POLITECNICO
DI TORINO

Master of Science Degree in ICT for Smart Societies

OBJECT DETECTION WITH NON-TRADITIONAL SENSORS

Supervisor:

Prof. Marco Piras

Co-supervisors:

Prof. Marcello Chiaberge

Dott. Angelo Tartaglia

Dott. Vittorio Mazzia

Candidate:

Alice Tumiatti

July 2019

Abstract

Today, the navigation in known places is quite easily reached using several solutions as GNSS and visual odometry but when the space is unknown, the navigation is more critical: no reference, no maps. How to do the navigation under this condition? One of the challenges in the Smart Societies this kind of navigation in outdoor and indoor condition, even using UGV (Unmanned Ground Vehicle) system. ICT could have an important role in this domain, considering the competences on the technologies and innovative solutions. These devices can autonomously acquire important data, but they need to use some particular tools and algorithms devoted to investigate and analyze the space. Typically, expensive sensors and robust computer are required to solving the navigation in this complex condition.

The goal of this thesis is to test not-conventional sensors, as low cost systems, mass market solution and IR camera for object recognition in order to support navigation in indoor environments, implementing the algorithms on low cost platform, such as a Raspberry Pi.

With a smartphone, photos of an extinguisher were taken, and the training sets were created, one for each algorithm. Once trainings were done, performances of the chosen algorithms were evaluated on sample images taken with four sensors: Smartphone, Official Pi Camera, Longruner Camera, MAPIR Survey3.

In term of algorithms, the navigation is partially supported by the Object Detection (OD) which has a significant practical importance and it is used across a variety of fields as: autonomous vehicles, workplace automation and surveillance.
The use of autonomous surveillance systems is increasingly common, moreover a UGV could be useful in dangerous situations to identify emergency exits or useful objects such as the extinguisher.

There are many models available and, in this thesis, Haar Cascade (HC) and YOLO have been compared. The tools used are: OpenCV to implement HC and Darknet to implement YOLO.

Several tests have been done and the workflow can be summarized by these steps:

1. Gathering training data
2. Training the model
3. Prediction on new images

4. mAP (*mean Average Precision*) evaluation

OD suffers from the complexity of creating the training set, because for a good training a huge number of images is needed.

The training of both algorithms was done on the images taken with the smartphone, to evaluate whether the sets of images, already present on the web, can be used for the detection on images acquired with different sensors, as IR cameras.

Since low cost sensors have been used, the calibration tools of MATLAB and OpenCV have been exploited. The mAP achieved by the algorithms have been evaluated on images with and without distortions.

The mAP of HC is lower than the mAP of YOLO, that is not able to identify the extinguisher in the datasets of the MAPIR and the night vision cameras. HC reaches lower mAP compared to the one gained on the smartphone and Official Pi Camera images, but still manages the detection.
This greater versatility can be due to the use of grayscale images during training and testing, while YOLO works on RAW images and so has trouble recognizing the object on images different from those used during training.

Furthermore, since the application to be obtained should work in real time, it has been tested how descriptors, SIFT and SURF, combined with the RANSAC algorithm, can speed up the detection on videos.

However, the results obtained are promising and with the improvement of ICT, the application studied will be more efficient.

Acknowledgements

Inizio con il ringraziare i miei genitori che mi hanno sempre sostenuta e incoraggiata. Ringrazio mamma perché è il mio modello e la mia forza, ringrazio papà perché crede nelle mie capacità e non mi permette di mollare mai.
Ringrazio anche le mie sorelle che hanno reso le sessioni d'esame un po' più leggere.

Un grazie speciale poi, lo voglio dedicare al mio fidanzato, che è sempre stato al mio fianco e non ha mai criticato le mie scelte.

Ringrazio anche tutta la sua famiglia per avermi fatto sentire sempre amata e per essere diventata un mio punto di riferimento.

Un altro grazie, lo dedico agli amici vecchi e nuovi per le risate, le parole di conforto e tutti i momenti condivisi.

Ringrazio anche il professor Marco Piras per avermi seguita e guidata fino alla fine.

Un grazie, lo dedico anche a tutto il dipartimento DIATI e al PIC4Ser, che con la loro disponibilità e gentilezza mi hanno fatta sentire parte del gruppo.

GRAZIE A TUTTI!

"*The beginning of knowledge*
is the discovery of something we do not understand."
- Frank Herbert -

List of Figures

List of Tables

Index

Nomenclature

GNSS Global Navigation Satellite System

UGV Unmanned Ground Vehicle

ICT Information Communication Technology

CPU Central Processing Unit

GPU Graphics Processing Unit

IR InfraRed

OD Object Detection

HC Haar Cascade

Y.O.L.O. You Only Look Once

NN Neural Network

ANN Artificial Neural Network

FNN Feedforward Neural Network

CNN Convolutional Neural Network

RNN Recurrent Neural Network

GAN Generative Adversarial Network

GD Gradient Descent algorithm

SGD Stochastic Gradient Descent algorithm

BP BackPropagation algorithm

R-CNN Regional Convolutional Neural Network

RoI Region of Interest

TP True Positive

FP False Positive

TN True Negative

FN False Negative

IoU Intersection over Union

AP Average Precision

mAP mean Average Precision

SIFT Scale-Invariant Feature Transform

SURF Speeded-Up Robust Features

RANSAC RANdom Sample Consensus

RGB Red-Green-Blue

RGN Red-Green-NearIR

NV Night Vision

Introduction

Nowadays, UGV (Unmanned Ground Vehicle) are used in a wide variety of different situations and applications, as for example in the fields of civil and military industry. An UGV can be used by emergency services such as fire brigades, ambulances, police, and others. It can also provide a great support for a wide variety of tasks as: harvesting, transporting, detection, investigation, exploration and inspection at tunnels, buildings and others. In addition, UGV can be also used reconstruct the navigated environment.

UGV is a vehicle that navigates autonomously, retrieves data and is suitable for any situation. Navigation is general and includes various scenarios: indoor or outdoor environments, known or unknown environments, dangerous due to toxic substances or others and all their combination. Todays, outdoor navigation is the most developed because it can exploit GPS localization, while indoor or unknown environments navigation, are more difficult and obstacle recognition techniques, to avoid collisions, can be very useful. ICT technologies are very useful in supporting navigation not only for the recognition of obstacles, but also for the object detection task, in fact once the UGV identifies an object it can perform some actions like taking it to transport it or mark its position on a map and others.

UGV are built and integrated with sensors, drive mechanisms, computers and software.

The sensors are devices that allow the collection of data and can be grouped into different categories based on their function. Some categories are: vehicle sensors, location sensors, vision sensors, obstacle detectors, orientation and communication sensors.

The integration of the various sensors makes a synergistic intelligent system that should work well in various environment. Computing all sensors data will improve interpretation and control of the vehicle in any dynamic situation. However, the sensors selection is important and specify for each application and goal.

Drive-by-wire technology is mature and reaches complete control of the movement through switch operations and other accessories. At a distance driving can be done with a guide on board and RF technology, then using Light detection and ranging (Lidar) technology, the obstacle assessment should be obtained [B. Rohini et al. 2008].

To navigate the UGV must first localize itself, so it must be able compute or update its position through information gathered from sensors. The vehicle should achieve localization in its operational environment for path planning and navigation algorithms to work effectively.

Borenstein states that since there is no complete solution to the positioning problem, developers of mobile robots usually combine two methods for position measurements. For example, a vision system is rarely used alone for navigation; it is usually used in combination with laser rangefinders and/or ultrasonic sensors for distance measurements [J. Borenstein, et al. 1997]

Moreover, another project of considerable importance and actuality is the autonomous driving, which would not be possible without the recognition of pedestrians, vehicles etc.

In addition, the detection and tracking of objects are the basis of lot of applications in the sector of surveillance and activities' recognition. Indoor positioning systems are used to locate people or required objects in large buildings and closed areas. For example, locating patients in the hospital, finding people trapped in a burning building or finding workers in a large office block are a few applications of indoor positioning systems.

Using the concept of machine learning, a car can be automated (self-driving) [A. Geiger et al. 2013]. It should be trained with specific images and whenever it detects the trained images, it operates according to the trained instruction.

On a UGV the microcontroller can be a low cost solution (as a Raspberry Pi) and then additional sensors like pi camera and laser can be used to detect objects and obstacles.

Traditional objects' detection methods are built on handcrafted features and shallow trainable architectures. Then, with the improvement in deep learning field, more powerful tools, which can learn semantic, high-level, deeper features, are introduced to address the problems existing in traditional architectures [Y. LeCun et al. 2013].

These models behave differently in network architecture, training strategy and optimization function.

The objects' detection task usually consists of different subtasks such as: face detection and pedestrian detection. This task is also able to provide useful information for semantic understanding of images and videos, and it is related to many applications, including image classification, human behaviour analysis and face recognition [P. Viola et al. 2001].

However, due to large variations in viewpoints, poses, occlusions and lighting conditions, it's difficult to perfectly accomplish objects' detection and localization tasks.

The problem definition of object detection is to determine where objects are in a given image (object localization) and which category each object belongs to (object classification). The pipeline of traditional object detection models can be mainly divided into three stages:

1. informative region selection
2. feature extraction
3. classification

In the first stage, the image is scanned with multi-scale sliding window since different objects may appear in different positions of the image and with different dimensions. Even if this exhaustive strategy can find out all possible positions of the objects, its drawbacks are also evident. Due to many candidate windows, it is computationally expensive and produces many redundant windows. But if only a fixed number of sliding window templates are applied, unsatisfactory regions may be produced.

In the second stage, visual features need to be extracted in order to recognize different objects. It is important that the features provide a robust representation of the objects. Because due to the diversity of appearances, illumination conditions and backgrounds, it is difficult to design a robust feature descriptor that perfectly describes all kinds of objects.

In the third and last stages, a classifier is needed to distinguish the target object from all the others and to make the representations more hierarchical, semantic and informative for visual recognition.

The objectives of this work are: studying the performance of low cost systems in the Object detection field, comparing the performance of two different Object Detection model, understanding how the training images and settings influence the performance, if the calibration improve the performance. All these questions were analyzed in the case of indoor navigation of an unknown environment in emergency situation.

Below the description of the thesis' structure:

- in this first part, a general view of the problem faced in this thesis and of the state of the art on the subject has been given;
- the first chapter is dedicated to a formal description of what machine and deep learning are. The basics' concept to understand how neural networks work and the main problems that affects their usage are described;
- the second chapter concerns a theoretical description of the techniques analysed in this thesis. These techniques are related to objects' detection in different scenarios.
- In the third chapter, there are the descriptions of: feature detection and matching techniques, and the RANSAC algorithm;
- the fourth chapter concerns the metric used to evaluate the performance of the detection algorithms;

- in the fifth chapter, there is the description of the sensors used in this thesis;
- the sixth chapter concerns the geometric calibration that has been done on the sensors of the previous chapter;
- the seventh chapter contains all the trainings made;
- the eighth chapter is dedicated to the description of the tools used to conduct the trainings and testing of the detection algorithms;
- the ninth chapter contains the description of the tests done and the obtained results;
- in the last part, conclusions and several promising directions are provided to serve as guidelines for future work in both objects' detection and relevant neural network based learning systems.

1. Introduction of Machine and Deep learning

Machine and Deep learning are fields of study that give computers the ability to learn without being explicitly programmed [A. Samuel. 1959].

A computational model, for neural networks based on mathematics and algorithms called threshold logic, has been created by Warren McCulloch and Walter Pitts in 1943. The model opened the way for neural network research to two approaches: one focused on biological processes in the brain and the other focused on the application of neural networks to artificial intelligence.

1.1 Neural Network basics' concepts

Neural Networks (NNs) are computing systems inspired by, but not necessarily identical to, the biological neural networks that constitute animal brain. The systems "learn" how to perform tasks by considering samples. An example is image recognition, in which a NN learns how to identify images that contain dogs by analysing sample images that have been manually labelled as "dog" or "no dog" and using the results to identify dogs in new images. NNs do this without any prior knowledge about dogs, NNs automatically generate features from the learning material that they process. The NN learning material is composed by training and testing datasets. The learning process includes both the training and testing phases of the network.

From the mathematical point of view, the NN implements a *huge* multidimensional nonlinear function.

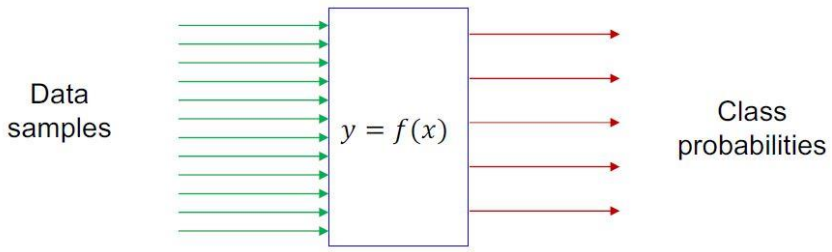


Figure 1.1: example of NN representation

The basic element of a NN is the perceptron, which weights different factors to make a decision. NNs are used to: define a model, use training examples to fit the model and determine decision rule. But the real problem is how to define this model, because it is difficult to write a computer program that "describes" the data.

The perceptron is defined as $x \cdot w = \sum_j x_j w_j$. Then a perceptron "fires" when the output becomes 1. The output is defined by the comparison with a threshold *b,* called bias, as shown in *Figure 1.2.*

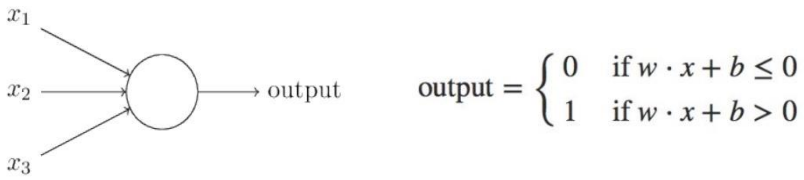


Figure 1.2: NN perceptron

More complex decisions are made by NNs composed by multiple hidden layers, an example is shown in *Figure 1.3*. The hidden levels are all those included between the input layer and the output layer.

There are different types of NNs:

1. Feedforward Neural Network (FNN)
2. Convolutional Neural Network (CNN)
3. Recurrent Neural Network (RNN)
4. Generative Adversarial Network (GAN)

An FNN is characterized by connections in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes [A. Zell. 1994]. FNN was the first and simplest type of NN devised. In this network, the information goes from the input nodes, through the hidden nodes (if any) and to the output nodes. In FNN network, the information goes from the input nodes, through the hidden nodes (if any) and to the output nodes. An FNN is called *deep* if it has many hidden layers (up to hundreds).

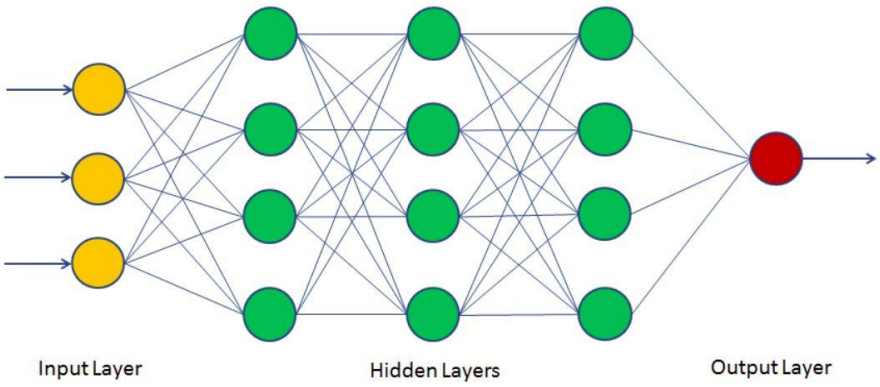


Figure 1.3: Feedforward NN with multiple hidden layers

The second type of NNs are CNNs that "share" weights, an example in *Figure 1.4* (for more details, see §1.5).

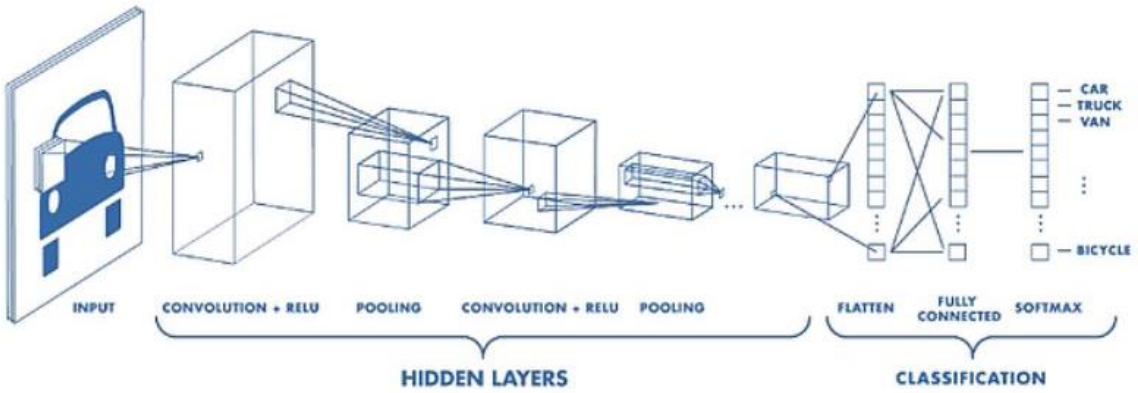


Figure 1.4: CNN structure example

In an RNN (*Figure 1.5*) the output values of a layer of a higher level are used as an input to a lower layer.

This interconnection between layers allows the use of one of the layers as state memory, and allows, by providing an input time sequence of values, to model a dynamic temporal behavior dependent on the information received at the previous time instants. RNNs are used for tasks of predictive analysis on data sequences, such as speech recognition [H. Sak et al. 2014].

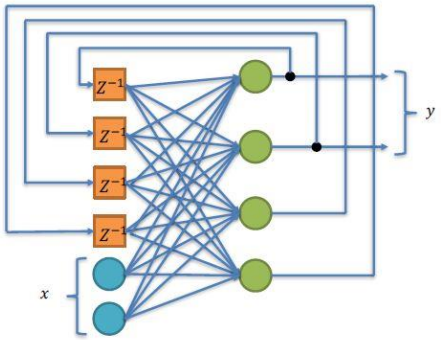


Figure 1.5: RNN structure example

GANs are a class of artificial intelligence algorithms used in unsupervised machine learning, which implements two systems of neural networks that challenge one another in a zero-sum game framework. They were introduced by Ian Goodfellow in 2014 and were used to produce samples of photorealistic images in the field of interior design and industrial design.

However, in this work only FNN and CNN will be described more in detail as the first is the basis of the second which is used to implement object recognition.

To go deeper in details, variables must be formally defined, and the following notation will be used:

- x is the input vector
- w_{jk}^{l} is the weight from the k^{th} neuron in the *(l-1)* th layer to the j^{th} neuron in the l^{th} layer
- b_j^l is the bias of the neuron j in layer l
- a_j^l is the activation of the neuron j in layer l
- $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$ is the j^{th} neuron in the l^{th} layer

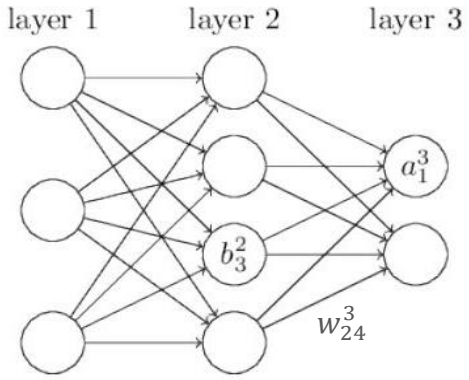


Figure 1.6: NN notations

Logistic functions are often used in neural networks as activation functions to introduce nonlinearity in the model or to hold signals within a specified range. The activation a_j^l can be written as $a_j^l = \sigma(z_j^l)$, where $\sigma()$ is a generic activation function.

The goal of the learning is that a small change in the weights will generate a small change in the output (*Figure 1.7*).

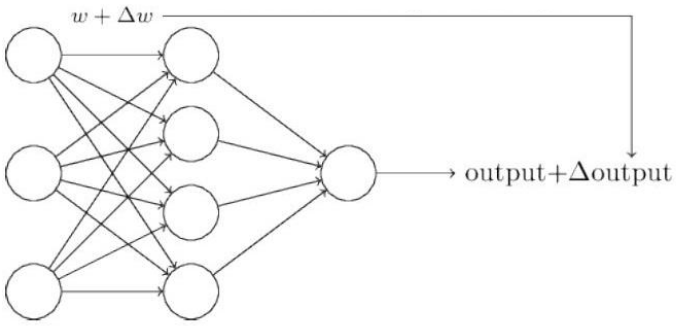


Figure 1.7: effect of weights variations on the output

The more common activation functions are:

1. the “sigmoid” function

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

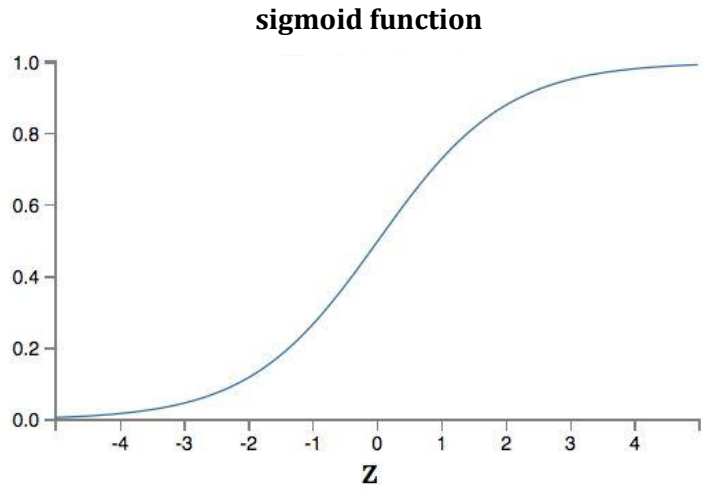


Figure 1.8: sigmoid function

The sigmoid function is used to reduce the effect of small variations and balance them to the final output.

2. the “tanh” function

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

then the activation function is

$$\sigma(z) = \frac{1 + \tanh(z/2)}{2}$$

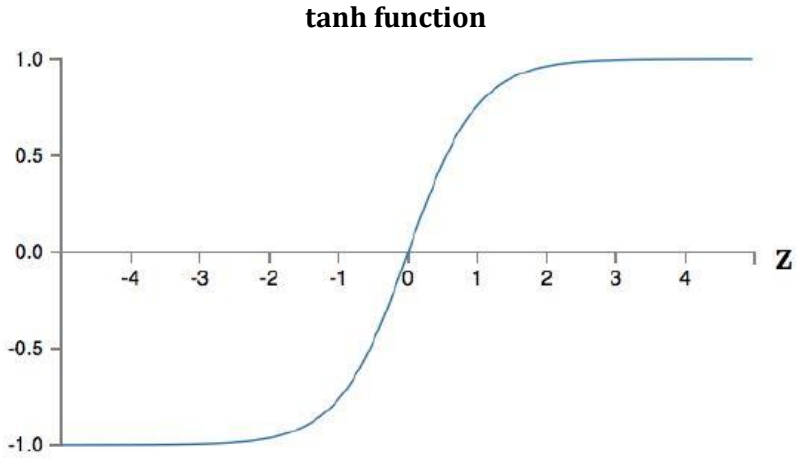


Figure 1.9: tanh function

This function is a good alternative to the sigmoid one and the major difference is that the output's range is (-1, 1) and not (0, 1).

3. the "Linear Unit" function

$$\sigma(z) = z$$

It is a transfer function which produces an output equal to the activation potential and does not modify the input.

4. the "ReLu" function

$$\sigma(z) = \max(0, z)$$

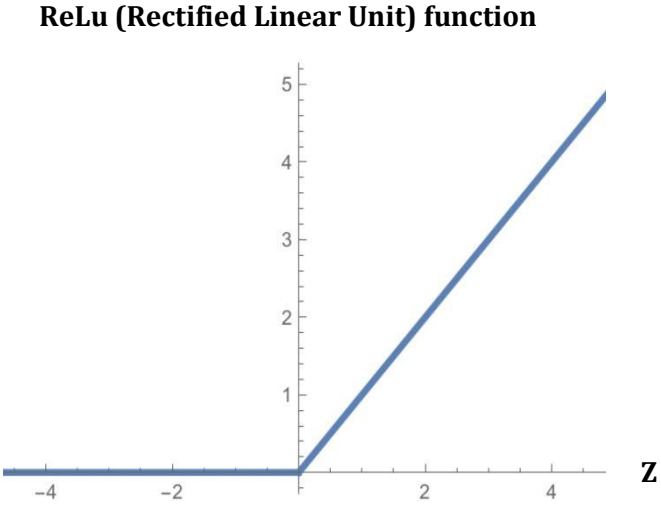


Figure 1.10: ReLu function

It is the most popular activation function for deep neural networks for many factors, among which the most important is the vanishing gradient.

5. the "softmax" function

$$\sigma\left(z_j^L\right) = \frac{z_j^L}{\sum_{k=1}^{n} z_k^L}$$

This function is usually used in the last hidden layer for classification problem. In fact, L stays for the last hidden layer and the sum is over the n possible classes which the output can belongs to.

1.2 Gradient Descent (GD) algorithm

To understand the purpose of the GD algorithm it is necessary to say that there are two types of learning: supervised and unsupervised. Object detection and image classification are two cases of supervised learning as we already know, at least for the training datasets, if an object is present or not and the class to which it belongs.

In the case of unsupervised learning the training dataset is not composed of input and output pairs, but only from input therefore it is not known a priori the output to be obtained.

Having made this clarification, the cost function can be applied to the case of supervised learning and the whole purpose of training a NN is finding weights and biases such that the output approximates the target function for each input.

GD is a first-order iterative optimization algorithm for finding the minimum of the cost function.

An example of the most used cost function is the quadratic one:

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

where *x* is the input, *y(x)* is the desired output and *a(x, w, b)* is the network output.

The more accurate a training is, the more the cost function tends to zero (*Figure 1.11*).

The GD algorithm is an iterative procedure that is used to find weights and biases that minimize the cost function. To explain how it woks, some variables need to be defined.

Let's define υ as the vector which contains the huge number of variables present in the NN.

$$\upsilon = [\upsilon_1, \upsilon_2, \ldots, \upsilon_n]$$

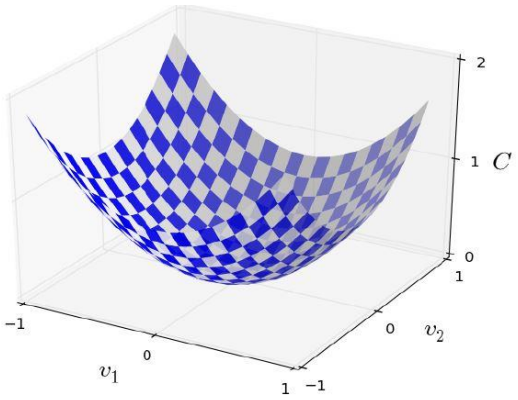


Figure 1.11: example of cost function

Then *C* is a function of υ, *C(υ),* and when the values of the variables change, the cost function changes as

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 + \cdots + \frac{\partial C}{\partial v_n}\Delta v_n \qquad \textit{(eq. 1.1)}$$

$\Delta v = [\Delta v_1, \Delta v_2, \ldots, \Delta v_n]^T$ must be defined such that ΔC is negative.

The gradient of *C* is

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}, \ldots, \frac{\partial C}{\partial v_n}\right)^T \qquad \textit{(eq. 1.2)}$$

and so

$$\nabla C \approx \nabla C \cdot \Delta v \qquad \textit{(eq. 1.3)}$$

Choosing

$$\Delta v = -\gamma \nabla C \qquad \textit{(eq. 1.4)}$$

Then substituting *equation 1.4* into *equation 1.3*:

$$\nabla C \approx -\gamma ||\nabla C||^2 \qquad \textit{(eq. 1.5)}$$

where γ, called learning coefficient, is a positive and small number.

Then in order to minimize *C(υ)*, the following update rule is used

$$v \leftarrow v + \Delta v = v - \gamma \nabla C \qquad \textit{(eq. 1.6)}$$

The value of γ is very important and the right value must be evaluated case by case; because if it is too small the gradient algorithm will be too slow and instead if it is too big, then the algorithm will not find the minimum (*Figure 1.12*).

Too big Just right Too small

Figure 1.12: learning rate values

The explanation above refers to a very general case, then the previous expressions change if instead of υ, the vector of input *x* of dimension n, the weights *w* and biases *b* are used. Then the cost function relative to input *x* is function of *w* and *b*, $C_x(w, b)$. The overall cost function is $C = \frac{1}{n}\sum_x C_x$ and the algorithm is used to find weights and biases that get the cost function towards its minimum, applying the following updates:

$$w_i \leftarrow w_i - \gamma \frac{\partial C}{\partial w_i}$$

$$b_i \leftarrow b_i - \gamma \frac{\partial C}{\partial b_i}$$

1.2.1 Stochastic Gradient Descend (SGD) algorithm

Since $C = \frac{1}{n}\sum_x C_x$ where $C_x = \frac{1}{2}\,||y(x) - a||^2$, hence the computation of $\nabla C = \frac{1}{n}\sum_x \nabla C_x$ can results too slow for a huge amount of variables.

Then a variation of the gradient algorithm, called stochastic gradient algorithm can be used in order to speed up the minimization of the cost function.

It computes ∇C for a small sample of training inputs, called mini-batch, at each iteration.

So, assumed that X is the set of all the training inputs, X will be divided into *m* mini-batch, called X_1, X_2, …, X_m.

It is expected that

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} \approx \nabla C$$

The updates became

$$w_i \leftarrow w_i - \frac{\gamma}{m}\sum_j \frac{\partial C_j}{\partial w_i}$$

$$b_i \leftarrow b_i - \frac{\gamma}{m}\sum_j \frac{\partial C_j}{\partial b_i}$$

In the next iteration another mini-batch is taken and when all the mini-batches have been considered, an epoch is completed and the algorithm starts a new cycle, with a new epoch. The mini-batches can be of different sizes, even mini-batches of just an element can be chosen. This procedure is called online or incremental learning and is similar to how human brains work.

1.3 Backpropagation (BP) algorithm

Backpropagation algorithms are a family of methods used to efficiently train NNs following a gradient descent approach that exploits the chain rule.

The BP algorithm was invented in the 1970's and became popular with a 1986 paper by Rumelhart, Hinton and Williams.

The main feature of BP is its iterative, recursive and efficient methodology for calculating the weights updates in order to improve the network learning, until it is able to perform the task for which it has been trained. As the goal of any supervised learning algorithm is to find the function that best maps a set of inputs to their correct output.

This algorithm is called *backpropagation* because, starting from the output layer of a network and going back, it is able to compute the two partial derivatives of the cost function and subsequently compute easily weights and biases.

To better describe the procedure, it is necessary to define the following points:

- for each layer l, w^l is the weight matrix, b^l the bias vector, z^l the neuron vector and a^l the activation vector.
  - $a^l = \sigma(w^l a^{l-1} + b^l) = \sigma(z^l)$ with components
    $a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l)$, where $\sigma()$ is a generic activation function.
  - $z^l = w^l a^{l-1} + b^l$ with components $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$
- the Hadamard product of two vectors is their element-wise product: $(s \odot t)_j = s_j t_j$
- the cost function $C = \frac{1}{2n}\sum_x ||y(x) - a^L(x)||^2$, where L is the number of layers
- a new variable $\delta_j^l = \frac{\partial C}{\partial z_j^l}$, called error that refers to the neuron j of layer l

The goal is to minimize the cost function, and this requires the computation of $\frac{\partial C}{\partial w_{jk}^l}$ and $\frac{\partial C}{\partial b_j^l}$ to understand how changing weights and biases affects the cost function.

The BP algorithm is based on four key equations:

1. the error in the output layer:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \text{ or } \delta^L = \nabla_a C \odot \sigma'(z^L)$$

   where the first term indicates how much the cost function is changing as a function of the output activation j and the second term measures how fast the activation is changing at z_j^L

2. the error δ^l in layer l as a function of the error in the next layer $l+1$:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

   then if the error in layer $l+1$ is known, through this equation the error on every previous layer can be computed and combining these first two equations the error at any layer of the network can be computed

3. the rate of change of the cost function with respect to any bias

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

4. the rate of change of the cost function with respect to any weight

$$\frac{\partial C}{\partial w_{jk}^{l}} = a_k^{l-1}\delta_j^l$$

Considering equation at point 3 and 4 if δ_j^l is small, weights and biases learn slowly. So, in the case δ_j^L is small and a weight in the last layer learns slowly, the neuron has saturated.

Finally, algorithms such as gradient descent or stochastic gradient descent are always related with BP, that makes computations feasible. For combining BP and gradient algorithm the following steps must be taken:

1. taken an input x → activation a^l, initial weights w^l and biases b^l for all layers (e.g. random)
2. Feedforward → for each layer *l=2, 3, ..., L* compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$
3. output error→ compute $\delta^L = \nabla_a C \odot \sigma'(z^L)$
4. Backpropagation → for each *l=L-1, L-2, ..., 1* compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
5. update weights and biases → $w_{jk}^l \leftarrow w_{jk}^l - \gamma \frac{\partial C}{\partial w_{jk}^l} = w_{jk}^l - \gamma a_k^{l-1}\delta_j^l$ and

$$b_j^l \leftarrow b_j^l - \gamma \frac{\partial C}{\partial b_j^l} = b_j^l - \gamma \delta_j^l$$

Otherwise if BP is combined with SGD, there will be an external for loop on the epoch and an internal for loop on the mini-batch that will work exactly as described above.

1.4 Training problems

The training of a general NN is a very hard task and there is the risk of making mistakes due to following factors:

1. neuron saturation
2. overfitting
3. learning rate
4. vanishing gradient

1.4.1 Neuron saturation

Neuron saturation occurs when the partial derivatives of the cost function are small and therefore the learning is slow and sometimes prevents any kind of improvement.

The problem can be addressed using two different approaches:

a. changing the cost function in the output layer;
b. initializing all variables with a Gaussian probability distribution with mean 0 and standard deviation equal to $\frac{1}{\sqrt{n_{in}}}$ where n_{in} is the number of input weights of the selected neuron.

Considering the first option, the most used cost functions are the cross-entropy and the log-likelihood. They can be described as:

- Cross-entropy cost function

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

where the first sum is over the inputs x and the second over the multiple outputs y.

