
Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Ingegneria dell'Informazione e delle Comunicazioni

ELABORATO FINALE

# REMOTE IMAGE AND VIDEO OBJECT DETECTION USING YOLOV3

*Client and Server Infrastructure*

Supervisore
Fabrizio Granelli

Laureando
G. Alessio Dimonte

Anno accademico 2019/2020

# Ringraziamenti

***Questa Laurea è il frutto di tre intensi anni e senza l'aiuto di straordinarie persone lungo il mio percorso sarebbe stato decisamente più complicato arrivare mentalmente stabile al traguardo finale.***

***Un ringraziamento speciale alla mia famiglia: Mamma, Papà, Sorella, anche se non avete la minima idea del nome preciso del mio corso di Laurea vi ringrazio per avermi sempre sponsorizzato e appoggiato in ogni mia scelta ed avermi aiutato nei periodi più bui.***

***Ringrazio particolarmente anche Simone: senza il tuo supporto difficilmente avrei ritrovato la motivazione e la forza per tenere duro e dedicarmi a ciò che più amo.***

**INDEX**

# CHAPTER 1: INTRODUCTION

## 1.1) The problem

Object detection is one out of several problems related to Machine Learning applied to Computer Vision, but there are countless other types of applications, such as Face and Hand Gesture Recognition or Image Reconstruction.

The obvious question is: why is everyone speaking about Artificial Intelligence, Machine Learning and Deep Learning only now? The answer is simple. In fact, AI bases its functioning on three factors, namely Big Data, Computational Power and Algorithms, and even if in the past years already existed efficient algorithms able to analyze and develop applications in this field, the other two elements were weak or missing and especially in the most recent years exploded like never before, making AI accessible to many more people around the world.
As a matter of fact, in the last few years the availability of powerful and cheaper microchips boosted as never before, as well as the accessibility to Big Data, that we can formally think as the result of everything we do in the digital world, like sending a message or a tweet, taking a picture or buying something at the grocery shop with a credit card.

What this thesis proposes to develop is the option of performing remote object detection on a system portioned in a Server and a Client, meaning that the importance of having a strong computational power is reflected only Server-side.

## 1.2) The goal

The major purpose of this dissertation is the creation of an infrastructure able to communicate in a safe way and to perform object detection using one of the best state-of-art algorithms renowned in literature.

The infrastructure to be developed is separated in two parts, theoretically on two different computers and connected through the Internet. The first part is the Server, which has supposedly more powerful computational resources than any other device and with that the ability of resolving Machine Learning problems in a shorter time. The second part is a Client, who owns just a connection to the Internet and requests to the Server the elaboration of a content (a video or just an image), which is also stored by the Client in order to watch it at a later time.

The algorithm used for running object detection is YOLO, acronym of "You Only Look Once", which is known as one of the best available algorithms to perform real-time object detection: in fact, YOLO is able to compute at most 45 FPS if run on the best conditions possible.
In order to accomplish these results, the methods of analysis operated are:

- **Python**: programming language employed for writing the script
- **OpenCV for Python**: library employed to solve computer vision problems
- **YOLO v3**: algorithm employed for performing object detection

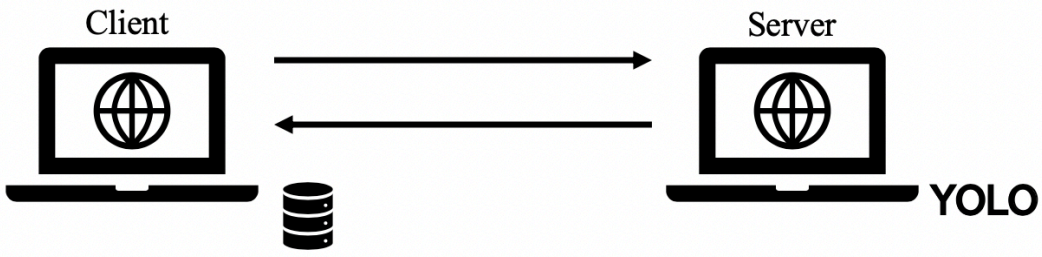



*Figure 1.a - Client and Server communicate through TCP socket: while the Server runs YOLO's algorithm and sends back to the Client the outcomes of the elaboration, the Client stores the results processed Server-side*

## 1.3) The results

The following work aims to develop a system where Client and Server communicate in a safe way through the use of TCP sockets and Cryptographic functions, performing remote object detection, but also giving the possibility of storing the results Client-side of the elaboration done Server-side, in order to watch them at a later stage.

During the thesis is shown that Client and Server are able to communicate correctly and that the system, even if it is far from doing a real-time object detection in terms of FPS, is able to successfully perform remote object detection and to send different kinds of data through the TCP sockets, of which Client and Server are equipped.

## 1.4) Thesis' structure

The chapter headed "Object detection using YOLO" presents how to perform Image Analysis using Machine Learning. Afterward, the chapter focus on YOLO's algorithm, its network architecture and a short comparison with the other object detection algorithms known in literature.

The chapter headed "Illustration of the developed system" guides the reader in understanding the procedure followed in order to accomplish the final result, explaining the most meaningful parts, such as YOLO's algorithm, the use of Python' socket module and the implementation of the Cryptographic part.

The chapter headed "Experimental results" states if the system works in a correct way or not and how the errors are handled. It also shows the results obtained comparing the use of different image resolutions to perform object detection. In addition to that, particular emphasis is placed on the analysis of the bandwidth used, with the support of the "Wireshark" packet sniffer software.

The chapter headed "Conclusion" closes the work, summing up the reasons why YOLO is one of the best algorithms for object detection and focusing on the advantages that the separation in Client and Server could lead to. In addition to that, it introduces new cues regarding future developments and improvements that could be added to the project.

At the end of the thesis is presented an additional part entitled "Glossary", where I decided to organize alphabetically all the specific terms used during the dissertation instead of giving their definition throughout the thesis. In this way if the reader already knows the meaning of the detailed words employed, he can continue the reading without interruptions.

# CHAPTER 2: OBJECT DETECTION USING YOLO

In this chapter the theory behind the functioning of YOLO is explained: starting from a brief overview about Computer Vision and Machine Learning, the focus is then shifted on the explanation of YOLO algorithm, its architecture, its strengths and weaknesses and a rapid comparison with other object detection algorithms known in literature.

## 2.1) Computer Vision and Machine Learning: Image Analysis

Computer Vision in combination with Machine Learning gives the opportunity to study plenty of cases in almost every imaginable field. For example, Computer Vision enables computer scientists to perform Image Analysis, extracting data with the aim of Artificial Intelligence: for this area, a Convolutional Neural Network is applied because of its capability of analyzing effectively images and videos.

**Reminder:**

- A Convolutional Neural Network (CNN) is a class of Deep Neural Network employed to analyze images and videos.
- A Deep Neural Network (DNN) is an Artificial Neural Network with multiple layers between the input and the output layers: its work is to find the correct mathematical manipulation to turn the input into the desirable output.
- An Artificial Neural Network (ANN) is a computing system inspired by the biological neural networks that constitute animal brains, it is trained by giving a sufficient number of examples: in this way the network automatically becomes capable of predicting outputs from inputs using the associations created during the training.

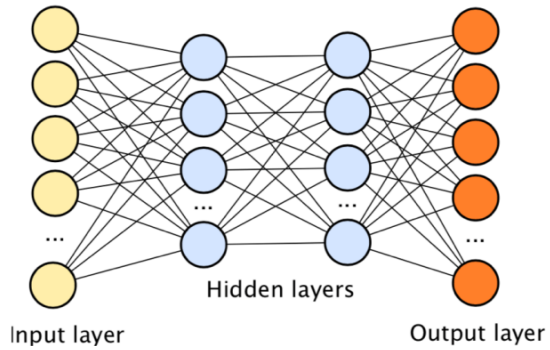



*Figure 2.a – Artificial Neural Network Architecture*

### 2.1.1) How a CNN works

1. A Convolutional Neural Network takes as input an image, which is a 2D array of pixels
2. To the input image are then applied different filters [q] or feature detectors [k, l] to generate as output feature maps [m], that are spatially small compared to the input image
3. Multiple convolutions are performed in parallel by applying non-linear function ReLU [a] to the convolutional layer
4. Then, Pooling [w] is applied to the convolutional layer: this technique helps with translational invariance, which implies that an object will be still recognized as the same object, even when an image is rotated, size differently or viewed in a different way
5. The pooled layer to input is then flattened [n] to a fully connected neural network [13]

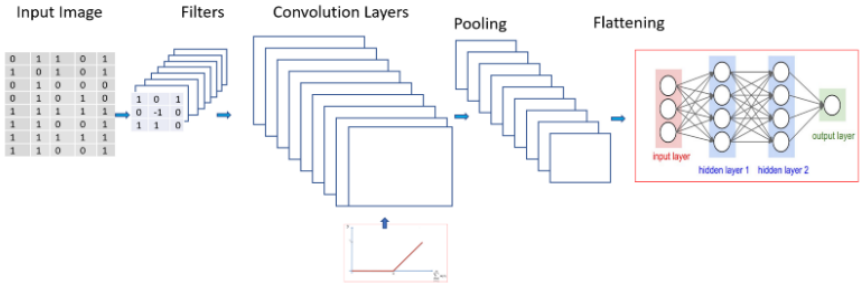



*Figure 2.b – Convolutional Neural Network Architecture, from [15]*

### 2.1.2) Object recognition challenges

Computer Vision and Artificial Intelligence in general have several fascinating problems and thanks to the enormous improvements in computational power we had in the recent years, it is possible to approach these problems much more easily than in the past years.

Talking about Object Recognition problems, they can be grouped in four different categories. The first problem is designated as Image Classification. Its purpose is to assign a single category out of several different categories for a single image. Its major problem is that a single image can only have maximum one assigned category. Secondly, we have Object Localization, which implements ways to locate the object in the image.
Those two methods can be only applied for single objects: in fact, if our goal is to find all the possible objects in an image, we need to use a different method named Object Detection. This method is particularly useful in real life scenarios and it is able to identify multiple objects, drawing the so-called bounding boxes around them. If we want to go further and find the exact boundaries of our objects at the pixel level, we can use Instance Segmentation.

