Politecnico di Torino

Master of Science Degree in MECHATRONIC ENGINEERING

Master Thesis

Autonomous recognition and pose estimation in dusty environment for space application

Supervisors:
prof. Marcello Chiaberge

Candidate:
Gabriele Gennaro
S253757

Academic year 2019-2020

Abstract

Robotics plays a central role in space exploration since the beginning of the famous «space race». In particular, *Artificial Intelligence* development allowed several challenging missions and supported scientist and astronauts in the spacecraft control, autonomous navigation or in object detection tasks.
In 2018, NASA provided 330k $ to a researchers team, aiming to develop an intelligent system able to navigate amid the debris, detecting and avoiding them . During same year a partnership between NASA and Intel helped astronauts during their training phase for their mission. Researchers trained a neural network with thousands of Moon images in order to create a virtual moon environment and compare it with the local environment .
Nowadays one of the most significant space challenge for humanity is Mars exploration. In 1965 *Mariner 4* performed a movement called flyby, passing in the vicinity of Red Planet and capturing a close-up picture of it. That was the very first picture of Mars from so close. The Mariner was a robotic craft of small size and its purpose was to explore Mars,Venus and Mercury. During the years, martian mission have become more and more sophisticated , driven by incredible technological advancement especially in autonomous system. Successful examples of how intelligent machines play a pivotal role in space exploration are the Spirit and Opportunity rovers. Involving object detection algorithm and autonomous navigation, mars rovers have managed to perform geological analyses trough their portable laboratories on the spot . It is considered one of the most successful mission for NASA .
The evolution of the study of Mars characteristic has lead to "Mars Sample Return" mission. The scope of this mission is to pick samples , pre-filled with martian soil and gas, and bring them to Earth, in order to analyse samples with more complex instruments than the previous mission. This project represents a fundamental step towards an overall comprehension of the past history of Mars and the possibility to contemplate a human colonization.
In this thesis project, an autonomous recognition system is proposed as possible solution for the picking task of the MSR rover. The system has to detect the sample and perform a pose estimation in order to collect information for the picking phase. Technological limits related to spaceflight and Mars environment factors have been taken into account aiming to create a system designed for a real application .

Contents

Chapter 1

Introduction

1.1 Research objective

The aim of this thesis project is to develop an *autonomous recognition and pose estimation* system for *Mars Sample Return* mission. The latter is the last programmed Mars mission and it will take during the 2026 launch window. It is the evolution of the famous Mars Exploration Rover mission in which martian rovers had to perform geological analyses on soil and rocks on Mars ground trough their portable laboratory.
In this new challenging mission researchers's point of view is totally changed : the analyses will not take place on the spot , but they will be carried on in laboratories on Earth. In fact, during the mission, rover's purpose will be to pick samples filled with martian soil and rocks and bring them back on Earth. So far, no sample has been studied on Earth and it is an hard challenge due to numerous risk factors.
With the purpose to develop a feasible solution, martian environment conditions and technological spaceflight constraints represent an hard obstacle to overcome considering peculiar limits, e.g. a space qualified on-board computer to work with that can bear a spaceflight and related cosmic rays, martian lighting condition and a dust environment with frequent extreme weather phenomena.

1.2 Research project background

1.2.1 Company introduction

Thales Alenia Space is a joint-venture specialized in the sector of space telecommunication equipment, satellite-based system and space exploration; for example, its contributions has been fundamental for the International Space Station realization: many modules construction began in its factory in Turin such as *Columbus* laboratory in figure 1.1 .

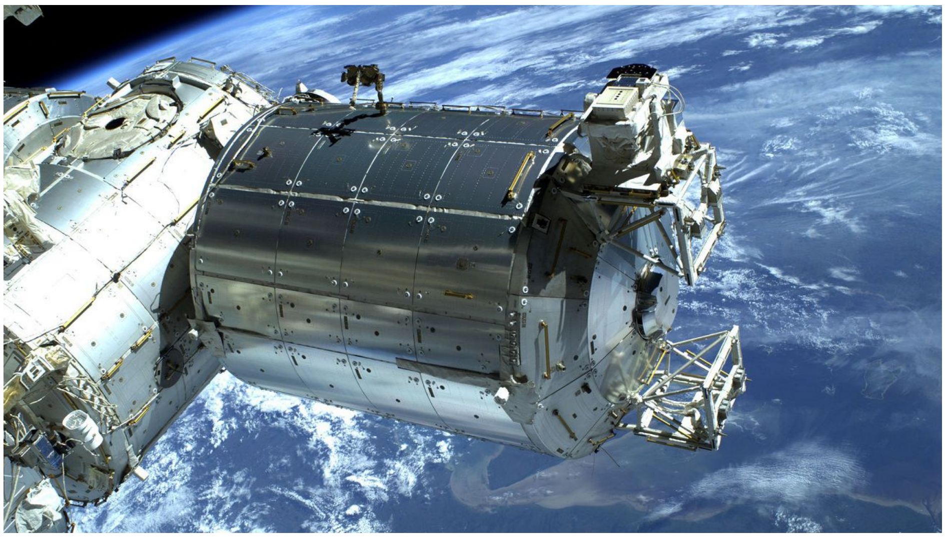


FIGURE 1.1: Columbus Laboratory: it was the first permanent European research facility in space. [1]

The company has invested a large amount of its profits in R&D area in order to support innovations in space related sectors, involving academic and industrial partners. Thales owns several offices settled all over in Italy within the main cities, but the most important one for this research project has been the ones in Turin and its linkages with Turin Polytechnic . Indeed, thanks to Thales , for the purpose of this research thesis , a company's facility located in Turin district has been used, aiming at the development of an object detection system, useful for Sample Return Mission in which Thales is involved as one of the main player.

1.2.2 *Mars Sample Return* mission

Mars Sample Return is the most ambitious mission towards Mars investigation that Esa and Nasa programmed involving Red Planet: up to this mission, no Mars material was ever brought back to the Earth. Previous missions focus was to study Mars on Mars itself: the first important mission *Viking*(1975) was expected to investigate Mars atmosphere composition and look for traces of life through a scientific lab equipped on the lander. With *Path Finder-Sojourner* and *Spirit and Opportunity* missions, the scientific laboratory was able to move around thanks to the rovers, where it was fixed on. The lander was in charge to protect the rovers during the landing phase while the rovers aimed at extracting soil pieces and interacting with martian environment .
Last and most important mission is the *Mars Sample Return* due to its high technical difficulty and a purpose never achieved before. Focus of the mission is to bring back martian soil and other interesting elements like martian gas to the Earth so they can be analyzed through processes that cannot be reproduced on Mars. Thanks to the orbiter *MSR-ERO*, a lander will be released aimed at protecting the *Sample Fetching Rover*(SFR) inside. The rover should fetch the samples containing martian elements, collected by a previous rover of *Mars 2020*. Once collected 36 of the 43 samples, they

will suppose to be carried to the landing site and then to the orbiter by the *Ascent veichle* . Their final journey will be the one towards the Earth. According to mission details, the samples should be spread in a predefined region. Communication between Earth and Mars needs at least 20 minutes in one way, so it means that a teleoperation an its confirm should need at least 40 minutes. Moreover, due to *visibility constraints*, only few communication per sol are possible. So it's crucial to involve a certain "autonomy" for the system during the fetching phase and this research project aims to propose a possible solution considering all the aformentioned limits including technological space constraints.

1.2.3 *R.O.X.Y.* facility

ROvers eXploration facilitY is a technological area located inside Thales's factory in Turin aimed at simulating a real martian environment for robotic systems design, development, validation and verification in a real-world weather conditions, where especially visual algorithms are stressed. It covers an area of 400 square-meters of red soil and rocks, similar to those that a rover would find on the Red Planet. Even the composition, sizes, shapes and distributions of the rocks are meant for emulate the real conditions. The facility was set up in order to train three rovers and study the navigation in a real scenario.
In this research project Roxy facility is used in order to perform the detection of the target as it was on Mars, trying to reproduce all the possible scenarios .

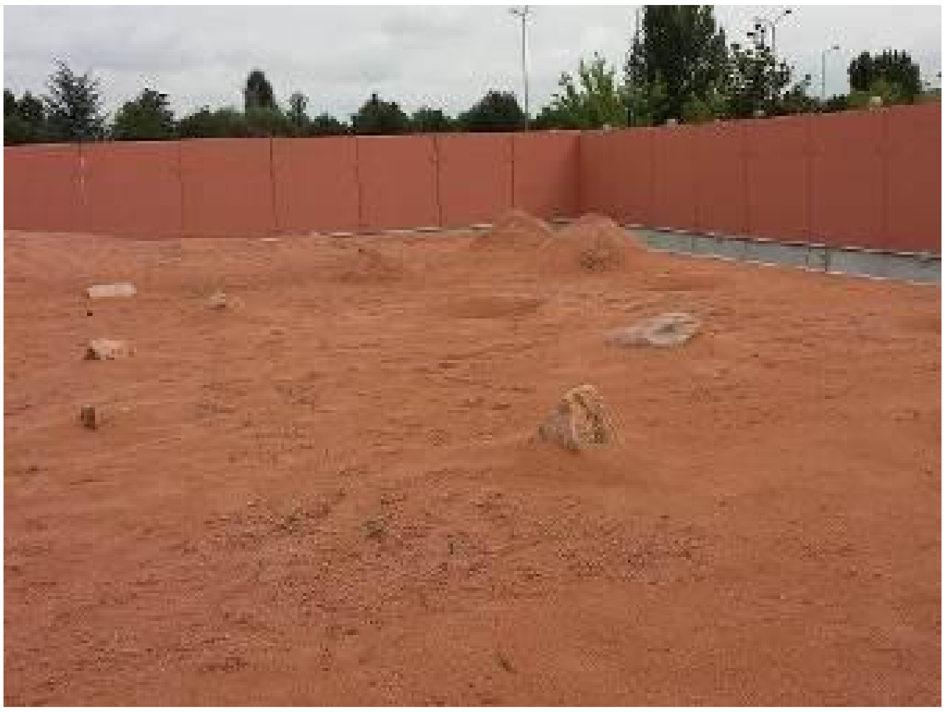


FIGURE 1.2: Internal view of ROXY area .

Chapter 2

Preliminary studies

2.1 Mars conditions

For the purpose of studying which would be the possible limits of an autonomous detection system application on Mars, it is worth highlighting relevant environmental factors . Two of the most important aspects to focus on are related to atmospheric phenomena and lighting conditions , so in this section some details about Mars will be introduced.
Mars mass is 11% of the Earth mass and it is 50% more distant from the Sun than our planet. According to this data and recalling the *intensity power for a light point source*

$$I(x) = \frac{P}{4\pi x^2}$$

it is easy to notice that sun intensity power on Mars is about 50% less than on Earth. Anyway, this is only an initial step for the comprehension of light conditions on Mars, because the atmosphere and its composition plays a central role with respect to what an observer would see on the surface of the planet . Due to its smaller mass, Mars has a thinner atmosphere than Earth , about 1% of it . Moreover , it is different in term of composition :the major part is composed by CO2, with some trace of water vapor and other chemical components like methane.
According to the details introduced above, some hypothesis can be made about solar radiations on the surface of Mars. The Sun emits radiations principally in the visible range (380 to 740 nm) , but there are also IR and UV components of its light . On the Earth the IR is absorbed and diffused by the water vapor, while the UV range in blocked by the atmosphere : only 1% of UV reaches the surface. On the Red Planet the CO2 plays the same role of the water vapor so that on its surface arrives approximately the same range of light in terms of visible and IR radiation . For what concerns the UV range, the CO2 manages to block almost completely the *UV-C*(280 to 100 nm), but it is not so effective for the rest of the range[2].
Another aspect of the lighting condition is the diffusion : despite the thin atmosphere and with dishomogeneous condition, including the sun elevation angle , light is well

diffused on Mars, so that shadow edges are not defined, creating the common soft transition of the shadow from dark to light .
The sky colour seems to be in the range of blue when sun is on the horizon and tend to become redder when the angle elevation is higher due to the dust presence. This latter is preponderant element of the extreme environment on Mars. In fact, everything is covered by a red dust made by iron oxide and similar chemical components. This peculiar element is present in the atmosphere,visible from the Earth and, cyclically, global dust clouds spread all over the atmosphere. On the surface, dust wind blows with a high velocities from 16 to 32 km/h : the Viking lander measured velocities up to 113 km/h during dust storms. In spring and summer, very often dust whirlwind suddenly compare. They are similar to those that compares in desertic area of the Earth but they are larger and can be traced as they left wide zone of darker soil bringing in the air dusts from the ground during their wake . This extreme event is usually known as *dust devil* and it is at the base of the formation of dunes and dark craters .

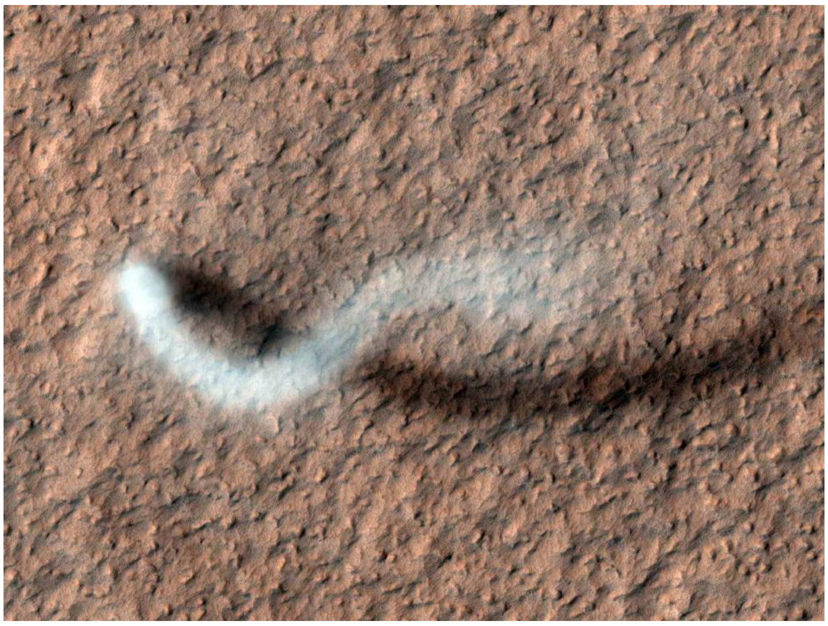


Figure 2.1: Serpent dust devil acquired by the High Resolution Imaging Science Experiment (HiRISE) camera on NASA's Mars Reconnaissance Orbiter. [3]

All this weather phenomena and environmental factors tend to obscure and absorb sun light so that on Mars planet surface receive less light than Earth and the environment would seems darker to a human eye. As an example, during one of the frequent dust storms, the lander of the *Mars 3* mission reported a photo of 50 lux of illuminance. For comparison during a sunset of a fully overcast day, the illuminance can reaches the same level .
A last interesting phenomenon is the presence of solid CO2 in a *falling snow event*, that due to the small size of its particles appears like dense fog.

2.2 Target

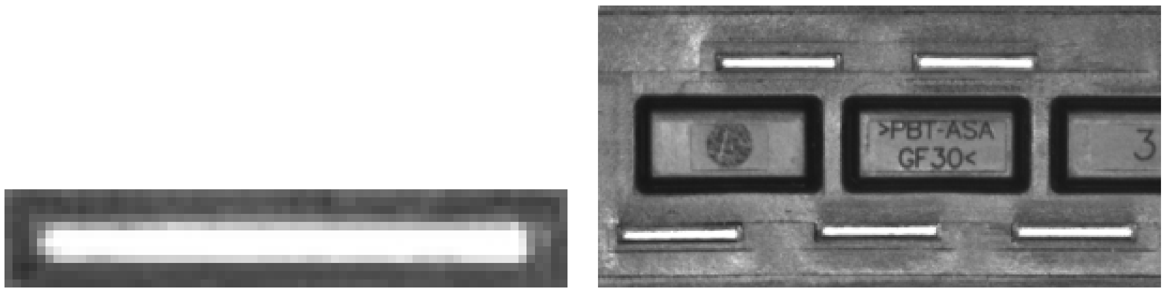
The target of the object detection system is the sample in which the rover should insert soil, rocks or gas. Since the intellectual property does not allow to use a real one copy, in this research project a slightly different target was designed . The target was 3D printed for running the same task of the sample and it has about the same size and shape of the real model . It was designed with SolidWorks, a 3D modeling software and 3D printed with the help of Ultimaker CURA software where it is possible to import CAD models and translate it into 3D printer models . The target has a cylindrical shape and encumbrance of 14,5 cm x 4,5 cm .

2.3 Possible solutions

In order to obtain a good compromise between computational costs, time required and space application constraint, different possible approaches can be taken into account. In this research project it has been decided to perform a "deep-learning" approach with the aim to demonstrate its applicability to the space sector as it is an hot point of scientific research, nowadays. For the sake of completeness other approaches will be shown in the next subsections.

2.3.1 Template matching approach

The *template matching* is a basic approach and in general "it involves defining a measure or a cost to find the “similarity” between the (known) reference patterns and the (unknown) test pattern by performing the matching operation" [4] . It has a variety of application fields from speech to data analysis, but in computer vision area it consists of matching a "template image" against the image that has to be analyzed looking for the "template content"(figure 2.2). Usually the matching search is realized by shifting the template image, taking notes of the correlation between the image area analyzed and the template image for each position. The shifting path could be of 1 pixel or more, depending on the peculiar template matching application. At the end of the process a final thresholding filter is applied on the correlation values results and the best ones are indicated as possible matches . This approach is not robust due to its extreme simplicity and shape variation can significantly compromise the matching result. It is used mainly in the industrial field where the standardization of the realized objects can grantee similarity between template image and analyzed image. Possible results improvement can be obtained involving not only a single templates but a set of different templates of the same object with different position and light condition .
In summary , this approach it is easy to be implemented due to its low computational cost also on space qualified computer, but it’s extremely unreliable for the purpose of the thesis since a certain level of unknown environment information is considered.



(a) Template image. [5] (b) Input image. [5]

Figure 2.2

2.3.2 3D model approach

A more complex approach is based on 3D model matching approach: trough a stereo camera , a dense point cloud is obtained, where a 3D object has to be searched comparing the known 3D model with point cloud regions. This constitutes the evolution of the previous simpler approach and it is more reliable of it, but it presents some critical issues: the heuristic function, that looks for similarities, works pretty well with a non-occluded target, but shows drastic drop in performances when "non expected" conditions are present. Moreover the long distance application represents another weak point for this approach. As in the previous technique , it is used in the industrial context and robotic field .

2.3.3 Other approaches

There are many other approaches implementable for an object detection application, however, most of them, including the two previous methods, needs a sort of “spaghetti approach” to be suitable. Spaghetti approach is the framework of many images processing detection problems in which multiple local functions have to be found and an heuristic composition of local algorithms has to be implemented in order to analyse each possible scenarios captured by the camera.

2.3.4 Deep learning approach

This approach will be explained in details in the following chapters. It is worth noticing that if previous solutions suffer of “spaghetti code” problem, the deep learning approach incorporates flexibility in its solutions.Indeed, it does not need different algorithms for every cases, but only one algorithm capable of abstracting the “concept” exactly like a human brain manages to distinguish the same object in different scenarios or to define the general category of two totally different objects (i.e. a horse and a dog look very different, but they could be inserted in the same category of "animals").

Because of this human-like capacity, its stunning abstracting skill and a growing interest, the deep learning approach has been chosen as the best performing technique for this research project.

Chapter 3

Machine learning

In this chapter, a brief overview of Machine learning history will be discussed and some concept and technical details will be analyzed. As a starting point a clarification has to be highlighted : "Machine Learning" it is not equal to "Artificial Intelligence" even if the two terms are used as synonyms. In particular, Artificial Intelligence refers to " *"the science and engineering of making intelligent machines"* while Machine Learning is just one of the possible technique to reach that goal. Moreover in the field of machine learning, Deep Learning is another specializations of particular technique related to machine learning . Conceptually, Deep Learning is a field included in Machine Learning and the latter is included in artificial intelligence domain . In the figure 3.1 this concept can be visualised.

3.1 History of machine learning

The history of machine learning can be traced back to Aristotele's associationism , 300 B.C. . Investigating on our brain, Aristotele expressed ita theory called Associationism. *Associationism is a theory states that mind is a set of conceptual elements that are organized as associations between these elements* [7] . Aristotle examined the processes of remembrance and recall and brought up with four laws of association: contiguity, frequency, similarity, contrast. The philosopher thought that those laws are implemented in our brain through our sense, for example: the feel, the smell, or the taste of an apple should naturally lead to the concept of an apple, as common sense. Some of those method are actually very common in some machine learning techniques such as grouping elements w.r.t. their distance (close distance => same elements) or variables that occur frequently draw more attention from the model . In the figure 3.2 the timeline of main events that led to machine learning is reported reported.
The first approach to this field was to emulate how brain works: in this context Warren McCulloch and Walter Pitts invented the *thresholded logic unit (TLU)* , trying to reproduce the biological neuron structure. They thought that the right way to make intelligent machine was to replicate human brain in a "machine brain" .

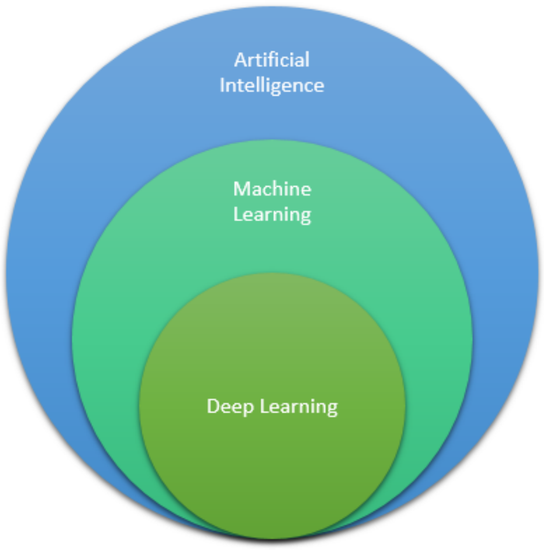


FIGURE 3.1: Domain relationship between Artificial Intelligence, Machine Learning and Deep Learning[6].