- Log-likelihood cost function

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

However, using this first option there is the risk to saturate some neurons in the hidden layers. Considering the second option the learning slowdown is greatly reduced if the weight are initialized with the Gaussian distribution. The biases initialization, instead, less affects the slowdown problem then they can be either set all equal to zero or using a Gaussian distribution with 0 mean zero and standard deviation equal to 1.

1.4.2 Overfitting

Another great problem is overfitting which occurs when a huge amount of variable is trained and the network instead of generalizing the model learns the peculiarity of the training dataset. To remedy this inconvenience, there are different ways.

One of the most used is to divide the data into three subsets: the training dataset, that contains about the 80% of the total data, the validation set that contains around 10% of the total data and the test set which contains the 10% left of the total data.

Then during training at the end of each epoch the network is tested on the validation test in order to see if the accuracy on the validation test is increasing while the accuracy on the training dataset is increasing. Because a way to see if overfitting is occurring is to see the accuracy on the validation set stops growing while the accuracy on training set continues to grow. The test set is used just at the end as a final check.
Sometimes if the dataset available is not so big, the training dataset with the 80% of the total is created and the test set with the remaining 20%; then the test set is used as the validation set described above.
Another way to overcome overfitting is to use the K-fold cross validation techniques. The overall data will be divided into K subsets, then the training is done K times considering each time a different subset as test set and all the others together as the training set. At the end if the accuracy is more all less the same each time, then overfitting has not occurred.

A completely different approach to face with overfitting is called Dropout. For each training step only the 50% of neurons is activated and the weights and biases related to the others aren't updated. At the next training iteration, the process is repeated with a different subset of neurons randomly chosen. When the full network is used the weights must be halved. In this way the co-adaptation of neurons is reduced, since a neuron cannot rely on the presence of particular other neurons and so the resulting model is more robust against the loss of an individual piece of information. Dropout is especially useful in training large, deep networks, where overfitting is a major problem.

1.4.3 Learning rate

As already said, the value of the learning rate of the gradient algorithm is very important.
It has a great impact on the training performance because it controls the changing of the model in response to the estimated error each time the weights and biases are updated. Its value must be evaluated case by case because it is very hard to know its optimum value a priori. However, a variation of the GD algorithm, called steepest descent, can be used, since it computes the optimum value of the learning rate at each iteration.
Steepest descent algorithm follows the same steps of the GD algorithm, but in addition it computes the Hessian matrix [L. O. Hesse, 19th century].

This algorithm computes the Hessian matrix of the inputs with respect to the weights and then sets the learning coefficient equal to

$$\gamma = \frac{||\nabla C(w_i)||^2}{\left(\nabla C(w_i)\right)^T H(w_i) \nabla C(w_i)}$$

where C is the cost function and H the Hessian matrix.

The Hessian matrix contains the second derivative of the function with respect to any variable, an example is $H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2} \dots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \dots & \dots & \frac{\partial^2 f}{\partial x_n} \end{bmatrix}$ where f: $\mathbb{R}^n \rightarrow \mathbb{R}$ is a function taking as input a vector $x \in \mathbb{R}^n$ and outputting a scalar $f(x) \in \mathbb{R}$. If all second partial derivatives of f exist and are continuous over the domain of the function, then the Hessian matrix H of f is a square $n \times n$ matrix.

1.4.4 Vanishing gradient

In machine learning, the vanishing gradient problem is a difficulty found in training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training. The problem is that in some cases, the gradient will be too small and then the weights values cannot be efficiently updated. In the worst case, this may completely stop the training.

As example of the problem's cause are the traditional activation functions such as the sigmoid function that have gradients in the range (0, 1), and backpropagation computes gradients by the chain rule. This has the effect of multiplying n of these small numbers to compute gradients of the "front" layers in an n-layer network, meaning that the gradient decreases exponentially with n while the front layers train very slowly.

Back-propagation allowed researchers to train supervised deep artificial neural networks from scratch, initially with little success. In 1991 Hochreiter formally identified the reason for this failure in the "vanishing gradient problem", which not only affects many-layered feedforward networks, but also recurrent networks.

To solve the problem different options are available:

- using CNN, where fewer parameters are needed;
- using dropout;

- using ReLU activations for speed-up (3-5x times) the training process;
- using GPUs and training for much longer time;
- increasing size of training dataset;
- using right cost function and good weight initialization;
- using pre-training early layers.

1.5 Deep learning and Convolutional Neural Network (CNN)

Deep learning is a vast and debated topic, so in this thesis only the techniques related to the classification of images and the recognition of objects will be treated.

Classification is a subfield of the recognition of the model, classifying means assigning to each input value an output value that corresponds to the class to which it belongs. Thanks to the advent of deep learning, CNN has improved a lots image classification and objects' detection and so it became the most used architecture in object recognition field.

Image classification labels the entire image. Instead, detection is finding the position of the objects, usually defined by rectangular coordinates, as shown in *Figure 1.13*.

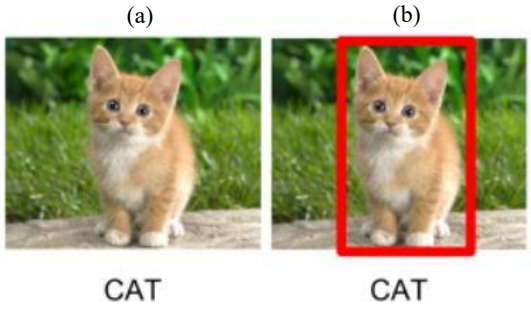


Figure 1.13: classification (a) and detection (b)

To understand how a CNN works, it is necessary to know three fundamental concepts:

1. local receptive fields
2. shared weights/biases
3. pooling

Taken an image and the input neurons, the input neurons are basically the pixel intensities of an input image. Connections are not made from every input to every hidden neuron, but they are made only in small regions KxK (e.g. 5x5), called local receptive field (*Figure 1.14*).

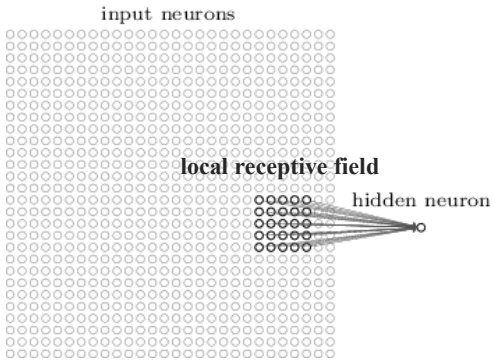


Figure 1.14: input neurons example

The local receptive field slides over the whole image (*Figure 1.15*) and every position corresponds to a hidden neuron. The stride length is by how much the receptive field is shifted, at each neuron. For example, for a 28x28 image, 5x5 receptive field and 1 stride length, in the first layer there will be 24x24 neuron.

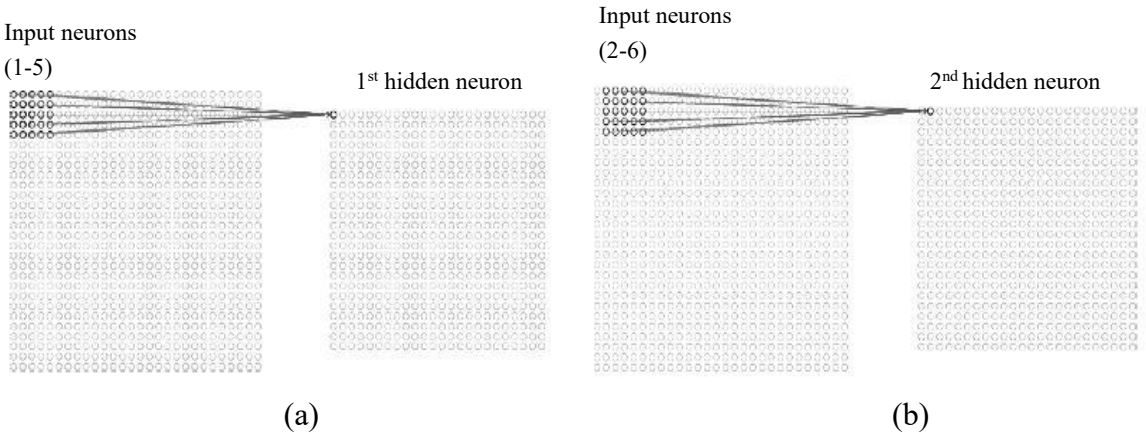


Figure 1.15: example of sliding receptive field (a, b)

The connections from the local receptive field to each neuron have all the same weights:

$$\sigma(b + \sum_{l=0}^{K}\sum_{m=0}^{K} w_{l,m}\, a_{j+l,k+m})$$

All neurons in the first layer detect the same feature, at different locations in the image. This exploits (possible) shift-invariance of the image content. The shared weights/bias define a filter or kernel and a complete convolutional layer consists of several different filters.

In *Figure 1.16*, an example of complete convolutional layer with depth equal to three.

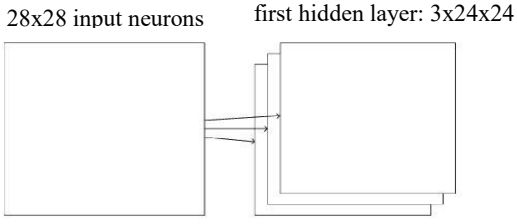


Figure 1.16: example of filters

The use of shared weights reduces the number of parameters and therefore overfitting is less likely. For example, with a 5x5 filter there are 26 parameters and with 20 filters, the number of parameters increases to 520. But if there are a first fully connected layer and then 30 hidden neurons, this would require 23550 parameters.

Pooling layers "simplify" the information output from a convolutional layer, performing a sort of down sampling. Max-pooling (*Figure 1.17*) means that each pooling unit outputs the maximum activation in a 2x2 input region.

Pooling layers are applied independently to each feature map.

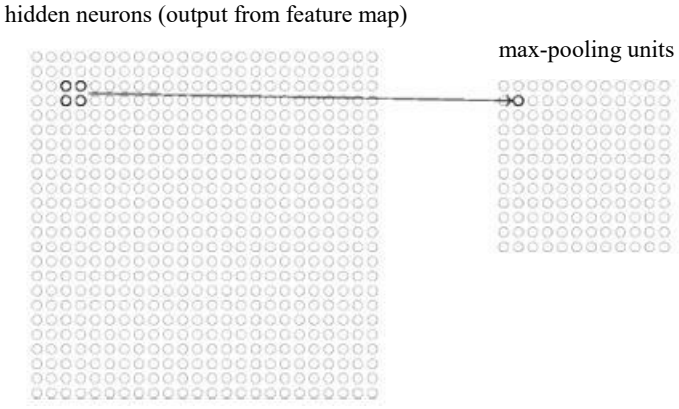


Figure 1.17: example of max-pooling

In *Figure 1.18* an example of a simple and complete CNN, obviously the number of layers can be large.

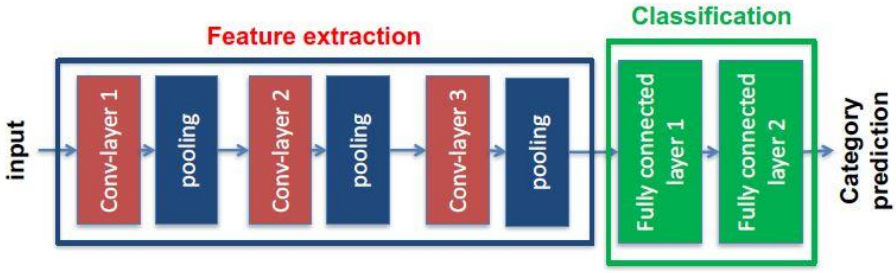


Figure 1.18: example of CNN

The training of a CNN is still performed using backpropagation, with few adaptations for convolutional and max-pooling layers.

1.5.1 Performance of CNNs for objects' detection

CNN combined to Selective Search produces Regional-CNN (*Figure 1.19*). Selective Search is a combination of exhaustive search and segmentation. Since an object can be located everywhere and scale in the image, it is natural to search everywhere [Dalal and Triggs 2005; Harzallah et al. 2009; Viola and Jones 2004]. However, the search space is huge, and this

makes the exhaustive search computationally expensive. Segmentation aims a partitioning of the image through a generic algorithm, where there is a part for all the profiles of the object in the image.

Given an image, each pixel is analysed, and then "similar" pixels are grouped together. A greedy algorithm is used to recursively combine "similar" regions into larger ones.

At the end the generated regions are used to produce the final candidate region proposals. These candidate region proposals are warped into a square and fed into a convolutional neural network that produces a feature vector as output.

The CNN acts as a feature extractor and the output dense layer consists of the features extracted from the image. The extracted features are fed into an SVM (Support Vector Machine) to classify the presence of the object within that candidate region proposal. SVM are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis.

In addition, to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values to increase the precision of the bounding box.

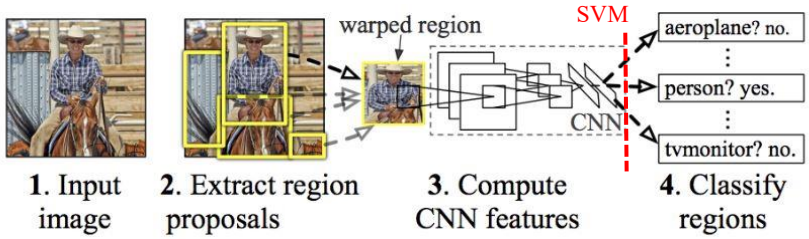


Figure 1.19: example of R-CNN

However, R-CNN needs a huge amount of time to train the network and it cannot be used in real time application as it takes around 47 seconds for each test image [R. Girshick et al. 2015]. Moreover, selective search algorithm is fixed, therefore, no learning is happening at that stage and this could lead to the generation of bad candidate region proposals.

Then the Fast R-CNN (*Figure 1.*20) approach was introduced by Ross Girshick in 2015. The input image is given to the CNN in order to generate a convolutional feature map and from the feature map, the region of proposals is identified and warped into squares.

Then through a RoI pooling layer, the squares are resized to a fixed size so that it can be fed into a fully connected layer. From the RoI feature vector, a softmax layer is used to predict the class of the proposed region and also the offset values for the bounding box.

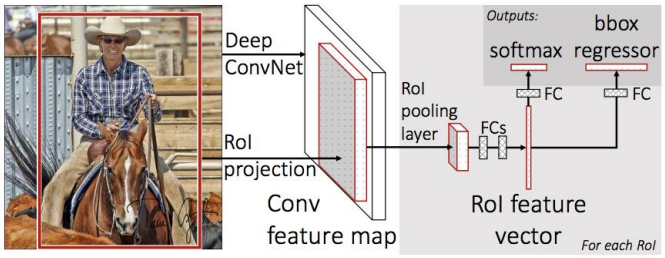


Figure 1.20: example of Fast R-CNN

This approach is faster because the convolution operation is done only once per image and a feature map is generated from it. Fast R-CNN is faster both in training and testing.

However, comparing the performance of Fast R-CNN during testing time, including region proposals, shows that region proposals become bottleneck (*Figure 1.21*).

Both R-CNN and Fast R-CNN uses selective search to find out the region proposals.

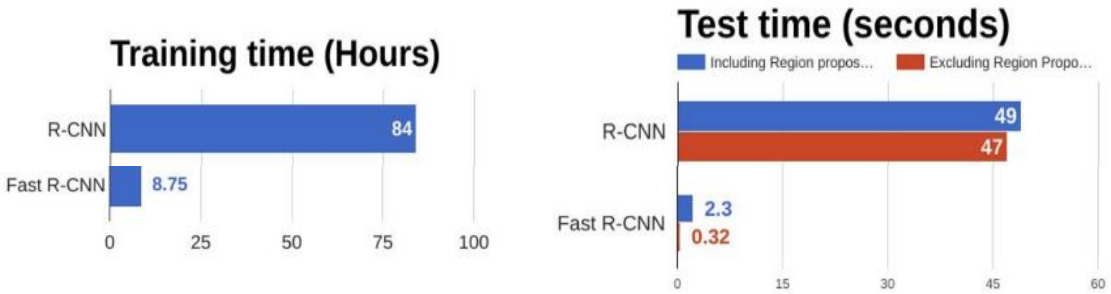


Figure 1.21: comparison of R-CNN and Fast R-CNN training and testing time

Therefore, the Faster R-CNN (*Figure 1.22*) was introduced by S. Ren, R. Girshick and J. Sun in 2016. As for the Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map. But differently from the Fast R-CNN that uses selective search on the feature map to find the region proposals, the Fasters R-CNN uses a separate network to predict the region proposals. Then they are reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

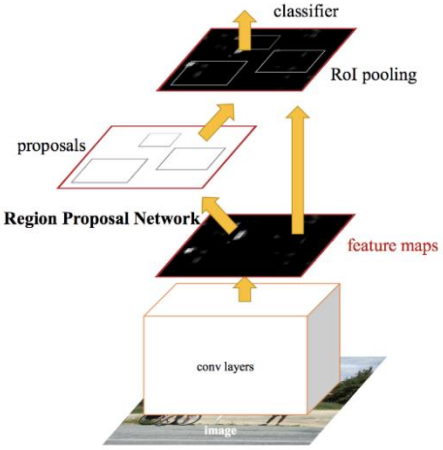


Figure 1.22: example of Faster R-CNN

Again, a comparison of the testing time, including region proposals, required by: R-CNN, Fast R-CNN and Faster R-CNN is shown in the *Figure 1.23*. It clearly emerges that Faster R-CNN can be used for real-time object detection.

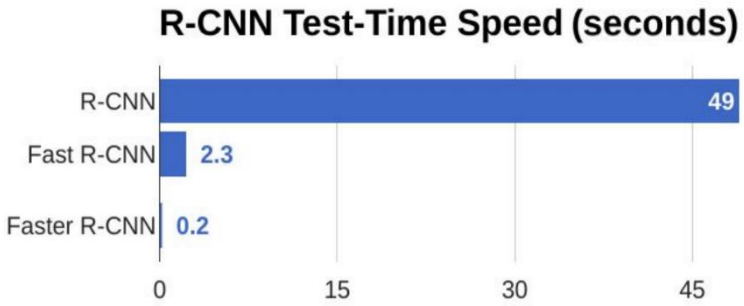


Figure 1.23: R-CNN, Fast R-CNN and Faster R-CNN testing time

1.6 Platforms and Libraries

Nowadays machine learning is used in a wide domainis and therefore online it is possible to find different platforms and libraries that help in the development of one's NN.

To make the right choice, given the variety of solutions it is necessary to understand is the purpose of the training is part of a simpler case of machine learning or if it requires something more powerful and therefore it is a case of deep learning.

Most of the solutions found online are based on: C ++, Java and Python programming languages. So given my personal programming experience, we chose to use Python as a programming language and therefore platforms and libraries that support it.

The list of the most popular is in the following:

- **TensorFlow** was designed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization. The system is developed to help research in machine learning. It provides stable Python and C APIs as well as non-guaranteed backwards compatible API's for C++, Go, Java, JavaScript, and Swift.
- **Microsoft Cognitive Toolkit** (CNTK) is a unified deep learning toolkit that describes NNs as a series of computational steps via a directed graph. In the graph, leaf nodes represent input values or network parameters, while others represent matrix operations upon their inputs. CNTK allows users to realize and combine popular model types such as FNNs, CNNs and RNNs. It implements SGD and BP

with automatic differentiation and parallelization across multiple GPUs and servers. CNTK has been available under an open-source license since April 2015.

- **Scikit-learn** is simple and efficient tools for data mining and data analysis, accessible to everybody. It is a Python module built on top of SciPy and is distributed under the 3-Clause BSD license. The project was started in 2007 by David Cournapeau as a Google Summer of Code project, and since then many volunteers have contributed, it is currently maintained by a team of volunteers.
- **Keras** is, a high-level NNs API, written in Python and able to run on top of TensorFlow, CNTK, or Theano. It allows easy and fast prototyping and supports both CNNs and RNNS, as well as their combinations. It runs on both CPU and GPU.
- **PyTorch** is a Python package that provides: Tensor computation (like NumPy) with strong GPU acceleration and Deep NNs built on a tape-based autograd system.
- **Caffe** is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center (BVLC) and community contributors.
- **Neon** is Nervana's Python-based deep learning library. It is Intel's reference deep learning framework committed to best performance on all hardware. It provides ease of use while delivering the highest performance.
- **PyBrain** is a modular Machine Learning Library for Python. It offers flexibility, simplicity and powerful algorithms. It has for a variety of predefined environments to test and compare different algorithms.
- **Darknet** is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.
- **Open Source Computer Vision Library** (OpenCV) is an open source computer vision and machine learning software library, it is originally developed by Intel, and later it was maintained by Willow Garage and now by Itseez. The programming language mainly used to develop OpenCV is C ++, but it is also possible to interface through C, Python and Java.

In this thesis OpenCV and Darknet have been exploited.

1.7 Online available dataset

As said above, training of a NN requires a huge amount of data. Therefore, exploiting the data sets already available online for image classification and object detection is very helpful. These datasets vary in number of objects, ranging from 20 to 200 annotated in these datasets. In addition, some datasets have too many objects in a single image compared to others with just an object per image.

ImageNet is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a "synonym set" or "synset". There are more than 100 thousand synsets in WordNet, majority of them are nouns (80,000+). In ImageNet, the aim is to provide about 1000 images to illustrate each synset. Images of each concept are quality-controlled and human-annotated. In its completion, the hope is that ImageNet will offer tens of millions of cleanly sorted images for most of the concepts in the WordNet hierarchy.

The **PASCAL VOC** datasets were provided as part of the PASCAL Visual Object Classes challenge from 2005 to 2012. The goal of the datasets is to recognize objects from a number of visual object classes in realistic scenes. The dataset has more than 11 thousand images with over 27 thousand of annotations. This dataset can recognize objects of the following 20 classes:

- person: person
- animal: bird, cat, cow, dog, horse, sheep
- vehicle: airplane, bicycle, boat, bus, car, motorbike, train
- indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

The **Common Objects in Context** (COCO) is a large-scale object detection, segmentation, and captioning dataset. It is designed for the detection and segmentation of objects occurring in their natural context and it is the most extensive publicly available object detection database. It has about 330 thousand images with more than 200 thousand object annotations in more than 90 stuff categories (*Figure 1.24*).

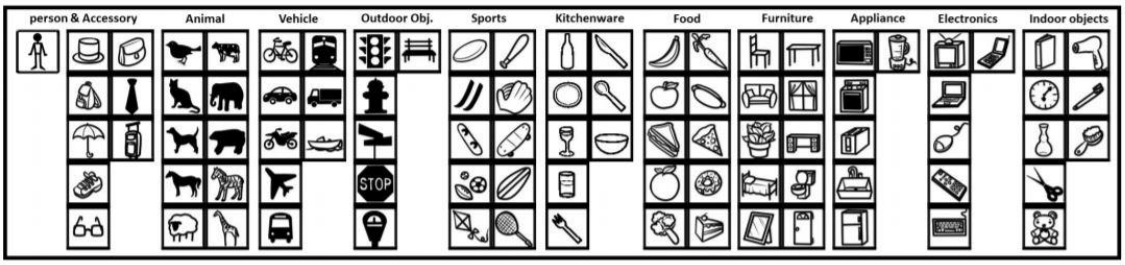


Figure 1.24: icons of 91 categories in the COCO dataset grouped by 11 super-categories

In this thesis both ImageNet and the COCO datasets have been exploited.

2. Object detection algorithms

As described in the previous chapter there are several online datasets, containing images and labels that can be used as they are or integrated with new images of objects already present in the dataset or new, the important thing is that the additional images are labeled.
The images in the online datasets are traditional RAW data, that may vary in image size and quality.
Online, there are also available several platforms and libraries that implement object recognition.
Object detection task is a critical part of any surveillance system and the analyzed recognition algorithms look for the characteristics of the input image and therefore work well on images taken in good lighting conditions. In low light condition, the performance of surveillance system using the traditional camera is poor, because the objects captured by traditional cameras have low contrast against the background due to the absence of ambient light in the visible spectrum. Night vision is the ability to see things in low light conditions and it has made possible by a combination of two techniques: enough spectral range and sufficient intensity range.
Thermal cameras have been a popular choice of implementation of night vision surveillance systems. They can be used to detect humans, but their cost is high, and they cannot be used to detect indoor or outdoor objects that are not warm.

2.1 Object detection using Haar feature

Object detection using Haar feature was proposed by P. Viola and M. Jones in 2001. It is a machine learning approach where a cascade function is trained with two sets of images: positive and negative. The set of positive images consists of images which contain the object to be recognized. The set of negative images, instead, can include any image that does not contain the chosen object.

First, the algorithm needs the two sets of images to train the classifier and then, features need to be extracted from them. Haar features shown in *Figure 2.1* are used.

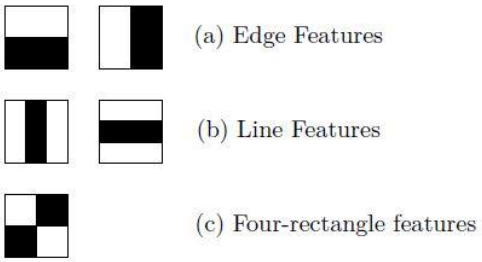


Figure 2.1: Haar features

Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle (for example a 24x24 window has over 160000 features). All possible sizes and locations are used to calculate lots of features.

To compute the features, the integral image is used. Because it reduces the calculations for a given pixel to an operation involving just four pixels, so it makes things faster.

The integral image can be computed from an image using a few operations per pixel. The integral image at location *(x, y)* contains the sum of the pixels above and to the left of *(x, y)*:

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y') \qquad \textit{(eq 2.1)}$$

$$s(x,y) = s(x,y-1) + i(x,y) \qquad \textit{(eq 2.2)}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \qquad \textit{(eq 2.3)}$$

where: *ii(x, y)* is the integral image and *i(x', y')* is the original image; *s(x,y)* is the cumulative row sum, *s(x, -1) = 0* and *ii(-1, y) = 0*.

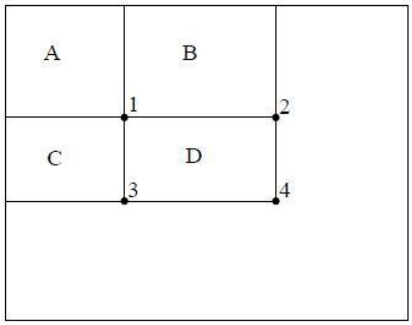


Figure 2.2: integral image

Using the integral image, any rectangular sum can be computed in four array references, as shown in *Figure 3.2*. In particular, the sum of the pixels within rectangle D can be computed with the four array references: 1, 2, 3 and 4. Since, the value of the integral image at location 1 is the sum of the pixels in rectangle A, at location 2 is A+B, at location 3 is A+C and at location 4 is A+B+C+D. Then the sum within D can be computed as 4+1-(2+3). Then more

in general, the difference between two rectangular sums can be computed in eight references. Since the two-rectangle features, defined above, involve adjacent rectangular sums, they can be computed in six array references, eight in the case of the three-rectangle features, and nine for four-rectangle features. Following this technique, the integral image can be computed in one pass over the original image.

However, among all the features, most of them are irrelevant. In fact, chosen a feature, it is useful only in a specific area of the image. For example, consider the feature that stores the property that the region of the eyes is often darker than the region of the nose and cheeks; if this feature is moved to cheeks or any other place, it results useless.
Now, it is clear that the number of features effectively needed by the classifier is small and to select them the AdaBoost algorithm is used.
The AdaBoost algorithm works as follow:

1. given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
2. initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for yi= 0, 1 respectively, where m and l are the number of negatives and positives respectively.
3. for t = 1, …, T:
   i. normalize the weights $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$ so that w_t is a probability distribution.
   ii. for each feature j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
   iii. choose the classifier, h_t, with the lowest error ϵ_j.
   iv. update the weights: $w_{t+1,i} = w_{t,i}\beta_t^{1-\epsilon_i}$ where $\epsilon_i = 0$ if example x_i is classified correctly, $\epsilon_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
4. the final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2}\sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log\frac{1}{\beta_t}$.

In the algorithm each feature is applied on all the training images. For each feature, AdaBoost finds the best threshold which will classify the image to positive or negative.
Obviously, there will be errors or misclassifications. The features selected (*Figure 2.3*) are the ones with minimum error rate, which means they are the features that most accurately classify the presence and absence of the object in the training images.

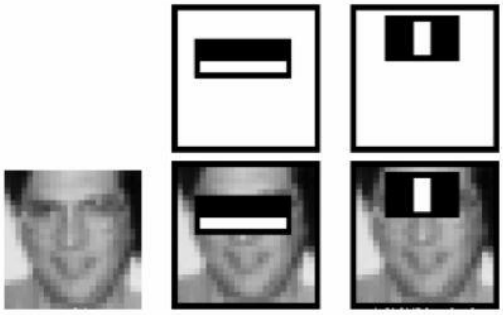


Figure 2.3: example of Haar feature selected by AdaBoost

To each image is given an equal weight at the beginning; then at each classification, weights of misclassified images are increased.
The same process is done again, so new error rates and new weights are calculated. The process is repeated until the required accuracy or error rate is achieved or the required number of features is found.

The final classifier is a weighted sum of these weak classifiers. They are called weak because each of them alone can't classify the image, but together form a strong classifier.

In face detection 200 features provide detection with 95% accuracy but the final setup had around 6000 features (the reduction is from 160000+ features to 6000 features). So taken an image, each 24x24 window will be applied 6000 features to it, in order to check if it is face or not.
But in general, given an image, most of the areas are non-face region. So, it is better to have a simple method that checks if a window is or not a face region. If it is not, discard it in a single shot, and don't process it again. Instead, focus on regions where there can be a face.
In this way, more time is spent for checking possible face regions.
The concept of Cascade of Classifiers has been introduced and instead of applying all 6000 features on a window, the features are grouped into different stages of classifier. The stages are applied one-by-one and normally the first few stages will contain fewer features.

If a window fails the first stage, discard it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

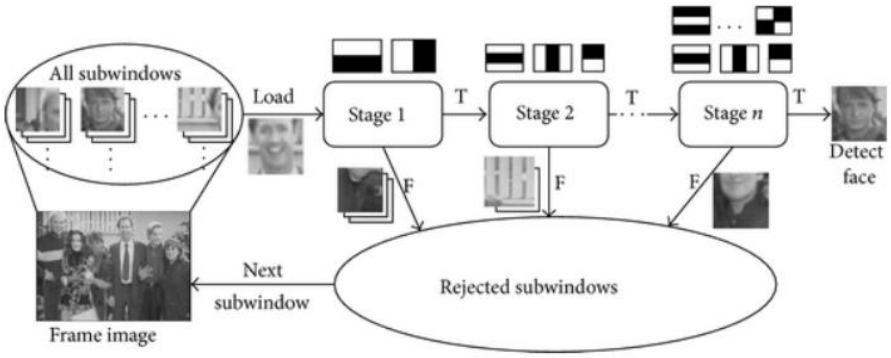


Figure 2.4: cascade of classifiers

2.2 Object detection using Y.O.L.O.

All previous object detection techniques: R-CNN, Fast R-CNN and Faster R-CNN use regions to localize the object within the image and the network does not look at the complete image. Y.O.L.O. (You Only Look Once) is an object detection algorithm in which a single convolutional network predicts the bounding boxes and the class probabilities for these boxes [J. Redmon et al. 2013].

Y.O.L.O. trains on full images and directly optimizes detection performance.

This unified model has several benefits over traditional methods because it is extremely fast and achieves more than twice the mean average precision of other real-time systems.

Unlike region proposal-based techniques, Y.O.L.O. sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.

It makes less than half the number of background errors compared to Fast R-CNN and learns generalizable representations of objects. So Y.O.L.O. is less likely to break down when applied to new domains or unexpected inputs.

A negative aspect of this new technique is that it is less accurate than state-of-the-art detection systems. It fast identifies objects in images, but it strives to precisely localize some objects, especially small ones.

In the Y.O.L.O. procedure first of all, the input image is divided into an SxS grid. Then, if the centre of an object falls into a grid cell, that cell is responsible for detecting that object.

Each grid cell predicts B bounding boxes and confidence scores for those B boxes. The boxes with the thickest edges are the ones with the highest scores.

The confidence score reflects how sure the model is that the box contains a certain object and how accurate it thinks the predicted box is, formally it is computed as:

$$Probability(Object)*IoU$$

where IoU is Intersection over Union (§**4.2**).

Each grid cell also predicts C conditional class probabilities, $Probability(Class_i|Object)$, and these probabilities are conditioned on the grid cell containing an object. Only a set of class probabilities per grid cell, regardless of the number of boxes B, is computed.

During testing, the conditional class probabilities is multiplied by the individual box confidence score predictions:

$$Probability(Class_i|Object) * Probability(Object) * IoU = Probability(Class_i) * IoU$$

The result is the class-specific confidence scores for each bounding box; these values represent both the probability of the $Class_i$ appearing in the box and how well the predicted box encloses the object.

Else if no object exists in that cell, the confidence scores should be zero.

To make the procedure described above clearer, *Figure 2.5* and the following summary are inserted.

1. it divides the image into an SxS grid *(a)*
2. for each grid cell predicts:
    a. B bounding boxes *(b)*
    b. confidence scores for those boxes *(b)*
    c. C class probabilities *(c)*
3. these predictions are encoded as S * S * (B * 5 + C) tensor
4. detection output *(d)*

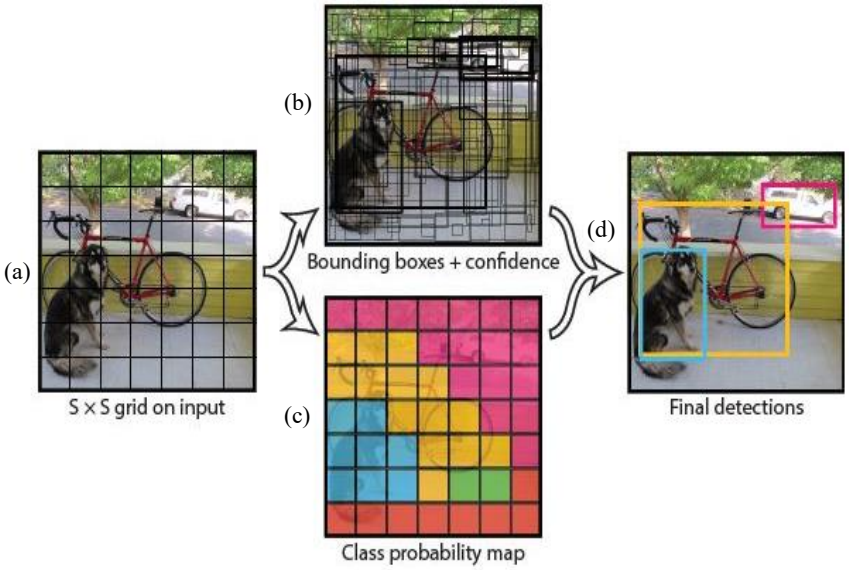


Figure 2.5: Y.O.L.O. flowchart (a, b, c, d)

In particular, the evaluation of Y.O.L.O. on PASCAL VOC uses the following parameters: S=7, B=2 and C=20. *Figure 2.6* illustrates the structure of the convolutional NN used.
It has 24 convolutional layers followed by 2 fully connected layers. The convolutional layers are used to reduce the feature maps and the last one is used to output a tensor with the desired shape (7x7x30).

