POLITECNICO DI TORINO

MASTER's Degree in Computer Engineering

MASTER's Degree Thesis

Increasing the abilities of mobile robots with Computer Vision algorithms

---

Supervisors
Prof. FABRIZIO LAMBERTI
Dr. OSCAR PISTAMIGLIO

Candidate
SAMUELE GIANNETTO

December 2023

Summary

General Overview

In the field of robotics, there is a growing trend towards the automation of mechanisms and tasks, reflecting a dynamic shift towards increased efficiency and productivity, with artificial intelligence playing a fundamental role.

This thesis explores various use cases and research conducted on Boston Dynamics©' four-legged robot Spot, focusing on the integration of computer vision algorithms, enabling Spot to achieve remarkable level of self-awareness of the environment and improving its capacity to collaborate with humans.

Spot offers a wide range of use cases where it can be employed. Patrolling is implemented through the registration of missions, which consists of recorded path containing saved locations where custom actions can be added. Some examples of custom actions are: reading of the gauge when Spot is in front of a manometer or detection of the leakage in front of a valve. The robot can also be guided using a tablet, substituting humans in dangerous scenarios. However the main effort remains directed towards the automation of processes.

This thesis work is born through the collaboration between Politecnico di Torino and Sprint Reply, all these research activities took place in Reply laboratory named "Area42", in Lingotto, Turin.

Research proposed

Two main activities are performed in this thesis work:

- Implementation of a REST API which takes Spot streaming and performs inference over three neural networks giving the robot the possibility to detect exit signals, fire extinguishers, objects from MS-COCO dataset and to anonymize faces.
- Creation of a mission where Spot autonomously navigates to a valve and closes it in case of leakage.

Computer vision algorithms for Spot

This thesis presents different neural networks aimed at enhancing Spot capabilities in the industrial context, where it is thought to be addressed.

In this context, it is necessary to find an algorithm that anonymizes faces to protect the workers' privacy. For that purpose, a single-shot multi-level face localization Python library is chosen, called RetinaFace, which performs simultaneously face detection, 2D face alignment and 3D face reconstruction in a single step inference. The bounding boxes of the detected faces are then pixeled to show on screen the anonymized video.
Another computer vision algorithm is employed to detect safety exits and fire extinguishers, ensuring workplace safety in industrial settings. The purpose is to provide safety by making Spot autonomously identifies these security objects, facilitating efficient evacuation procedures in emergency situations and reporting to operators the fire extinguishers correct position during daily patrols.
A third neural network is implemented for the detection of the 80 classes from the MS-COCO dataset, further improving Spot's surroundings awareness. To this project is also added an OCR, applied on bounding boxes labelled as vehicles, used to identify and extract the license plates.
These last two algorithms are implemented using the YOLOv8 neural network provided by Ultralytics Python library. This implementation is adopted because it is a single-stage approach algorithm, so it has a higher inference rate, and it gives the possibility to choose between different network structures with high variation on the number of parameters, playing on the trade-off over performance and faster prediction time.

These neural networks are integrated into a project that takes as input the streaming from the robot's cameras. Depending on the selected network, the system displays the inference results to the operator. Such versatility can be useful in many situations, as the robot can replace humans in repetitive inspections, which could lead to errors because of the lack of attention.
This project is structured as a REST API, and the neural networks are organized as resources reachable via URL. The idea is to capture a frame from Spot and to perform predictions on a remote server, also a PC, endowed with GPU or higher performance CPU, instead of the one mounted on the robot. Spot sends to the resource of interest each frame of the streaming decoded in a HTTP request and receives the predicted image in a HTTP response.

Robot cooperation

Another relevant use case examined is the cooperation between robots.

Spot Enterprise, equipped with PTZ 30x zoom cameras, is sent on an autonomous mission, which is a pre-recorded path with saved reference locations (waypoints) where Spot performs a proper action. The goal is to identify potential leakages within an industrial plant, an example could be leakage detection of a valve in a waypoint located near a tap. Collected data, as photos and videos, are transmitted to a cloud platform, where computer vision algorithms provide predictions, reporting alerts to an operator in case of faults. Spot Arm, equipped with a robotic arm, receives a trigger from the cloud, assigning a specific mission on the base of the detected fault. In case of faulty valve, its task is to close it.

In this thesis is deeply covered the structural logic and the creation process of the second mission, starting from the establishment of a pipeline, which generates a synthetic dataset using photos of valves, passing through the training of the neural network and arriving to the setup of the mission, allowing Spot to autonomously navigate to the valve, identify the position, grasp and close it.

An initial training is done for the detection on a YOLOv8 model using the synthetic dataset of valves generated with the pipeline. To improve the inference performance a fine-tuning process is done training the obtained weights over a set of real photos manually labelled.
Boston Dynamics© provides a software development kit (SDK) to customize Spot's behaviours. This code offers a large number of basic functions, implementations for Spot movement and configuration. Combining these functions with the valve detection algorithm it is possible to implement the action of closure.
The navigation path is registered with the Spot tablet and a waypoint in front of the tap is saved, here adding the custom action previously explained it is possible to create the complete mission.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
: Artificial Intelligence

API
: Application Programming Interface

EAP
: Enhanced Autonomy Payload

NN
: Neural Network

PTZ
: Pan Tilt Zoom

SDK
: Software Development Kit

Chapter 1

Introduction

Spot is a four-legged robot developed by Boston Dynamics©. This well-known robot is easily controllable using its dedicated tablet but can also be programmed for autonomous missions. It dynamically balances thanks to proper hardware, allowing it to walk even in rough terrain with agility, and is small enough to navigate in industrial plants, being endowed of obstacle avoidance it is suitable for cohabitation and cooperation with humans [1].

Working inside Reply laboratory named "Area42", the goal is to increase Spot autonomous capabilities, using computer vision algorithms and Spot software development kit to implement the robot movement and to create missions.

Figure 1.1: Boston Dynamics© robot Spot and 'Area42' logo

1.1 Robot composition

Spot is a dynamic sensing platform, the base structure is the one shown in Figure 1.1. Two ports located on the back of the robot provide power, communication, time-synchronization and safety system integration, giving the possibility to mount sensors, Boston Dynamics© refers to with the name "payloads". Different payloads can be added to the back of Spot, in order to configure it on the base of the needed task. Temperature and humidity sensors, different types of cameras (thermal, x30 scope, 360), LIDARs, x-ray sensors, proximity and distance sensors are only some examples of the possible payloads that can be used, the possibilities of combination and use cases are nearly infinite. This huge capability of variation is the reason why Spot can be used in different scenarios such as industries, research labs, private offices [1].

In particular in "Area42" there are two Spot robots (Figure 1.2), with these payloads:

- Spot Enterprise: equipped with Spot EAP (enhanced autonomy payload), temperature and pressure sensors and Spot CAM+ (cameras);
- Spot Arm: equipped with Spot EAP2 and Spot robotic arm.

Figure 1.2: Robots in "Area42"

1.1.1 Spot CAM+

The Spot CAM+ (Figure 1.3) payload significantly improves the robot's image capturing capability and gives the operator much better situational awareness as

Spot moves through space [2]. It includes a spherical camera that creates 360° panoramas (Spot CAM), a PTZ camera with 30x optical zoom and four pairs of LED lights to allow the robot operate even in dark environments. It is also provided with two speakers for broadcasting audio and two microphones for receiving audio.

Figure 1.3: Spot CAM+

1.1.2 Spot EAP and EAP2

Spot EAP (Enhanced Autonomy Payload) and Spot EAP2 are both provided with Spot CORE and the same LIDAR distance sensor (Figure 1.4).
The LIDAR Velodyne VLP-16 sensor significantly increases Spot's depth perception to approximately 120 meters, which without sush a sensor is limited to 2-4 meters. The EAP includes Spot CORE, which is a computer with Linux operating system, 16 GB RAM, i5 Intel, 8th Gen CPU whereas the EAP2 includes Spot CORE I/O with a 6-core CPU NVIDIA Carmel ARM 64-bit, a GPU 384-core NVIDIA Volta GPU and 16GB RAM [3].

Figure 1.4: Spot EAP/EAP2: Spot CORE + LIDAR

1.1.3 Spot Robotic Arm

Spot can be equipped with a robotic arm as official Boston Dynamics©' payload (Figure 1.5).
The arm offers various features:

- Pick and place of a variety of objects, with a lifting capacity of up to 11 kg and a dragging capacity of up to 25 kg.
- Manipulation of objects by grasping and performing constrained actions, such as rotating valves, pulling levers and opening drawers based on the object's characteristics.

The arm has 6 degrees of freedom plus the one of the gripper.
The control of the arm is handled both on joint-space control, specifying each joint angle and velocity or end-effector control, specifying position, velocity and force trajectories in cartesian space.
The gripper has a depth of 90 mm and can open up to a maximum of 175 mm. Inside the gripper, there is a camera that provides Spot with a closer environmental perception when interacting with objects [4].

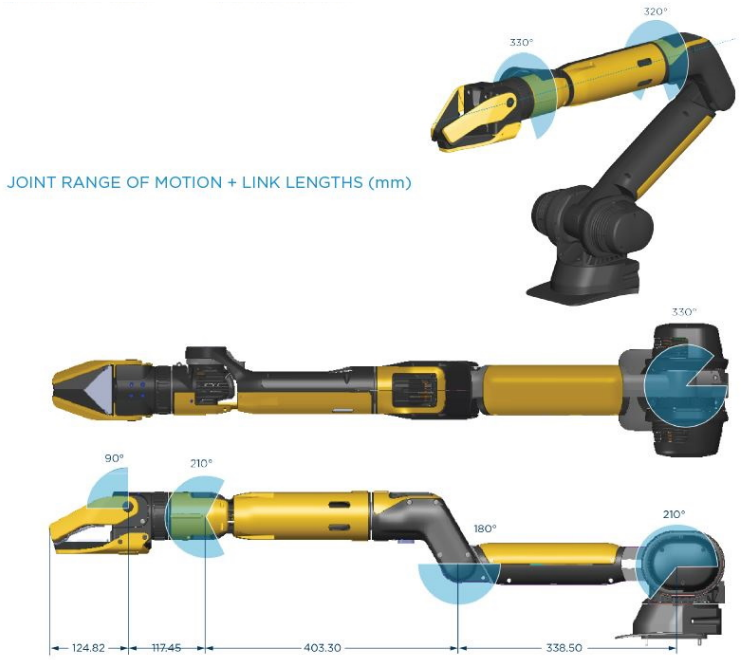

Figure 1.5: Robotic arms mounted on Spot

1.2 Spot connectivity

Spot has built-in WiFi. There exists two different types of configurations:

- Access Point: users connect to Spot via WiFi, which operates at 2.4GHz and 5GHz;
- Client Mode: Spot is connected to a shared WiFi network, the same network to which users are connected.

The second scenario is particularly beneficial in large industries with extensive WiFi coverage, ensuring that Spot maintains its connection with remote users.
It is also possible to connect Spot via Ethernet. Wired connection is faster than WiFi and could be used for uploading or downloading large files such as robot updates, payload software or large logs [5].

1.3 Spot controller

Boston Dynamics© allows to control the robot using a tablet (Figure 1.6). By adding the robot WiFi to the selected network and entering name and password it is possible to operate and drive the robot via a user-friendly interface. It is also possible to register and repeat autonomous missions, adding custom actions on saved waypoints, particularly useful in industrial contexts [6].

Figure 1.6: Spot tablet view

1.4 Spot Dock and Spot Charger

Spot battery has a capacity of 605 Wh, a charging power of 400W and weighs 5.2 kg. It provides Spot an autonomy which varies from 90 minutes to 180 minutes, depending on whether it is under heavy use or in standby mode, and it can be fully charged in approximately 120 minutes using two different methods:

- Spot Dock (known as Dock Station): the robot has physical modifications and behaviors that support docking and undocking from the station. While Spot is 'sitted' on the Dock Station its attached battery is charged (Figure 1.7).
- Spot Charger: using the "shore power cable" is possible to connect the charger to the back of Spot and to charge the battery inserted in the robot. The standalone battery can also be charged directly using the 'battery charging cable' as in Figure 1.8 [7].

