \theta) = P(y = 0|x; w, b) = 1 - \hat{F}(x; w, b) \quad (2.31)$$

The first equation explains that the approximate function of the model ( $\hat{F}(x)$ ) estimates the probability of  $y = 1$ . Now,  $y$  can be either a 0 or 1. So, probability of  $y = 0$  will be  $1 - \hat{F}$ . We can merge the above two equations and write  $P(y|x; w, b)$  as:

$$P(y|x; w, b) = (\hat{F}(x; w, b))^y \cdot (1 - \hat{F}(x; w, b))^{(1-y)} \quad (2.32)$$

Consequently when  $y = 1$ , the above equation gets transformed into the first equation of the assumption, and when  $y = 0$ , the equation turns out to be the second equation of the assumption.

As with the *least square function*, the *maximum likelihood principle* will be employed to estimate an optimum value of parameters for a binary classification problem.

## 2.2. Machine Learning and Deep Learning for Computer Vision

Let's now define the *likelihood function*  $L(w, b)$ . Therefore:

$$p(Y = 1|X; \theta) = L(w, b) = L(w, b; X, Y) \quad (2.33)$$

For  $m$  independent training examples, we can then write  $L(w, b)$  as:

$$L(w, b) = \prod_{i=1}^m p(y^i|x^i; w, b) \quad (2.34)$$

$$L(w, b) = \prod_{i=1}^m (\hat{F}(x^i; w, b))^{y^i} \cdot (1 - \hat{F}(x^i; w, b))^{1-y^i} \quad (2.35)$$

Let's take log of  $L(w, b)$  and define this as the *log likelihood function*  $l(w, b)$ :

$$\begin{aligned} l(w, b) &= \log \left( \prod_{i=1}^m (\hat{F}(x^i; w, b))^{y^i} \cdot (1 - \hat{F}(x^i; w, b))^{1-y^i} \right) \\ &= \sum_{i=1}^m m y^i \log(\hat{F}(x^i; w, b)) + (1 - y^i) \log(1 - \hat{F}(x^i; w, b)) \end{aligned} \quad (2.36)$$

And the total cost of the model may be defined as:

$$l(w, b) = \frac{1}{m} \sum_{i=1}^m y^i \log \hat{F} + (1 - y^i) \log(1 - \hat{F}(x^i; w, b)) \quad (2.37)$$

So, to find the best value of parameters that fit the model, we will have to maximize the *log likelihood function*  $l(w, b)$  or minimize the negative of  $l(w, b)$ . So, the *cost function* may be written as:

$$\left( -\frac{1}{m} \sum_{i=1}^m y^i \log(\hat{F}(x^i; w, b)) + (1 - y^i) \log(1 - \hat{F}(x^i; w, b)) \right) \quad (2.38)$$

For simplicity, we will define the approximate output  $\hat{F}(x^i; w, b) = a^i$ . So, eq. 2.38 can be written as:

$$J(w, b) = -\left( \frac{1}{m} \sum_{i=1}^m y^i \log(a^i) + (1 - y^i) \log(1 - a^i) \right) \quad (2.39)$$

We will maintain the same simplistic notation for the approximate output of network.[1]

It is possible to extend the *cross-entropy* applicability for variables with  $n$  probable values.

For the purposes of binary classification it can be assumed that:

$$P(x^i; \theta) = P(x; w, b) = \widehat{F}(x; w, b) = a \quad (2.40)$$

For  $n$  possible values, we can rewrite the generic form of the above equation as:

$$P(x^i; \theta) = P(x^i; w, b) = \widehat{F}(x^i; w, b) = a^i \quad (2.41)$$

In this case, we will get a vector of output  $a$ , of which, the value of each element of the vector will set between 0 and 1 and the total sums of all the elements will be equal to 1.

In mathematical terms, the *softmax function* is defined as:

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_n \exp(x_i)} \quad (2.42)$$

Knowing the *cross-entropy function* for binary variable from the equation 2.39, we can extend the equation for  $n$  values and write it as follows:

$$J(w, b) = -\frac{1}{m} \left( \sum_{i=1}^m \sum_{j=1}^n y^j \log(a^j) \right) \quad (2.43)$$

where  $m$  is the total number of samples and  $n$  is the total number of classes. We can see that, for  $n = 2$ , we will get the same *cost function* that we derived for the binary variable, where  $y^1 = y_1$ ,  $y^2 = 1 - y_1$ . Similarly, if  $a^1 = a$ , then  $a^2 = 1 - a$ .

During the prediction process, the output class has been transformed in an *one-hot encoder vector*. *One-hot encoding* is a mechanism where one categorical variable is decomposed into a set of continuous variables [2]. As regards to the input value, the *cost function* therefore can be written as follows:

$$J(w, b) = -\sum_{j=1}^n y^j \log(a^j) = -(0 + \dots + y^k \log(a^k) + 0) = -\log(a^k) \quad (2.44)$$

[1]

The *softmax function* is also employed as an *activation function* for the output layer, when *multi-class classification* is needed. The *sigmoid function* (further discussed below), is instead used as activation function for the output layer in a *binary classification*.

## Optimization

The optimization techniques are used to train the neural network. Essentially, this means to estimate the corresponding model parameters at which the *cost*



## 2.2. Machine Learning and Deep Learning for Computer Vision

*function* takes its minimum value.

The *Gradient Descent* and the *Stochastic Gradient Descent* are considered to be two important optimization techniques.[1]

The *Gradient Descent* is employed to determine the direction of steepest descent used to converge the *cost function*. To get the converging point, the slope of the function  $f(x)$  needs to be calculated and consequently infinitesimal displacements  $\delta x$  will have to be carried out in order to reach the minimum point.

The function  $y = f(x)$  can be calculated as follows:

$$\delta y = f(x + \delta x) - f(x) \Rightarrow \frac{\delta y}{\delta x} = \frac{f(x + \delta x) - f(x)}{\delta x} \quad (2.45)$$

For very infinitesimal displacement, the following equation can be employed:

$$\frac{\delta y}{\delta x} = \frac{dy}{dx} = f'(x) \quad (2.46)$$

$$f'(x) \approx \frac{f(x + \delta x) - f(x)}{\delta x} \Rightarrow f(x + \delta x) \approx f(x) + \delta x \cdot f'(x) \quad (2.47)$$

More comprehensively, it can be written  $f(x + \delta x) = f(x - \delta x \cdot \text{sign}(f'(x)))$ . It is also known that in an increasing function,  $f'(x) > 0$ , whereas for a decreasing function  $f'(x) < 0$ . We can therefore infer from this equation that, in the increasing function  $f(x - \delta x)$  is less than  $f(x)$ . Hence, to decrease the value of  $f(x)$ , the value of infinitesimal displacement  $\delta x$  will have to be imputed towards the opposite direction of  $f'(x)$ . Similarly, the same procedure can be applied for the decreasing function.

Based on what has been discussed above, it can be deduced that the new point will be  $x' = x - \alpha f'(x)$ , where  $\alpha$  is defined as *learning rate*, which determines the size of the infinitesimal displacement.

Generally, it is preferable to set a very low value for  $\alpha$  in consecutive steps in order to obtain the conversion of an optimal rate to the desired minimum point without any unwanted 'jumps'. In multiple dimensions, when there are multiple independent variables,  $f'(x)$  it will be necessary to compute in both direction. For multivariate inputs, it will be necessary to calculate the gradient of function  $f(X)$ , which is denoted as  $\nabla_x f(x)$ . This gradient computation technique is defined as *gradient descent*, also known as *batch gradient descent*; the *cost function* for the *batch gradient descent* is therefore defined as:

$$J(w, b) = \frac{1}{m} \left( \frac{1}{2} \sum_{i=1}^m m \frac{1}{2} (\hat{y}^i - y^i)^2 \right) \quad (2.48)$$

[1]

The *Batch gradient descent* utilizes all the samples placed in the training set and sums up the gradients to update parameters at every step. When the optimization of a model for a large dataset is required, there might be issues connected with the slow speed of the algorithm and its tendency of becoming computationally expansive. To overcome these problems, researchers have devised a new algorithm called *Stochastic Gradient Descent*, also known as SGD.

In SGD the gradient for the first training example is calculated first. Then, its value gets imputed into the sample in order to update the parameter. The updated parameter becomes than the initial parameter for the second training example, a process which is then repeated again through the various steps. In conclusion, in SGD, when working with training samples, the gradient has to be calculated in the first place. The subsequent errors are then employed to update the parameters of the model itself. In contrast to the *batch gradient descent*, in SGD a complete set of training dataset is needed to implement the infinitesimal displacements for the steepest descent. However, with the employment of SGD there might be a correlated unwanted factor called the zig-zag motion which affects the primary objective i.e. the obtainment of convergence to the global minimum. A local minimum is instead obtained which in most cases can be considered as an acceptable result because close enough to the global minimum

The *cost function* for the *batch gradient descent algorithm* is given by the equation 2.47. In SGD instead, as the gradient is calculated for each training sample, the *cost function* gets changed to the equation:  $J(w^i, b^i) = \frac{1}{2}(\hat{y}^i - y^i)^2$  that can be also written as  $J(w, b) = \frac{1}{m} \left( \frac{1}{2} \sum_{i=1}^m J(w^i, b^i) \right)$ . [1]

### Activation Function

A linear model with a single neuron is fed with a linear combination of *weights* and inputs.

In mathematical terms, a prediction for the output  $y$  is expressed as:

$$y = \text{linear} \left( \sum w.x + b \right) \quad (2.49)$$

The formula produces a linear decision boundary and consequently, a *linear classifier*. We use a neural network to build non-linear models, in which each neuron gets an input similar to that generalized in linear models.

A generic equation can therefore be written in which the neuron gets the input values as follows:

$$y = \text{Activation function} \left( \sum w.x + b \right) \quad (2.50)$$

## 2.2. Machine Learning and Deep Learning for Computer Vision

In a linear model, the *linear function* is used as an *activation function*. It can be said that if a *linear activation function* is used for a neural network, the network will always compute linear functions regardless of the number of layers it might or might not have in its system. In other words the network will behave like there are no hidden units. Similarly, it can be observed that if the *activation function* is modified in the output layer from *linear* to *sigmoid*, the same neural network will behave like a *logistic regression model* because the link from the input to the output unit is a linear combination of *weights* and original inputs. A *linear activation function* can therefore be used in output units when issues such as *regression* and *time series prediction* occur. A different *activation function* - non-linear - must be used when dealing with *hidden layers*. [1]

### The Sigmoid Function

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.51)$$

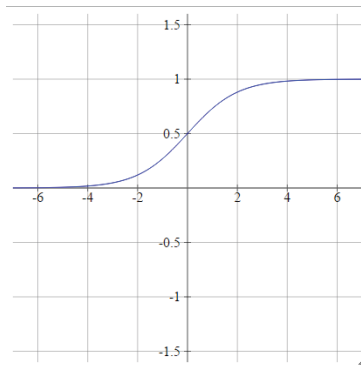


Figure 2.4.: Sigmoid Function. Source:[19]

Figure 2.4 shows a graph of the *sigmoid function*. Traditionally, the *sigmoid function* has been used as an *activation function* in neural networks. As previously discussed, an its characteristic is to limit the range of its values between 0 and 1. It is widely used in *binary classification*. Although the *sigmoid function* encompasses the non-linearity, it has a few drawbacks:

- *Vanishing gradient*: When the output value of *hidden units* correspond to 0 or 1, gradient become zero due to saturation (slope = 0). We know that the gradient should not carry a zero value while updating the parameters, otherwise there will be no errors propagating backwards, (the *backward propagation effect* will be discussed later) and the parameters won't get updated. Due to this potential problem, extra caution is required while initializing the parameters.

- *Non-zero centered output:* An output value in any given hidden unit generates non-zero centered data, thus creating a zig-zag dynamic descent during gradient descent updates, which will inevitably lead to a slower convergence.