In 1950 Alan Turing proposed a renowned method aiming at answering to the question : *"Can machines think?"* [9].
In 1956 the term 'artificial intelligence' was born during the Dartmouth Workshop, which is widely considered as the founding event of artificial intelligence as a field. Next year, the psychologist Franck Rosenblatt presented the *perceptron* in his paper "The perceptron: a probabilistic model for information storage and organization in the brain" [10]. It was the ancestor of the current neural network. Perceptron can learn trough learning algorithms and it had the capability to detect numbers and letters. It was the first example of *supervised learning* thanks to its capability to learn from mistakes. Many others *intelligent programs* such as *chess game* were invented on the same path. Even if the interest was growing in this field, year-by-year, in 1969 something stop that progress: Marvin Minsky and Seymor Papert publicated an article *"Perceptron"* [11] in which they demonstrated that was impossible for perceptron to learn the *XOR* . The problem was that perceptron is a linear function modeler and could reproduce only linear functions, while XOR function, despite its simplicity, is a non-linear function. In the very next time the academic world lost interest in this field of machine learning leading to the so called *AI winter* .
In 1986 the researchers Geoffrey Hilton, Ronald Williams and David Rumelhart publicated a paper *"Learning representations by back-propagation errors"* in which neural networks were used with multiple hidden layers trough an innovative algorithm called *backpropagation*, which is still used today. Thanks to the hidden layers, the network had the capability to reproduce every types of functions. This new solution led to so many new applications as in 1988, when the researcher Yann Lecun gave birth to a new

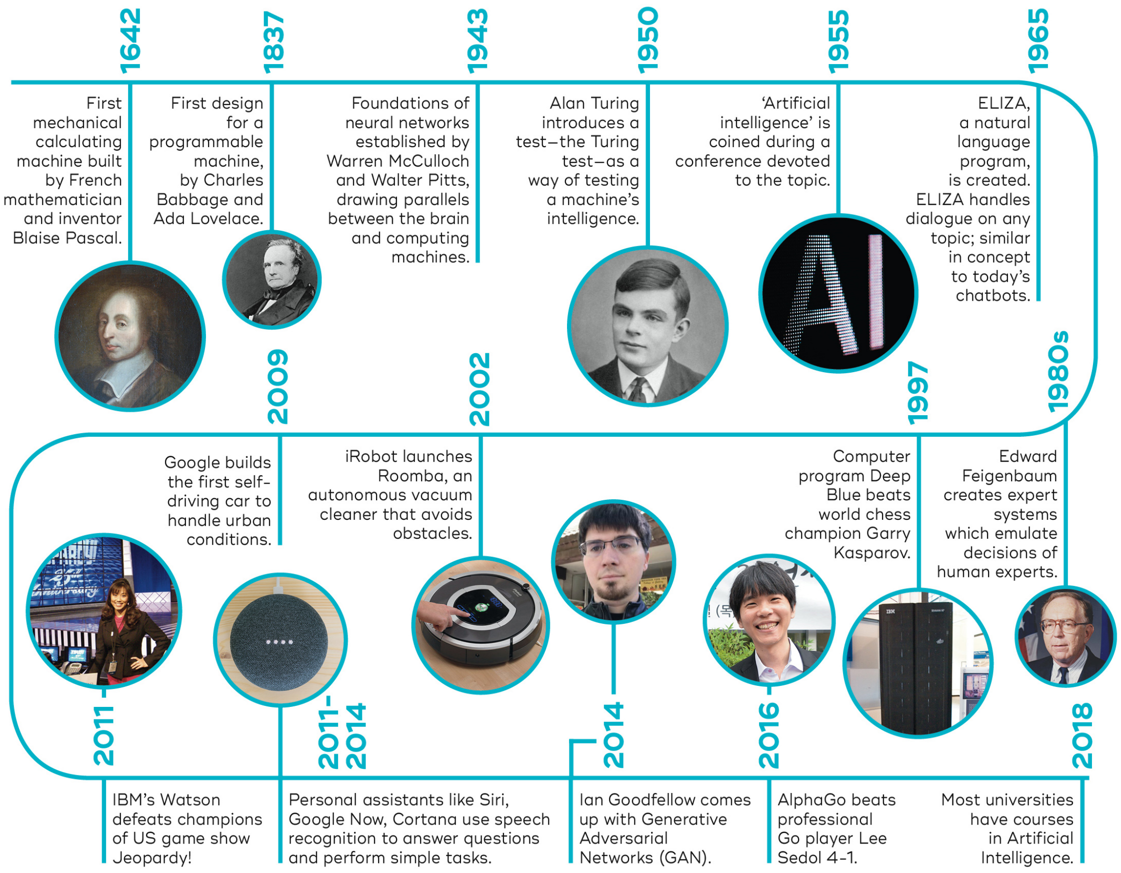

1642
First mechanical calculating machine built by French mathematician and inventor Blaise Pascal.
1837
First design for a programmable machine, by Charles Babbage and Ada Lovelace.
1943
Foundations of neural networks established by Warren McCulloch and Walter Pitts, drawing parallels between the brain and computing machines.
1950
Alan Turing introduces a test—the Turing test—as a way of testing a machine's intelligence.
1955
'Artificial intelligence' is coined during a conference devoted to the topic.
1965
ELIZA, a natural language program, is created. ELIZA handles dialogue on any topic; similar in concept to today's chatbots.
1980s
Edward Feigenbaum creates expert systems which emulate decisions of human experts.
1997
Computer program Deep Blue beats world chess champion Garry Kasparov.
2002
iRobot launches Roomba, an autonomous vacuum cleaner that avoids obstacles.
2009
Google builds the first self-driving car to handle urban conditions.
2011
IBM's Watson defeats champions of US game show Jeopardy!
2011-2014
Personal assistants like Siri, Google Now, Cortana use speech recognition to answer questions and perform simple tasks.
2014
Ian Goodfellow comes up with Generative Adversarial Networks (GAN).
2016
AlphaGo beats professional Go player Lee Sedol 4-1.
2018
Most universities have courses in Artificial Intelligence.

FIGURE 3.2: Time-line of progress in AI and ML[8].

type of neural network, the *convolutional neural network* , in the AT&T Bell Laboratories . This new technology was able to recognize handwritten words and digits. Due to the poor calculation power at the time, those new techniques weren't applicable to large problems, so researchers put aside neural network , looking for less costly computational algorithm and a lot of different machine learning algorithms gained popularity. In 2006, Professor Hinton gives another boost to the machine learning field, publicating its revolutionary paper *A Fast Learning Algorithm for Deep Belief Nets* [12] in which he presented new techniques for unsupervised learning algorithms. The "unsupervised" algorithms are those in which the machine learns by itself its object, there are no labeled data: those type of algorithms are used, for example, in the recognition of patterns in a group of data or for grouping the data. Thanks to Hinton's work, the *Deep Learning era* was born. Deep learning had great spread due to its impressive capacity to overcome old problems with accuracy and rapidity as tasks related to computer vision or speech recognition.Next following years, more and more researchers around the globe studied new implementation, when in 2009, Stanford professor Fei-Fei Lee launched *ImageNet*, an open database including milions of labeled images.
In 2012 Alex Krizhevsky won several international machine and deep learning competitions with his creation AlexNet, a new convolutional neural network , based on LeNet5

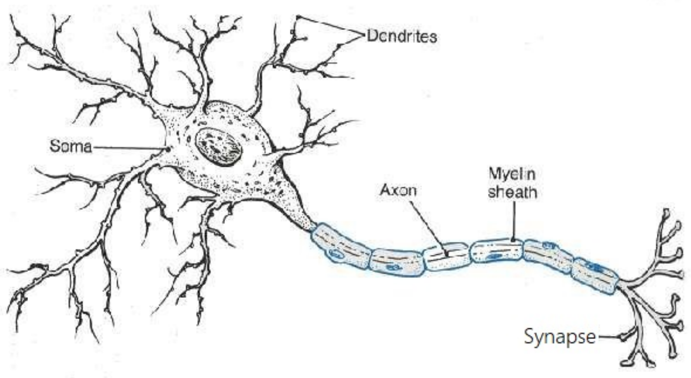


FIGURE 3.3: Neuron representation[13].

(by Y. LeCun years earlier) . The improvement proposed were the *dropout* mechanism to prevent the overfitting problem and the introduction of *rectified linear activation unit*(ReLU) related to the last output layer. Probably the last unexpected boost came from Ian Goodfellow, an authority in the field, with its *Generative Adversarial Networks*(GAN), composed of two branches of distinguished and competing networks that tries to learn faster and became smarter w.r.t the other branch. Y. Lecunn asserted : *This and the variations that are now being proposed, is the most interesting idea in the last 10 years in machine learning".* Nowadays almost every tech company tries to push on the accelerator and machine learning is a today standard , improving lives of human kind: "I think people need to understand that deep learning is making a lot of things, behind-the-scene, much better", Geoffry Hinton .

3.2 Key concepts in machine learning

In this section some of the basic knowledge and key concepts will be provided .

3.2.1 Threshold logic unit (TLU)

The TLU comes from the idea of McCulloch (neurophysiologist) and Walter Pitts (logician) who developed a *computational model of the "nerve net" in the brain.* In order to better understand how it works, it is worth highlighting how our brain neurons work :
A neuron is composed of 3 elements :

- Soma : also called *Central Body*, it contains the nucleus of the neuron and its main purpose is to process signals
- Axon : a projection that starts from a neuron and transmits signal to other neurons. At the end of this channel another type of extension is present : *synapses.* Through synapses the connection with other neurons takes places
- Dendrites : extension of soma, it receives signals from other neurons

The working principle is relatively simple : an electric signal (actually voltage generated by electric charged chemical substances) passes trough the axon and arrives at its ended area, in the synapses zone. Here the synapses release some chemical substances(neurotransmitters) able to attenuate or boost the electric signal depending on the amount of neurotransmitters released. The other neuron synapses are equipped with receptors able to catch the neurotransmitters. When this happens, local currents are generated for each synapses and added up in the soma. If the total current reaches a specific threshold a new current is generated in the new axon and the process can be repeated. McCulloch and Pitts replicated the biological structure and the process in an artificial structure, like an artificial neuron , the *TLU* :

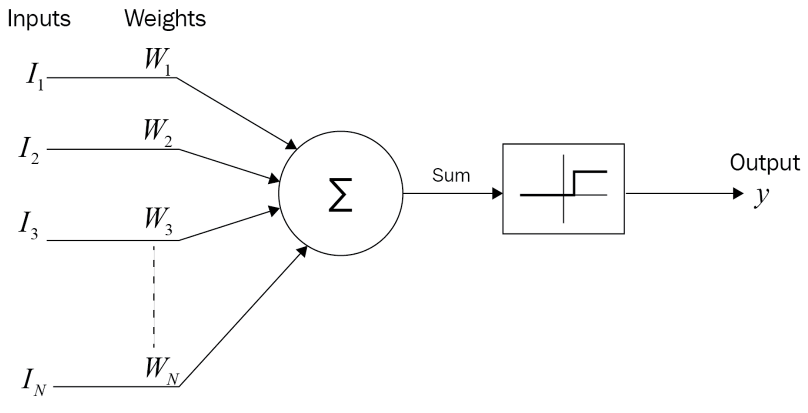


FIGURE 3.4: Thereshold Logic Unit representation[14].

there is a precise parallelism between the artificial structure and the biological one: the signal in the axon is replaced by a binary signal , 0 or 1 . The synapses conductivity is replaced by the weights of each input. The signal addition presented in the soma is replaced by an addition function in the TLU . The biological threshold is replaced by an *activation function* with a θ *thereshold* . Moreover,there's an inhibitory channel presented in each neuron : if the inhibitory signal is disabled the addition happens taking into account the weights and the result pass through the activation function; if the result exceeds the threshold, a new signal "1 signal" is generated and can reaches other neurons. If the inhibitory signal is active or the addition result doesn't exceeds the threshold, the signal will be *"0"*. The model behaviour in a formal mathematical form is :

$$y(n) = \begin{cases} 1 & : \sum_{j=1}^{n} w_j x_j \geq \theta \wedge \text{no inhibition} \\ 0 & : \text{otherwise} \end{cases}$$

3.2.2 Perceptron

The Rosenblatt's *percerptron* starts from the previous *TLU*, but it presents some feature :

- Weights and bias values are not identical

- Weights values can be positive and negative
- The inhibitory signal is no more present
- Perceptron has its learning rule

The activation function is similar to the TLU's one :

$$y = \begin{cases} 0 & : \sum w_j x_j \leq \theta \\ 1 & : \sum w_j x_j > \theta \end{cases}$$

If we consider w *(weights)* and x *(input)* as an array and b *(bias)* as

$$b = -\theta$$

the equation can be rewritten as :

$$y = \begin{cases} 0 & : w \cdot x + b \leq \theta \\ 1 & : w \cdot x + b > \theta \end{cases}$$

It's worth noticing that the new bias can have great influence on the output, indeed for high bias values the output will likely be equal to one. Moreover the new learning rule totally changes the way perceptron works: on the contrary of its predecessor, it's able to learn how to solve problem on its own, adjusting its parameters. If the desired output is defined as $y = y(x_j)$ and a is the actual one, it's easy to compute the *delta error*, as the difference between those two values. Once δ is obtained, a weights values correction can be performed in order to obtain $actual\,output \approx desired\,output$:

$$\delta = (y(x_j) - a)$$
$$\Delta w_j = \eta \cdot \delta \cdot a$$

In the *delta error* formula , η is the *learning rate.* Its value must be between $0 \div 1$ and it defines the rapidity of the weights value change .

3.2.3 Framework of Neural Networks

The common *single output* of the previous structure doesn't permit to learn non-linear function such as the *XOR*. Due to this lack a lot of researchers moved to other field and the so called *AI winter* came. In order to obtain a more complex functionality, the old structures were transformed adding more layers mutually interconnected, leading to the *neural network* as we know today. The new architecture had many weights and biases because of the multiple layers and needs obviously a more complex learning algorithm . In the figure it is possible to see how the common notation is used.
The leftmost layer is composed of units called *input neurons.* All the input neurons form a layer know as *input layer.* At the end of the structure there is a layer known

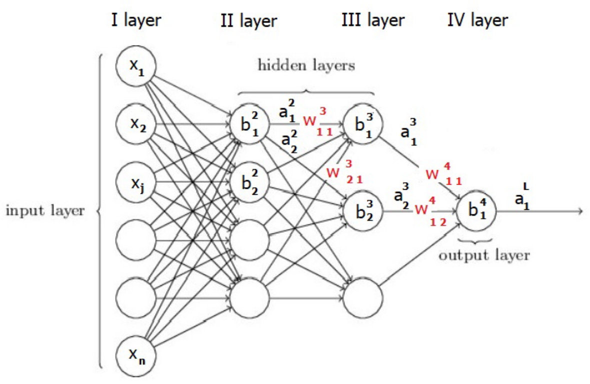


FIGURE 3.5: Neural Network architecture example

as *output layer* made by output neurons. Between the external layers there are the intermediate *hidden layers*, the truly innovation. The neural network can be defined by a set of 4 parameters:

- x_j is the j_{th} input in the input layer. Inputs are collected in a single *input vector* ;
- b_j^l is the j_{th} bias of the lth layer. Bias are collected in a *bias vector* b^l for each layer;
- a_j^l is the activation function of the j_{th} unit for the l_j layer. The activation function are collected in the a^l, the related vector for each layer;
- w_{jk}^l is the weight that links the k_{th} neuron in the $(l-1)_{th}$ layer to the j_{th} neuron in the l_{th} layer. Weights are collected in matrix form , w^l ,for each layer .

The differences between this new neural network and the perceptron can be highlight thanks to the activation function generated at each neuron given by:

$$a_j^l = \sigma\Big(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\Big) \longrightarrow a^l = \sigma\Big(w^l a^{l-1} + b^l\Big)$$

The computational capabilities grow exponentially the more hidden layers are present. . As example, with one hidden layer only, the neural net it's capable to pass from the impossibility of learning XOR function to the computation of a much more complex structure .

3.2.4 Gradient Descent

At this point, it's clear how a simple neural net works. More complex structures need different computational approach w.r.t. the basic ones, so they can learn and give valid results. Therefore it's useful to introduce a key concepts in neural networks:the *cost function* . This factor quantifies how the network is far from its goals. One of the first, elementary and today most used is *Mean Squared Error, MSE*:

$$C(w,b) = \frac{1}{2n}\sum_{x}^{2} \|(x) - a\|^2$$

where $y(x)$ is the desired output and a is the actual one. An hypothetical precise training algorithm should adjust values such that $C(w,b) \approx 0$ and the net should cyclically change the variables in order to minimize cost function. With the purpose to define this algorithm, it is convenient to consider a generic problem in which the cost function input is a n-dimensional array v. It is nonfunctional and in some cases not even feasible to use minimizing calculus the $C(v)$ function. Another technique has to be persecuted due to the huge number of variables given by v. Taking into consideration small variations for each components v_j of the vector, C function varies as:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 + \frac{\partial C}{\partial v_3}\Delta v_3 + \cdots + \frac{\partial C}{\partial v_n}\Delta v_n$$

All distinct Δv can be included in an array $\Delta v = (\Delta v_1, \Delta v_2)^T$ and all derivatives in $\Delta C = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$ establishing the gradient of C :

$$\Delta C \approx \nabla C \cdot \Delta v$$

Thanks to this equation it's possible to choose Δv so as to make ΔC negative. Therefore, highlighting that ∇C contains C varietions for one component v_j , is opportune to design Δv as:

$$\Delta v = v' - v = -\eta \nabla C$$

where η is the *learning rate* . Substituting this last equation in the previous one, the ovious result is obtained:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta ||\nabla C||^2$$

The last equation can be considered as an update rule. It determines what is called *gradient descent algorithm.* This remarkable approach cyclically updates v aiming to minimize the cost function C, so that :

$$v \rightarrow v' = v - \eta \nabla C$$

This algorithm form is broadly used in machine learning and needs only few adaptations. The main issue related to this computation is connected to the learning rate parameter setting (figure 3.6): with an high value it's possible to deal with a huge number of variables and speed up the computation calculus without converging to the final solution; to the contrary, setting a small value can produce an unproductive algorithm because of the unreasonable training process time cost.
The previous equations are for a generic case,but substituting bias and weights in to the generic variables v, it leads to the algorithm actually used for a neural network train process.The equation related to C has w_j and b_j as components, such that :

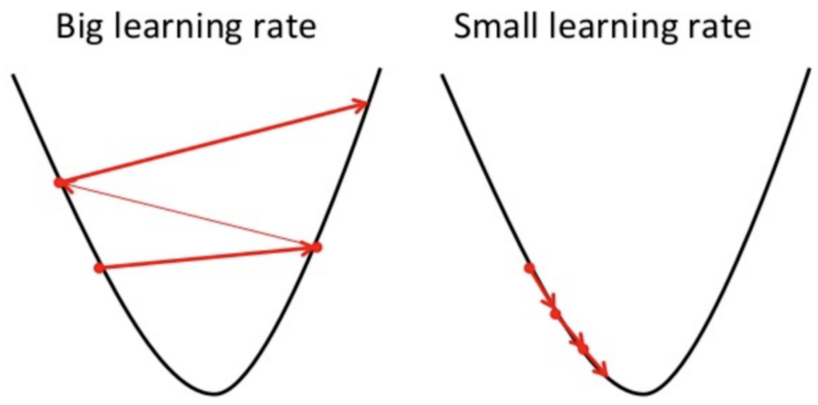


FIGURE 3.6: Representation of different processes according to different learning rate [15].

$$w_j \rightarrow w'_j = w_j - \eta \frac{\partial C}{\partial w_j}$$

$$b_j \rightarrow b'_j = b_j - \eta \frac{\partial C}{\partial b_j}$$

The cost function ($C(w,b) = \frac{1}{2n}\sum_x \|y(x) - a\|^2$), can be re-written in a more compact form as $C = \frac{1}{n}\sum_x C_x$, that is barely an average over the elements $C_x = \frac{||y(x)-a||^2}{2}$ where x is an input array of the input layer. Therefore, aiming to obtain ∇C , the gradients ΔC_x have to be singly computed for each input x and then averaged over all the inputs, so that $\Delta C = \frac{1}{n}\sum_x \Delta C_x$.

3.2.5 Stochastic Gradient Descent

It is a possible approach aiming to use the gradient descent to a large training input number. The *stochastic gradient descent* substantially reduce the learning process time; in order to compute ∇C, only a small number of ∇C_x are included. Rather than facing all the n training inputs, a subset m of them are included.

$$\nabla C = \frac{1}{n}\sum_x \nabla C_x \approx \frac{1}{m}\sum_j \nabla C_j$$

The parameter n is the number of all training inputs and m is the selected mini-batch of input vectors. The equation can be re-written with weight and bias updated :

$$w_j \rightarrow w'_j = w_j - \frac{\eta}{m}\frac{\partial C}{\partial w_j}$$

$$b_j \rightarrow b'_j = b_j - \frac{\eta}{m}\frac{\partial C}{\partial b_j}$$

Now the sum is only on the inputs of the considered mini-batch. Once weights and biases are updated another random mini-batch is selected. The gradient stochastic algorithm cyclically takes a new random mini-batch from training data until all inputs have been chosen at least once. At this point a *training epoch* is ended and the algorithm launch the loop again with the next epoch. This algorithm manages to deal with mini-batches of different dimensions, even a unitary mini batch. This techniqu is called *online* or *incremental learning* and corresponds to the human brains biological algorithm.

3.2.6 Back-propagation algorithm

At this point, the central problem in order to apply gradient descent and make the network learn is the estimation of $\frac{\partial C}{\partial w^l_{jk}}$ and $\frac{\partial C}{\partial b^l_j}$. This algorithm is know as *backpropagation* since, starting from output layer of a network and moving back, it leads to compute the two partial derivatives of the cost function and then compute easily weights and biases values. Essentially, it is a method that grant to use gradient descent and all its versions in every specific condition.Using this algorithm a new quantity is designed , known as output error δ^l_j . According to this notation, the output error involves the j^{th} neuron of the l^{th} layer. The following equation shows the new quantity. It is partial derivative of a cost function with respect to the potential activation function. Once more, z involves the j^{th}neuron of the l^{th} layer.

$$\delta^l_j = \frac{\partial C}{\partial z^l_j}$$

It is possible to include all the components in a single vector δ^l. So, back-propagation uses this quantity δ^l_j to compute $\frac{\partial C}{\partial w^l_{jk}}$ along with $\frac{\partial C}{\partial b^l_j}$. Thus, with some mathematical passages, the four main equations of back-propagation can be extracted :

$$\begin{aligned}
\delta^L &= \nabla_a C \odot \sigma'\left(z^L\right) \\
\delta^l &= \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right) \\
\frac{\partial C}{\partial b^l_j} &= \delta^l_j \\
\frac{\partial C}{\partial w^l_{jk}} &= a^{l-1}_k \delta^l_j
\end{aligned}$$

The last equations express the aformentioned cost function. Paying attention to the four equations of the backpropagation algorithm, it is possible to notice how it works. First of all, error values are computed for the output layer and back propagated until the input layer thanks to the second equation. The last two equations relate these previously computed errors with the partial derivatives, fundamental for gradient descent. Now, it is easy to use the results for the two learning process update equations

. In conclusion, backpropagation is crucial for gradient descent or stochastic gradient descent algorithms because it makes computations feasible. For example, utilizing stochastic gradient descent and backpropagation, selecting a mini-batch of m training inputs, the following steps must be performed:

1. A group of m samples is taken from the n available of the dataset;
2. For each x of the m available:
   - x is assigned to the input layer;
   - *Feedforward*: following all layers $l = 2, 3, ..., L$, potentials $z^{x,l} = w^l a^{x,l-1} + b^l$ .Moreover rhe activation functions are evaluated $a^{x,l} = \sigma(z|^{x,l})$;
   - *Output errors*: $\delta^{x,L}$ is calculated as $\delta^{x,L} = \nabla a C_x \odot \delta'(z^{x,L})$;
   - *Backpropagation*: for each layers $l = L-1, L-2, ..., 2$, $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$ are calculated
3. For each layer $l = L, L-1, ..., 2$ the algorithm updates all variables ,biasesand weights .
$$w^l \to w^l - \frac{\eta}{m} \sum_x \delta^{x,l} \left(a^{x,l-1}\right)^T$$
$$b^l \to b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$$
4. A random mini-batch of input vectors is chosen from the $n-m$ available and the loop is starts again until the epoch is over.

Thanks to this algorithm and involving adequate computational power, the training of a general multi-layer network is possible. Regrettably, , this is the most essential algorithm and different issues may occur, make heavier the network training phase.

3.3 Neural network

Before go into following concepts, it is opportune to make preliminary notes. The neural networks form precedently are known as *feed-forward neural networks*, due to the signal direction . They are certeinly the most known nets, other types of neural network were created such as long short-term memory units, recurrent neural networks, deep belief nets, Hopfield networks and others. Another point is the training method in general can be distinguished and divided into three main groups : *supervised learning*, *unsupervised learning*,*reinforcement learning.* The peculiar characteristic of a supervised learning method is that inputs and outputs are known and the net try to understand the correlation between the proposed elements, while in an unsupervised learning training only the inputs are provided and the neural networks tries to understand the unknown correlations(e.g. a clustering techniques, pattern recognition).
In the next sections different problems regarding neural networks training and related current solutions will discussed .

3.4 Neural network problems

3.4.1 Neuron saturation

With the purpose to better tackle the problem we will discuss about a simplified single neural network . As highlight in the previous sections, partial derivatives of a chosen cost function $\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}$ involves how neurons learn. Working with the basic quadratic cost function, readjusted for a single neuron $C = \frac{(y-a)^2}{2}$, the subsequent results will be:

$$\frac{\partial C}{\partial w} = (a-y)\dot{\sigma}(z)x = a\dot{\sigma}(z)$$
$$\frac{\partial C}{\partial b} = (a-y)\dot{\sigma}(z)x = a\dot{\sigma}(z)$$

In this basic situation it is worth to notice that both bias and weights are driven by the activation function derivative. Thus , substantially, when $\sigma(z)$ is ≈ 1 or ≈ 0, its derivative $\dot{\sigma}(z)$ has small values. This produces a learning slowdown for the neural network that may preclude any improvement. The problem can be tackled with different techniques.

Learning Slowdown in the Final Layer

Aiming to fix the problem in the output layer, researchers studied different type of cost functions .

- *Cross-entropy cost function.* it is widely the the most common cost function. Considering multiple outputs, the expression is :

$$C = -\frac{1}{n}\sum_x \sum_y \left[y_j ln\left(a_j^L\right) + (1-y_j)\, ln\left(1-a_j^L\right)\right]$$

  Once more, facing a multi-inputs single output neuron unit, the gradient is in the the form:

$$\frac{\partial C}{\partial w} = \frac{1}{n}\sum_x x_j(\sigma(z)-y)$$

  The previous equation indicates that weights updating is proportional to $(\sigma(z) - y)$ and the activation function derivative is not involved in it.