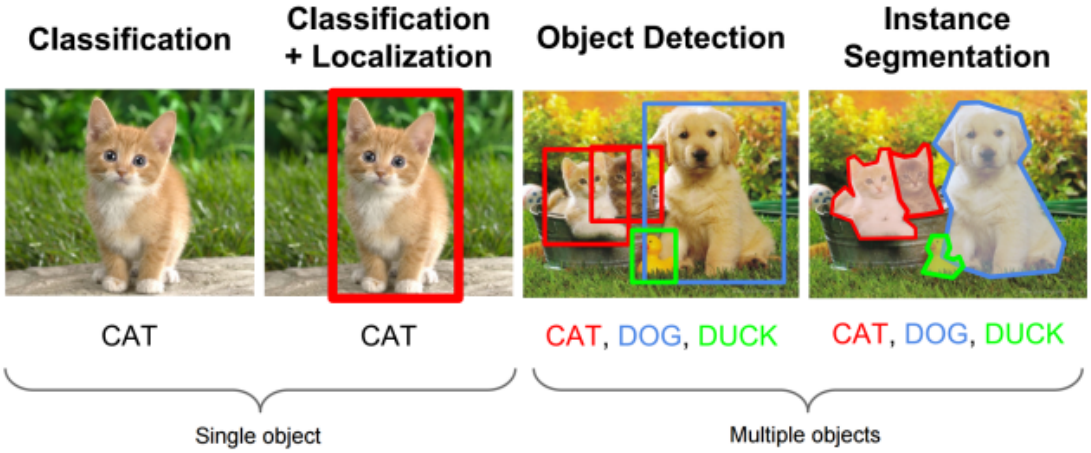



*Figure 2.c – Object Recognition Problems, from [16]*

## 2.2) What is YOLO*

YOLO is an object detection algorithm based on a CNN-technique, which predicts the presence of an object based on the different parts of the image that have higher probability to contain the object.
This algorithm was developed by Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi. The first version was described in 2015 in the paper titled "You Only Look Once: Unified, Real-Time Object Detection" and thanks to its astoundingly speed it can be used for real-time object detection.
Up to now YOLO is in its 5$^{th}$ version and has reached important improvements in terms of performance and error-handling.

### 2.2.1) How YOLO works

**How YOLO performs object detection**

Using YOLO, object detection is reframed as a single regression problem directly from image pixels to bounding box coordinates and associated class probabilities: it is because of that that it is extremely fast compared to the other object detection systems.

In Machine Learning a regression problem is a technique of the "Supervised Machine Learning" family. Supervised Machine Learning is used in the majority of practical Machine Learning uses and its method of operating is that, given an input variable X and an output variable Y, the algorithm learns the mapping function from the input to the output: Y = f(X). In addition to that, Supervised learning problems can be grouped into two smaller problems: "Regression" and "Classification". The regression problem (used in YOLO) is when the output variable is a real or continuous value, such as "salary" or "weight", while the classification problem is when the output variable is a category, like "green" or "disease". [14]

*most of this section is taken from [1, 2, 3]

Methods like R-CNN, Fast and Faster R-CNN and SPP-net handle detection as a classification problem by building a pipeline [v] where first object proposals are generated and then these proposals are sent to classification or regression heads. Other methods, like YOLO and SDD (Single Shot Detector), pose detection as a regression problem.

YOLO is particularly simple: in fact, a single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. Moreover, YOLO sees the entire image during training and test time, in this way it reasons globally about the image when it is called to make a decision. The ability of seeing the entire image enables the system to implicitly encode contextual information about classes: for example, if YOLO recognizes a laptop, it is likely that it also recognizes the desk where it is placed, since during the training it has encoded that a laptop needs a surface where it needs to be leaned.

**YOLO's accepted image sizes**
YOLO accepts three sizes, for which it is able to extract features from the image with different levels of accuracy and speed analysis:

- **320x320**: it is the smallest resolution, with lower accuracy but a faster speed
- **416x416**: it is the middle resolution, with balanced accuracy and speed between the other two accepted resolutions
- **608x608**: it is the highest resolution, with higher accuracy but a slower speed

**YOLO's algorithm**
The main steps followed by the YOLO's algorithm are:

1. YOLO divides the input image into an SxS grid, and each grid cell predicts one object
2. Each grid predicts a fixed number of boundary boxes, however the one-object rule limits how close detected objects can be
3. For each grid cell YOLO:
    - Predicts the boundary boxes and for each box computes a box confidence score: the appropriate bounding box is selected as the bounding box with highest IoU (Intersection over Union) between the ground truth [o] box and the anchor box [b]
    - Detects only one object, independently of the number of boxes
    - Predicts the conditional class probabilities (one per class)

Each boundary box contains 5 elements:
- (x, y) are the coordinates that represent the center of the box relative to the bounds of the grid cell
- (w, h) are the width and the height, which are predicted relatively to the whole image
- The confidence prediction represents the IoU between the predicted box and the ground truth box

**Intersection over Union**
IoU is the acronym of Intersection over Union and it is calculated with the following formula:

$$\text{IoU} = \frac{\text{Overlapped area between ground truth and predicted bounding boxes}}{\text{Total area of ground truth and predicted bounding boxes}}$$

**Reminder:**
When IoU is equal to 1, it means that the predicted bounding box and the ground truth bounding box perfectly overlap.

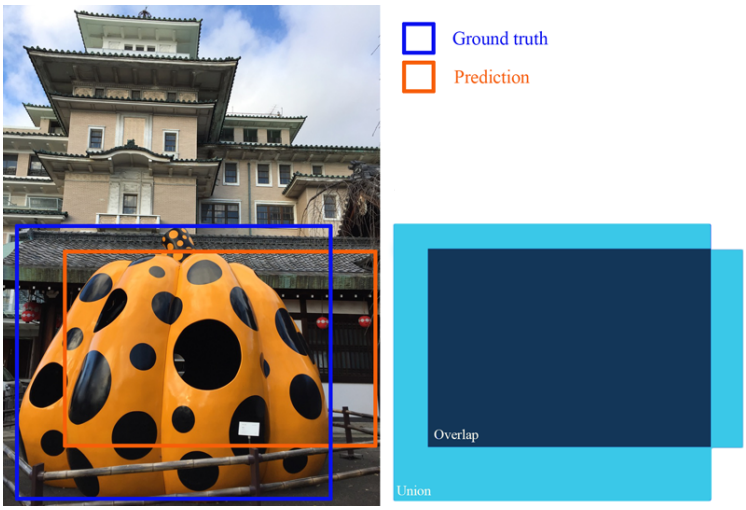



*Figure 2.d – Visual Representation of IoU, from [17]*

**Confidence Score**

The Confidence Score reflects how likely the box contains an object (also called Objectness) and how accurate the boundary box is. Formally the confidence is:

$$\text{Confidence Score} = \Pr(\text{Object}) * \text{IoU}^{\text{truth}}_{\text{prediction}}$$

- If the object does not exist in the cell the confidence scores should be zero
- Otherwise we want the confidence score to be equal to the IoU between the predicted box and the ground truth

**Conditional Class Probability**

The Conditional Class Probability is the probability that the detected object belongs to a particular class and it is conditioned on the grid cell containing an object. Formally the conditional class probability is:

$$\text{Conditional Class Probability} = \Pr(Class_i | \, Object)$$

**Class Specific Confidence Score**

At test time we multiply the Conditional Class Probability and the individual Box Confidence Predictions: the result is a class-specific confidence score for each box. These scores encode both the probability for the class appearing in the box and how well the predicted box fits the object.

$$\text{Class Specific Confidence Score for selected box} = \Pr(Class_i|Object) * IoU^{truth}_{pred} = \Pr(Class_i) * IoU^{truth}_{pred}$$

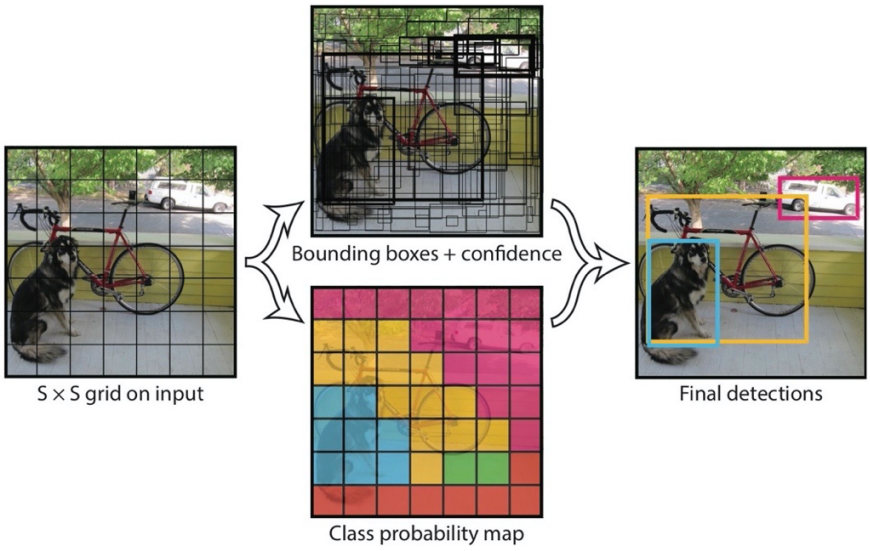



*Figure 2.e –YOLO's Algorithm Visualization, from [18]*

**Non-Max Suppression**
It may happen that YOLO can make duplicate detections for the same object: in order to fix this, Non-Max Suppression is applied as a means of removing duplications with lower confidence. In this way it is added a 2-3% in mAP, acronym of "mean Average Precision" (a popular metric in measuring the accuracy of object detectors). Non-Max Suppression makes sure that the object detection algorithm only detects each object once, removing boxes with low object probability and bounding boxes with the highest shared area.

This algorithm works in a very simple way:

1. First, it sorts the predictions by the confidence scores
2. Then, starting from the top scores, it ignores any current prediction if any previous prediction with the same class and IOU<0.5 with the current prediction is found
3. Lastly, the second step is repeated until all predictions are checked

### 2.2.2) Network architecture

YOLO model is implemented as a Convolutional Neural Network and it is evaluated on the PASCAL VOC detection dataset, used for building and evaluating algorithms for image classification, object detection and segmentation.

The initial convolutional layers of the network extract features from the image, while the fully connected layers predict the output probabilities and the coordinates.