[1]

### The Tanh Function

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \quad (2.52)$$

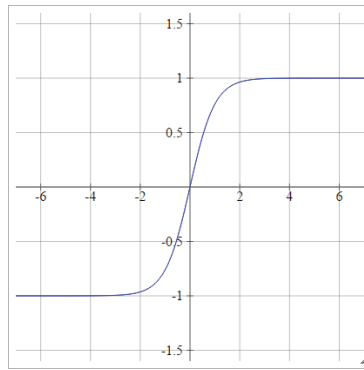


Figure 2.5.: Tanh Function. Source:[19]

The *Tanh Function* is another *non-linear function* used as an *activation function*. Its output value is always limited to a range between  $-1$  and  $1$ . The *Tanh function* can be interpreted as a 'elongated' version of the *sigmoid function*. In mathematical terms, it can be expressed as  $\tanh(x) = 2\sigma(2x) - 1$ . The *Tanh function* also carries some side effects such as the *vanishing gradient problem*, but is generally preferred over the *sigmoid function* due to its *zero-centered output*. [1]

### The Relu Function

$$g(x) = \max(0, x) \quad (2.53)$$

Rectified linear units commonly known as *relu function*, use the equation 2.53. The *Relu function* is easy to optimize due to its similarity with the *linear function*. The difference between a *linear function* and a *Relu function* is that has a zero output value when  $x$  is less than or equal to zero. Figure 2.6 shows the graph of the *relu function*. The derivative of the *Relu function* is equal to 1 in the middle of its domain. The hidden units will remain active when  $x$  is greater than zero. It helps to avoid the problem of the vanishing gradient when

## 2.2. Machine Learning and Deep Learning for Computer Vision

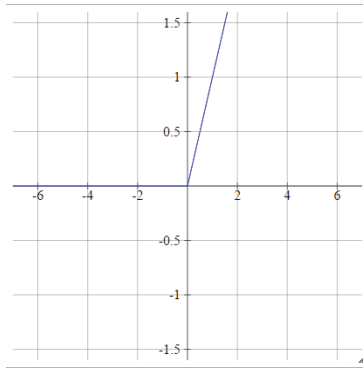


Figure 2.6.: ReLU Function. Source:[19]

$x$  is greater than zero. In contrast to the *Sigmoid* and *Tanh functions*, where the gradient value decreases as the value of  $x$  gets larger, the gradient of the *Relu function* remains constant thus helping to achieve a faster convergence.[1]

**The Leaky relu Function** Unlike the *Sigmoid* and *Tanh functions*, *Relu* provides sparse network due to its characteristics (outputs 0 when  $x \leq 0$ ). It has certain advantages over the *Sigmoid* and *Tanh*, the *Relu function* has its disadvantages too. It can be observed that range of the *Relu function* goes from 0 to infinity. This means a possibility of a very high output of the activation function which might lead to exploding gradients and a slower convergence. When  $x \leq 0$ , gradients become zero, so weights cannot be updated during *back propagation*. This problem is commonly known as the *dying relu problem*. To mitigate this problem, a new activation function, called *Leaky relu*, has been devised.

$$g(x) = \max(\alpha x, x) \quad (2.54)$$

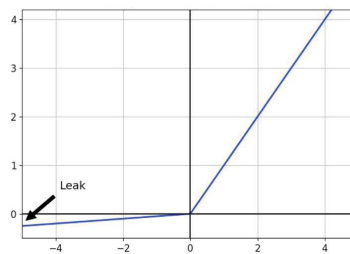


Figure 2.7.: Leaky relu Function. Source:[20]

It is based on the  $g(x) = \max(\alpha x, x)$  function, where  $\alpha$  is a very small number.

In *Relu function*, when  $x \leq 0$ , it returns to 0, but in *Leaky relu*, it will return to the  $(\alpha x)$  value. The value of  $\alpha$  is generally kept close to 0.01 or even smaller.[1]

### Backpropagation

In *feedforward neural networks*, we take an input  $x$  and then information from  $x$  flows through the hidden layers to produce output  $\hat{y}$ . As information from  $x$  can only flow forward, this process takes the name of *forward propagation*. Once the  $\hat{y}$  output is obtained, the  $J$  cost function can then be defined. This represents the error between a true  $y$  output and the predicted  $\hat{y}$  output. Using the *backpropagation algorithm*, information (essentially the gradients) is fed from the cost function back to the network. This essentially can be seen as a process that uses the feedback mechanism to enhance the model performance. In *backpropagation*, information is fed again to the network, but this time backwards that's why this technique is known as *backpropagation*. *Backpropagation* can be carried out also in a generalized linear model and can therefore be employed to compute the gradient of even function.[1]

### Overfitting and Underfitting

In any model learning process, it is essential for the model to generalize from the training data so it can make predictions for the future based on the data previously unknown. In practical term however, two scenarios normally occur:

1. *Overfitting*: the model has managed to assimilate training data too well capturing even the tiniest of details such as noise. This can produce undesired side-effects and seriously undermine the performance of the model which is not capable of understanding new information which differs even slightly from previously stored data;
2. *Underfitting*: the model is not able to learn significant details necessary to classify data supplied from the input area, i.e. it can't adapt data from the training set or can't generalize new incoming data.

There are several issues that might be related to the *overfitting* side-effect. The abundance of data although a positive factor when establishing parameters and characteristic as close as possible to reality should always be supported by the model's capability of being able to effectively generalize the scenario. Deep neural networks, might have millions of different parameters (weights and biases) which may cause the model to overfit. To effectively train the model, it is therefore crucial to detect those cases in which the model overfits which isn't an easy task. A lower *accuracy* on the unforeseen data (test data) compared to that of the *training set* might be a sign of *overfitting*. We can use the following methods to prevent *overfitting* in deep neural networks:

## 2.2. Machine Learning and Deep Learning for Computer Vision

- A larger amount of data: more data means an increased capability for the model to capture all the significant details and estimate the parameters appropriately;
- *Train/Dev/Test set*: it is always recommendable to have a separate *dev/validation set* in order to be able to finely tune *hyperparameters*, keeping the test set completely unforeseen. This is because in neural networks there are a lot of *hyperparameters* to play with. Using the test set *accuracy* to set the *hyperparameters* leads to leaking of the test (unforeseen) data information, a factor often responsible for the failure of the model in production system;
- Reduce the size of the network: this is a solution that in most case can not be implemented as larger more powerful networks have substantial advantages over small ones;
- *Regularization*: It is the most commonly used method for preventing *overfitting*. It basically adds an extra component (*penalty term*) to the *cost function* in order to modify the network. In this way the model is punished for being too complex. The simplification of the model leads to a better generalization performance. The *L1* and *L2* are two *regularization* techniques which add an extra penalty term in the *cost function* so that network is able to learn small *weights*;
- *Drop out*: In this technique, neurons (or nodes) in each layer (except those belonging to the output layer) of the network are dropped out (deactivated) with a *probability p*. *Probability p* is a *hyperparameter* which is set during the training process and which default value is normally set at 0.5. It is worth noting that all neurons are kept in an activated state (no drop out) while testing/inferencing;
- *Early stopping*: it has been often observed that by increasing the number of *epochs*, the *accuracy* during the *validation phase* stops increasing when it reaches a certain *epoch*, while the *loss* value stabilizes or decreases. Therefore by implementing the *early stopping*, the training process gets interrupted before the model reaches the *overfitting* stage.

Although *underfitting* is a less common problem than *overfitting*, it can equally affect the performance and efficiency of the model by over simplifying certain operations which can lead the system to overlook certain details supplies by the training data. Furthermore the implementation of a very high *regularization coefficient* ( $\lambda$ ) to avoid *overfitting* might at the end lead to the creation of the opposite problem i.e. *underfitting*. *Underfitting* is easier to detect with a good performance metric-like *accuracy*. The underfit model gives poor performance

during testing as well as during the training data process. To prevent *underfitting* in deep neural networks, there are two solutions:

- An increment in the number of layers or the number of neurons: with a larger number of layers or neurons, it is possible to create more free parameters to be estimated in order to increase the learning capacity of the model;
- Decreasing the *regularization coefficient*.

[1]

## 2.2.4. Convolutional Neural Networks (CNN)

### Introduction to CNN

*Convolutional neural networks* are a family of models which have been devised by using the same principle utilized by the human brain when it recognizes and then processes data (objects, people, colours, etc.) through the visual cortex. Effectiveness in extracting relevant characteristics is the key to a good performance of any machine learning algorithm.

Some types of neural networks, such as *convolutional neural networks*, are able to automatically learn the characteristics from the raw data that are most useful for a certain task. For this reason, the layers of *convolutional neural networks* are rightly considered feature extractors: the first layers (those placed just after the input layer) extract low-level features from the raw data while the subsequent layers (which are often fully connected layers, such as those in a multi-layer perceptron) use these characteristics to predict a continuous target value or label of a class.

Some types of multi-layer neural networks, and in particular *deep convolutional neural networks*, are organized in a so called *hierarchy of characteristics*, combining for example low-level characteristics (low or high). For example, if we are dealing with images, then low-level features, such as edges and protusions, are extracted from the first few layers and then combined together to form high-level features. These high-level features can reveal more complex shapes, such as the general outlines of objects. A *convolutional neural network* computes feature maps from an input image, where each element comes from a local group of pixels in the input image. This local group of pixels is called a *local receptive field*. *Convolutional neural networks* are normally very effective when employed for image-related tasks essentially because of two main factors:

- *Sparse connectivity*: a single feature map element is connected to only a small group of pixels. This behaviour is very different from connecting to the entire input images, as it happens for example with the *perceptron*;



## 2.2. Machine Learning and Deep Learning for Computer Vision

- *Sharing parameters*: the same *weights* are used for different pixel groups for the same input image.

Replacing a fully connected standard multi-level perceptron with a *convolutional layer* substantially reduces the number of *weights (parameters)* of the network and improves the ability of the system to capture relevant features. In the context of images, it makes sense to assume that neighboring pixels are typically more relevant to each other than pixels that are more distant.

*Convolutional neural networks* normally consist of several *convolutional* and *subsampling layers*, followed by one or more *fully connected layer*. *Fully connected layers* are essentially *multi-layer perceptrons*, where each unit of input,  $i$ , is connected to each output unit,  $j$ , with weight  $w_{ij}$ .

It is important to note that *subsampling layers*, commonly called *pooling layers*, have no parameters to learn; for example, there are no *weights* or units of *bias* in the *pooling layers*. In contrast, the *convolutional* and *fully connected layers* both have *weights* and *biases* to be optimized through the training phase. [3]

### Convolution operation

A *discrete convolution* (sometimes simply called *convolution*) is a fundamental operation in a *convolutional neural network*.

The relevant indexes indicate the dimensions of a multidimensional array (*tensor*); for example,  $A_{n_1 \times n_2}$  is a bidimensional array of  $n_1 \times n_2$  dimension. The square brackets are also considered as tools to denote the indices of a multidimensional array; for example,  $A[i, j]$  refers to the element at the index  $i, j$  of the array  $A$ . Finally, the special symbol  $*$  indicates the *convolution operation* between two vectors.

With these notations, it is possible to define a *convolution* for two vectors,  $\bar{x}$  and  $\bar{w}$ , as  $\bar{y} = \bar{x} * \bar{w}$ , where the vector  $\bar{x}$  is the input (also called signal) and  $\bar{w}$  is denominated *filter* or *kernel*. In mathematical terms, a *convolution* is defined as follows:

$$\bar{y} = \bar{x} * \bar{w} \rightarrow y[i] = \sum_{k=-\infty}^{+\infty} x[i-k]w[k] \quad (2.55)$$

The index  $i$  scrolls through each element of the output vector  $\bar{y}$ . There are two problematic aspects to the above formula, which which have to be considering: the fact that the summation crosses the indices from  $-\infty$  to  $+\infty$  is fairly peculiar, since in machine learning we always have to deal with vectors of finite characteristics. For example, if  $\bar{x}$  has 10 characteristics with indices 0, 1, 2, ..., 9, then the indices from  $-\infty$  to  $-1$  and from 10 to  $+\infty$  are outside the limits of  $\bar{x}$ . Therefore, to correctly calculate the sum represented in the formula 2.54, it must be assumed that  $\bar{x}$  and  $\bar{w}$  are padded with zeros. This

will produce an infinite sized output vector  $\bar{y}$  with many zeros. Since this is not useful in practical situation,  $\bar{x}$  is padded with only a finite number of zeros. This process is called *zero padding* or, simply, *padding*. Here, the number of zeros added on each side is denoted by  $p$ .