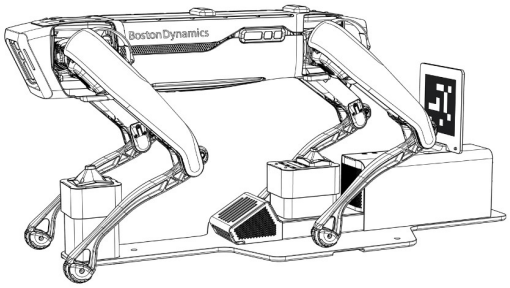

Figure 1.7: Spot Dock

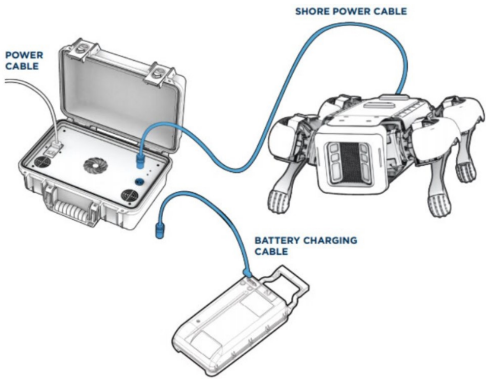

Figure 1.8: Spot Charger

1.5 Technical information about Spot

The table 6.2 shows the main numerical specifications of Spot [8].

CATEGORY	SPECIFICATION	VALUE
Dimensions	Length	1100 mm (43.3 in)
	Width	500 mm (19.7 in)
	Height (standing)	840 mm (33.1 in)
	Height (sitting)	191 mm (7.5 in)
	Net weight	32.5 kg (71.7 lbs)
	Degrees of freedom	12
	Max speed	1.6 m/s
Environment	Ingress protecion	IP54
	Operating temperature	-20C to 45C
	Slopes	+/- 30 degrees
	Stairways	Stairs that comply with US building code, typically with 7 in. rise for 10-11 in. run.
	Max step size	300 mm (11.8 in)
Power	Battery Capacity	605 Wh
	Max battery voltage	58.8V
	Typical runtime	90 minutes
	Standby time	180 minutes
	Charger power	400W
	Max charge current	7A
	Time to charge	Approximately 2 hours
	Battery weight	4.2 kg (9.3 lbs)
Payload	Max weight	14 kg (30.9 lbs)
	Max power per port	150W
	Payload ports	2
	T-slot rail bolt size	M5 x 1.0
	Camera type	Projected stereo
	Field of view	360 degrees
	Operating range	4 m (13 ft)
Connectivity	802.11	Wifi
	Ethernet	1000Base-T

Table 1.1: Spot characteristics.

Spot is thought to work near humans, hence it is provided with many functionalities to help this cooperation, ensuring safe work and proficient co-working.
In particular:

- Spot is particularly agile and stable: the implemented dynamics allow the robot to maintain balance, resisting collisions, pushes and to be agile even in ruined or muddy terrains.
- Spot implements automatically obstacle avoidance, in particular for object of more then 30 cm of height. Its perception system consists of five stereo cameras with a 360 degree field of view and a detection range up to 4m, which automatically avoids running into obstacles.
- Spot considers obstacles lower then 30 cm as stairs and automatically navigates over them; however, it is good practice not to operate on stairs that are less than 60 cm (24 in) wide, pitched more than 45 degrees or with a step height greater than 22 cm, in order to preserve robot stability.

1.6 Use cases

Spot has a vital role in industries environment. Boston Dynamics© says "set up for autonomous inspections to power the predictive maintenance program and create digital twins of your facilities". Clarifying this sentence, Spot can be seen as a mobile platform of sensors, able to learn a mission, following a chosen path and performing actions in each waypoint of the saved map, like the identification of facilities or the reading of measures. One or, better, a fleet of Spots repeat periodically the missions along the industrial plant, detecting anomalies and reporting to a human user or revolving them through the cooperation with others Spot. All these action are performed autonomously by Spot as in Figure 1.9.

A particular achievement this thesis deeply focuses, which will be explained in Chapter 6, is a practical example to understand this use case: Spot Enterprise executes patrolling over "Area42" laboratory, it takes respectively a photo to measure pressure from a manometer and a video to detect dripping of a tap. In case of water losses Spot Enterprise sends an alarm to the human user and a trigger to Spot Arm that, knowing the position of the tap, goes to its waypoint and turning the knob, closes the tap.

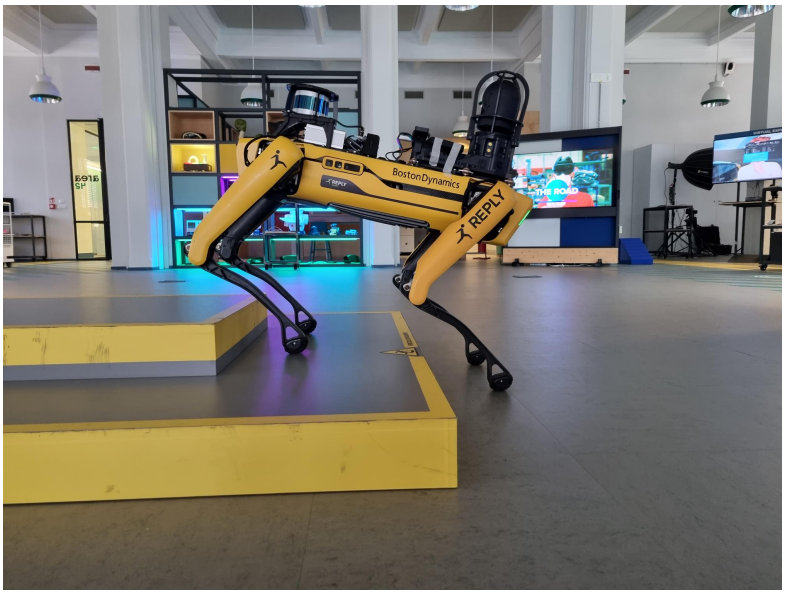

Figure 1.9: Spot walking on stairs during autonomous missions

Another particular scenario in which Spot can be used is in dangerous environments where it can substitute human as in unsafe buildings before intervention by police and firefighters or in archaeological excavations.

1.7 Research proposed

In this thesis, after a general overview over most renowned computer vision algorithms that could be helpful for the activities, two research directions are specifically addressed:

- In Chapter 4 it is shown the 'Spot POV' (Point of View) project, a REST API which allows Spot to send each frame from its streaming to an endpoint where different neural networks (whose details are given in Chapter 3) makes predictions, providing the robot of computer vision.
- In Chapter 6 it is described another project that is aimed to make Spot autonomously close a valve. The neural network for the valve detection is trained over a synthetic dataset created through a pipeline explained in Chapter 5.

Chapter 2

Computer vision overview

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to extract meaningful information from digital images, videos and other visual inputs and to take actions, generate reports and give meaning to the data basing on such information. If AI allows computers to think, computer vision allows them to see.

Arthur Samuel, one of the pioneers and founders of artificial intelligence, used the definition: "Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed." [9].

Machine learning utilizes algorithm and models that enable a computer to independently learn from the context of visual data. If this model is 'fed' with enough data on input, the computer 'observes' the data and learns by itself to distinguish one image from another. The algorithms allow the machine to learn on its own, without someone explicitly programming it to recognize an image.

Neural networks are a subset of machine learning algorithms, their structure emulates the human brain. A neural network consists of interconnected processing units called '*neurons*', working together to solve complex problems. These networks are organized into layers of interconnected neurons that process and transform data, Figure 2.1 is an example. Deep learning involves deep neural networks with multiple layers inside the network (hidden layers), hence the term 'deep'. These neural networks have the ability to automatically learn features and patterns from raw data, particularly powerful in domains like image and speech recognition and natural language processing [10].
Learning patterns from data, the network can adjust the weights of connections between neurons to adapt its capability over a specific task. Depending on the particular task there are many techniques and algorithms used to tackle them, so the model is built ad hoc for the purpose [11].

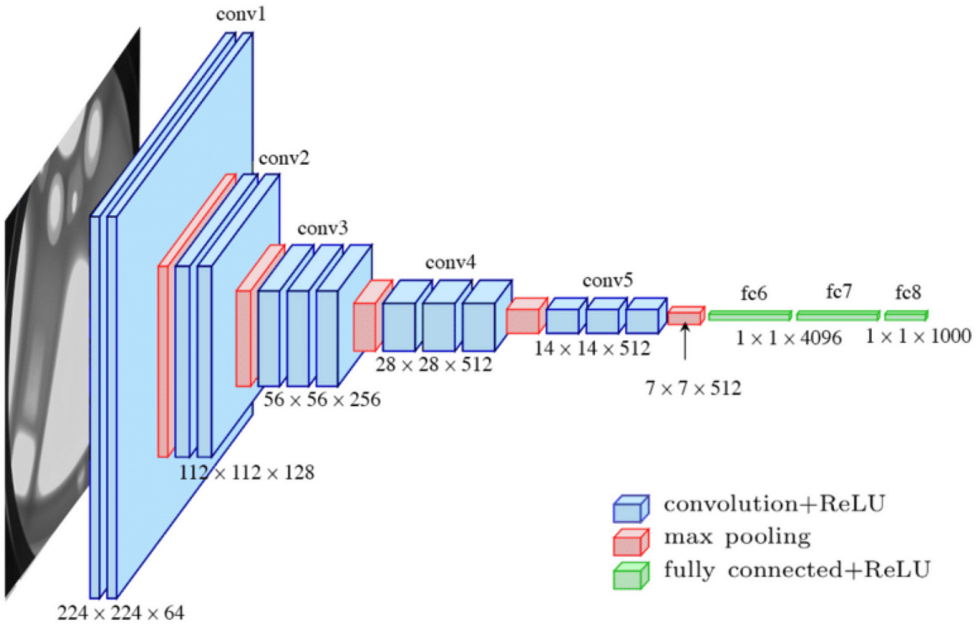

Figure 2.1: VGG-16, a famous neural network

Machine learning can be supervised or unsupervised.
Supervised learning: the model is trained using a labelled dataset. The labels indicate the desired response for each training example. The model compares predicted data with real label, becoming able to understand which type of object is shown in a photo.
Unsupervised learning: the model is trained using a dataset that lacks of labels. The model finds correlation and similarity between structures. Typical use cases are clustering (grouping similar objects) and anomaly detection (identify outliers: patterns that deviate significantly from the norm).
Also a third case of machine learning exists, it is called reinforcement learning. Differently from the previous two cases it does not rely on a set of static data, instead it works on a dynamic environment, learning from collected experience, taking better reward as its sequential decisions get better.

2.1 Classification

A typical computer vision task is the '$classification$'. It is a supervised learning problem where the goal is to assign predefined categories or '*classes*' to input data, like images or videos.

Common examples of classification tasks include email spam detection, image recognition (identifying objects in images), sentiment analysis (determining if a text expresses positive or negative sentiment) and medical diagnosis (categorizing diseases based on patient symptoms).

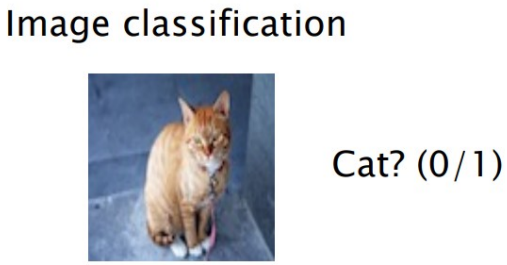

Figure 2.2: Classification example: does the picture contains a cat or not?

The choice of the algorithm depends on the nature of the problem. Also the model evaluation metrics depends on the type of problem. Usually, it is convenient to reach desired trade-offs between different types of values (like precision-recall). Accuracy, precision, recall and F1-score are common metrics used to evaluate the performance of classification models.