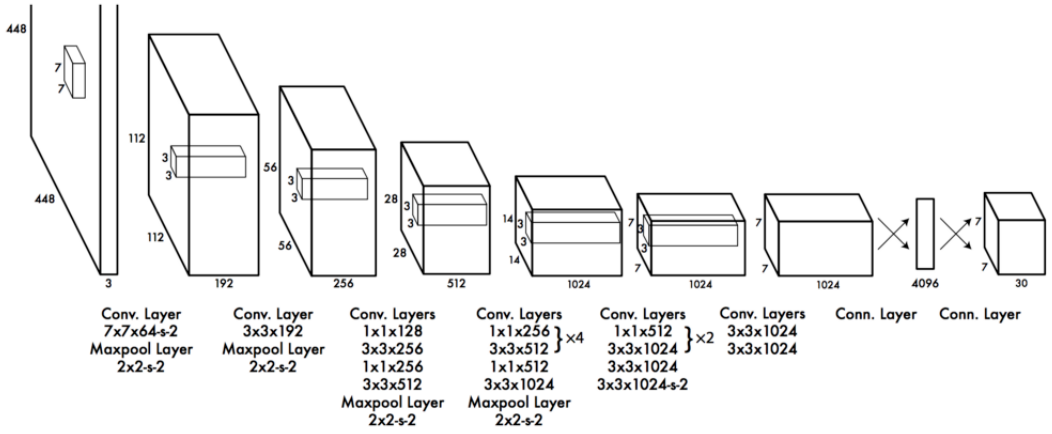


Figure 2.6: Y.O.L.O. CNN

Y.O.L.O. main source of error is incorrect location because of the strong spatial constraints on bounding box predictions, because each grid cell only predicts two boxes and can only have one class and this constraint limits the number of nearby objects that the model can predict. As a consequence, the model makes mistakes also with small objects that appear in groups, as flocks of birds.
Unlike classifier-based approaches, Y.O.L.O. is trained on a loss function that approximates detection performance, so the errors on both small and large bounding boxes are treated the same. This is not optimum because a small error in a large box is less significant than a small error in a small box that has a greater effect on the IoU.

2.2.1 YOLOv3

In this thesis YOLOv3 has been used, because it is the latest and fastest version of Y.O.L.O. YOLOv3 uses a few tricks to improve training and increase performance, including multi-scale predictions, a better backbone classifier, and more.
Its network predicts 4 coordinates and an objectness score for each bounding box.
Objectness is evaluated using logistic regression that is 1 if the bounding box prior overlaps a ground truth object by more than a threshold (IoU > 0,5) and more than any other bounding box prior. If the bounding box prior is not the best, it is discarded and if a bounding box prior

is not assigned to a ground truth object it incurs no loss for coordinate or classification, only objectness.

YOLOv3 performs a multilabel classification because it uses independent logistic classifiers. Indeed, during training the cross-entropy is use as loss function for class prediction.

This formulation helps in complex domains where there can be many overlapping labels (i.e. Woman and Person). In fact, using a softmax function implies that each box has exactly one class which is often not the case, then a multilabel approach better models the data.

More in details, YOLOv3 predicts boxes at 3 different scales and from the base feature extractor several convolutional layers are added. Features are extracted as follow:

1. the prediction is made in the last feature map layer
2. 2 layers back are considered and up sampled by 2
3. the feature map with higher resolution in the considered layer is merged with the up-sampled feature map using concatenation to get more meaningful information
4. few more convolutional filters are added on the merged map and applied in order to generate prediction tensor
5. at the end, the same steps are performed again to predict boxes for the final scale that is a tensor of NxNx[3*(4+1+80)] for the 4 bounding box coordinates, 1 objectness prediction, and 80 class predictions of the COCO dataset.

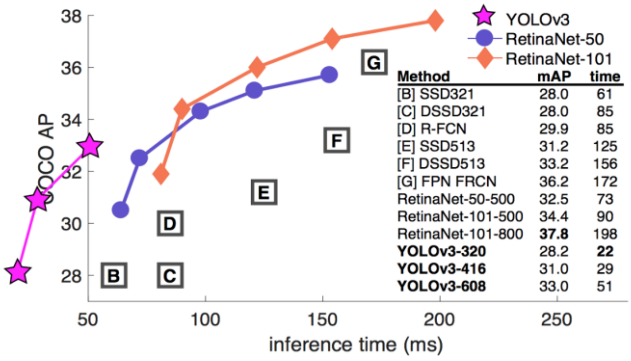
YOLOv3 still uses k-means clustering to determine bounding box priors and on COCO dataset the number of clusters is 9: (10x13); (16x30); (33x23); (30x61); (62x45); (59x119); (116x90); (156x198); (373x326). Clusters are grouped and assigned to specific feature map to improve the object detection.

	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 2.7: YOLOv3 network (Darknet-53)

YOLOv3 achieves the highest measured floating-point operation per second, so its application requires the utilization of GPU hardware.

Figure 2.8 [J. Redmon et al. 2018] illustrates the comparison between the performance of several object detection model, so YOLOv3 is the fastest one and that it has improved the detection of small object.



Model	Train	Test	mAP	FLOPS	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244
SSD321	COCO trainval	test-dev	45.4	-	16
DSSD321	COCO trainval	test-dev	46.1	-	12
R-FCN	COCO trainval	test-dev	51.9	-	12
SSD513	COCO trainval	test-dev	50.4	-	8
DSSD513	COCO trainval	test-dev	53.3	-	6
FPN FRCN	COCO trainval	test-dev	59.1	-	6
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20

Figure 2.8: performance comparison

3. Detection and matching of features

The human brain does a lot of pattern recognition to make sense of raw visual inputs. As the eyes focuses on an object, the brain automatically identifies the characteristics of this object, such as the shape, the color, the texture and others. Then, always in a completely automatic manner, it searches for these characteristics in other objects to recognize it if it is revised again. Consequently, the desirable property for a feature detector is repeatability: whether the same feature will be detected or not in two or more different images of the same scene.

In computer vision, the process of deciding what to focus on is called feature detection.

A feature can be formally defined as "*one or more measurements of some quantifiable property of an object, computed so that it quantifies some significant characteristics of the object*" [R. Castelman, et al. 1996]. Using easier word: a feature is defined as an "interesting" part of an image.

Therefore, the purpose of the recognition of the features in an image is to find the unique characteristics of the image, to later recognize these characteristics in other images similar to the first.

Two of the most famous feature detectors are SIFT (Scale-Invariant Feature Transform) and SURF (Speeded-Up Robust Features).

Features descriptor are also useful to compare images, comparing the descriptor of the two images is a way to find their match. The combination of feature descriptors and their match is a way to identify objects, as shown in *Figure 3.1*.

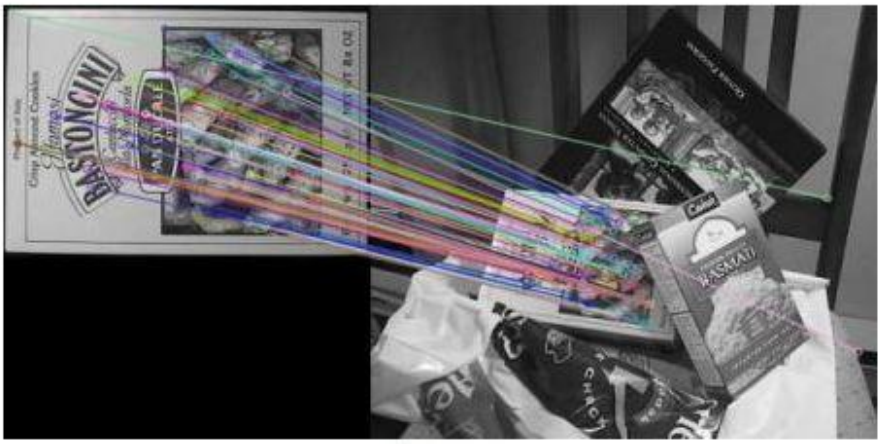


Figure 3.1: feature matching example

3.1 SIFT

SIFT was introduced by Lowe in 2004 and it solves the image rotation, affine transformations, intensity, and viewpoint change in matching features.

The SIFT algorithm is composed by 4 steps:

1. *Scale-space Extrema Detection* is used to estimate a scale space extremum using the Difference of Gaussian (DoG). DoG is the difference of the Gaussian blurring of an image with two different σ: σ and *k*σ.

   This computation is done for various octaves of the image in Gaussian Pyramid, an example in *Figure 3.2*.

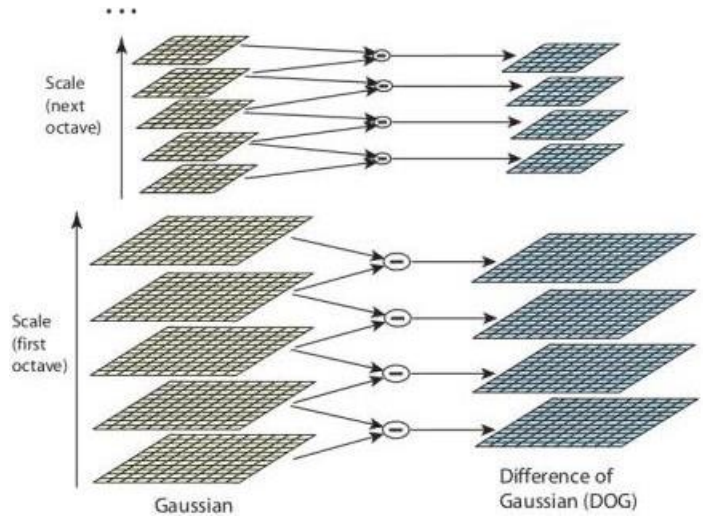


Figure 3.2: DoG for different octave

As DoG are computed, images are searched for local extrema over scale and space. For example, a pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extremum, it is a potential key point. This means that key point is best represented in that scale.

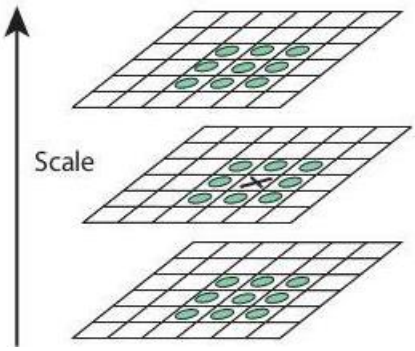


Figure 3.3: searching for local extremum

Regarding different parameters, Lowe gives some empirical data which can be summarized as: 4 octaves, 5 scale levels, σ = 1.6 and $k = \sqrt{2}$ as optimal values.

2. *Key point localization* is where the key point candidates are localized and refined by eliminating low contrast and edges key points. To get more accurate results the potential key points locations found are compared with two thresholds. The Taylor series expansion of scale space is used to compute the intensity of the extremum and if this intensity is lower than a threshold (e.g. 0,03 per Lowe), the extremum is refused.

   Then to remove edges a 2x2 Hessian matrix is used to evaluate the principal curvature. From Harris corner detector is known that for edges, one eigen value is larger than the other. So, if this ratio is greater than a threshold (e.g. 10 for Lowe), that key point is discarded.
3. *Key point orientation assignment* is based on local image gradient. To each key point an orientation is assigned to achieve invariance to image rotation. Around the key point location depending on the scale a neighbourhood is selected, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. It is weighted by gradient magnitude and gaussian-weighted circular window with σ equal to 1.5 times the scale of key point. The highest peak is kept and any peak above the 80% of it is considered to calculate the orientation. This generates key points with same location and scale, but different directions and contributes to stability of matching.
4. *Key point descriptor* is used to compute the local image descriptor for each key point based on image gradient magnitude and orientation. A 16x16 neighbourhood around the key point is taken, then it is divided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bins orientation histogram is generated and then 128 bins values are available, in total. The key point descriptor is represented as a vector. In addition to this, several measures are taken to achieve robustness against illumination changes and rotation.

Key points between two images are matched by identifying their nearest neighbours. But if the second closest match is too near to the first, due to noise or some other reasons, then the ratio of closest-distance to second-closest distance is taken. If it is greater than 0,8, they are rejected. This removes about 90% of false matches while discards only around 5% of correct matches. The lower the threshold, which is used to compare the ratio-test the lower the number of false matching.

3.2 SURF

In 2006, H. Bay, T. Tuytelaars and L. Van Gool have introduced SURF which is a new algorithm for the detection of features. This new algorithm approximates the DoG with box filters. Instead of Gaussian averaging the image, squares are used for approximation since the convolution with square is much faster if the integral image is used (*Figure 3.4*). This can be done in parallel for different scales.

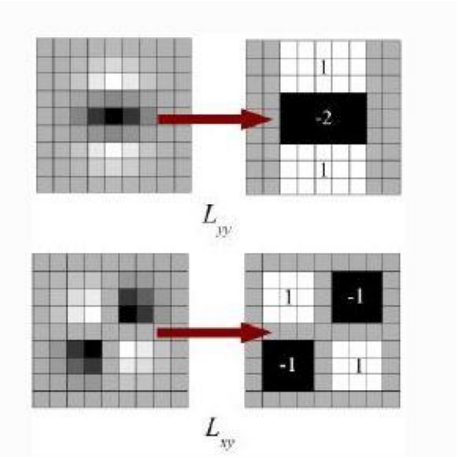


Figure 3.4: DoG approximation

Also, SURF relies on determinant of Hessian matrix for both scale and location of the key points. For orientation assignment, it uses wavelet responses in both horizontal and vertical directions by applying adequate Gaussian weights for a neighbourhood of size 6s.

Then they are plotted in a space as shown in *Figure 3.5*.

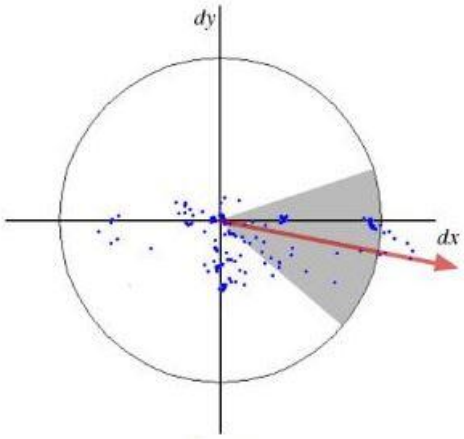


Figure 3.5: SURF orientation assignment

The dominant orientation is estimated by the summation of all responses within a sliding orientation window of angle 60 degrees.

Wavelet response can be found out using integral images easily at any scale. For many applications, rotation invariance is not required, so no need of finding this orientation, and

SURF provides a functionality called Upright-SURF or U-SURF that improves speed and is robust up to ±15 degrees.

For feature description it uses the wavelet responses. A neighbourhood of size 20sx20s around the key point is selected and divided into subregions of size 4x4. Then, for each subregion the wavelet responses are taken and represented to get SURF feature descriptor as a vector like $v = (\sum dx, \sum dy, \sum|dx|, \sum|dy|)$ with total dimension 64.

Lower the dimension, higher the speed of computation and matching, but provide better distinctiveness of features. For more distinctiveness, SURF feature descriptor has an extended version of dimension 128.

Another important improvement is the use of sign of Laplacian (trace of Hessian Matrix) for underlying interest point. It adds no computation cost since it is already computed during detection.

The sign of the Laplacian distinguishes bright blobs on dark backgrounds from the reverse situation. Then in the matching stage, only features that have the same type of contrast (*Figure 3.6*) are compared.

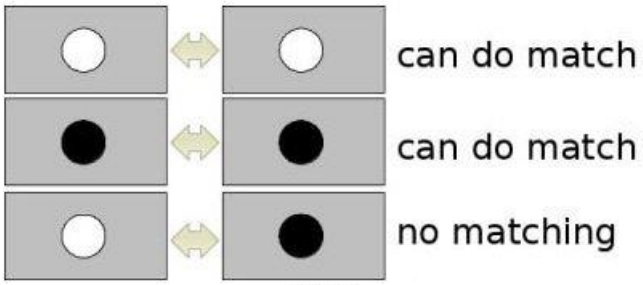


Figure 3.6: contrast matching

This minimal information allows for faster matching, without reducing performance.

Briefly, SURF adds a lot of features to improve the speed in every step.

Analysis shows that SURF should be 3 times faster than SIFT and it is good at handling images with blurring and rotation, but not good at handling viewpoint change and illumination change.

3.3 RANSAC algorithm

RANSAC stands for RANdom SAmple Consensus and in computer vision, it is used as a method to calculate features matching and homography between two images [L. Dung et al. 2013].

Chosen two images, their point correspondences are found by matching features using SIFT or SURF. Then, a transformation is calculated based on these matched features that warps one image into the other one.

Four initial feature matches are taken in the random selection step of each iteration in RANSAC, and a correct homography in obtained only after the final iteration if they are the real inliers.

In the field of computer vision, any two images of the same planar surface in space are related by a homography. This has many practical applications, such as image rectification, image registration, or computation of camera motion between two images. Once camera rotation and translation have been extracted from an estimated homography matrix, this information may be used for navigation, or to insert models of 3D objects into an image or video, so that they are rendered with the correct perspective and appear to have been part of the original scene.

From a practical point of view, homography is a 3×3 matrix that maps the points of one image to the corresponding points in the other image.

To calculate a homography between two images, at least four points correspondences between the two images need to be known. If more than four corresponding points are known, it is even better.

The algorithm's procedure can be described by the following points:

1. select four feature pairs (at random)
2. compute homography H
3. compute inliers
4. keep largest set of inliers
5. re-compute least-squares H estimate using all the inliers

As already said, homography is 3×3 matrix:

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

Let (x_1, y_1) be a point in the first image and (x_2, y_2) be the coordinates of the same physical point in the second image.

Then, the Homography H relates them in the following way:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

If the homography is known, it could be applied to all pixels of one image to obtain a warped image that is aligned with the second one.

An example of result in *Figure 3.7* to better understand what can be achieved using this procedure.

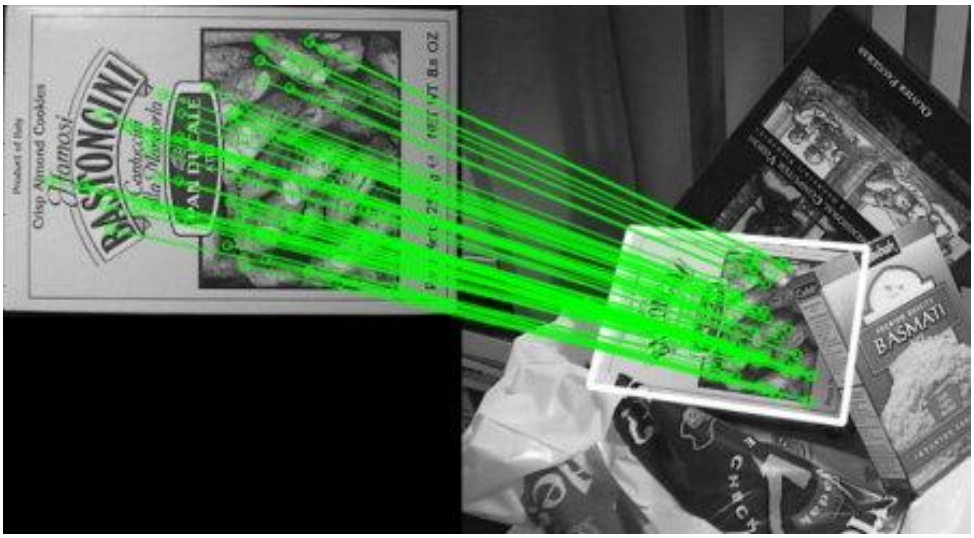


Figure 3.7: feature matching + homography

4. Evaluation metrics

To compare the performances of the chosen techniques it is necessary to define an evaluation metric that will be performed on datasets.

4.1 Confusion Matrix

In the field of machine learning, confusion matrix is a specific table layout that allows to visualize the algorithm performance, typically a supervised learning one.

A learning is supervised if the training and the testing set are formed by input-output pairs. So, the correct solution is known. Instead, a learning is unsupervised if both training and testing set are formed by only inputs and the right solution is unknown.

Confusion matrix is a special kind of contingency table, with two dimensions: "truth" and "predicted", and identical sets of "classes" in both dimensions.

Each row of the matrix represents the instances in the right class while each column represents the instances in the predicted class or vice versa.

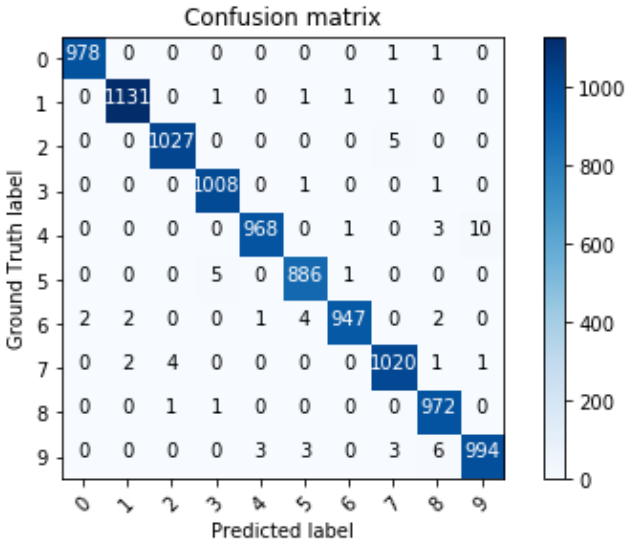


Figure 4.1: example of confusion matrix of 10 classes classification

If all the elements in a row are divided by the sum of all the elements in that row, then the result is the normalized confusion matrix, that in each cell (i, j) stores the probability that the estimated class is j given that the true class is i. Then the aim is to obtain the identity matrix as confusion matrix.

In this thesis, the aim is to recognize a single object then the confusion matrix is 2x2 [T. Fawcett. 2006], because there are only two classes: presence and absence of the object.

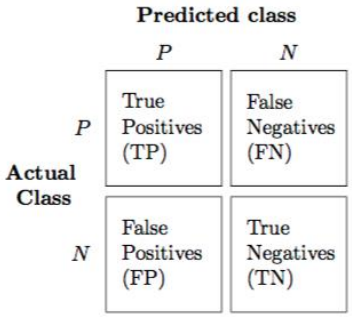


Figure 4.2: example of confusion matrix 2x2 (in this case)

Since the object detection is a supervised technique, it is already known if the image fed to the detector contains or not the object, then:

1. taken an image with the object:
    a. If the detector finds it, the detection is classified as TP
    b. Else the detection is classified as FN
2. taken an image without the object:
    a. If the detector recognizes the presence of the object, the detection is classified as FP
    b. Else the detection is classified as TN

Once the confusion matrix is obtained, it is possible to calculate the following parameters [L. David. 2008] to know the performance of the tested techniques:

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ *(eq. 4.1)*

  Accuracy measures the overall accuracy of the model

- $Precision = \frac{TP}{TP+FP}$ *(eq. 4.2)*

  Precision measures the accuracy of a single class, how accurate are the predictions

- $Recall = \frac{TP}{TP+FN}$ *(eq. 4.3)*

  Recall is the proportion of True Positive cases that are correctly predicted

- $Specificity = \frac{TN}{TN+FP}$ *(eq. 4.4)*

  Specificity is the proportion of True Negative cases that are correctly found

4.2 Intersection over Union (IoU)

Intersection over Union is a performance evaluation used to evaluate the accuracy of an object detector.

IoU is the ratio between the overlapping area of the ground-truth box (light blue) and the detector box (green) and the area of union of the same boxes.

The ground-truth boxes are the ones manually created with LabelImage on the test images.

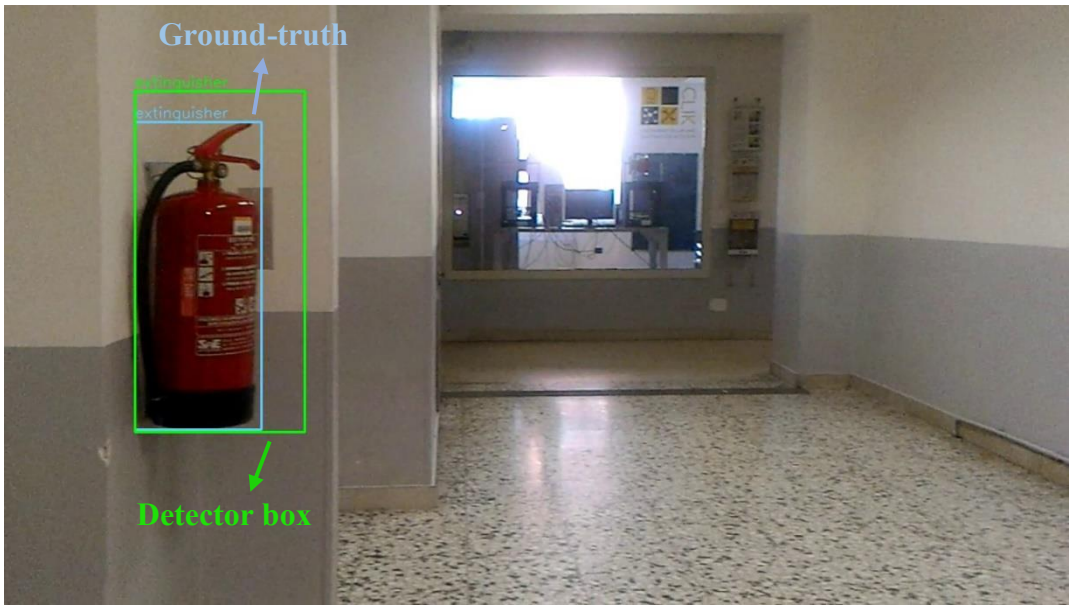


Figure 4.3: example of IoU of a test image

To clarify how the ratio is calculated, *Figure 4.4* has been inserted. It is evident that a complete and total match between predicted and ground-truth boxes is unrealistic. Then it is important that the performance evaluation gives higher scores to detector boxes for heavily overlapping with the ground truth.

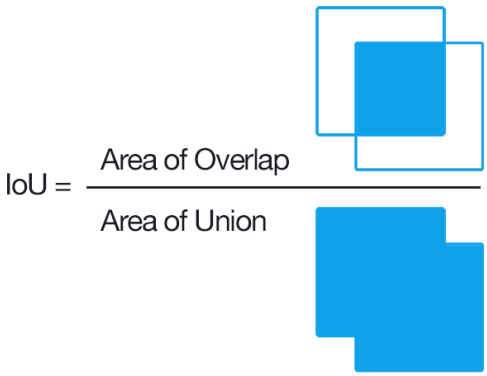


Figure 4.4: IoU

IoU is a number between 0 (no matching) and 1 (perfect matching) and in this thesis different threshold has been used to see the effect of the IoU on the mean Average Precision.

Figure 4.5: example of IoU values

4.3 Mean Average Precision (mAP)

The AP (Average Precision) is the average of the maximum precisions (*eq 4.2*) at different recall (*eq. 4.3*) values.

Both precision and recall depend on the threshold set for the IoU because a detector box with IoU under the threshold is classified as TN while over as TP.

Figure 4.6 shows the Precision-Recall curve, the more predictions are included the more recall increases, but precision goes up with TP and down with FP. So, a good classifier precision will stay high as recall increases and a poor classifier will have to take a large hit in precision to get a higher recall.

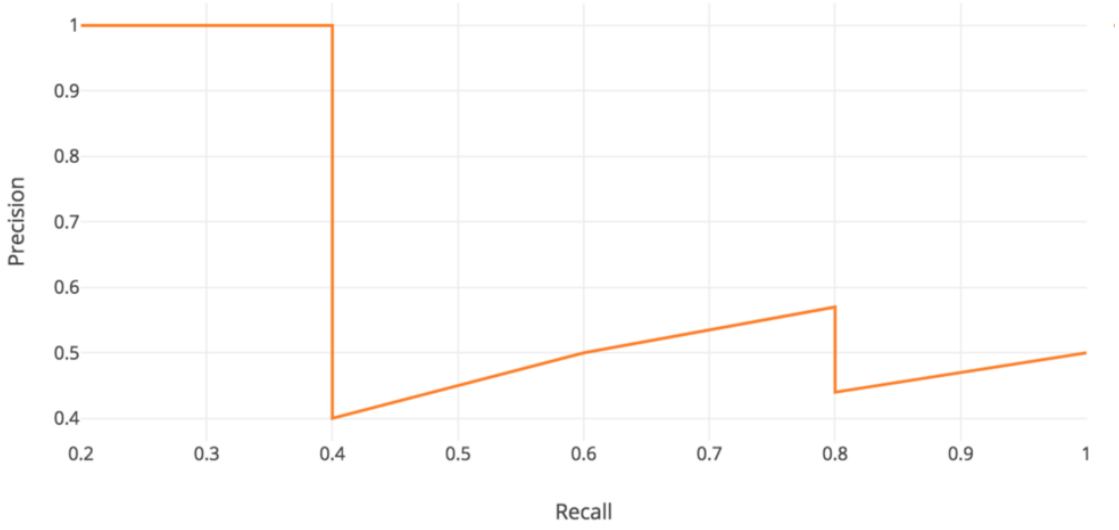


Figure 4.6: example of Precision-Recall curve

The Average Precision (AP) value is the area under the approximation of the orange curve. So, the area under the green curve of *Figure 4.7*.

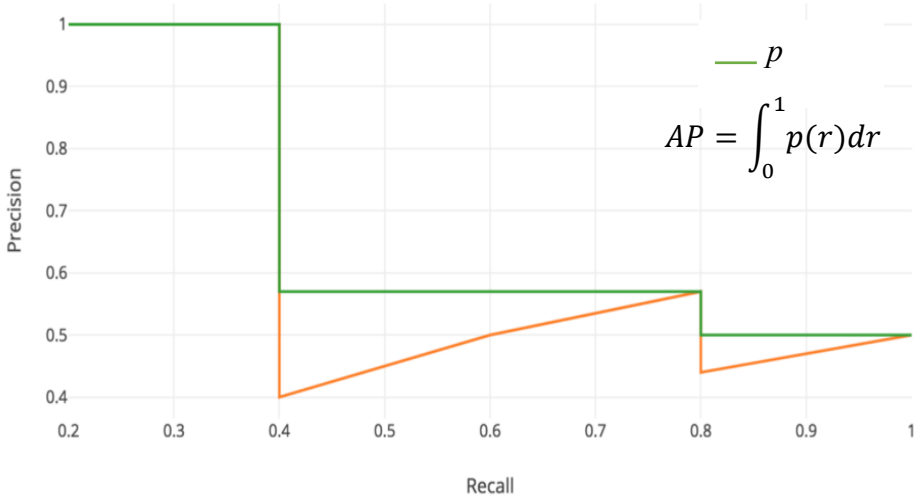


Figure 4.7: example of Average Precision-Recall curve

If the network has to evaluate different classes, an AP for each one is computed and the mean between all of the considered classes is the mAP value. Figures below show an example of AP (*4.8*) and mAP (*4.9*) computation on 85 images and 36 classes.

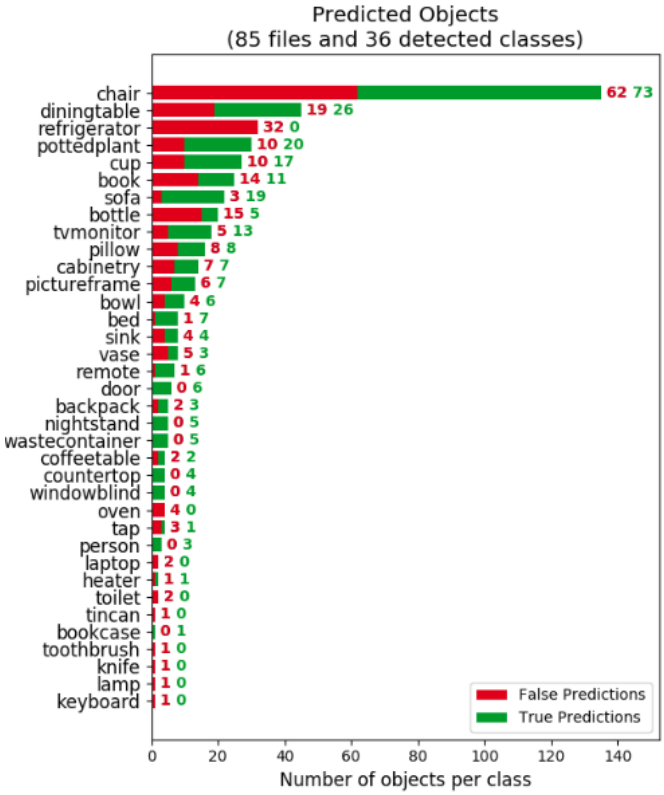


Figure 4.8: example of number of FP and TP predictions of 36 classes

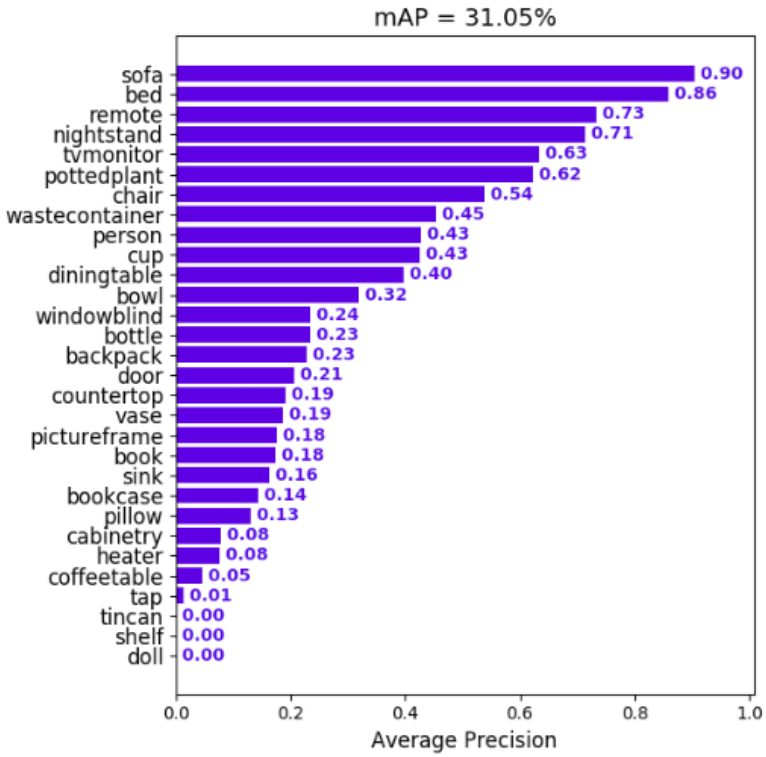


Figure 4.9: example of AP and mAP of 36 classes

In this thesis there is just a class, so AP and mAP are coincident.