The network architecture is inspired by the "GoogLeNet" model for image classification and it can be described as it follows:

- YOLO's network has 24 convolutional layers in its first version and 53 in its third one, followed by 2 fully connected layers
- Instead of the Inception modules used by GoogLeNet, 1x1 reduction layers are used alternatively to reduce the depth of the features maps and they are followed by 3x3 convolutional layers
- Convolutional layers are pre-trained on the ImageNet classification task at half the resolution (224x224 input image) and only then the resolution is doubled for detection
- YOLO uses a linear activation function [a] for the final layer and a leaky ReLU for all the other layers
- YOLO predicts the coordinates of the bounding boxes with the aim of the fully connected layers on top of the convolutional feature extractor

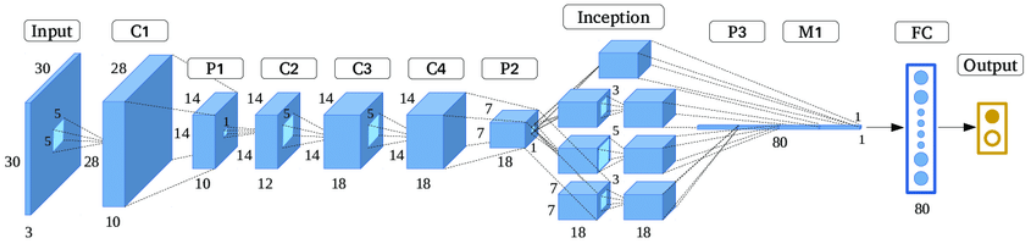



*Figure 2.f – GoogleNet Model, from [19]*

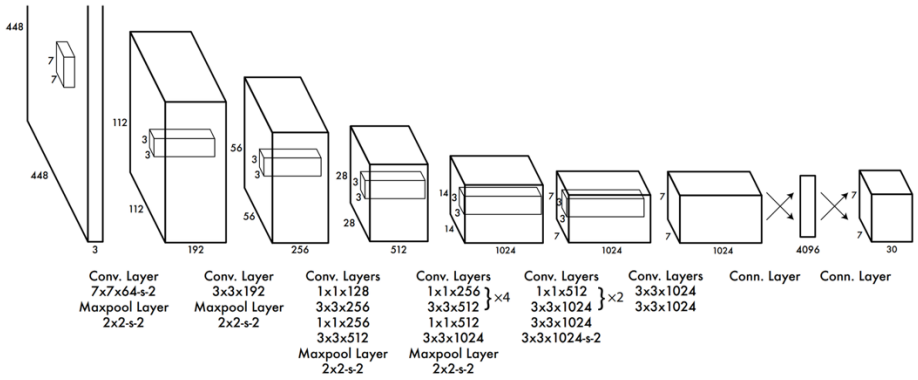



*Figure 2.g – YOLO Model, from [18]*

### 2.2.3) Advantages and Disadvantages

**Strengths:**

- YOLO is extremely fast if compared to other object detection algorithms
- YOLO sees the entire image during training and test time, so it implicitly encodes contextual information about classes and their appearance
- YOLO learns generalizable representation of objects so that, when it is trained on natural images and it is tested on artwork, the algorithm outperforms other top detection methods

**Weaknesses:**

- YOLO enforces spatial constraints on bounding box predictions: in this way each grid cell predicts two boxes and can only have one class. This spatial constraint limits the number of adjacent objects that the model is able to predict
- YOLO struggles with small objects that appear in groups, like flocks of birds
- YOLO struggles to generalize objects in new or unusual aspect ratios or configurations, since the model learns to predict bounding boxes from data
- YOLO uses relatively coarse features for predicting bounding boxes since the architecture has multiple down sampling layers from the input image
- Because the model is trained on a loss function [t] that approximates detection performance, this loss function treats errors the same in small bounding boxes as well as large bounding boxes: in this way a small error in a large box is generally benign, while a small error in a small box has a greater effect on IoU
- The main source of error is incorrect localization

### 2.2.4) Different versions of YOLO

For the development of this project it was used YOLOv3, which presents several improvements over the previous versions. However, the earlier description of YOLO is still valid because the operating of YOLO remains the same and the changes done concerns the architecture and the training model, not the approach to the object detection, which remains a single regression problem directly from image pixels to bounding box coordinates and class probabilities.

The major differences that distinguish YOLOv3 from the previous versions are: [3]

- It uses 9 anchors instead of the 5 used in YOLOv2: having more anchor boxes means having a higher IoU
- It uses logistic regression [s] to predict the objectiveness score, instead of the Softmax function [y] used in YOLOv2: YOLOv3 replaces the Softmax function with independent logistic classifier [e] to calculate the probability of the input belongs to a specific label (named also objectness score for each bounding box)
- It uses the Darknet-53 network for feature extractor, which has 53 convolutional layers: having more convolutional layers gives the possibility to increase the accuracy of the network and helps to extract more features, but it is important to consider that having too much convolutional layers can lead to an overfit [u] of the model
- It uses multi-label classification: the same output label can be both “pedestrian” and “child”, which are non-exclusive

### 2.2.5) Comparison to other object detection systems

Object detection is a fundamental problem in Computer Vision. Detection pipelines start by extracting a set of robust features from input images, then classifier and localizers are used to identify objects in the feature space. These classifiers/localizers are run either in sliding-window-mode over the whole image or on some subset of regions in the image.

**DPM**

How DPM works

Deformable parts model uses a sliding window approach to object detection. It uses a separate pipeline to extract static features, classify regions and predict bounding boxes for high scoring regions.

How is YOLO different

YOLO replaces all of the disparate parts with a single convolutional neural network. Instead of static features, YOLO trains the features in-line and optimizes them for the detection task. Having a unified architecture leads to a faster and more accurate model than DPM.

**R-CNN**

How R-CNN works

R-CNN uses region proposals instead of sliding windows to find objects in images.

- Selective search [x] generates potential bounding boxes
- A convolutional network extracts features
- A SVM [z] scores the boxes
- A linear model [r] adjusts the bounding boxes
- Non-Max Suppression deletes duplicate detections

Each stage of this complex pipeline must be precisely adjusted independently and the resulting system is very slow: more than 40 seconds per image at test time.

How is YOLO different

Even if in both networks each grid proposes potential bounding boxes and scores those boxes using convolutional features, YOLO puts spatial constraints on the grid cell proposals which helps mitigate multiple detection of the same object. Furthermore, YOLO proposes far fewer bounding boxes: 98 instead of the 2000 from Selective Search. Lastly, YOLO combines the individual components into a single and connected optimized model.

**Fast R-CNN and Faster R-CNN**

How Fast R-CNN and Faster R-CNN work

Both networks focus on speeding up the R-CNN framework by sharing computation and using neural networks to propose regions instead of Selective Search.

How is YOLO different

YOLO throws out the pipeline entirely, instead of trying to optimize individual components of a large detection pipeline. Secondly, YOLO is a general-purpose detector that learns to detect a variety of objects simultaneously, rather than optimize the detection.

# CHAPTER 3: ILLUSTRATION OF THE DEVELOPED SYSTEM

In this chapter is presented how the system was built, starting from the basics and explaining the different steps followed until reaching the final result. On top of that, the libraries used and the procedures followed are explained using snippets of code, tables, flow diagrams and images.

## 3.1) Goal

The main goal of the system is to build a Client-Server interface able to do remote object detection: the Client must be able to capture frames and to send those frames over a TCP socket to the Server, which runs the YOLO algorithm and sends back to the Client the results for the processed frame.
The reason for building a similar system is the need of having a strong computational power only Server-side, but still be able to do object detection on a Client with low computational power and an available Internet connection.
Since it is wished that, both Client-side and Server-side, the data transferred satisfies data integrity, TCP (Transmission Control Protocol) and a Cryptographic function named HMAC (Hashed MAC) are used. The first one guarantees that the data communication between the two parts is reliable, the latter assures that the data received are not being read or manipulated by a man-in-the-middle.

## 3.2) Hypothesis

In order to see YOLO working it is needed to install a Deep Learning Framework [h] where it is possible to run the algorithm. In this project the Framework used is OpenCV, the best beginner's approach for doing Deep Learning. In fact, it is ready to use and it does not require any intricate installation process, like other Frameworks (e.g. Darknet or Darkflow). The major disadvantage of OpenCV is that, unlike other Frameworks, it only works with CPU, meaning that it cannot reach really high speed in terms of FPS (reachable instead with the use of a GPU).

## 3.3) Needed resources

| NAME | TYPE | VERSION | DESCRIPTION |
|---|---|---|---|
| OpenCV [6] | Python library | 3.4.2 or higher | Library designed to solve computer vision problems |
| NumPy [7] | Python library | 1.18.0 or higher | Open source mathematical library |
| PyCrypto [8] | Python library | 2.6.1 or higher | Python Cryptography toolkit |
| Socket [9] | Python library | Installed along with the current Python version | It provides access to the BSD socket interface [d] |
| Pickle [10] | Python library | Installed along with the current Python version | Library for Python object serialization [g] |
| Struct [11] | Python library | Installed along with the current Python version | Library for interpreting bytes as packed binary data |
| yolo3.weights | Binary weight file | - | It is the trained model, which is the core of the algorithm to detect the objects |
| yolo3.cfg | Configuration file | - | It is the configuration file, where there are all the settings of the algorithm |
| coco.names | Names list file | - | List of all the possible detectable objects for which the training has been made |

*Table 3.a – Required Modules*

### 3.4) Description of the system step by step

The developed system is the final result of a series of different steps: I followed an approach based on Divide and Conquer, splitting the bigger task into smaller, various and simpler problems and then merged them together in the final system. I decided to use a step by step approach in order to fully understand how the different slices were working stand-alone, being able to implement them in the complete project and being capable of comprehending in advance how they would behave in combination with the other parts.
The chronological order of the steps I followed was:

1. YOLO object detection using OpenCV with Python
2. YOLO real-time detection on CPU
3. Client-Server interface
4. Client-Server with Pickle: "loads()" and "dumps()" functions
5. Client-Server: sending data over a socket
6. Client-Server: UDP sockets
7. Video storing Client-side
8. Cryptography: RSA and HMAC
9. Installation and implementation of the Virtual Machine environment

The steps 4 and 5 are interconnected and they cannot work alone, by the way for giving a clear explanation of their functioning I am going to explain them separately anyway.