The main and simplest classification metric is the accuracy. It is the ratio between correct and total predictions.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions\ made}$$

Precision is an indicator of how many predictions are correct; this metric should be high when the model needs to be confident in assigning a specific label [10].
Recall is an indicator of how many predictions of a class the network has lost; this metric should be high when the model should not overlook potentially dangerous cases, so it assigns a label even if it is not entirely certain about it.
Both formulas are shown in Figure 2.3. Generally when one is higher the other one is lower.
In order to compare two different models a good metric is the 'F1 score' or 'harmonic mean', which takes care of both precision and recall.

$$F1 = \frac{Precision \cdot Recall}{Precision + Recall}$$

Its value decreases quickly if one of the two metrics decreases, so it is better then a simple mean. Greater is this value, better are the model performance.

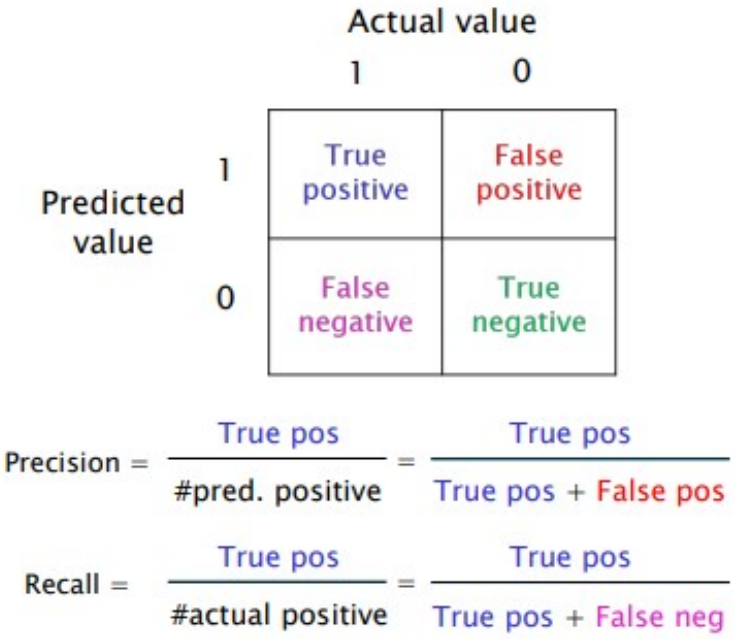

Figure 2.3: Precision and Recall metrics.

2.2 Object detection

Object detection is a computer vision task that involves identifying and locating multiple objects within an image or a video frame. Unlike image classification, where the goal is to assign a single label to an entire image, object detection aims to localize and classify multiple objects of interest within the image [12].
In object detection, the primary objectives are:

- Localization: determining the precise location of objects in an image by drawing bounding boxes around them. A bounding box is a rectangle that encompasses the object's spatial extent.
- Classification: assigning a class label to each detected object to identify what type of object it is (examples: car, human, dog).

This task has to do with the need to handle varying object sizes and orientations, multiple presences of objects in a single image and occlusions. Over the years, various techniques have been developed to solve this issues.
Traditional methods use handcrafted features and object detectors, such as Haar cascades or Histogram of Oriented Gradients (HOG), combined with machine learning algorithms to classify and localize objects [13].

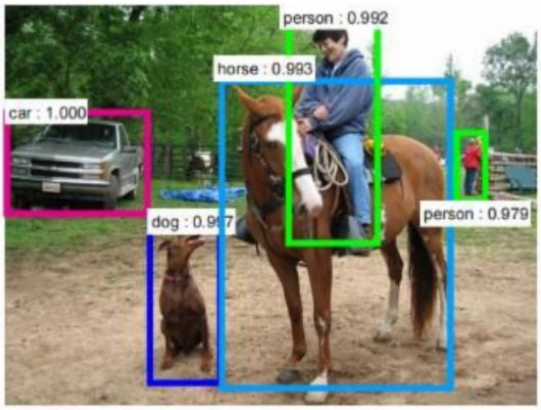

Figure 2.4: Example of object detection

Deep Learning has revolutionized object detection through methods like Faster R-CNN, YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector). These approaches use deep neural networks to simultaneously predict bounding box coordinates and class probability for multiple objects in a single step [10].

Object detection finds applications in various fields, including self-driving cars, surveillance, medical purposes, commercial analyses and more. The ability to accurately detect and locate objects within images and videos has opened up opportunities for automation, safety and decision-making in diverse domains. Today it is used in numerous fields, such as autonomous driving, crowd control, speech to test, in this thesis is especially analysed industrial robotics.

2.2.1 CNN approach

One of the first approaches to object detection was the utilization of a convolutional neural network (also called ConvNet or CNN). An example of CNN is the one in Figure 2.5. The image to be classified is the input of some convolutional layers that 'shrink' the first two dimension of the image and enlarge the third, which is the number of channels, equal to the number of filters of the convolution level. Using various types of filters the network extracts features from the image, representing different patterns in the image. Usually after a convolution is added a 'pooling layer' which reduces the spatial dimensions of the features while retaining important information. It helps in reducing computation and controlling overfitting (later in this chapter it will explained). Different types of pooling can be implemented, from a region it extracts the maximum 'max pooling' or the average value 'avg pooling' [10].

At the end of the network are added 'fully connected layers' which flattens the pooled features. As the name suggests all the output are provided as input to the neurons of the following layer. These layers learn to combine high-level features for classification; this is the reason why they are used as last layers. In fact, they combine the extracted patterns in order to make the final decision. Then an output layer produces the final classification results, typically using a softmax activation to output class probabilities [10].

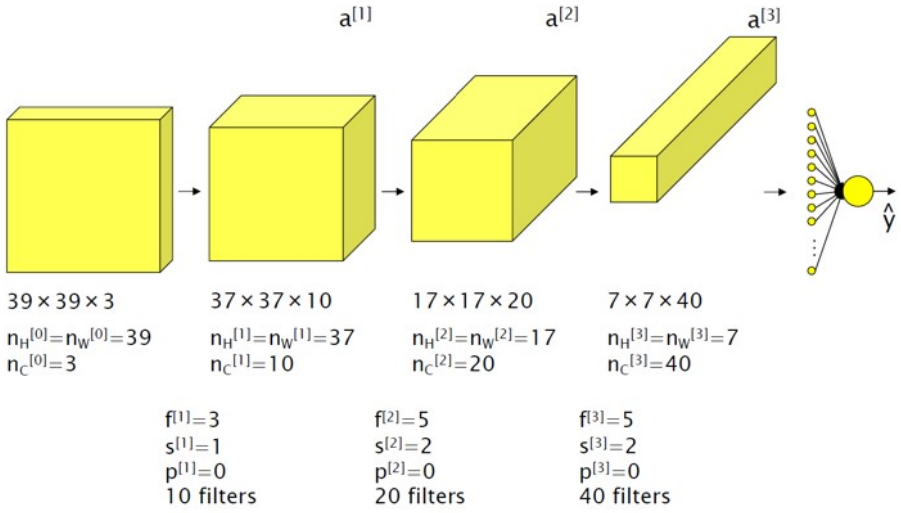

Figure 2.5: Example of ConvNet

A first approach is called 'sliding window detection': the image is divided in some portions, each passing through a CNN assigning the class probability. The portion with the highest probability for a certain object is the bounding box. The fact that the image is divided 'by hand' and that the portions dimension are fixed and not adaptable to the object dimension makes this approach not optimal. An evolution of this method id called 'convolution sliding window'. The CNN is modified, fully connected layers are replaced with convolutional layers, the output is a vector of length equal to the number of classes and each element represents a class probability. The network is trained for images of smaller dimension respectively to the inference images. This new CNN predicts portions of images with the same dimension of the training images. This dimension is the one of the bounding box of the predicted object in the bigger inference image. However this method is not optimal because of the bounding boxes fixed dimension and the inference slowness due to the fact that the CNN analyses areas in the images without any object of interest [14].

Trying to resolve this last mentioned problem, new approaches are born called 'two

stage methods' or 'region proposal'. The first 'stage' finds regions, called ROIs (Regions of Interest), where the object probability is high (fast step), the second is the classification algorithm (slow step).

In 2014 was created a first region proposal neural network, called 'R-CNN' (Region with CNN features). The regional proposal stage is performed using a 'selective search' algorithm, the classification stage is performed on a wrapped ROI with a VGG-16 [15]. This method is slow (inference takes 49 seconds) and computationally expensive, so an evolved 'Fast R-CNN' was designed in 2015. The first step is performed by a ConvNet which output is a feature map, then a ROI extraction algorithm is applied. Loop unrolling algorithm is applied to the ROI so that their dimension is aligned with the input of the following fully connected layers that execute classification of the bounding box (3 seconds inference) [16].

In order to minimize the first 'slow stage' time a further approach, called 'Faster R-CNN', makes region proposal using a convolutional network, named 'Region Proposal Network' which uses the feature map of the image to extract the ROI. The selected regions pass through a Fast R-CNN which makes classification. This method is the nearest to real-time object detection since inference takes only 200 ms. However the training procedure is tough: two different training must be performed for the first Region Proposal Network and one for the Fast R-CNN. All these approaches has also the capability to tune the dimension of the bounding box, which not always coincide to the ROIs [17].

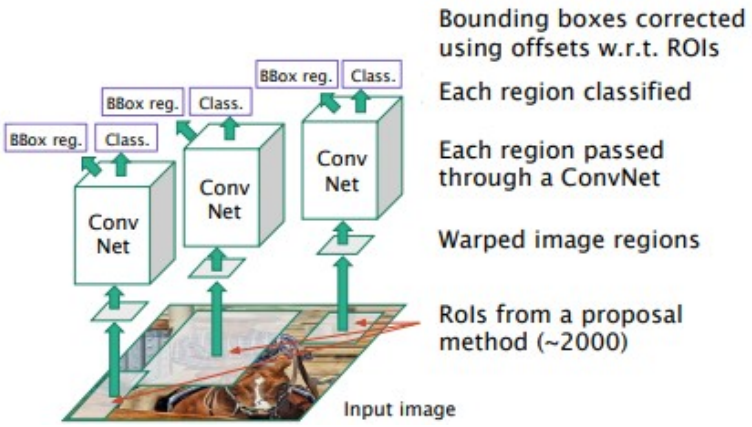

Figure 2.6: Region proposal approach

2.2.2 YOLO approach

YOLO is an acronym that stands for "You only look once", the innovative idea is to process an image without extracting the ROI, for this reason it is also called 'single stage approach'. In order to do that the image is overlapped with a grid, so each cell of the grid contains a portion of the image. A convolution network is then applied to each portion. The output is one vector for each cell composed of a binary value indicating whether the object is present on the cell, followed by 4 values indicating the coordinate of the center, height and width of the object bounding box and other N binary values (with N equal to the number of classes recognised by the neural network) indicating to which class the object present in that cell belongs. The union of all these vectors is the prediction of the neural network for the whole image [18].

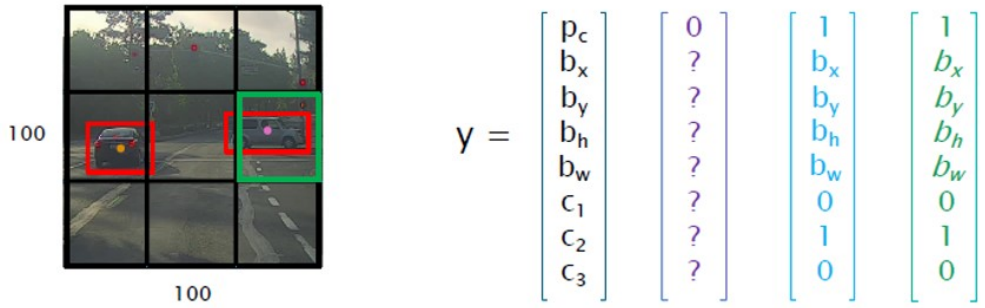

Figure 2.7: YOLO grid division of an image and relative output

In order to evaluate object detection algorithms the usual precison, recall and accuracy are not so useful because they do not work for each pixel. A relevant metrics is, instead, the 'Intersection over Union' (IoU). This metric calculates the localization accuracy, so how closely the predicted bounding box aligns with the original one through the comparison of the areas of the two bounding boxes.