5. Sensors

The field of this thesis study are the smart societies and therefore the sensors chosen are part of the category of low cost technologies. In particular it was not chosen to use only low cost sensors but also low cost platforms such as the Raspberry Pi. In fact, two cameras used are specifically designed to work with the Raspberry Pi, in particular: the Raspberry Pi Official Camera Module V2 and the Longruner Camera Module, which is a Night Vision camera.
The Raspberry Pi Official Camera is a high quality 8 Mp Sony IMX219 image sensor and the Longruner Camera Module is composed by OV5647 sensor and two infrared illuminators that work at 850nm. The second camera is a Night Vision camera that works well up to 8 meters, but if the goal is to reach a top-quality night vision results, then the ideal distance is between 3 and 5 meters.

First of all, however, a smartphone has been used because nowadays everybody has ones and it is a mass market ICT technology.

Finally, the MAPIR Survey3 camera has been also used since it is available in the DIATI laboratory. This camera is designed to make surveys and therefore could be used on object detection in agriculture field. The images returned by this last sensor are RGN (Red-Green-NearIR), so it seemed very interesting to use an Object Detection algorithm on a different image format since all the previous sensors return the classic RGB (Red-Green-Blue) images. Another important aspect that led us to choose to use this sensor is the fact that the MAPIR NearIR spectrum has a peak at 850 nanometres which is the wavelength at which the Night Vision camera illuminators work, therefore combining the illuminators with the MAPIR it was possible to see in low light condition.

Below a list, an image (*Figure 5.1*) and a table (*Table 5.1*) that contains the characteristics of the sensors, have been inserted.
The list of sensors is:

a) ASUS Zenfone 2 Deluxe, hereafter called smartphone
b) Raspberry Pi Official Camera Module V2, hereafter called Official Pi camera
c) Longruner Camera Module for Raspberry Pi, hereafter called Night Vision (NV) camera
d) MAPIR Survey3, hereafter called MAPIR

a) Smartphone

b) Official Pi Camera

c) NV Camera

d) MAPIR

Figure 5.1: sensors

Table 5.1: sensors characteristics

Sensor	Cost [€]	Max Resolution (Mp)	Radiometric channel	GPS	Internal data storage
Smartphone	300	13	RGB	yes	yes
Official Pi camera	27	8	RGB	yes, but external	no
Night Vision camera	26	5	RGB	yes, but external	no
MAPIR Survey3	600	12	RGN	yes	yes

6. Geometric camera calibration

In the previous chapter the sensors used were described and therefore as all optical sensors, the images could be affected by optical and sensor distortions.

In order to correct the image, a calibration is necessary. In object detection, this aspect could play a relevant role, because without this correction, the shape and size could be totally different with respect the "truth" used in the learning (§**9.2** and §**9.3**).

A camera characterized by a small hole, through which light rays pass and project an inverted image on the opposite side of the camera. The virtual image plane can be seen as a plane positioned in front of the camera and containing the vertical image of the shot (*Figure 6.1*).

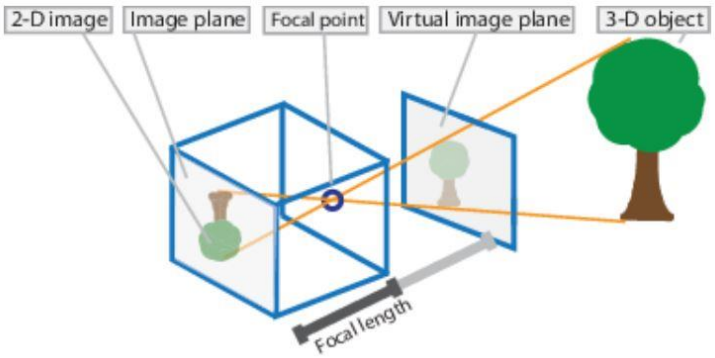


Figure 6.1: camera model

The camera parameters are represented in a matrix (4x3) called camera matrix, that maps the 3D shot in the 2D image plane. The calibration algorithm calculates the camera matrix using:

- the extrinsic parameters that represent the position of the camera in the 3D shot
- the intrinsic parameters that represent the X, Y, Z and the attitude

In particular, world points are transformed into camera coordinates using extrinsic parameters and camera coordinates are mapped in the image plane using intrinsic parameters (*Figure 6.2*). The equations are:

$$w[x\ y\ 1] = [X\ Y\ Z\ 1]P$$

$$P = \begin{bmatrix} R \\ t \end{bmatrix} K$$

where:

- w is the scale factor
- x, y are the image points
- X, Y, Z are the world points
- P is the camera matrix
- R is extrinsic rotation parameter

- t is translation extrinsic parameter
- K is intrinsic matrix

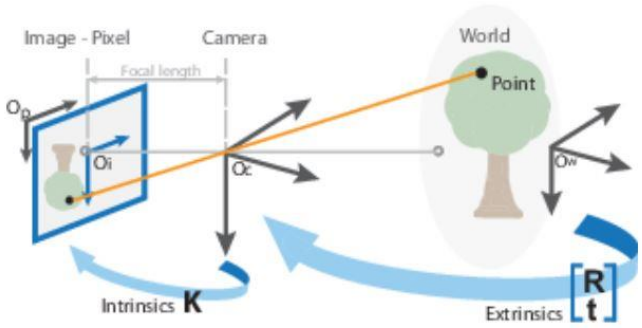


Figure 6.2: example of world points transformation into camera coordinates

Intrinsic parameters are specific to a camera. They include information like focal length (f_x, f_y) and optical centre (c_x, c_y). All these parameters are stored in the camera matrix, that is different for each camera.

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Extrinsic parameters correspond to rotation R and translation vectors t which translate a coordinate of a 3D point to a 2D coordinate system (*Figure 6.3*).

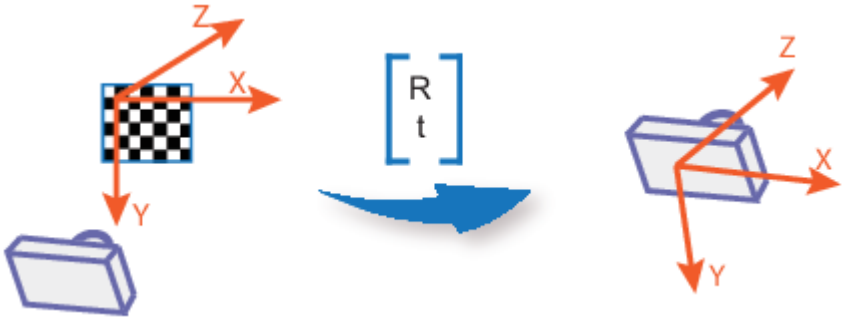


Figure 6.3: extrinsic camera parameters

The two major distortions introduced by cheap lens are radial and tangential and they can be solved using a mathematical model.

Due to radial distortion, straight lines will appear curved and the effect increases towards the edges of the image.

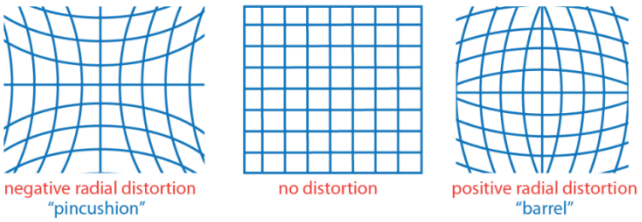


Figure 6.4: radial distortion

Tangential distortion occurs because image taking lens is not aligned perfectly parallel to the imaging plane. So, some areas may look nearer than expected (*Figure 6.5*).

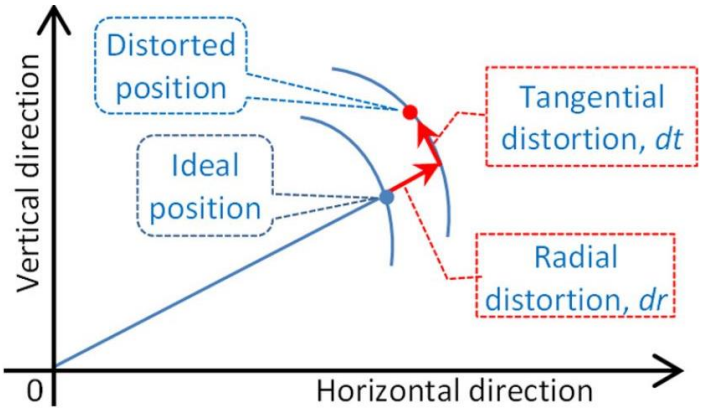


Figure 6.5: example of distortion effects

The following procedure was used to find the parameters:

1. a chessboard was chosen
2. the length of the side of a square of the chessboard was measured (10 cm)
3. multiple pictures of the chessboard have been taken with each camera, and in order to get good results, at least 20 different pictures varying the angles and distance should be taken.
4. all the pictures are given to the camera calibration tool and at least 10 images must be accepted to get reliable results.
5. the parameters are given as the output of the tool and then images without distortion can be obtained

It is important to say that sometimes images do not appear to be affected by any distortion even before the calibration, but the calibration is important to improve the accuracy of the camera.

6.1 Dataset for calibration

The pictures of the chessboard were taken in good light conditions and leaving the sensor setting parameters in their default values, except for the Official Pi camera and Night Vision camera in which a lower resolution than the default one was set.

Figure 6.6: example of chessboard photo

Table 6.1 contains all the information relative to the sets of images used to calibrate each camera.

Table 6.1: calibration datasets

Sensor	Num. of images	Resolution (pixel)	Type of image
Smartphone	27	4026x2304	RGB
Official Pi camera	33	720x576	RGB
NV Camera	33	720x576	RGB
MAPIR Survey3	24	3840x2160	RGN

To calibrate the cameras both the Camera Calibrator MATLAB App and OpenCV calibrateCamera() function have been used.

The results of both calibrations are described in the following sections.

6.2 MATLAB Camera Calibrator

In MATLAB there is a suite of calibration functions used by the Camera Calibrator that provides the workflow for camera calibration procedure.

Figure 6.7: MATLAB Camera Calibrator

Images of the chessboard can be added to a session and then once the accepted images are enough, calibration can be done. After evaluating the first results, their accuracy can be

improved adding or removing images and then make a new calibration or re-running the tool.

More in details, as shown by *Figure 6.8*, calibration accuracy can be improved by examining: reprojection errors (red box), pattern-centric view (light blue box) or viewing the undistorted image (green box).

Generally, mean reprojection errors of less than 1 pixel are acceptable and it should be as close to 0 as possible. In the histogram, highlighted by the red box, there are the values of the reprojection error for each image, then to improve accuracy the images related to higher error should be removed and then the tool must be run again. In the histogram there is also a dotted line which represents the mean reprojection error value.

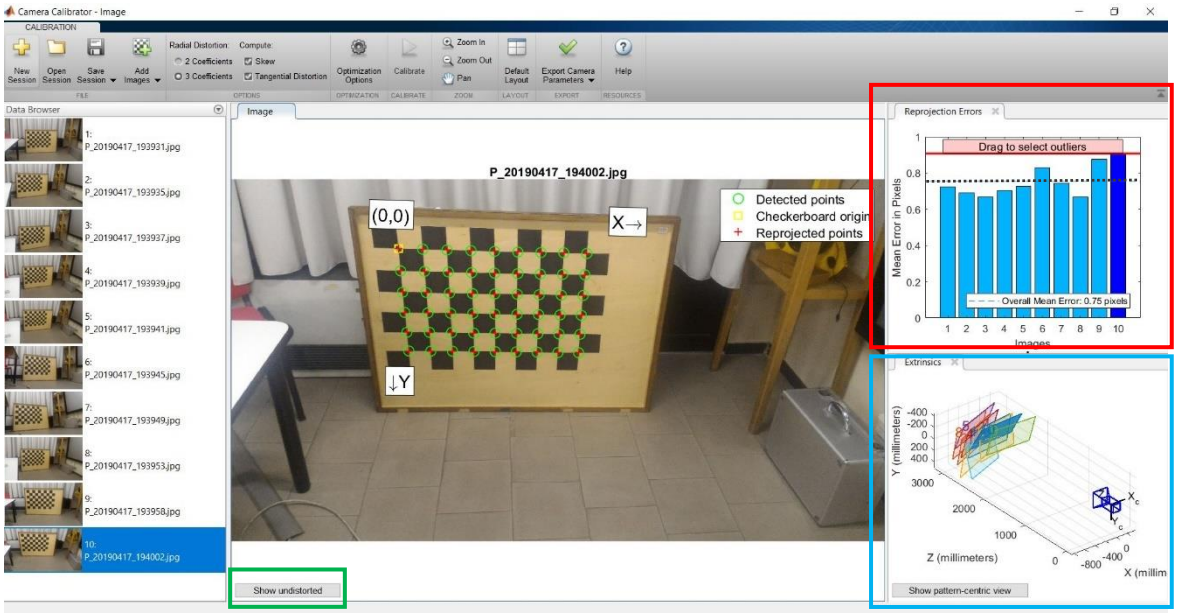


Figure 6.8: Camera Calibrator interface

The MATLAB Tool for camera calibration does not works well on the sets of images taken with the MAPIR and the smartphone cameras. The Camera Calibrator has been tested on different datasets created with the two sensors, but every time a lot of images are discarded and the reprojection error on the accepted images is always too close to 1 pixel.

In *Table 6.2* the results of the Camera Calibrator.

Table 6.2: MATLAB calibration results

Sensor	Mean Reprojection Error (pixel)	Accepted image	Input image
Smartphone	0,97	19	27
Official Pi camera	0,15	38	38
NV Camera	0,24	33	33
MAPIR	1,02	10	24

6.3 OpenCV Camera Calibration and 3D Reconstruction

OpenCV has a specific library called **cv2.calibrateCamera()** which estimate the camera matrix, distortion coefficients, rotation and translation vectors. Then new images can be undistorted.

The OpenCV calibration function has not GUI (Graphical User Interface) as the MATLAB Camera Calibrator, but it is however easily usable with python. In fact, through the **cv2.drawChessboardCorners()** function, it is possible to see the pattern of the chessboard recognized on each picture (*Figure 6.9*) and decide whether to accept or reject an image.

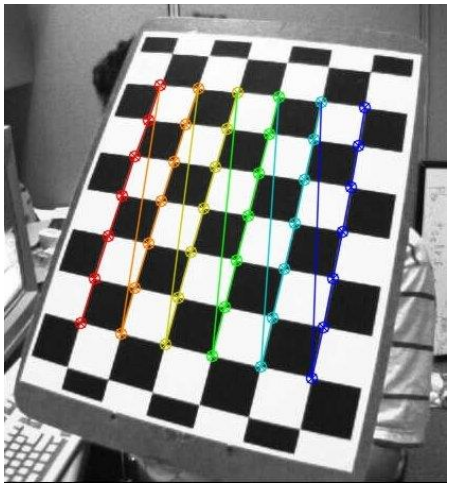


Figure 6.9: OpenCV chessboard pattern

OpenCV works well with all the sensors, since it works on the black and white images of the chessboard. Then if the original RGB or RGN have good contrast, it has no difficulties in findings the chessboard corners.

However, in the set of images taken with the MAPIR camera the function does not find the chessboard pattern in all the images but in 18 out of 24 totals. Instead in the set of images captured with the other sensors the chessboard's pattern is always recognized.

Below an example of the chessboard images before and after calibration.

(a)

(b)

Figure 6.10: NV Camera photo with (a) and without (b) distortions

In *Table 6.3* there are the mean reprojection error in pixel and the number of good images for each sensor.

Table 6.3: OpenCV calibration results

Sensor	Mean Reprojection Error (pixel)	Accepted image	Input image
Smartphone	0,20	26	27
Official Pi camera	0,03	38	38
NV camera	0,11	33	33
MAPIR	0,26	18	24

Below the camera matrix and distortion vector obtained for each sensor:

1. **Smartphone:**

$$matrix = \begin{bmatrix} 3{,}45 * 10^3 & 0 & 1{,}90 * 10^3 \\ 0 & 3{,}46 * 10^3 & 1{,}08 * 10^3 \\ 0 & 0 & 1 \end{bmatrix}$$

$$vector = [2{,}03 * 10^{-1};\ -5{,}69 * 10^{-1};\ -7{,}69 * 10^{-3};\ -1{,}67 * 10^{-2}; 7{,}39 * 10^{-1}]$$

2. **Official Pi camera**

$$matrix = \begin{bmatrix} 5{,}95 * 10^2 & 0 & 3{,}78 * 10^2 \\ 0 & 5{,}93 * 10^2 & 2{,}92 * 10^2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$vector = [2{,}00 * 10^{-1};\ -2{,}75 * 10^{-1};\ 6{,}27 * 10^{-5};\ -2{,}82 * 10^{-2};\ 2{,}31 * 10^{-2}]$$

3. **Night Vision camera**

$$matrix = \begin{bmatrix} 2{,}21 * 10^3 & 0 & 3{,}50 * 10^2 \\ 0 & 1{,}48 * 10^3 & 2{,}85 * 10^2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$vector = [-1{,}17;\ 1{,}29 * 10;\ 1{,}01 * 10^{-2};\ -1{,}08 * 10^{-2};\ -1{,}96 * 10^2]$$

4. **MAPIR**

$$matrix = \begin{bmatrix} 5{,}45 * 10^3 & 0 & 1{,}58 * 10^3 \\ 0 & 5{,}44 * 10^3 & 1{,}31 * 10^3 \\ 0 & 0 & 1 \end{bmatrix}$$

$$vector = [2{,}11 * 10^{-3};\ -3{,}74 * 10^{-1};\ 5{,}22 * 10^{-3};\ -6{,}24 * 10^{-3};\ 9{,}52 * 10^{-1}]$$

7. Training of the algorithms

This thesis aims to get a Object Detection algorithm able to work with low-cost where this means to indicate both the sensors used and described in Chapter 5 and the platforms such as the Raspberry Pi, on which we intend to use the algorithm.
To achieve the goal, it was first necessary to choose the software and to remain faithful to the low cost policy, open source software was chosen.
Obviously, it's not possible to do the training on the low cost platform, but it is necessary to use something more powerful like a computer, which does not need very expensive components. However, the characteristics required by the training for the hardware, depend on the technique chosen, in fact in this work two very different techniques for the implementation of Object Detection were analysed. An older model (Haar Cascade) that does not require particular hardware features and a newer one (Y.O.L.O.) that, instead, requires higher hardware performance.
However, in the following paragraphs the technical characteristics of the hardware used will be specified.

To implement the Haar Cascade, OpenCV (Open Source Computer Vision Library) has been chosen. It is an open source computer vision and machine learning software library and the distribution comes with a trained frontal face detector that works remarkably well. It is possible to train the algorithm on other objects and it works well for rigid and characteristic views object.

To implement Y.O.L.O., Darknet has been used. Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation. The Y.O.L.O. training using a pre-trained model can be done to recognize an object that is not already present in the basic dataset. Darknet installation makes the Y.O.L.O. re-training very easy.

In this thesis, the object to be recognized is the fire extinguisher. It is chosen because it is relevant in the indoor security field and present in a very large number of buildings. In addition, it seems to have all the characteristics to be recognize after appropriate training as it has a rigid, standard and well-defined shape.

7.1 Haar Cascade training

To build a Haar Cascade, "positive" and "negative" images are needed.

The "positive" images contain the chosen object. These can either be images that just mainly have the object, or they can be images that contain the object, and the ROI (Region Of Interest) where the object is, must be specified.

Instead, the negative images can be anything, except they cannot contain the chosen object.

With a single positive image and a command, a bunch of positive examples using the negative ones, can be created. In this way, the positive image will be superimposed on these negatives, and it should be angled and all sorts of things.

This technique can work well to detect one specific object, as the fire extinguisher. Instead if the aim is to identify, for example, all breeds of cats, thousands of unique images of cats are needed.

To train the cascade many parameters are needed: the vector file; the background file; the number of positive and negative images to use; the number of stages; the width and height in pixels of the maximum rectangle that can contain the fire extinguisher; the maximum False Alarm Rate and others.

It is important to use fewer positive images than all available because at each new stage the training will add some images. Using as many negatives as the half of the number of positive samples is a good practice

7.2 Haar Cascade training dataset

A fire extinguisher model, presents in the Politecnico di Torino, has been chosen and 6 photos (*Figure 7.1*) of it, from different point of view, have been taken with a mobile phone.

Figure 7.1: fire extinguisher (a, b, c, d, e, f)

Subsequently, the 6 photos have been resized into 100x200 pixels (width x height), using a python script and the OpenCV library tools. Then 600 positive samples were created from each of these in order to obtain 3600 positive images.

Thousands of negative images have been taken from *ImageNet,* where images of about anything can be found.
In this thesis, three categories have been chosen and all their images have been downloaded and used as negative samples. Once enough negative images have been collected, they have been resized to 300x300 pixels because they must be larger than the positive samples and not too big otherwise the *opencv_traincascade* can't process them.
At the end, 2290 negative images have been obtained.

Then for both positive and negative images, a description file is needed. For positives, this file (info.lst) has been built via the *opencv_createsamples* command. Each line of info.lst is:

- Name of the image
- The number of fire extinguisher inside the image (always 1)
- The top left corner, (x, y) in pixels, of the square inside which the fire extinguisher has been placed by the *opencv_createsamples*
- The size of the square (width, height) in pixels

Then the vector file (positives.vec), which contains all positives image for training, has been created via the *opencv_createsamples* command.

Instead, for the negative images' description file, called background file (bg.txt), a python script has been used and each of its line contains the path of an image.

Several trainings have been done over an i7 of 5^{th} generation and OpenCV 4.0.0.
Table 7.1 shows the settings and the duration of all the trainings done. Trainings can be divided into 4 cases which differ precisely in the characteristics of the images present in the training datasets (transparency and rotation). Then each case includes within it other subcases that differ in the maximum False Alarm Rate used.

Table 7.1: Haar Cascade trainings

Training	Max False Alarm Rate	Max rotation (rad)	Num. of stages	Acceptance Ratio (last stage)	Time
First trial					
1	0,5	0,5	10	$4{,}305*10^{-3}$	4d
Second trial					
2	0,4	0,5	10	$2{,}32*10^{-4}$	1d 11h
3	0,3	0,5	10	$7{,}23*10^{-5}$	1d 16h
			9	$1{,}56*10^{-4}$	1d 9h

Third trial					
4	0,2	0	10	$1,81*10^{-5}$	2d 10h
			7	$2,02*10^{-4}$	1d 10h
5	0,3	0	6	$2,10*10^{-5}$	9h
6	0,4	0	4	$8,71*10^{-4}$	2,5h
7	0,5	0	3	$5,73*10^{-3}$	1h
Fourth trial					
8*	0,2	0,2	5	$1,05*10^{-4}$	9h
9*	0,3	0,2	8	$1,27*10^{-4}$	12h
10*	0,4	0,2	8	$2,42*10^{-4}$	10h

7.2.1 First trial

The set of positives images is composed by 3600 images (*Figure 7.2*). For each of the photos of *Figure 7.1,* 600 positive samples have been created with a maximum rotation of 0,5 radiant on the x, y and z axis.

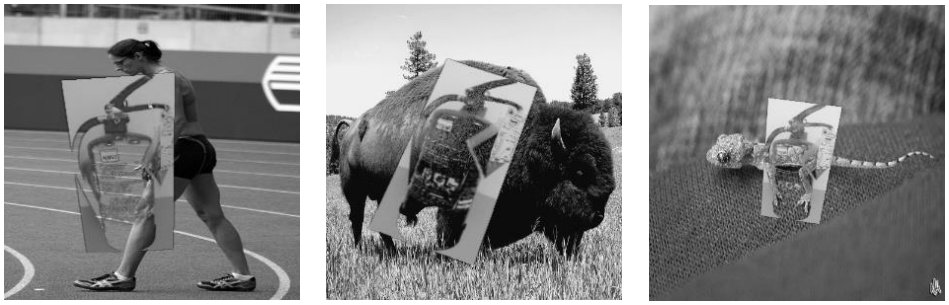


Figure 7.2: positive images (1st trial)

The maximum False Alarm Rate is equal to 0,5 and the number of stages is 10. The training lasted 4 days.

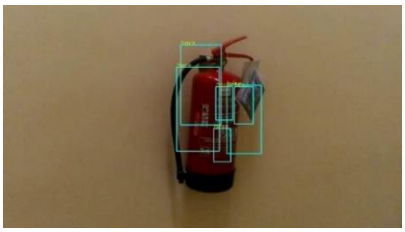
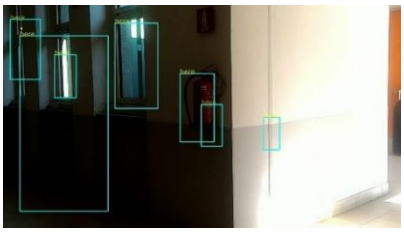
In *Figure 7.3a,* the fire extinguisher is in front of the camera and on a homogeneous wall. In this simple scenario, it is assumed that the fire extinguisher is perfectly recognized, but it does not.

Figure 7.3b shows the fire extinguisher in a non-optimal light condition and in a more complex scenario. In *Figure 7.3c* the fire extinguisher is in good light condition, but on a more complex background than *Figure 7.3a.*

The results (*Figure 7.3*) are very bad, because the cascade found fire extinguisher everywhere and a lot of boxes overlap.

* trainings who's the mAP of Chapter 9 refers to

So once obtained those results, some changes have been done and new trainings performed.

(a)   (b)

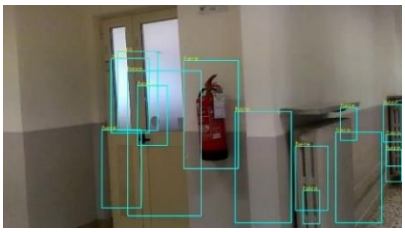
(c) 

Figure 7.3: first cascade output (a, b, c)

7.2.2 Second trial

The transparency present in the positive images (*Figure 7.2*) of the previous training has been removed and the maximum False Alarm Rate has been reduced to 0,4, while the maximum rotation remained the same. The training lasted 1 day and 11 hours for 10 stages. The results seem a little bit better but there are still lots of false detection.

Figure 7.4: positive images (2nd trial)

The detection in *Figure 7.5a* is good, but in the others the results are not well enough.

(a)   (b)

(c) 

Figure 7.5: second cascade output (a, b, c)

Then a new training with a maximum False Alarm Rate of 0,3 and the same dataset has been done. The training duration is 1 day and 16 hours for 10 stages.

The results (*Figure 7.6*) seem a little bit better in good light condition, but if there is little light the extinguisher is not seen.

(a)   (b)

(c) 

Figure 7.6: third cascade output (a, b, c, 10 stages)

However, for this new training 10 stages seems to many, because the *AcceptanceRatio* parameter is of the order of 10^{-5} in the last stage and a good guideline is to train not further than 10^{-5}, to ensure the model does not over train on the training data. *AcceptanceRatio* is used to determine how precise the model should keep learning and when to stop. By default, its value is set to -1 to disable this feature but it is possible to set it equal to any reasonable small number as, 10^{-5}, to stop the training when its value becomes smaller than the chosen number.

The results (*Figure 7.7*) are better than the results obtained with 10 stages (*Figure 7.6*), since the side effect of overfitting is removed by the 9 stages cascade.

Comparing *Figure 7.7c* and *Figure 7.6c,* it is evident that the reduction of the maximum False Alarm Rate, during the training, has reduce the false detection and better results has been obtained in general.

(a)   (b)

(c) 

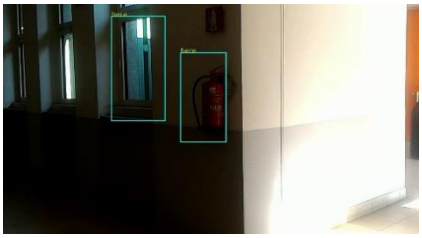
Figure 7.7: third cascade output (a, b, c, 9 stages)

However, the results are not yet well enough, so a new training with a maximum False Alarm Rate equal to 0,2 has been done and its duration is 2 days and 10 hours for 10 stages. But again 10 stages are too many and overfitting affects the results *(Figure 7.8 vs. Figure 7.9)*.

(a)   (b)

(c) 

Figure 7.8: fourth cascade output (a, b, c, 10 stages)

(a)   (b)

(c) 

Figure 7.9: fourth cascade output (a, b, c, 7 stages)

Those results are not good enough, so new changes have been done.

7.2.3 Third trial

A new dataset of 3600 positive images without rotation has been used and different False Alarm Rates have been tested.

Figure 7.10: positive images (3rd trial)

The training of images without rotation is much faster and the number of features used by each stage is smaller.

For these trainings the *AcceptanceRatio* values has been set lower than 10^{-5} to avoid overfitting.

In *Figure 7.11* the results related to a maximum False Alarm Rate of 0,3, the training duration is 9 hours for 6 stages and the fire extinguisher is never detect

(a)

(b)

(c)

Figure 7.11: fifth cascade output (a, b, c, 6 stages)

Then a new training with a maximum False Alarm Rate equal to 0,4 has been performed (*Figure 7.12*). The training lasted 2 hours and a half for 4 stages.

(a)

(b)

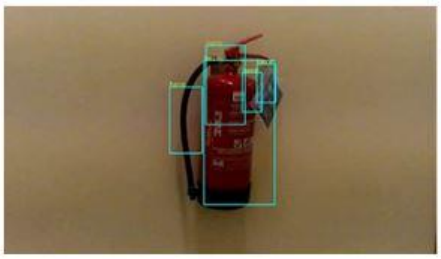
(c)

Figure 7.12: sixth cascade output (a, b, c, 4 stages)

Since the fire extinguisher is not detected most of the time, a new training with a maxFalseAlarmRate of 0,5 have been done (*Figure 7.13*) and its duration is 1 hour and a half for 3 stages.

From *Figure 7.13,* it is evident that the results are bad, because there are a lot of false detection and the box that should identify the fire extinguisher does not encose it well.

(a)



(b)

(c)

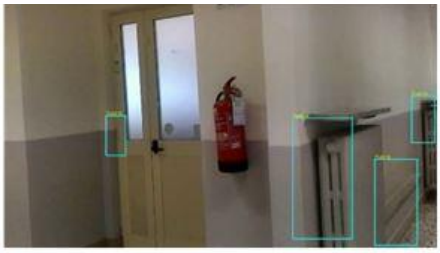


Figure 7.13: seventh cascade output (a, b, c, 3 stages)

Observed those results a new dataset was created and new trainings were carried out.

7.2.4 Fourth trial

Figure 7.14 positive images (4th trial)

A new dataset of 3600 positive images with a maximum rotation of 0,2 radiant on the x, y and z axis has been used and different False Alarm Rate have been tested.

The maximum False Alarm Rate tested are: 0,2 (*Figure 7.15*); 0,3 (*Figure 7.16*); 0,4 (*Figure 7.17*). The results related to 0,2 are the best ones in good light condition, but only with a maximum False Alarm Rate equal to 0,4 the fire extinguisher is detected in the darkest scenario

(a) (b)

(c)

Figure 7.15: eighth cascade output (a, b, c, 8 stages)

(a) (b)

(c)

Figure 7.16: ninth cascade output (a, b, c, 8 stages)

(a)   (b)

(c) 

Figure 7.17: tenth cascade output (a, b, c, 8 stages)

7.2.5 New model detection

During the performance tests of the previous cascades, it was noted that in the Politecnico di Torino, also another model of fire extinguisher very similar to that of *Figure 7.1* is recognized. The two extinguishers are analog in shape and the most evident differences are: the position of the black wire and the color of the base.

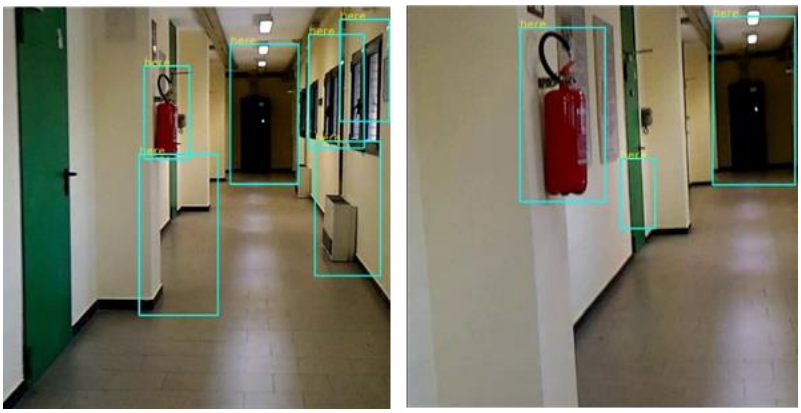
(a) (b)

Figure 7.18: (a) new extinguisher model, (b) previous extinguisher model

In *Figure 7.19* and *7.20* there are the outputs related to the cascade of the second trial (§**7.2.2**) with maximum False Alarm Rate equal to 0,4 and 0,2 respectively.

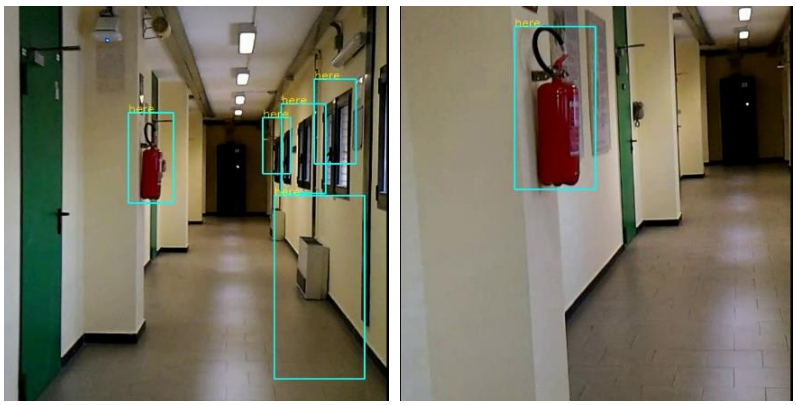
There are a lots of false detection, but the interesting thing is the fact that also this model of fire extinguisher is well squared.