Assuming that the original input,  $\bar{x}$ , and filter,  $\bar{w}$ , have  $n$  and  $m$  elements respectively, where  $m \leq n$  and the padded vector,  $\bar{x}^p$  has dimensions  $n + 2p$ , the practical formula for calculating a *discrete convolution* can be expressed as follows:

$$\bar{y} = \bar{x} * \bar{w} \rightarrow y[i] = \sum_{k=0}^{k=m-1} x^p[i + m - k]w[k] \quad (2.56)$$

Once the infinite index problem has been solved, there is a the second second issues which needs to be addressed i.e. the indexing of  $\bar{x}$  with  $i + m - k$ . It is important to note that  $\bar{x}$  and  $\bar{w}$  are indexed in different directions in the summation. Calculating the summation with an index running in the reverse direction is equivalent to calculating the summation with both indices proceeding in the same direction after the inversions one of these vectors,  $\bar{x}$  or  $\bar{w}$  has taken place alongside the zero padding. At this point the *dot product* can be calculated. The filter  $\bar{w}$  is inverted (rotated) ( $\bar{w}^r$ ), the scalar product  $\bar{x}[i : i + m].\bar{w}^r$  is then calculated to obtain an element,  $y[i]$ , which  $\bar{x}[i : i + m]$  is a part of  $\bar{x}$  with  $m$  dimensions. This operation is repeated with a *sliding method*, in order to obtain all the output elements. The values obtained by the use of the *sliding* method form a so called *stride*, another *hyperparameter* of a *convolution* which must be a positive number and smaller than the size of the input vector.[3]

The *cross-correlation* between an input vector and a filter is denoted by  $\bar{y} = \bar{x} * \bar{w}$  and it is very similar to a *convolution*, with one small difference: in a *crosscorrelation*, the multiplication is performed in the same direction. Therefore, it is not mandatory to rotate the filter matrix,  $\bar{w}$ . In mathematical terms, the *crosscorrelation* is defined as:

$$\bar{y} = \bar{x} * \bar{w} \rightarrow y[i] = \sum_{k=-\infty}^{+\infty} x[i + k]w[k] \quad (2.57)$$

As for *padding* and *sliding* the same rules apply to *correlation*.[3]

Technically, *padding* can also be applied with  $p \geq 0$ . Depending on the choice of  $p$ , the peripheral cells can be treated differently from those located "inside" $\bar{x}$ . There are three commonly used padding modes:

- *full padding*: the *padding* parameter,  $p$ , is set to  $p = m - 1$ . Full *padding* increases the size of the output; therefore, it is rarely used in convolutional

## 2.2. Machine Learning and Deep Learning for Computer Vision

neural network architectures;

- *same padding*: this *padding* is normally used to ensure that the output vector is identical to the input vector,  $\bar{x}$  in term of size. In this case, the *padding* parameter,  $p$ , is calculated according to the size of the filter, provided that the input and output sizes are the same;
- *valid padding*: it refers to the case when *no padding* ( $p = 0$ ) has been performed.

The most commonly used *padding* mode in *convolutional neural networks* is the *same padding*. One of its advantages over other *padding* modes is that it preserves the vector dimensions - both the height and the width of the input images when image processing and computer vision tasks are carried out- a factor which greatly simplifies very much the design of a network architecture. In *valid padding*, as opposed to *full padding* and *same padding* the volume of tensors is substantially reduced in multi-layer neural networks, a big disadvantage that can considerably affect the performance of the model.

In practice, when working with *convolutional network* it is advisable to preserve the spatial dimensions by using the *same padding* for *convolutional layers*. If a reduction of spatial dimension needs to be carried out, it is preferable to employ *pooling layers*. When *full padding* is employed, the output produced is larger than the size of the input. *Full padding* is therefore normally used in signal processing applications, where it is important to minimize the effects of the extremities. However, in the context of deep learning, the effects of extremities are usually not a problem, so a practical use of *full padding* is rarely seen.[3] The output size of a *convolution* is determined by the total number of times the filter  $\bar{w}$  is scrolled along the input vector. Assuming that the input vector is of size  $n$  and the filter is of size  $m$ , the size of the output resulting from  $\bar{y} = \bar{x} * \bar{w}$ , with padding  $p$  and stride  $s$ , would be determined as follows:

$$o = \left\lfloor \frac{n + 2p - m}{s} \right\rfloor + 1 \quad (2.58)$$

where  $\lfloor \cdot \rfloor$  denotes the *floor operation*, which returns the largest integer that is less than or equal to the input.[3]

When we are dealing with a 2D input, such as a matrix  $\bar{X}_{n_1 \times n_2}$  and a filter matrix  $\bar{W}_{m_1 \times m_2}$ , where  $m_1 \leq n_1$  and  $m_2 \leq n_2$ , then the matrix  $\bar{Y} = \bar{X} * \bar{W}$  is the result of a 2D convolution between  $X$  and  $W$ . In mathematical terms it can be defined as:

$$\bar{Y} = \bar{X} * \bar{W} \rightarrow Y[i, j] = \sum_{k_1=-\infty}^{+\infty} \sum_{k_2=-\infty}^{+\infty} X[i - k_1, j - k_2] W[k_1, k_2] \quad (2.59)$$

If one of the above dimensions is omitted, the remaining formula will be the same as in the *1D convolution*. All the techniques used for *1D convolution*, such as *zero padding*, rotation of the filter matrix and the use of strides, are also applicable in *2D convolution*, as long as they extend independently in both dimensions. [3]

### Pooling

Subsampling is typically applied in two forms of *pooling* operations in *convolutional neural networks*: *max pooling* and *mean pooling* (also called *average pooling*). The *pooling layer* is normally denoted by  $P_{n_1 \times n_2}$ . Here, the index determines the size of the neighbouring pixels (the number of adjacent pixels in each dimension), on which the max or average operation is calculated. These neighboring elements are called the *pool size*. The benefit of *pooling* is a multiple. *Max-pooling* introduces a local invariance. This means that small variations in the surrounding area do not change the max-pooling result. Therefore, it helps to generate characteristics that are more resistant to the noise present in the input data. Additionally, *pooling* reduces the size of features, thereby improving computational efficiency. Finally, a reduction in the number of features helps to reduce the propensity for overfitting. While *pooling* is still an essential element of many *convolutional neural network* architectures, a number of devices have been developed without a *pooling layer*. In order to reduce the size of features, researchers employ *convolutional layers* with a *stride* of 2 to avoid using *pooling layers*. In a sense, a *convolutional layer* with a *stride* of 2 can be considered as a *pooling layer* with *weights* to be learned.[3]

### The Dropout

Choosing the most appropriate size of a network, either for a *traditional (fully connected) neural network* or for a *convolutional neural network*, is always a challenging issue. For example, the size of a *weight matrix* and the number of *layers* must be optimized in order to achieve a reasonable performance. A simple network, with no *hidden layers*, can only capture a linear decision boundary. The efficiency of a network is measured by the capacity of the model to learn and approximate the *cost function*. Small networks, or networks with a relatively small number of parameters, have a relatively low capacity and therefore are more prone to *underfitting* and to a less efficient performance, as they cannot learn the underlying structure of complex datasets. However, large networks, as previously mention, can be prone to *overfitting*: the network stores training data with such precision, that might find difficult to understand new details coming from brand new data. When dealing with real machine learning issues, it is not know a priori what the size of the network should be. One way to solve this

## 2.2. Machine Learning and Deep Learning for Computer Vision

problem is to build a network with a relatively large capacity (it is preferable to choose a slightly higher capacity than what a forecast suggests) and big enough to operate well on the training dataset. To avoid *overfitting*, one or more *regularization* schemes should be applied to achieve a good generalization performance on new data, such as the test dataset. *L1* and *L2* *regularizations* can also be used for neural networks, with *L2* at present being the most common choice. However, there are other methods to achieve the *regularization*, such as *drop out*. In recent years, *drop out* has emerged as a widely used technique for the *regularization* of deep neural networks with the aim of avoiding *overfitting*, i.e. to improve the generalization capacity of the network. The *drop out* is normally applied to the hidden units of the upper layers and works in the following way: during the training phase of a neural network, at each iteration a fraction of the hidden units are randomly discarded with a *probability*  $p_{drop}$  or kept with a *probability*  $p_{keep} = 1 - p_{drop}$ . The *drop out* value is set by the operator with a probability of  $p = 0.5$  being the most common choice. Because of a certain number of input neurons are being discarded during the *drop out* process, the *weights* associated with the remaining neurons acquire a new value and a new size (*resizing*) to account for the missing neurons. Thanks to the effect resulting from the random *drop out*, the network is forced to learn a redundant representation of the data. As the network cannot count anymore on the activation of its *hidden units* (they could be deactivated at any time during training) it is obliged to learn more general and solid patterns from the data. In this way *overfitting* is prevented.

### Loss Functions for a CNN

Some *activation functions*, such as *Relu*, are mainly used for the *hidden layers* of a neural network to introduce a share of non-linearity into the model. But others, such as *sigmoid* (employed mainly when confronted with *binary issues*) or *softmax* (used mainly during *multiclass classification*), are added to the *output layer* which is responsible for establishing what kind of probability a certain data output has to belong a certain class (destined to become the output of the model). If *sigmoid* or *softmax* activations are not included in the *output layer*, the model would compute the *logits* - logarithm of probabilities - instead of calculating the probabilities of certain data input belonging to a certain class. For issues connected to *classification* (whether *binary* or *multiclass*), and according to the type of output (*logit* or *probability*) when training a model, an appropriate *cost function* needs to be employed. The *binary cross-entropy* is mainly used for problems connected to the *binary classification*, while the *categorical cross-entropy* is employed for issues connected with *multi-class classification*.



## Chapter 3.

# Materials and methods

### 3.1. Introduction to the project

With *ConvNets* becoming more of a commodity in the computer vision field, a number of attempts have been made.

As reported in the first chapter, the following work takes part for an European project, which is based on the development of a device that aims to allow better detection of defects in an automotive production chain. In particular, this tool, the *G3F*, consists of a smartphone cover with a triangulation laser sensor inside. Pointing the laser at a car in a production chain, the sensor shows in output *Gap&Flush* measurement of the area under consideration. These quantities allow the operator to determine if the various parts of the car are well assembled or not, thus avoiding going to sell defective components. To obtain the best possible acquisition, it is necessary to adjust the power emitted by the laser according the components and colors of the car that are taken into consideration from time to time.

For this purpose, with the following work, a software architecture based on convolutional neural networks has been studied to guarantee this classification. Two main classification problems were treated: a binary classification that allows you to distinguish a separation between two sheet-metal or that between sheet-metal and headlight, and a multiclass classification, with which the neural network was trained to recognize ten different classes. These classes were made to be able to identify various car colors. In detail, five different colors have been selected and for each color two classes have been created: one class relating to the separation between two sheets-metal and the other class relating to the division between light and sheet.

The efficiency of the various models was assessed by considering *accuracy* and *loss* during the training and validation phase. For a classification problem, the *accuracy* is defined by the ratio between the number of records predicted correctly and the total number of records. It is a probability metric, and value ranges from 0 and 1. A higher value infers a better *accuracy* and classification problem. It also can be expressed in term of percentages like probability. [2]

The term *loss*, on the other hand, refers to the loss measured for a single data point. Unlike *accuracy*, the *loss* is not a percentage value and can be even greater than one. The greater the *loss*, the less efficient is the classifier. [3]

## 3.2. Binary classification

### 3.2.1. Multiple architectures with an online cats and dogs dataset

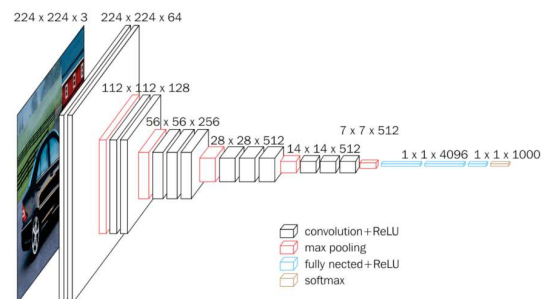
We began by building a program making use of the iteration process that would allow the user to obtain all the possible combination of hyperparameters. To begin, a program, that through iteration would allow you to try out all the various combinations of hyperparameters was built. In particular, five models were chosen: VGG16, Xception, InceptionResNetV2, InceptionV3 and ResNet50. Each model was combined with two *batch size* values, 32 and 64, two *learning rate* values, 0.001 and 0.0001 and five *optimizers*: the SGD, the Adam, the Adadelta, the Adagrad and the RMSprop.