The IoU is the ratio between the intersection and the union area of the predicted and the original bounding boxes. If it is equal to 0 the two areas do not overlap so the model obtained does not predict correctly, whereas, the more the value is near to 1 the better is the prediction and the model performance. If the value is greater then 1 it means the predicted bounding box is greater then the original, so it fully covers the object area. This metric is associated to a threshold value, usually equal to 0.5. If IoU has an higher value then the accuracy of the model is good, otherwise the training has to be restarted or fixed.

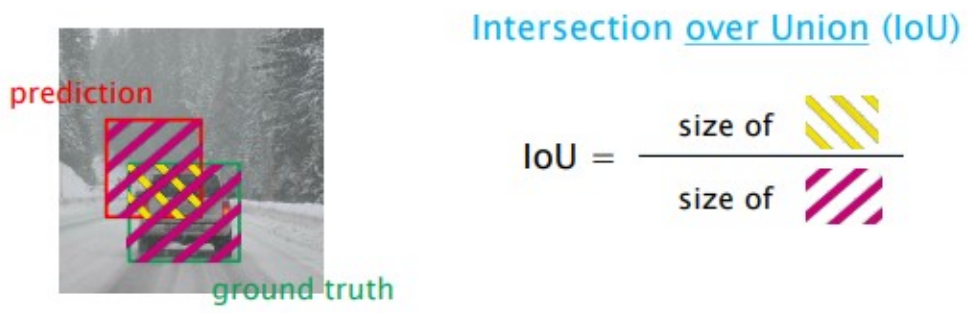

Figure 2.8: IoU metric

The huge success obtained by YOLO is not only due to the fact that it is a 'single-shot' object detection but also to this two important algorithms, that enforce and enlarge the prediction capabilities:

- Non-max suppression algorithm: the underlying idea is to identify the bounding box with the highest probability, examine all overlapping bounding boxes and if they have an Intersection over Union (IoU) value greater than a threshold, they are discarded. In fact if there is a high IoU value, it implies that the two bounding boxes are sufficiently close and/or overlapping, but the probability value of one is greater than the other. Hence, it makes sense to retain only the one with the higher probability.
- Overlapping objects algorithm: to perform object detection even on overlapping objects, the fundamental idea is to use a set of anchor boxes (reference examples) of different sizes for each label.

To calculate precision and recall for object detection tasks it is possible to establish a threshold of the IoU value. A prediction with a value of IoU bigger then this threshold is classified as true positive, instead, if the value is smaller it is a false positive. If the prediction is not done over a certain object it is a false negative, instead, no bounding box over a background area is true negative. This logic allows to create the confusion matrix, which is useful to resume the obtained results.

The ideal condition is that both precision and recall are high. In classification problems, the network output are the probabilities a bounding box belongs to a certain class. Defining a probability confidence threshold, all prediction with higher-equal probability are classified by the network. If the threshold is small, the model predicts an higher number of bounding boxes, so generally the chances to miss the ground-truth labels are lower, hence the recall is higher. If the threshold

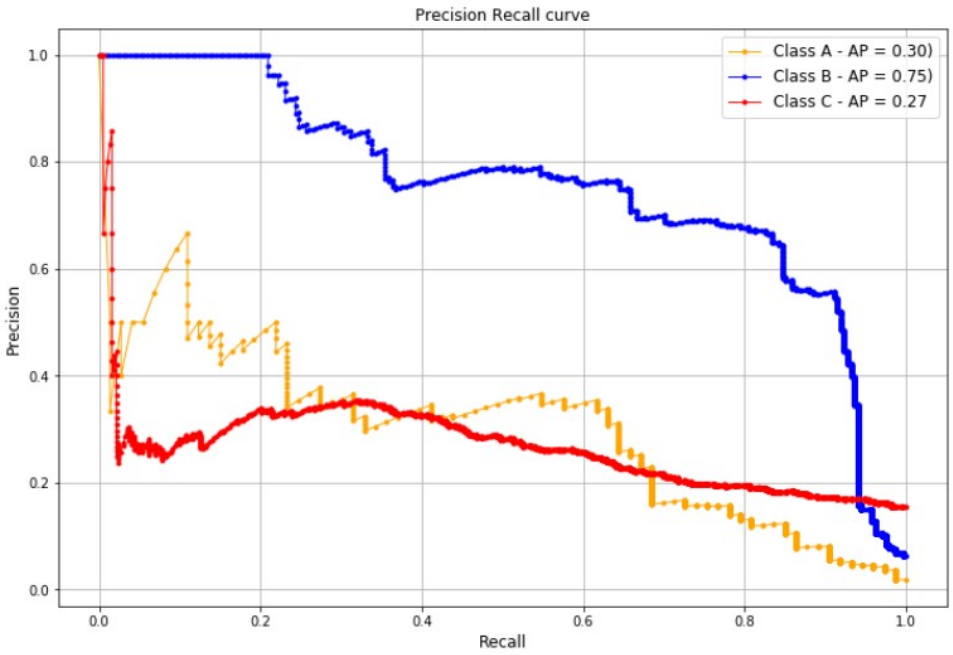

Figure 2.9: Precision-Recall curve

is high, the model predicts a smaller number of bounding boxes, so generally the model is more confident in what it predicts, hence the precision is higher.
It is possible to create a plot with recall on x-axis and precision on y-axis, called 'Precision-Recall curve' (PR curve). The Precision-Recall curve is obtained by plotting the model's precision and recall values as a function of the model's confidence score threshold: choosing a proper value of this threshold it is important for the definition of the desired trade-off between precision-recall.
Moreover, the underlying area by each class PR-Curve is called Average Precision (AP). Each class has its own value of AP. In order to have a good estimation of this value across all the classes the mean Average Precision (mAP) is calculated as the mean of all APs [19].
This scalar value is a good evaluation metric. In fact this value is high if both precision and recall are high, but decreases fast if one of these two metrics is low. Relying on this value the arbitrary process of choosing the better confidence threshold can be avoided, in fact AP is calculated over a set of confidence thresholds that covers tail ends of precision and recall values. The formula:

$$mAP = \frac{1}{n} \sum_{k=1}^{n} AP_k$$

with n equal to the number of classes and AP_k the average precision of class k.

Another useful plot (Figure 2.10) is the F1 score-confidence used to 'tune' a proper threshold of the confidence value. Since the higher is F1 the better is the 'compromise' between P-R, taking the maximum of this function is a proper choice as confidence value [20].

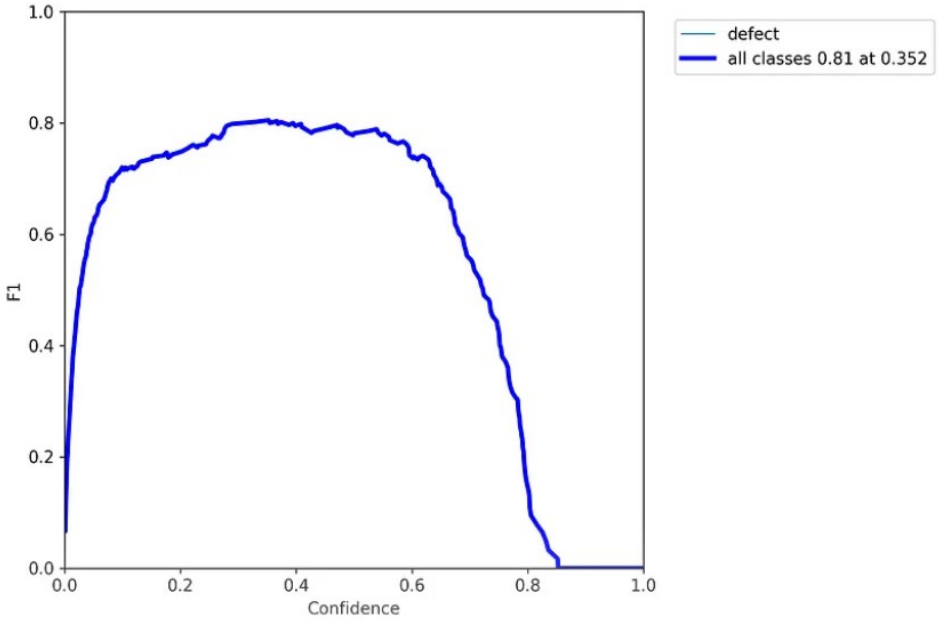

Figure 2.10: F1 score-confidence curve

2.3 Training process

A neural network comprises thousands of interconnected neurons. At the first level, these neurons receive an input in the form of an image, which the neural network perceives as a matrix of numerical values and transmits their output as input for the subsequent layer of neurons. The output of the last layer of neurons represents the neural network's prediction, denoted as h_θ. This prediction is a discrete value, which is the percentage of an object belonging to a specific label in a classification algorithm.

Figure 2.11 shows a simplified example of neural network, illustrating all computations performed by each neuron to output its prediction $a_i^{[j]}$, whose value is influenced by the input features x and by θ_i, commonly referred to as 'weights' of the neural network.

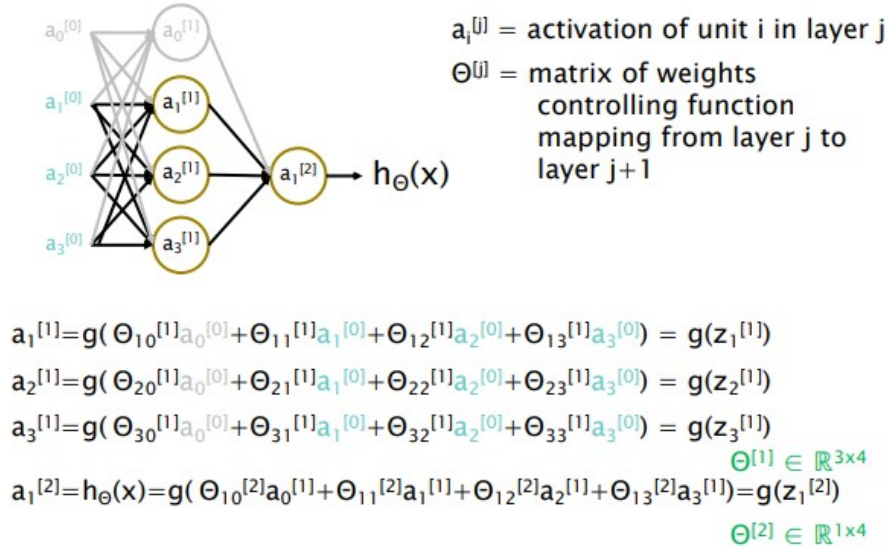

Figure 2.11: Neural network

The formula used to compute the value of $a_i^{[j]}$ is known as 'logistic regression':

$$h_\theta(x) = g(\theta^T x)$$

The logistic regression is a non linear function applied to a linear polynomial regression. In other words:

- Linear regression: first step, $\theta^T x$ equation gets a prediction starting from input x and tuned parameters θ vector. Its output is continuous.

- Sigmoid function $g()$: it transforms the linear regression problem into a logistic regression one, with the output included in the range [0, 1].

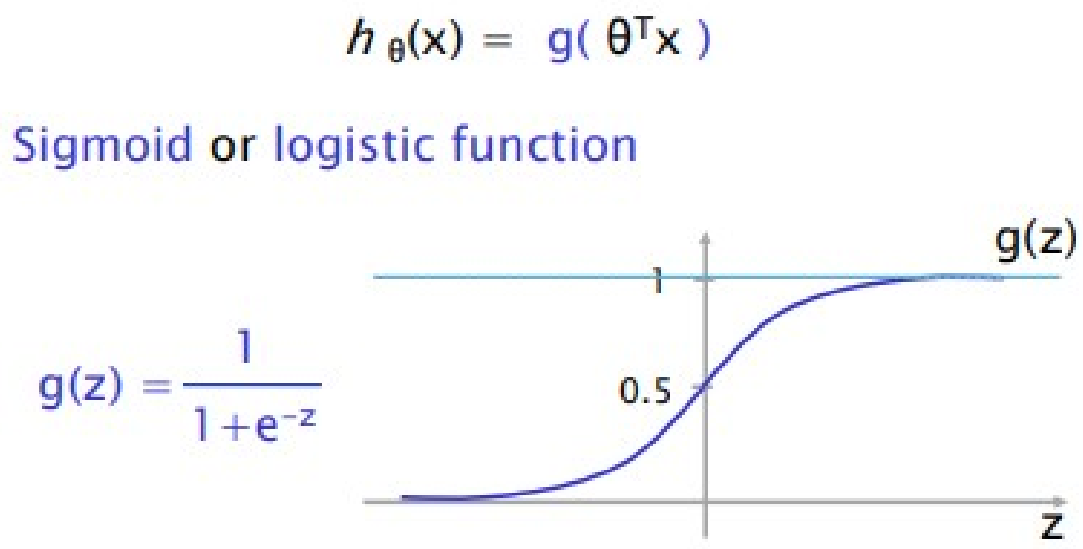

Figure 2.12: Logistic regression

In Figure 2.12 it is shown the plot obtained with the logistic regression function, which now becomes:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Since the output of classification problem is discrete, putting a threshold in 0.5 is assumed that all values greater then the threshold are predicted as 1, smaller instead are 0. This value is the output of the neuron.