#### 3.4.1) YOLO object detection using OpenCV with Python

Firstly, I followed an online lesson by Sergio Canu for YOLO working with OpenCV and Python [4]. Thanks to it, I understood how YOLO code works and what are the different meanings of the code's chunks.

First of all, we need to load the algorithm and for doing so we need three essential files:

- **yolov3.weights**: it is the trained model used to detect the objects
- **yolov3.cfg**: it is the configuration file where there are the different settings of the algorithm
- **coco.names**: it contains the names of the objects that the algorithm can detect and it consists in eighty different classes (i.e. person, microwave, carrot)

Next, we need to load the image we want to analyze, but we need to take into consideration that YOLO accepts only three sizes, so we need to convert our image into a blob [c]:

$$\text{blob} = \text{cv.dnn.blobFromImage}(\text{imageToTransform}, \text{scaleFactor}, \text{size}, \text{meanValue}, \text{swapRB}, \text{crop})$$

Where:
- imageToTransform: it is the image/frame we want to transform
- scaleFactor: it is used in order to scale the pixel in the range [0-1]
- size: it is the size we choose between 320x320, 416x416 and 608x608
- meanValue: it is usually set 0 by default
- swapRB: it is an option available for YOLO, actually YOLO uses BGR instead of RGB with some frameworks
- crop: it is an option used if we want to crop our image or not

After the detection, our results are stored in an array named "outs" and each element of the array, in turn, stores three variables:
- **class_ids[]**: it is an array that stores the information about the detected objects
- **confidences[]:** it is an array that stores the confidence about the detections
- **boxes[]:** it is an array that stores the position of the detected objects

During the detection it is possible that we have more boxes for the same object and in order to have only one box per object we perform Non-Max Suppression:

```
indexes = cv2.NMSBoxes(boxes, confidences, scoreTreshold, NMSTreshold)
```

If we want to show the results on the screen, we need to extract the information contained in our array of arrays, visit each cell and display their content on the screen.

### 3.4.2) YOLO real-time detection on CPU

The second thing I did was following another tutorial by Sergio Canu, where he explained how to run YOLO on CPU and doing a real-time object detection [5]. In this section the presenter states and makes the listeners aware that running YOLO on CPU is way slower than running it on a GPU. The reason why he continued with the explanation is because running YOLO in combination with OpenCV is straightforward and it is the best procedure that a beginner can follow as a means of doing object detection.

The main differences with the "static" object detection are two:

- firstly, with real-time object detection we are interested in calculating how many FPS our system is processing and for doing so we need to get the starting time, the elapsed time and the frame ID
- secondly, we are going to read the various frames from our input file and for doing so we use the following OpenCV function:

  ```
  cap = cv2.VideoCapture(source)
  ```

**Reminder:**
"Source" can be the video captured from the webcam (that we can call by putting source=0) or a video that we have stored on the computer and in this last case we need to put the name of the video file, such as "videosource.mp4"

After that, we need a loop to read our video up until there are frames to analyze and the way for doing it is by putting inside the cycle the following function:

```
ret, frame = cap.read()
```

The next steps (the detection and the displaying of the results on the screen) are identical to the "static" version.

As it was mentioned before, the only difference is the calculation of the FPS, which is done in this way:

- At the beginning we have calculated the starting time with:

  ```
  starting_time = time.time()
  ```

- At the end of the elaboration we calculate the elapsed time with:

  ```
  elapsed_time = time.time() - starting_time
  ```

- Having the frame ID and the elapsed time, we can easily calculate the FPS as:

  ```
  FPS = frame_ID/elapsed_time
  ```

### 3.4.3) Client-Server interface

The third step was creating a Client-Server interface able to send frames and information. I used the Python module "socket" and I proceeded with the connection between the two parts with a try-except block. In addition to that, I also reorganized the YOLO algorithm, assigning different sections to the Client and to the Server in my script.

The main methods of the "socket" module are:

- **socket.socket():** it is used to create sockets and it is required that both Client and Server have a socket
- **socket.accept():** it is used to accept a connection and it returns two values (conn, address)
- **socket.bind():** it is used to bind the address specified as a parameter
- **socket.connect():** it is used to mark the Socket as closed
- **socket.listen():** it is used to connect a remote address specified as a parameter

The procedure embraced is:
Client-side

1. Creation of the socket with the command:

```
socket_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

   The socket we want to create is a socket with particular specs:
   - AF_INET designates the type of addresses that the socket can communicate with and in this case they are IPv4 addresses
   - SOCK_STREAM means that the socket created is a socket based on TCP

2. Connection to the Server with the command:

```
socket_client.connect((indirizzo_server, porta_client))
```

3. Sending the frame with the command:

```
socket_client.send(data)
```

4. Receiving the elaboration done by the Server with the command:

```
socket_client.recv(bufferSize)
```

Server-side

1. Creation of the socket with the command:

```
socket_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

2. Connection of the socket to the machine address with the command:

```
socket_server.bind("", porta_server)
```

3. Waiting the connection from the Client with the command:

```
socket_server.listen(backlog)
```

4. Acceptance of the client request of connection with the command:

```
conn, indirizzo_client = socket_server.accept()
```

5. YOLO elaboration

6. Sending the elaboration done in the previous step with the command:

```
conn.send(data)
```

### 3.4.4) Client-Server with Pickle: “loads()” and “dumps()” functions

The fourth step I followed was the use of the Python module named Pickle, which is a module used for Python object serialization. Using the methods “loads()” and “dumps()” it is possible to send objects through the socket in a manner that Python can understand.

With “dumps()” it is possible to convert the object to a serialized string:

```
data_to_send = pickle.dumps(data)
```

With “loads()” it is possible to de-serialize an object:

```
data_to_receive = pickle.loads(data)
```

Pickle, as it is indicated in its documentation, is extremely unsecure and the idea of implementing a cryptographic part (that I implemented subsequently) came up to my mind while I was reading its paperwork and the related insecurity warning.

Before proceeding with the fifth step, we need to clarify that:

The data we want to send is different, depending if the sender is the Client or the Server:
- Client-side we want to send the frames to the Server
- Server-side we want to send the information resulting from the elaboration done to the Client

The data we want to receive is also different, depending if the receiver is the Client or the Server:
- Client-side we want to receive the information resulting from the elaboration done Server-side
- Server-side we want to receive the frames by the Client

If we want to send a frame, we need to follow this formula:
- Encode our frame
- Serialize the codified frame
- Pack the serialized data

If we want to receive a frame, we need to follow this formula:
- Unpack the received data
- De-serialize the unpacked data
- Decode the de-serialized information

**Reminder:**
if the data we want to send is not an image, we must skip the encoding and the decoding steps.

### 3.4.5) Client-Server: sending data over a socket

The fifth step I implemented was a way to send the frames or other data in general through the sockets, doing the streaming of the video file or returning the results of the elaboration done using YOLO Server-side.

Sender-side

Firstly, we need to encode our frame, in order to convert an image into a binary data information. To encode positively the frame, we use:

```
result, frame = cv2.imencode(".jpg", frame, encode_parameters)
```

Secondly, we need to serialize the codified frame with the Pickle method “dumps()”.

Finally, we need a way to pack the data, in order to successfully handle binary data:

```
socket.sendall(struct.pack(">L", size) + data_to_send)
```

Receiver-side

Firstly, we need to unpack the received packed data:

```
message = struct.unpack(">L", packed_message)
```

Secondly, we need to de-serialize the unpacked data with the Pickle method “loads()”.

Finally, we need to decode the received information and convert it back to an image:

```
frame = cv2.imdecode(frame, cv2.IMREAD_COLOR)
```

### 3.4.6) Client-Server: UDP socket

I tested just for personal interest how the streaming of the video was working with UDP (User Datagram Protocol) sockets and to check if it was in some way possible to send the frames faster and to save time.
I saw with my own eyes why Transmission Control Protocol is so useful. In the Figure 3.b we can notice that the frame exchanged has some kind of error regarding the transmission: the frame is not transmitted correctly and we can assume that by looking carefully at the image, which is corrupted with some kind of noise and it is not the same as the one the client attempted to send.
In fact, differently than TCP, UDP does not manage the reordering of the packets and the retransmission of the lost ones, but it sends packets without control. It is for this reason that UDP is used in application like VoIP (Voice over IP), where a loss does not affect significantly the information exchanged.

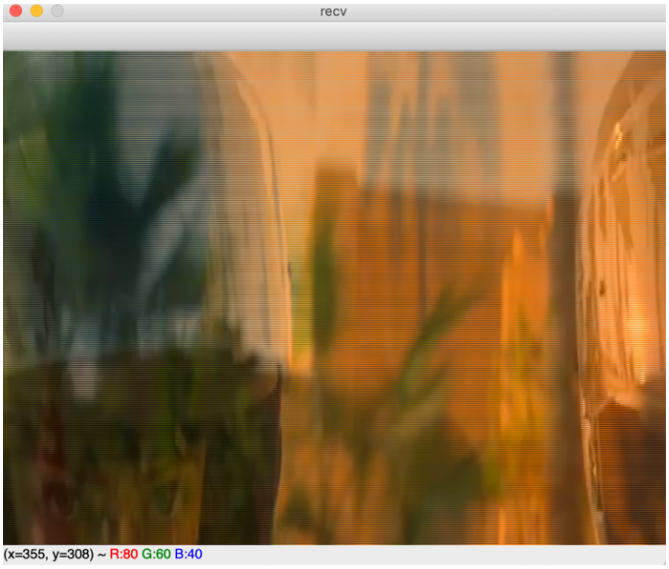



*Figure 3.b – Exchanging Frames Through UDP Sockets*

### 3.4.7) Video storing Client-side

The subsequent stage I implemented was a way of storing Client-side the video along with the results of the elaboration done Server-side, with the aim of watching it at a later stage.
For doing so, OpenCV already provides a method:

$$\text{outputVideo} = \text{cv2.VideoWriter}(\text{fileName}, \text{FourCC}, \text{FPS}, (\text{width}, \text{height}))$$

Where:

- filename: it is the name we want to assign to the output file we are going to create
- FourCC: it is the acronym of Four Character Code, used for expressing the codec [f] we prefer to use, like “MJPG” or “XVID”
- FPS: it is the parameter expressing how many FPS we want our video to have
- (width, height): they are the width and the height we want our video to have

### 3.4.8) Cryptography: RSA and HMAC

The eighth and last step I followed was the implementation of the cryptographic part. As I stated before, the idea of implementing this part came up to my mind when I was reading the Pickle module documentation, where it was stated that “the pickle module is not secure” and to “only unpickle data you trust” because “it is possible to construct malicious pickle data which will execute arbitrary code during unpickling”. [10]

In this project I decided to avoid following the standard technique with the implementation of Digital Certificates and Digital Signatures and in its place, I opted that the best way to have a safe communication was to exchange a shared key through RSA between the Client and the Server and use this shared key for operating HMAC both Server-side and Client-side.