#### VGG Net

*Convolutional networks (ConvNets)* have recently enjoyed a great success in large-scale image and video recognition which has become possible thanks to the spreading of large public image repositories, such as *ImageNet*, and also thanks to a number of high-performance computing systems, such as *GPUs* or large-scale distributed clusters. In particular, an important role in the advance of deep visual recognition architectures has been played by the *ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)*, which has served as a testbed for several large-scale image classification systems, such as high-dimensional shallow feature encodings and the deep *ConvNets*. [15]

One of the most popular deep *ConvNets* is the VGG-16 architecture, shown below:

Figure 3.1.: VGG-16 architecture. Source:[21]





### 3.2. Binary classification

The *VGG Net*, firstly proposed by the *Visual Geometry Group*- University of Oxford, is known for its simplicity as it utilizes only  $3 \times 3$  *convolutional layers* stacked on top of one. The *VGG Net* has very attractive features because of its uniform architecture and relatively small filter with a size of  $3 \times 3$  that help to reduce the total number of parameters. The blocks featuring the same filter size are utilized a number of times. However, since the *VGG Net* uses a great number of filters in different layers, the number of parameters can be quite large resulting in heavy and sometimes slow computation. The following image is about the various depths:

Figure 3.2.: VGG Nets of Various Depth. Source:[21]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

[1]

### Inception network

The winning architecture of the *ImageNet challenge 2014* was the *GoogLeNet/Inception Network*, which has proved to be an important milestone in the field of *CNN architecture*.

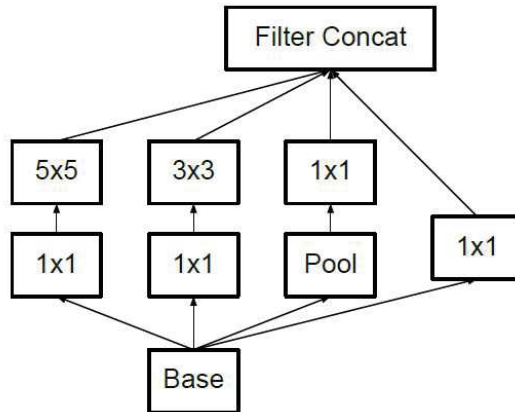
Deeper networks have their own disadvantages; one of them being the issue connected to the *vanishing gradients* which make difficult for those layers who first receive data, to process, learn and memorize information. When in the presence of a large number of layers, it can be observed that the number of *weights* is also high. As a consequence, training becomes computationally very expensive, both in terms of memory and time. Deep networks are also prone to *overfitting* and as a result they do not generalize well. [1]

Although *VGGNet* has compelling features due to its architectural simplicity, there are two sides of the same coin: the proper evaluation of a network requires a lot of computation and this might require more time to carry out the requested tasks. The computational cost of *Inception* model instead is much lower than *VGGNet*. This feature has made the *Inception* networks an attractive choice in big-data scenarios, where a huge amount of data needs to be processed at a reasonable cost. *Inception* is also employed in scenarios where memory or computational capacity is inherently limited, for example in mobile vision settings. It is certainly possible to find the most appropriate solutions in order to address specific issues such as memory shortages, or to customize up to a certain degree the performance of a given system by optimizing of a certain operations via computational tricks. However, it must be noted that this methods often add extra complexity to the system. [16]

The network architecture of *GoogLeNet* builds upon the classical architectures we have discussed previously and exploits some new techniques that will be examined shortly devised to address issues such as the *vanishing gradients*, the *overfitting* and *underfitting*.

- *1x1 convolution* and *Inception module*: The idea behind the *inception*

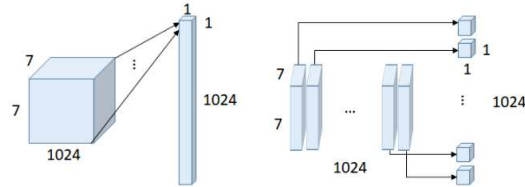
Figure 3.3.: Inception module. Source:[22]



*module* is to use *convolutions* of different sizes to capture information at varied scales. In Fig.3.3, we see that *convolutions* of different sizes - *1x1*, *3x3* and *5x5* have been used to extract all different levels of features from the previous layer. *1x1 convolutions* serve the purpose of reducing the computations by limiting the number of input channels;

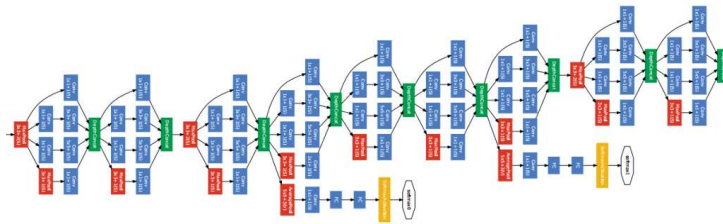
- *Global average pooling layer*: As can be observed from Fig.3.4, in a *fully connected network*, all inputs are connected to every output. In a *global*

Figure 3.4.: Fully connected layer vs global average pooling. Source:[23]



*average pooling* case, each  $7 \times 7$  slice gets mapped to  $1 \times 1$  filter by *pooling* on average all the  $7 \times 7$  values, thereby creating a set of 1024 values with no parameters left required to be estimated, unlike what happens in a the *fully connected layer*. This simple trick of *average pooling* requires less computation helps to improve the *accuracy* of the model.

Figure 3.5.: GoogLeNet architecture. Source:[24]



[1]

## Res Net

Ideally, in a model, *accuracy* should improve when increasing the number of layers. In practice, this doesn't happen very often, because as seen before, the *vanishing gradient issues* arises. Due to the increasing depth, *input layers* do not receive the required information necessary to change *weights* from the *output layers*. As a result, the training error starts increasing after a particular depth is reached, causing what is called in technical terms a *degradation problem*.

Introduced in 2015 by *Microsoft Research* as part of *ImageNet Challenge 2015*, the revolutionary *residual Networks* (or *ResNet*) have greatly contributed to solving the *degradation problem* and have led to a better *accuracy* in the *training phase* of *very deep neural networks*. [1]

### Xception

*Xception* is a *convolutional neural network* architecture based entirely on depth-wise separable *convolution layers*. This architecture has 36 *convolutional layers* forming the feature extraction base of the network. The 36 *convolutional layers* are structured into 14 modules, all of which have linear residual connections. This makes this type of architecture very easy to implement and modify, as only 30 to 40 code lines are used with high level libraries such as *Keras* or *TensorFlow*, (the simplicity of this type of architecture resembles for example the *VGG-16*). In contrast, a model of architecture such as the *Inception V3* is instead for complex to develop and implement. [17]

### Program structure

To train the classifier, a dataset of *cats and dog* has been extrapolated from the web. This dataset consists of 21000 images utilized for the *train set* and 4004 images destined to the *validation set*.

To iterate the various models described above with the different values of the *hyperparameters*, a *list*, featured below, was created:

```
numClass = 2

shape=(150, 150, 3)

input_tensor = Input(shape)

base_models = [VGG16(weights=None, include_top=False,
                    input_tensor = input_tensor,
                    input_shape = shape),
               Xception(weights=None, include_top=False,
                        input_tensor = input_tensor,
                        input_shape = shape),
               InceptionV3(weights=None, include_top=False,
                           input_tensor = input_tensor,
                           input_shape = shape),
               ResNet50(weights=None, include_top=False,
                        input_tensor = input_tensor,
                        input_shape = shape),
               ResNet50V2(weights=None, include_top=False,
                           input_tensor = input_tensor,
                           input_shape = shape)]

batch_size = [32,64]
```

### 3.2. Binary classification

```
lr = [0.001,0.0001]
opt = [0,1,2,3,4]

p = [base_models, batch_size, lr, opt]
import itertools
t = list(itertools.product(*p))

parameters = t[0]

for parameters in t:

    if parameters[3] == 0:
        optimizer = SGD(parameters[2])
    if parameters[3] == 1:
        optimizer = Adam(parameters[2])
    if parameters[3] == 2:
        optimizer = Adadelta(parameters[2])
    if parameters[3] == 3:
        optimizer = Adagrad(parameters[2])
    if parameters[3] == 4:
        optimizer = RMSprop(parameters[2])
```

Various layers were then added to the basic models; in particular, a *pooling layer* has been added in order to reduce the size of the image, maintaining its main characteristics. Later two *dense layers* (a fully connected layers) were added (one with 1024 neurons and the other with 2 neurons). In the 1024-neurons *dense layer*, the *relu* was used as an *activation function*. In the 2-neurons last layer a *sigmoid* was instead used, because the problem in question regards a *binary classification*. A *binary cross entropy* was set as a *cost function* and *accuracy* was chosen as a method for evaluating the performance of the model.

```
x = parameters[0].output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)

# and a logistic layer — let's say we have 200 classes
```

Chapter 3. Materials and methods

```
predictions = Dense(numClass, activation='sigmoid')(x)

model = Model(inputs=parameters[0].input,
              outputs=predictions)

model.compile(loss='binary_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])

batch_size = parameters[1]
```

The *data augmentation* was then carried out to expand the starting dataset.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
```

Once the model was trained with the *training* phase, the *validation* phase was then implemented.

```
# this is a generator that will read pictures found in
# subfolders of 'data/train', and indefinitely generate
# batches of augmented image data
train_generator = train_datagen.flow_from_directory(
    'data/train', # this is the target directory
    target_size=(150, 150), # all images will be resized
    batch_size=batch_size,
    class_mode='binary') # since the
                          # binary_crossentropy
                          # was used,
                          # binary labels will need
                          # to be employed

# this is a similar generator, for validation data
validation_generator = test_datagen.flow_from_directory(
    'data/validation',
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary')
```

50 *epochs* have been set with a number of steps per epoch equal to  $\frac{2000}{batchsize}$ .

### 3.2. Binary classification

The results were saved on a csv file in which the *accuracy* and the *loss* for the various combination of *hyperparameters* were reported.

```
history = model.fit_generator(  
    train_generator ,  
    steps_per_epoch=2000 // batch_size ,  
    epochs=50,  
    validation_data=validation_generator ,  
    validation_steps=800 // batch_size)  
  
acc = history.history['accuracy'][-1]  
val_acc = history.history['val_accuracy'][-1]  
loss = history.history['loss'][-1]  
val_loss = history.history['val_loss'][-1]  
  
cols = ["model", "batch_size", "lr", "opt", "acc",  
        "val_acc", "loss", "val_loss"]  
pp = [parameters[0], parameters[1], parameters[2],  
      parameters[3], round(acc,2), round(val_acc,2),  
      round(loss,2), round(val_loss,2)]  
  
df = pd.DataFrame([pp], columns = cols)  
df.to_csv('architettura_varie.csv', mode='a', header=False)
```

Having obtained the best results with the *VGG16 architecture*, we went to work on the project with this model.

#### 3.2.2. Binary classification with VGG16 architectures

After evaluating the performance of various models of neural networks on a dataset of images of *cats and dogs*, we moved on to study the *classification* related to the real issues connected with the identification and classification of various parts and colours of vehicles. With a dataset consisting of 31174 photos utilized for the *training* phase and 6002 photos employed during the *validation* phase, the classifier was then trained by using the *VGG16 architecture* as the start up model, to recognize two classes: one which concerned the separation between two parts made of metal sheets, and the other which deal with the separation between sheet-metal and a headlight. Two images samples belonging to the two classes used for the dataset are shown in figure 3.6.

To obtain an efficient and comprehensive classification of dataset with the widest combinations of light settings and weather conditions, photos were taken at different times of day and in different days.

In *computer vision* it is essential to limit as much as possible the creation of

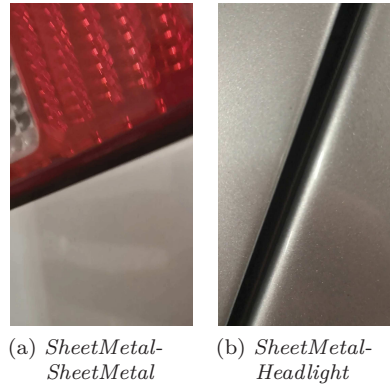


Figure 3.6.: Dataset images used for binary classification

*bias* by the model itself.

For example, during the training of a network with a dataset of images that are too similar to one another, the architecture could memorize photos it receives as input in the *training phase*, without being able to generalize efficiently new photos that might show slightly different characteristics. A problem, as discussed before, called *overfitting*.

It is therefore fundamental to create a large dataset with images as different as possible.

As for the program code, a *list* was firstly created which allowed to iterate the various *hyperparameters*; in particular, we set *learning rates* of 0.0001 and 0.000001, with *batch size* values of 32 and 64 and the same five types of *optimizers* for the previous code were also used.