For simplicity, assume the neural network is composed of just one neuron and h_θ is its output. Once obtained the expected output h_θ it is necessary to compare it with the true value y, the one the neural network is supposed to predict. Thus is used the Loss function, which measures the discrepancy between the prediction and the real label:

$$Loss(h_\theta(x), y) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

It is possible to sum all these Loss functions in order to create the Squared error cost function (Fgirue 2.14) able to compare all predictions made on a set of n inputs:

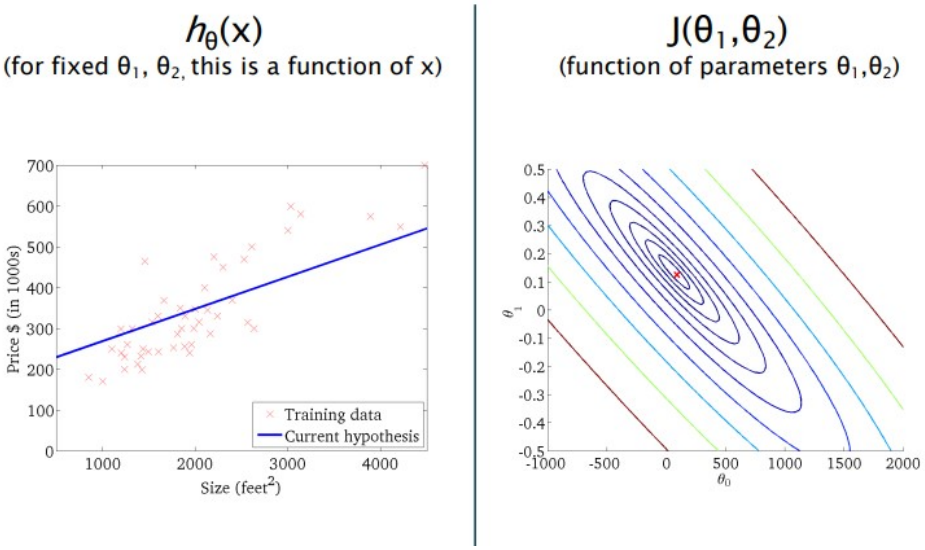

Figure 2.13: Fitting curve (left) and cost function (right)

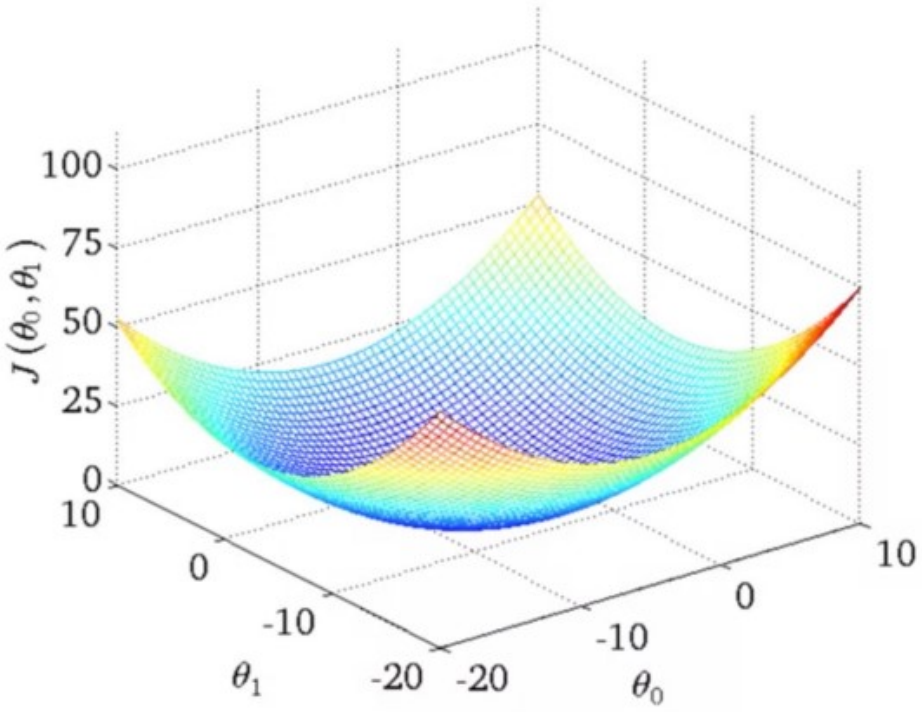

Figure 2.14: Cost function

$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$$

The goal is to minimize this function in order to have predictions and true labels as similar as possible.
Since h_θ value depends on all θ_i, the objective is to tune θ_i parameters ad hoc for the prefixed purpose. The cost function is convex over θ_i values. Convex functions property is that they have always a global minima, so solution of the optimization problem is always granted. An example of cost function is provided in Figure 2.14.

The gradient descent algorithm tries to create the best approximation graph, named fitting curve, which better approximates input data (left image in Figure 2.13).

This algorithm is based on the utilization of the gradient of the cost function [21]. The gradient of a function at a point provides the direction and magnitude in which the function grows most rapidly; consequently, the negative gradient at a point provides the direction and magnitude of maximum decrease, which is the fastest path to the minimum of a function. Therefore, it is possible to use this property of functions to construct an iterative algorithm for solving linear regression problems:

- Start from a generic point (θ_0, θ_1) and evaluate $J(\theta_0, \theta_1)$.
- Repeat the calculation of (θ_0, θ_1) until convergence by performing the following formula:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \text{ for } j = 0, j = 1$$

So, at each iteration, the algorithm updates the values of (θ_0, θ_1) by taking a step towards the minimum of the cost function. The size of the step taken toward the minimum is determined by the parameter α, known as the learning rate. This parameter is set by the programmer, choosing a small value of learning rate ensures convergence but it may bring to slow convergence, a big value instead does not ensure convergence to global minima but if succeed, it is fast.
In order to be sure that the algorithm performs property it is possible to do 'debug of gradient descent', plotting on x axis the number of iteration and on y axis the cost function. The function has to decrease and reach small values, the training must be done until the function does not reach a 'plateau', so even continuing the training the function does not decrease anymore.

Some conditions must be avoided: overfitting and underfitting.
A simple explanation of these two conditions is in Figure 2.16. The linear regression function (blue line) describes the output of the network.

- If the network has few features, a polynomial of too small degree or the training not yet sufficient to 'tune' proper weights for that particular problem, it causes 'underfitting'. Summarizing it means that the obtained logistic regression (blue line) does not fit well the training data (red crosses). The example is the left plot.
- If the network has a lot of features, a polynomial of too high degree or the training has continued for a too large number of epochs, it causes 'overfitting'. It means the obtained logistic regression fits too much the training data, not being able to generalize to different data. The example is the right plot.

A good compromise is like the polynomial in the center plot, as it would allow to have better inference on new data.

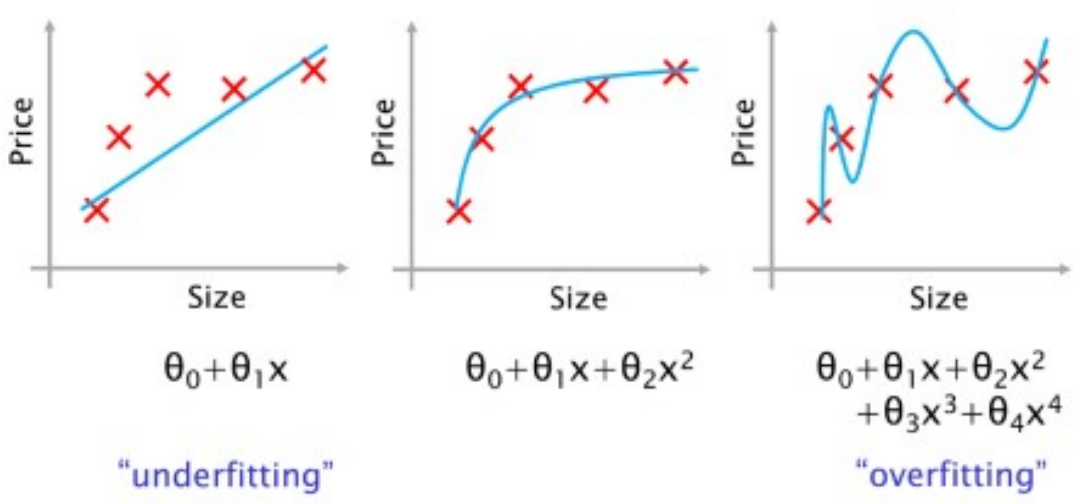

Figure 2.15: Overfitting and underfitting examples

A similar explanation can be given for the plots in Figure 2.16.

If training and validation losses has too high values it means prediction are different from the expectation, the network is not enough trained or the model not enough complex, causing 'underfitting' or 'high bias' [10] [22].

If training loss reaches little values increasing model complexity but the validation loss becomes bigger, it means that the model has adapted too much to training

data and it is not able to generalize to data that are not already seen, reason why the validation error is big. This is a case of 'overfitting' or 'high variance' [10] [23].

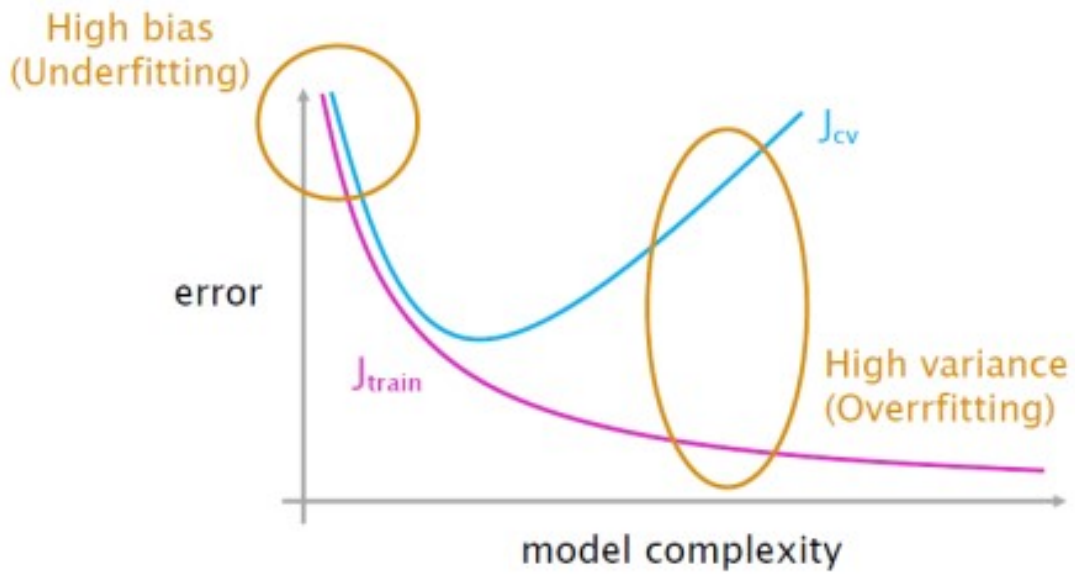

Figure 2.16: Overfitting and underfitting cost functions

Chapter 3

Computer vision tasks for Spot

Essential for the process of automation is that Spot autonomously recognises the objects to interact with. Computer vision algorithms have been employed to process the input stream from the Spot CAM, enabling the robot to detect these objects. In the next sections, some tasks are introduced and the neural network used to perform object detection are explained.