- *Log-likelihood cost function.* It is largely used with a specific activation function: the softmax unit. Its linked cost function, called log-likelihood cost function is expressed by the following equation:

$$C = -ln(a_x^L)$$

  where x is the selected training input and L points to the output a of the final layer. If the network has a great prediction confidence , it will estimates an output value close to one, and the related cost function will assume a small value. Once more, it is possible to notice that for the the cost function gradients the partial derivatives don't influence it and so the learning slowdown is prevented:

$$\frac{\partial C}{\partial b_j^L} = a_j^L - y_j$$
$$\frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1}\left(a_j^L - y_j\right)$$

Weights and Biases Initialization

Due to the previous technique there is an considerable chance to saturate some neurons in the hidden layers because the cost function modification only affects the output layer. A possible solution is to initialize all variables with a Gaussian probability distribution setting mean=0 and standard deviation equal to $\frac{1}{\sqrt{n_{in}}}$ where n_{in} is the input weights numbers of the neuron. Through this technique, learning slowdown problem is consistently reduced and all neuron units has less chance to saturate. With regards to biases initialization, two main methods are considered: a basic approach is setting all biases equal to zero at the beginning,while the other one is to use the same previous approach. That's because biases has no large influence in the slowdown issues and related adjustments are not required.

3.4.2 Data overfitting

Another relevant issue is caused by *data overfitting.* Due to the large variables number to train it is possible to face an overfit data and the networks risk to learn peculiar characteristics of the training set more than abstract general concepts. A network trained in this is way is probably useless because it will respond positively only to datasets with the same peculiar characteristic . Different methods can be involved to overcome this crucial issue.

Dividing Data

This is a widely used approach : it requires to split all the available data in three different subgroups: the already known *training data* for the learning phase of the network, *validation data* aimed at testing the network after each epoch and a *test data* for checking final performances of the net after the training session. During the training process, if the accuracy of the validation dataset stops increasing while the training dataset accuracy keep increasing , something is not properly working and it is a signal of a possible overfitting issue. A common strategy known as *early stopping* is to stop the training phase if the validation data accuracy reaches a characteristic *plateau* trend. Typically, supervising these three class of data during training process manages to overcome overfitting issues and set more suitable *hyper-parameters* .

Artificially Expanding the Training Data

Nowadays neural networks presents an enormous numbers of parameters and so, it is a common problem to overfit training data with no generalization. The most simple way to avoid this is to increase the amount of training example ,but this approach is not always possible because of the limited data. To overcome this problem, researchers developed an technique known as *artificial expanding of the training data* that manages an enhancement of the dataset with no extra training samples. The key concept is to upgrade the available dataset , modifying for example ,images with reshape or hue shifting,reflection or cutting, thus operations that in some way occur in real world variations.

Regularization Techniques

Generally ,the previous approaches are not sufficient to prevent and overcome the overfitting problem. There is a set of methodologies, called *regularization techniques*, able to reduce issues even with fixed dataset. In this section two of the most used are presented, even if many researchers have developed valid alternatives. The first one is know as *L1 regularization.*In practice it needs an adjustment of the desired cost function. Working with the cross entropy equation presented above, the L1 regularization adds a parameter called *regularization* term:

$$C = -\frac{1}{n}\sum_x \sum_y \left[y_j \ln\left(a_j^L\right) + (1-y_j)\ln\left(1-a_j^L\right)\right] + \frac{\lambda}{n}\sum_w |w|$$

where in the second sum w represents the number of weights and λ is a hyper-parameter that has to be chosen. It is desirable to simplify this last equation as:

$$C = C_o + \frac{\lambda}{n}\sum_w |w| \rightarrow \begin{cases} \text{if } \lambda \text{ is small the regularization term can be omitted} \\ \text{if } \lambda \text{ is large the analyzed model learns small weights} \end{cases}$$

Obviously, the proposed approach lead the model to learn small weights value while they are allowed to have high values only if they remarkbly reduce the value of chosen cost function. Observing the results, the regularization L1 keeps the most weights values near zero and concentrates non-zero variables into a bounded region of important connections.

The second technique, very popular, is called *L2 regularization.* Similar to the previous one, it imposes an upgrade of the chosen cost function with another extra term:

$$C = C_o + \frac{\lambda}{2n}\sum_w w^2$$

Computing all the calculus it is worth to notice that gradients has the following form:

$$\frac{\partial C}{\partial w}* = \frac{\partial C_o}{\partial w} + \frac{\lambda}{n}$$
$$\frac{\partial C}{\partial b} = \frac{\partial C_o}{\partial b_o}$$

and so gradient descent algorithms is defined as

$$w \to w - \eta\frac{\partial C_o}{\partial w} - \frac{\eta\lambda}{n}w = \left(1 - \frac{\eta\lambda}{n}\right)w - \eta\frac{\partial C_o}{\partial w}$$

Dropout

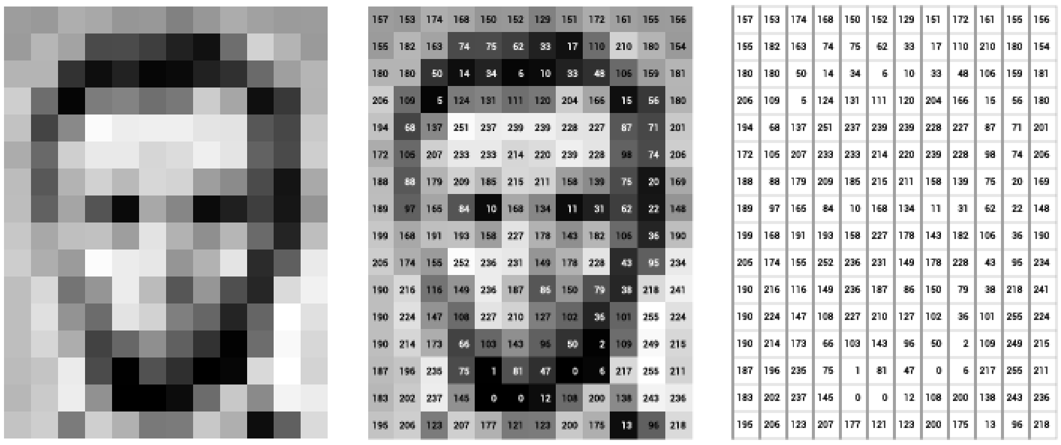
It is a totally different regularization technique. Opposite to L1 and L2 regularization, this method requires no changes to be made to the chosen cost function. During e training step a percentage of neurons are de-activated (generally 50%). The rest of the them is bypassed and weights and biases linked to them are not updated. At following training step the procedure is repeated involving another set of neurons randomly picked. The authors of this quite recent techniques say that *"[...]this technique reduces complex co-adaptation of neurons, since a neuron cannot rely on the presence of particular other neurons"*. Therefore, thanks to aformentioned approach each neuron is forced to learn more suitable features in order to abstract data.

3.5 Deep learning

In this section the inner part of the artificial intelligence world , the *deep learning* , will be analysed . It is a machine learning method base on neural networks , started to spread since 2006 . The basic idea behind this method is to split a huge problem in simpler sub-problems,more or less as human brain has works , but with a different level of complexity . At the end, the single solutions for each sub-problems are composed togheter as part of the final solution . In this framework the neural networks are no more composed of a series of fully connected layers that have to accomplished the same task, but each layer or part of layer has some specific task , some specific problem to tackle. This type of nets are called *deep neural networks.* The following sections will focus on object detection elements in deep learning, but this elements are actually common for a large range of deep learning applications like natural language processing or self-driving system .

3.5.1 Convolutional neural networks

In all the neural networks shown so far , fully connected layers were presented. That type of network allowed researchers to obtain great results in so many fields and today it is one of the most popular network configuration. Despite its powerful potential,



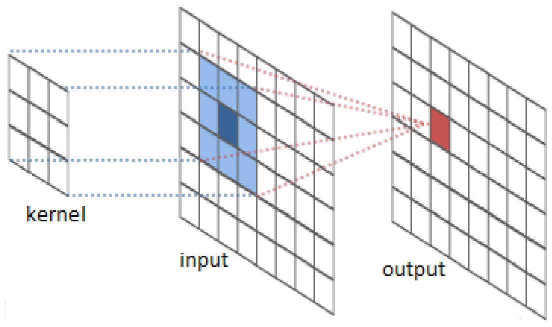
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

FIGURE 3.7: Image presented as a matrix of pixel values[16].

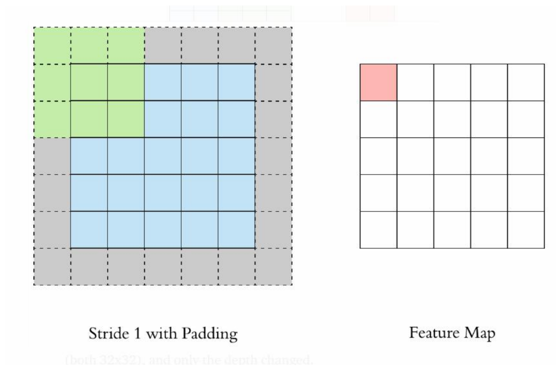
those neural networks suffer of lack a spatial structure information. If they get an image as input, they just cannot recognise the existing relationship between pixel in the space . Due to this reason, another type of net became popular , the *convolutional neural network*. Thanks to its configuration, it has the ability to distinguish spatial information and it is mainly used in the object detection field , but it has a variety of application. Aiming to better understand a convolutional neural network, some of related elements and principles will be analysed .
Previously specified, the common neural network lacked in the analyses of the spatial relationship between pixels so a new operation was introduced , the *convolution*.

3.5.2 Key elements

The convolution is the key element of a convolutional neural network. It is a mathematical operation that in general involves two function as input and produces a third function as output. Intuitively it "blends" togheter the information of those two function, returning the overlapping area of the shifted function . In this particular case convolution involves two matrices : it performs a matrices multiplication and a final sum. In computer vision applications, an image is analysed as matrix of pixel.The most simple matrix to analyse is a black and white matrix in which each pixel contains a number that represents how "bright" is that point, while a coloured image is identified with a NxNx3 matrix in which each of the three layer is related to a standard color (red, green and blue, the rgb common triad is just a possible color triad) . In the figure .a it is possible to notice how a computer *see* an image . The values into the matrix are the aformentioned "bright values" . When an image in used as input of a convolutional neural network, each neuron of the hidden layer is connected only to a certain portion on the image. This specific window of the image involved in the process is called *receptive field*. The other matrix involved in the convolution is the so called *kernel* or *filter*, a window containing some weights that will be used in the convolution. Each neuron of the hidden layer is related to the kernel and has a bias value. The required parameters are the weights plus one bias so, for a NxN kernel matrix, a



(a) Monochrome picture processed by a 3x3 kernel . In blue the receptive field[17].



(b) Padding with stride length equals to one and related feature map[18] .

FIGURE 3.8

total amount of NxN+1 parameters are required . In principle, for K neurons, a total number of (NxN+1)*K parameters are required , but usually a shared weights and bias approach is used . Infact the kernel is an element that tries to extract some important information from the image like borders or peculiar pixel portion , so intuitively it is more efficient not to change a kernel but use the same one for each portion in order to extract information . The convolution equation is reported in the following form :

$$\sigma = \left(\sum_{m=0}^{K} \sum_{n=0}^{K} w_{m,n} o_{i+m,j+n} + b \right)$$

where the filter has KxK size and performs the operation previously explained .
On the 3.5.2.b there is the representaion on a 2D convolution process for a monochrome picture , but colored pictures usually have a depth dimension so the kernel has to adapt to the dimesions of the input . For a common rgb picture of NxNx3 size the kernel should have KxKx3 size . The matrix multiplication is performed in 3D space , but the result is again a single value . Moreover in convolutional neural network , different filters are applied to an input in order to extract different information from the same picture . The result of all this processes will be L layers related to L filters .

3.5.3 Stride and padding

Once the kernel values are chosen, it slides across the entire image with a certain step between each position. The step values in pixel is called *stride length.* The more the stride length is reduced, the more convolutional processes are performed, with a dense extraction point . Commonly the stride and the kernel size are related to the matrix size where they are applied to. In order to have a suitable amount of results to deal with, a larger input picture requires a larger stride and kernel size . Aligning the center of the kernel windows with the center of the receptive field during the sliding,it can happen that kernel size is applied outside the picture borders . A common approach to overcome the issue is to introduce the *padding* . namely a region of zero value pixel

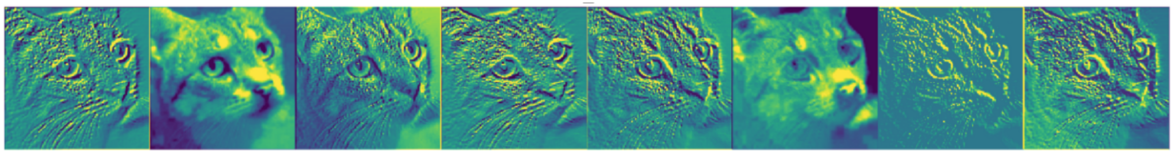


FIGURE 3.9: First block of eight feature maps of a cat classification[19].

or the nearest border value in order to match the kernel size and obtain as output a matrix with the same size of the input one .

3.5.4 Feature map

The final result of the convolution operation is the *feature map*. It contains the final values of each convolution process in matrix form . An other omitted operation that usually is added at the end of each operation is the application of an activation function . The most common activation function is the *relu* presented in the previous section . The feature map represents the important information that the filter has been able to extract . For each kernel there will be a related feature map and each feature map contains a specific feature that was supposed to be important in order to train the CNN with its specific dataset . As it is possible to notice in eight kernels was involved in convolution process in order to extract some information from a cat picture . In general, the first layer blocks is aimed to detect edges, indeed in the eight pictures the "activated region" are those that represents edges of the cat . Those first layers are supposed to detect low-level of abstract information while deep layers are related to an high level of abstraction .

3.5.5 Pooling layer and max pooling

The last performed operation in a common CNN is the *pooling* operation . It is aimed to reduce the "quantity" of the information we are dealing with and working only on the most decisive part of it . Thanks to this method , the numbers of parameters tend to decrease as the deeper layers are reached, so there is a sort of simplification of the net . Moreover it helps to overcome an overfitting issue in the training phase : reducing the parameters allows the network to maintain an "abstract approach" in the detection of the information . There are different pooling operation that can be involved, but a very common one is the *max pooling* . It is applied to the feature map, downsampling the matrix and obtaining a pooling layer . It usually consists in a window of 2x2 size, stride equals to two and no padding. It slides on the feature layer and for each step it takes the largest number presented in that inspected portion and write it in the following pooling matrix. At the end of the operation a pooling layer will be composed with all the largest number presented in the previous feature matrix : a large number is related with a crucial element presented in the matrix so, for example, clearly visible edge will be detect as a region of large numbers related to its. Taking the largest one allows to get that information without carrying on the entire interested region .

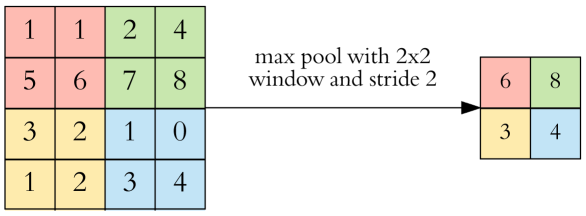


FIGURE 3.10: Max pooling operation on a feature map[18].

3.5.6 Convolutional neural network architecture

Aiming to create a CNN, a multitude of different architectures can be developed. It depends on the type of problem, the complexity and specific constraint . Nevertheless the basic elements are always the same and so the basic architecture . It is always based on convolution layer - pooling layer - fully connected layer . In general the first part of the layer is aimed to "extract" so convolutional and pooling layers are involved . The last part, the f.c. layer, is aimed to infer something , based on the information extracted from the preceding layers . Usually the last part is composed of two f.c. layers : the first one takes as input the results of the pooling layer, resizing it in a monodimensional vector and at this stage is processed by the layer.This layer is equiped with a standard relu activation function . The second layer is equiped with a softmax activation function that guarantees a probabilistic info as result . As general example. in the a VGG16 network is presented . It was a simple architecture that in 2014 won the ImageNet competition on object detection and classification . Although it has approximately 140 milions of parameters , during that competition was one with less parameters . Indeed , instead of focuses on parameters numbers , its advantage was those peculiar architecture with a series of convolution and max pooling layers . It is worth to notice that it is considered one of the excellent architecture for that time although the key elements are the three elements previously analysed .

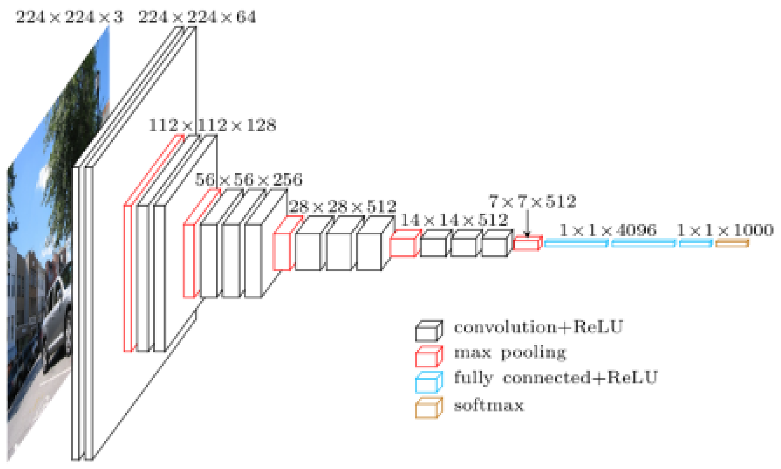


FIGURE 3.11: Architecture of VGG16 network[20].

Chapter 4

Object detection

4.1 Introduction of object detection

Object detection is a crucial research area involved in the computer vision field .This technique was born in the late 1960s,beside the important discoveries in artificial intelligence . In the 1970s the creation of new algorithms gave a great impulse in the sector. Those algorithms were related to specific tasks like edge detection or line labelling and they became the basic theory on which the rest of technique were built .
Nowadays , object detection is indissolubly related to deep neural network technologies and permits computer to perform an hard task with simple solution . In order to deepen the object detection sector it is worth to highlight some differences between common tasks that could seems very similar . Starting from its definition in a.i. , *classification* is a machine learning task in which an algorithm categorize a picture on the basis of certain specific classes introduced by the "user" , so for a picture only one class can be chosen. Instead , *object detection* is the ability to detect one or more different "targets" in the picture, defining relative position and *bounding box*. The bounding box is a common term that describes the box that the algorithm draws to localize the detected object in the picture . A third common application is the *instance segmentation* : it is a sort of upgrade of the object detection in which each pixel of the image is linked with a class (or to "background" if it doesn't belong to a specific class) .

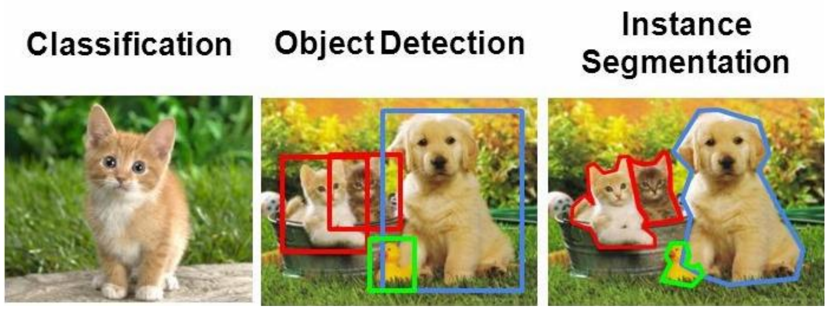


FIGURE 4.1: Difference between Classification,Object Detection and Instance segmentation[21].

4.1.1 Object Detection framework

In the object detection application, a common scheme is present. It can be divided into three main steps , involving the *region of interest.* It is a concept shared with many other applications, but in general it is the identification of a sub-set of data in a larger set. In this case it is the identification of some region in the picture with some rule .The three main steps of an object detection are :

1. Generation of RoIs , specific for the involved algorithm.The entire picture is divided in particular regions;
2. Each RoI is analysed independently. Visual feature are extracted in order to detect the presence of a pre-classified object inside each RoI. If the object is detected, the algorithm draws a bounding box to confine it.
3. The algorithm tries to merges the overlapping boxes with the same object, aiming to define a unique box containing the entire object

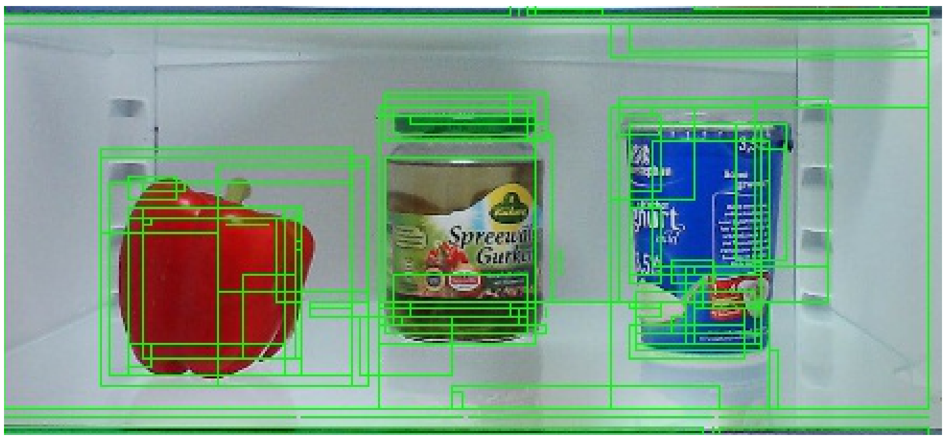


FIGURE 4.2: Multiple ROIs generation[22].

4.2 Evolution of object detection

In this section will be reported some of the most famous object detection methods. Even if they seems to be quite different, the aformentioned general scheme continues to hold true .

4.2.1 Haar Feature-based Cascade classifiers

It is a famous object detection algorithm use to detect object in image or video , developed by Paul Viola and Michael Jones. They explained this method in the paper *Rapid Object Detection using a Boosted Cascade of Simple Features* in 2001. The algorithm is based on a cascade function previously trained with false and positive

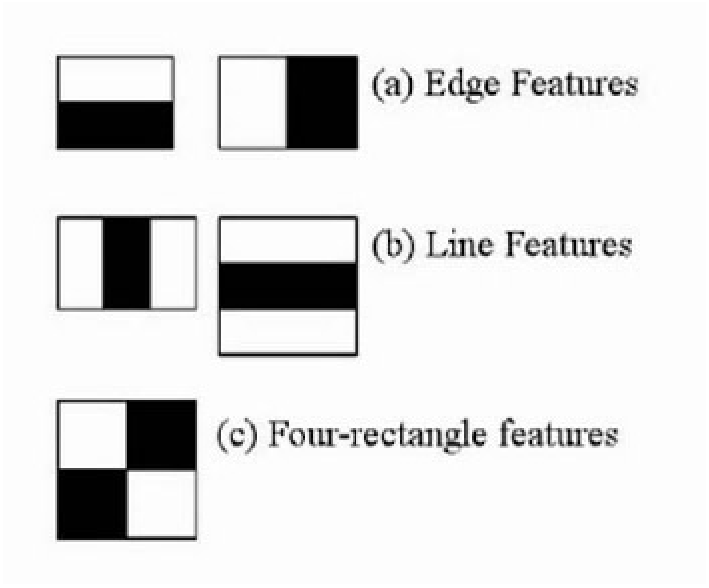


FIGURE 4.3: Example of Haar Feature[23].

images . In the end that function is used on new pictures. It is well know that this approach works pretty well in case of face detection. This algorithm is based on the Haar features , a set of features used to analyse the images. Some of them are reported in . The set of Haar feature involved has more than 160000 . They are used in a computation in order to select the best feature that permits to identify some elements in the face . Even if they are a huge number , only a subset of them are useful for the purpose . The great discoveries of this algorithm is that it uses a series of "weak classifier" trained with the positive and negative images by the haar feature computations . Each of them, called "stage", is a weak classifier because their accuracy is slightly better than a random guessing, but using a cascade of them it is possible to obtain powerful results . A specific cooperative algorithm is involved in the selection of the correct Haar feature and it build a "strong" classifier as a linear combination of weighted "weak" classifier. Following this framework, an images can be considered as "positive" only if each stage of the classifier has considered "positive" the region it was looking in .

4.2.2 Histogram of Oriented Gradients (HOG)

This is is a very powerful method founded in 2005 by two french researchers [24] while they was working on pedestrian detection . In order to better deepen how it works, the concept of *image gradient vector* has to be explained first.