```
classes = os.listdir('data_new/train')
num_classes = len(classes)

vgg_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))

batch_size = [32, 64]
lr = [0.0001, 0.00001]
opt = [0, 1, 2, 3, 4]

p = [batch_size, lr, opt]
import itertools
```



### 3.2. Binary classification

```
t = list(itertools.product(*p))

parameters = t[0]

for parameters in t:

    if parameters[2]==0:
        optimizer=Adam(parameters[1])
    if parameters[2]==1:
        optimizer=Adagrad(parameters[1])
    if parameters[2]==2:
        optimizer=SGD(parameters[1])
    if parameters[2]==3:
        optimizer=RMSprop(parameters[1])
    if parameters[2]==4:
        optimizer=Adadelta(parameters[1])
```

The *layers* shown below were then added to the basic *VGG16 model*; three *dropout* with three *dense layers* were added on top of the *flatten layer* which was employed to allow the mapping between the *input layer* and the first *hidden layer*:

```
def build_model():
    model = tf.keras.models.Sequential([
        # our vgg16_base model added as a layer
        vgg_base,
        # here is our custom
        #prediction layer (same as before)
        Flatten(),
        Dropout(0.50),
        Dense(1024, activation='relu'),
        Dropout(0.20),
        Dense(512, activation='relu'),
        Dropout(0.10),
        Dense(1, activation='sigmoid')
    ])
    vgg_base.trainable = True

    model.compile(optimizer=optimizer,
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model
```

### Chapter 3. Materials and methods

```
batch_size=parameters [0]
```

```
model = build_model()
```

Since this architecture deals with a *binary classification* issue, as reported in the previous section, it has been necessary to set for the last layer a *sigmoid* as the *activation function* while *binary\_crossentropy* has been employed as the *loss function*.

*Data generators* were then added, which were used for carrying out various operations of *data augmentation*, necessary to expand the data set.

```
# build our image data generators —
```

```
train_datagen = ImageDataGenerator(  
    rescale=1.0/255,  
    rotation_range=40,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    shear_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True,  
    fill_mode='nearest')
```

```
# NOTE: no image aug for eval & test datagenators
```

```
val_datagen = ImageDataGenerator(rescale=1.0/255)  
test_datagen = ImageDataGenerator(rescale=1.0/255)
```

The *training* and *validation* images were then recalled from the directory, resizing each image to the dimensions of (150, 150, 3).

```
# Step-2: create generators, which 'flow' from directories
```

```
# create the generators pointing to folders created above
```

```
# ALL IMAGES TO BE RESIZED to (150,150,3)
```

```
train_generator = train_datagen.flow_from_directory(  
    'data_new/train',  
    target_size=(150,150),  
    batch_size=batch_size,  
    class_mode='binary')  
val_generator = val_datagen.flow_from_directory(  
    'data_new/validation',  
    target_size=(150,150),  
    batch_size=batch_size,  
    class_mode='binary')  
test_generator = test_datagen.flow_from_directory(  
    'data_new/test',  
    target_size=(150,150),  
    batch_size=batch_size,  
    class_mode='binary')
```

### 3.3. Multi-class classification

```
'data_new/test ',
    target_size=(150,150),
    batch_size=batch_size,
    class_mode='binary ')
# train model on generator with batch size = 32
train_steps = train_generator.n // batch_size
val_steps = val_generator.n // batch_size
test_steps = test_generator.n // batch_size
```

The neural network was then trained and the results were subsequently saved to a csv file.

```
# Step-3: train our model
history = model.fit_generator(
    train_generator,
    steps_per_epoch=2000 // batch_size,
    epochs=50,
    validation_data=val_generator,
    validation_steps=800 // batch_size)

# Step-4: evaluate model's performance on
#train/eval/test datasets
acc = history.history['accuracy'][-1]
val_acc = history.history['val_accuracy'][-1]
loss = history.history['loss'][-1]
val_loss = history.history['val_loss'][-1]

cols = ["batch_size", "lr", "opt", "acc", "val_acc",
        "loss", "val_loss"]
pp = [parameters[0], parameters[1], parameters[2],
      round(acc,2), round(val_acc,2), round(loss,2),
      round(val_loss,2)]

df = pd.DataFrame([pp], columns = cols)
df.to_csv('VGG16_binary.csv', mode='a', header=False)
```

### 3.3. Multi-class classification

Once we obtained a classifier that successfully addressed issues connected with a *binary classification* issue, we moved on to a *multiclass classification*. In particular, a dataset of 41818 images was created for the *training phase*. These images were made in order to create ten different classes; five colours were

Chapter 3. Materials and methods

selected: white, black, red, light grey and dark grey; for each colour, two classes were chosen, one that identified the separation between two sheet metal parts and another class that represented the separation between sheet metal and the headlight.

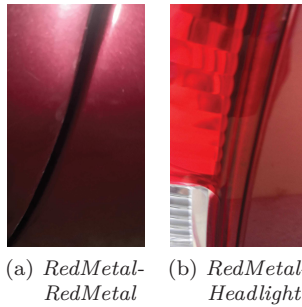


Figure 3.7.: Red car images dataset



Figure 3.8.: Comparison of white and light gray car dataset images

Figures 3.7, 3.8 and 3.9 show examples of the images used for the dataset. As the black and dark grey and the white and light grey colours were very similar in tone, this posed a serious problem to the ability of the neural network to recognize classes efficiently, an issue that sometimes even the human eye finds

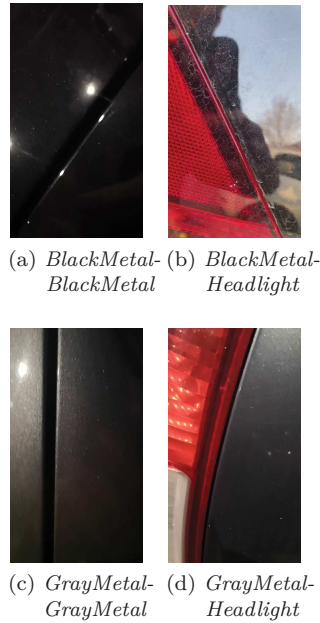


Figure 3.9.: Comparison of black and dark gray car dataset images

difficult to deal with.

### 3.3.1. VGG16 base model

Given the excellent results obtained by using the *VGG16 architecture* as a basic model within *binary classification*, it was then decided to use the very same basic model for *multiclass classification* as well.

The code used is similar to that employed for *binary classification*. The *hyperparameters* that have been iterated are the same: *batch size* of 32 and 64, *learning rate* of 0.0001 and 0.00001. Adadelta, Adagrad, RMSprop, SGD and Adam mathematical models were used as *optimizers*.

```

classes = os.listdir('data_new_1/train')
num_classes = len(classes)

vgg_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))

batch_size = [32, 64]
lr = [0.0001, 0.00001]
opt = [0, 1, 2, 3, 4]

```

```

p = [batch_size, lr, opt]
import itertools
t = list(itertools.product(*p))

parameters = t[0]

for parameters in t:

    if parameters[2]==0:
        optimizer=Adam(parameters[1])
    if parameters[2]==1:
        optimizer=Adagrad(parameters[1])
    if parameters[2]==2:
        optimizer=SGD(parameters[1])
    if parameters[2]==3:
        optimizer=RMSprop(parameters[1])
    if parameters[2]==4:
        optimizer=Adadelta(parameters[1])

```

When the various *layers* were added to the basic model, it was necessary to insert *softmax* (required by a *multiclass classification* instead *sigmoid*) as an *activation function*. Furthermore a *categorical crossentropy* had to be chosen as the *loss function*, instead of *binary crossentropy*.

```

def build_model():
    model = tf.keras.models.Sequential([
        # our vgg16_base model added as a layer
        vgg_base,
        # here is our custom prediction layer (same as before)
        Flatten(),
        Dropout(0.50),
        Dense(1024, activation='relu'),
        Dropout(0.20),
        Dense(512, activation='relu'),
        Dropout(0.10),
        Dense(num_classes, activation='softmax')
    ])
    vgg_base.trainable = True

    model.compile(optimizer=optimizer,

```

### 3.3. Multi-class classification

```
        loss='categorical_crossentropy',
        metrics=['accuracy'])
    return model
```

```
batch_size=parameters[0]
```

```
model = build_model()
```

The *data augmentation* parameters to build the *image data generator* were the same as those used for *binary classification*, changing only the class mode from *binary* to *categorical*. The network was trained with 20 *epochs* and the results were saved in a csv file.

```
# build our image data generators —
datagen = ImageDataGenerator(
    validation_split=0.3,
    rescale=1.0/255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

# Step-2: create generators, which 'flow' from directories
# create the generators pointing to folders created above
# ALL IMAGES TO BE RESIZED to (150,150,3)

train_generator = datagen.flow_from_directory(
    'data_new_1/train',
    subset = 'training',
    target_size=(150,150),
    batch_size=batch_size,
    class_mode='categorical'
)

val_generator = datagen.flow_from_directory(
    'data_new_1/train',
    subset = 'validation',
    target_size=(150,150),
```

### Chapter 3. Materials and methods

```
        batch_size=batch_size ,
        class_mode='categorical'
    )

test_generator = datagen.flow_from_directory(
    'data_new_1/test',
    target_size=(150,150),
    batch_size=batch_size ,
    class_mode='categorical'
)

# train model on generator with batch size = batch size
train_steps = train_generator.n // batch_size
val_steps = val_generator.n // batch_size
test_steps = test_generator.n // batch_size

# Step-3: train our model
history=model.fit_generator(
    train_generator ,
    steps_per_epoch=train_steps ,
    epochs=20,
    validation_data=val_generator ,
    validation_steps=val_steps

)

# Step-4: evaluate model's performance on
#train/eval/test datasets
acc = history.history['accuracy'][-1]
val_acc = history.history['val_accuracy'][-1]
loss = history.history['loss'][-1]
val_loss = history.history['val_loss'][-1]

cols = ["batch_size", "lr", "opt", "acc", "val_acc",
        "loss", "val_loss"]
pp = [parameters[0], parameters[1], parameters[2],
      round(acc,2), round(val_acc,2), round(loss,2),
```



```

round(val_loss,2)]

df = pd.DataFrame([pp], columns = cols)
df.to_csv('VGG16_multiclass.csv', mode='a', header=False)

```

### 3.3.2. Inception models

In an attempt to achieve better results from those obtained by using the *VGG16 architecture* as a basic model, the classifier was modified, inserting firstly the *Inception Res Net V2* and subsequently the *Inception V3*. Having obtained the best result with *InceptionV3* as a base model, utilizing a *batch size* of 32, a *learning rate* of 0.0001 and the SGD as *optimizer*, an analysis with 200 *epochs* was performed with these *hyperparameters*.

#### InceptionV3 with SGD optimizer

The final code used was similar to the previous ones. In this case, however, the various combinations of *hyperparameters* weren't iterated; instead, the values of the *hyperparameters* which led to the best result were used: *batch size* of 32, *learning rate* of 0.0001 and the SGD *optimizer*.