3.1 Face Anonymization

An important achievement is to make Spot recognize faces and show them blurred as output for privacy reasons (Figure 3.1).

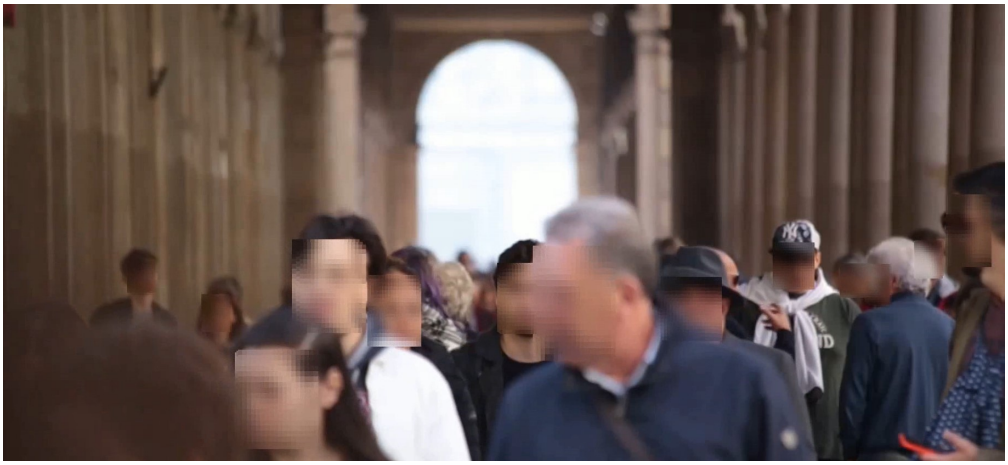

Figure 3.1: Example of RetinaFace output

For this purpose, it is adopted the 'RetinaFace' neural network, available on GitHub, to detect faces on images and a Python code to blur the identified bounding boxes using OpenCV library capabilities [24].
RetinaFace is a deep learning based cutting-edge facial detector for Python coming with facial landmarks. Its detection performance are good even in the crowd [25].

RetinaFace is a face detection Python library which also includes face pose estimation, 2D face alignment, face segmentation (done through a 2D facial landmark localisation) and 3D face reconstruction, reason why it is called multi-level face localization. It is based on a single-shot inference since all these predictions are done simultaneously [26].

For each training anchor i, it is minimized the following multi-task loss:

$$\mathcal{L} = \mathcal{L}_{cls}(p_i, p_i^*) + \lambda_1 p_i^* \mathcal{L}_{box}(t_i, t_i^*) + \lambda_2 p_i^* \mathcal{L}_{pts}(l_i, l_i^*) + \lambda_3 p_i^* \mathcal{L}_{mesh}(v_i, v_i^*)$$

where

$$\mathcal{L}_{mesh} = \mathcal{L}_{vert} + \lambda_0 \mathcal{L}_{edge}$$

with

$$\mathcal{L}_{vert} = \frac{1}{N} \sum_{i=1}^{N} \|V_i(x, y, z) - V_i^*(x, y, z)\|_1$$

$$\mathcal{L}_{edge} = \frac{1}{3M} \sum_{i=1}^{M} \|E_i - E_i^*\|_1$$

In this loss function are considered each of the tasks of RetinaFace. The classification loss $\mathcal{L}_{cls}$ is the softmax loss for binary classes (face/not face). The detection task is underlined by the $\mathcal{L}_{box}$ loss, 2D facial landmark localisation is ensured by the $\mathcal{L}_{pts}$, instead 3D face reconstruction by $\mathcal{L}_{mesh}$. In fact t_i, l_i, v_i are box, five landmarks and 1k vertices predictions, t_i^*, l_i^*, v_i^* are the corresponding ground-truth, p_i is the predicted probability of anchor i being a face, and p_i^* is 1 for the positive anchor and 0 for the negative anchor.

Focusing particularly on $\mathcal{L}_{mesh}$, a mesh is a 3D model used to define an object. It is the union of two different representations, one with vertices, one with triangles [10].
The first one is taken into consideration by $\mathcal{L}_{vert}$ where $N = 1103$ is the number of vertices, V is the prediction of our model and V^* is the ground-truth. The second one by $\mathcal{L}_{edge}$ where $M = 2110$ is the number of triangles, E is the edges length calculated from the prediction and E^* is the edges length calculated from the ground truth.

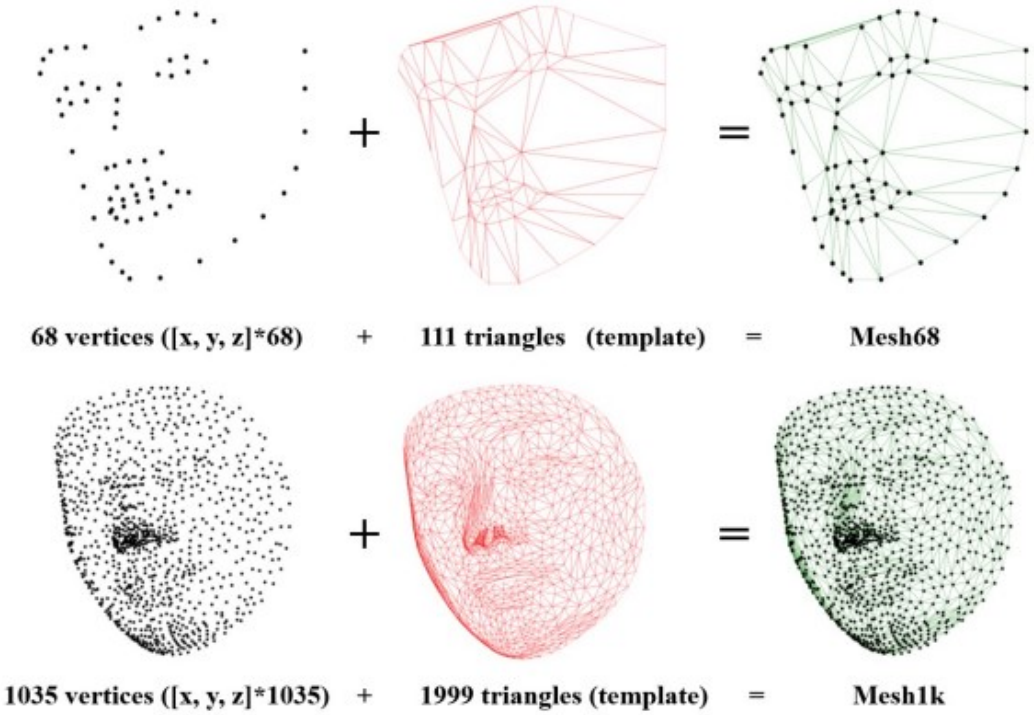

Figure 3.2: Example of meshes of faces

Comparing Mesh68 and Mesh1k in Figure 3.2, it becomes obvious that more vertices make the mesh more informative and smooth.

Figure 3.3 shows RetinaFace network structure, the feature pyramid network gets the input face images and outputs five feature maps of different scale. A first context head module predicts the bounding box from the regular anchor, while subsequently, the second context head module predicts a more accurate bounding box using the regressed anchor, which is generated by the first context head module [26].

Anchors enable users to predict how a model would behave on unseen instances with less effort and higher precision. Anchors are reference examples, bounding box proposals. The model matches the anchors to a ground-truth box if IoU it is over a specific threshold and to the background if IoU is under this threshold [26] [27].

The created script uses OpenCV [24], capture sstream video. Each frame is sent to the RetinaFace function '$detect\_faces(frame)$', which performs inference. Then, its output are the facial area coordinates and the five landmarks (eyes, nose and mouth) with a confidence score shown in Figure 3.4.

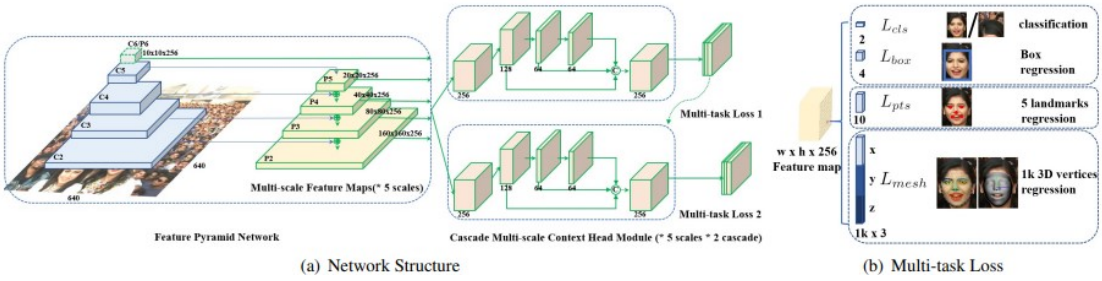

Figure 3.3: RetinaFace network structure(a) and detailed loss illustration (b)

This neural network has even more capabilities than what is needed. From each predicted face it is taken just the "facial_area" prediction, so the bounding boxes of the predicted faces.


```
{
    "face_1": {
        "score": 0.9993440508842468,
        "facial_area": [155, 81, 434, 443],
        "landmarks": {
          "right_eye": [257.82974, 209.64787],
          "left_eye": [374.93427, 251.78687],
          "nose": [303.4773, 299.91144],
          "mouth_right": [228.37329, 338.73193],
          "mouth_left": [320.21982, 374.58798]
        }
  }
}
```


Figure 3.4: RetinaFace output

These areas are then pixeled by subdividing into blocks and applying OpenCV mean for each of them. Then the script overlays the anonymized versions onto the original image in order to preserve privacy.

3.2 License place recognition

In this section is explained how the YOLOv8 model implemented by Ultralytics can be used to perform real-time object detection and image segmentation on MS-COCO (Microsoft Common Objects in Context) dataset [28]. Ultralytics offers weights of different types of YOLOv8 models pre-trained on MS-COCO dataset. Yolov8 model is available on GitHub and takes inspiration from previous versions of this algorithm. It is currently the latest version of the famous object detection algorithm characterized by better performance in terms of accuracy and speed [29].

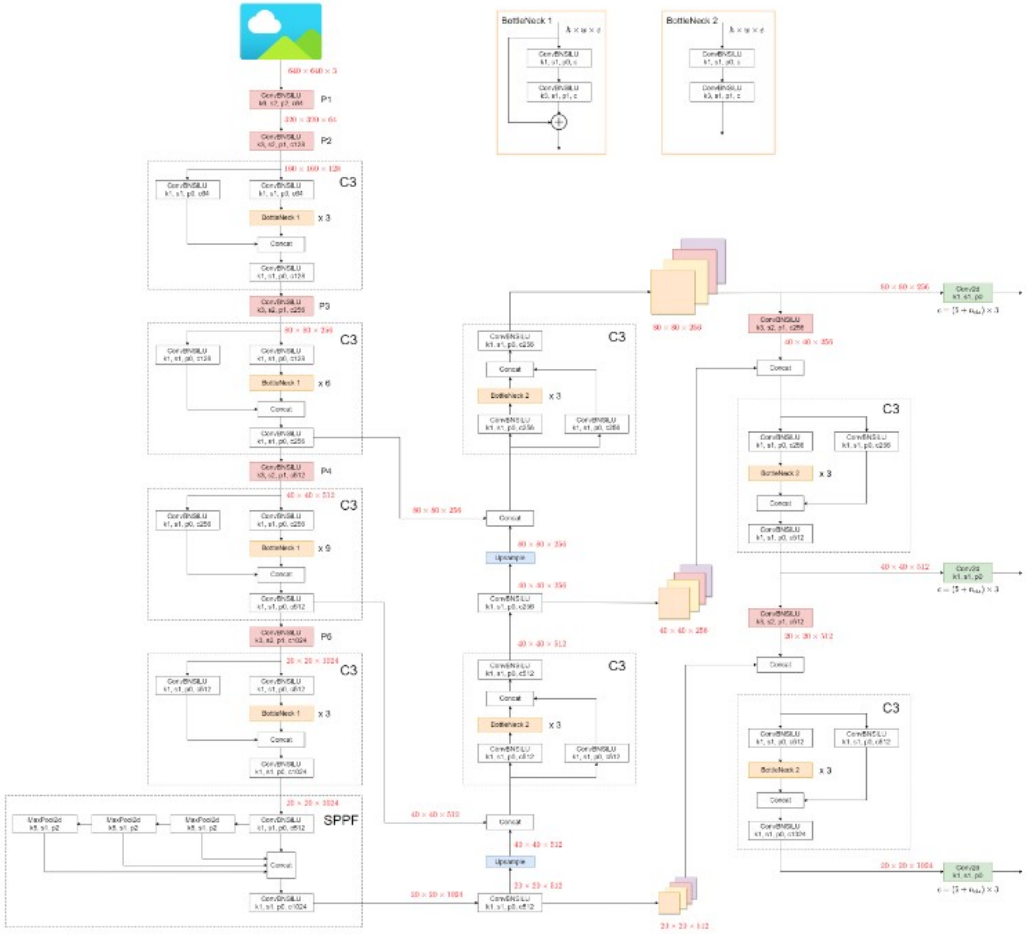

Figure 3.5: YOLOv5 architecture

The architecture shown in Figure 3.5 is the YOLOv5 implemented by Ultralytics. The implementation of YOLOv8 is based on that model enriched with some

architectural tweaks as explained by the creators. The neural network is a classical convolutional, starting from 640x640x3 images applies a series of conv levels. The structure is characterised by the repetition of a 'C3' structure, which are basically 'residual blocks'. This type of structures give two alternatives to data flow: pass through a bottleneck (so slower) structure of two convolutional levels or directly go to the next structure following a 'skip connection' or 'short-cut' [30].