It is clear that the performance of each classifier varies greatly depending on the scenario and the lighting conditions.



(a) (b)

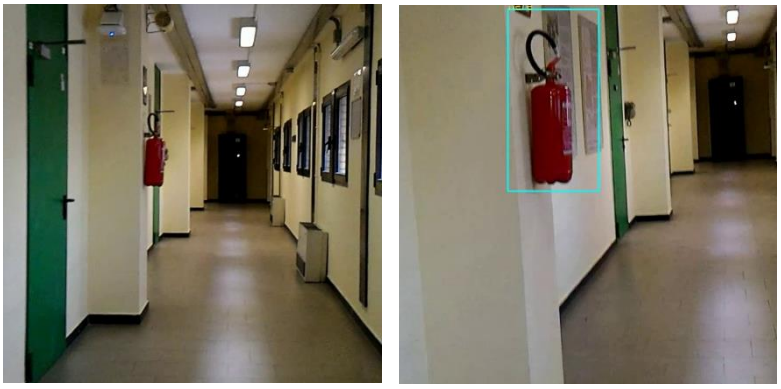
Figure 7.19: second cascade output (a, b)



(a) (b)

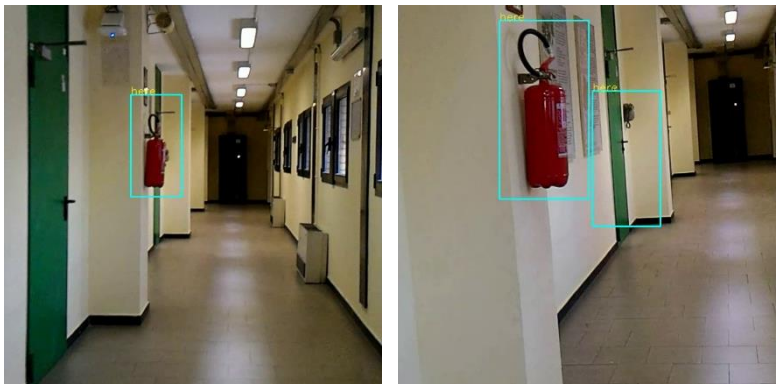
Figure 7.20: fourth cascade output (a, b, 7 stages)

Also, the cascades presented in the fourth trial (§**7.2.4**) have been tested on this new fire extinguisher model and they work better than the previous ones. The maximum False Alarm Rate increases from *Figure 7.21* to *7.23*.



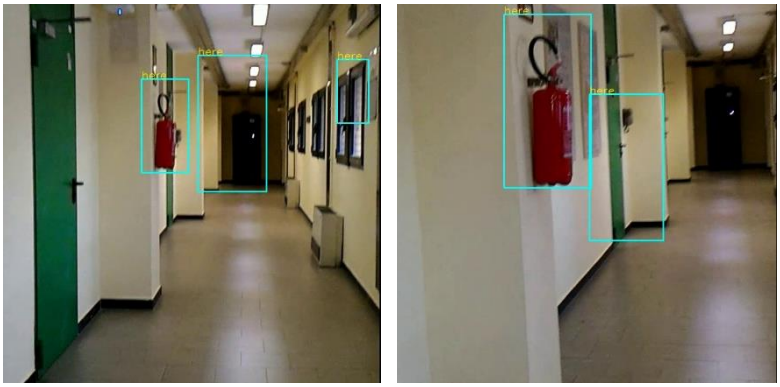
(a) (b)

Figure 7.21: eighth cascade output (a, b)



(a) (b)

Figure 7.22: ninth cascade output (a, b)



(a) (b)

Figure 7.23: tenth cascade output (a, b)

However, these results are unexpected because they are like the previous ones even if the cascade has never seen this second model.

7.3 Y.O.L.O. re-training

A pre-trained model, on the COCO dataset, has been used because training a network starting from scratch would have been too long and resource intensive.

The re-training has been done over a computer with a GeForce GTX 670MX and an i7 of the 7th generation.

Since, the followed procedure is based on a pre-trained model some changes need to be done. Firstly, the number of classes has been changed from 80 to 1 (extinguisher), since the aim is to detect only the fire extinguisher. In this way the network output referred only to a specific object resulting more performing and accurate. Also, the filters parameter has been changed and set to 18 since it depends on the number of classes. Another change is the batch size that has been set equal to 64 with a subdivision of 32.

7.4 Y.O.L.O. re-training dataset

For the re-training 400 images has been manually labelled with LabelImage, 320 images have been used for training and the remaining ones for testing.

LabelImage is a graphical image annotation tool through which you can create a box (green box) around the object and assign it a label (red box).

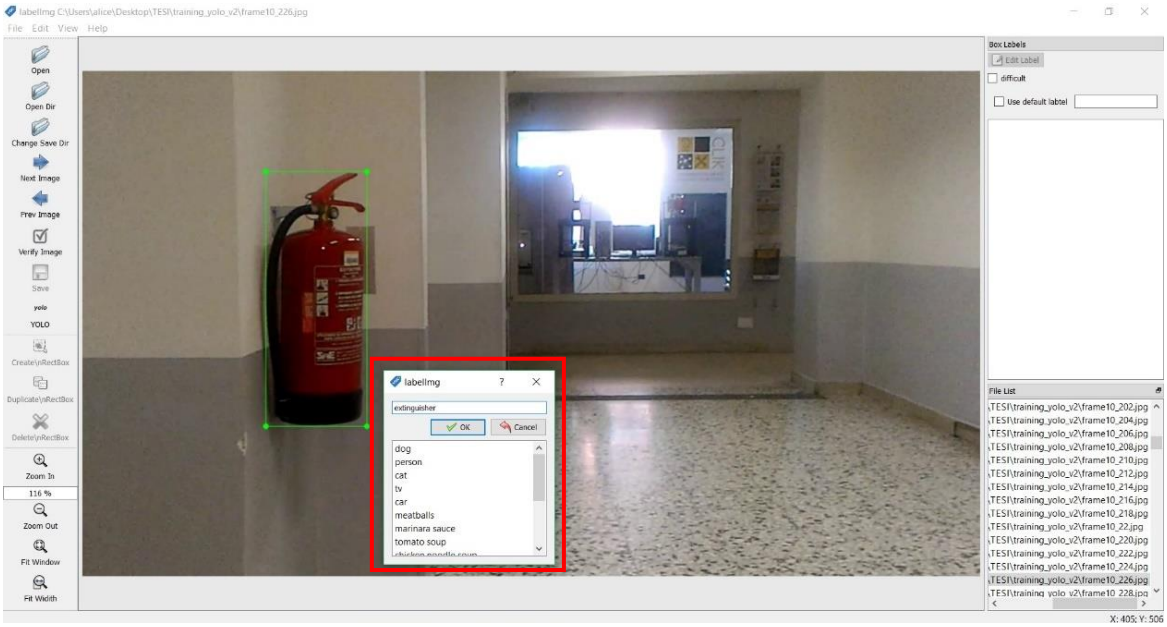


Figure 7.24: example of labelling using LabelImage

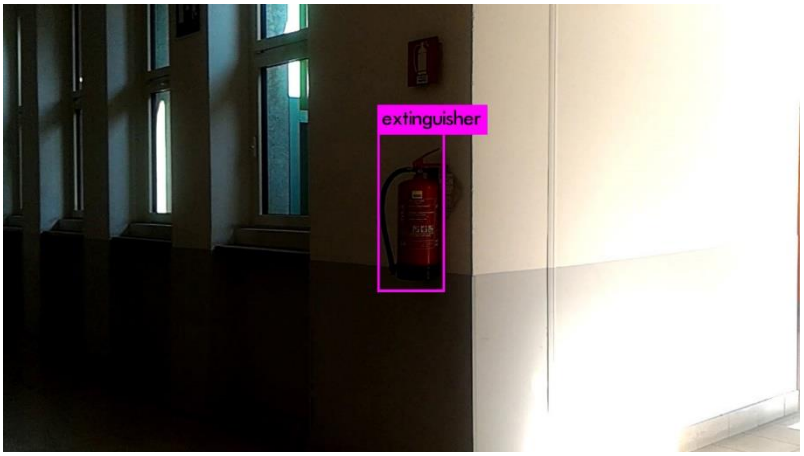
Then a text file (.txt) with the same name of the image (.jpg) can be saved. Each line of these text file is:

- name of the class
- the x (pixels) of the centre of rectangle drawn, divided by the width of the image (1280 pixels)
- the y (pixels) of the centre of rectangle drawn, divided by the height of the image (720 pixels)
- the width (pixels) of the rectangle drawn, divided by the width of the image
- the height (pixels) of the rectangle drawn, divided by the width of the image

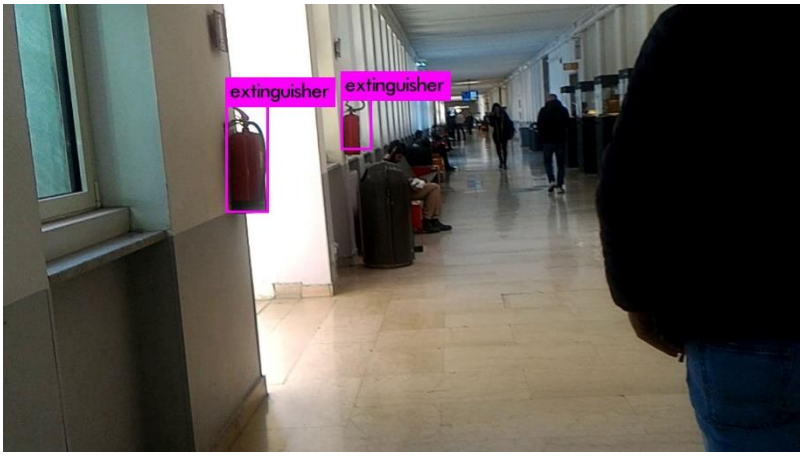
The re-training has lasted 13 hours for 5000 epochs.

From *Figure 7.25*, it is evident that Y.O.L.O. predictions enclose better the two fire extinguisher models. In *Figure 2.25a* is shown the extinguisher model, used in the training, in non-optimal light condition, while in *Figure 2.25b* there are both extinguisher models in good light condition. The closest extinguisher is the model used in training images, the farthest model is the similar one, also recognized by the Haar cascade (§**7.2.5**).

Moreover, the predictions made by Y.O.L.O. do not give rise to all the false detections given by the Haar Cascade.



(a)



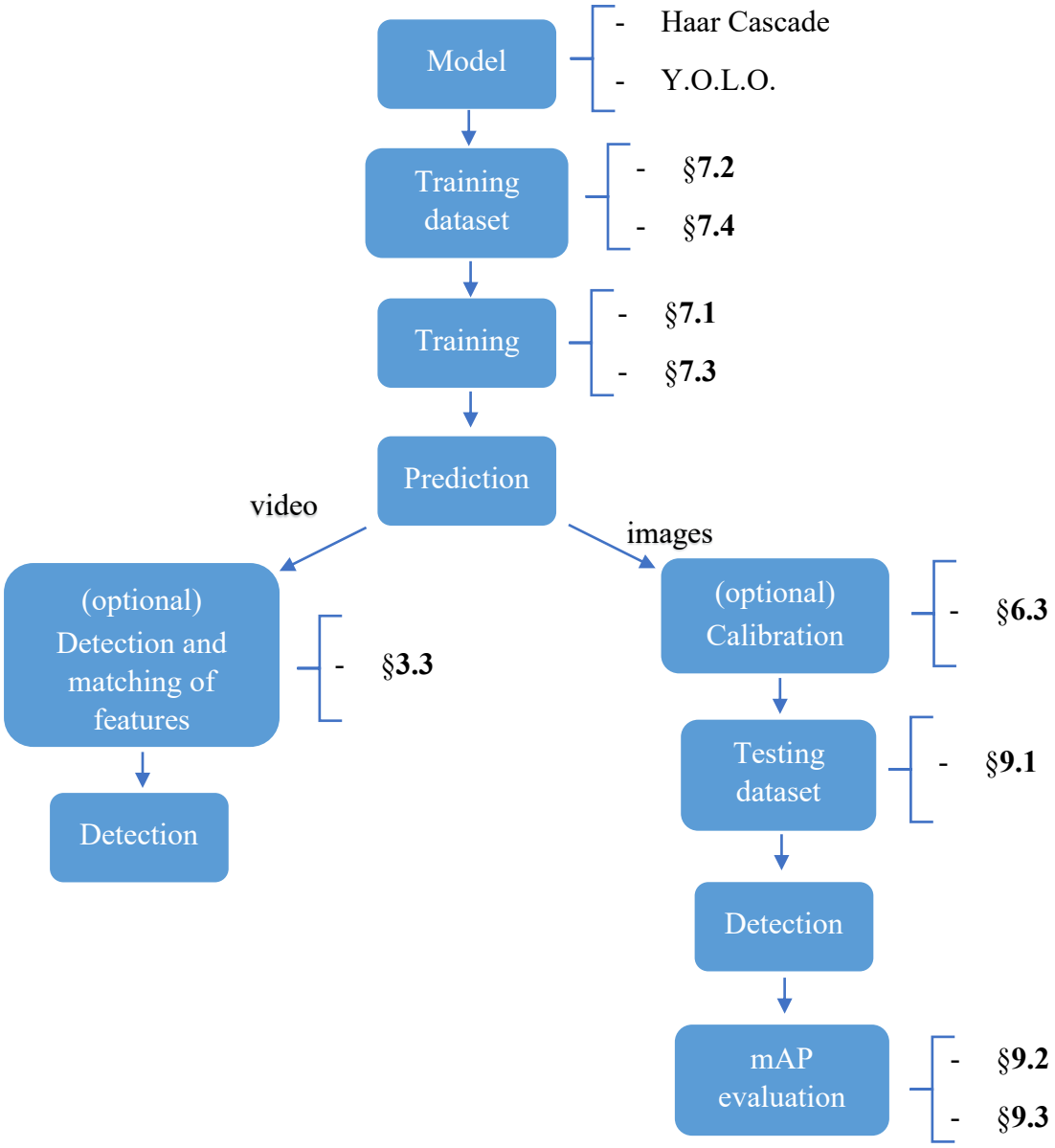
(b)

Figure 7.25: Y.O.L.O. re-training predictions (a, b)

8. Development of the tools used

As previously mentioned, the purpose of the thesis is to get an Object Detection algorithm able to find the extinguisher in images or videos. To achieve the goal, a model was first chosen and after the training dataset was created. The training dataset can be composed of images (.jpg) of any type: grayscale, RGB, RGN, high or low resolution, with or without distortion, etc. Once the dataset is obtained, the training of the chosen model can be done.

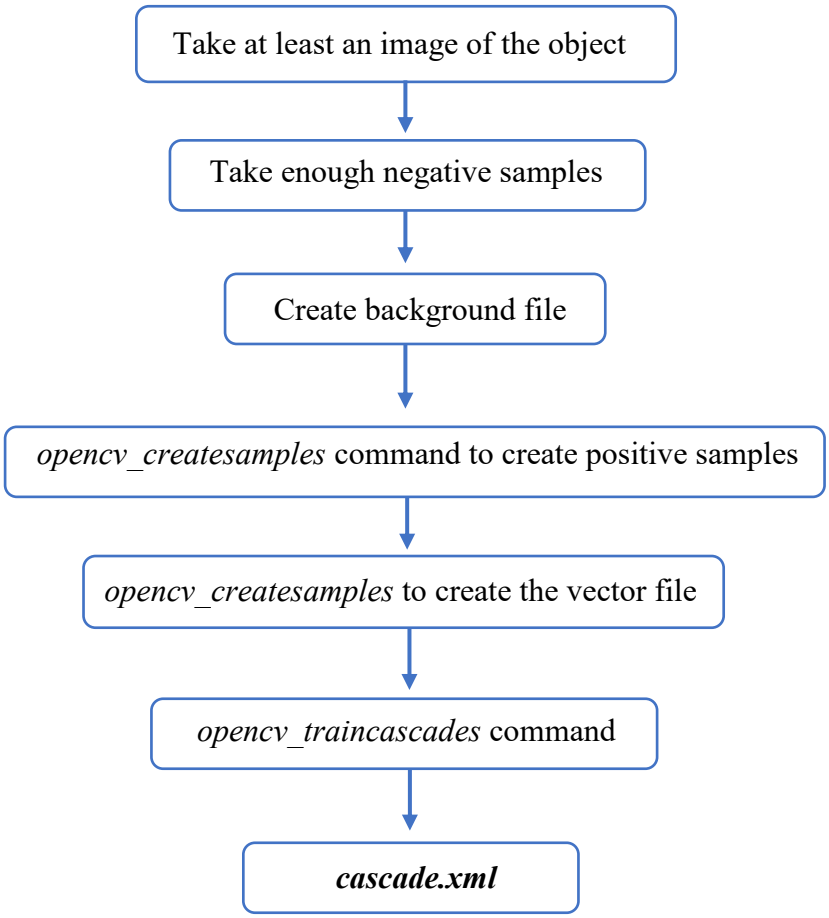
After the training, you can choose whether to apply the algorithm on a video stream or on a set of images. If you choose the video, you can optionally apply a detection and matching of feature among frames, otherwise you have to create a testing dataset of images that can have the same or different characteristics than those of the training dataset.



8.1 Haar Cascade training

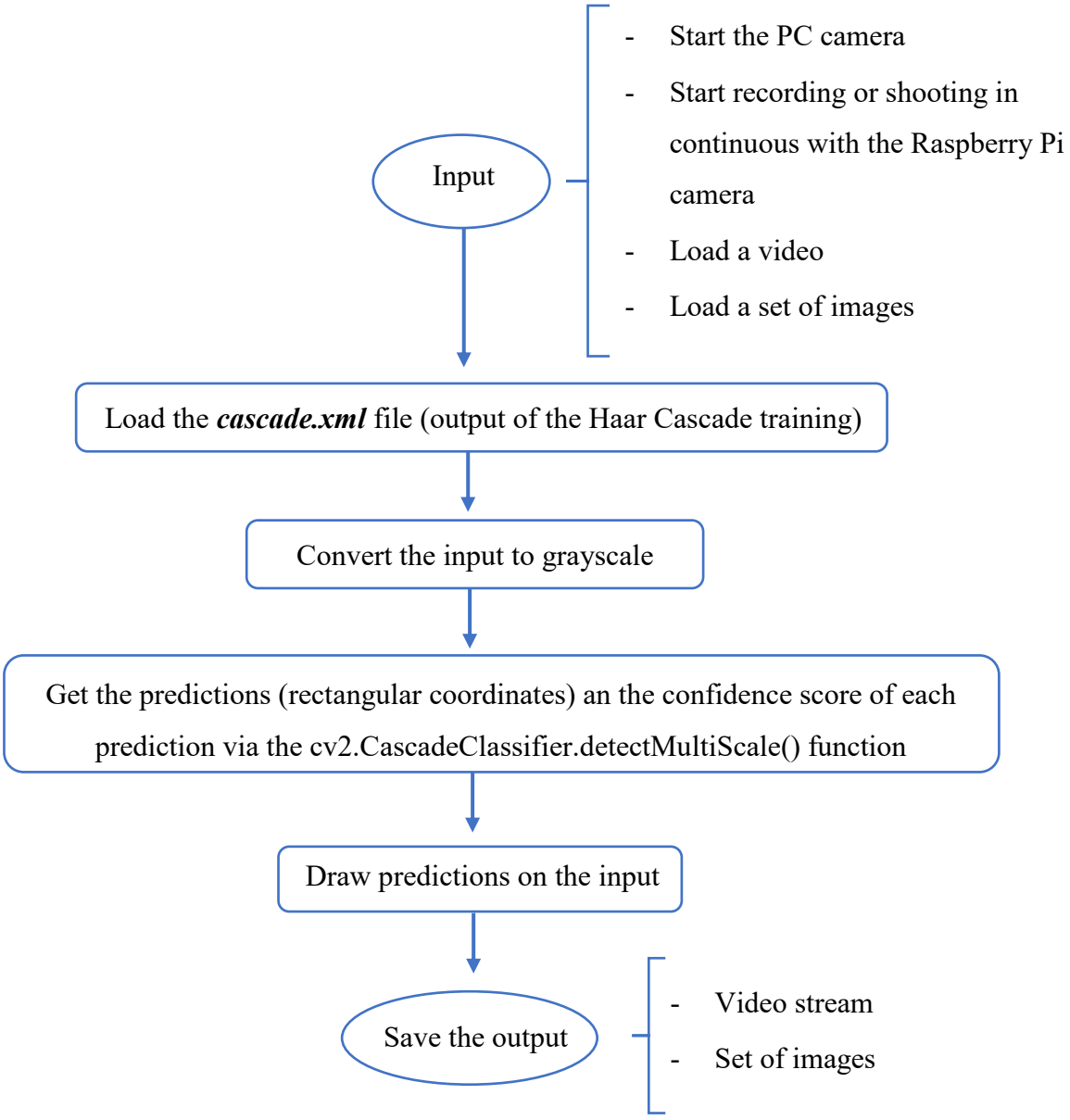
Firstly, the OpenCV library has been installed and some tools have been used to: resize the fire extinguisher photos, convert to grayscale both positive and negative samples, create the text files. Then three commands have been executed from the command line:

i. *opencv_createsamples* to superimpose the photos of the extinguisher on the negative samples
ii. *opencv_createsamples* to get the vector file
iii. *opencv_traincasacdes* followed by the setting described in the previous chapter (§**7.1**) to start the training



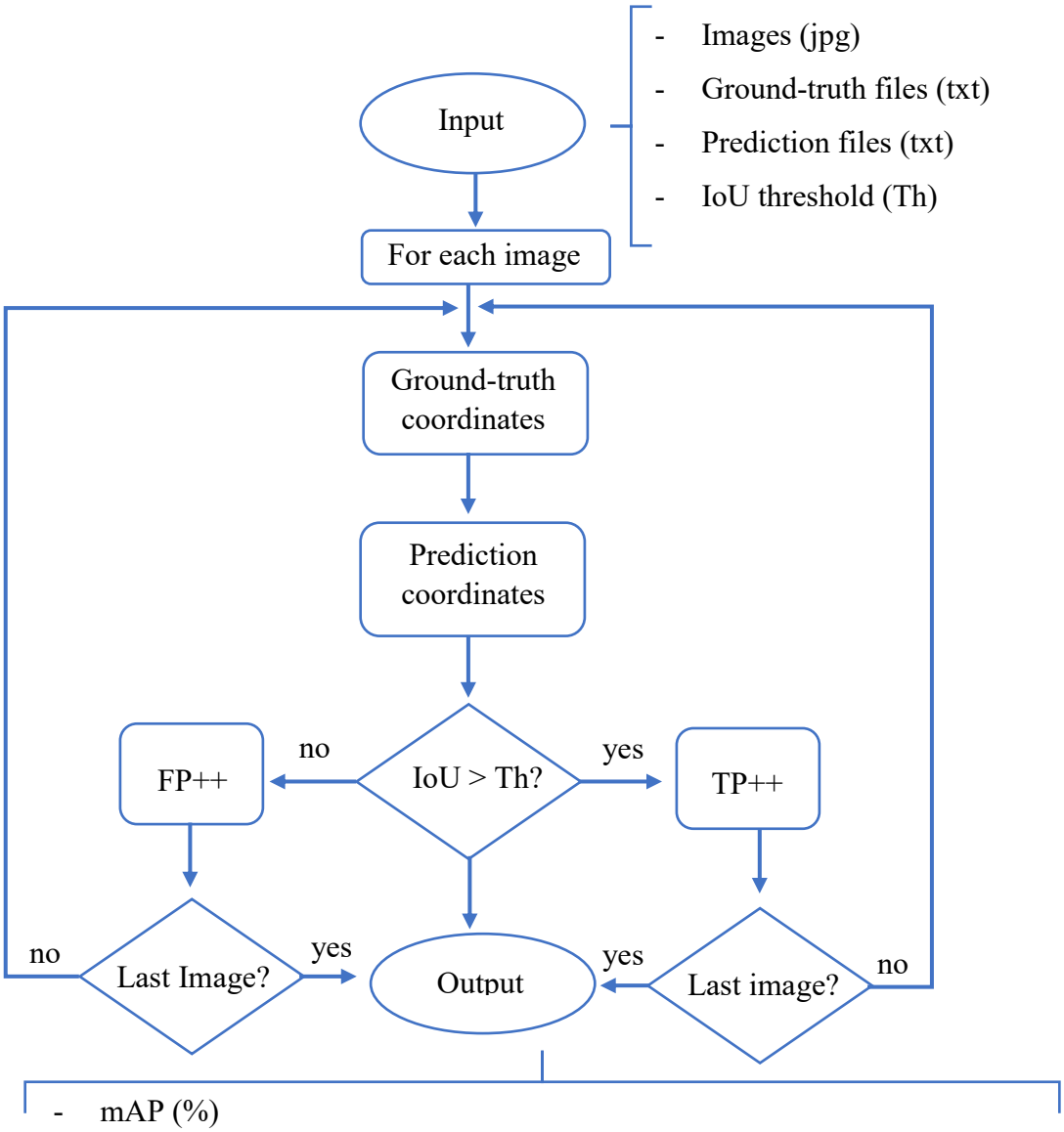
8.2 Haar Cascade testing

The detection using the ***cascade.xml*** file is done by OpenCV function *cv2.CascadeClassifier.detectMultiScale()*. Chosen the input, image or frame of a video, it is firstly converted to grayscale and then given to the function named above. The function returns the predictions in a matrix whose rows contain the coordinates of the prediction boxes. Using the coordinates and others OpenCV tools the predictions can be drawn.



8.3 mAP evaluation on Haar Cascade predictions

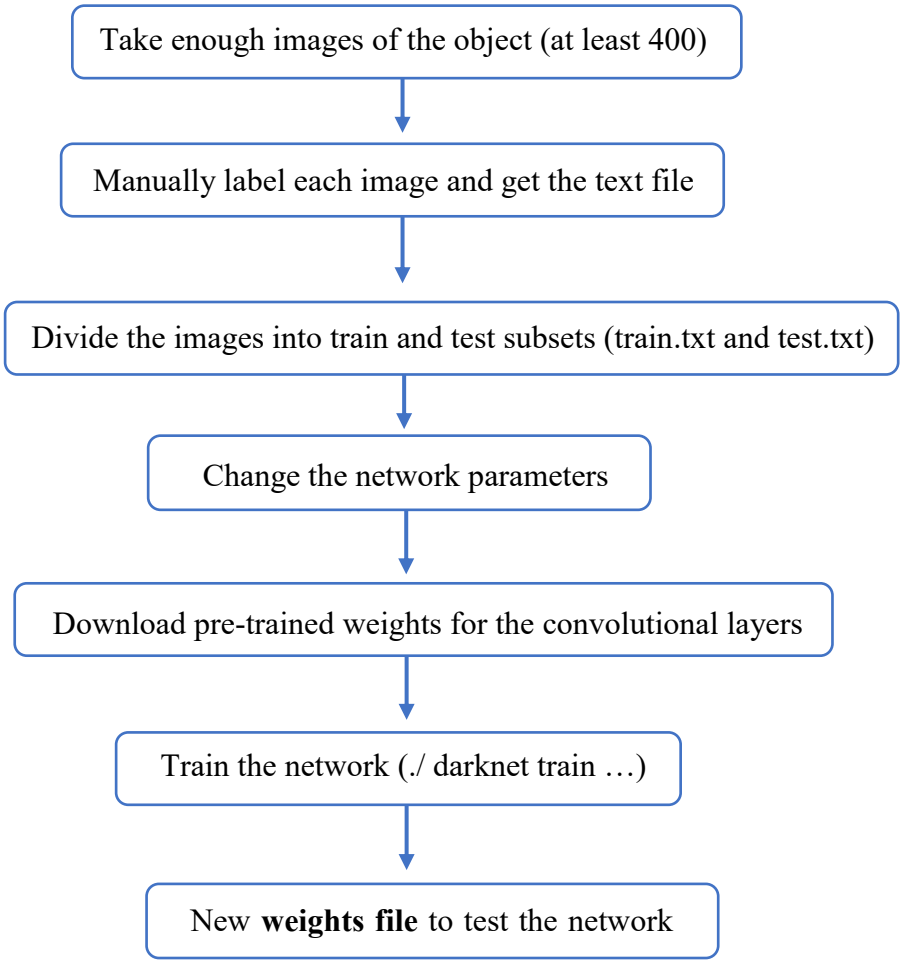
A set of images has been created with each sensor. Then each image has been manually labelled and the ground truth files obtained. Subsequently on these images, the detection was made, and the prediction files were obtained. Then choose a threshold for the IoU parameter, the prediction files have been compared with the ground truth ones, to estimate the performance.



8.4 Y.O.L.O. re-training

Firstly, Darknet and all the necessary packages has been installed. The re-training must be done on a machine that has a GPU with CC greater than or equal to 3.0 [https://en.wikipedia.org/wiki/CUDA#GPUs_supported].

Then the images chosen for the training have to be manually labelled and then divided into training and testing sets. After that, the parameter of the pre-trained network have been changed, the pre-trained weights file of YOLOv3 downloaded and via the *./darknet train* command the training phase has been started.

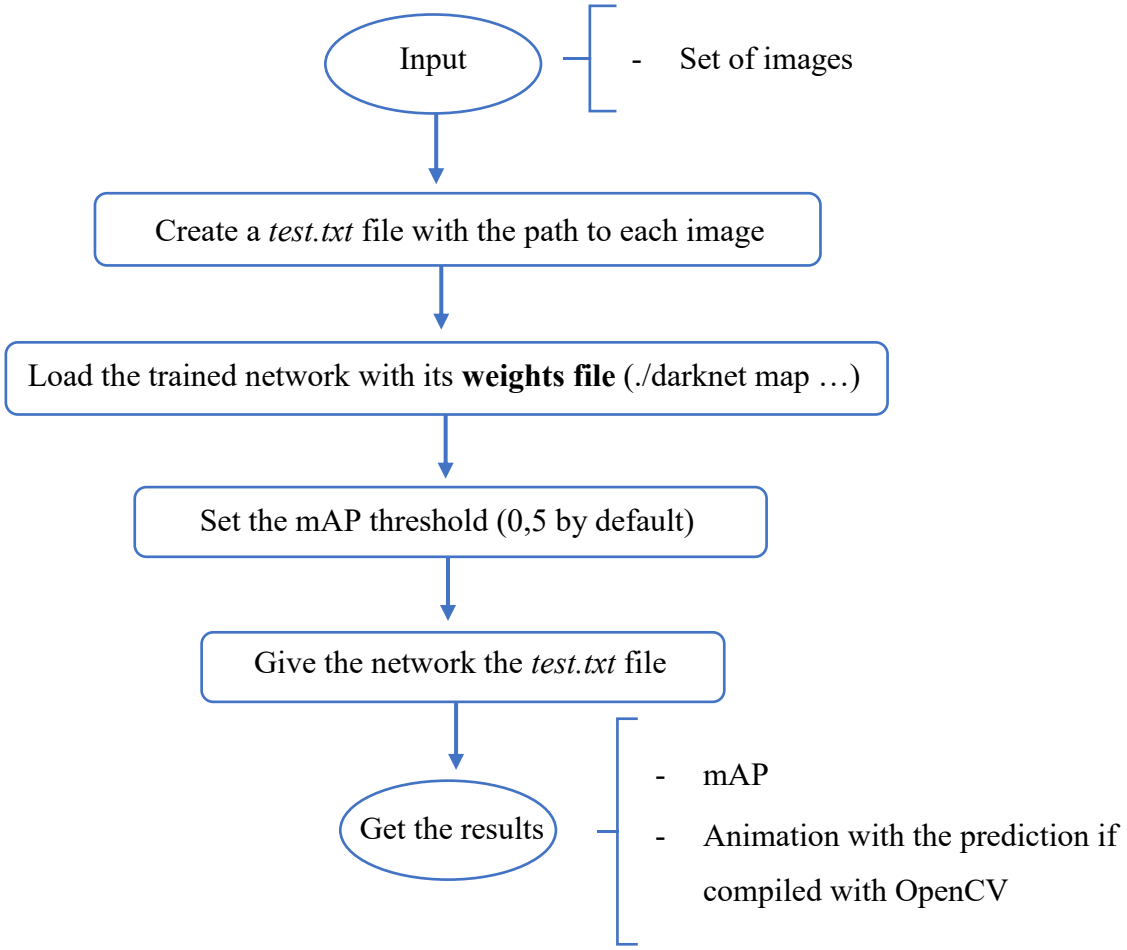


8.5 Y.O.L.O. testing and mAP evaluation

Taken a set of images they have to be manually labelled in order to get the text files.