```

classes = os.listdir('data_new_2/train')
num_classes = len(classes)

model_base = InceptionV3(weights='imagenet',
                          include_top=False,
                          input_shape=(150, 150, 3))

batch_size = 32
lr = 0.0001
opt = SGD(lr)

def build_model():
    model = tf.keras.models.Sequential([
        # our vgg16_base model added as a layer
        model_base,
        # here is our custom prediction layer
        #(same as before)
        Flatten(),

```

### Chapter 3. Materials and methods

```
        Dropout(0.50),
        Dense(1024, activation='relu'),
        Dropout(0.20),
        Dense(512, activation='relu'),
        Dropout(0.10),
        Dense(num_classes, activation='softmax')
    ])
    model_base.trainable = True

    model.compile(optimizer=opt,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

batch_size=batch_size

model = build_model()

# build our image data generators —
datagen = ImageDataGenerator(
    validation_split=0.3,
    rescale=1.0/255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

# Step-2: create generators, which 'flow' from directories
# create the generators pointing to folders created above
# ALL IMAGES TO BE RESIZED to (150,150,3)

train_generator = datagen.flow_from_directory(
    'data_new_2/train',
    subset = 'training',
    target_size=(150,150),
    batch_size=batch_size,
    class_mode='categorical')
```

### 3.3. Multi-class classification

```
)

val_generator = datagen.flow_from_directory(
    'data_new_2/train',
    subset = 'validation',
    target_size=(150,150),
    batch_size=batch_size,
    class_mode='categorical'
)

# train model on generator with batch size = batch size
train_steps = train_generator.n // batch_size
val_steps = val_generator.n // batch_size

# Step-3: train our model
history=model.fit(
    train_generator,
    steps_per_epoch=train_steps,
    epochs=250,
    validation_data=val_generator,
    validation_steps=val_steps,
)

model.save('model.h5')

# Step-4: evaluate model's performance on train/eval datasets
acc = history.history['accuracy'][-1]
val_acc = history.history['val_accuracy'][-1]
loss = history.history['loss'][-1]
val_loss = history.history['val_loss'][-1]
```

### Chapter 3. Materials and methods

```
cols = ["batch_size", "lr", "opt", "acc", "val_acc",  
        "loss", "val_loss"]  
pp = [batch_size, lr, opt, round(acc, 2), round(val_acc, 2),  
      round(loss, 2), round(val_loss, 2)]  
  
df = pd.DataFrame([pp], columns = cols)  
df.to_csv('InceptionV3_SGD.csv', mode='a', header=False)
```

In addition to saving the results on a csv file, the accuracy and loss trends in the training and validation phases were plotted.

```
# list all data in history  
print(history.history.keys())  
# summarize history for accuracy  
plt.plot(history.history['accuracy'])  
plt.plot(history.history['val_accuracy'])  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train', 'validation'], loc='upper left')  
plt.show()  
# summarize history for loss  
plt.plot(history.history['loss'])  
plt.plot(history.history['val_loss'])  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train', 'validation'], loc='upper left')  
plt.show()
```

All the achieved results are shown in the next chapter.

## Chapter 4.

### Results and conclusion

With the tests and tools utilized as described in the third chapter, the following figures reported in the grids below were obtained. The results achieved with the initial dataset of the *cats and dogs* case are shown in the tables in appendix A, in which the "lr" is the *learning rate* of the classifier, the "acc" and the "val acc" indicate the *accuracy* for the *train set* and the *validation set*, and the "loss" and the "val loss" point out the *loss* for *training* and *validation set*.

The table below shows an example of the results obtained during a the first sample test carried out using the *VGG16* as the basic model:

Table 4.1.: VGG16 results for cats and dogs dataset

Model	batch size	lr	opt	acc	val acc	loss	val loss
VGG16	32	0.001	SGD	0.53	0.55	0.69	0.69
VGG16	32	0.001	Adam	0.5	0.49	0.69	0.69
VGG16	32	0.001	Adadelta	0.56	0.55	0.69	0.69
VGG16	32	0.001	Adagrad	0.51	0.49	0.69	0.69
VGG16	32	0.001	RMSprop	0.49	0.49	0.69	0.69
VGG16	32	0.0001	SGD	0.49	0.51	0.69	0.69
VGG16	32	0.0001	Adam	0.51	0.5	0.69	0.69
VGG16	32	0.0001	Adadelta	0.51	0.48	0.69	0.69
VGG16	32	0.0001	Adagrad	0.49	0.47	0.69	0.69
VGG16	32	0.0001	RMSprop	0.5	0.51	0.69	0.69
VGG16	64	0.001	SGD	0.5	0.47	0.69	0.69
VGG16	64	0.001	Adam	0.51	0.49	0.69	0.69
VGG16	64	0.001	Adadelta	0.49	0.48	0.69	0.69
VGG16	64	0.001	Adagrad	0.51	0.51	0.69	0.69
VGG16	64	0.001	RMSprop	0.5	0.48	0.69	0.69
VGG16	64	0.0001	SGD	0.52	0.5	0.69	0.69
VGG16	64	0.0001	Adam	0.5	0.52	0.69	0.69
VGG16	64	0.0001	Adadelta	0.49	0.51	0.69	0.69
VGG16	64	0.0001	Adagrad	0.5	0.49	0.69	0.69
VGG16	64	0.0001	RMSprop	0.53	0.51	0.69	0.69

#### Chapter 4. Results and conclusion

The table below shows the results obtained for the *binary classification* using *VGG16* as a basic model:

Table 4.2.: VGG16 results for binary classification

Model	batch size	lr	opt	acc	val acc	loss	val loss
VGG16	32	0.0001	Adam	0.99	0.99	0.02	0.05
VGG16	32	0.0001	Adagrad	1.0	0.99	0.0	0.08
VGG16	32	0.0001	SGD	1.0	0.99	0.0	0.09
VGG16	32	0.0001	RMSprop	0.57	0.62	0.95	0.6
VGG16	32	0.0001	Adadelta	0.5	0.51	0.7	0.69
VGG16	32	0.00001	Adam	0.5	0.5	0.69	0.69
VGG16	32	0.00001	Adagrad	0.5	0.5	0.69	0.69
VGG16	32	0.00001	SGD	0.5	0.5	0.69	0.69
VGG16	32	0.00001	RMSprop	0.5	0.5	0.69	0.69
VGG16	32	0.00001	Adadelta	0.5	0.5	0.69	0.69

The code was stopped after ten *iterations* since the results obtained with the first combinations of *hyperparameters* were considered to be more than satisfactory. It can also be noted that the best results were obtained with a *batch size* of 32, a *learning rate* of 0.0001, using Adam, Adagrad or SGD *optimizers*.

Once the *binary classification* problem was solved, we moved on to address the *multiclass* issue. Table 4.3 shows the results of the network for the *multiclass classification* with the *VGG16* base model.

Using Adam, Adagrad and SGD mathematical models as *optimizers*, with a *batch size* of 32 and a *learning rate* of 0.0001 the network achieved satisfactory results. The *accuracy* in the *training phase* was, in fact, very high, with values between 0.99 for the SGD and the Adam and 1.0 for the Adagrad; also the *loss*, during *training*, is satisfactory, with value of 0.04 with the SGD and 0.02 for the Adam and the Adagrad *optimizers*.

During the *validation phase* there was a slight reduction of efficiency, which led to *accuracy* values of 0.94 for all *optimizer* and *loss* of 0.49 for the Adam *optimizer* and 0.25 for the SGD and Adagrad.

Table 4.3.: VGG16 results for multiclass classification

Model	batch size	lr	opt	acc	val acc	loss	val loss
VGG16	32	0.0001	Adam	0.99	0.94	0.02	0.49
VGG16	32	0.0001	Adagrad	1.0	0.94	0.02	0.25
VGG16	32	0.0001	SGD	0.99	0.94	0.04	0.25
VGG16	32	0.0001	RMSprop	1.0	0.9	0.04	149.88
VGG16	32	0.0001	Adadelta	0.9	0.89	0.88	2.96
VGG16	32	0.00001	Adam	1.0	0.94	0.0	1.75
VGG16	32	0.00001	Adagrad	1.0	0.94	0.15	0.69
VGG16	32	0.00001	SGD	0.98	0.94	0.36	0.51
VGG16	32	0.00001	RMSprop	1.0	0.95	0.0	1.68
VGG16	32	0.00001	Adadelta	0.41	0.66	1.81	1.59
VGG16	64	0.0001	Adam	1.0	0.94	0.01	0.39
VGG16	64	0.0001	Adagrad	1.0	0.94	0.03	0.21
VGG16	64	0.0001	SGD	1.0	0.94	0.09	0.24
VGG16	64	0.0001	RMSprop	1.0	0.94	0.04	1.71
VGG16	64	0.0001	Adadelta	0.92	0.9	1.04	1.09
VGG16	64	0.00001	Adam	1.0	0.95	0.0	0.41
VGG16	64	0.00001	Adagrad	1.0	0.95	0.21	0.29
VGG16	64	0.00001	SGD	0.97	0.94	0.52	0.5
VGG16	64	0.00001	RMSprop	1.0	0.95	0.0	0.68
VGG16	64	0.00001	Adadelta	0.48	0.8	1.72	1.54

In an attempt to achieve even better results than those obtained with the *VGG16 architecture* as a basic model, the classifier was modified, inserting first the *Inception Res Net V2* and then the *Inception V3* as basic models. The results obtained with the *inception* base models are shown in tables 4.4 and 4.5

Table 4.4.: InceptionResNetV2 results for multiclass classification

Model	batch size	lr	opt	acc	val acc	loss	val loss
InceptionResNetV2	32	0.0001	Adam	0.98	0.94	0.05	0.31
InceptionResNetV2	32	0.0001	Adagrad	0.99	0.93	0.03	0.43
InceptionResNetV2	32	0.0001	SGD	0.99	0.93	0.04	0.44
InceptionResNetV2	32	0.0001	RMSprop	0.51	0.38	3.69	4.96
InceptionResNetV2	32	0.0001	Adadelta	0.2	0.26	2.54	2.21
InceptionResNetV2	32	0.00001	Adam	0.95	0.9	0.18	0.64
InceptionResNetV2	32	0.00001	Adagrad	0.59	0.64	1.28	1.12
InceptionResNetV2	32	0.00001	SGD	0.62	0.65	1.13	1.11
InceptionResNetV2	32	0.00001	RMSprop	0.96	0.91	0.18	1.28
InceptionResNetV2	32	0.00001	Adadelta	0.14	0.19	4.17	2.69
InceptionResNetV2	64	0.0001	Adam	0.18	0.18	2.21	2.21
InceptionResNetV2	64	0.0001	Adagrad	0.18	0.18	2.3	2.3
InceptionResNetV2	64	0.0001	SGD	0.18	0.18	2.29	2.29
InceptionResNetV2	64	0.0001	RMSprop	0.18	0.18	2.21	2.21
InceptionResNetV2	64	0.0001	Adadelta	0.18	0.18	2.3	2.3
InceptionResNetV2	64	0.00001	Adam	0.18	0.18	2.28	2.28
InceptionResNetV2	64	0.00001	Adagrad	0.18	0.18	2.3	2.3
InceptionResNetV2	64	0.00001	SGD	0.18	0.18	2.3	2.3
InceptionResNetV2	64	0.00001	RMSprop	0.18	0.18	2.28	2.28
InceptionResNetV2	64	0.00001	Adadelta	0.18	0.18	2.3	2.3

Table 4.5.: InceptionV3 results for multiclass classification

Model	batch size	lr	opt	acc	val acc	loss	val loss
InceptionV3	32	0.0001	Adam	0.99	0.94	0.04	0.26
InceptionV3	32	0.0001	Adagrad	0.99	0.95	0.03	0.21
InceptionV3	32	0.0001	SGD	0.99	0.95	0.04	0.18
InceptionV3	32	0.0001	RMSprop	0.99	0.94	0.07	0.53
InceptionV3	32	0.0001	Adadelta	0.89	0.89	1.07	1.0
InceptionV3	32	0.00001	Adam	1.0	0.95	0.0	0.34
InceptionV3	32	0.00001	Adagrad	1.0	0.95	0.27	0.3
InceptionV3	32	0.00001	SGD	0.98	0.95	0.57	0.48
InceptionV3	32	0.00001	RMSprop	1.0	0.95	0.01	0.59
InceptionV3	32	0.00001	Adadelta	0.99	0.94	0.03	0.28
InceptionV3	64	0.0001	Adam	1.0	0.95	0.03	0.23
InceptionV3	64	0.0001	Adagrad	0.99	0.95	0.05	0.2
InceptionV3	64	0.0001	SGD	0.99	0.95	0.02	0.64
InceptionV3	64	0.0001	RMSprop	0.89	0.89	1.4	1.36
InceptionV3	64	0.0001	Adadelta	1.0	0.95	0.0	0.36
InceptionV3	64	0.00001	Adam	1.0	0.95	0.27	0.34
InceptionV3	64	0.00001	Adagrad	0.95	0.94	0.74	0.69
InceptionV3	64	0.00001	SGD	1.0	0.95	0.0	0.67
InceptionV3	64	0.00001	RMSprop	0.46	0.67	1.77	1.56

Having obtained the even better result with the *InceptionV3* as a base model, adopting a *batch size* of 32, a *learning rate* of 0.0001 and the SGD as *optimizer*, a test with 200 *epochs* was performed with these *hyperparameters*.

In order to have a better view of the results obtained, the graphs of the two fundamental parameters used to evaluate the performance of a *classification problem*, namely the *accuracy* and the *loss*, have been plotted. The *accuracy* and *loss* trends are shown in the figure 4.1

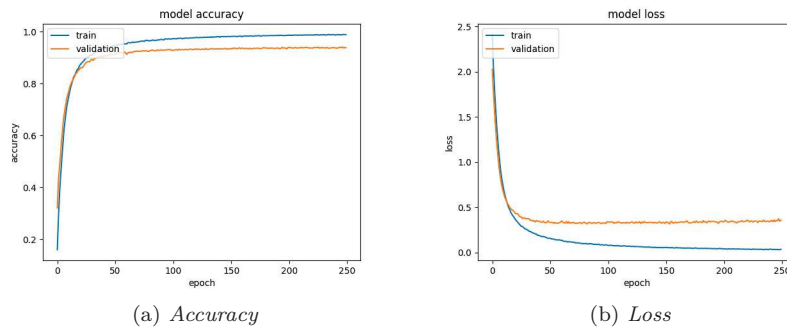


Figure 4.1.: Model loss and accuracy

It can be noted how the *accuracy* values show an increasing trend both for the *training* and *validation phase*, until a maximum value is reached. As for the *training*, the *accuracy* starts from very low values, close to zero, up to values



tending to the maximum, close to 0.98/0.99; as regards to the *validation phase*, values initially start higher than those registered during the *training phase*, to reach maximum values around 0.93/0.94.

The *loss* figures showed instead an opposite trend. They started with figures close to the maximum value and then set close to zero during the *training phase* and at 0.3 during the *validation phase*.

This difference in *accuracy* and *loss* values between the *training* and *validation* phases might be a symptom of a slight tendency of the model to *overfitting*, which could be partially addressed by trying to further generalize the starting dataset.

After evaluating the *accuracy* and *loss* of the *training* and *validation phases*, about 800 images were employed, which showed only white and dark grey cars. Around of a quarter of the 800 images were classified under "WhiteSheetmetal-WhiteSheetmetal" category, while the following 200 images belonged to the "WhiteSheetmetal-Headlight" class; images from 400 to 600 concerned the "DarkGreySheetmetal-Headlight" classification and finally, the last 200 images (from 600 to 800) dealt with the class "DarkGreySheetmetal-DarkGreySheetmetal". To evaluate the efficiency of the model, a matrix was extracted showing the probabilities for each photo to belong to a class. The code for extrapolate this *probability matrix* is reported below.