**RSA**
RSA is the standard when we think about modern asymmetric cryptography algorithms. Its security is based on the problem of factoring large numbers: in fact, modern cryptography employs from 200 to 300 digits for doing the calculations and even with the power of modern computers it would take years before successfully break RSA encryption. The difficulty is based on the use of particular sets of numbers and the use of a one-way function, which is extremely simple to compute in one way (the encryption) and extremely difficult in the inverse way (decryption) if the key used is not known. [12]

RSA works in a very simple way:

- Alice and Bob are our entities, trying to communicate in a secure way
- Alice chooses two secret and big prime numbers p and q
- Alice computes a public value $n = p \cdot q$
- Alice chooses a number e, which is the public key value, such that $e \in Z^+ | \text{ g. c. d}[e, \emptyset(n)] = 1$
- Alice calculates d, which is the private key value, such that $d \mid ed = 1 \bmod[\emptyset(n)]$
- When Bob wants to send a message to Alice, he uses Alice's public key e:
$$\text{Ciphertext} = \text{Message}^e \bmod(n)$$
- When Alice wants to verify the message sent by Bob, she uses her private key d:
$$\text{Message} = \text{Ciphertext}^d \bmod(n)$$

**Reminder:**
$\emptyset(\circ)$ is called Euler's totient function. [j]

**HMAC**
My idea was giving to the system a way of exchanging information through a HMAC function. A HMAC function, also known as Hashed MAC, is a particular Message Authentication Code (MAC) that involves a cryptographic hash function [p] and a secret cryptographic key, and it may be used to verify simultaneously data integrity and authenticity of a message. [12]

Server-side the HMAC is going to verify if the frame is properly received, while Client-side the HMAC function is going to determine if the information elaborated by the Server was received correctly.

The HMAC function can be described as follows:

$$\text{HMAC}(\text{Key}, \text{Message}) = \text{Hash}\{(\text{Key}' \oplus \text{opad}) \,||\, \text{Hash}[(\text{Key}' \oplus \text{ipad}) \,||\, \text{Message}]\}$$

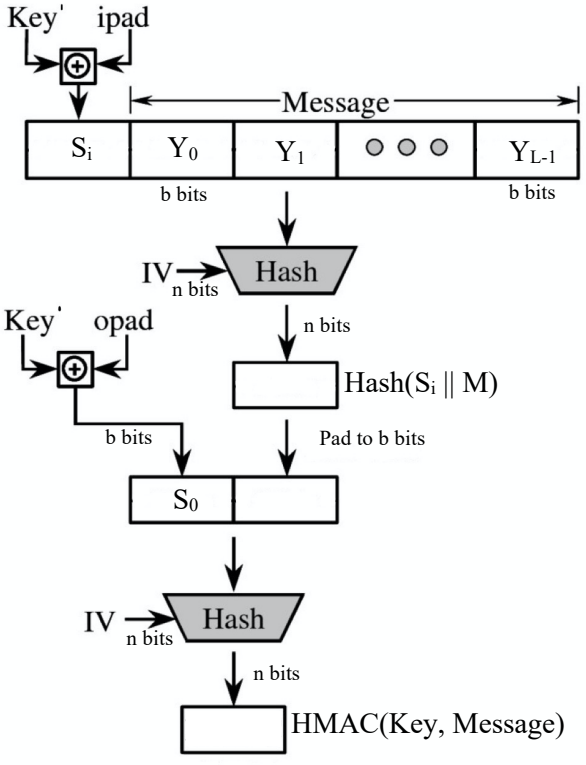



*Figure 3.c – HMAC Scheme, from [20]*

Where:

- **Key':** it is the key padded with 0s on the left until reaching a length b
- **b:** it is the processed block length in bits
- **ipad**: it is 0x36 repeated b/8 times
- **opad:** it is 0x5C repeated b/8 times
- **⊕:** it is the XOR operation
- **||:** it is the concatenation operator

**Implementation of the cryptographic part in my system:**
The way I implemented the security system can be summarized in this way:
Client-side:

- Creation of the RSA keypair:

```
def RSA_keys_generation_client(self)
```

- Sending of the RSA public key to the Server:

```
def sending_public_key_client(self, socket, public_key)
```

- Receiving of the shared key generated by the Server:

```
def receiving_shared_key(self, socket, private_key_client)
```

- Before sending the frame, the system creates a HMAC digest [i] and it appends this digest after the frame to send, through a Python dictionary:

```
def HMAC_digest_creation(self, message, shared_key)
```

- Before opening the received information (sent by the Server), which should be the result of the YOLO elaboration done Server-side, the system verifies the HMAC digest:

```
def HMAC_digest_verification(self, message, digest, shared_key)
```

Server-side:

- Receiving of the RSA public key of the Client:

```
def receving_public_key_client(self, socket)
```

- Generation of the shared key:

```
def shared_key_creation(self)
```

- Sending of the just created shared key:

```
def sending_shared_key(self, socket, shared_key, client_public_key)
```

- Before receiving the frame sent by the Client, the system verifies the HMAC digest:

```
def HMAC_digest_creation(self, message, shared_key)
```

- Before sending the result of the YOLO elaboration, the system creates a HMAC digest and it appends this digest after the other data:

```
def HMAC_digest_verification(self, message, digest, shared_key)
```

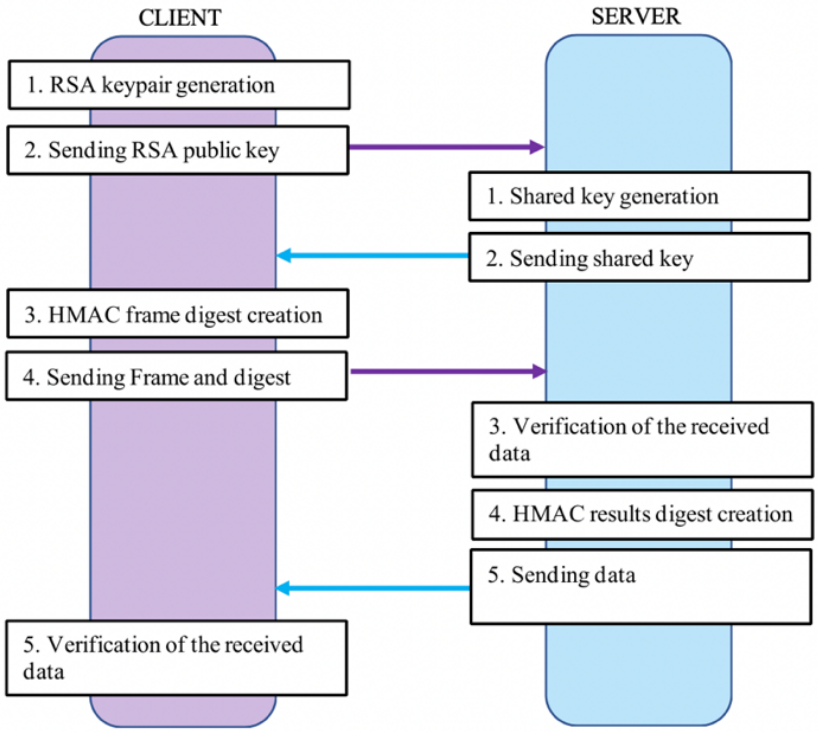



*Figure 3.d – Visual Representation of Cryptographic Exchange*

### 3.4.9) Installation and implementation of the Virtual Machine environment

In order to separate Client and Server on a single laptop and replicate that the two parts were on different computers, I created two different Virtual Machines using VirtualBox v6.1.10. On both machines I installed Ubuntu 20.04 and then I proceeded with the installation of the additional libraries and modules that my system requires to work properly, as it is explained below.

- First of all, in order to check the IP address of the two machines from the command line interface, I installed the net-tools with the command:

```
sudo apt install net-tools
```

- Then, I installed the Python module "Pip" and OpenCV for Python with the commands:

```
sudo apt install python3-pip
sudo apt install python3-opencv
```

- Afterwards, I installed "PyCrypto", the Cryptographic toolkit for Python with the command:

```
pip3 install pycrypto
```

- Finally, I checked if everything was updated, in the negative case I upgraded all the modules and libraries to their latest version with the commands:

```
sudo apt update
sudo apt upgrade
```

# CHAPTER 4: EXPERIMENTAL RESULTS

The implementation of the system works in the correct way. In fact, Client and Server are able to establish a safe communication, through which the two parts are able to exchange frames and the related elaboration done by the YOLO object detection algorithm. The major drawback is that the elaboration is extremely slow, reaching an elaboration of few frames per second.

## 4.1) Adapting the problem on my resources

I run my system on a computer with a "3.1 GHz Intel Core i5 dual-core" processor, not the most powerful CPU on the market. Since I do not have two available computers, as I explained in the chapter 3.4.9, I separate Client and Server creating two different Virtual Machines, with two different IP address: one is my Server and the other one is the Client who wants to ask to the Server to run YOLO's algorithm in its place.

I cannot test the results in different Internet speed conditions because everything was running locally on my laptop. By the way, I suppose that having a strong internet speed has its significant consequences on reaching the best possible results.
The problem of having a weak connection and being able to compute only few frames per second can be solved with the possibility of storing Client-side the video with the elaboration and watch it at a later stage without interruptions.

## 4.2) Calculating Frames per Second

As Sergio Canu mentioned in his tutorial, running an object detection algorithm on a framework like OpenCV, which can only exploit the CPU computational power and not the GPU's one, is resulting on the elaboration of few frames per seconds. This is just the time for elaboration, but in my system is also implemented a cryptographic part that, even if a CPU is able to run this section in milliseconds, has its weight on the overall project. In addition to that we must also consider the time spent for the Internet communication between Client and Server, which may vary a lot depending on the quality of the Internet Connection that the two parts have.