Image Gradient Vector

It is a computer vision technique used in many applications. The main idea behind this is to find the direction of colors changing among image pixels. This information is identified as a decisive element that can describe the image . In this case the gradient is discrete due to pixels finite numbers. The vector is computed as color difference between adjacent pixels in vertical and horizontal direction. The formula for a pixel in (x,y) position is :

$$\nabla f(x,y) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} f(x, y+1) - f(x, y-1) \\ f(x+1, y) - f(x-1, y) \end{bmatrix}$$

The gradient is defined by two values :

- *Magnitude* namely the L2-norm of the vector $g = \sqrt{g_x^2 + g_y^2}$
- *Direction* computed as $\theta = arctan\left(\frac{g_y}{g_x}\right)$

Usually , this operation is boosted implying a convolution operation with certain kernels . It is performed trough two direction kernels, like those in . One of the most known operator used today is the *Sobel operator*.

-1	0	1

-1
0
1

FIGURE 4.4: Kernels used to perform gradient on matrix.

Once explained the image gradient vector concept, a further path was done thanks to the HOG algorithm that is based on the aformetioned technique. In this approach the gradient is able to removed non-essential info from the images info the . It can be split into 5 steps :

1. Pre-processing : the image is reduce to a predefined size
2. Magnitude and directions are computed for the entire images thanks to the gradient.
3. The image is divided into 8x8 cells. For each each cell an histogram based on 9 direction value from 0 to 180 is created. Based on direction value , each related magnitude value is inserted into one of the nine value classes. If a direction lays between two direction classes , the related magnitude value will be splitted and inserted in both of them

4. Aiming to compare those histogram a normalization will be performed. In this process a 16x16 block will slide along the image . For each step, 4 histogram will be concatenated to form a 36x1 vector and then normalized , performing an L2-norm . The window then will slide of 8 pixels and repeat the process.

5. HOG features is generated for the entire image concatenating the normalized vectors

An example of the HOG application is presented in the figure 4.5 .

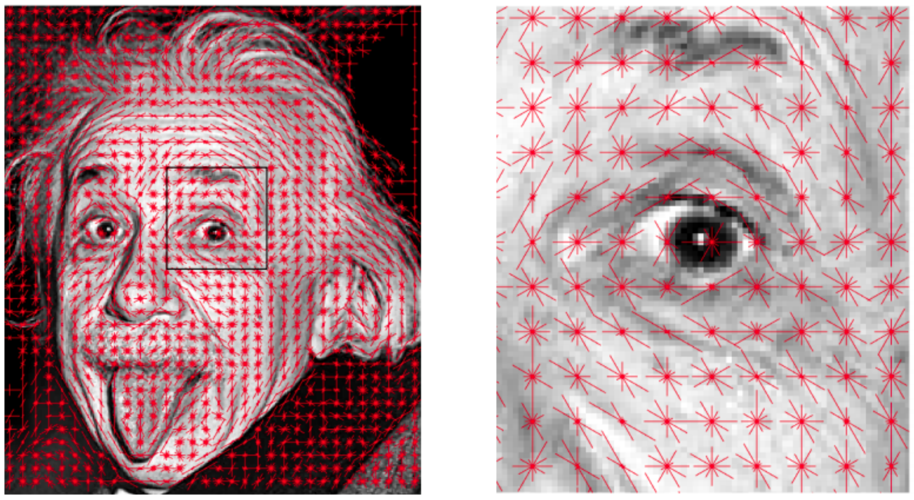


FIGURE 4.5: HOG performing on Einstein images and focus on the eye[25] .

4.2.3 Regional CNN (R-CNN)

It is an innovative approach explained first in the paper of 2014 [26], based on a standard CNN and a new ROIs generative algorithm. Indeed the intrisic problem of the ROIs method is that they could be a huge number and obvously an algorithm that implies a tremendous waste of time is a useless algorithm . The solution in the paper is to work with a standard number of ROIs equals to 2000 focusing on the selection on the right regions, involving a *selective search*. It start from the analyses of the similarity between pixels. Each pixel is merged with other adjacent similar pixels that presents close texture and going on in order to form vast region of similar pixels. At the end 2000 ROIs are selected based on the results of the previous merging. choosing ROIs of different shape and size at different scales. A claryfing example is reported in figure 4.6 .
The overall process includes four steps :

1. Selective search algorithm defines 2000 ROIs from the image

2. A CNN works with each ROI resazing it and extracting the visual features. Their a collected with the category label and the ground truth bounding box

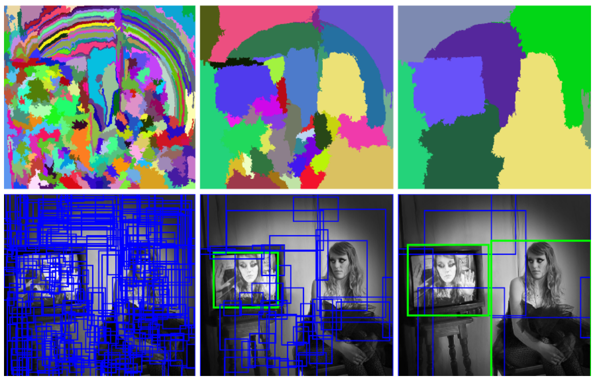


FIGURE 4.6: R_CNN performing object detection. The upper images shows the merge of similar pixels region. The lower images show ROIs generation and selection of the right ones[27].

3. the information are analysed in a set of Support Vector Machines, in order to train them for classification. Each of them should define the right category of the selected region
4. At the end those infos transfer trough a linear regression model in order to predict the bounding box.

Although the R-CNN work with a predefined numbers of ROIs and pretrained CNN, the request time is a weak point of this algorithm . The real-time usage is impossible due to its 47 s of time involved in the analysis of a single image .

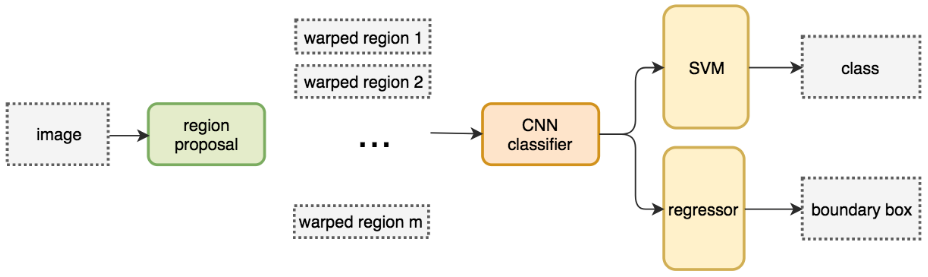


FIGURE 4.7: R-CNN architecture [28].

4.2.4 Fast regional CNN (Fast R-CNN)

It is the evolution of the previous approach , aiming to reduce the time cost. The bottleneck of the R-CNN was the feature extractions for all the proposed region. In the new approach this process is applied to the whole image . Once the visual feature are known the algorithm uses only those related to the region analysed . Moreover the SVMs are removed , in favor of a pooling process . This time the pooling layer purpose is to redefine the shape of each region according to a predefined one. A last sub-net of fully connected layers is the added. In those last layers a softmax and a

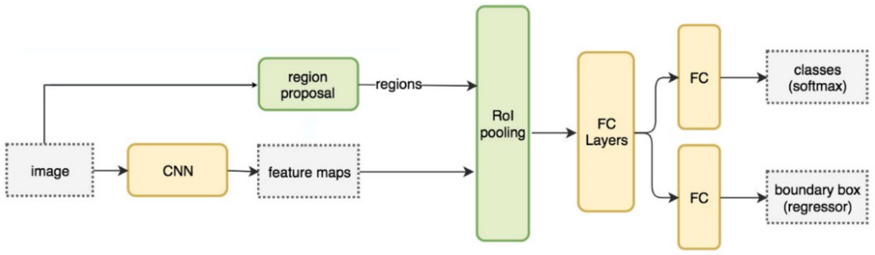


FIGURE 4.8: Fast R-CNN architecture[28] .

regression function are implied , aiming to define respectively class and bounding box of the object in the region .

4.2.5 Faster regional CNN (Faster R-CNN)

The evolution of the previous methods led to an upgrade in terms of architecture and time . This time the bottlneck identified was the search algorithm. It was so expensive due to the large numbers of ROIs and occupied the large part of the processing time. In the new approach that part was replaced with an internal deep neural network based on the feature maps coming from the CNN . This network was named *Region proposal network (RPM)*. Thanks to new method the number of proposed region drastically decreased as the time cost of the process. The main core of the RPM was the generation of the anchor boxes of different size that had probability to contain an object . Those boxes were then sent to the pooling layer as proposed regions. The rest of the architecture was the same of the previous algorithm .

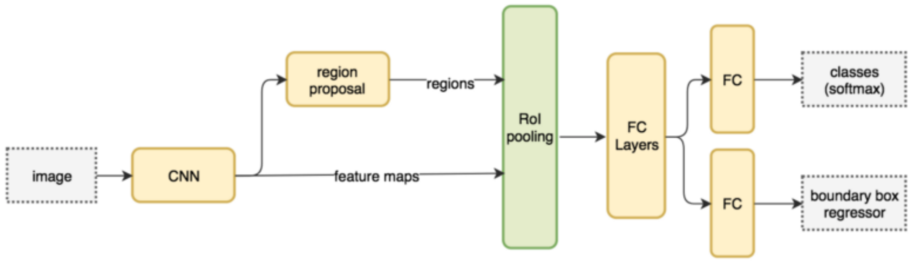


FIGURE 4.9: Faster R-CNN architecture[28].

4.2.6 Single Shot Multibox Detector (SSD)

This algorithm was one of the most performative one when its developer shared the paper [29] in 2016 . It was impressive in terms of performance and precision with a 74% mAP (*mean Average Precision*, on standard dataset used to compare algorithm performances . It is called *Single Shot* due to the single forward pass of the net for

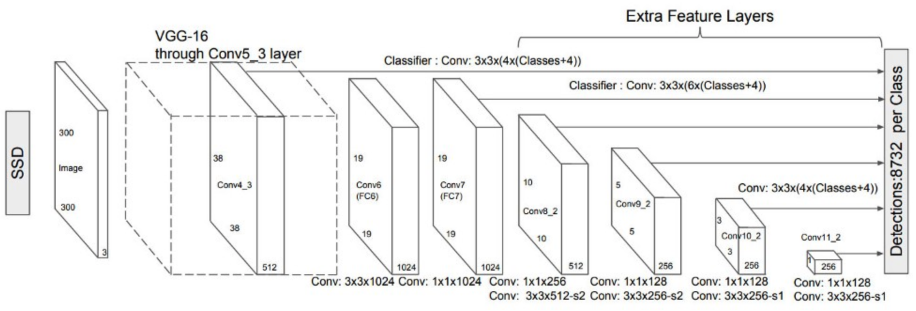


FIGURE 4.10: SSD architecture[30].

the purpose of localization (bounding box) and classification , while the *Multibox* is a specific bounding box regression developed by the researchers .
As is reported in figure 4.10 , the SSD is based on the aformentioned VGG16 architecture , reported in the previous section . It was choosen for its high performance in term of classification. Its main purpose is to extract the feature maps . The fully connected layers of the basis network is replaced with a new series of auxiliary convolutional layers, aimed to extract visual features at multiple scales and sizes . Progressively the original image size is reduced to subsequent layers size . Another improvement is the *priors* introduction. They are the equivalent of the anchors in Fast R-CNN, but they are chosen prefixed, trying to follow the original distribution. The priors has to have an IoU (intersection over union) > 0.5 , that is not so good but it is a strong starting point to get the final results .Indeed the algorithm starts with priors as actual prediction and then tries to regress closer to ground truth bounding box .

Chapter 5

YOLO

5.1 YOLO

You Only Look Once (YOLO) [31] is the first version of the popular object detection algorithm for real-time application . Based on the SSD concept, it performs a more accurate and faster detection thanks to the single network able to localize and classify object. Moreover it can be trained end-to-end implying a further improvement in the accuracy . Despite some week points in localization errors, it was able to reduce the false positive in background, compared to the state-of-art of the object detection in 2016 .

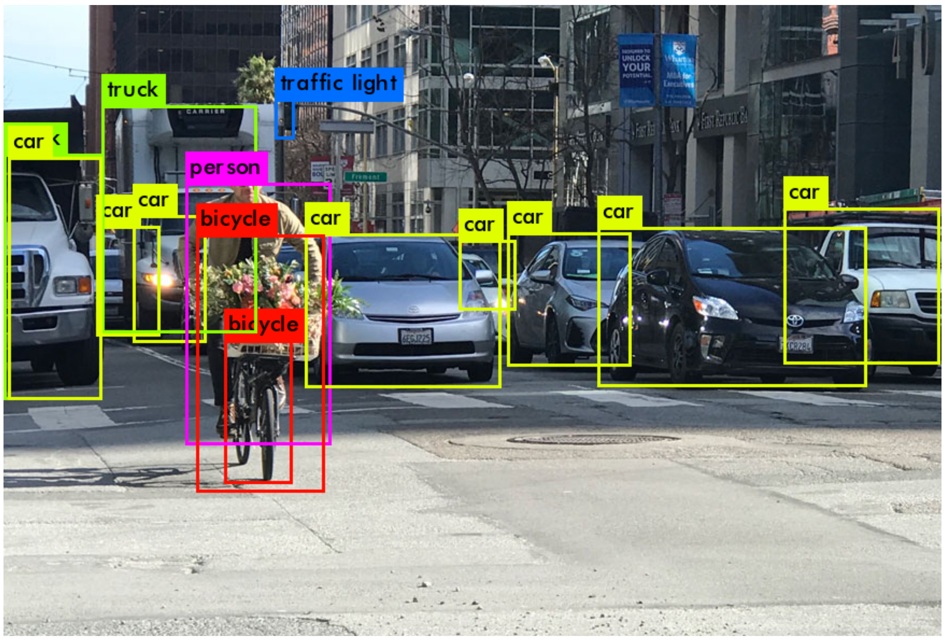


FIGURE 5.1: YOLO object detection example[32].

5.1.1 How it works

The core of the overall process can be sinthetized in four steps :

1. The image is dived into SxS cells (the original algorithm had S=7)
2. Each cell predicts B bounding box and box confidence score . This latter defines how sure the network think there is an object in a bounding box. The bounding box with higher score are drawn fatter ,o in order to visual display the confidence . Each bounding box presents 5 related values : *x,y,w,h,c* . The *(x,y)* is the center position of the bounding box in the image, while *(w,h)* are the weight and height of the bounding box . Last element *c* is the confidence score
3. Regardless of the bounding box, each cell predicts C class probabilities values . This value is conditioned to the presence of an object in the box . Depending on the training dataset C has variable values . In *PASCAL VOC* classes number was equal to 20, so C=20 .
4. A *Non-Maximal Suppression* process is applied in the end, in order to remove duplicates with lower scores and improve the accuracy of 2-3& mAP . The output is a tensor of SxS(Bx5+C) tensor .

5.1.2 Network Design

It is inspired by the *GoogLeNet*, a famous object detection algorithm that partecipated to the ILSVRC 2014 . It is composed of 24 convolutional layers plus 2 ending fully connected layers.The convolutional part is devoted to extract features while the fully connected part is responsible for the final predictions and coordinations. In the algorithm the *inception modules* of GoogLeNet is replaced by a series of 1x1 reduction layer followed by 3x3 convolutional layers.

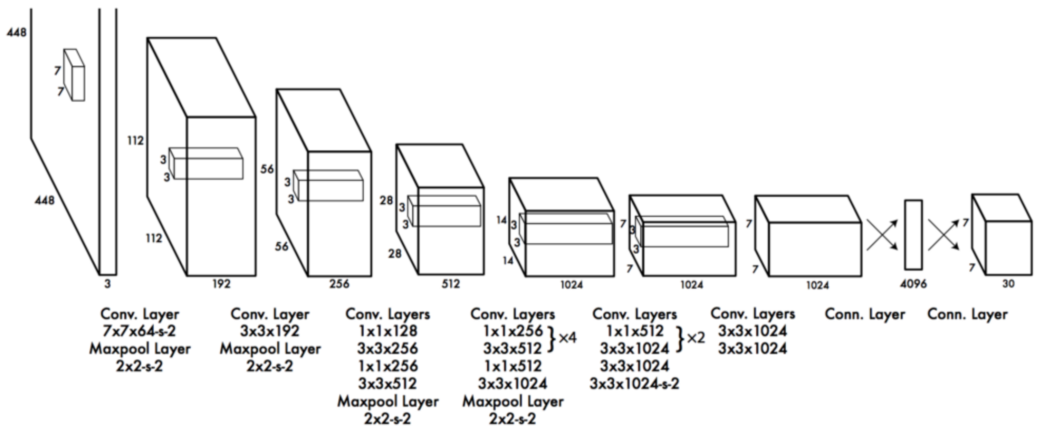

448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

FIGURE 5.2: YOLO 1 architecture[31].

5.1.3 Loss Function

YOLO predicts multiple bounding boxes per each cell. During the training phase only one bounding box predictor is devotes to focus on each object. The respondible

```
Layer        Kernel  Stride      Output
--------------------------------------------
Input                          (416, 416, 3)
Convolution    3x3      1      (416, 416, 16)
MaxPooling     2x2      2      (208, 208, 16)
Convolution    3x3      1      (208, 208, 32)
MaxPooling     2x2      2      (104, 104, 32)
Convolution    3x3      1      (104, 104, 64)
MaxPooling     2x2      2      (52, 52, 64)
Convolution    3x3      1      (52, 52, 128)
MaxPooling     2x2      2      (26, 26, 128)
Convolution    3x3      1      (26, 26, 256)
MaxPooling     2x2      2      (13, 13, 256)
Convolution    3x3      1      (13, 13, 512)
MaxPooling     2x2      1      (13, 13, 512)
Convolution    3x3      1      (13, 13, 1024)
Convolution    3x3      1      (13, 13, 1024)
Convolution    1x1      1      (13, 13, 125)
--------------------------------------------
```

FIGURE 5.3: YOLO 1 layers description[32].

predictor is chosen on the base of the highest IoU with the ground truth . This leads to specialize each predictor to better predict certain classes or sizes of object. The loss fucntion optimized for this process is composed of three sub-loss-function :

- Localization loss
- Confidence loss
- Classification loss

Localization Loss
The error is related to bounding box size and location. In order to penalize only the bounding box with an object inside, the function has the " $\mathbb{1}_{ij}^{obj}$ " element that is devoted to this task :

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left[(x_i-\hat{x_i})^2+(y_i-\hat{y_i})^2\right]+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left[(\sqrt{w_i}-\sqrt{\hat{w_i}})^2+(\sqrt{h_i}-\sqrt{\hat{h_i}})^2\right]$$

Where:

- $\mathbb{1}_{ij}^{obj}$ has only two valuse : 1 if an object is in the cell, otherwise is 0. The j denotes that the predictor of the *j-th* bounding box is responsible for that prediction
- λ_{coord} is a coefficient used to stabilize the model , penalizing localization error more than confidence error. In the original paper tis coefficient was set to 5
- the square root of w and h are used in order to penalize more small deviation in small bounding box rather than small deviation in larger bounding box

Confidence Loss
The error related to confidence is :

$$\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\Big(C_i-\hat{C}_i\Big)^2$$

Where :

- λ_{noobj} is a coefficient involved in the stabilization of the function. It is normally set to 0.5
- $\hat{C}_i$ is the confidence of i-th box

Classification Loss
In classification error the formula used is :

$$\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{i}^{obj}\Big(p_i(c)-\hat{p}_i(c)\Big)^2$$

Where :

- $\mathbb{1}_{i}^{obj}$ has the same function of the previous similar , but this this depends only on the presence of an object in i-th cell
- $\hat{p}_i(c)$ is the conditional class probability in that cell

5.1.4 Inference

In terms of prediction , YOLO is extremely fast due to the single network evaluation. Furthemore *grid design enforces spatial diversity in the bounding box predictions* . When large objects are involved, usually they can be predicted two times form different box, but thanks to the aformentioned non-maximal suppression the multiple detection issues is fixed

5.1.5 Limits

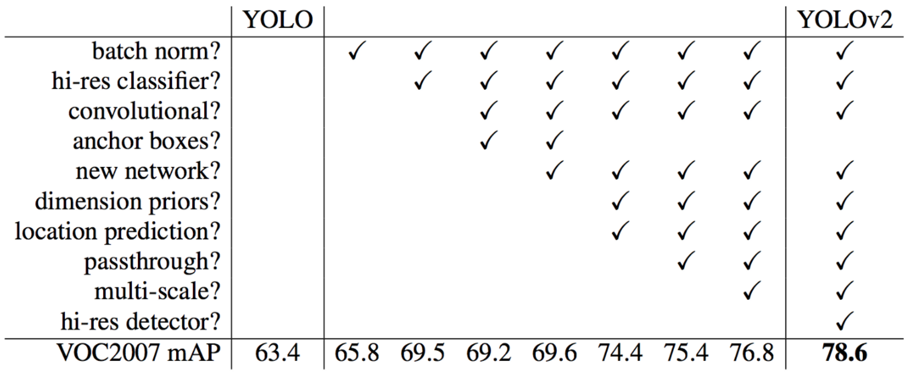
The main limitation is in the number of predictible objects. Indeed , do to fixed bounding box and cells numbers, only SxS object can be predicted . Moreover nearby object are limited by the spatial constraints . It presents some problems in the generalization of an object, like new aspect ratio or configurations .

5.2 YOLO 2

It is the evolution of the previous YOLO algorithm , released in 2016 [33]. It contains some improvement and changes in architecture .

5.2.1 Accuracy Improvements

The improvements are reported in the figure below, from the original paper :



	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

FIGURE 5.4: YOLO 2 improvements description[33].

5.2.2 Batch Normalization

Thanks to batch normalization , no other regularization form like dropout are applied. Indeed it leads to a convergence improvement and 2% more on mAP. This method is applied on all the convolutional layers .

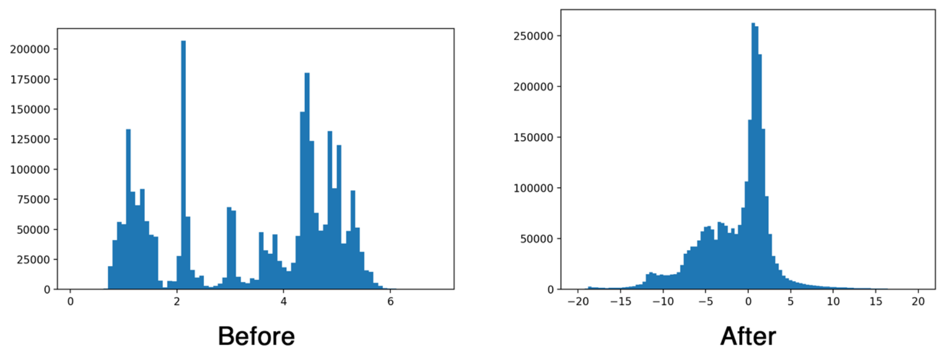


FIGURE 5.5: Batch Normalization example[34].

5.2.3 High Resolution Classifier

Another improvement is related to the augmenting of the classifier resolution. The previous version had the first part used as classifier as 224x224 while the ending part as detector as 448x448 .In YOLOv2 the first 10 epochs use a 448x448 resolution both for classifier and detection in order to let the network adjust filters for high resolution . Thanks to this change, researcher was able to get a n increase of 4% in mAP .

5.2.4 Convolutional with Anchor Boxes

In order to make predictions , YOLO involves the so called *anchor boxes*. Despite the competitors like Fast R-CNN who uses priors with a random selection of boxes, the algorithm starts by a prefixed offset range . Working with the offset instead of coordinate make it easier for the network to learn . According to this concept in YOLO v2 the fully connected layer was removed in favor of anchor boxes . The resolution changed from 448x448 to 416x416 with the purpose to have a central cell that can be responsible for a central large object .

5.2.5 Dimension Clusters

Instead of choose anchors by hand researchers decide to let a clustering algorithm help them in decision . Indeed starting by a good priors can help the network in learning . Thank to a k-means clustering on the training dataset suitable anchor boxes are decide . The k value is the numbers of clusters in which the dataset can be grouped . This number has to be chosen in order not to increase too much model complexity , but good enough to obtain an acceptable average Iou as it possible to see in the picture .