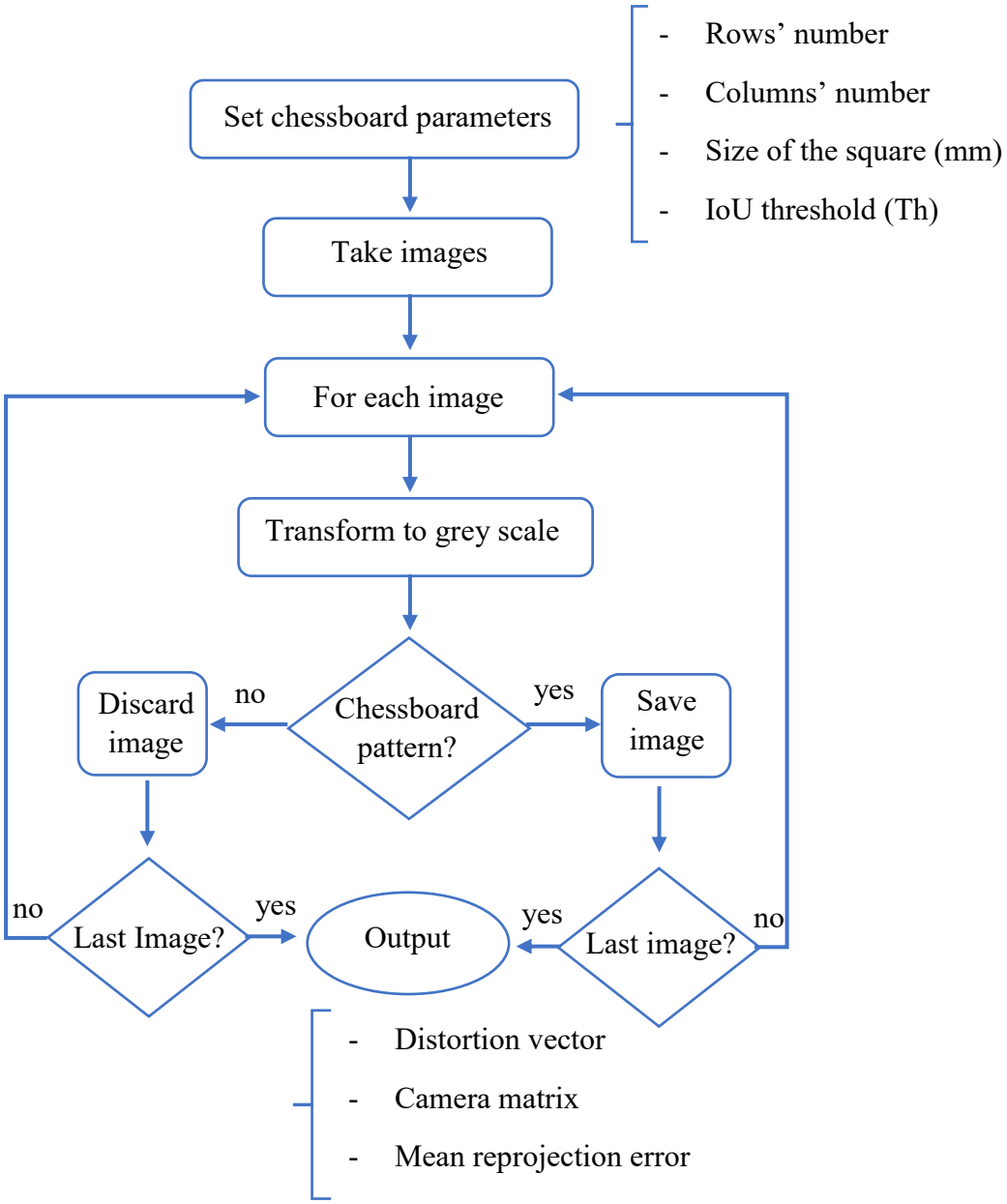
Then both images and text files are given as testing dataset to the re-trained network.

After that, the pre-trained network with its new **weights file** needs to be loaded via *./darknet map* command. As results it returns the mAP evaluated on the testing dataset.



8.6 Calibration of the sensors

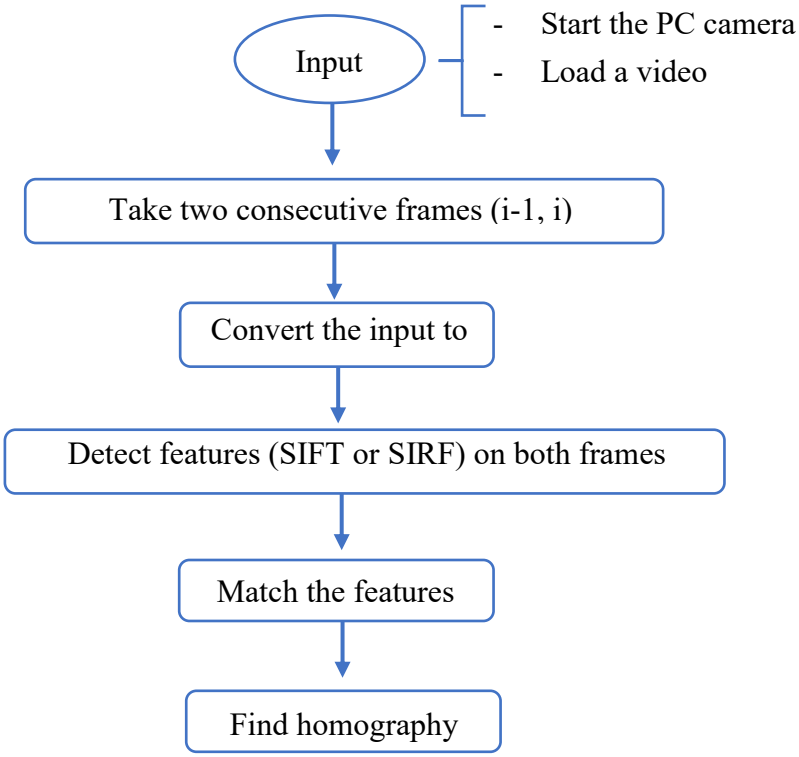
A set of chessboard photos has been created with each sensor (§**6.1**). Then using OpenCV library tools each image is converted to grayscale and the chessboard pattern is searched. If the pattern is found, the images can be either accepted or discarded at choice. At the end when all images have been considered the distortion parameters of the camera and the mean reprojection error can be estimated.



8.7 Feature detection and matching on video stream

This implementation has been done to evaluate the detection only on the homography to speed up the prediction on video stream.

Using the OpenCV library, the video stream has been analysed frame by frame.
On each frame, using SIFT and SURF, the features have been detected. Then the features, of the current frame and the previous one, have been matched. Afterwards, the RANSAC algorithm has been used to find the homography.



9. Tests and Results

In the previous chapters the purpose of the thesis, the tools used, and the procedure followed were described. Now the results obtained will be presented and analyzed in the following paragraphs.

9.1 Datasets for the mAP evaluation

The mAP has been evaluated on four different testing datasets, one for each sensor in order to understand in such conditions the recognition algorithm works better or worse.

In *Table 9.1* the number of images present in each dataset.

Table 9.1: mAP datasets

Sensor	Num of image
Smartphone	80
Official Pi camera	79
NV camera	80
MAPIR	65

In order to get the best camera performance some parameters need to be configured. The parameters to which particular attention has been paid are the ISO and the shutter speed, because they must be set according to the lighting conditions.

ISO stands for *International Organization of Standardization*. It measures the level of sensitivity of the camera to available light. A lower number represents lower sensitivity to available light, while a higher number means more sensitivity. High ISO is used in low light condition while a small value of ISO is used with good illumination. Examples of ISO values are: 100, 200, 400, 800 and 1600.

Shutter Speed influences the length of time a camera shutter is open to expose light into the camera sensor. Slow shutter speed values allow more light into the camera sensor and are used for low-light and night photography, while fast shutter speeds help to freeze motion. Shutter speed values are characteristic of each camera.

In *Table 9.2* the settings used to record the video stream from which frames have been saved to create the testing datasets.

Table 9.2: sensors settings

Sensor	Resolution [pixel]	fps	ISO	Shutter speed
Smartphone	1280x720	30	/	/
Official Pi camera	720x576	20	100	0 (default) **
NV Camera	720x576	20	1600	33120 µs **
MAPIR Survey3	1920x1080	60	1600	1/30 s

9.2 Haar Cascade Results

In Haar Cascade trainings, mAP has been evaluated only on the following cases: 8, 9 and 10 of *Table 7.1*. These cases have been chosen because they are the ones that have the best results as shown by *Figures 7.15, 7.16 and 7.17* (§**7.2**). They are characterized by different values of maximum False Alarm Rate that are respectively: 0,2, 0,3 and 0,4 (*Table 7.1*).
Each training generates a casade.xml file (§**8.2** and §**8.3**) that is used to make predictions and to get the following values:

- the confidence
- the coordinate in pixels of the top-left corner
- the width and height in pixels of the box

The performances were evaluated on the dataset of the images with and without distortions.

Moreover, a filter, denominated “overlapping filter”, that discards the box with the lowest confidence value in case two detections overlap more than 30%, has been applied to see if it is possible to increase the accuracy of the algorithm.

For simplicity all the results will refer to those cases:

A. images before calibration
B. images before calibration, with the overlapping filter
C. images after calibration
D. images after calibration, with the overlapping filter

** if shutter speed is set to 0 (auto), then you can read the actual shutter speed being used from this attribute. The value is returned as an integer representing a number of microseconds.

In the state of art, the most used IoU threshold is 0,5, but the mAPs evaluated with thresholds not lower than 0,3 are acceptable. In general, the lower the IoU threshold used, the higher the mAP and the higher the number of TP predictions.
In this thesis the IoU threshold used are:

- 0,3; 0,4 and 0,5 for the Haar Cascade
- 0,5 and 0,75 for Y.O.L.O.

On all the following cases the variation, on the various trainings and IoU thresholds, in percentage of the mAP and number of FP have been evaluated considering different combination:

- To see the effect of the filter, the comparisons are:
  - A vs. B

    where we mean the computation of $\Delta_{mAP}[\%] = \frac{mAP_B - mAP_A}{mAP_A} * 100$
  - C vs. D
- To see the effect of the calibration, the comparisons are:
  - A vs. C
  - B vs. D

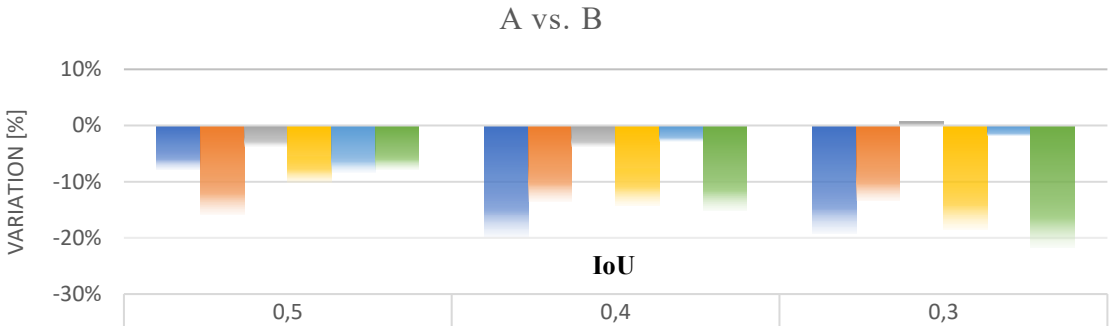
If a variation is greater than zero it means that the quantity (mAP, FP) increases, while if the variation is lower than 0, the quantity decreases. Therefore, for example, the overlapping filter is expected to lead to a decrease in the number of detections classified as FP.

All the following numerical values of the various cases are included in appropriate tables in Appendix A:

- the number of ground-truth boxes present in the dataset
- the different thresholds used for the IoU
- the mAP in percentage
- the number of True Positive boxes found (TP)
- the number of False Positive boxes found (FP)

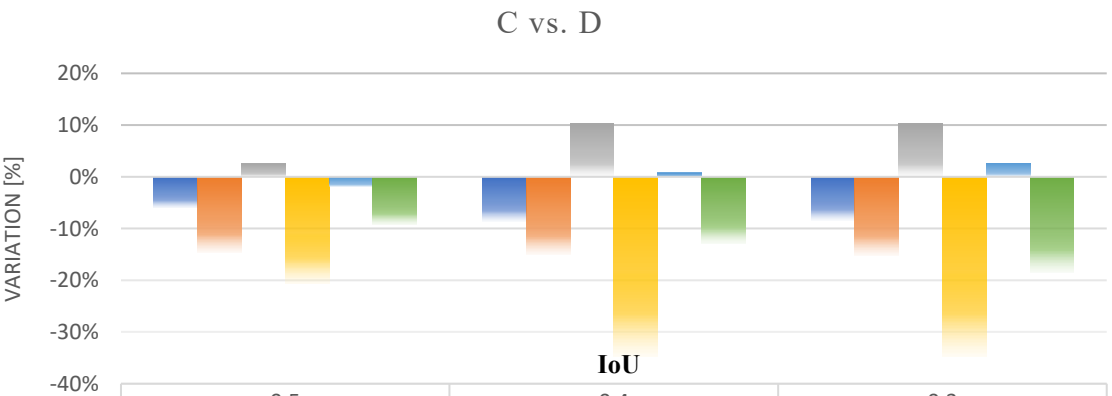
9.2.1 Smartphone dataset

Considering the set of images before calibration, the overlapping filter has been applied to remove some FP detections. As the IoU threshold decreases the variations are negative and grow in absolute value, this means that both the number of FP and the mAP.



	0,5	0,4	0,3
10-mAP	-7,82%	-19,80%	-19,24%
10-FP	-15,84%	-13,48%	-13,41%
9-mAP	-3,74%	-3,82%	0,71%
9-FP	-10,20%	-14,29%	-18,52%
8-mAP	-8,29%	-2,82%	-1,84%
8-FP	-7,84%	-15,15%	-21,74%

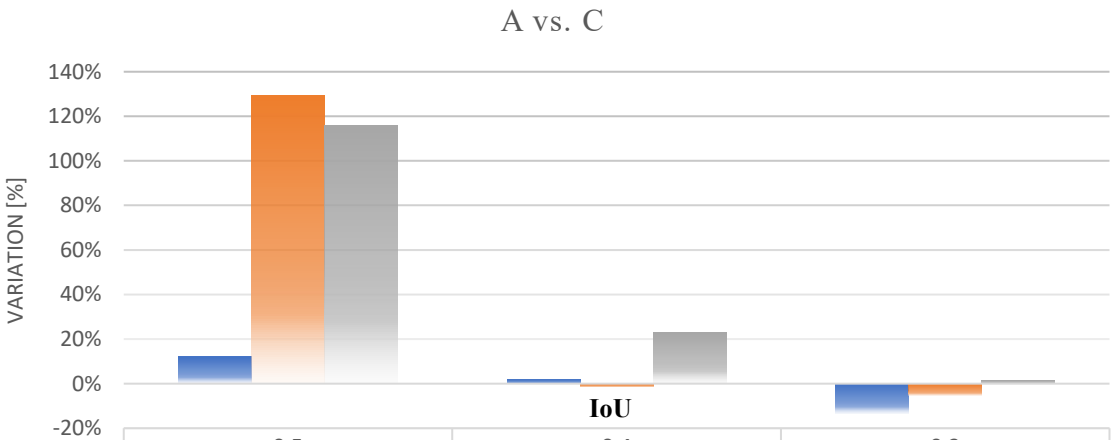
Considering the set of images without distortion, the training number 8 shows a different behaviour, the mAP variations are positive, while the FP variations are negative, this means that the overlapping filter improves the accuracy of the algorithm and removes some false detections.



	0,5	0,4	0,3
10-mAP	-5,98%	-8,60%	-8,38%
10-FP	-14,61%	-15,00%	-15,19%
9-mAP	2,56%	10,36%	10,36%
9-FP	-20,69%	-34,78%	-34,78%
8-mAP	-1,94%	0,75%	2,67%
8-FP	-9,30%	-12,90%	-18,52%

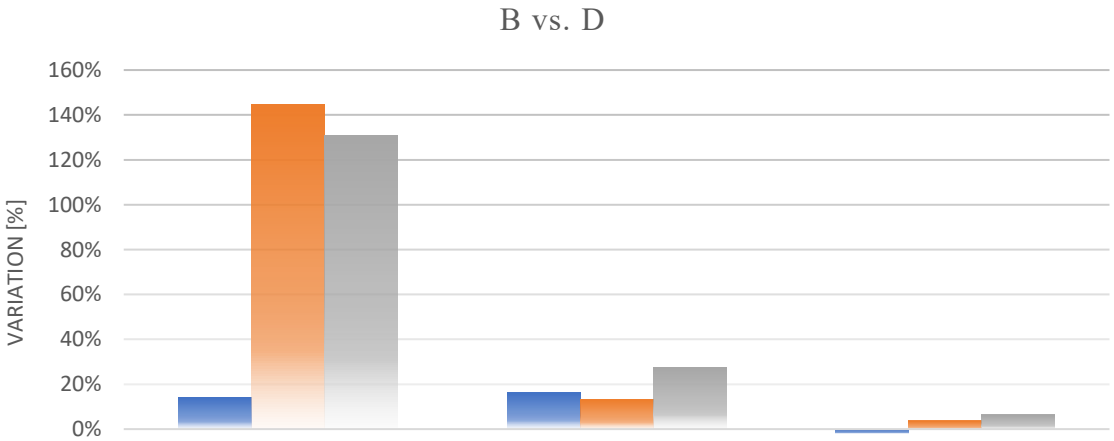
Comparing the mAP variation between images with and without distortion, the variations are mostly positive but not very large, so no major changes are seen.

Except for the 0,5 IoU case in which there are large positive variations for trainings number 9 and 8, this implies that the removal of distortions from the images has greatly improved the performance of the recognition algorithm.



	0,5	0,4	0,3
10-mAP	12,02%	1,99%	-13,49%
9-mAP	129,55%	-1,32%	-5,41%
8-mAP	115,91%	23,10%	1,64%

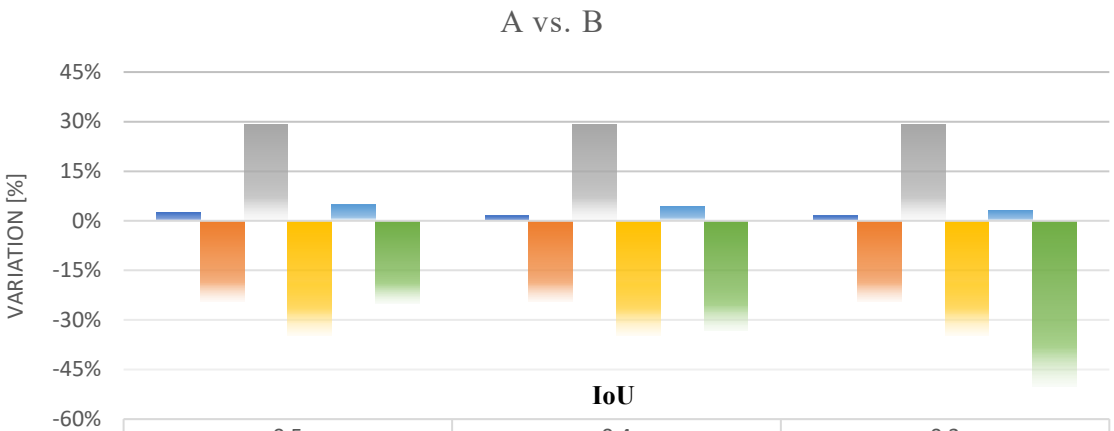
The same considerations done before are true also when the filter is used on images with and without distortion. Moreover, with respect to the previous histogram, a clear improvement can be noted in the case of IoU ≥ 0,4.



	0,5	0,4	0,3
10-mAP	14,26%	16,23%	-1,85%
9-mAP	144,55%	13,22%	3,66%
8-mAP	130,87%	27,63%	6,31%

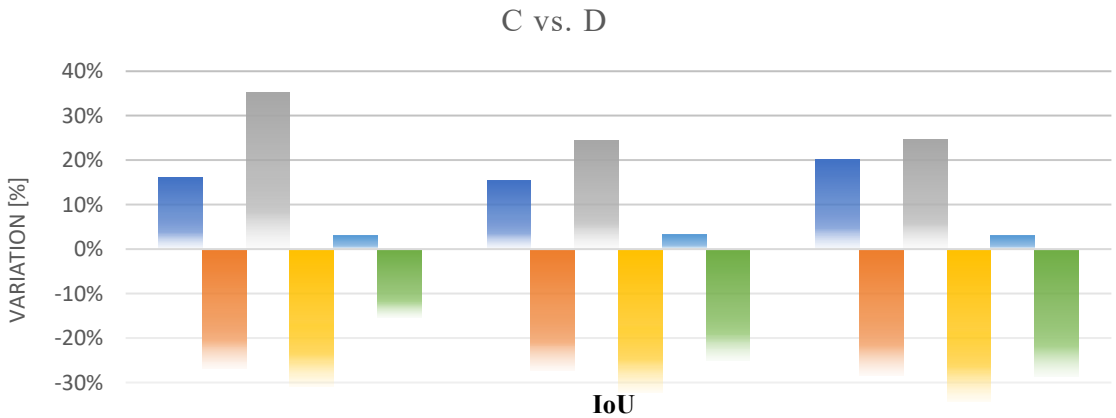
9.2.2 Official Pi camera dataset

Considering the set of images before calibration, the overlapping filter has been. As the IoU thresholds decreases, the mAP variations are positive and increases, while the FP variation are negative and decreases. This means that the overlapping filter has a good impact on the predictions.



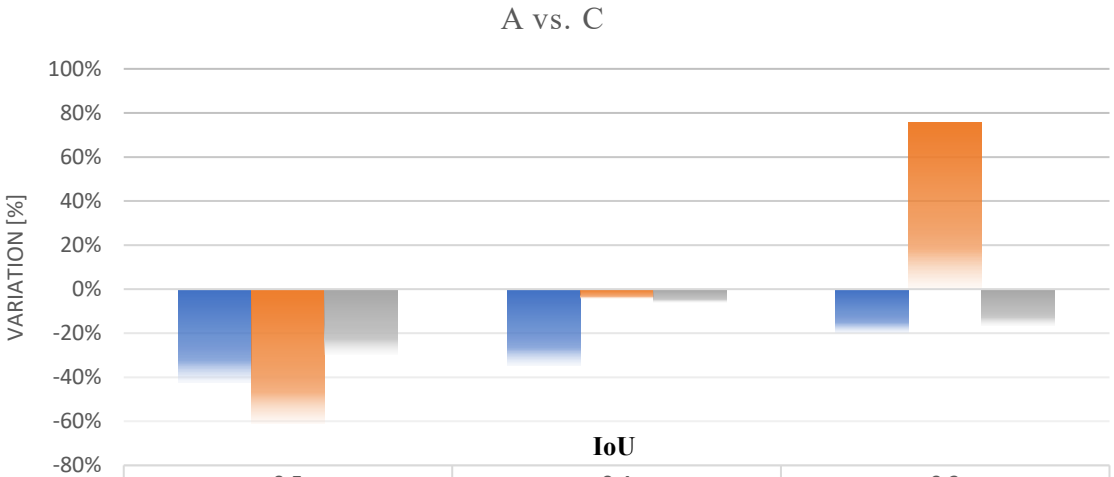
	0,5	0,4	0,3
10-mAP	2,62%	1,61%	1,61%
10-FP	-24,27%	-24,51%	-24,51%
9-mAP	29,26%	29,26%	29,26%
9-FP	-34,72%	-34,72%	-34,72%
8-mAP	4,89%	4,45%	3,26%
8-FP	-25,00%	-33,33%	-50,00%

On the set of images without distortion, the overlapping filter leads to a greater improvement with respect to the case of images before calibration for the training number 9, while for the other trainings the variations are very similar.



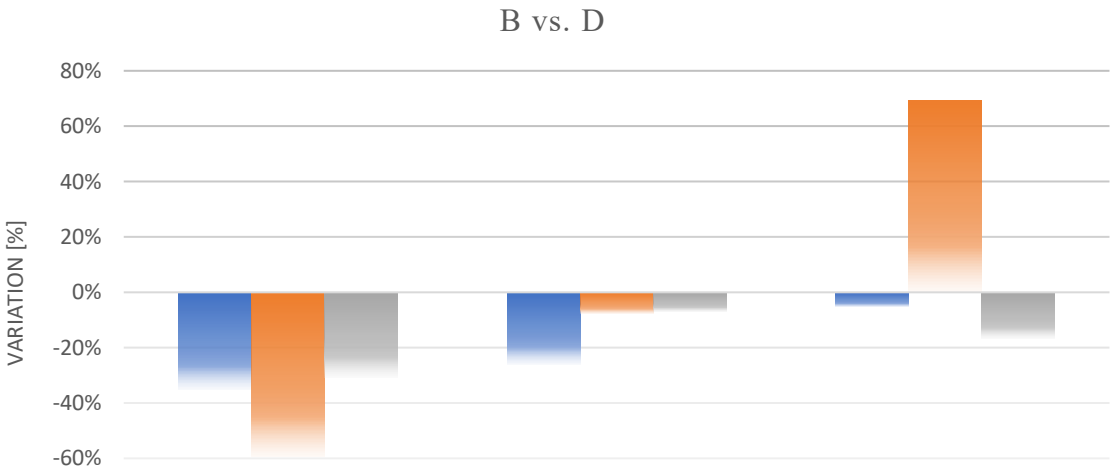
	0,5	0,4	0,3
10-mAP	16,18%	15,34%	20,13%
10-FP	-26,85%	-27,36%	-28,43%
9-mAP	35,24%	24,32%	24,68%
9-FP	-30,99%	-32,35%	-34,38%
8-mAP	3,00%	3,20%	2,95%
8-FP	-15,38%	-25,00%	-28,57%

Comparing the variations of the mAP between the set of images with and without distortion, the variations are mostly negative, so the removal of distortions from the images makes the recognition of the extinguisher more difficult. Except for training number 9 and IoU > 0,3, in which calibration improves the detection ability of the algorithm, since the variation is big and positive.



	0,5	0,4	0,3
10-mAP	-42,53%	-34,82%	-19,85%
9-mAP	-61,11%	-4,07%	75,56%
8-mAP	-29,63%	-5,90%	-16,55%

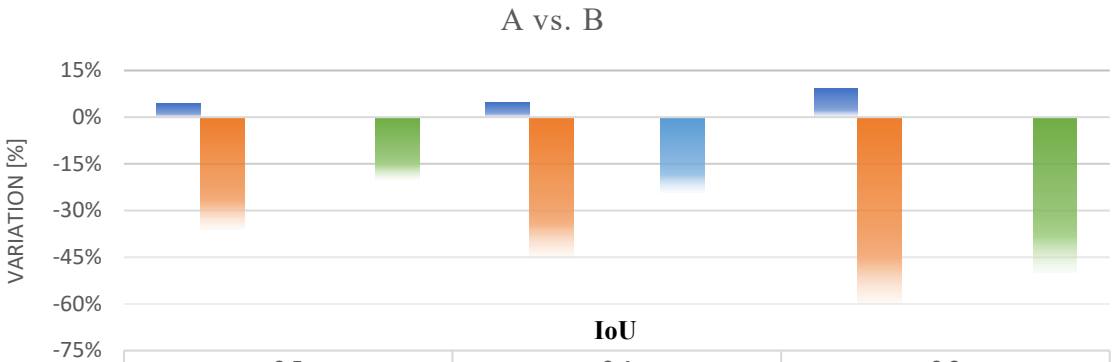
The same considerations done before are true also when the filter is used on images with and without distortion, even if, the variation of mAP of training number 9 and IoU ≥ 0,3 has been reduced a bit. This means that in this case it is better to apply only the overlapping filter and not also the calibration.



	0,5	0,4	0,3
10-mAP	-34,93%	-26,01%	-5,24%
9-mAP	-59,31%	-7,74%	69,34%
8-mAP	-30,90%	-7,03%	-16,80%

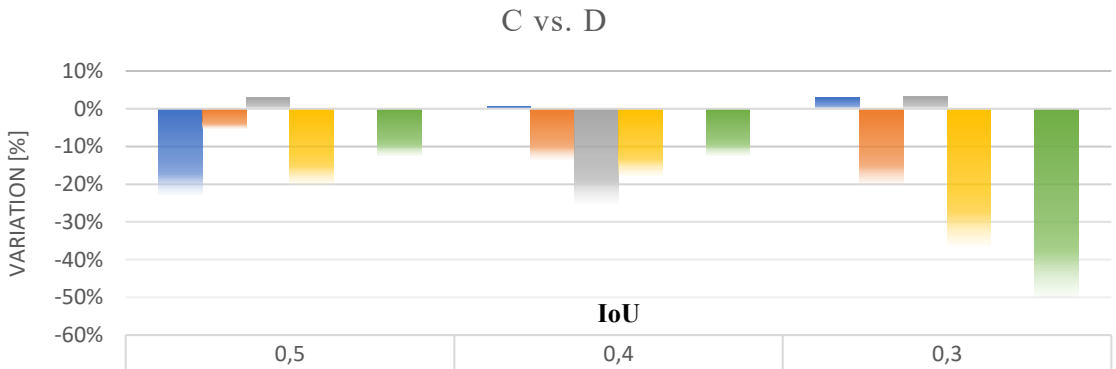
9.2.3 Night Vision camera dataset

Considering the set of images before calibration, the overlapping filter has been. As the IoU thresholds changes, the mAP and FP variations, of training number 9, are null. This means that the overlapping filter has not impact on both mAP and FP. Considering the other trainings, the FP variations are negative and big in absolute value, so the filter has removed a lot of false detection without touching the predictions classified as TP.



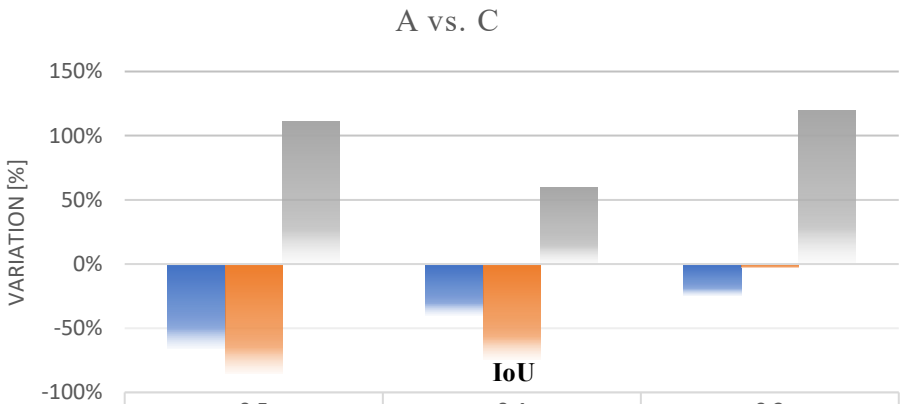
	0,5	0,4	0,3
10-mAP	4,45%	4,83%	9,27%
10-FP	-36,00%	-45,00%	-60,00%
9-mAP	0,00%	0,00%	0,00%
9-FP	0,00%	0,00%	0,00%
8-mAP	0,00%	-24,09%	0,00%
8-FP	-20,00%	0,00%	-50,00%

Considering the set of images without distortion, the effect of the overlapping filter is more or less the same as before, the difference is that the variations relative to training number 9 are no more zero. This means that removing distortion makes the algorithm more sensible to the overlapping filter.



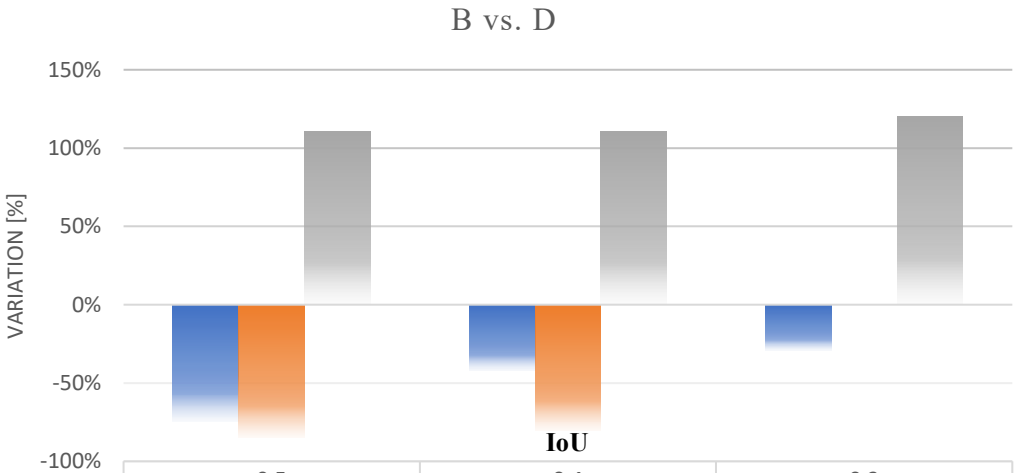
	0,5	0,4	0,3
10-mAP	-22,69%	0,70%	3,00%
10-FP	-5,26%	-13,33%	-20,00%
9-mAP	2,90%	-25,00%	3,17%
9-FP	-20,00%	-17,65%	-36,36%
8-mAP	0,00%	0,00%	0,00%
8-FP	-12,50%	-12,50%	-50,00%

Comparing the variations of the mAP between the set of images with and without distortion, the variations are mostly negative, except for the training number 8, in which the calibration produces a greater improvement, since the mAP variation is big and positive.



	0,5	0,4	0,3
10-mAP	-65,66%	-39,71%	-24,87%
9-mAP	-85,38%	-74,11%	-2,76%
8-mAP	110,78%	60,00%	120,00%

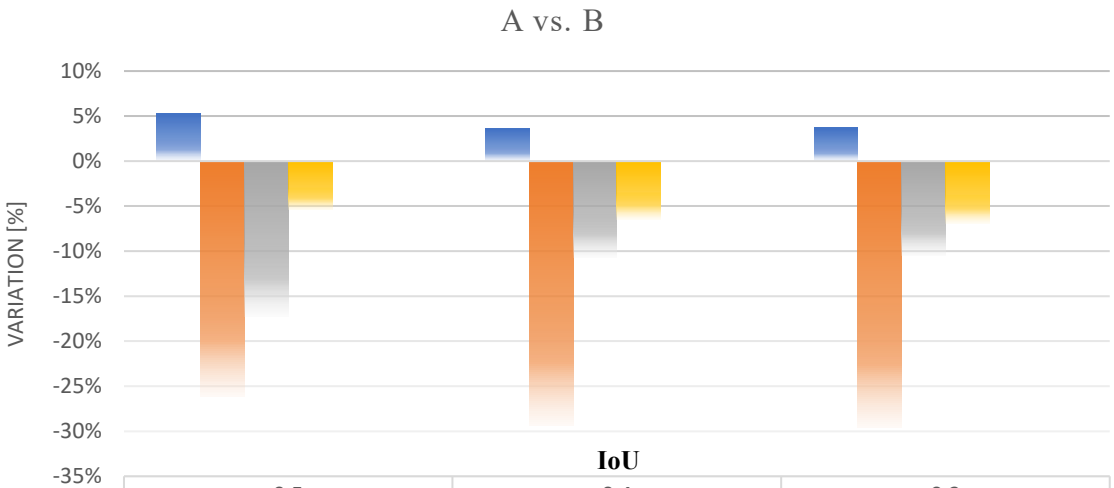
The same considerations done before are true also when the overlapping filter is used on images with and without distortion. This time the variations relative to training number 8 are bigger than the previous, so also the use of the filter improved the detection ability of the algorithm.