I tested my system on a clip of the duration of 1 minute and 46 seconds at 30 FPS of the film "Mission Impossible: Fallout" and I calculated the elaborated frames per second during all the clip interval with the three accepted sizes: FPS had a huge variation in the first frames and then a stabilization during most of the elaboration near to the values of the higher outcomes.

320x320: $FPS_{min} = 0.89$ 416x416: $FPS_{min} = 0.3$ 608x608: $FPS_{min} = 0.36$


*Figure 4.a – Speed Comparison with Different Image Resolutions (Lower Outcomes)*

320x320: $FPS_{max} = 1.13$ 416x416: $FPS_{max} = 0.76$ 608x608: $FPS_{max} = 0.42$

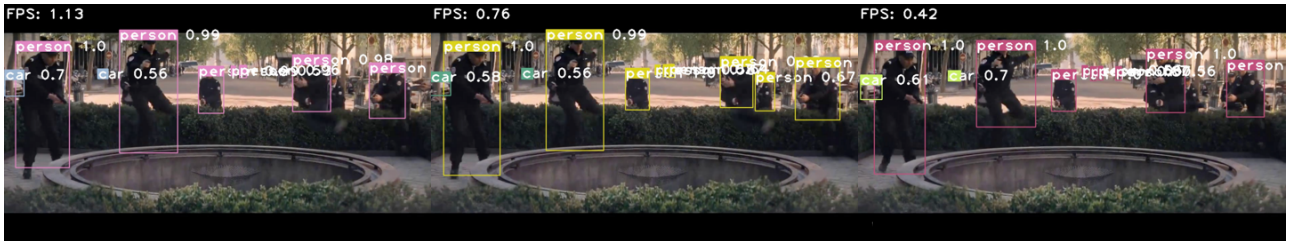



*Figure 4.b – Speed Comparison with Different Image Resolutions (Higher Outcomes)*

### 4.3) Error Handling

In order to make the system work, it is needed to insert at the bottom of the file "client.py" and "server.py" the IP address of the Server where YOLO is running, respectively in the functions "server_connection(server IP address)" and "server_elaboration(server IP address)". Then, thanks to the try-except block and the socket module, the system is able to automatically detect if there is any kind of problem Client-side or Server-side, returning the error number.

The first problem I run into was that, if the Server was not initialized before the Client, when I was about to run the file "client.py" the system raises the error "[Errno 61] Connection refused", as it is showed in the Figure 4.c. Actually, the connection is refused because, even if it is possible to connect to the Server, the Server has not opened the possibility to listen to requests and it refuses any kind of connection attempt.

```
Something went wrong, exiting...
 ERROR: [Errno 61] Connection refused

Process finished with exit code 0
```

*Figure 4.c – Screenshot of the Returned Error*

### 4.4) Performance analysis using different blob sizes

As it was mentioned in the chapter 2.2.1, YOLO accepts three different sizes: 320x320, 416x416 and 608x608. With a lower resolution it is possible to have a faster speed in recognizing objects, but the compromise is having a lower accuracy, while with the highest resolution the accuracy is higher and the speed lower.
I verified this assumption with my system, changing in the code the resolution and adding a way to calculate the time necessary to process the same input video file:

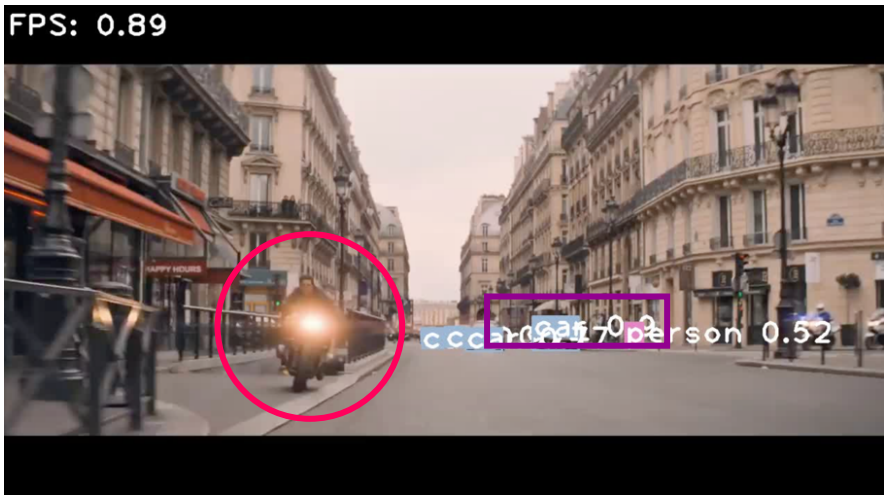
```
- blob = cv2.dnn.blobFromImage(frame, 0.00392, (320, 320), (0, 0, 0), True, crop=False)
- blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
- blob = cv2.dnn.blobFromImage(frame, 0.00392, (608, 608), (0, 0, 0), True, crop=False)
```

As mentioned earlier, the clip duration is 1 minute and 46 seconds at 30 FPS. With the lower resolution the time spent to process the video was: *47 minutes and 6 seconds*. With the middle resolution the time spent to process the video was: *1 hour, 9 minutes and 56 seconds*. With the higher resolution the time spent to process the video was: *2 hours, 6 minutes and 17 seconds*. The speed assumption is verified: with a lower size the processing is much faster rather than the higher size.

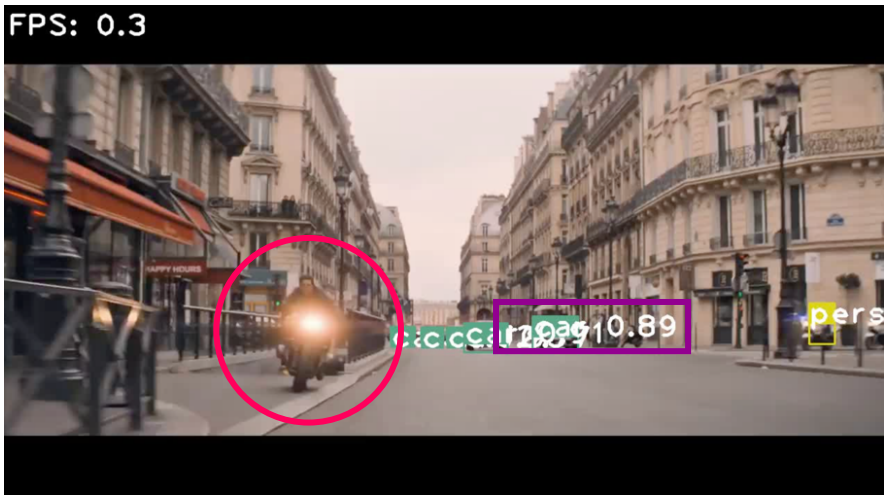
Concerning the accuracy of the analysis it is possible to compare the results, available on my YouTube Channel at this URLs:

- 320x320: https://youtu.be/ho7RV02wp0w
- 416x416: https://youtu.be/2kNS57uTeEU
- 608x608: https://youtu.be/nhG_NjKn2nI

320x320




416x416




608x608

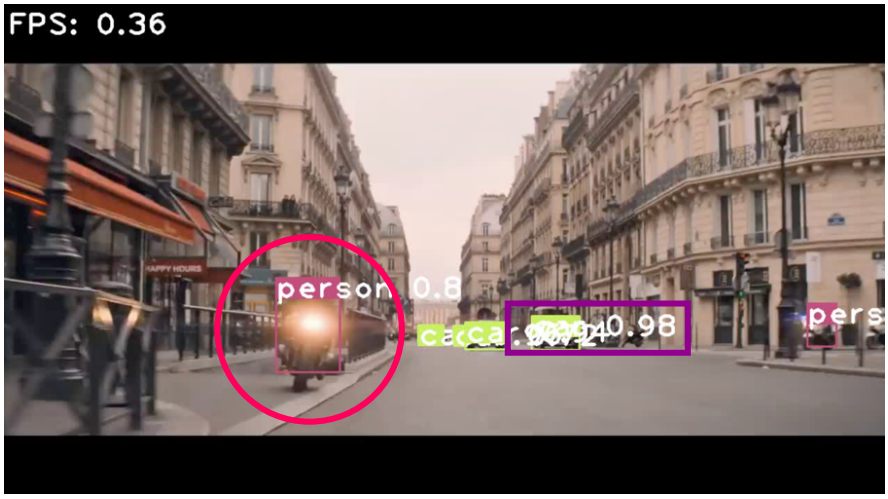



*Figure 4.d – Accuracy Comparison with Different Image Resolutions*

From the resulting videos, I extracted the same frame and I compared the results. As we can see, with the three red circles it is marked that, with the higher resolution, YOLO is able to detect the person on the motorbike, while on the other two sizes it does not. In addition to that, with the three purple rectangles it is marked that, with the higher resolution, the system is also able to predict with more accuracy the confidence score of the detected car: 0.9 for the size 320x320, 0.89 for 416x416 and 0.98 for 608x608.

## 4.5) Used Bandwidth

In order to measure the bandwidth trend during the communication between Client and Server, I installed Wireshark both Client-side and Server-side on my two Virtual Machines. It can be helpful mention that Wireshark is a network protocol analyzer (or packet sniffer), that lets the user capture and browse the traffic running on a computer network.

Before running the communication between Client and Server, I launched Wireshark on the Client. When the trade between the parts finished, I filtered the Wireshark capture results with the aim of finding the exact packets exchanged.

**Reminder:**
it is the same launching Wireshark filtering on the Server or on the Client, the important thing is using the correct filters then.