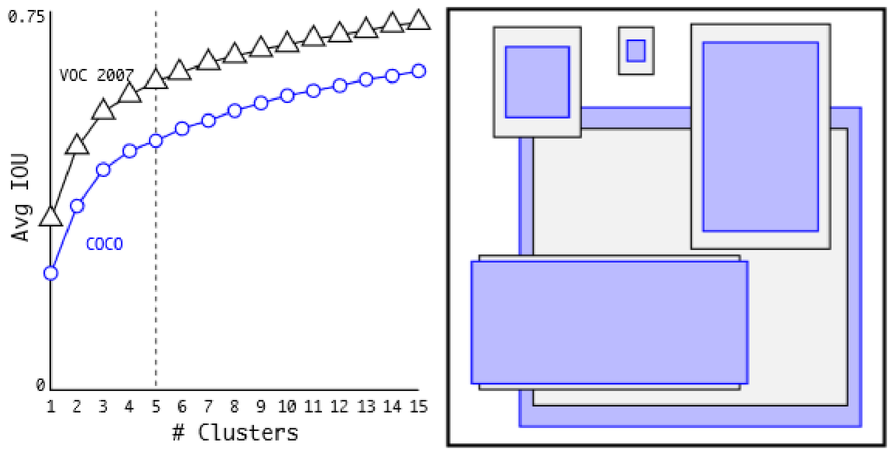


FIGURE 5.6: Anchors selection based on k-mean and average precision[33].

In this case k is equal to 5 as good tradeoff between model compelxity and high recall . In the picture can be notice that there are more thin and tall boxes than short and wide .

5.2.6 Direct location prediction

Most of the instability in training phase comes from the bounding box location prediction *(x,y)*. A possible approach as in Faster R-CNN is to use the offset, but it will take tame to stabilize the model with a random inizialization. Instead of predicting offset,

in YOLO v2 the focus is on predict location relative to location of the grid cell. This way there is a bounded range between 0 and 1 from the ground truth.
The network predicts 5 bounding boxes for each cell and 5 values are related to each bounding box . They are the bounding box coordinates plus a confidence score *tx,ty,tw,th,to*. If the cell has (c_x, c_y) offset from the top left corner of the image and anchors has width and height p_w, p_h it is possible to obtain :

$$b_x = \theta(t_x) + c_x$$

$$b_y = \theta(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

$$Pe(object) * IOU(b, object) = \theta(t_o)$$

Where :

- bx,by are the predicted bounding box coordinates
- bw,bh are the weight and height of the predicted bounding box
- c_w, c_h are normalized cell coordinates by the image size
- $\theta(t_0)$ confidence score related to the box

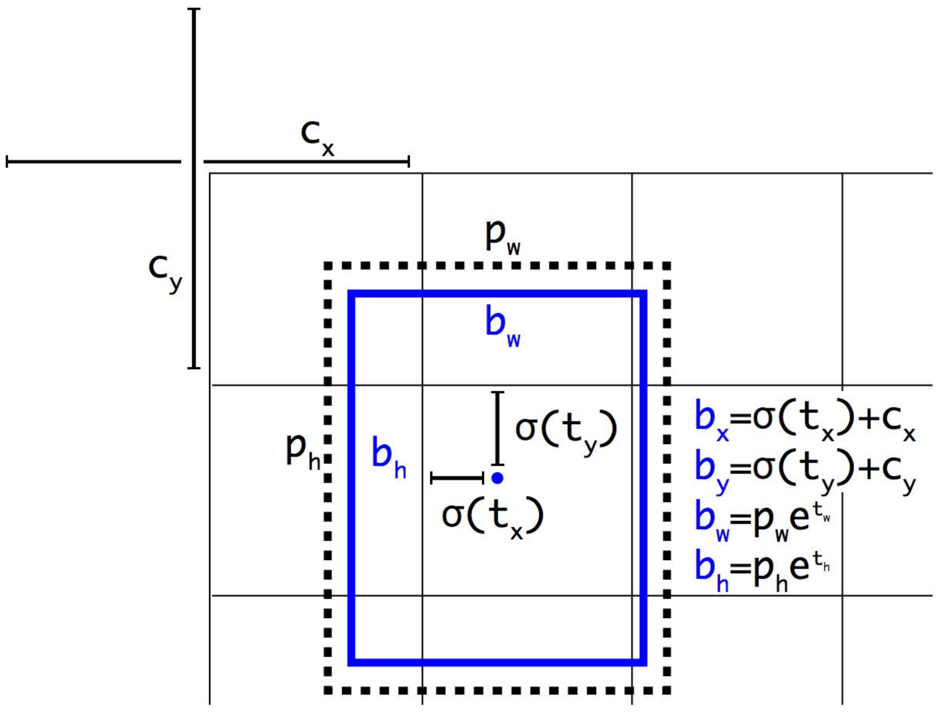


FIGURE 5.7: YOLO 2 bounding box description[33].

5.2.7 Fine-Grained Features

The algorithm works with a 13x13 feature map in order do perform detection. This is a quite good resolution for larger objects, but it is not suitable for smaller object. While other competitors, like SSD or Faster R-CNN, make prediction on different feature maps scales, in YOLO 2 the approach is different . In fact, it simply adds a passthrough layer in order to obtain a 26x26 feature map resolution. This method guarantees the single forward in detection . The two feature maps can be concatenated and the detector can perform on the expanded feature maps, getting access to the fine-grained features , improving performances of 1%.

5.2.8 Multi-Scale training

Instead of prefix an input image resolution in YOLO 2 the resolution changes randomly every 10 batches . This improvement leads the network to predict across a variety of input dimension. The architecture permits downsample of factor of 32, so the suitable resolutions start from 320x320 to 608x608 with a path of 32 per each resolution between this values . Thanks to this approach YOLO v2 can run fro lower resolution input, having performances comparable with Fast R-CNN with a 90 FPS or for high resolution input performig extremely well, with 78.6 mAP .

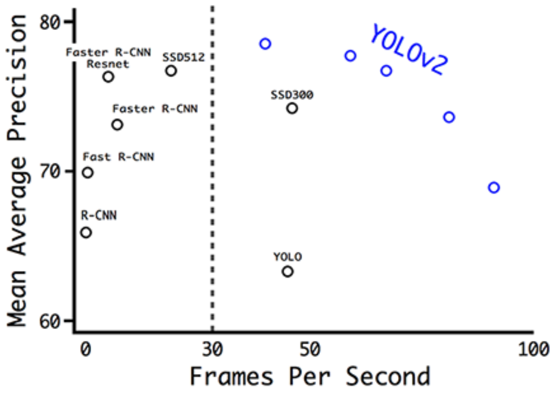
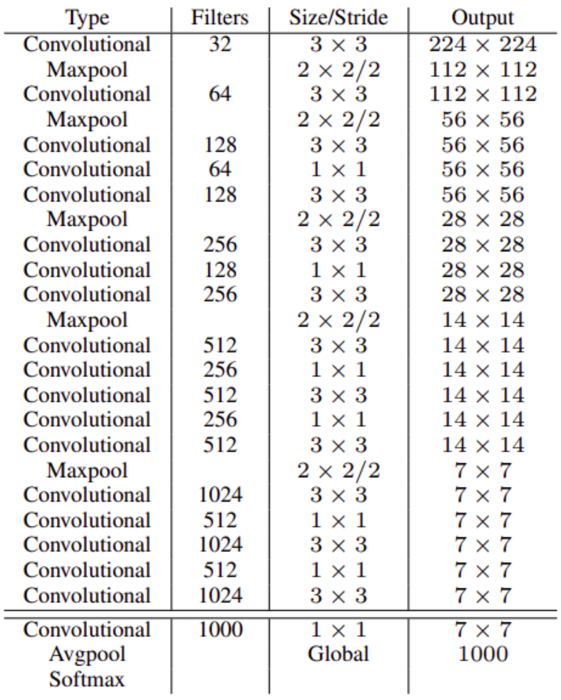


FIGURE 5.8: YOLO 2 comparison w.r.t comptetitors[33].

5.2.9 Architecture Design

Because of the computational cost of the VGG16 (30.69 BFLOPs) , in YOLO v2 the feature extractor has been replaced by GoogLeNet that requires only 8.52 BFLOps for the analysis of an image . Despite a slight drop in accuracy from 90% to 88% it gets a impressive increase in terms of speed .
The new adopted classifier is called *Darknet-19*. As in VGG models 3x3 filters are used, but doubling the channels number after each pooling layer .Moreover a global average pooling process is been involved in order to prediction .The feature map is is compressed by 1x1 filters between 3x3 convolutional layers. The overall architecture

presents 19 convolutional layers and 5 maxpooling layers. It requires only 5.58 BFLOPs to perform detection on images .



Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

FIGURE 5.9: YOLO 2 architecture[33].

5.2.10 Hierarchical classification

Usually object detection dataset has less object classes than dataset specifically created for classification . They both are used in YOLO v2. When the image is labeled for detection the loss function used is the entire function previously seen, while in case of classification image , only the classification part of the function is involved . The problem in the use of both types of dataset is that as assumption the dataset is flat so classes are mutually exclusive, but this is not the case. For example *Yorkshire terrier*(ImageNet) and *dog*(COCO) are not mutually exclusive. The solution of merging different dataset is the *hierarchical classification* . It is an approach that allows to link to different classes from to different dataset in order to create a *WordTree* in which *Yorkshire terrier* could be a child-node of *dog*, ans so on with all the possible classes . YOLO v2 predict the confidence of all the node in a branch in order to have the final result of the last node . Frequently the more node is deep in branch the less confidence it will have. The starting node is *physical object* node .

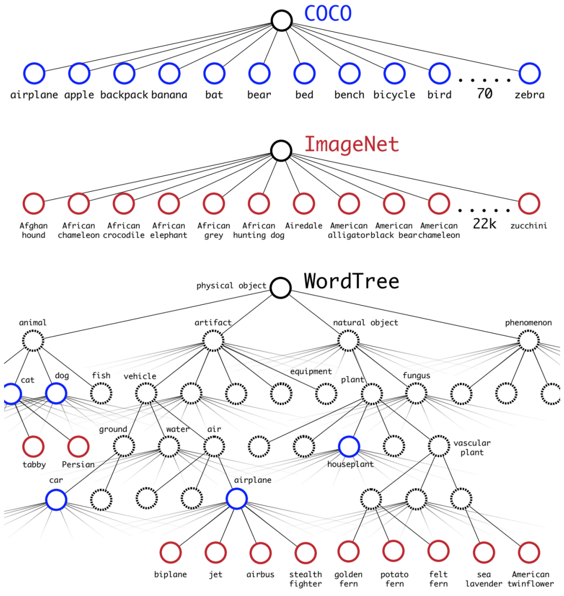


FIGURE 5.10: World Tree visualization[33].

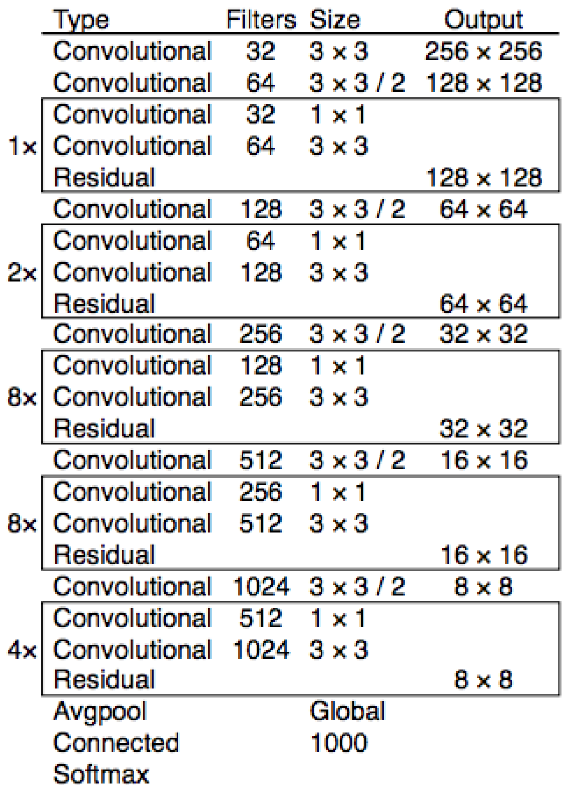
5.3 YOLO 3

YOLOv3 [35] is the last evolution of YOLO's project. It contains some new upgrade in terms of architecture and other design changes. The softmax function is replaced by an indipendent logistic classifier so that can handle non-mutually exclusive classes prediction . The mean square error is replaced by a binary cross-entropy loss for each label . Moreover it changes the cost function approach. If the anchors overlaps the ground truth more than other priors, it will recevie an objectness score equal to one. The rest of the anchors with IoU>0.5 won't be penalized .

5.3.1 Prediction

The new predictive algorithm works simultaneously with 80 classes prediction due to the non mutually exclusive approach and make 3 prediction per location. Each prediction is composed of 80 classes prediction, an objectness score and 4 values related to bounding box location and size. In the end there are NxN*(80+1+4) results . In order to make 3 predictions at 3 different scales, the algorithm :

- Realize the first prediction in the last feature map layer
- Goes back to 2 layers back and upsample that layer by 2 . Then merges the upsampled feature map with the other feature map at high resolution trough an element-wise addition. Then apply a convolutional filters to make the second prediction
- Repeat the second step so as to have a fine-grained feature map in order to obtain a good resolution spatial info on object location and male the third prediction



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

FIGURE 5.11: YOLO 3 layers description[35].

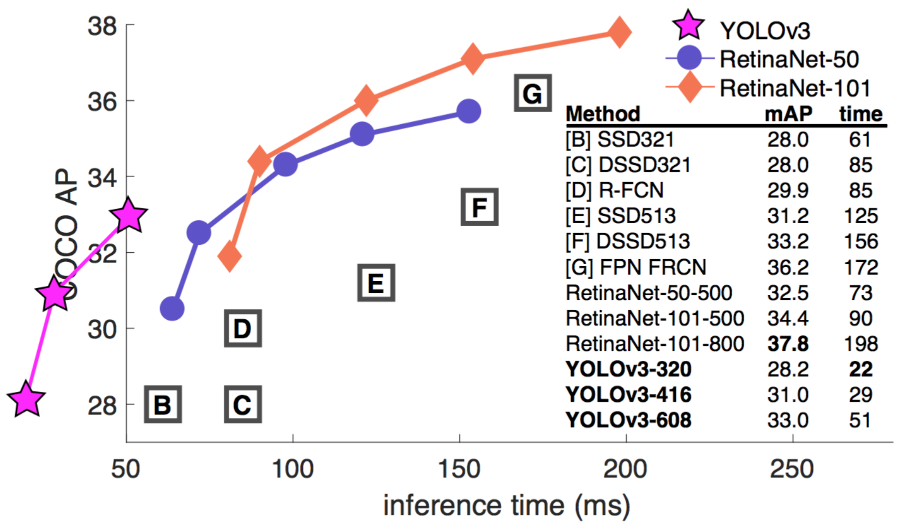
As in YOLO v2, in this last evolution the k-means is used to cluster the dataset, but 9 priors(or ancors) is used for the new algorithm .

5.3.2 Feature Extractor

The feature extractor increases its layers arriving to *Darknet-53*. YOLO v3 has 53 layers, mainly 3x3 and 1x1 convolutional layers . Moreover it presents *skip connection* between layers . It has the highest BFLOPs of the different version so it requires a GPU hardware . Nevertheless it performs 2x faster then ResNet-152 with less BFLOps .

5.3.3 Performance

YOLO v3 has the highest performance in terms of inference time on COCO dataset. Moreover it obtains a good performance even in small objects detection .



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

FIGURE 5.12: YOLO 3 comparison w.r.t. competitors[35].

Chapter 6

Metrics

Once a machine learning or deep learning algorithm has been trained the following crucial step is to evaluate it. The evaluation step is important in order to understand the correctness of all the previous step and better understand the results . Moreover it is the based on which the different algorithms proposed are compared .
During the evolution of deep learning many methods have been proposed and each of them tries to focus on some aspects . In the following sections some of them will be analysed .

6.0.1 IoU

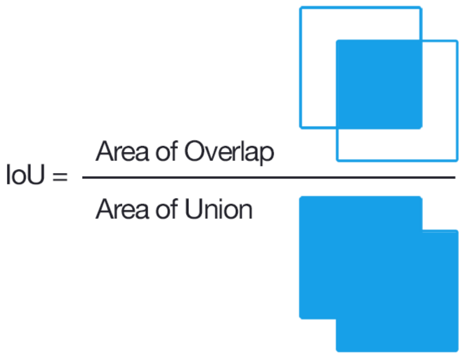
It is called *Intersection over Union* and it is the most common metric in object detection field. Moreover it is a basic metric for building other metrics . This metric is particularly used in presence of bounding box and represents the accuracy in detection of the algorithm performing on a validation dataset. It is based on the *ground truth*,namely the bounding box labelled by programmer and the predicted bounding box. The IoU measures the overlapping area between those two boxes :

$$IoU = \frac{AreaofOverlap}{AreaofUnion}$$

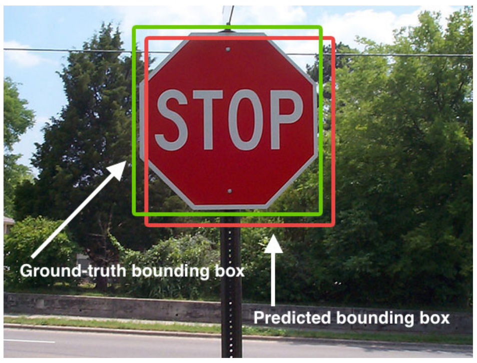
Where :

- Area over Overlap is the overlapping area of the two boxes
- Area of Union is the merge of the two area

IoU values are bounded between 0 ans 1. If IoU=1 the predicted bounding box perfectly match the ground truth, while if IoU=0 there is no overlap between those two region. The IoU is a crucial element in the construction of other metrics . Indeed on the contrary of a classification problem, in object detection there is no boolean metrics; classification prediction a possible metric is to check if the predicted class is correct or not, a boolean metric. In object detection the problem is " how to define if the



(a) IoU example



(b) Ground truth and predicted bounding box

FIGURE 6.1: [36]

prediction is good or not ? " . The most common solution is based on the IoU value : if the IoU>threshold the prediction is considered *positive* . In general the standard threshold is equal to 0.5. but other thresholds are considered too.

6.0.2 Confusion Matrix

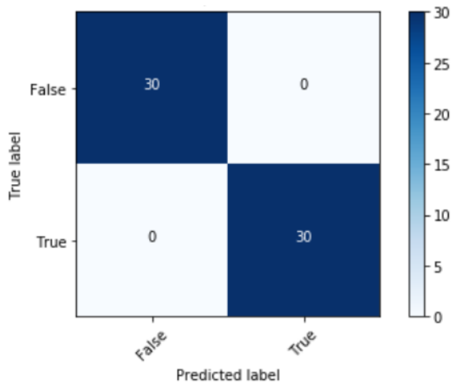
It is a simple method to visualize the correctness of an object detection system. It is based on four elements :

- TP : True positive
- TN : True negative
- FP : False Positive
- FN : False negative

Where *True* or *False* refers to the ground truth, while *Positive* or *Negative* refers to the predicted bounding box. E.g., a *True positive* result means that the predicted bounding box is considered positive , with a IoU over some threshold and the related ground truth is there. A *False positive* indicates a prediction in which there is no ground truth to compare hence there is no object, but the algorithm predicted a bounding box . In a *confusion matrix* all the possible cases are considered and visualised in a matrix structure. In this case :

		Predicted P	Predicted N
Real	P	True Positive (TP)	False Negatives (FN)
	N	False Positive (FP)	True Negatives (TN)

(a)



(b) [37]

FIGURE 6.2: Confusion matrix with four possible cases

6.0.3 Precision and Recall

Those two metrics are widely used in most of the object detection algorithm. They are based on the four cases previously examinated . Ther mathematical definition is :

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$$

Precision indicates the accuracy of an object detection algorithm that performs on a validation dataset. It is the rate of the true positive on the entire predicted boxes. If *Precision*=1 it means that all the predicted objects are correct.
Recall indicates how well the algorithm perform on the total amount of possible detectable cases. it is the rate between true positive detected and all the positive detectable.
The definition of TP and TN depends on the IoU threshold. As previously said, a common threshold is IoU=0.5 . Decreasing the IoU could lead to increase the precision because of a possible decreasing of the FP and so of the denominator of *precision* definition. It is worth to notice that the performances of an algorithm depends on the IoU set and for a complete comprehension of them metrics have to be analysed under different IoU thresholds . A good classifier tends to maintain an high precision increasing the racall value, while for a weak classifier precision tends to decrase if recall increase.In order to better visualise the behaviour of the twe metrics a further metric element was introduced, it is the*Precision-Recall curve* . In this curve for different recall points, the precision behaviour is shown:
Usually the number of points depends on the dataset involved . In order to take into account precision and recall, researchers has defined a third metric , the *F1 score.* It is the harmonic mean of the two metrics :

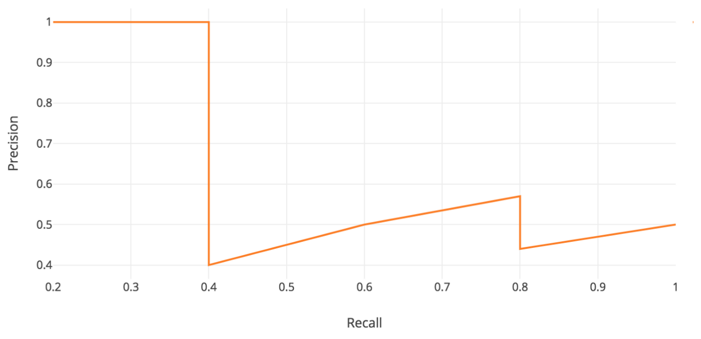


FIGURE 6.3: [38] Precision-Recall curve.

$$\text{F1} = 2\frac{PxR}{P+R}$$

There are many others metric that combines p. and r. , but it is the most used because gives the same weight on both of them and punisces extrems so if one on the two metrics is too low , F1 decrease significantly .

6.0.4 Average Precision(AP) and mean Average Precision(mAP)

Average Precision

Average Precision is a popular metric used to compare object detector algorithms. It is based on *Precision* values in *Precision-Recall curve.* It is worth to notice that generally the curve presents a zig zag pattern, hence it is not easy to handle directily with those values. During the tim,e a lot of different definition has been chosen in order to compute this value in an easy way . One of the most popular defines the AP as the interpolated precision. Another very similar approach is the trapezoidal interpolation of precision called *Area Under Curve(AUC).* Many times the two terms coincide, depending on the kind of interpolation .

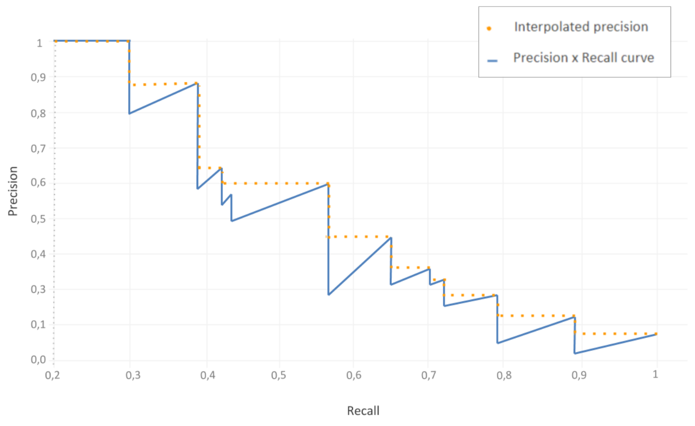


FIGURE 6.4: [39] Average Precision computed on P-R curve.

mean Average Precision

The current popular dataset like MS COCO present many different classes. In order to compare the general performance of a multi-class object detector , the mAP is used. It is the mean of all the AP belonging to the different classes.

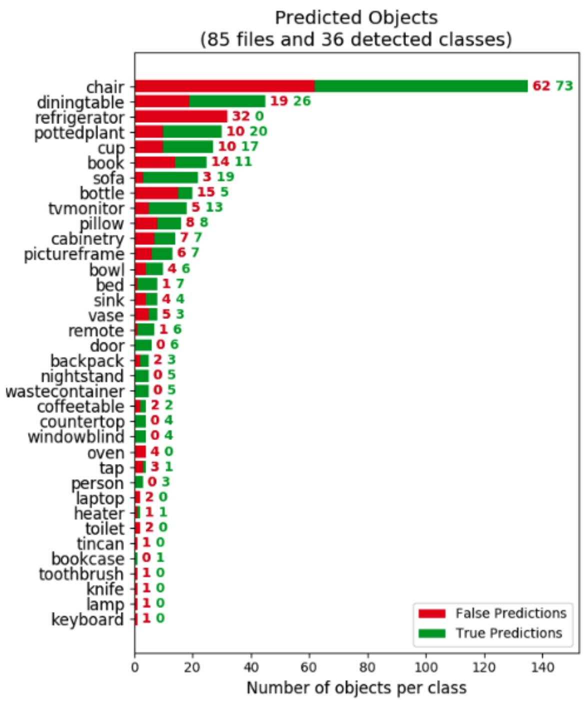


FIGURE 6.5: Average Precision on COCO dataset of YOLO 3[40].