	0,5	0,4	0,3
10-mAP	-74,58%	-42,08%	-29,18%
9-mAP	-84,96%	-80,58%	0,32%
8-mAP	110,78%	110,78%	120,00%

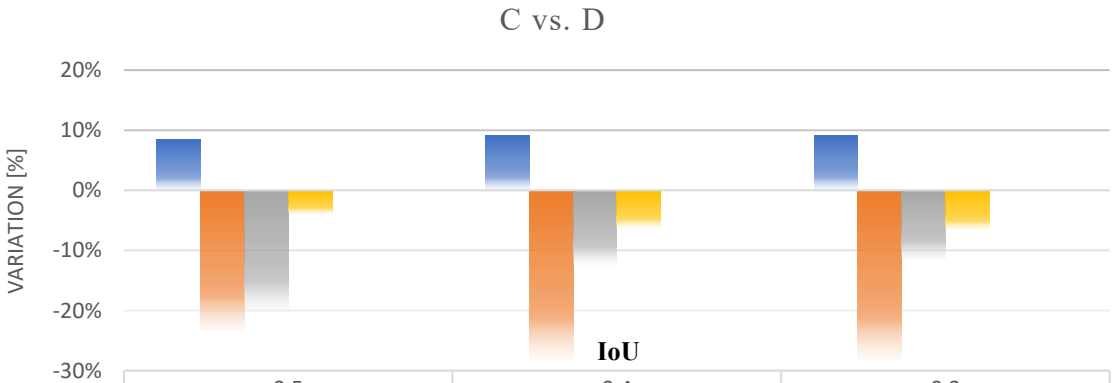
9.2.4 MAPIR dataset

Considering the set of images before calibration, the overlapping filter has been applied. The training number 8 shows no variation. Instead, training number 10 shows positive variations for the mAP and negative variations for the number of FP. Training number 9 has negative variations for both mAP and FP.



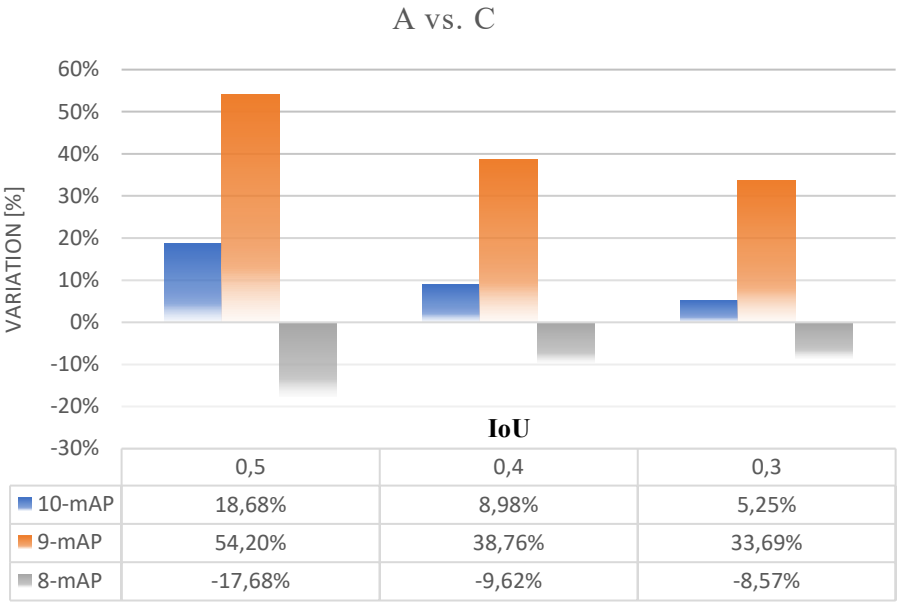
	0,5	0,4	0,3
10-mAP	5,32%	3,62%	3,69%
10-FP	-26,15%	-29,36%	-29,63%
9-mAP	-17,21%	-10,67%	-10,48%
9-FP	-5,41%	-6,45%	-6,90%
8-mAP	0,00%	0,00%	0,00%
8-FP	0,00%	0,00%	0,00%

Considering the set of images without distortion, the effect of the filter is the same.



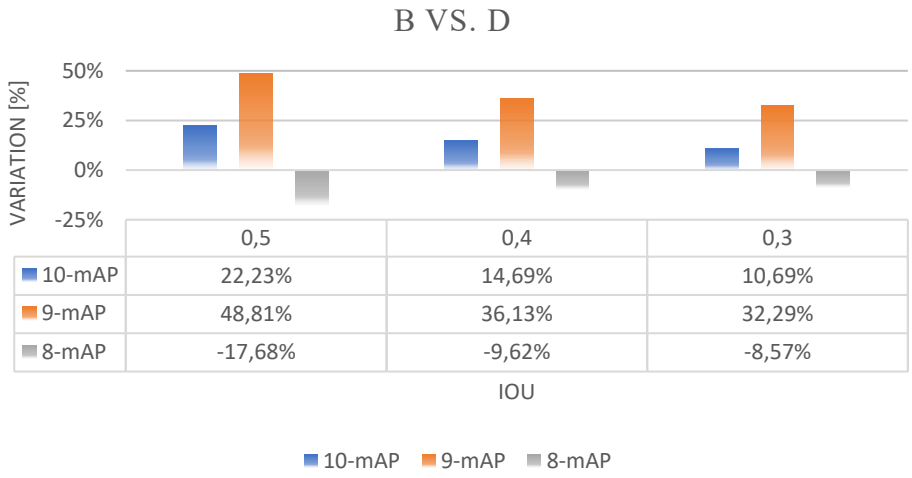
	0,5	0,4	0,3
10-mAP	8,47%	9,05%	9,05%
10-FP	-23,28%	-28,13%	-28,13%
9-mAP	-20,10%	-12,37%	-11,42%
9-FP	-3,90%	-6,25%	-6,56%
8-mAP	0,00%	0,00%	0,00%
8-FP	0,00%	0,00%	0,00%

Comparing the variations of the mAP between the set of images with and without distortion, the variations are positive for trainings number 10 and 9, then for them calibration helps the detection.



	0,5	0,4	0,3
10-mAP	18,68%	8,98%	5,25%
9-mAP	54,20%	38,76%	33,69%
8-mAP	-17,68%	-9,62%	-8,57%

The same considerations done before are true also when the overlapping filter is used on images with and without distortion. In particular, for training number 10 the variations are bigger than the previous histogram, this means that the overlapping filter has a greater impact in this case.



	0,5	0,4	0,3
10-mAP	22,23%	14,69%	10,69%
9-mAP	48,81%	36,13%	32,29%
8-mAP	-17,68%	-9,62%	-8,57%

9.2.5 Considerations

As you can see from the previous histograms it is not possible to say absolutely that the calibration or the use of overlapping filter will improve the performance of the algorithm, but the effects vary from case to case.

9.3 Y.O.L.O. re-training results

As shown in the flowchart 8.7, the mAP relative to a set of images is obtained through a command, then is not possible applying some filter as Haar Cascade procedure.

Only the comparison between images with and without distortion (A vs. C) has been done.

9.3.1 Smartphone dataset

Comparing the variations of the mAP between the set of images with and without distortion, the variations relative to IoU threshold equalt to 0,5 is very small, while the others is big and negative. This means that calibration has reduce a lot the performance of the trained network.

In fact, comparing the numerical value present in *Table 9.3* and *Table 9.4*, relative to the same IoU threshold:

- the mAP values decrease
- the TP values decrease
- the FP values increase

Case A

Table 9.3: Y.O.L.O. mAP (Smartphone, A)

Ground truth	IoU	mAP [%]	TP	FP
91	0,5	89,93	86	3
	0,75	37,53	51	38

Case C

Table 9.4: Y.O.L.O. mAP (Smartphone, C)

Ground truth	IoU	mAP [%]	TP	FP
91	0,5	88,67	82	9
	0,75	7,45	22	69

In *Table 9.5* instead, there are the variations of the mAP values (A vs. C)

Table 9.5: Y.O.L.O. mAP variations (Smartphone)

IoU	ΔmAP [%]
0,5	-1,40
0,75	-80,15

9.3.2 Official Pi camera dataset

Comparing the variations of the mAP between the set of images with and without distortion, the variations relative to IoU threshold equal to 0,5 is big and negative, while the other one goes down to -100%, because the mAP of IoU ≥ 0,75 of case C is 0%. This means that the network never recognize the extinguisher in the images of the dataset, again as before calibration has reduce the performance of the trained network.

Comparing the numerical value present in *Table 9.6* and *Table 9.7*, relative to the same IoU threshold:

- the mAP values decrease
- the TP values decrease
- the FP values increase

Case A

Table 9.6: Y.O.L.O. mAP (Official Pi camera, A)

Ground truth	IoU th	mAP [%]	TP	FP
79	0,5	61,88	13	1
	0,75	14,11	6	8

Case C

Table 9.7: Y.O.L.O. mAP (Official Pi camera, C)

Ground truth	IoU th	mAP [%]	TP	FP
79	0,5	6,89	4	16
	0,75	0	0	0

In *Table 9.5* instead, there are the variations of the mAP values (A vs. C)

Table 9.8: Y.O.L.O. mAP variations (Official Pi camera)

IoU	ΔmAP [%]
0,5	-88,87
0,75	-100,00

9.4 Y.O.L.O. vs. OpenCV

As already said in the Abstract, comparing Y.O.L.O. and OpenCV it is evident that the previous one is more accurate and gives rise to lower false detection. Indeed, the threshold use for the IoU with Y.O.L.O. are higher. However, Y.O.L.O. is not able to find the fire extinguisher on the images taken with the MAPIR and the Night Vision camera. In addition, from the tables and above is evident that on images like those of training Y.O.L.O. has excellent performance, but any variation drastically reduces its performance.

In order to better understand the limits of the network trained with Y.O.L.O., we have converted to grayscale the images of the smartphone dataset, on which Y.O.L.O. for a IoU $\geq$ 0,5 has a mAP of about 90%. The Y.O.L.O. mAP on grayscale images with a IoU $\geq$ 0,5 was around 9%, this means that to get the best performance from Y.O.L.O. re-training, it is necessary to use images very similar to those of the training, so for example, if you want to use Y.O.L.O. on calibrated images of any sensor it is necessary to create the training dataset using calibrated images taken with the same sensor.

Considering the results obtained, it was hypothesized that if grayscale images were used during training, Y.O.L.O. could be more adaptable to different sensors, because a sensor such as the Night Vision camera used works well both in good light and darkness conditions. Then images taken in both situations have a good contrast and so their grayscale version will be very similar.

9.5 Descriptors and Features matching results

As mentioned in Chapter 3, how the combination of SIFT or SURF with RANSAC influences the speed of the recognition algorithm on a video stream, has been analyzed.

This combination has been tested only with the Haar Cascade training, while Y.O.L.O. needs to load the trained neural network and so it is not possible to analyze the single frame before passing it to the network.

The characteristics of the videos used are in *Table 9.2* and 500 frames of each video were analyzed. On each frame, the following parameters were measured:

(a) time interval to recognize features (s);
(b) time interval to match features of two consecutive frames (s);
(c) time interval to find homography (s);
(d) number of features;

(e) number of matching features;

(f) average fps

Below, for each sensor used are shown the curves of the time intervals and of the number of features over the frame. In order to make comparison all x and y axis have the same scale and to make the reading of the results easier a table with minimum, maximum and average values of each variable has been inserted.

9.5.1 Smartphone video stream

Comparing the time required to find the features (a), SURF is faster, but comparing the average fps at the end, SURF is however faster, but the speed difference is significantly reduced. In addition, SURF finds more features and more matching.

However, the resulting average fps is lower using the descriptors than only detection (last row of *Table 9.9*), this means that for this application they take longer to process than they should have saved.

Table 9.9: descriptors results on the Smartphone

	SIFT			SURF			Only detection
	Max	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Avg**
(a)	0,2058	0,1459	0,1601	0,1809	0,0098	0,0320	
(b)	0,0444	0,0024	0,0173	0,0867	0,0098	0,0320	
(c)	0,0164	0,0004	0,0012	0,0357	0,0010	0,0026	
(d)	1845	82	706	3696	506	1592	
(e)	693	17	24	1415	97	549	
(f)			6			7	19

From the following graphs it can be noted that:

- step (a), blue line, that is the one that measures the time required to find the features is the highest, using both SIFT (*Figure 9.1 and 9.2*) and SURF (*Figure 9.3 and 9.4*);
- before frame number 150 there is a decrease in times and number of features, both with SIFT and SURF, the decrease with SURF is more marked.

This descent could be due to the fact that the extinguisher is no longer present in the video for some frames and then returns to the foreground.

SIFT

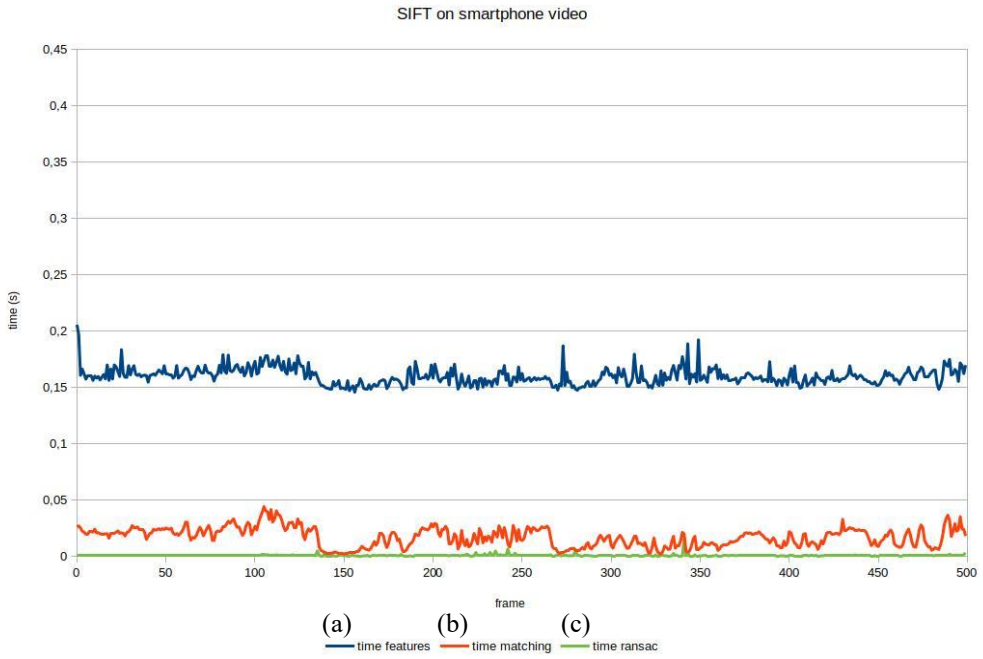


Figure 9.1: time intervals required by SIFT (Smartphone)

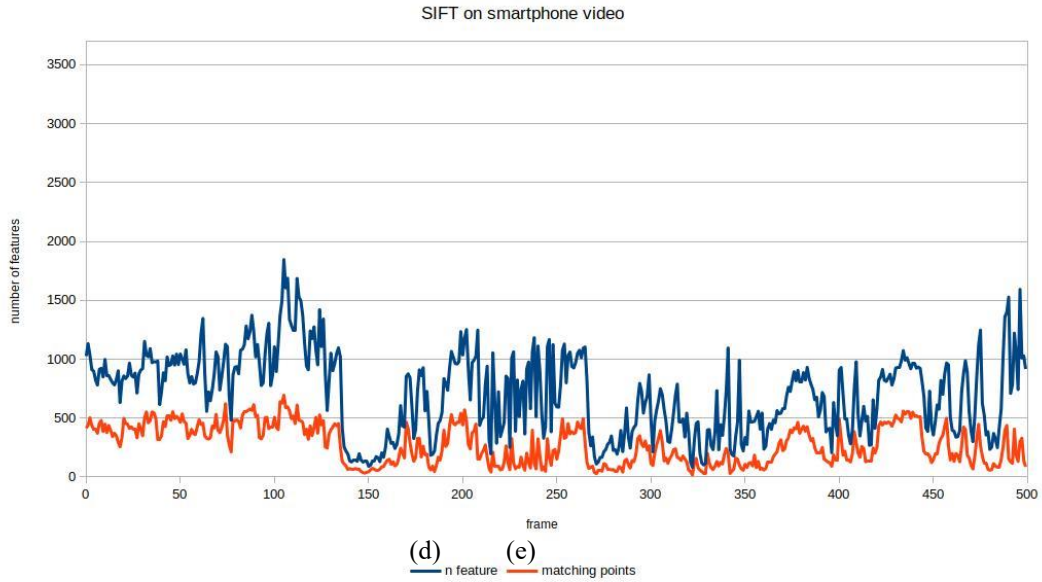


Figure 9.2: features found by SIFT (Smartphone)

SURF

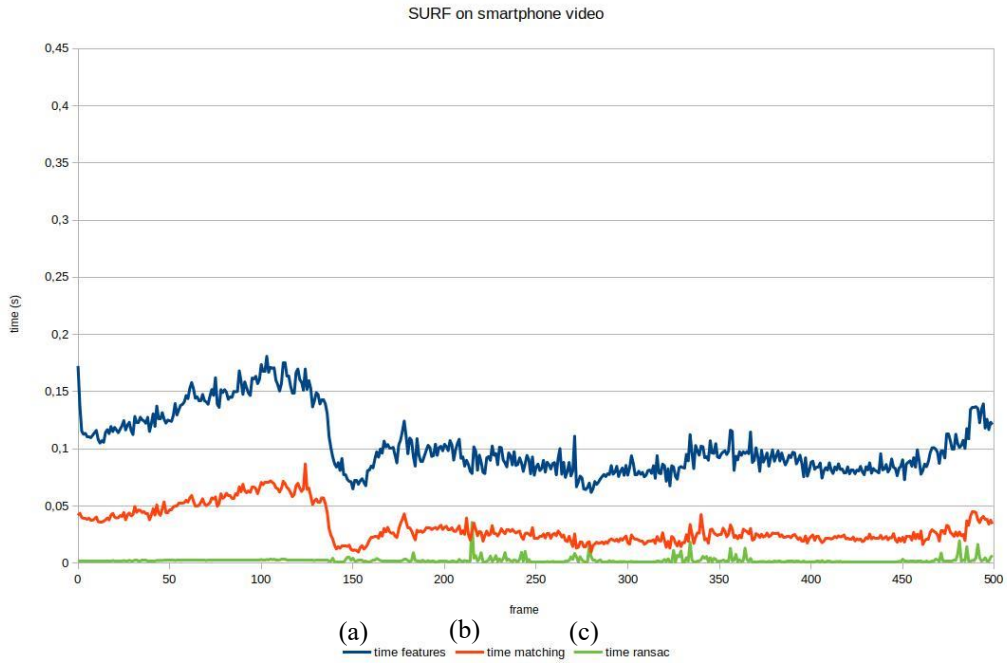


Figure 9.3: time intervals required by SURF (Smartphones)

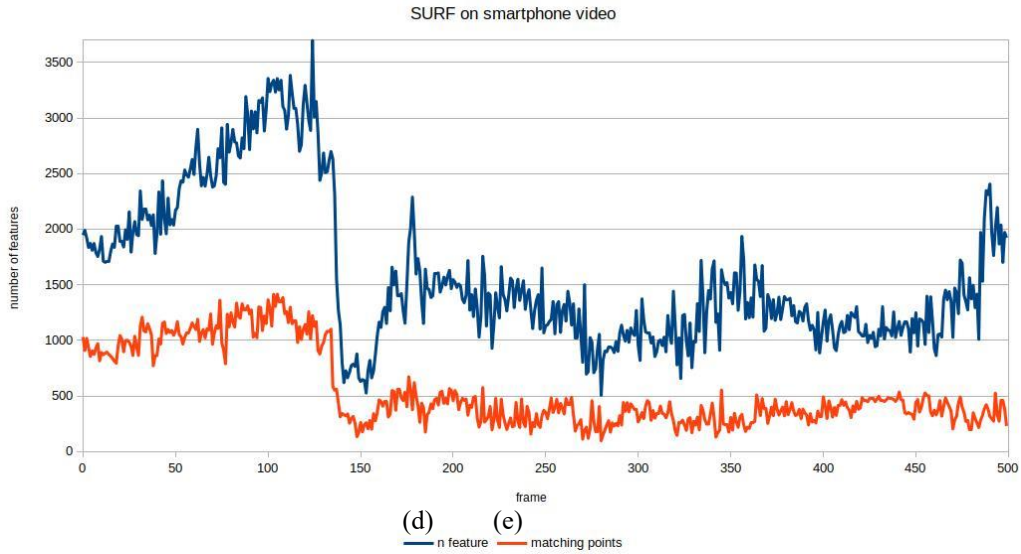


Figure 9.4: features found by SURF (Smartphone)

9.5.2 Official Pi camera video stream

As noted in the previous case SURF is faster in finding features than SIFT, but comparing the average fps, they are the same, so the speed difference is not significantly at the end. In this case the number of features found in the frame is lower than the number of features found on the frame of the smartphone video.

Table 9.10: descriptors results on the Official Pi camera

	SIFT			SURF			Only detection
	Max	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Avg**
(a)	0,0936	0,0700	0,0744	0,1424	0,0596	0,0745	
(b)	0,0092	0,0032	0,0061	0,0394	0,0140	0,0216	
(c)	0,0010	0,0005	0,0006	0,0251	0,0009	0,0014	
(d)	419	143	267	1567	710	1134	
(e)	219	44	118	656	156	409	
(f)			6			6	37

SIFT

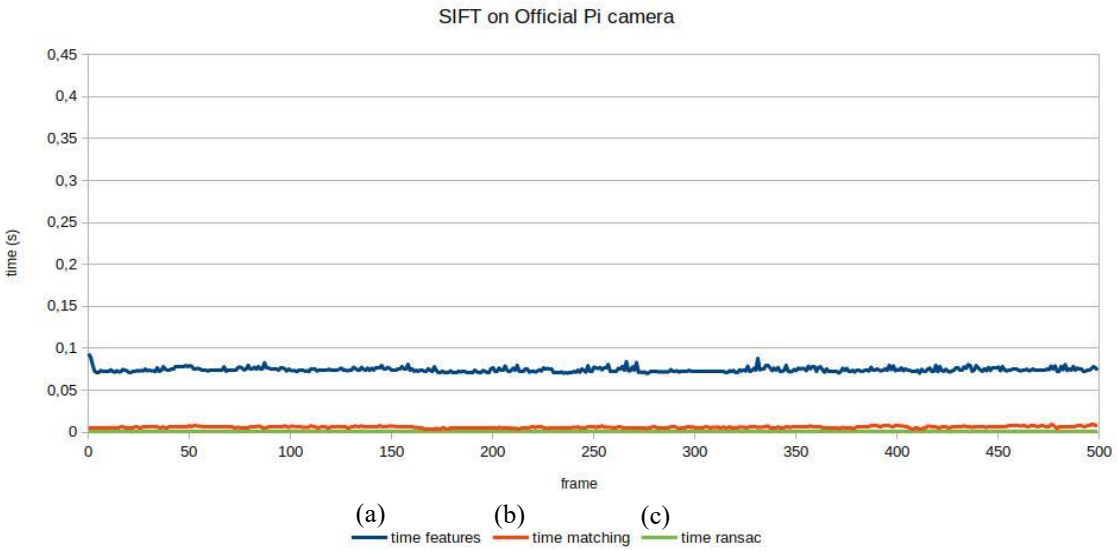


Figure 9.5: time intervals required by SIFT (Official Pi camera)

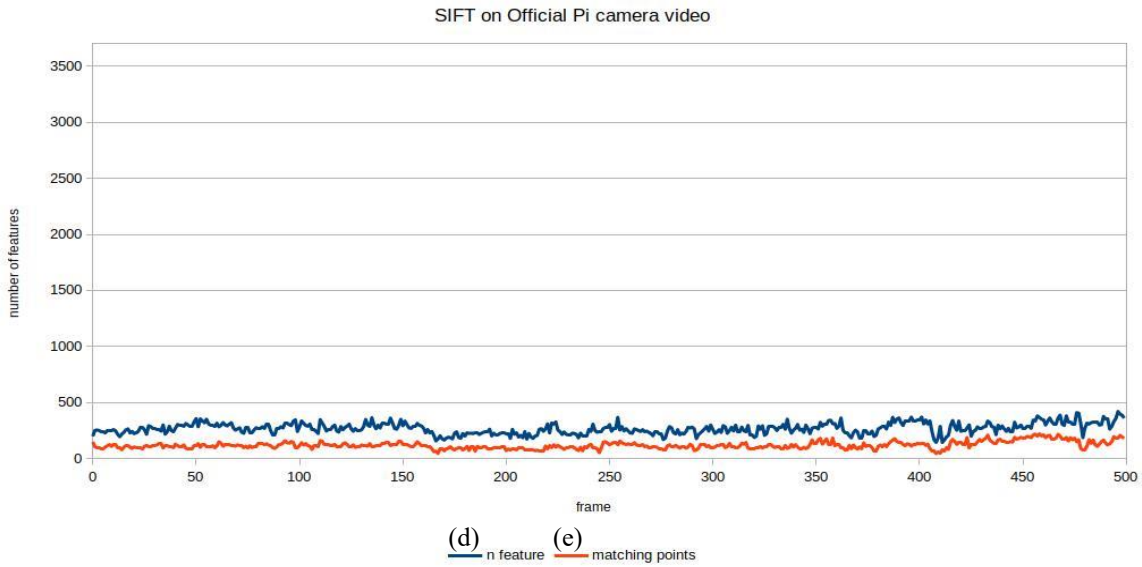


Figure 9.6: features found by SIFT (Official Pi camera)

SURF

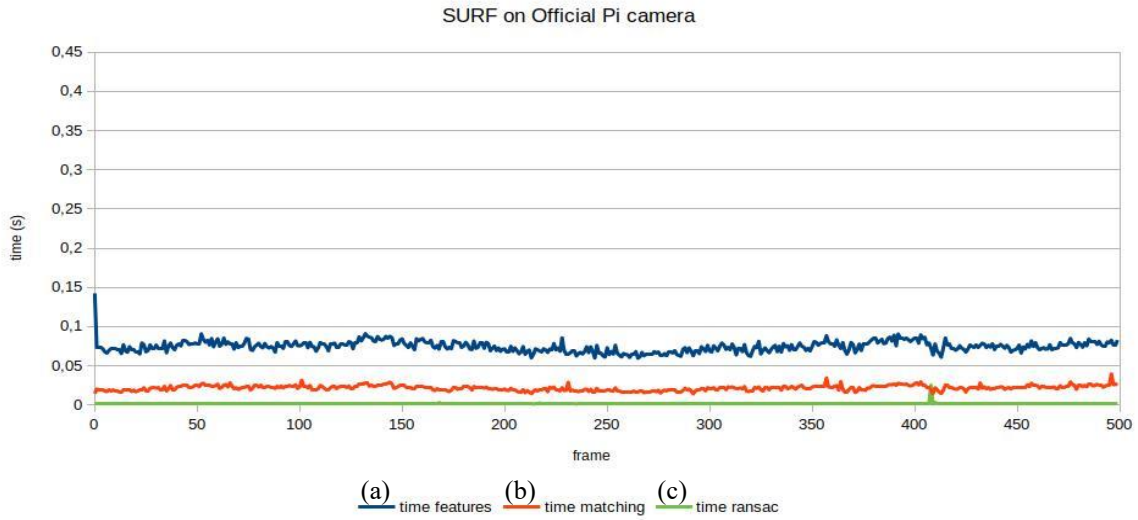


Figure 9.7: time intervals required by SURF (Official Pi camera)

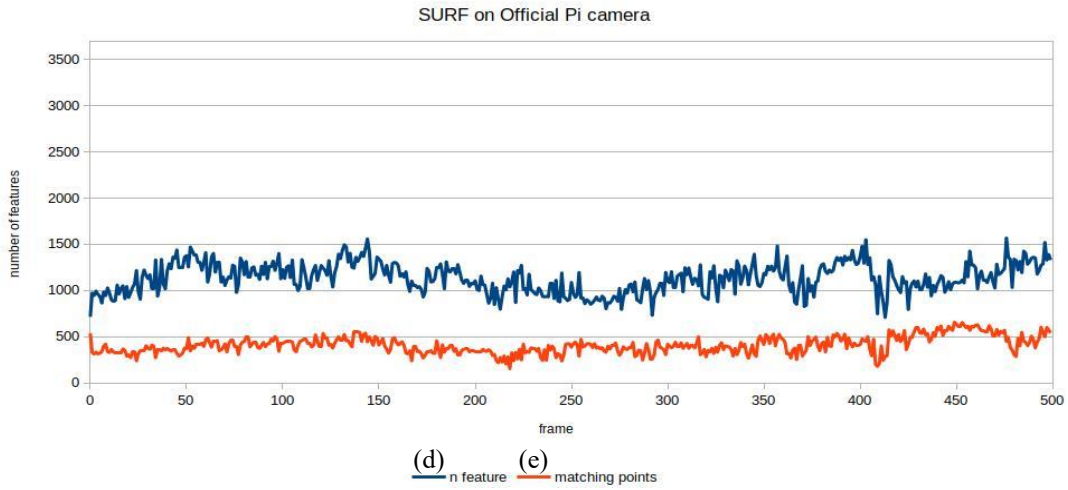


Figure 9.8: features found by SURF (Official pi camera)

9.5.3 Night Vision camera video stream

Again, the same consideration done for time intervals in the Smartphone and Official Pi camera cases are true also Night Vision camera.

In *Table 9.11* the minimum of features matching is zero, because between some frames less than 10 matches have been found, so the homography cannot be done and the entire frame is passed to the detection algorithm. The average fps is higher than the one of the Official Pi cameras and this could be since less features are found. In the graphs of *Figures: 9.9, 9.10, 9.11 and 9.12*, it is not visible due to the scale used for the y-axis, but also in this case all the variables oscillated a lot.

Table 9.11: descriptors results on the Night Vision camera

	SIFT			SURF			Only detection
	Max	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Avg**
(a)	0,0884	0,0666	0,0695	0,0686	0,0194	0,0245	
(b)	0,0026	0,0001	0,0007	0,0066	0,0003	0,0017	
(c)	0,0021	0,0003	0,0003	0,0039	0,0003	0,0005	
(d)	100	2	25	371	17	83	
(e)	68	0	13	137	0	50	
(f)			14			38	52

SIFT

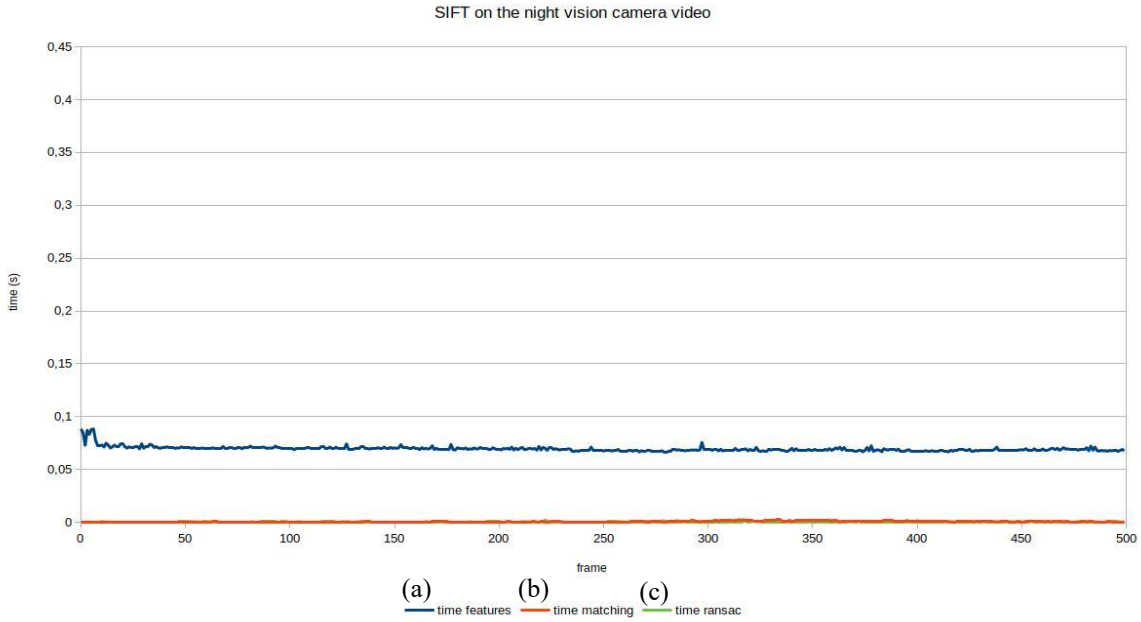


Figure 9.9: time intervals required by SIFT (NV camera)

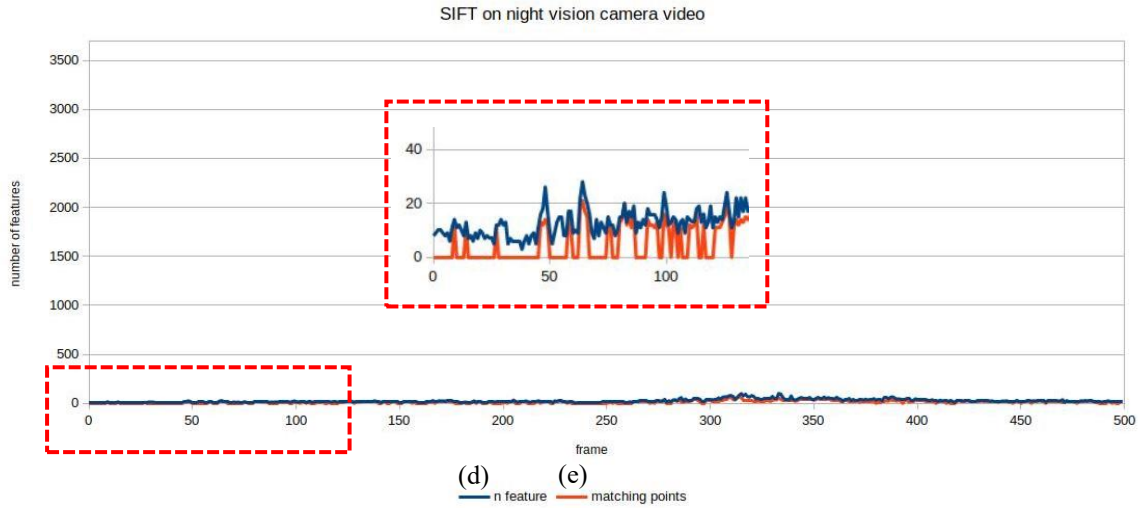


Figure 9.10: features found by SIFT (NV camera)

SURF

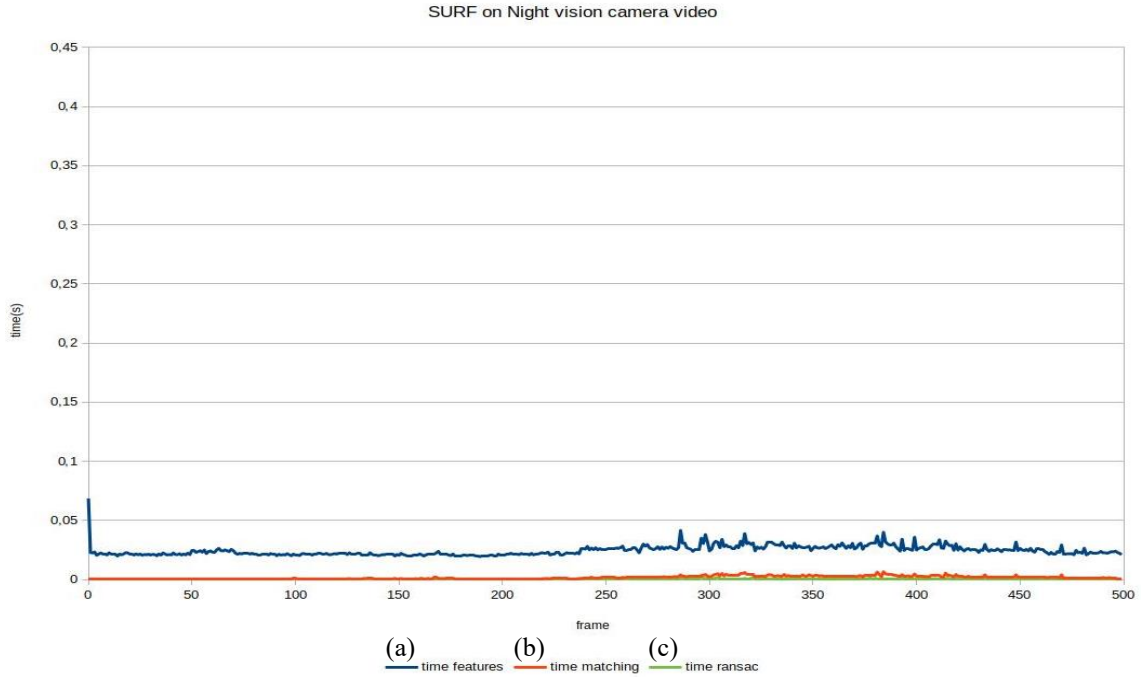


Figure 9.11: time intervals required by SURF (NV camera)

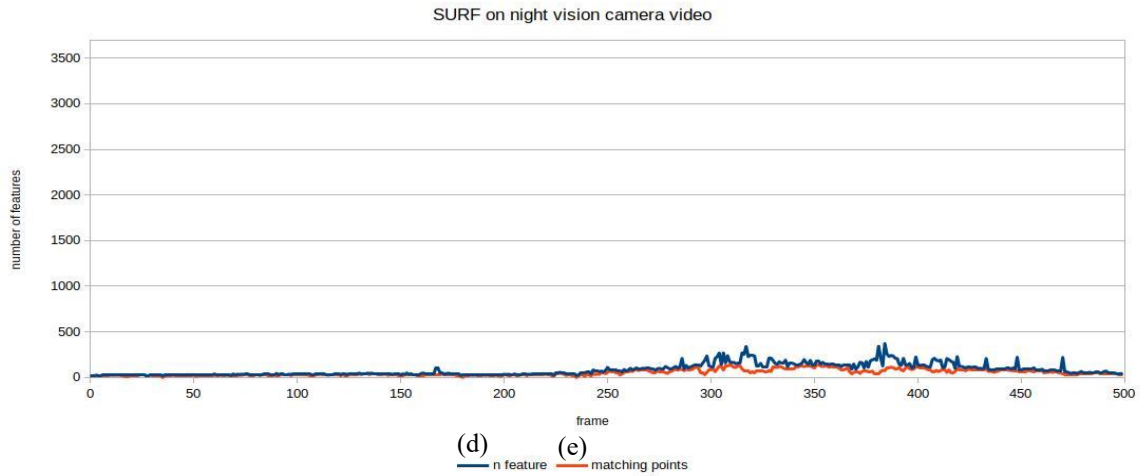


Figure 9.12: feature found by SURF (NV camera)

9.5.4 MAPIR video stream

The consideration on the time intervals are again the same. On the MAPIR video the average fps, in every case, is the lowest and this is probably due to the high resolution used

In *Figure 9.15*, related to SURF cases a, b and c, RANSAC time interval (green) has some peaks, this means that it has difficulty to find homography, but the cause is unknown since in the video are not present any changes of illumination or others.