Knowing that:

- the port addressed for the application Client-side is the port number 40034 (it changes every time the program is run)
- the port addressed for the application Server-side is the port number 12345 (it stays the same every time the program is run)
- the IP address of the Client is 192.168.1.31
- the IP address of the Server is 192.168.1.32

I used the following instructions to filter the exact packets exchanged between the two parts:

- tcp. port == 12345 and tcp. port == 40034 and ip. src == 192.168.1.31 and ip. dst = 192.168.1.32 in order to extract the packets sent by the Client to the Server (red graph)

- tcp. port == 12345 and tcp. port == 40034 and ip. src == 192.168.1.32 and ip. dst = 192.168.1.31 in order to extract the packets sent by the Server to the Client (purple graph)

The filtering ends in a successful way. If we want to see graphically the results we must click on Statistics and then select I/O Graph, which produces the following results:

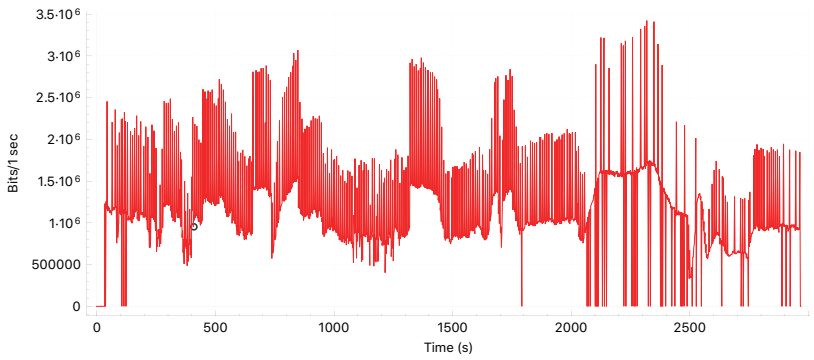



*Figure 4.e – Bit/sec sent by the Client*

In red we can see the graph of the first filtering, which corresponds to the Client sending to the Server the video file frame by frame. The results make sense because the bandwidth used is around 2-3 Mbps, which is consistent to the expected value.

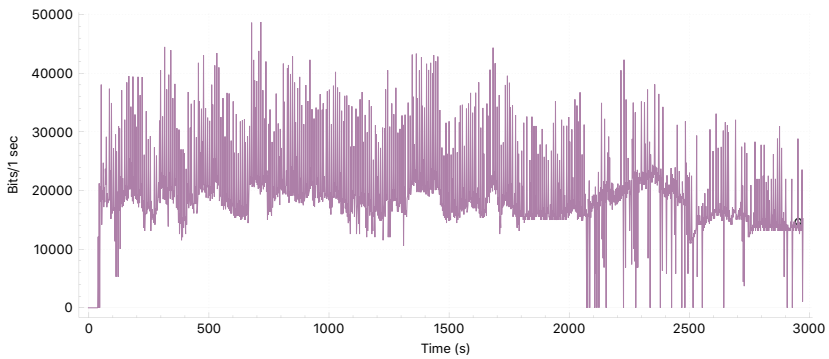



*Figure 4.f – Bit/sec sent by the Server*

In purple we can see the trend of the Server bandwidth used: in fact, since the Server does not send back to the Client the frame but only the results of the elaboration, it is obvious that the bandwidth used is lower.

# CHAPTER 5: CONCLUSION

The final conclusions are enclosed in this short chapter, whose purpose is to recap the work presented so far and to highlight why YOLO is one of most powerful real-time object detection algorithms known in the literature of Computer Vision.

## 5.1) Final considerations

YOLO, if compared to other object detection algorithms, is extremely fast thanks to its ability of predicting bounding boxes and class probability using one single convolutional neural network. Furthermore, the second major advantage is the algorithm's ability of seeing the entire image and contextualize additional information during the training, that may result extremely useful during the detection.

The added option developed throughout this thesis, namely the possibility of dividing the algorithm in two parts, respectively Client and Server, gives to the algorithm another notable advantage: in this way the computational power of the device who wants to perform object detection using YOLO it is not central anymore. In fact, since the algorithm is running on a Server that allegedly has a stronger computational power, the only concern the Client should worry about is to have a connection to the Internet.

This approach is called "Cloud Computing" and many technologies take advantage of that: just to give an example we can call out "NVIDIA GeForce NOW" or "Google Stadia". These two are online platforms where it is possible to connect to from any device via Internet and to play videogames. Actually, instead of having the computational power locally on every single Computer, PlayStation or Xbox, it is employed the computational power of the Server, who runs the game and sends via Internet every required information and command for playing the game with the lower ping possible.

During my work I already knew that the use of a framework like OpenCV would lead me to process few frames per second, even without the separation in Client and Server and the Cryptographic part. For this reason, I am not impressed or disappointed with the results obtained in terms of FPS processed. Instead, I am highly satisfied with the results concerning the elaboration: YOLO recognizes correctly numerous objects and the Cryptographic part in addition to the infrastructure Client-Server works in the proper way not slowing down the system significantly.

## 5.2) Possible applications

YOLO, thanks to its speed, is particularly useful for video surveillance purposes, helping humans supervising places like airports, shopping centers, schools and other public places, notifying the presence of harmful situations and objects and supporting the intervention of public authorities in dangerous circumstances.

2020 was a challenging year, where people were forced to get used to situations never faced before: quarantine, smart-working and social distancing were used in order to prevent the spread of covid-19. In this awkward period, YOLO and Computer Vision algorithms in general could help society by giving techniques to recognize if in public places everyone is using masks or if social distancing is respected, since masks are mandatory and social distancing recommended.

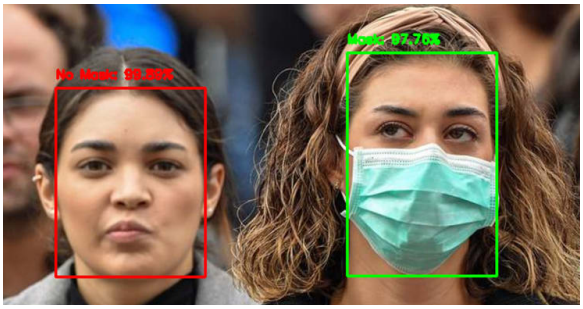


*Figure 5.a – Mask detection, from [21]*


*Figure 5.b – Social distancing detector, from [22]*

## 5.3) Future developments

Computer Vision in combination with Machine Learning involves several different problems: classification, localization, object detection and instance segmentation are just few of them and many more exist, like Motion or Medical Analysis. This area of study is extremely innovative and it is always open to investigate new cases, focus on new challenges and achieve new solutions.

Future developments for the work carried out in this thesis could be numerous, but here are presented the ones I consider the most substantial. If we want to improve the YOLO algorithm there are two modifications possible. The first improvement imaginable is the choice of training the model on a dataset with more than eighty objects and make the system able to recognize additional objects. The second refinement possible has a meaningful effect on performances and it involves the use of faster frameworks that work on GPU, instead of using OpenCV, that works only on CPU and it is inexorably slower. Moreover, there are also enhancements referring to data recovery and improvement of elaboration rapidity. It is possible to implement a way of requesting the retransmission of the specific part of data, for which HMAC verification failed, but also the implementation of a multi-thread system, which is able to perform various elaboration in parallel, making the Server able to process much more frames per second.

**[a] ACTIVATION FUNCTION:** an activation function is a mathematical equation that determine the output of a neural network. There are various kind of activation functions that exists, and some researchers are still working on finding better functions, which can help networks to converge faster or use less layers. [25]

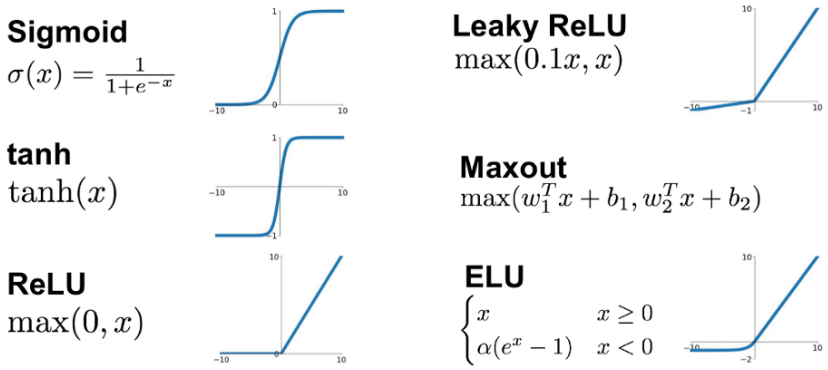



*Figure GLOSSARY.a –Different types of Activation Functions, from [23]*

**[b] ANCHOR BOX (or PRIOR BOX):** an anchor box represents the ideal location, shape and size of the object that the algorithm is predicting. [26]

**[c] BLOB:** a blob, that stands for "Binary Large Object", is a collection of binary data stored as a single entity and in Python it is stored as a 4D NumPy array object (images, channels, width, height). [27]

**[d] BSD SOCKET INTERFACE:** it is an application programming interface (API), with a set of standard functions that allow programmers to add Internet communication to their products. [28]

**[e] CLASSIFIER:** a classifier is the algorithm used for doing classification, that is the problem of identifying to which set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known. [29]

**[f] CODEC:** a codec is a device/computer program which encodes/decodes a digital stream or signal. A codec is a portmanteau (linguistic blend of words) between coder and decoder. [30]

**[g] DATA SERIALIZATION:** data serialization is the process of converting structured data to a format that allows the sharing or the storage of the data in a form that permits recovery of its original structure. [31]

**[h] DEEP LEARNING FRAMEWORK:** a deep learning framework is an interface, library or a tool which allows us to build deep learning models more easily and quickly, without getting into the details of underlying algorithms. They provide a clear and concise way for defining models using a collection of pre-built and optimized components. [32]

**[i] DIGEST:** a digest is the output of the Hash function. [33]

**[j] EULER'S TOTIENT FUNCTION $\emptyset(N)$:** Euler's totient function counts the positive integers up to a given integer N that are relatively prime to N. [33]

**[k] FEATURE:** in computer vision and in image analysis a feature is a piece of information which is relevant for solving the computational task related to a certain application. [34]

**[l] FEATURE DETECTOR:** in computer vision and in image analysis a feature detection includes methods for computing abstractions of image information and making local decisions at every image point whether there is an image feature of a given type at that point or not. [34]

**[m] FEATURE MAP**: a feature map is the output of the elaboration done by the various layers of filters and its depth corresponds to the number of filters used. [34]

**[n] FLATTENING:** flattening is converting the data into a 1-dimensional array for inputting it to the next layer, flattening is operated in order to insert the data into an artificial neural network. [35]

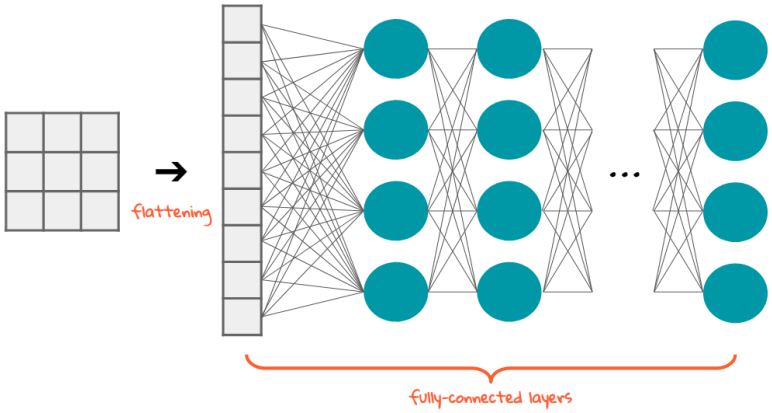



*Figure GLOSSARY.b – Flattening visualization, from [24]*

**[o] GROUND TRUTH:** the ground truth is what my model is required to predict and it is usually different from what is actually predicted by the predicted box, in other words the ground-truth box is the expected box to be recognized, which is the one resulting from the training.