Chapter 7

Sensors and embedded system

As a general framework in object detection two elements are fundamentals : embedded system and sensors. The first one is the "brain" where information are computed. An embedded system it is a computer system designed for a specific purpose. They can have a User Interface or not, like an embedded system integrated in a more complex system that has to perform some specific tasks. Sensors are the *interface* with the external world . In many cases and in particular in object detection field the cameras are a crucial sensor in order to get the information that the embedded system can process.
A third element that has to be taken into account are the constraints in which the entire system (embedded system+sensors) has to work. For this thesis project the major constraint is the particular application involved,namely an object detection application on Mars. The space environment (space travel and work on another planet) leads to numerous risks for electronic devices [41]. One of the most dangerous element in space are radiation. Due to their capacity to penetrate matter, space radiations are able to interact with electronic devices and especially with *memories parts* and change their settings or internal states of the memories. Moreover there are many problems related to *degradation.* Change in temperatures , extreme temperatures , vacuum and electronic discharged are others problem suffered by satellites even in low earth-orbit . In the following sections it will be shown sensors and boards involved in this thesis project .

7.1 Camera

Camera sensors are an essential part of an object detection system. Many types of cameras was built for robotic application from commercial to industrial types. Many of them are based on different visual system and numerous cameras present more than one type of visual system . Before analysing the cameras involved in this thesis , a brief overview of the general visual system will be reported.

RGB Camera

This is the standard type of camera present in every cell phone. It is widely used in robotics because it is very similar to our visual system, getting a similar information from the external world. Moreover today they are a very cheap sensors and mounted almost on every robotic system that perform visual analysis, from rovers to drones,under-water robots or industrial machines for vision application.
The information extracted from RGB camera are analysed and processed in order to give-back a 2D image of the external reality. Those 2D images can be processed together so as to reproduce a 3D world, as we perceive it.

Depth Camera

They are cameras able to give-back the *distance* between the framed objects and the camera. Actually this types of cameras don't rely on a particular *"depth sensor"* as the RGB camera, but there are different solutions based on different sensors and a following information processing that perform the task to give-back the distance of each pixels.
The most popular depth camera systems are :

- Time of Flight(ToF)
- Stereo-camera
- Laser scanner
- Structured-light scanner

Time of Flight is based on light impulse . Those impulse emitted by the camera hit an object and come back to their source. Camera sensors capture the returning impulse and analysed the time needed for the impulse to be emitted and come back, called *Time of Flight*. It is related to the distance from the hit object. This is a very accurate system that allows a long working range and small size , but it is expensive .

3D Laser Scanner is based on triangulation. The laser is first emitted by the camera. When the laser hit an object ,this latter reflects it and laser initial trajectory changes. Camera sensors pick up the trajectory modification and from trigonometric calculation can be determined the change in angle, directly linked to the distance from the scanned object.

Structured-light is based on projection of a known light pattern onto a scene. When the pattern points strike the object they are deformed by object surface. A sensor capture the distorted pattern and the information is analysed in order to reproduce the struck reflective surface. It is very similar to 3D Laser scanner approach but it is faster due to the pattern projection onto the entire seen simultaneously . The more the pattern is dense, the more accurate will be the results. Usually the light source work in infrared range.

Stereo-camera is mainly based on human vision system. In fact they reproduce a binocular vision with two lenses that capture they same scene from different perspective. As in human case the single image is flat, in a 2D world but processing together the information from both of them lead to re-create a 3D world.
In a stereo vision process the images are first analysed with a *stereo-matching* algorithm. This algorithm tries to find correspondence region between the two images so as to link pixels . Those linked pixels identify the same external point ,for example the same hand or foot as in figure 7.1 . The difference calculated in pixels between the two points is called *disparity* and is related to the distance between the object and the camera. It is a phenomenon that everybody has had experience : if we alternatively close one eye , the more an object is near our eyes the more we tend to perceive it shifted between the single eye images.

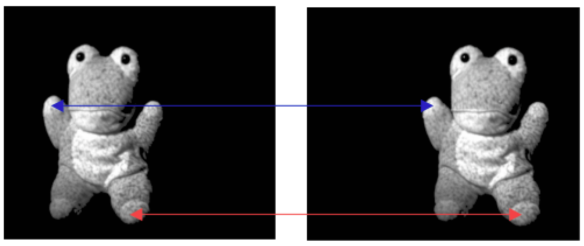


FIGURE 7.1: Feature Matching on stereo image[42].

If all the disparity values are computed the algorithm can visualize the disparity map, namely an image in which for each pixel is associated the related disparity value . Choosing the right color visualization it easy to understand the distance information behind the disparity map :

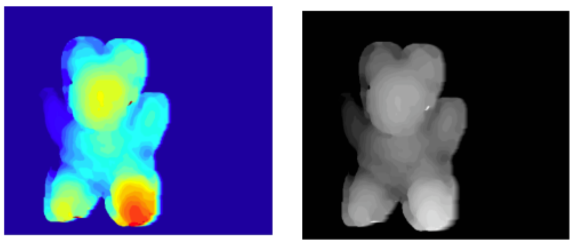


FIGURE 7.2: Disparity Map with different colours representation[42].

Once the disparity map has been obtained the depth can be calculated trough triangulation methods . In figure 7.3 it is shown a simplification of two lenses looking at a P point. Coordinates (X,Y,Z) are the coordinates in [mm] of the point in a camera reference frame(generally the reference is aligned with the origin of the left lens), while (x,y) and (x',y') are the coordinates in pixels of the point in a 2D camera reference frame.

Supposed the camera lenses perfectly aligned, working on similar triangles POO' and PAA' it is possible to compute the distance Z as :

$$Z = \frac{B * f}{x - x'} = \frac{B * f}{d}$$

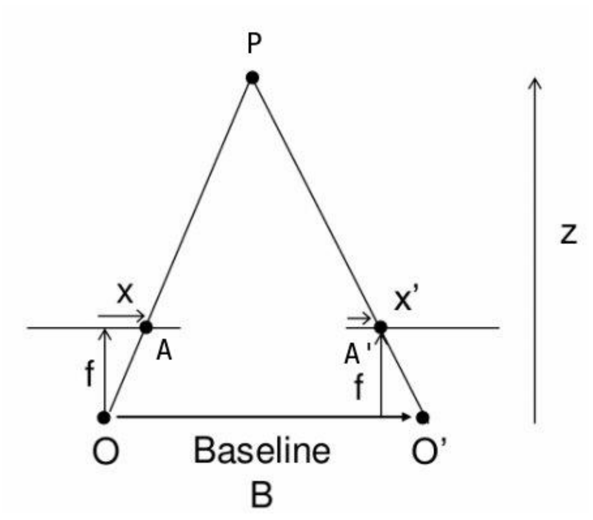


FIGURE 7.3: Modeling of two lenses capturing the same point[43].

Where :

- *Z* [mm] is the distance of the point from the lenses
- *O , O'* are the origin of the lenses coordinates and represent their centers
- *x , x'* [px] are the horizontal coordinates of the point in the *image plan*
- *B* [mm] is called *baseline* and it is the horizontal distance between the lenses
- *f* [px] is called *focal length* and it is the distance form the center of the lenses to the *image plane.*
- *d* [px] is called disparity

In order to have the other two coordinates of the point a simple proportion has to be computed. Finally the 3D coordinates are :

$$\begin{cases} X = \dfrac{x * Z}{f} \\ Y = \dfrac{y * Z}{f} \\ Z = \dfrac{B * f}{d} \end{cases}$$

7.2 Hardware components

7.2.1 Camera

Due to availability constraints and according to company willingness two cameras have been used in this thesis project :

- Bumblebee® XB3 FireWire
- Intel® RealSense™ Depth Camera D435i

Bumblebee® XB3 FireWire

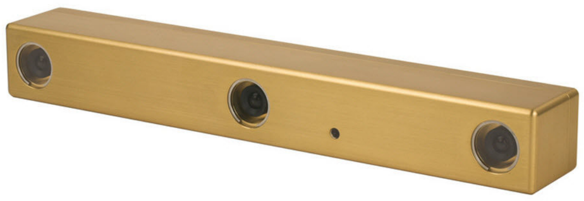
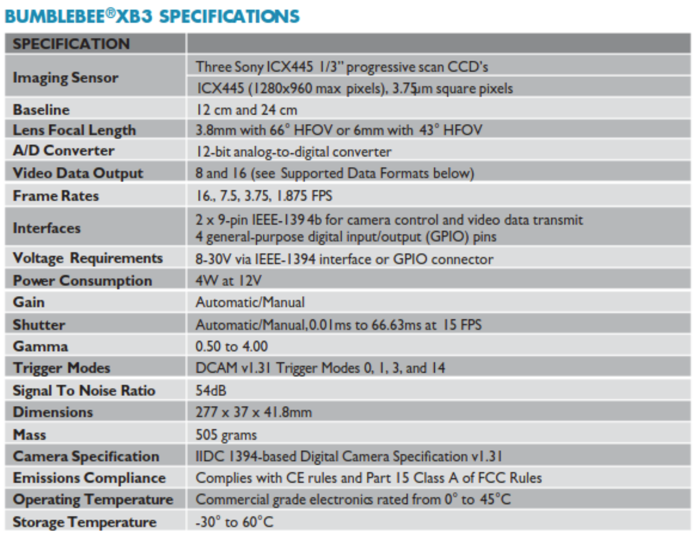


FIGURE 7.4: Bumblebee XB3 FireWire[44] .

The Bumblebee® XB3 is a 3-sensor multi-baseline camera.It has 1.3 mega-pixel sensors and presents two baselines. Due to this advantage it is able to change the baseline involved in order to modify the reachable distance range . The extended baseline and high resolution provide more precision at longer ranges, while the narrow baseline improves close range matching and minimum-range limitations. It was designed to improve the flexibility thanks to its 3 "eyes". The camera has a resolution of 1280x960 . For further information it is possible to visit the product website [45] .
It works on IEEE-1394b interface and it has been a problem to connect to common notebook because to the lack of that specific peripheral. The solution was to introduce an "ExpressCard- Fire-wire " adapter .It works with an external power supply of 12V. The developers camera kit provides *Triclops Stereo SDK* and *FlyCapture SDK* with the purpose to perform a fast stereo-matching and easy image acquisition. In order to use the camera, ROS platform has been used.Two packages have been fundamentals in the use of the camera : *camera1394stereo* package, that allows you to use a camera that relies on the IEEE-1394 standard and *bumblebeexb3* package. The main problem is that all of the internal specific camera parameters cannot be picked automatically by the software. Because of this lack of information ,initially the images taken from the camera in ROS suffer from distortion problem. In the interest of getting a suitable image results a first *calibration process* is needed. Thanks to *cameracalibration* package and *OpenCV*, a popular computer vision software, it possible to calibrate the camera, namely to find internal camera parameters that allows the *rectification* of the images. Due to a know pattern framed by the camera in different position and with different



BUMBLEBEE®XB3 SPECIFICATIONS

SPECIFICATION	
Imaging Sensor	Three Sony ICX445 1/3" progressive scan CCD's ICX445 (1280x960 max pixels), 3.75µm square pixels
Baseline	12 cm and 24 cm
Lens Focal Length	3.8mm with 66° HFOV or 6mm with 43° HFOV
A/D Converter	12-bit analog-to-digital converter
Video Data Output	8 and 16 (see Supported Data Formats below)
Frame Rates	16., 7.5, 3.75, 1.875 FPS
Interfaces	2 x 9-pin IEEE-1394b for camera control and video data transmit 4 general-purpose digital input/output (GPIO) pins
Voltage Requirements	8-30V via IEEE-1394 interface or GPIO connector
Power Consumption	4W at 12V
Gain	Automatic/Manual
Shutter	Automatic/Manual,0.01ms to 66.63ms at 15 FPS
Gamma	0.50 to 4.00
Trigger Modes	DCAM v1.31 Trigger Modes 0, 1, 3, and 14
Signal To Noise Ratio	54dB
Dimensions	277 x 37 x 41.8mm
Mass	505 grams
Camera Specification	IIDC 1394-based Digital Camera Specification v1.31
Emissions Compliance	Complies with CE rules and Part 15 Class A of FCC Rules
Operating Temperature	Commercial grade electronics rated from 0° to 45°C
Storage Temperature	-30° to 60°C

FIGURE 7.5: Bumblebee XB3 details[44] .

angles it is possible to recognize those parameters , reconstructing the camera *pinhole model*.

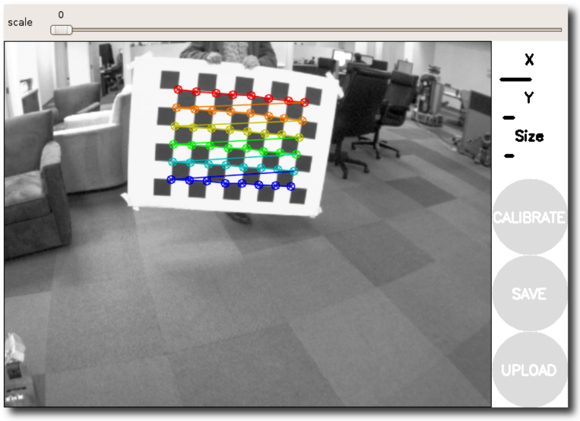


FIGURE 7.6: Camera calibration trough camera-calibration ROS package[46].

Most of the dataset produced has been obtained using this camera model. Only RGB images has been used.

Intel® RealSense™ Depth Camera D435i

It is one of the most used camera in robot application due to its performance, low cost and small size factor.It is designed for an simple integration, an easy to use setup and appriaceted for its portability. The *D435i* belongs to the *D400* Intel series and it is based on the *D4 Vision Processor*. It can performs an HD RGB image acquisition and depth image processing with no need of external computational power. It is equipped with a *Depth Module* able to work in indoor and outdoor environment with an high depth resolution and in long range.

FIGURE 7.7: IntelR©RealSenseTMDepth Camera D435i[47].

The parent model is the *D435*, but in this model an *Inertial Measurement Unit*(IMU) has been added . This upgrade allows to obtain better depth measurement and it is a starting point for SLAM and tracking application . In the end it can give-back a 6DoF information .

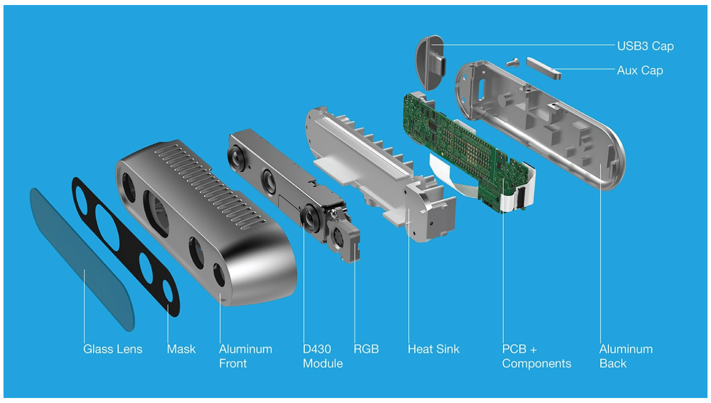


FIGURE 7.8: D435i internal modules details[48] .

It is composed of :

- RGB Module
- 2 Infrared Modules
- IR Projector

The RGB module designed for a resolution up to 1920x1080. The two infrared modules are sensors able to capture the infrared range acquiring infrared images. Combining those two infrared images the system virtualize a *depth camera* with a resolution of 1280x720 . The virtual camera is supposed to be locate in the actual left infared sensor . The IR Projector allows to improve the performance of the depth camera thanks to an *active stereo* method. In fact the passive stereo process drops in performance under poor light condition or low textured object, but thanks to the projector an IR pattern is emitted in order to increase the texture of the framed scene and simplifies the stereo-matching .
The application range can vary from 0.105 [m] to 10 [m], depending on lighting condition . The camera is equipped with its developers kit Intel RealSense SDK 2.0, but

in this thesis the *realsense2camera* has been used aiming to simplifies the connection with other software involved.

7.2.2 Board

The focus of this thesis is to reproduce a suitable object detection algorithm under some real space constraint.In designing a space flight only some validated on-board computers can be used. In fact the computers involved has to pass a space-qualification process to be admitted in the list of the possible computers. For this research two computers have been used: a workstation and *Nvidia Jetson Nano*.
The working station have been used only to be able to connect the Bumblebee camera , relying on its ExpressCard slot connector. Moreover the dataset acquisition and other initial setting test have been validated on this computer due to its flexibility .

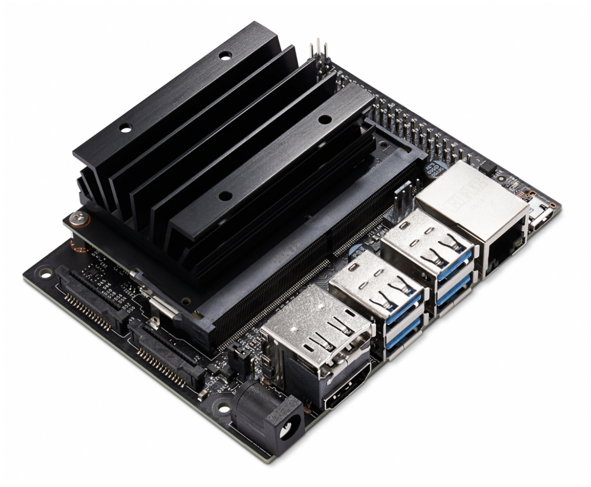
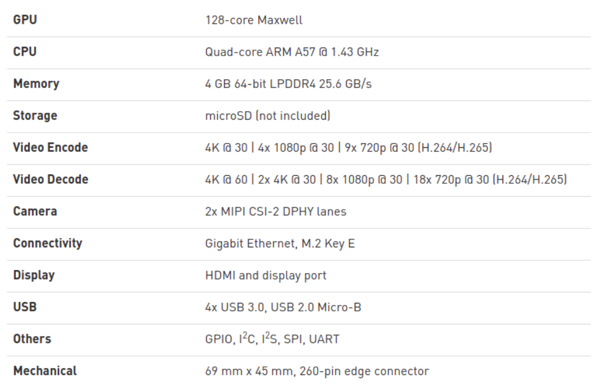


FIGURE 7.9: NVIDIA Jetson Nano[49].

The **NVIDIA Jetson Nano** belongs to the family of *NVIDIA Jetson*, an high performance and low power boards of able to run real-time artificial intelligence algorithm . They are perfect in robot application due to their small factor and easy integration into other system. In particular the Jetson Nano board is the smallest device in the Jetson family and needs only 10 [W] of power in high performace setting. It is based on ARM Cortex CPU and a 128 core Maxwell GPU. It is designed for deep learning application as object detection, image classification or segmentation. It can run multiple neural network in parallel.The board is provided in *NVIDIA® Jetson Nano™ Developer Kit* along with the NVIDIA JetPack SDK . It is equipped with four 3.1 USB ports permits the connectio with up to 4 cameras connected . Moreover it presents two possible power supply method with a barrel-jack or trough type-c connector in order to improve the integration. The board specifics are riported in the figure 7.10 :
For further details it can be useful the specifics website [50].



GPU	128-core Maxwell
CPU	Quad-core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (not included)
Video Encode	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I^2C, I^2S, SPI, UART
Mechanical	69 mm x 45 mm, 260-pin edge connector

FIGURE 7.10: NVIDIA Jetson Nano details[49].

Because of the camera energy consumption and other peripherals like keyboard and monitor it has been necessary to use a barrel-jack that can guarantee 5V and 4A . In fact due to peaks in the energy consumption during the run of different software, some lack of connection occurred between camera and board.

Chapter 8

Software and tools

In this thesis project Ubuntu has been used as operating system. It is free open-source operating system Based on Linux. In particular the version involved is the 18.04 LTS . Other software used will be explaining in the following section

8.1 ROS and packages

Robot Operating System is a free platform containing tool and libraries that simplifies the creation of robot application. Thanks to ROS libraries a large amount of work ,e.g camera initial setting is already done so as it is to connect camera and use pre-built basic task in order to create more complex application. It allows developers to focus on their research problem, while communication between different software is already solved in ROS .
The fundamental unit for managing software in ROS is the *package*. It may contains libraries, dataset or any other configuration file and they are all coherently organized together. Furthermore other important elements are present like *meta-package* or *repositories*, namely a collection of packages . For the purpose of communication in ROS there are some important elements :

- **Node**: process that run in order to perform some task. It is able to *subscribe* or *publish* to a *topic* .
- **Message**: it is a data structure used by nodes to communicate. In ROS primitive type are integrated , but a message can be composed of an arbitrary data structure .
- **Topic**: nodes exchange information through message. "The topic is a name that is used to identify the content of the message" . A node can take information subscribing to a topic or can send information publishing to a topic. The general framework is to decouple the information from their source or the consumption of them.

In the figure 8.1 is it possible to see the standard communication structure:

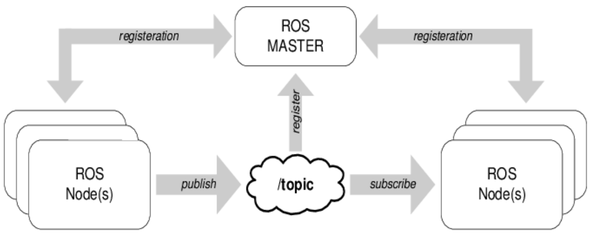


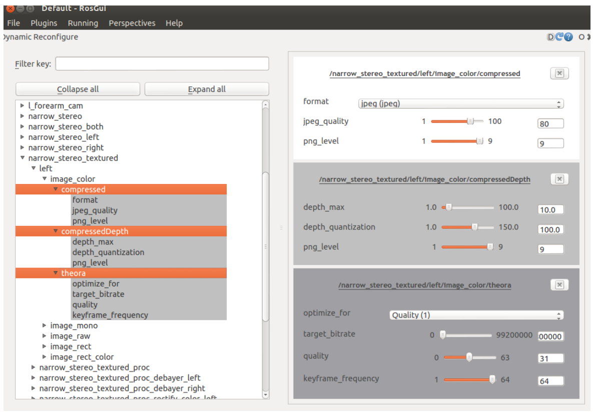
FIGURE 8.1: ROS basic communication structure[51].

Packages used

- **camera1394stereo** : it is used to communicate with stereo devices that exploits IEEE-1394 standard communication
- **bumblebeexb3** : it is a specific package able to communicate with Bumblebee XB3 camera and run camera functions
- **image_pipeline**: it contains fundamentals packages useful to manipulate raw image data
- **cameracalibration**: it allows calibration of cameras connected to ROS
- **vision_opencv**: package for interfacing ROS with OpenCV
- **librealsense2** : it is the libraries based on the official RealSense SDK 2.0
- **darknet_ros**: it is the package based on darknet release of YOLO

rqt_reconfigure

It is a GUI interface that is able to modify camera parameters by modifying parameters related to camera node. Thanks to this tools it has been possible to modify the initial calibration file and optic parameter of the Bumblebee camera.



```
▸ l_forearm_cam
▸ narrow_stereo
▸ narrow_stereo_both
▸ narrow_stereo_left
▸ narrow_stereo_right
▾ narrow_stereo_textured
  ▾ left
    ▾ image_color
      ▾ compressed
          format
          jpeg_quality
          png_level
      ▾ compressedDepth
          depth_max
          depth_quantization
          png_level
      ▾ theora
          optimize_for
          target_bitrate
          quality
          keyframe_frequency
    ▸ image_mono
    ▸ image_raw
    ▸ image_rect
    ▸ image_rect_color
  ▸ narrow_stereo_textured_proc
  ▸ narrow_stereo_textured_proc_debayer_left
  ▸ narrow_stereo_textured_proc_debayer_right
```

/narrow_stereo_textured/left/image_color/compressed

format	jpeg (jpeg)		
jpeg_quality	1	100	80
png_level	1	9	9

/narrow_stereo_textured/left/image_color/compressedDepth

depth_max	1.0	100.0	10.0
depth_quantization	1.0	150.0	100.0
png_level	1	9	9

/narrow_stereo_textured/left/image_color/theora

optimize_for	Quality (1)		
target_bitrate	0	99200000	00000
quality	0	63	31
keyframe_frequency	1	64	64

FIGURE 8.2: RQT reconfigure software[52].