Table 9.12: descriptors results on the MAPIR camera

	SIFT			SURF			Only detection
	Max	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Avg**
(a)	0,4380	0,3395	0,3470	0,1784	0,1072	0,1487	
(b)	0,0046	0,0015	0,0025	0,0344	0,0042	0,0203	
(c)	0,0019	0,0004	0,0006	0,2020	0,0006	0,0072	
(d)	221	60	100	2131	204	1251	
(e)	59	13	28	994	47	238	
(f)			3			6	9

SIFT

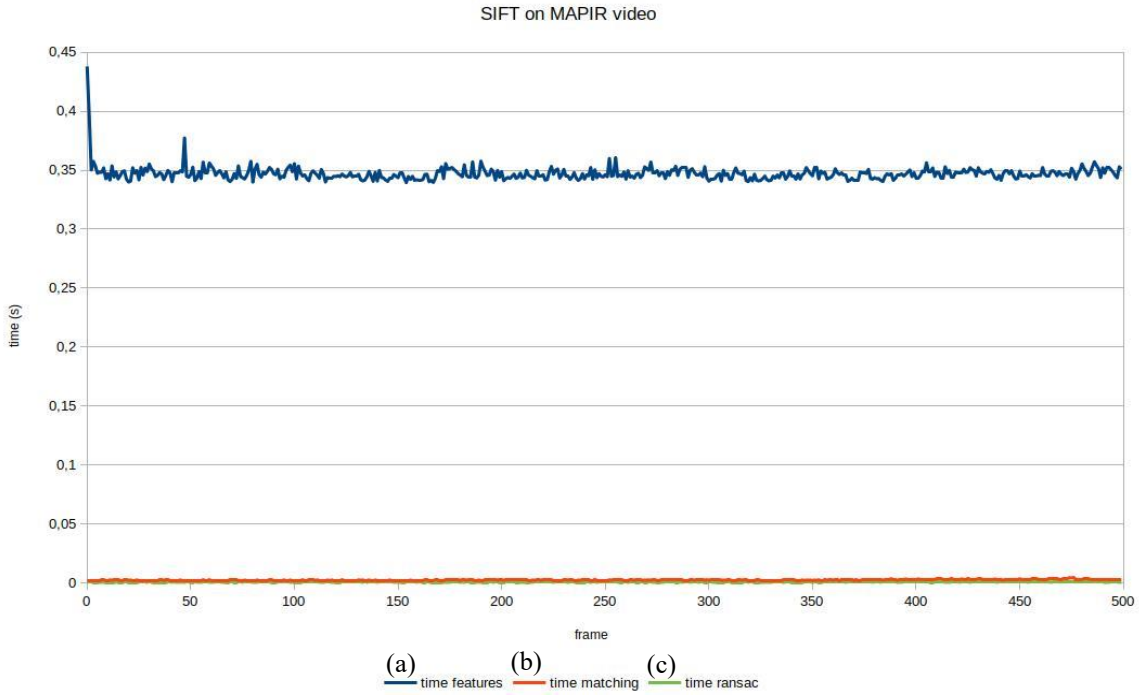


Figure 9.13: time intervals required by SIFT (MAPIR Survey3)

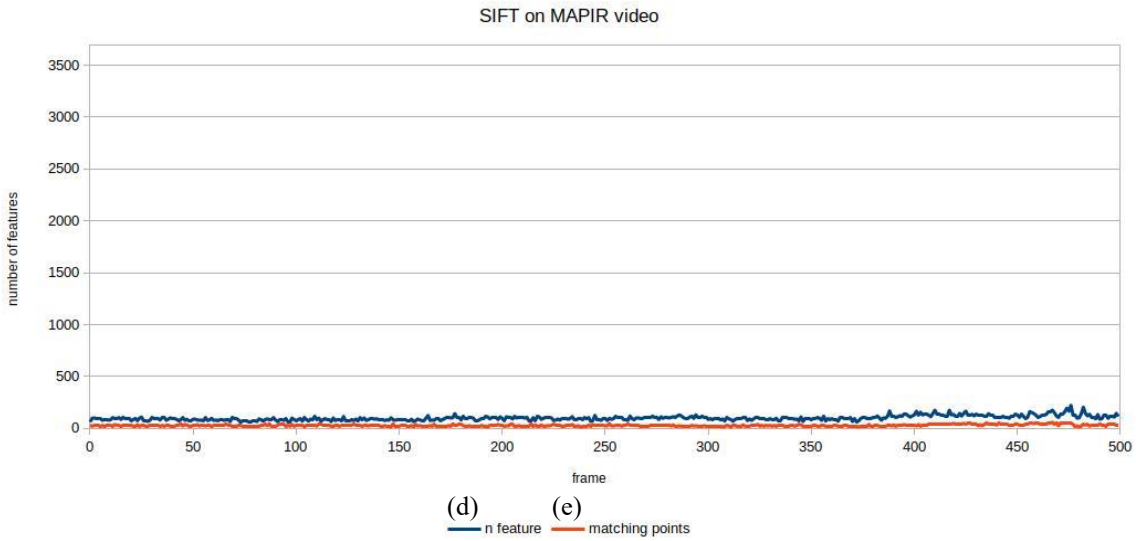


Figure 9.14: features found by SIFT (MAPIR Survey3)

SURF

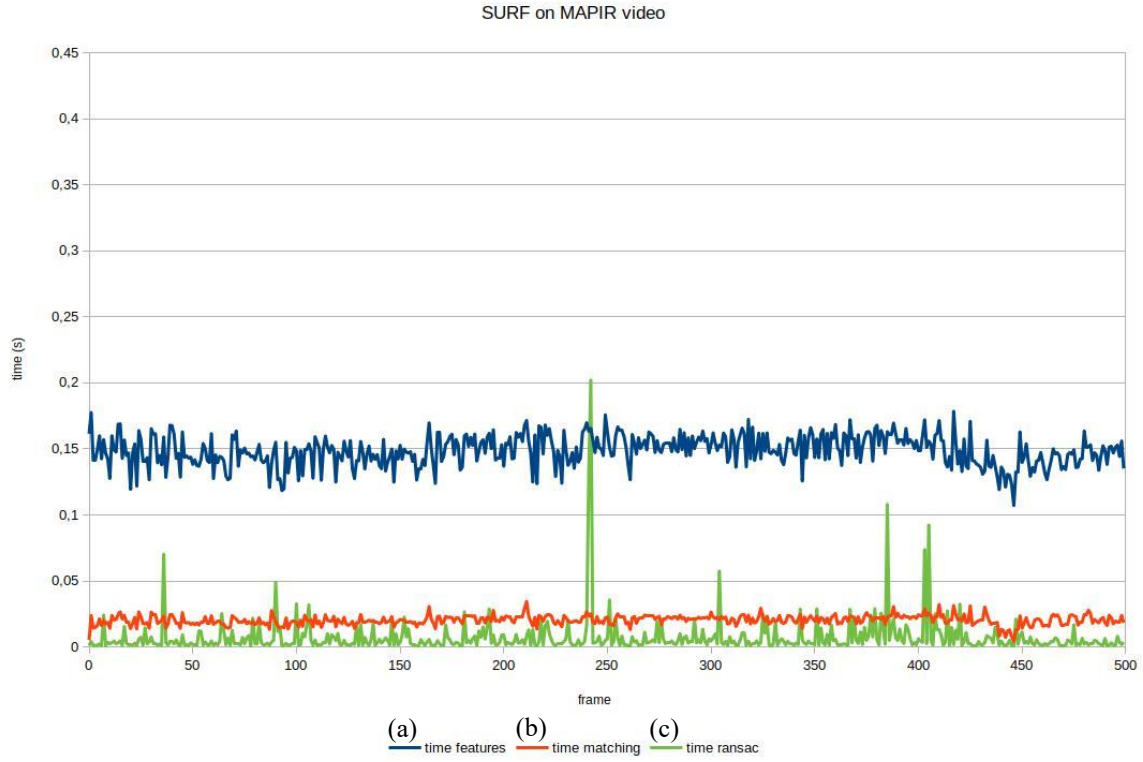


Figure 9.15: time intervals required by SURF (MAPIR Survey3)

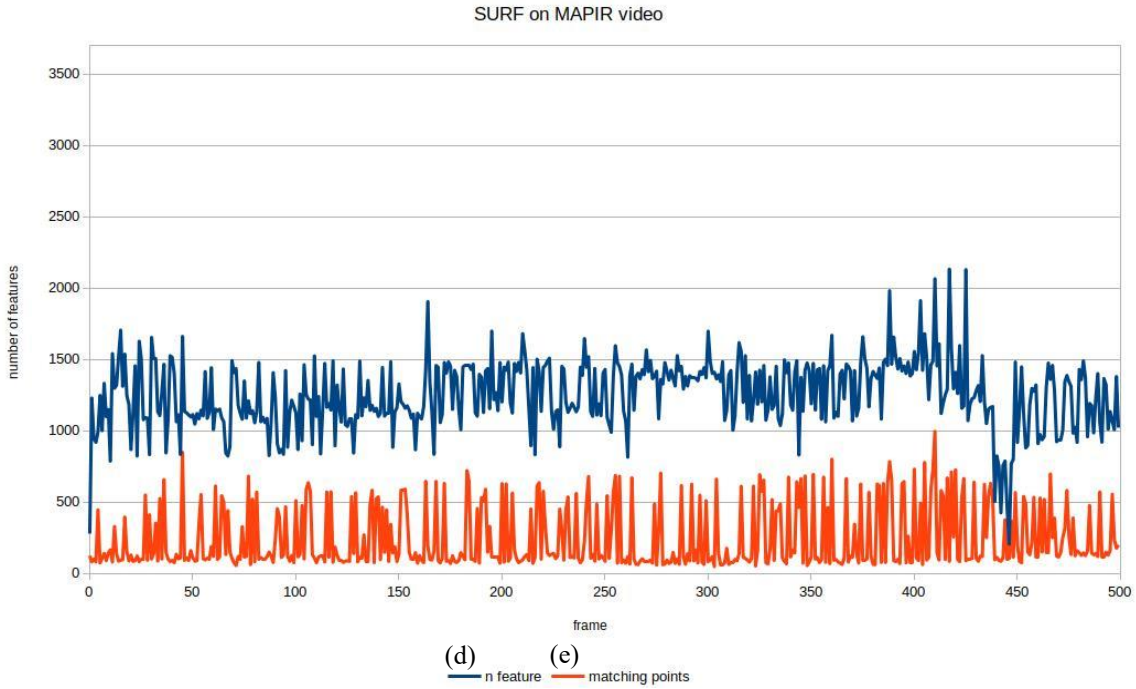


Figure 9.16: features found by SURF (MAPIR Survey3)

9.5.5 Considerations

From the results obtained the calculation of features, their matching and the search for homography require more time than we thought they would have earned us, so it is not advisable to use this technique. Moreover, the higher the resolution, the slower the algorithm is, so if you don't need an HD or full HD resolution, it is better to use a lower one.

9.6 Detection using Raspberry Pi

As already said, this thesis aims to obtain an Object Detection algorithm which is able to identify the extinguisher, using low cost sensors and platforms. Therefore, in Chapter 5 the sensors were described and in §**8.2** there is the flowchart related to the testing of the Haar Cascade training. The procedure described in that flowchart can be safely used with the Raspberry Pi.

To obtain optimal performance as input, we have used the mode "take photos continuously" of the Raspberry Pi camera, then on each photo the detection is applied and after thst each photo is saved as a video frame, which is the chosen output format. More in details, the detection made with the Raspberry Pi works as follows:

- the connected camera takes photos continuously
- every 700ms a photo is shot

- on every photo taken the detection is done
- the photo with the prediction boxes is saved as a video frame
- the output of this procedure is a video stream

700ms has been used because if the camera is used at maximum resolution between one shot and the other one without any processing 667ms are required. However, with the resolution used 720x576, 500ms are enough. To correctly choose the time interval, some tests were made in which the resolution was set, and the time required by the camera between two shots was measured, the higher the resolution, the longer the required time.

The predictions on the Raspberry Pi are done only using the Haar Cascade trainings, because Y.O.L.O. requires more pervasive hardware components.

The Raspberry Pi is a single-board computer that has powerful hardware compared to its small size (85.60 mm x 56 mm). In addition, its low cost (around 35€ the Raspberry Pi 3 Model B+) has made it one of the most used hardware for simple projects.

Unfortunately, even if it is extremely powerful due to its size, it is not enough for the computation needed using a Neural Network.

Figure 9.17: night vision camera connected to Raspberry Pi

Conclusion

At this point before drawing the conclusions on the work done, it seems better making a short reference to the objectives of this thesis and tools used to achieve them.

The aims were: studying the performance of low cost systems in the Object detection field, comparing the performance of two different Object Detection model, understanding how the training images and settings influence the performances, if the calibration improve the performance. All these questions were analyzed in the case of indoor navigation of an unknown environment in emergency situation and the object to be identified is the fire extinguisher.

The low cost sensors chosen functioned as desired. Through the PiCamera library the two cameras designed for the Raspberry Pi, were managed in a very simple way, both for setting all the desired parameters (resolution, fps, ISO and shutter speed) and for choosing the operating modes, such as video, single shot and continuous shots. MAPIR has also allowed us to achieve the goal, as its NearIR component peaks at the same wavelength as the Night Vision camera illuminators, it was possible to see in low light conditions.

The models chosen and the trainings done have been shown that they influence a lot the results. Each model has its pros and cons: Haar Cascade is more versatile with different sensors, but it reaches lower performances (smaller mAP, higher number of false detections). Haar Cascade is a rather old and traditional technique that has shown excellent potential (§**9.2**), as regards the use of different sensors as it works on grayscale images, as regard the comparisons between images with and without distortion, in which has demonstrated to have sometimes better performances on the calibrated images (smartphone and MAPIR cameras) and as regarding the low cost platform implementation with reduced hardware performance, like the Raspberry Pi.

The Haar Cascade training is influenced by: the *AcceptanceRatio* parameter that has to be no lower than 10^{-5} in the last stage to ensure the model does not over train on the training data, the number of positive and negatives samples used in the first stage that must be big enough but lower than all available because at each new stage the training will add some images and using as many negatives as the half of the number of positive samples is a good practice, the maximum False Alarm Rate that if is too big as 0,5 the resulting cascade finds fire extinguisher everywhere and does not enclose well the object, the characteristics of the training images that if they present transparency the resulting cascade does not identify the

object well and this also leads to many false detection and prediction boxes that do not fit the extinguisher well and finally the orientation sets for the training images that if it is null or too low it leads to a cascade that identifies the extinguisher only in a few cases since the dataset without rotation is not quite varied in terms of points of view.

The best performance (higher mAP) gathered with the Haar Cascade training has been reached in the MAPIR dataset and the datasets that have undergone an improvement and a reduction in the number of false detection (False Positive) thanks to the "overlapping filter" are those of the Official Pi camera and the Night Vision camera. For the other two sensors the filter has reduced the number of false detection but also the mAP, which means that the filter has discarded some detection that had been classified, in the computation of the mAP, as True Positive.

Y.O.L.O., instead, showed to achieve excellent results, but these results depend very much on the images used in the training dataset. In fact, on the images of the smartphone and the Official Pi camera has higher performance than the Haar Cascade, but on the datasets created with the MAPIR and the Night Vision cameras it is no longer able to recognize the extinguisher (§**9.3**). Also, in the comparisons between images with and without distortion it showed a dramatic decrease in performance. The best performance from Y.O.L.O. re-trainingwere obtained on the smartphone dataset, since even the training images were taken with the smartphone, therefore to get the most of Y.O.L.O.'s potential it is necessary to use images very similar to those of the training, so for example, if you want to use Y.O.L.O. on calibrated images of any sensor it is necessary to create the training dataset using calibrated images taken with the same sensor. Considering the results obtained, it was hypothesized that if grayscale images were used during training, Y.O.L.O. could be more adaptable to different sensors, because a sensor such as the Night Vision camera used works well both in good light and darkness conditions. Then images taken in both situations have a good contrast and so their grayscale version will be very similar.

In addition, Y.O.L.O. uses a CNN for learning and this makes it useless on a platform like Raspberry Pi that has not enough pervasive hardware components for the computation needed by a Neural Network.

One aspect of this work that did not produced the desired results was the use of detection and matching of features between the frames of a video stream. Independently of the sensor, this experiment has led to a remarkable slowing in the speed of frame processing, in fact the

fps resulting from the single detection are always greater than the fps resulting from the combination of descriptor and feature matching with the detection (§**9.5**).

Now, it is important to remember that the application field is the indoor navigation in unknown environments and therefore it was considered appropriate to find an object recognition algorithm, so that a UGV (Unmanned Ground Vehicle) can return not only a map of the navigated environment but also the location of known objects .

Another important aspect of this work is the choice of using low cost sensors and platforms, since with expensive tools and equipment there are already several platforms and datasets useful for autonomous driving, as the Benchmark KITTI dataset.

Furthermore, given that the use of descriptors and feature matching did not produce the desired result, that is making the detection faster. This purpose could be achieved differently thanks to the use of hardware accelerators, nowadays much used.

Comparing the objectives placed at the beginning and those achieved, we can be satisfied as the results obtained are good and promising, especially if we also consider the fact that ICT technologies are in continuous development. We have got a recognition algorithm able to work on a Raspberry Pi at a speed of 700 ms per image, which makes it possible to implement a real-time application with the ICT technology improvements.

Moreover, it is important to say that even if it has not been possible to test Y.O.L.O. on Raspberry Pi, there are some devices like "Intel Movidius Neural Compute Stick" which is defined as a small USB fanless deep learning unit designed to learn Artificial Intelligence programming. This product has been added under the VPU (Vision Processing Unit) class, similar to the classic GPU but for built-in purposes, and it supports the computation needed from a CNN so that, it can be used for many deep learning applications. Movidius has low energy consumption, which makes it perfectly applicable to embedded systems. The VPU includes 4 Gbit of LPDDR3 DRAM, imaging and vision accelerators and a set of 12 VLIW vector processors called SHAVE processors, that are used to accelerate neural networks by running parts of neural networks in parallel.

In conclusion, the study carried out has shown that ICT technologies can give a great contribution to the Smart Societies challenge of outdoor and indoor navigation in an unknown environment. In the near future, it is expected that the development of this sector will lead to the achievement of ever more performing results.

Appendix A

A.1 Smartphone tables

Case A

Table A.1: Haar Cascade mAP (Smartphone, A)

Ground truth	Training	IoU	mAP [%]	TP	FP
91	8	0,5	11,94	19	51
		0,4	33,37	37	33
		0,3	44,54	47	23
	9	0,5	13,64	28	49
		0,4	39,31	49	28
		0,3	41,01	50	27
	10	0,5	24,8	44	101
		0,4	36,27	56	89
		0,3	43,44	63	82

Case B

Table A.2: Haar Cascade mAP (Smartphone, B)

Ground truth	Training	IoU	mAP [%]	TP	FP
91	8	0,5	10,95	17	47
		0,4	32,43	36	28
		0,3	43.72	46	18
	9	0,5	13,13	26	44
		0,4	37,81	46	24
		0,3	41,3	48	22
	10	0,5	22,86	40	85
		0,4	29,09	48	77
		0,3	35,08	54	71

Case C

Table A.3: Haar Cascade mAP (Smartphone, C)

Ground truth	Training	IoU	mAP [%]	TP	FP
91	8	0,5	25,78	34	43
		0,4	41,08	46	31
		0,3	45,27	50	27
	9	0,5	31,31	40	29
		0,4	38,79	46	23
		0,3	38,79	46	23
	10	0,5	27,78	48	89
		0,4	36,99	57	80
		0,3	37,58	58	79

Case D

Table A.4: Haar Cascade mAP (Smartphone, D)

Ground truth	Training	IoU	mAP [%]	TP	FP
91	8	0,5	26,12	44	76
		0,4	33,81	52	68
		0,3	34,43	53	67
	9	0,5	32,11	38	23
		0,4	42,81	46	15
		0,3	42,81	46	15
	10	0,5	22,86	40	85
		0,4	29,09	48	77
		0,3	35,08	54	71

A.2 Official Pi camera tables

Case A

Table A.5: Haar Cascade mAP (Official Pi camera, A)

Ground truth	Training	IoU	mAP [%]	TP	FP
79	8	0,5	20,86	22	8
		0,4	24,93	24	6
		0,3	30,03	26	4
	9	0,5	2,7	12	72
		0,4	2,7	12	72
		0,3	2,7	12	72
	10	0,5	12,58	27	103
		0,4	13,7	28	102
		0,3	13,7	28	102

Case B

Table A.6: Haar Cascade mAP (Official Pi camera, B)

Ground truth	Training	IoU	mAP [%]	TP	FP
79	8	0,5	21,88	22	6
		0,4	26,04	24	4
		0,3	31,01	26	2
	9	0,5	3,49	12	47
		0,4	3,49	12	47
		0,3	3,49	12	47
	10	0,5	12,91	24	78
		0,4	13,92	25	77
		0,3	13,92	25	77

Case C

Table A.7: Haar Cascade mAP (Official Pi camera, C)

Ground truth	Training	IoU	mAP [%]	TP	FP
79	8	0,5	14,68	18	13
		0,4	23,46	23	8
		0,3	25,06	24	7
	9	0,5	1,05	8	71
		0,4	2,59	11	68
		0,3	4,74	15	64
	10	0,5	7,23	19	108
		0,4	8,93	21	106
		0,3	10,98	25	102

Case D

Table A.8: Haar Cascade mAP (Official Pi camera, D)

Ground truth	Training	IoU	mAP [%]	TP	FP
79	8	0,5	15,12	18	11
		0,4	24,21	23	6
		0,3	25,8	24	5
	9	0,5	1,42	8	49
		0,4	3,22	11	46
		0,3	5,91	15	42
	10	0,5	8,4	19	79
		0,4	10,3	21	77
		0,3	13,19	25	73

A.3 Night Vision camera tables

Case A

Table A.9: Haar Cascade mAP (NV camera, A)

Ground truth	Training	IoU	mAP [%]	TP	FP
80	8	0,5	1,67	2	5
		0,4	2,2	3	4
		0,3	6,25	5	2
	9	0,5	9,44	9	12
		0,4	12,67	11	10
		0,3	12,67	11	10
	10	0,5	6,29	8	25
		0,4	11,81	13	20
		0,3	18,66	18	15

Case B

Table A.10: Haar Cascade mAP (NV camera, B)

Ground truth	Training	IoU	mAP [%]	TP	FP
80	8	0,5	1,67	2	4
		0,4	1,67	2	4
		0,3	6,25	5	1
	9	0,5	9,44	9	12
		0,4	12,67	11	10
		0,3	12,67	11	10
	10	0,5	6,57	8	16
		0,4	12,38	13	11
		0,3	20,39	18	6

Case C

Table A.11: Haar Cascade mAP (NV camera, C)

Ground truth	Training	IoU	mAP [%]	TP	FP
80	8	0,5	3,52	5	8
		0,4	3,52	5	8
		0,3	13,75	11	2
	9	0,5	1,38	2	20
		0,4	3,28	5	17
		0,3	12,32	11	11
	10	0,5	2,16	4	19
		0,4	7,12	8	15
		0,3	14,02	13	10

Case D

Table A.12: Haar Cascade mAP (NV camera, D)

Ground truth	Training	IoU	mAP [%]	TP	FP
80	8	0,5	3,52	5	7
		0,4	3,52	5	7
		0,3	13,75	11	1
	9	0,5	1,42	2	16
		0,4	2,46	4	14
		0,3	12,71	11	7
	10	0,5	1,67	3	18
		0,4	7,17	8	13
		0,3	14,44	13	8

A.4 MAPIR tables

Case A

Table A.13: Haar Cascade mAP (MAPIR, A)

Ground truth	Training	IoU	mAP [%]	TP	FP
65	8	0,5	1,64	4	28
		0,4	27,24	23	9
		0,3	43,85	30	2
	9	0,5	11,68	26	74
		0,4	22,96	38	62
		0,3	27,19	42	58
	10	0,5	29,34	36	130
		0,4	59,6	57	109
		0,3	61,71	58	108

Case B

Table A.14: Haar Cascade mAP (MAPIR, B)

Ground truth	Training	IoU	mAP [%]	TP	FP
65	8	0,5	1,64	4	28
		0,4	27,24	23	9
		0,3	43,85	30	2
	9	0,5	9,67	22	70
		0,4	20,51	34	58
		0,3	24,34	38	54
	10	0,5	30,9	34	96
		0,4	61,76	53	77
		0,3	63,99	54	76

Case C

Table A.15: Haar Cascade mAP (MAPIR, C)

Ground truth	Training	IoU	mAP [%]	TP	FP
65	8	0,5	1,35	4	31
		0,4	24,62	23	12
		0,3	40,09	29	6
	9	0,5	18,01	32	77
		0,4	31,86	45	64
		0,3	36,35	48	61
	10	0,5	34,82	38	116
		0,4	64,95	58	96
		0,3	64,95	58	96

Case D

Table A.16: Haar Cascade mAP (MAPIR, D)

Ground truth	Training	IoU	mAP [%]	TP	FP
65	8	0,5	1,35	4	31
		0,4	24,62	23	12
		0,3	40,09	29	6
	9	0,5	14,39	32	77
		0,4	27,92	45	64
		0,3	32,2	48	61
	10	0,5	37,77	37	89
		0,4	70,83	57	69
		0,3	70,83	57	69

Bibliography

1) B. Rohini and J. Sreekantha Reddy. "*Sensors in Unmanned Robotic Vehicle*". In: Defence Science Journal, Vol. 58, No. 3, May 2008, pp. 409-413.
2) Dr. Georgios A. Demetriou. "*A Survey of Sensors for Localization of Unmanned Ground Vehicles (UGVs)*". In: Frederick Institute of Technology 3080, Lemesos Cyprus.
3) Y. LeCun, Y. Bengio, G. Hinton. "*Deep learning*". In: *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.
4) Hakan Koyuncu, Shuang Hua Yang. "*A Survey of Indoor Positioning and Object Locating Systems*". In: *IJCSNS International Journal of Computer Science and Network Security*, VOL.10 No.5, May 2010
5) D. E. Rumelhart, G. E. Hinton and R. J. Williams. "*Learning Internal Representations by Error Propagation*".
6) Gary Bradski and Adrian Kaehler. "*Learning OPENCV*". Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.
7) Zhong-Qiu Zhao, Member, IEEE, Peng Zheng, Shou-tao Xu, and Xindong Wu, Fellow. "*Object Detection with Deep Learning: A Review*". In: April 2019.
8) Paul Viola and Michael Jones. "*Rapid object detection using a boosted cascade of simple features*". In: Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on. Vol. 1. IEEE. 2001, pp. I–I.
9) Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. "*You Only Look Once: Unified, Real-Time Object Detection*". In: University of Washington, Allen Institute for AI, Facebook AI Research.
10) Joseph Redmon, Ali Farhadi. "*YOLOv3: An Incremental Improvement*". In: University of Washington, 2018.
11) Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. "*Rich feature hierarchies for accurate object detection and semantic sementation*". In: IEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.
12) Ross Girshick. "*Fast R-CNN*". In: IEEE International Conference on Computer Vision (ICCV), 2015.

13) Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. "*Faster R-CNN: Towards real-time object detection with region proposal networks*". In: Neural Information Processing Systems (NIPS), 2015.
14) Joseph Redmon, Ali Farhadi. "*YOLO9000: Better, Faster, Stronger*". In: University of Washington, Allen Institute for AI.
15) Ebrahim Karami, Siva Prasad, and Mohamed Shehata. "*Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images*". In: Faculty of Engineering and Applied Sciences, Memorial University, Canada
16) Dung, L., Huang, C. and Wu, Y. "*Implementation of RANSAC Algorithm for Feature-Based Image Registration*". In: (2013) Journal of Computer and Communications, **1**, 46-50. doi: 10.4236/jcc.2013.16009.

Sitography

1) https://www.kdnuggets.com/2018/02/top-20-python-ai-machine-learning-open-source-projects.html [14/07/19]
2) http://neuralnetworksanddeeplearning.com/chap6.html [08/07/19]
3) https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e [08/07/19]
4) https://github.com/tensorflow/tensorflow [04/07/19]
5) https://github.com/Microsoft/CNTK [04/07/19]
6) https://github.com/scikit-learn/scikit-learn [04/07/19]
7) https://github.com/keras-team/keras [04/07/19]
8) https://github.com/pytorch/pytorch [04/07/19]
9) https://github.com/BVLC/caffe [04/07/19]
10) https://github.com/NervanaSystems/neon [04/07/19]
11) http://pybrain.org/ [04/07/19]
12) http://image-net.org/ [14/07/19]
13) http://host.robots.ox.ac.uk/pascal/VOC/ [14/07/19]
14) http://cocodataset.org/#home [15/07/19]
15) https://docs.opencv.org/3.4.0/ [15/07/19]
16) https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html [15/07/19]
17) https://www.learnopencv.com/homography-examples-using-opencv-python-c/ [15/07/19]
18) https://github.com/tzutalin/labelImg [15/07/19]
19) https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/ [15/07/19]
20) https://pjreddie.com/darknet/yolo/ [14/07/19]
21) https://pjreddie.com/projects/pascal-voc-dataset-mirror/ [15/07/19]
22) https://picamera.readthedocs.io/en/release-1.13/ [14/07/19]
23) https://cdn.sparkfun.com/datasheets/Dev/RaspberryPi/ov5647_full.pdf [15/07/19]
24) https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e [15/07/19]
25) https://github.com/Cartucho/mAP [14/07/19]

26) https://www.kksb-cases.com/store/p99/Raspberry_Pi_Night_Vision_Camera.html [03/06/19]
27) https://picamera.readthedocs.io/en/release-1.13/api_camera.html [11/07/19]
28) https://www.raspberrypi.org/documentation/raspbian/applications/camera.md [12/07/19]
29) https://www.quora.com/Which-aperture-ISO-and-shutter-speed-should-I-use-for-daylight-or-night-photography [06/06/19]
30) https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html [11/07/19]
31) https://github.com/tizianofiorenzani/how_do_drones_work/tree/master/opencv [11/07/19]
32) http://www.cvlibs.net/datasets/kitti/ [12/07/19]
33) https://www.mouser.it/new/Intel/intel-movidius-stick/ [12/07/19]