**[p] HASH FUNCTION:** a Hash function is a one-way function practically infeasible to invert that produces a fixed length output. It works in a way such that a small change in the input produces an important change in the output. [33]

**[q] IMAGE FILTER:** an image filter is a technique through which size, color, shading or other characteristics of an image are altered

**[r] LINEAR REGRESSION:** in statistics, linear regression is a linear approach to model the relationship between a scalar response or dependent variable and one or more explanatory variables or independent variables. [36]

**[s] LOGISTIC REGRESSION:** it is a statistical model used to determine if an independent variable has an effect on a binary dependent variable: this means that there are only two potential outcomes given an input. [37]

**[t] LOSS FUNCTION:** in mathematical optimization and decision theory, a loss function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. [38]

YOLO's loss function is the sum of three distinct functions: [39]

- **Classification loss**: if an object is detected, the classification loss at each cell is the squared error of the class conditional probabilities for each class
- **Localization loss**: it measures the errors in the predicted boundary box locations and sizes. We only count the box responsible for detecting the object.
- **Confidence loss**: it measures the objectness of the box.

**[u] OVERFITTING:** in statistics and in machine learning overfitting is the production of an analysis which corresponds too closely or exactly to a particular set of data and may therefore fail to fit additional data or predict future observations reliably, in other words overfitting is a modeling error that occurs when a function is too closely fit to a limited set of data points. [40]

**[v] PIPELINE:** in machine learning model there are many moving parts that have to be tied together in order to execute the model correctly and produce results successfully: the process of tying together different pieces of the machine learning process is known as a pipeline. [41]

**[w] POOLING:** pooling layers provide a method to down sampling feature maps by summarizing the presence of features in areas of the feature map. The two most common pooling methods are average pooling, which summarize the average presence of a feature and max pooling that summarize the most activated presence of a feature. [42]

**[x] SELECTIVE SEARCH:** it is a region proposal algorithm used in object detection. It is designed to be fast with a very high recall. It is based on computing hierarchical grouping of similar regions based on color, texture, size and shape compatibility. [43]

**[y] SOFTMAX FUNCTION:** the Softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the Softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities. If one of the inputs is small or negative, the Softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1. [44]

**[z] SVM (SUPPORT VECTOR MACHINE):** SVM is an algorithm of machine learning tool, that can be used both for classification and regression problems. It is popular in applications like natural language processing, speech-recognition, image-recognition and in computer vision problems. SVM algorithm is more effective in binary classification problems. [45]

## BIBLIOGRAPHY and WEB REFERENCES


[1] https://arxiv.org/abs/1506.02640v4, Redmon Joseph, Divvala Santosh, Girshick Ross, Farhadi Ali, "You Only Look Once: Unified, Real-Time Object Detection", 05/08/2020

[2] https://arxiv.org/abs/1612.08242, Redmon Joseph, Farhadi Ali, "YOLO9000: Better, Faster, Stronger", 05/08/2020

[3] https://arxiv.org/abs/1804.02767, Redmon Joseph, Farhadi Ali, "YOLOv3: An Incremental Improvement", 05/08/2020

[4] https://pysource.com/2019/06/27/yolo-object-detection-using-opencv-with-python/, Canu Sergio, "YOLO object detection using OpenCV with Python", 05/08/2020

[5] https://pysource.com/2019/07/08/yolo-real-time-detection-on-cpu/, Canu Sergio, "YOLO Real time detection on CPU", 05/08/2020

[6] https://docs.opencv.org/4.4.0/, "OpenCV modules", 05/08/2020

[7] https://numpy.org/doc/stable/, "NumPy manual", 05/08/2020

[8] https://pypi.org/project/pycrypto/, "Python Cryptographic Toolkit", 05/08/2020

[9] https://docs.python.org/3/library/socket.html, "Socket, Low-level networking interface", 05/08/2020

[10] https://docs.python.org/3/library/pickle.html, "Pickle, Python object serialization", 05/08/2020

[11] https://docs.python.org/3/library/struct.html, "Struct, Interpret bytes as packed binary data", 05/08/2020

[12] Stallings William, "Cryptography and network security", Prentice hall, 5th edition

[13] https://towardsdatascience.com/computer-vision-a-journey-from-cnn-to-mask-r-cnn-and-yolo-1d141eba6e04, Khandelwal Renu, "Computer Vision, A journey from CNN to Mask R-CNN and YOLO - Part 1", 05/08/2020

[14] https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/, Shukla Sagar, "Regression and Classification | Supervised Machine Learning", 05/08/2020

Images:

[15] https://towardsdatascience.com/computer-vision-a-journey-from-cnn-to-mask-r-cnn-and-yolo-1d141eba6e04, Khandelwal Renu, "Computer Vision, A journey from CNN to Mask R-CNN and YOLO - Part 1", 05/08/2020

[16] https://appsilon.com/object-detection-yolo-algorithm/, Maj Michał and contributions from the Appsilon Data Science team, "YOLO Algorithm and YOLO Object Detection: An Introduction", 05/08/2020

[17] https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173, Hui Jonathan, "mAP (mean Average Precision) for Object detection", 05/08/2020

[18] https://arxiv.org/abs/1506.02640v4, Redmon Joseph, Divvala Santosh, Girshick Ross, Farhadi Ali, "You Only Look Once: Unified, Real-Time Object Detection", 05/08/2020

[19] https://www.researchgate.net/publication/320723863_Village_Building_Identification_Based_on_Ensemble_Convolutional_Neural_Networks, Guo Zhiling, Chen Qi, Wu Guangming, Xu Yongwei, Shibasaki Ryosuke, Shao Xiaowei, "Village Building Identification Based on Ensemble Convolutional Neural Networks", 05/08/2020

[20] Stallings William, "Cryptography and network security", Prentice hall, 5th edition

[21] https://github.com/chandrikadeb7/Face-Mask-Detection, Deb Chandrika, "Face Mask Detection", 05/08/2020

[22] https://www.pyimagesearch.com/2020/06/01/opencv-social-distancing-detector/, Rosebrock Adrian, "OpenCV Social Distancing Detector", 05/08/2020

[23] https://mc.ai/complete-guide-of-activation-functions/, Jain Pawan, "A practical guide related to benefits, problems, and comparison of activation functions like Sigmoid, tanh, ReLU, Leaky ReLU and Maxout", 05/08/2020

[24] https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480, Jeong Jiwon, "The Most Intuitive and Easiest Guide for Convolutional Neural Network", 05/08/2020

Glossary:


[25] https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092, Jadon Shruti, "Introduction to Different Activation Functions for Deep Learning", 05/08/2020

[26] https://medium.com/@andersasac/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9, Christiansen Anders, "Anchor Boxes, The key to quality object detection", 05/08/2020

[27] https://opencv-tutorial.readthedocs.io/en/latest/yolo/yolo.html, Holzer Raphael, "Yolo – object detection", 05/08/2020

[28] https://www.keil.com/pack/doc/mw6/Network/html/using_network_sockets_bsd.html,"BSD Socket", 05/08/2020

[29] Alpaydin, Ethem, 2010, "Introduction to Machine Learning", MIT Press.

[30] https://web.archive.org/web/20150405161202/http://desktopvideo.about.com/od/glossary/g/codec.htm, Siegchrist Gretchen, "Codec", 05/08/2020

[31] https://docs.python-guide.org/scenarios/serialization/, Reitz Kenneth, "Data Serialization", 05/08/2020

[32] https://www.analyticsvidhya.com/blog/2019/03/deep-learning-frameworks-comparison/, Sharma Pulkit, "5 Amazing Deep Learning Frameworks Every Data Scientist Must Know!", 05/08/2020

[33] Stallings William, "Cryptography and network security", Prentice hall, 5th edition

[34] Umbaugh Scott E., 27 January 2005, "Computer Imaging: Digital Image Analysis and Processing", CRC Press

[35] https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480, Jeong Jiwon, "The Most Intuitive and Easiest Guide for Convolutional Neural Network", 05/08/2020

[36] Freedman David Amiel, 2009, "Statistical Models: Theory and Practice", Cambridge University Press

[37] https://machinelearning-blog.com/2018/04/23/logistic-regression-101/, Donges Niklas, "The Logistic Regression Algorithm", 05/08/2020

[38] Wald Abraham, 1950, "Statistical Decision Functions", Wiley

[39] https://arxiv.org/abs/1506.02640v4, Redmon Joseph, Divvala Santosh, Girshick Ross, Farhadi Ali, "You Only Look Once: Unified, Real-Time Object Detection", 05/08/2020

[40] https://www.investopedia.com/terms/o/overfitting.asp, Kenton Will, "Overfitting", 05/08/2020

[41] https://dzone.com/articles/how-to-build-a-simple-machine-learning-pipeline, Das Sibanjan, Cakmak Umit Mert, "How to Build a Simple Machine Learning Pipeline", 05/08/2020

[42] https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/, Brownlee Jason, "A Gentle Introduction to Pooling Layers for Convolutional Neural Networks", 05/08/2020

[43] https://www.learnopencv.com/selective-search-for-object-detection-cpp-python/, Chandel Vaibhaw Singh, "Selective Search for Object Detection (C++ / Python)", 05/08/2020

[44] https://deepai.org/machine-learning-glossary-and-terms/softmax-layer, Wood Thomas, "Softmax Function", 05/08/2020

[45] https://lorenzogovoni.com/support-vector-machine/, Govoni Lorenzo, "Algoritmo Support Vector Machine", 05/08/2020