```
model = keras.models.load_model('model.h5')

# build our image data generators —
datagen = ImageDataGenerator(
    rescale=1.0/255)

#prediction test
test_generator = datagen.flow_from_directory(
    directory='data_new_2/test',
    target_size=(150, 150),
    class_mode=None,
    shuffle=False
)

# without this code weird outputs will come
test_generator.reset()

pred=model.predict(test_generator,verbose=1)
```

Chapter 4. Results and conclusion

```
# Running the above code will give output in probabilities
#so at first I need to convert them to class number
predicted_class_indices=np.argmax(pred , axis=1)
```

Parts of the probability matrix are shown below, in order to more easily understand how it works. At the top of the matrix, there are identification numbers of each class: 0 indicates the "WhiteSheetmetal-WhiteSheetmetal" class, the 1 the "WhiteSheetmetal-Headlight" class, the 2 the "GraySheetmetal-Headlight" class, the 3 the "GraySheetmetal-GraySheetmetal" class, the 4 the "DarkGraySheetmetal-Headlight" class, the 5 the "DarkGraySheetmetal-DarkGraySheetmetal" class, the 6 the "BlackSheetmetal-Headlight" class, the 7 the "BlackSheetmetal-BlackSheetmetal" class, the 8 the "RedSheetmetal-Headlight" class and the 9 the "RedSheetmetal-RedSheetmetal". In the left column, instead, the identification numbers of the photos of the test set are represented. As reported above, it can be considered indicatively that the first 200 photos should belong to class 0, photos from 200 to 400 should belong to class 1, photos from 400 to 600 should belong to class 4 and the last 200 photos (from 600 a 800) should belong to class 5. The probability values shown in the tables have been approximated to the second decimal place.

Table 4.6.: Probability matrix for test set (Images from 96 to 101)

Image	0	1	2	3	4	5	6	7	8	9
96	0.01	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00
97	0.01	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00
98	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
99	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
100	0.11	0.00	0.00	0.89	0.00	0.00	0.00	0.00	0.00	0.00
101	0.05	0.00	0.00	0.95	0.00	0.00	0.00	0.00	0.00	0.00

Table 4.7.: Probability matrix for test set (Images from 445 to 450)

Image	0	1	2	3	4	5	6	7	8	9
455	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
456	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
457	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
458	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
459	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
460	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00

Although there are no black and light grey cars in the test dataset, it can be noted that the model interpreted some colours of some vehicles as black and light grey. This happened because the classifier sometimes confused the white colour with light-grey, while the dark-grey colours were interpreted as black. This issue hasn't nevertheless posed a real problem for the purpose of the test

as the laser of the *G3F* would need to generate an almost identical power when faced with the black or dark grey colours. The same situation would apply for the colours white and light grey.

As already discussed in the previous chapters, for the realization of the project in question, it was fundamental in the beginning, to create a valid dataset. When working with *computer vision* in order to train a neural network and address *classification* problems in the most appropriate way, it is crucial to acquire a high number of images with the widest number of characteristics, colours and exposure. This method helps avoiding the creation of a *bias* in the architecture, which often leads the network to memorize almost perfectly the input images, without being able to generalize and recognize new images with different characteristics (*overfitting*).

The development of this project began with a research to find on the web a useful ready-made dataset. It is indeed possible today to find online various types of datasets and neural network architectures, which enable even those who don't have a vast experience in programming, to interface in a satisfactory manner with the vast world of *artificial intelligence*.

Once the tests with various basic architectures available on the web were completed employing a dataset of standard images of *cats and dogs*, we then created our own customized dataset.

Short videos were made on different models and colours of cars and at different times of the day. The frames were extracted from the videos and became the images of the dataset used to carry out the project.

The code used to extract the individual frames from the videos is shown below:

```
vidcap = cv2.VideoCapture('v1.mp4')
success , image = vidcap.read()
count = 0
while success :
    cv2.imwrite("frame" # count , image)
    success , image = vidcap.read()
    print ('Read_a_new_frame:' , success)
    count += 1
```

The final dataset, obtained by extrapolating each frame from each video, contains more than 40000 images. Such a large dataset has certainly proved a solid base for the creation of a good classifier.

Once a robust dataset is in place, it is then important to look for the most appropriate model for the project in order to obtain the best possible *classification*.

Instead of creating your own architecture from scratch, it is today possible,

thanks to the numerous opportunities offered by the web, to start with pre-existing *classification models*, such as the VGG16 and the InceptionV3, to see which of those models can provide the best result with the dataset utilized.

Another important step undertaken for this project was the creation of a program code, which allowed to iterate the various combinations of basic models and *hyperparameters* in order to effectively evaluate which would be the best combination in order to achieve the programmed objectives.

Once the most appropriate basic model with the relevant combination of *hyperparameters* is found, more *layers* can then be added to enhance the performance of the network. However by manipulating the system, unwanted side effects might appear that might result in a reduction in the efficiency and speed. *Overfitting* (when the network is too complex) or *underfitting* (when the model is too 'shallow' with not enough layers) are two issues linked to the dataset efficiency. As in most engineering project is therefore fundamental for a good engineer to find the right balance between the different components in order to achieve the best performance.

Each *machine learning* problem is unique. It is not possible therefore to find a priori the entire set of values for the most effective *hyperparameters*. It is instead a long process of trial and error, tests and experiments. A good knowledge of algorithm techniques, patience and utter dedication to work can put the project on the right track although the path to success might still be a long one.

The results that obtained with this work have been achieved thanks to the implementation of a number of tests and experiments. It was useful to start with a basic models, which was then modified by adding various layers to improve the efficiency of the network.

For the *binary classification*, the use of the VGG16 and an SGD *optimizer* as a basic model, with a *batch size* of 32 and a *learning rate* of 0.0001, brought very good results, with *accuracy* values close to unity and *loss* values tending to zero, both during the *train* and *validation* phases.

During the *multiclass classification*, in which the network was trained to classify ten different classes, the VGG16 was firstly used.

To further improve the results obtained in the first place, two *inception networks* were then subsequently employed. The final classifier was finally train to use a standard model InceptionV3 thanks to the best results obtained with this model. The final analysis showed an *accuracy* during the *training* phase of values close to 1, while in the *validation* phase, the results obtained were close to the 0.94 mark. The *loss* values instead set around zero during the *training* phase and shifted to 0.3 during the *validation* phase.

The effect caused by the slight decay of *accuracy* verified during the *validation* process caused by the tendency of the model to confuse very similar colours was nevertheless offset by the fact that the *G3F* laser would need to employ

a virtually identical power when the sensor is detecting those similar colours, thus making the results obtained very reliable for the purpose of our project. Another fundamental aspect that should be considered when dealing with *computer vision* is that of *data augmentation*. Through operations such as rotation, transformation, mirroring, etc. can be expanded thus allowing the network to better generalize new images.

Thanks to this kind of architecture based on *artificial convolutional neural networks*, it would be possible therefore to give further practical help to the operator, who's during the production phase utilizes the *G3F*.

Thanks to this kind of state-of-the-art *artificial intelligence*, it is possible today to effectively address problems that often arise with the loss of efficiency cause by human error during the different phase of production and quality checks.

This is what lies behind *Industry 4.0*. Technology far from being a replacement of human intellect and skills should be employed to be fully integrated during the different manufacturing phases interacting whenever possible with its human counterpart. To conclude, the work presented was based on the implementation of a *classification* system based on *convolutional neural networks* with the aim of being able to classify the various colours and the various parts of a vehicle. The project was devised as a support to the *G3F*, a portable device based on a *triangulation laser sensor*, which in the production phase, allows for measurement and control of *gaps and flushes*. Thanks to the architecture devised and utilized, it was possible to adjust the laser power of this device, according to the different characteristics of the vehicles used for the test. This in turn improved the effectiveness and accuracy of the device. Overall the results obtained can be considered to be excellent both with *binary* and *multiclass classifications*. The network was successfully trained to recognize ten different classes effectively. For the *binary classification* the VGG16 was adopted as the basic model, while for the recognition of the ten classes, the best results were those obtained by adopting an InceptionV3 with the SGD *optimizer*.

For the *multiclass* problem a test phase was also carried out, from which the *probability matrix* was extrapolated with mixed results. However as previously discussed, the use of a nearly identical laser power for similar colour mean that the test results proved to be more than reliable.

A fundamental aspect of this work can be found in the architecture developed for this project which can be used to classify all types of images. A change of datasets and the resetting of the parameters fine tuning should enable the network to learn entire new sets of *classifications* without any major issues.

Further developments of this work should be aimed at the study of *semantic segmentation*, which splits entire images into groups of pixels that can be subsequently labelled and classified individually. A simple *classification* system

#### Chapter 4. Results and conclusion

allows the network to recognize only what exist in the actual image, without taking into account the actual position of certain details within the picture that is getting classified. The *segmentation* instead, makes it possible for the *neural network* to be trained so that specific areas with intrinsically different characteristics can be identified as in the case of having to identify the edges separating two adjacent materials such as those of two metal-sheets or of a metal-sheet and a headlight.

Implementing the *segmentation* should drastically improve the effectiveness of the *G3F*, ensuring a better performance during the acquisition process.

# Appendix A.

## Results tables

Table A.1.: Xception results for cats and dogs dataset

Model	batch size	lr	opt	acc	val acc	loss	val loss
Xception	32	0.001	SGD	0.53	0.51	0.69	0.69
Xception	32	0.001	Adam	0.5	0.48	0.69	0.69
Xception	32	0.001	Adadelta	0.5	0.5	0.69	0.69
Xception	32	0.001	Adagrad	0.47	0.48	0.69	0.69
Xception	32	0.001	RMSprop	0.5	0.51	0.69	0.69
Xception	32	0.0001	SGD	0.5	0.5	0.69	0.69
Xception	32	0.0001	Adam	0.5	0.52	0.69	0.69
Xception	32	0.0001	Adadelta	0.5	0.51	0.69	0.69
Xception	32	0.0001	Adagrad	0.5	0.51	0.69	0.69
Xception	32	0.0001	RMSprop	0.48	0.47	0.69	0.69
Xception	64	0.001	SGD	0.5	0.5	0.69	0.69
Xception	64	0.001	Adam	0.46	0.51	0.69	0.69
Xception	64	0.001	Adadelta	0.5	0.48	0.69	0.69
Xception	64	0.001	Adagrad	0.48	0.43	0.69	0.69
Xception	64	0.001	RMSprop	0.51	0.52	0.69	0.69
Xception	64	0.0001	SGD	0.5	0.46	0.69	0.69
Xception	64	0.0001	Adam	0.52	0.48	0.69	0.69
Xception	64	0.0001	Adadelta	0.5	0.5	0.69	0.69
Xception	64	0.0001	Adagrad	0.5	0.53	0.69	0.69
Xception	64	0.0001	RMSprop	0.49	0.53	0.69	0.69

Appendix A. Results tables

Table A.2.: InceptionResNetV2 results for cats and dogs dataset

Model	batch size	lr	opt	acc	val acc	loss	val loss
InceptionResNetV2	32	0.001	SGD	0.51	0.53	0.7	0.7
InceptionResNetV2	32	0.001	Adam	0.5	0.49	0.69	0.69
InceptionResNetV2	32	0.001	Adadelta	0.49	0.49	0.69	0.69
InceptionResNetV2	32	0.001	Adagrad	0.45	0.45	0.69	0.69
InceptionResNetV2	32	0.001	RMSprop	0.49	0.5	0.69	0.69
InceptionResNetV2	32	0.0001	SGD	0.53	0.5	0.69	0.69
InceptionResNetV2	32	0.0001	Adam	0.49	0.52	0.69	0.69
InceptionResNetV2	32	0.0001	Adadelta	0.49	0.46	0.69	0.69
InceptionResNetV2	32	0.0001	Adagrad	0.5	0.53	0.69	0.69
InceptionResNetV2	32	0.0001	RMSprop	0.48	0.5	0.69	0.69
InceptionResNetV2	64	0.001	SGD	0.51	0.49	0.69	0.69
InceptionResNetV2	64	0.001	Adam	0.49	0.52	0.69	0.69
InceptionResNetV2	64	0.001	Adadelta	0.47	0.44	0.69	0.69
InceptionResNetV2	64	0.001	Adagrad	0.49	0.49	0.69	0.69
InceptionResNetV2	64	0.001	RMSprop	0.51	0.51	0.69	0.69
InceptionResNetV2	64	0.0001	SGD	0.5	0.49	0.69	0.69
InceptionResNetV2	64	0.0001	Adam	0.48	0.51	0.69	0.69
InceptionResNetV2	64	0.0001	Adadelta	0.51	0.52	0.69	0.69
InceptionResNetV2	64	0.0001	Adagrad	0.51	0.49	0.69	0.69
InceptionResNetV2	64	0.0001	RMSprop	0.51	0.5	0.69	0.69

Table A.3.: InceptionV3 results for cats and dogs dataset

Model	batch size	lr	opt	acc	val acc	loss	val loss
InceptionV3	32	0.001	SGD	0.48	0.52	0.7	0.71
InceptionV3	32	0.001	Adam	0.5	0.48	0.69	0.69
InceptionV3	32	0.001	Adadelta	0.49	0.52	0.69	0.69
InceptionV3	32	0.001	Adagrad	0.5	0.49	0.69	0.69
InceptionV3	32	0.001	RMSprop	0.5	0.51	0.69	0.69
InceptionV3	32	0.0001	SGD	0.49	0.52	0.69	0.69
InceptionV3	32	0.0001	Adam	0.48	0.48	0.69	0.69
InceptionV3	32	0.0001	Adadelta	0.52	0.53	0.69	0.69
InceptionV3	32	0.0001	Adagrad	0.49	0.51	0.69	0.69
InceptionV3	32	0.0001	RMSprop	0.49	0.54	0.69	0.69
InceptionV3	64	0.001	SGD	0.51	0.5	0.69	0.74
InceptionV3	64	0.001	Adam	0.52	0.49	0.69	0.69
InceptionV3	64	0.001	Adadelta	0.48	0.51	0.69	0.69
InceptionV3	64	0.001	Adagrad	0.52	0.5	0.69	0.69
InceptionV3	64	0.001	RMSprop	0.52	0.48	0.69	0.69
InceptionV3	64	0.0001	SGD	0.5	0.49	0.69	0.69
InceptionV3	64	0.0001	Adam	0.5	0.51	0.69	0.69
InceptionV3	64	0.0001	Adadelta	0.5	0.48	0.69	0.69
InceptionV3	64	0.0001	Adagrad	0.52	0.55	0.69	0.69
InceptionV3	64	0.0001	RMSprop	0.5	0.49	0.69	0.69



Table A.4.: ResNet50 results for cats and dogs dataset

Model	batch size	lr	opt	acc	val acc	loss	val loss
ResNet50	32	0.001	SGD	0.52	0.49	0.7	0.7
ResNet50	32	0.001	Adam	0.49	0.5	7.67	7.67
ResNet50	32	0.001	Adadelta	0.48	0.5	0.69	0.7
ResNet50	32	0.001	Adagrad	0.48	0.51	0.7	0.69
ResNet50	32	0.001	RMSprop	0.51	0.52	7.67	7.67
ResNet50	32	0.0001	SGD	0.53	0.49	0.69	0.69
ResNet50	32	0.0001	Adam	0.49	0.52	0.69	0.69
ResNet50	32	0.0001	Adadelta	0.47	0.52	0.69	0.7
ResNet50	32	0.0001	Adagrad	0.5	0.5	0.69	0.69
ResNet50	32	0.0001	RMSprop	0.49	0.49	0.69	0.69
ResNet50	64	0.001	SGD	0.51	0.52	0.69	0.69
ResNet50	64	0.001	Adam	0.5	0.5	0.69	0.69
ResNet50	64	0.001	Adadelta	0.48	0.49	0.69	0.69
ResNet50	64	0.001	Adagrad	0.48	0.48	0.69	0.69
ResNet50	64	0.001	RMSprop	0.48	0.54	0.69	0.69
ResNet50	64	0.0001	SGD	0.52	0.51	0.69	0.69
ResNet50	64	0.0001	Adam	0.53	0.48	0.69	0.69
ResNet50	64	0.0001	Adadelta	0.49	0.47	0.69	0.69
ResNet50	64	0.0001	Adagrad	0.5	0.5	0.69	0.69
ResNet50	64	0.0001	RMSprop	0.47	0.45	0.69	0.69



## Bibliography

- [1] Nikhil Singh, Paras Ahuja, *Foundamentals of Deep Learning and Computer Vision*, bpb Publications.
- [2] Avishek Nag, *Pragmatic Machine Learning with Python*, bpb Publications.
- [3] Sebastian Raschka, Vahid Mirjalili, *Machine Learning con Python*, Apogeo.
- [4] Gianluca Rossi, *Misure meccaniche e termiche*, Carocci.
- [5] Nazzareno Bordi (2020), *Digital Transformation*, Advanced course in Industry 4.0, Università Politecnica delle Marche.
- [6] Nicola Paone (2020), *The role of measurements and sensors in Industry 4.0*, Advanced course in Industry 4.0, Università Politecnica delle Marche.
- [7] Elisa Minnetti, Paolo Chiariotti, Nicola Paone, Gisela Garcia, Helder Vicente, Luca Violini, Paolo Castellini, *A smartphone based hand-held gap&flush measurement system for in-line quality control of car body assembly*, Università Politecnica delle Marche, Department of Industrial Engineering and Mathematical Sciences, Via Breccie Bianche, 60131, Ancona, Italy, Volkswagen Autoeuropa, Q.ta do Anjo, 2954-024 Portugal.
- [8] Alessia Baleani, Paolo Chiariotti, Nicola Paone, Luca Violini, Paolo Castellini, *Analysis of reproducibility and repeatability of a hand-held laser scanner for gap&flush measurement in car-assembly line*, Università Politecnica delle Marche, Department of Industrial Engineering and Mathematical Sciences.
- [9] Elisa Minnetti, Paolo Chiariotti, Nicola Paone, Gisela Garcia, Helder Vicente, Luca Violini, Paolo Castellini, *A smart portable laser scanner for Gap&Flush measurment*, Università Politecnica delle Marche, Department of Industrial Engineering and Mathematical Sciences, Ancona, Italy, Volkswagen Autoeuropa, Palmela, Portugal.
- [10] Elisa Minnetti, Paolo Chiariotti, Nicola Paone, Gisela Garcia, Helder Vicente, Luca Violini, Paolo Castellini, *Smart portable laser triangulation system for assessing gap and flush in car body assembly line*, Università Politecnica delle Marche, Department of Industrial Engineering and

## Bibliography

Mathematical Sciences, Ancona, Italy, Volkswagen Autoeuropa, Palmela, Portugal.

- [11] Dr. Adrian Rosebrock, *Deep Learning for computer vision with Python*, pyimagesearch.
- [12] Dr. Adrian Rosebrock, *Practical Python and OpenCV*, pyimagesearch.
- [13] Nicola Paone, *Misure e controlli industriali*, Phd course for Industrial Engineering, Università Politecnica delle Marche, Ancona
- [14] Gianfranco Barone, *Online Course in Artificial Intelligence and Machine Learning*, Unione Professionisti
- [15] Karen Simonyan & Andrew Zisserman, *Very Deep Convolutional Networks for large-scale Image Recognition*, Visual Geometry Group, Department of Engineering Science, University of Oxford
- [16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, *Rethinking the Inception Architecture for Computer Vision*, Zbigniew Wojna, University College London
- [17] François Chollet, *Xception: Deep Learning with Depthwise Separable Convolutions*, Google, Inc.
- [18] *Machine learning e deep learning si spostano dal Cloud all'Edge: gli impatti sul settore manifatturiero*, <https://www.eurotech.com/it/news/edge-machine-learning-deep-learning>
- [19] <https://qastack.it>
- [20] <https://ichi.pro/it/funzioni-di-attivazione-99956793768697>
- [21] Muneeb ul Hassan, *VGG16-Convolutional Network for Classification and Detection*, <https://neurohive.io/en/popular-networks/vgg16/>
- [22] Jaejun Yoo, *Inception and Xception* <https://www.slideshare.net/thinkingfactory/pr12-inception-and-xception-jaejun-yoo>
- [23] Sik-Ho Tsang, *Review: GooLeNet (Inception v1)-Winner of ILSVRC 2014 (Image Classification)*, <https://medium.com/coinmonks>
- [24] <https://ichi.pro/it/bias-e-varianza-55598046545165>