8.2 Google Colab

It is a Google project called *Google Colaboratory* (Colab) designed for machine learning researchers and developers. It simplifies the researchers works thank to its on-line Jupyter Notebook environment that requires no setup. All the most popular packages are already installed . It is also equipped with pre-compiled machine learning models including a dedicated section to YOLO v3 . More over it can be connected to Drive account in order to save data . The most important feature of Google Colab is that it provides CPU,GPU or TPU computational power for free. This is crucial for machine learning application due to the computational cost of the algorithms. It has also some limitation concerning this free usage , in fact depending on the GPU personal usage or the total GPU power request it is possible that the system forces the notebook to switch on CPU mode.

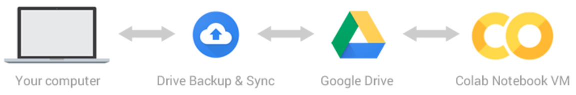


FIGURE 8.3: Google Colab drive communication structure[53].

Chapter 9

Implementation

The implementation of the thesis project has required the 3D printing of the sample target, the production of a proper dataset of images and the creation of the algorithm in order to find the 3D pose of the target.

9.1 Dataset

In order to create the dataset, several factors has been taking into account aiming to produce representative cases of a possible martian application. Moreover , also limits related of the object detection algorithm have been analysed.
The dataset has been produces using the bumblbee XB3 with a resolution of 1280x960 , after the calibration of the camera . Following the instruction related to the YOLO release involved in the thesis, the dataset should be composed of at least 2000 images per classes or more. In this case the dataset is composed of 5900 images splitted in *training-set* and*testing-set*, respectively 5000 and 900 images with a ration of 80%-20% between the two sub-set. The factors taking into account are:

- lighting condition
- occluded or covered by soil
- perspective angle
- partially or completely under shadow
- distance from the camera
- position into the image

For what concern the lighting condition a lux-meter has been used in order to define the lux quantity capturable by the camera. The lux level chosen are 30k-15K-8K-1K . The different distances have been chosen according to original mission goals from 5m to 0.5m.

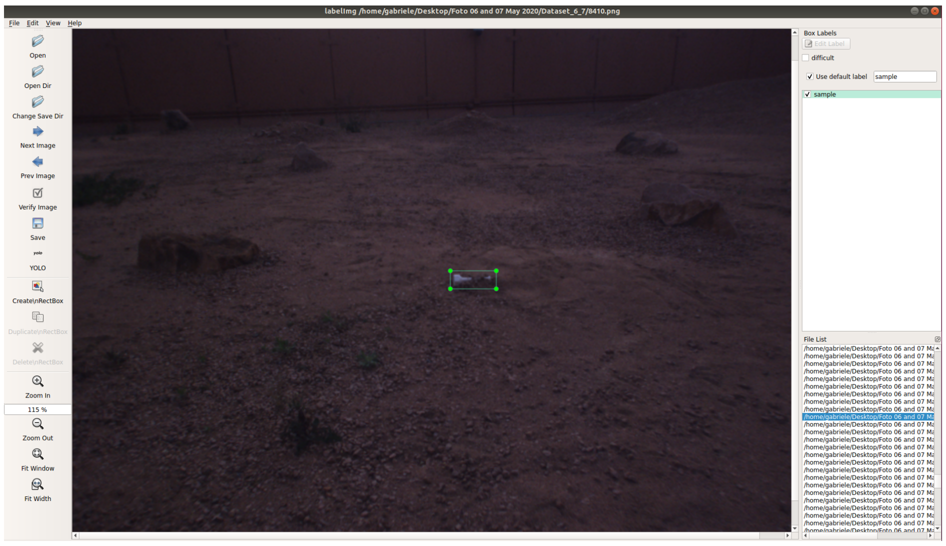


FIGURE 9.1: Labelling example on dataset image

All the other factors have been evaluated in qualitatively way during the capturing sessions as posing the target in different angles with respect to the camera .
It is really useful to produce in the dataset as many different images as possible aiming to improve the abstraction capcity of the algorithm to understand what is the object target with respect to what is not the object. The dataset should also include non-labelled images with negative examples, namely an object of the same shape or colour but that is not included in the detectable classes , aiming to improve the filters capacity to highlight peculiar features of the target object.

Labellimg

Once the images have been taken has to be labelled. It means drawing a bounding box called ground truth around the object that has to be detected , saving information of the bounding box position. In this project the *LabelImg* has been used . It saves the annotation for YOLO application as :

<object_class>   <x_center> <y_center> <width> <height>

Where:

- object_class is the class number, starting from 0. In this case there is only one class called "sample"
- (x_center , y_center) are the bounding box center coordinates normalized with width and height of the image. They are bounded between 0 and 1 .
- (width , height) are the bounding box sizes normalized with the width and height of the image .

In figure 9.1 is shown a labeling of a real captured image with LabelImg GUI . The related annotation was saved in a".txt" file :

0 0.537891 0.499479 0.061719 0.032292

9.2 Network modification and Training

In this thesis research the network has been trained through a *transfer learning* approach, namely using a pre-trained network weights. Usually it is a good approach to re-train the network , starting from an already trained network. In fact, some specific features that the trained network has learnt to detect could be useful in the new network too.Moreover it is suggested to follow this approach especially when the training dataset is small. In this case the dataset is large enough to learn from scratch, but because of the domain-specific dataset there is a possibility of overfitting. Furthermore, there is no disadvantage to use a pre-trained model. The training phase has been performed on Google Colab .
With the purpose to have a good compromise between the time cost and the computational cost , the YOLO models chosen are based on the YOLOv3-tiny version. It is a smaller version of YOLO with 23 layers and only two "yolo layer". It is useful when the hardware computational power is limited or when the speed factor is fundamental. Moreover, it shows extremely good results performing detection on small objects.
In order to adapt the configuration files to the specific dataset some adjustment has been made :

- *batch* : it is the set of images involved for each epoch.
- *subdivision*: it is the number of subset in which on bach is splitted in order to reduce the memory consumption
- *max_batches*: is the number of batches that have to be analysed. After the last batches the training is concluded. The suggested value is max_batches=2000*classes. More over it has to be at least 6000 and more than the number of testing images
- *step*: it is related to the *learning rate*. The more the training goes on, the more the learning rate should be reduce in order to optimize the training. Developer suggestion is to reduce the learning rate x0.1 at 80% and again at 90% of max_batches analysed .
- *filters*: it is related to the number of convolutional layer posed before the *yolo layer*. It has to be modified according to classes number . For example in YOLO-tiny it follow this rule "filters=(classes + 5)x3"
- *classes* : it is the number of classes that have to be detected

9.2.1 Custom Models

The chosen models are :

- *YOLOv3-tiny*
- *YOLOv3-tiny-3l* : it is based on the tiny version,but it presents a third extra yolo-layer. It is more accurate than the standard version, especially with small objects and it is more computational consuming

The network modifications are :

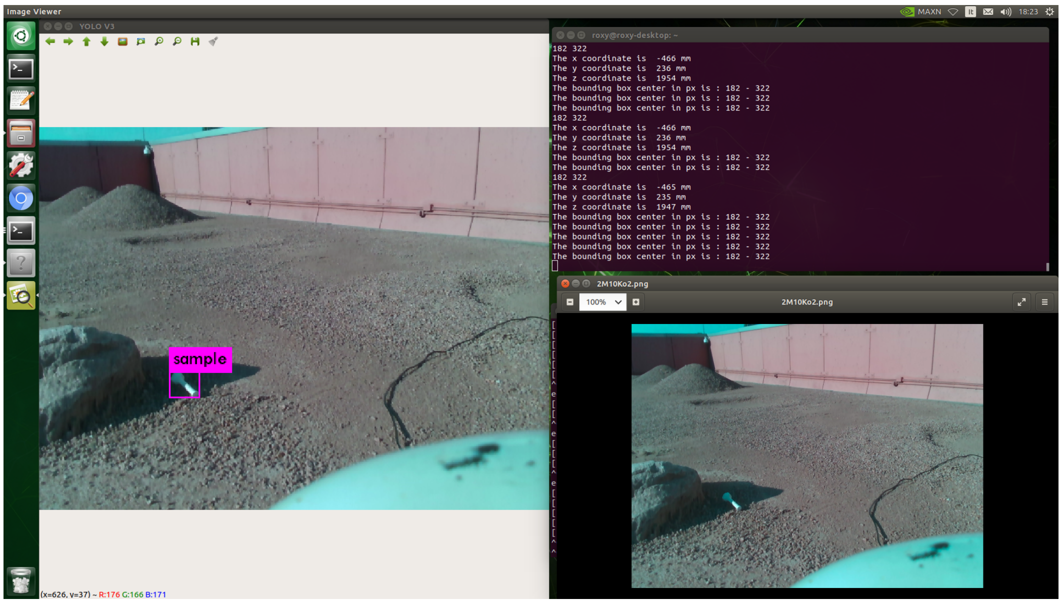
- batch = 64
- subdivision = 8
- max_batches = 6000
- step = 4800,5200
- filters = 18
- classes= 1

Another network parameters that has been modified is the *mask*. It represents the anchors size involved in the training. Even if they have been calculated for the COCO dataset trough the k-mean approach they can be re-adapted to the specific dataset.

9.3 3D coordinates estimation

The final scope of this thesis is to estimate the 3D pose of the sample. For this purposes the *rs_rgbd.launch* launcher has been used . It is contained is realsense-ros package. Its main purpose is to publish to the RGB image and the *coloured pointcloud* topics . The RGB image and the coloured pointcloud are aligned so that each (x,y) coordinates correspond to the same pixel. The coloured pointcloud message has the structure "XYZRGB" where the "XYZ" part correspond to the coordinates "of the pixel" with respect to the left infrared lens origin. The "RGB" part correspond to the standard RGB colours of the pixel.
Meanwhile the camera node publishes on its topics , the darknet node gets the RGB image information and perform detection . The results of this detection is published on the "bounding_bounding boxes" topics the information related to the bounding box coordinates and size. Moreover it publishes the information on the class detected and the related confidence.
A last node has been implemented in order to coordinates depth info from camera and detection infos from YOLO. The node subscribes to the bounding box topic and gets the coordinates in pixel. Then it analyses the pointcloud topic to find the "XYZ" part

of the center coordinates of the bounding box. At the end is scale the XYZ coordinates in millimiters and publish the bounding box center coordinates in pixels and the 3D pose if the center in millimiters.



```
182 322
The x coordinate is  -466 mm
The y coordinate is  236 mm
The z coordinate is  1954 mm
The bounding box center in px is : 182 - 322
The bounding box center in px is : 182 - 322
The bounding box center in px is : 182 - 322
182 322
The x coordinate is  -466 mm
The y coordinate is  236 mm
The z coordinate is  1954 mm
The bounding box center in px is : 182 - 322
The bounding box center in px is : 182 - 322
182 322
The x coordinate is  -465 mm
The y coordinate is  235 mm
The z coordinate is  1947 mm
The bounding box center in px is : 182 - 322
The bounding box center in px is : 182 - 322
The bounding box center in px is : 182 - 322
The bounding box center in px is : 182 - 322
The bounding box center in px is : 182 - 322
```

Figure 9.2: Performing detection in ROXY area

Chapter 10

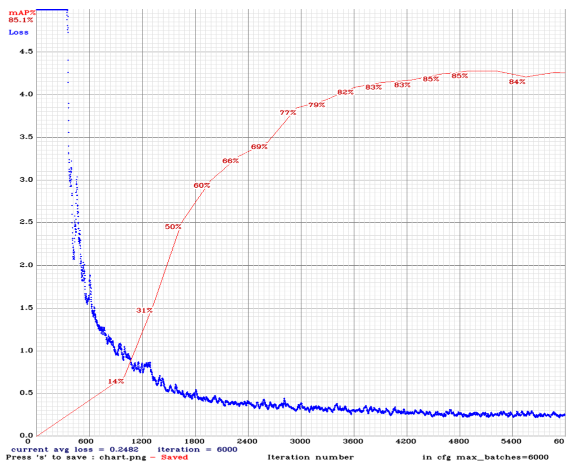
Results and conclusions

Results

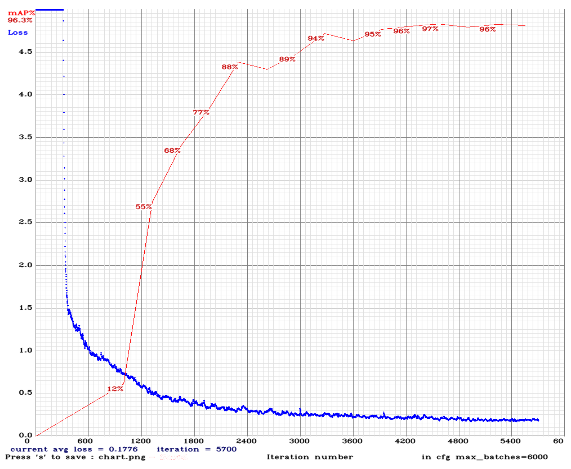
In the following table the main results have been reported in terms of average precision and average IoU with a threshold of IoU=0.5 . In order to avoid overfitting issues, the weights has been chosen analysing the mAP-Loss chart. The weights chosen for tiny and 3l tiny are those related respectively to epoch = 5000 and epoch = 5200 .

	YOLOv3 tiny	YOLOv3 tiny 3
AP	85.70 %	91.03 %
IoU@0.5	64.86 %	73.52 %
FPS	10	5

It is possible to notice that the larger network is more accurate in detection and accuracy in bounding box size. The third layer is very helpful , but due to its computational cost , the Frame per Second value drops drastically from 10 to 5 .

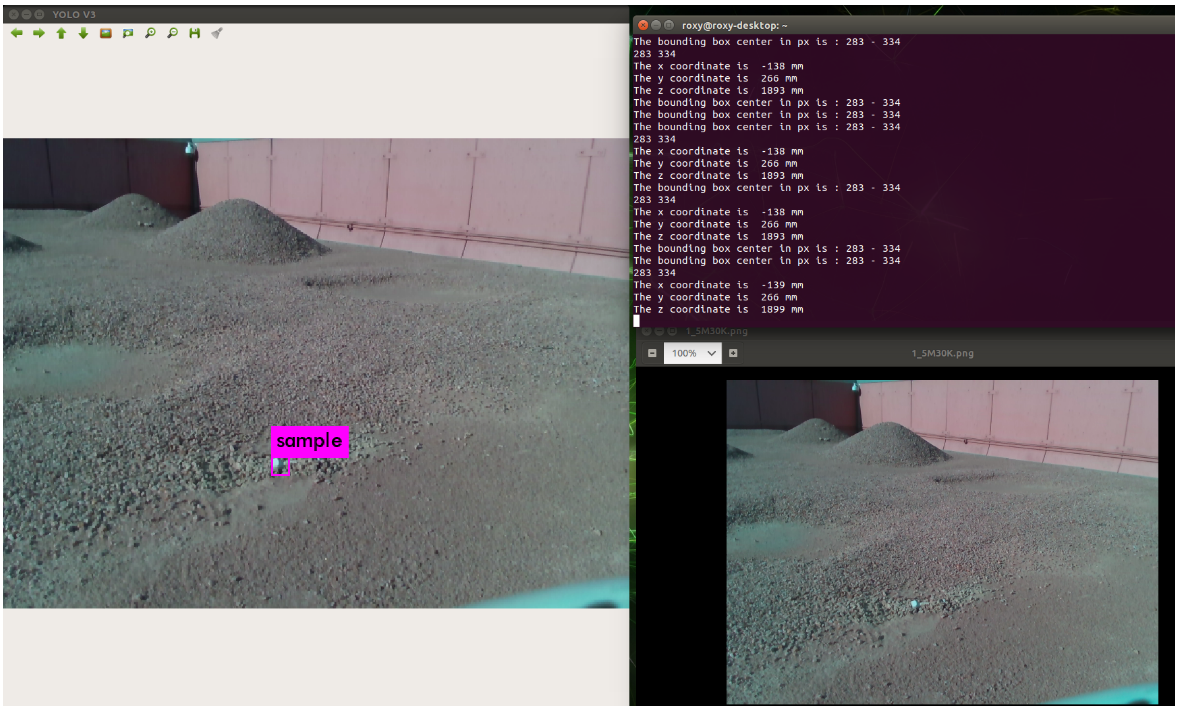


(a) mAP-Loss chart of the "YOLOv3 tiny" model training



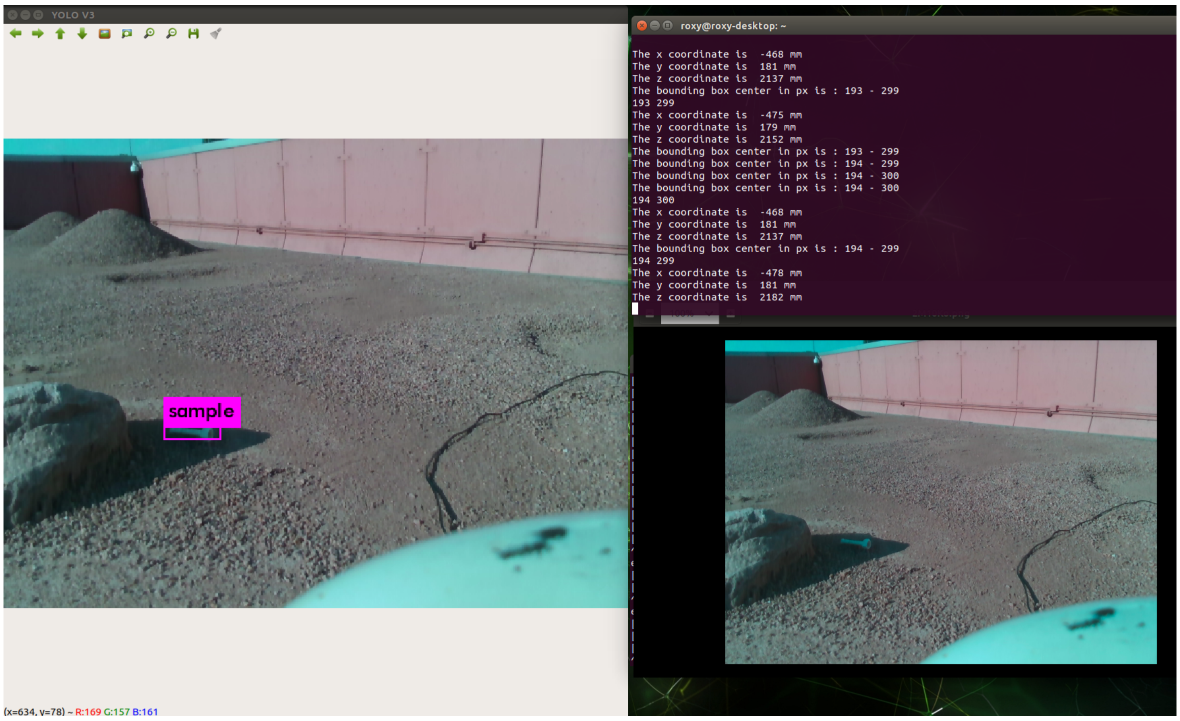
(b) mAP-Loss chart of the "YOLOv3 tiny 3l" model training

FIGURE 10.1



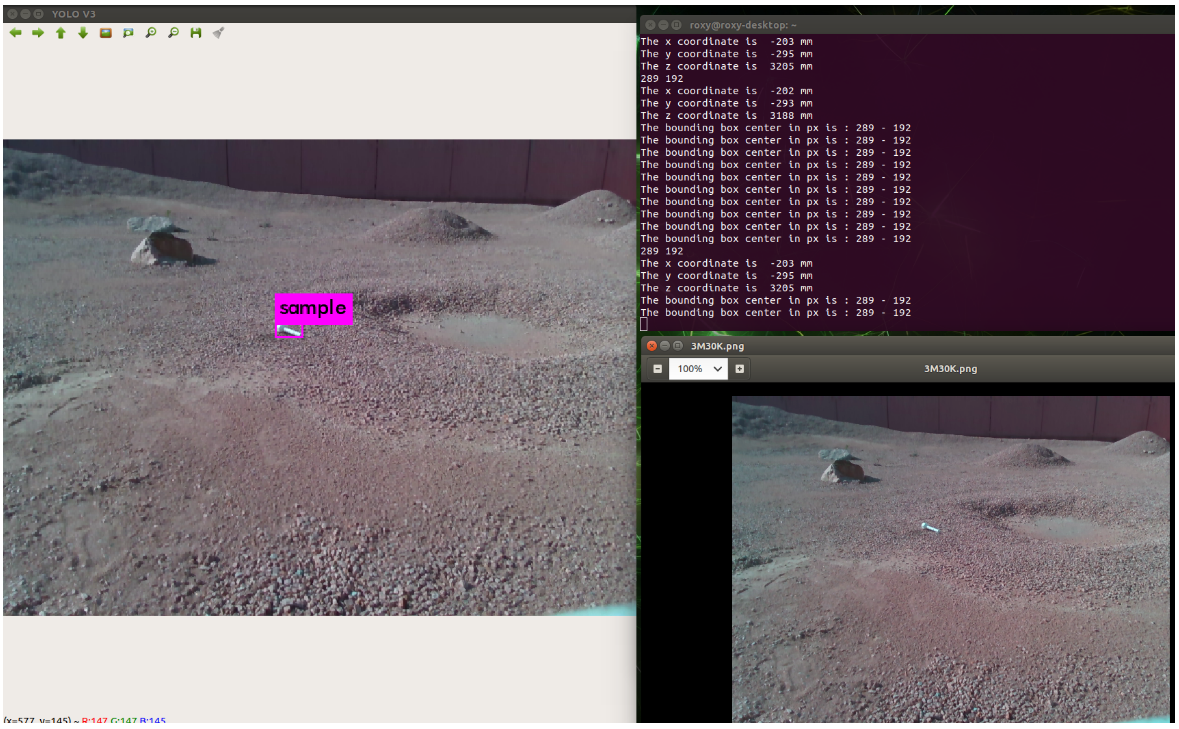
```
roxy@roxy-desktop: ~
The bounding box center in px is : 283 - 334
283 334
The x coordinate is  -138 mm
The y coordinate is  266 mm
The z coordinate is  1893 mm
The bounding box center in px is : 283 - 334
The bounding box center in px is : 283 - 334
The bounding box center in px is : 283 - 334
283 334
The x coordinate is  -138 mm
The y coordinate is  266 mm
The z coordinate is  1893 mm
The bounding box center in px is : 283 - 334
283 334
The x coordinate is  -138 mm
The y coordinate is  266 mm
The z coordinate is  1893 mm
The bounding box center in px is : 283 - 334
The bounding box center in px is : 283 - 334
283 334
The x coordinate is  -139 mm
The y coordinate is  266 mm
The z coordinate is  1899 mm
```

FIGURE 10.2: Detection performed at distance of 1.8 m, occluded by soil



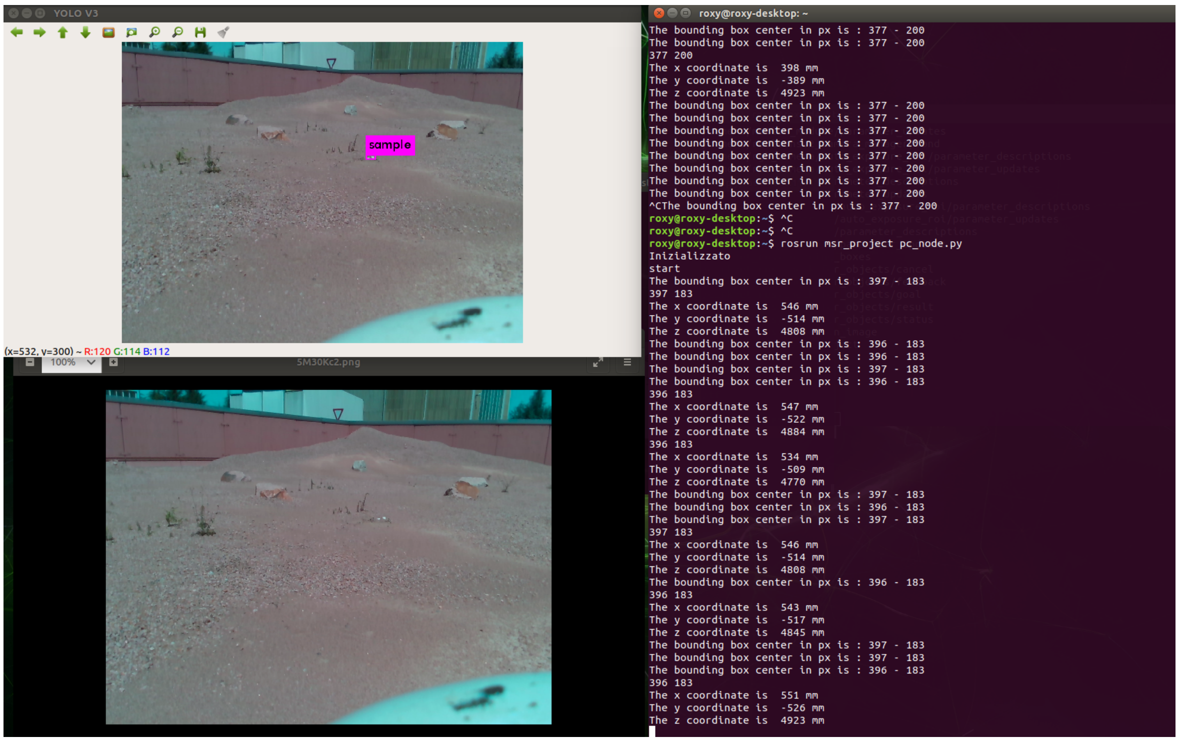
```
roxy@roxy-desktop: ~
The x coordinate is  -468 mm
The y coordinate is  181 mm
The z coordinate is  2137 mm
The bounding box center in px is : 193 - 299
193 299
The x coordinate is  -475 mm
The y coordinate is  179 mm
The z coordinate is  2152 mm
The bounding box center in px is : 193 - 299
The bounding box center in px is : 194 - 299
The bounding box center in px is : 194 - 300
The bounding box center in px is : 194 - 300
194 300
The x coordinate is  -468 mm
The y coordinate is  181 mm
The z coordinate is  2137 mm
The bounding box center in px is : 194 - 299
194 299
The x coordinate is  -478 mm
The y coordinate is  181 mm
The z coordinate is  2182 mm
```