Classical CNN with all weights greater then 1 risks to produce increasing activation functions, causing explosion of the gradient, whereas weights always smaller then 1, risk to produce decreasing activation functions, causing vanishing of the gradient. Since the gradient is the variation of the loss function during the training, if its value is not under control it risks to cause problems to the network learning procedure. The residual block structure passes the activation of a level directly as inputs to a distant layer, preventing vanishing and exploding gradients [10].

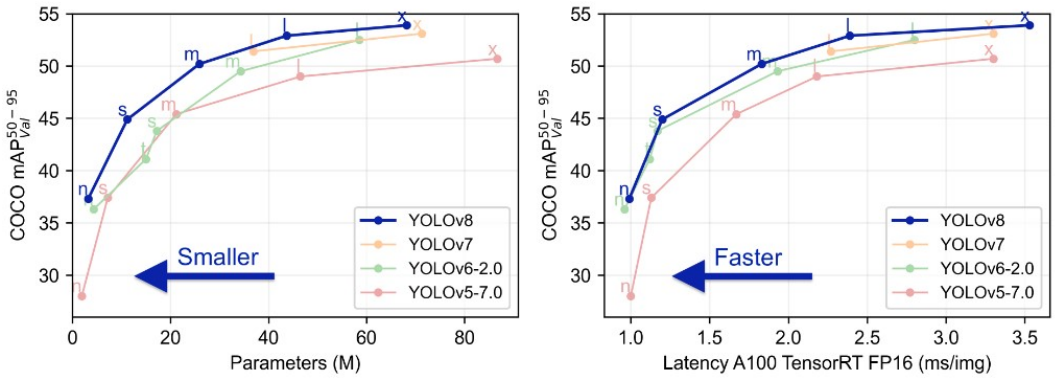

Figure 3.6: YOLOv8 models performance

As can be seen from Figure 3.6 and Figure 3.7 Ultralytics offers various implementation of YOLOv8. All these models can be found inside Ultralytics library as yaml files (ex.yolov8n.yaml). Moreover some weight files (ex.yolov8n.pt) can be found inside GitHub repository, each representing a certain model trained over a well know dataset, like MS-COCO. In other situations, when it is necessary to train the model over particular classes not available on well-known datasets, a training from scratch can be done using the .yaml file. The output of this training is a .pt file from which it is possible to resume the stopped training, putting it as input to the training function [31].
As previously mentioned, there are different models within the YOLOv8 family: YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l and YOLOv8x. The letters in these names stand for normal (standard one), small, medium, large, and extra-large,

Models download automatically from the latest Ultralytics release on first use.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

- **mAP^{val}** values are for single-model single-scale on COCO val2017 dataset.
  Reproduce by `yolo val detect data=coco.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance.
  Reproduce by `yolo val detect data=coco128.yaml batch=1 device=0|cpu`

Figure 3.7: Different YOLOv8 models performance and number of parameters

respectively. These names are indicative of the number of parameters that compose each neural network, so of the model increasing complexity.
Model "n" has the smallest number of parameters but also the slower mAP, in contrast to extra-large, which has the higher number of parameters but the higher mAP. The choice among these models depends on the specific requirements: whether a lighter model is needed or if superior performance and faster processing are the priorities.
In the case of Spot is better to have a higher fps (frame per second) inference capability, so to interact faster with the environment in case of necessity. This is the reason why for all the thesis tasks is selected the "n" model.
A plot() function allow to show on screen colored bounding boxes for each class and the model confidence about the objects to belong a certain class; these values are basically the outputs of the neural network.

Another implementation developed is the capacity to make Spot recognize and read license plates. In order to do this, it is used docTR, a open repository performing OCR (optical character recognition) [32]. This algorithm is used only on the bounding boxes that are predicted as vehicles (car, bus, motorcycle) from YOLOv8.

A possible use case involves sending Spot to patrol the parking areas of establishments or industries and to create a dictionary-based vehicle detection system. This system would identify vehicles with the necessary permissions to park in specific areas or notify users in case of any suspicious vehicle detection.

The OCR is a model able to recognise typewritten characters inside an image or frame [33]. The OCR output is filtered, only strings of length 7 are taken, in particular respecting the format (2 letters - 3 numbers - 2 letters). However OCR could 'read' incorrectly some characters, so an additional Python code is used to 'clean' the string. A list of negligible characters, which are impossible to be present on a license plate, is written so that if the OCR predicts one of them the code ignores it. Moreover a number-letter replace list is added (e.g. 'A': '4', 'B': '8','G': '6', 'I': '1', 'K': '4', 'P': '9', 'Q': '0', 'S': '5', 'T': '7', 'Y': '4', 'Z': '2'), associating similar characters. If in a slot (e.g. 4th) where it is supposed to be a number, but, instead, there is a letter (e.g. B), the algorithm assumes the OCR is wrong and substitutes it with the corresponding character (e.g. 8).

Figure 3.8: License place recognition with YOLOv8 of a frame from a video taken with Spot

3.3 Exit signals and fire extinguishers detection

The main environments where Spot is suited to work are the industrial plants. A further achievement is to make Spot recognise exit signals and fire extinguishers in this context. For this purpose another computer vision algorithm has been implemented using YOLOv8 model by Ultralytics. The task is the detection, whose purpose is to provide safety by making Spot autonomously identify these security objects, facilitating efficient evacuation procedures in emergency situations and reporting to users fire extinguishers correct position during daily patrols.

Differently then in previous cases, the training is performed starting from a dataset of 1200 real images manually labelled, using the yolov8n.yaml model. The stardard 'n' model is used instead of more powerful models because, as in previous cases, working with Spot, it is more necessary to have real-time scenarios prediction, so faster inference model, instead of better accuracy one [31].

The classes predicted by this network are four: exit signal, fire extinguisher, fire extinguisher signal and fire extinguisher gauge.
The results obtained after 30 epochs of training are reported in Figure 3.9:

epoch	train/box_loss	train/cls_loss	train/dfl_loss	metrics/precision(B)	metrics/recall(B)	metrics/mAP50(B)	metrics/mAP50-95(B)	val/box_loss	val/cls_loss	val/dfl_loss	lr/pg0	lr/pg1	lr/pg2
1	3,9044	4,8916	4,2719	0,00089	0,06061	0,00208	0,00097	3,6888	4,7237	4,0145	0,070411	0,0032877	0,0032877
2	3,089	3,4634	3,5357	0,86003	0,23734	0,28714	0,12742	2,4922	2,6748	3,2662	0,040192	0,0064025	0,0064025
3	2,4105	2,2763	2,681	0,48038	0,36961	0,43375	0,15475	2,4532	2,5518	2,8772	0,009754	0,0092974	0,0092974
	...												
28	0,7056	0,51421	1,0557	0,9904	0,75821	0,83247	0,60572	0,98384	0,61802	1,2105	0,00142	0,00142	0,00142
29	0,68087	0,50454	1,0411	0,93969	0,75574	0,82892	0,62749	0,87321	0,60452	1,1392	0,00109	0,00109	0,00109
30	0,6732	0,49743	1,0319	0,88247	0,80727	0,81943	0,62117	0,90896	0,62065	1,1204	0,00076	0,00076	0,00076

Figure 3.9: Losses and metrics obtained during the training

In Figure 3.10 are shown the plots obtained for each loss, with the number of epochs on the x-axis.

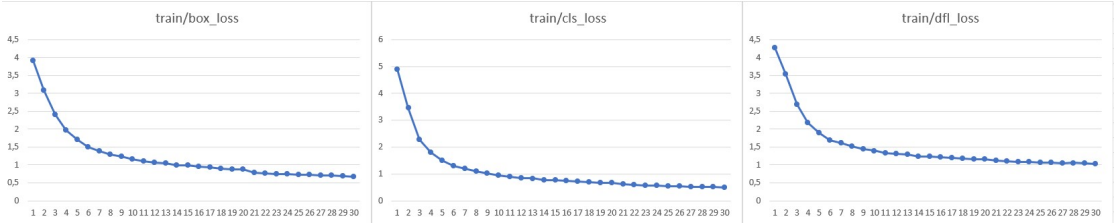

Figure 3.10: Exit signals and fire extinguishers train losses

The total Loss is the combination of three components: bounding box loss, classification loss and distribution focal loss. Respectively the first one measures how precise is the bounding box prediction, the second one describes how good is the network in predicting each class, whereas the last one is used to balance the

importance given to each class when the dataset classes are not equally distributed. A more extensive description of each of these losses is provided on Chapter 6.

As the plots show, all the losses reach good values, near to zero, however plateau seems not to be already reached, so the model could be further trained as a next step.

Figure 3.11 shows the loss obtained also for the validation set.

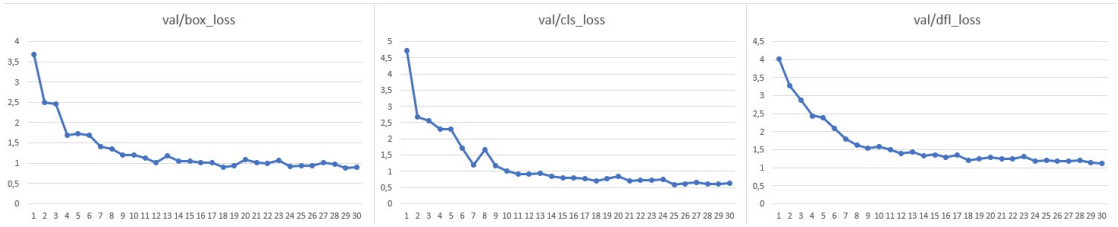

Figure 3.11: Exit signals and fire extinguishers validation losses

The metrics values of the validation set in Figure 3.12 show the confusion matrix (left), which has good predictive values for all classes with exception for the fire extinguisher gauge, with a true positive rateo of only 26%; the rest is classified as background. A better prediction accuracy could be obtained adding more images of gauges in the dataset.