FIGURE 10.3: Detection performed at distance of 2.2 m ,under a rock shadow



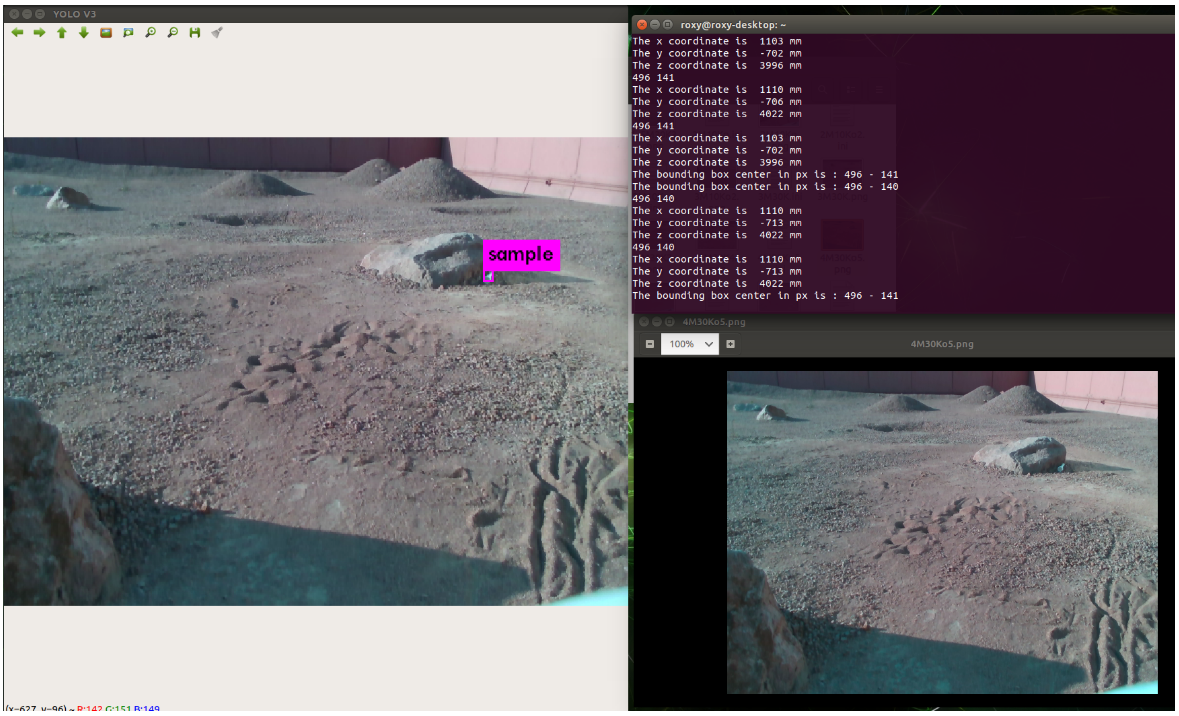
```
The x coordinate is  -203 mm
The y coordinate is  -295 mm
The z coordinate is  3205 mm
289 192
The x coordinate is  -202 mm
The y coordinate is  -293 mm
The z coordinate is  3188 mm
The bounding box center in px is : 289 - 192
The bounding box center in px is : 289 - 192
The bounding box center in px is : 289 - 192
The bounding box center in px is : 289 - 192
The bounding box center in px is : 289 - 192
The bounding box center in px is : 289 - 192
The bounding box center in px is : 289 - 192
The bounding box center in px is : 289 - 192
The bounding box center in px is : 289 - 192
The bounding box center in px is : 289 - 192
289 192
The x coordinate is  -203 mm
The y coordinate is  -295 mm
The z coordinate is  3205 mm
The bounding box center in px is : 289 - 192
The bounding box center in px is : 289 - 192
```

FIGURE 10.4: Detection performed at distance of 3 m



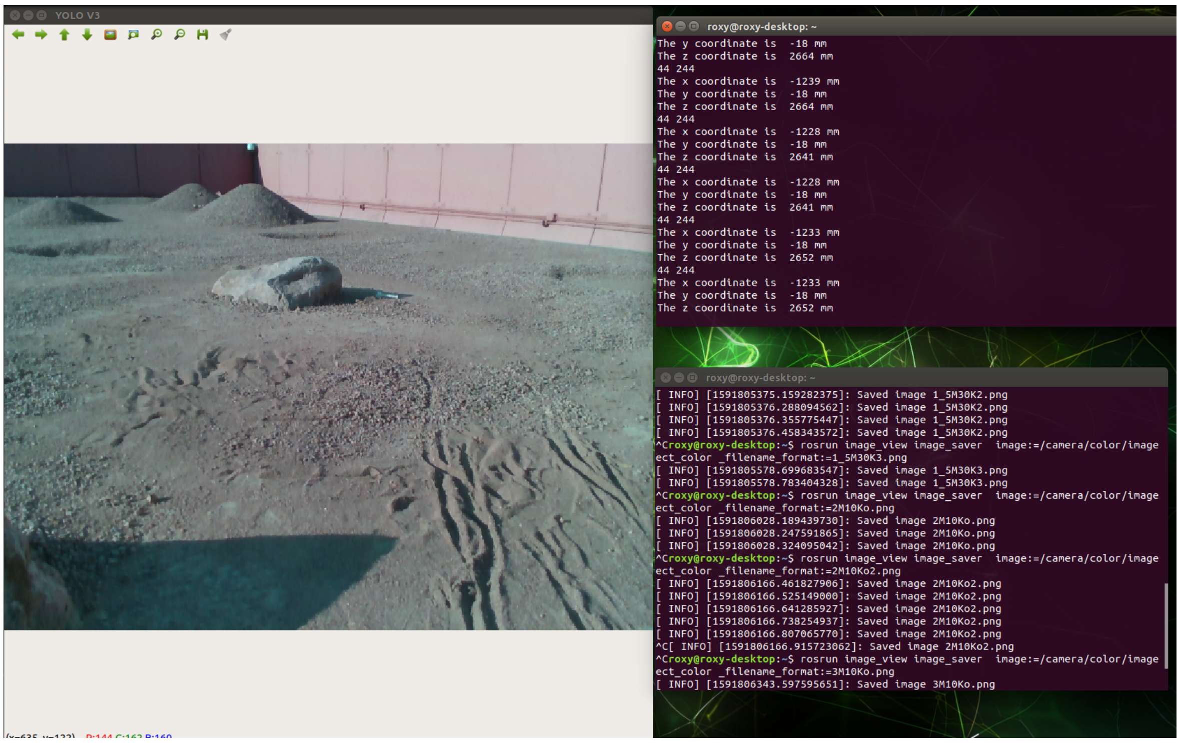
```
The bounding box center in px is : 377 - 200
The bounding box center in px is : 377 - 200
377 200
The x coordinate is  398 mm
The y coordinate is  -389 mm
The z coordinate is  4923 mm
The bounding box center in px is : 377 - 200
The bounding box center in px is : 377 - 200
The bounding box center in px is : 377 - 200
The bounding box center in px is : 377 - 200
The bounding box center in px is : 377 - 200
The bounding box center in px is : 377 - 200
The bounding box center in px is : 377 - 200
The bounding box center in px is : 377 - 200
^CThe bounding box center in px is : 377 - 200
roxy@roxy-desktop:~$ ^C
roxy@roxy-desktop:~$ ^C
roxy@roxy-desktop:~$ rosrun msr_project pc_node.py
Inizializzato
start
The bounding box center in px is : 397 - 183
397 183
The x coordinate is  546 mm
The y coordinate is  -514 mm
The z coordinate is  4808 mm
The bounding box center in px is : 396 - 183
The bounding box center in px is : 396 - 183
The bounding box center in px is : 397 - 183
The bounding box center in px is : 396 - 183
396 183
The x coordinate is  547 mm
The y coordinate is  -522 mm
The z coordinate is  4884 mm
396 183
The x coordinate is  534 mm
The y coordinate is  -509 mm
The z coordinate is  4770 mm
The bounding box center in px is : 397 - 183
The bounding box center in px is : 396 - 183
The bounding box center in px is : 397 - 183
397 183
The x coordinate is  546 mm
The y coordinate is  -514 mm
The z coordinate is  4808 mm
The bounding box center in px is : 396 - 183
396 183
The x coordinate is  543 mm
The y coordinate is  -517 mm
The z coordinate is  4845 mm
The bounding box center in px is : 397 - 183
The bounding box center in px is : 397 - 183
The bounding box center in px is : 396 - 183
396 183
The x coordinate is  551 mm
The y coordinate is  -526 mm
The z coordinate is  4923 mm
```

FIGURE 10.5: Detection performed at distance of 4.9 m



```
The x coordinate is  1103 mm
The y coordinate is  -702 mm
The z coordinate is  3996 mm
496 141
The x coordinate is  1110 mm
The y coordinate is  -706 mm
The z coordinate is  4022 mm
496 141
The x coordinate is  1103 mm
The y coordinate is  -702 mm
The z coordinate is  3996 mm
The bounding box center in px is : 496 - 141
The bounding box center in px is : 496 - 140
496 140
The x coordinate is  1110 mm
The y coordinate is  -713 mm
The z coordinate is  4022 mm
496 140
The x coordinate is  1110 mm
The y coordinate is  -713 mm
The z coordinate is  4022 mm
The bounding box center in px is : 496 - 141
```

FIGURE 10.6: Detection performed at distance of 4 m, under shadow



```
The y coordinate is  -18 mm
The z coordinate is  2664 mm
44 244
The x coordinate is  -1239 mm
The y coordinate is  -18 mm
The z coordinate is  2664 mm
44 244
The x coordinate is  -1228 mm
The y coordinate is  -18 mm
The z coordinate is  2641 mm
44 244
The x coordinate is  -1228 mm
The y coordinate is  -18 mm
The z coordinate is  2641 mm
44 244
The x coordinate is  -1233 mm
The y coordinate is  -18 mm
The z coordinate is  2652 mm
44 244
The x coordinate is  -1233 mm
The y coordinate is  -18 mm
The z coordinate is  2652 mm
```

```
[ INFO] [1591805375.159282375]: Saved image 1_5M30K2.png
[ INFO] [1591805376.288094562]: Saved image 1_5M30K2.png
[ INFO] [1591805376.355775447]: Saved image 1_5M30K2.png
[ INFO] [1591805376.458343572]: Saved image 1_5M30K2.png
^Croxy@roxy-desktop:~$ rosrun image_view image_saver  image:=/camera/color/image
ect_color _filename_format:=1_5M30K3.png
[ INFO] [1591805578.699683547]: Saved image 1_5M30K3.png
[ INFO] [1591805578.783404328]: Saved image 1_5M30K3.png
^Croxy@roxy-desktop:~$ rosrun image_view image_saver  image:=/camera/color/image
ect_color _filename_format:=2M10Ko.png
[ INFO] [1591806028.189439730]: Saved image 2M10Ko.png
[ INFO] [1591806028.247591865]: Saved image 2M10Ko.png
[ INFO] [1591806028.324095042]: Saved image 2M10Ko.png
^Croxy@roxy-desktop:~$ rosrun image_view image_saver  image:=/camera/color/image
ect_color _filename_format:=2M10Ko2.png
[ INFO] [1591806166.461827906]: Saved image 2M10Ko2.png
[ INFO] [1591806166.525149000]: Saved image 2M10Ko2.png
[ INFO] [1591806166.641285927]: Saved image 2M10Ko2.png
[ INFO] [1591806166.738254937]: Saved image 2M10Ko2.png
[ INFO] [1591806166.807065770]: Saved image 2M10Ko2.png
^C[ INFO] [1591806166.915723062]: Saved image 2M10Ko2.png
^Croxy@roxy-desktop:~$ rosrun image_view image_saver  image:=/camera/color/image
ect_color _filename_format:=3M10Ko.png
[ INFO] [1591806343.597595651]: Saved image 3M10Ko.png
```

FIGURE 10.7: False negative at a distance of 5 m, under shadow

Conclusions

The images confirms the good results on average precision computed on the dataset. The two networks perform well the more the system is close to the target. Moreover the algorithm shows some weak points due to false negative under shadow for large distances. The current FPS does not allow a real-time application, but this could be overcome involving a stand-alone software development, without the support of ROS platform.
Future development could involve the use of infrared images captured by the Intel RealSense D435i beside the RGB images, in order to augment the type of the dataset images and improve accuracy even in darker environment .

Bibliography

[1] NASA. *Mapping Sea Surface From the Space Station.* URL: https://www.nasa.gov/mission_pages/station/research/news/GEROS-ISS.

[2] NASA. *ULTRAVIOLET RADIATION ON THE SURFACE OF MARS.* URL: https://mars.nasa.gov/mgs/sci/fifthconf99/6128.pdf.

[3] NASA. *The Serpent Dust Devil of Mars.* URL: https://mars.nasa.gov/resources/3814/the-serpent-dust-devil-of-mars/?site=insight.

[4] Sergios Theodoridis and Konstantinos Koutroumbas. "Chapter 8 - Template Matching". In: *Pattern Recognition (Fourth Edition)*. Ed. by Sergios Theodoridis and Konstantinos Koutroumbas. Fourth Edition. Boston: Academic Press, 2009, pp. 481–519. ISBN: 978-1-59749-272-0. DOI: https://doi.org/10.1016/B978-1-59749-272-0.50010-4. URL: http://www.sciencedirect.com/science/article/pii/B9781597492720500104.

[5] Adaptive Vision. *Template Matching.* URL: https://docs.adaptive-vision.com/4.7/studio/machine_vision_guide/TemplateMatching.html.

[6] Richa Motwani. *Artificial Intelligence, Machine Learning, and Deep Learning.* URL: https://medium.com/@richamotwani/machine-learning-learning-tracker-day-1-b22a098e5467.

[7] Haohan Wang, Bhiksha Raj, and Eric P. Xing. "On the Origin of Deep Learning". In: *CoRR* abs/1702.07800 (2017). arXiv: 1702.07800. URL: http://arxiv.org/abs/1702.07800.

[8] The University of Queensland. *History of Artificial Intelligence.* URL: https://qbi.uq.edu.au/brain/intelligent-machines/history-artificial-intelligence.

[9] A. M. TURING. "I.—COMPUTING MACHINERY AND INTELLIGENCE". In: *Mind* LIX.236 (Oct. 1950), pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433.

[10] Frank ROSENBLATT. "The Perceptron: A Perceiving and Recognizing Automaton". In: (1957).

[11] Marvin Minsky and Seymour Papert. *Perceptrons*. Vol. 84. Dec. 1969. ISBN: 0262631113. DOI: 10.2307/1420478.

[12] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. “A Fast Learning Algorithm for Deep Belief Nets”. In: *Neural Comput.* 18.7 (July 2006), pp. 1527–1554. ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527. URL: https://doi.org/10.1162/neco.2006.18.7.1527.

[13] Mr. Kuldeep Singh Mr. Parveen Kumar. *Simulated Characteristic Model of Artificial Neuron in VHDL*. URL: hhttp://www.ijeert.org/pdf/v2-i3/25.pdf.

[14] *Mcculloch pitts neuron model*. URL: https://www.ques10.com/p/13297/what-is-mcculloch-pitts-neuron-model-with-the-he-1/.

[15] Will Traves. *An Introduction to Machine Learning*. URL: https://www.usna.edu/Users/math/traves/presentations/BN2018.pdf.

[16] Ava Soleimany. *Deep Learning for Computer Vision MIT 6.S191*. URL: http://introtodeeplearning.com/2018/materials/2018_6S191_Lecture3.pdf.

[17] Ulas Bagci. *-Filtering and Edges*. URL: http://www.cs.ucf.edu/~bagci/teaching/robotvision18/Lec4.pdf.

[18] Arden Dertat. *Applied Deep Learning - Part 4: Convolutional Neural Networks*. URL: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

[19] Arden Dertat. *Applied Deep Learning - Part 4: Convolutional Neural Networks*. URL: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

[20] Charlotte Cullip. *A Comparison of CNN Architectures (Part 2)*. URL: https://medium.com/@charlottecullip/a-comparison-of-cnn-architectures-part-2-8d03c67d8ec6.

[21] Shaunak Halbe. *Object Detection and Instance Segmentation: A detailed overview*. URL: https://medium.com/swlh/object-detection-and-instance-segmentation-a-detailed-overview-94ca109274f2.

[22] *Object detection using Fast R-CNN*. URL: https://docs.microsoft.com/en-us/cognitive-toolkit/object-detection-using-fast-r-cnn.

[23] OpenCV. *Cascade Classifier*. URL: https://docs.opencv.org/3.4.9/db/d28/tutorial_cascade_classifier.html.

[24] N. Dalal and B. Triggs. “Histograms of Oriented Gradients for Human Detection”. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* 1 (2005), pp. 886–893. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1467360.

[25] *Histogram of Oriented Gradients (HOG)*. URL: https://www-users.cs.umn.edu/~hspark/csci5561_S2019/hw1.pdf.

[26] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation.* 2013. arXiv: `1311.2524 [cs.CV]`.

[27] *Selective Search for Object Recognition.* URL: `http://www.huppelen.nl/publications/selectiveSearchDraft.pdf`.

[28] Jonathan Hui. *What do we learn from region based object detectors (Faster R-CNN, R-FCN, FPN)?* URL: `https://medium.com/@jonathan_hui/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9`.

[29] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *Lecture Notes in Computer Science* (2016), pp. 21–37. ISSN: 1611-3349. DOI: `10.1007/978-3-319-46448-0_2`. URL: `http://dx.doi.org/10.1007/978-3-319-46448-0_2`.

[30] *Selective Search for Object Recognition.* URL: `http://www.huppelen.nl/publications/selectiveSearchDraft.pdf`.

[31] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection.* 2015. arXiv: `1506.02640 [cs.CV]`.

[32] Jonathan Hui. *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3.* URL: `https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088`.

[33] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger.* 2016. arXiv: `1612.08242 [cs.CV]`.

[34] *Real-time object detection with YOLO.* URL: `https://machinethink.net/blog/object-detection-with-yolo/`.

[35] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement.* 2018. arXiv: `1804.02767 [cs.CV]`.

[36] Adrian Rosebrock. *Intersection over Union (IoU) for object detection.* URL: `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`.

[37] *Accuracy, Recall & Precision.* URL: `https://medium.com/@erika.dauria/accuracy-recall-precision-80a5b6cbd28d`.

[38] Jonathan Hui. *mAP (mean Average Precision) for Object Detection.* URL: `https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173`.

[39] Manal El Aidouni. *Evaluating Object Detection Models: Guide to Performance Metrics.* URL: `https://manalelaidouni.github.io/manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html`.

[40] M. Veloso J. Cartucho R. Ventura. "Robust Object Recognition Through Symbiotic Deep Learning In Mobile Robots". In: ().

[41] NASA. *Space Radiation Effects on Electronic Components in Low-Earth Orbit.* URL: https://llis.nasa.gov/lesson/824.

[42] Andrea Fusiello. *Stereo Matching: an Overview.* URL: http://www.diegm.uniud.it/fusiello/teaching/mvg/stereo.pdf.

[43] OpenCV. *Depth Map from Stereo Images.* URL: https://docs.opencv.org/master/dd/d53/tutorial_py_depthmap.html.

[44] FLIR. *Bumblbee datasheet.* URL: https://flir.app.boxcn.net/s/5v1q4t8ko0slv7u3pqez2ei
file/416892704437.

[45] FLIR. *Bumblebee® XB3 FireWire Details.* URL: https://www.flir.com/support/products/bumblebee-xb3-firewire#Overview.

[46] *How to Calibrate a Stereo Camera.* URL: http://wiki.ros.org/camera_calibration/Tutorials/StereoCalibration.

[47] *Intel® RealSense™ Depth Camera D435i.* URL: https://www.intelrealsense.com/depth-camera-d435i/.

[48] URL: https://support.intelrealsense.com/hc/en-us/community/posts/360034849354-About-the-structure-of-RealSense-Depth-Camera-D435i.

[49] *Jetson Nano Developer Kit.* URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[50] NVIDIA. *Jetson Nano Developer Kit.* URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[51] M.S. Achmad et al. "Tele-Operated Mobile Robot for 3D Visual Inspection Utilizing Distributed Operating System Platform". In: *International Journal of Vehicle Structures and Systems* 9 (Sept. 2017). DOI: 10.4273/ijvss.9.3.12.

[52] URL: http://wiki.ros.org/rqt_reconfigure.

[53] *How to train YOLOv3 using Darknet on Colab notebook and optimize the VM runtime load times.* URL: https://colab.research.google.com/drive/1lTGZsfMaGUpBG4inDIQwIJVW476ibXk_#scrollTo=Cqo1gtPX6BXO.

Appendix

```
#!/usr/bin/env python

import rospy
import threading

from std_msgs.msg import *
from darknet_ros_msgs.msg import *
import sensor_msgs.point_cloud2 as pc2
from sensor_msgs.msg import PointCloud2, PointField


class INIT():
        #Values Initialization
        def __init__(self):
                rospy.init_node('estimation_node',anonymous=True)
                global center_x, center_y, scale_factor

                scale_factor=1000
                center_x,center_y,u,v = 0,0,0,0

                print("Starting Pose Estimation")


class INPUT(threading.Thread):

        # Initialization of subscribing elements
        def __init__(self,bounding_boxes =
        ↪ '/darknet_ros/bounding_boxes',
        ↪ point_cloud="/camera/depth_registered/points"):
                threading.Thread.__init__(self)
                self.bounding_boxes = bounding_boxes
                self.point_cloud = point_cloud


        # Subscribing to bounding box and point cloud topics
        def run(self):
                self.bounding_boxes_sub =
                ↪   rospy.Subscriber(self.bounding_boxes,
                ↪   BoundingBoxes, self.bbox)
                self.pc_prova_sub = rospy.Subscriber(self.point_cloud,
                ↪   PointCloud2, self.pc_prova)

        # Reading bounding box coordinates and related center
        def bbox(self, data):

                global center_x,center_y

                values = data.bounding_boxes
                for data in values:

                        bb_xmin = data.xmin
                        bb_ymin = data.ymin
                        bb_xmax = data.xmax
                        bb_ymax = data.ymax

                        w = bb_xmax - bb_xmin
                        h = bb_ymax - bb_ymin
                        center_x = bb_xmin + w/2
                        center_y = bb_ymin + h/2


        # PointCloud2 message is saved in "point_saved" . The
        ↪   coordinates are scaled in mm and published
        def pose_estimation(self,data):

                #u : width
                #v : height

                u=center_x
                v=center_y

                point_saved = pc2.read_points(data, field_names = ("x",
                ↪   "y", "z"), skip_nans=True, uvs=[[u,v]])
                point=next(point_saved, None)

```

```python
                pt_x = int(point[0]*scale_factor)
                pt_y = int(point[1]*scale_factor)
                pt_z = int(point[2]*scale_factor)

                print ("The bounding box center (x,y) in px is : {} - {}".format(u , v) )
                print("The x coordinate is  %d mm " %pt_x )
                print("The y coordinate is  %d mm " %pt_y )
                print("The z coordinate is  %d mm " %pt_z )



if __name__ == '__main__':

        init = INIT()

        i = INPUT()
        i.start()

        rospy.spin()
```