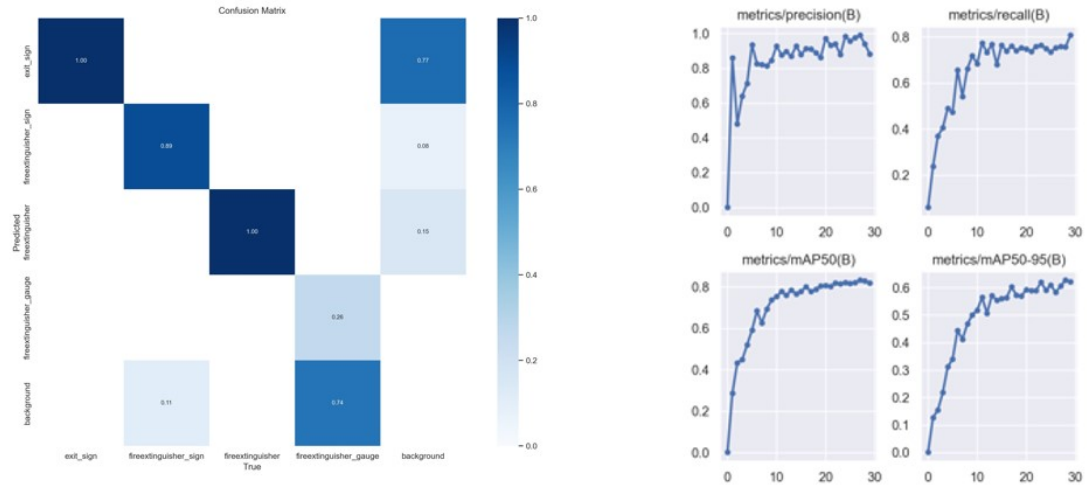

Figure 3.12: Exit signals and fire extinguishers confusion matrix, precision, recall and mAP

The plots of precision, recall and mAP all demonstrate a growing trend during

the pass of the 30 epochs, highlining an improvement of the performance (right of Figure 3.12).
The same behaviour is recognisable in Figure 3.13, which reports the precision-recall and F1 graph in function of the confidence. From this plot is possible to 'tune' a proper value of the confidence threshold, improving the model performance. Values of confidence between 0.4 and 0.6 maximise the F1 score of the model.

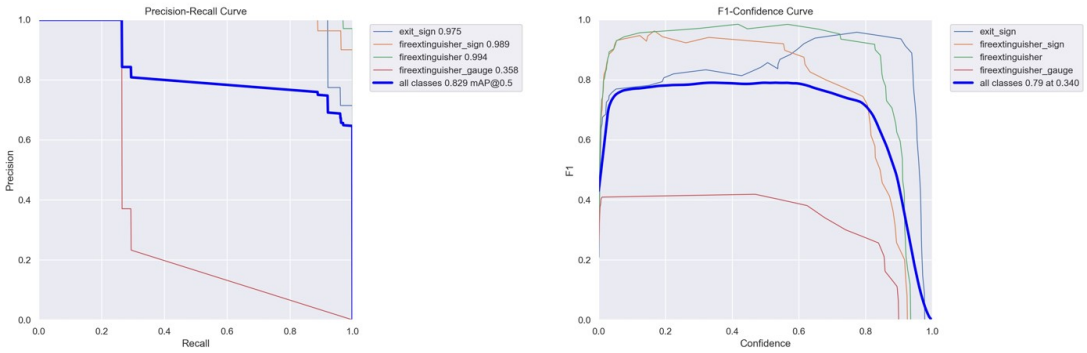

Figure 3.13: Exit signals and fire extinguishers PR and F1 curves

Figure 3.14 displays some examples of predictions obtained on the validation set.

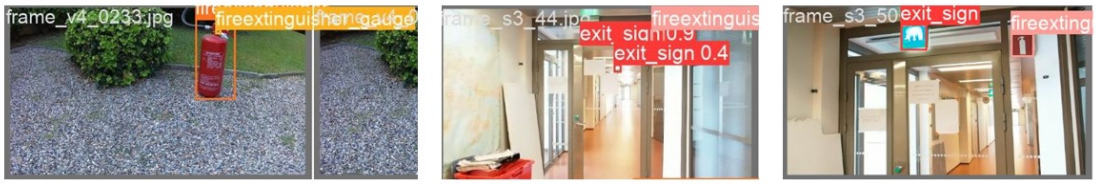

Figure 3.14: Exit signals and fire extinguishers inference examples

Chapter 4

Integration of computer vision algorithms on Spot

In this chapter it is explained the 'Spot POV' (Point of View) Python project, whose aim is to exploit and integrate on Spot streaming the computer vision algorithms developed (and explained in Chapter 3), in order to add AI on the robot, giving it the possibility to be conscious of the environment and to act on the basis of the objects or people surrounding it.

This code takes the video streaming from different Spot cameras and uses the previous explained neural network to predict the environment for each frame, then it shows on the screen of a PC each predicted frame.
This project gives the possibility to take images from different cameras placed all over the body of the robot [8].

- Five RGBD (color and depth) 'fisheye' cameras are included on the body of the robot (Figure 4.1), namely 'back fisheye', 'frontleft fisheye', 'frontright fisheye', 'left fisheye' and 'right fisheye'. The name of this cameras is due to the curved perspective of the vision. This type of cameras have a wide field of view, giving Spot a total overall optical field of view of 360 degrees.

Figure 4.1: Front fisheye cameras of Spot

Near these cameras there are also the stereoscopic depth camera, which integrate the robot with obstacle avoidance. These cameras are depth sensors, each of them has 90 degrees panoramas and observes over a distance of 2 meters.

- On the back of Spot can be added as payload a Spot CAM+ (Figure 4.2). This camera significantly improves the robot's image capability and the user's awareness of the environment. This camera has spherical camera, with 360 degrees panoramas. Moreover it is endowed of a PTZ (pan-tilt-zoom) camera with a 30x optical zoom.

Figure 4.2: Spot CAM+

- If the used Spot is endowed of a robotic arm it is also possible to take images from the camera mounted inside the gripper of the robotic arm (Figure 4.3).

Figure 4.3: Spot Arm Gripper camera

4.1 Project explanation

The project is developed in Python. The previously listed images sources capture frames and the API processes them using artificial intelligence models as YOLO and RetinaFace to detect fire extinguishers, exit signals, objects from MS-COCO dataset, to read license plates and to anonymize faces.

Using Spot SDK the application imports *bosdyn* library and creates a 'robot' object. To take control of the robot authentication is mandatory for security and safety reasons [34]. The idea is to create a REST API so a programming interface that uses HTTP protocol to allow communication between systems. The API waits for HTTP POST request containing the image to classify, the allocated resource makes the prediction and turns back the result of classification through a HTTP POST response. Figure 4.4 clarifies this idea.
REST API organizes resources using URLs (Uniform Resource Locators) which are unique identifier for the neural network functionalities. In fact the second step of the code is to create an endpoint for YOLO and RetinaFace, so to allocate their URLs.

Varying a boolean value on a configuration file the user can decide which neural network makes inference during the execution of the application, if 'yolo' is set to False and RetinaFace to True the streaming shown on screen identifies only anonymized faces, so each frame is passed only to RetinaFace service [35].

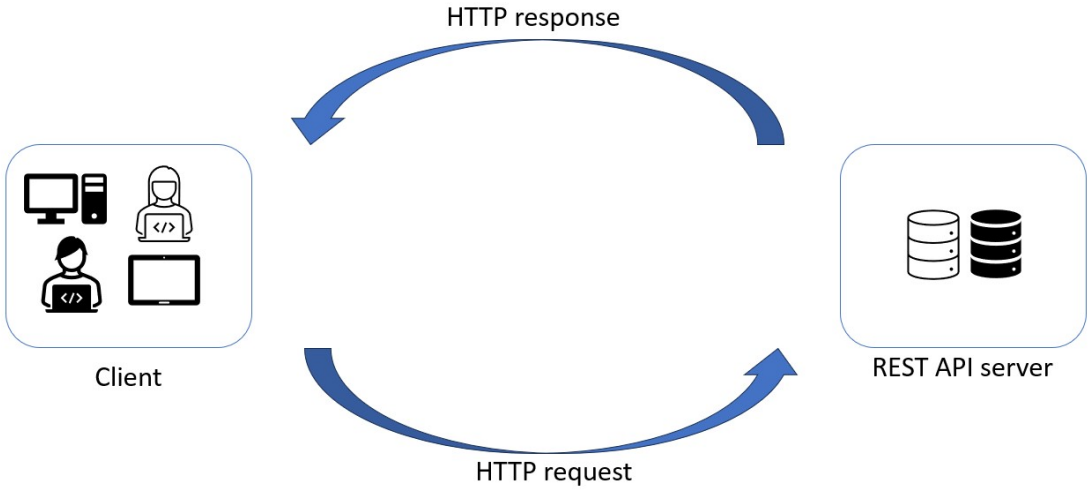

Figure 4.4: Rest API

The main reason why this computer vision streaming application is implemented as a REST API is the need for a fast algorithm, allowing Spot to have a real-time

perception with the minimum delay. Since neural networks, in this case especially RetinaFace, need a lot of time to make inference, running them on the robot could slow down the process. In fact inference on a Intel Core i7 CPU - 2.20GHz - 16GB RAM - 6 cores has a prediction time of about 200 ms per frame for the RetinaFace network and 20 ms per frame for the YOLO one, allowing a quite good streaming. The processors mounted on Spot are even slower (Intel i5 - 16GB RAM - 1 core) [36], so performance would not be good. The idea is to take a frame from Spot and to make the prediction on a remote server, also a PC, endowed of a GPU or higher performance CPU.

4.2 REST API logic

Each streaming frame is taken by a function of the SDK and it is initially saved in OpenCV image format [34].

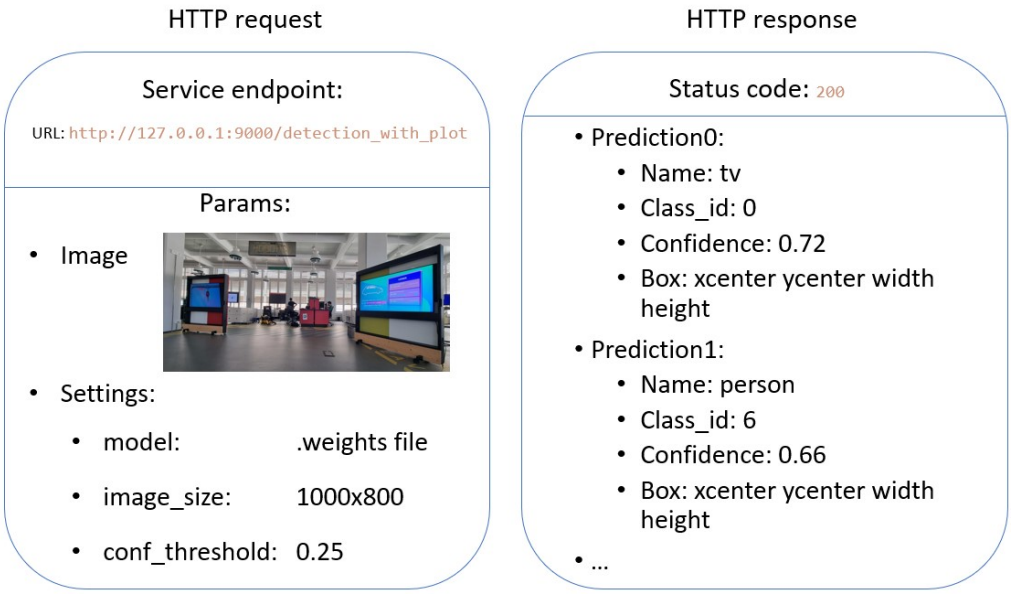

Figure 4.5: HTTP messages

The REST API uses HTTP protocol to talk with the client. First the user sends an HTTP POST request containing the URL of the computer vision service, the image decoded from OpenCV to base64 so that it can be inserted inside a .json file and all the possible settings, including the weight file of the neural network chosen for detection, the image size and the confidence threshold over which the object detection prediction is considered valid and reliable.

The obtained response is another HTTP POST which contains a status code and a list of predictions, one for each identified object. This predictions contain the name and id of the class, the detection confidence and its bounding box position. A schema resuming the content of these HTTP message is shown on Figure 4.5.

The '*status code*' in the HTTP response identifies if the action has been correctly performed or there has been some errors. These are the most frequent codes:

- 200 OK: standard response for successful responses;
- 404 Not Found: required resource not found;
- 500 Internal server error: generic error.

The image predicted in base64 is then converted back to Pillow format, a Python library with image editing capabilities, and it is shown on screen, with all names and confidences plotted as in Figure 4.6.

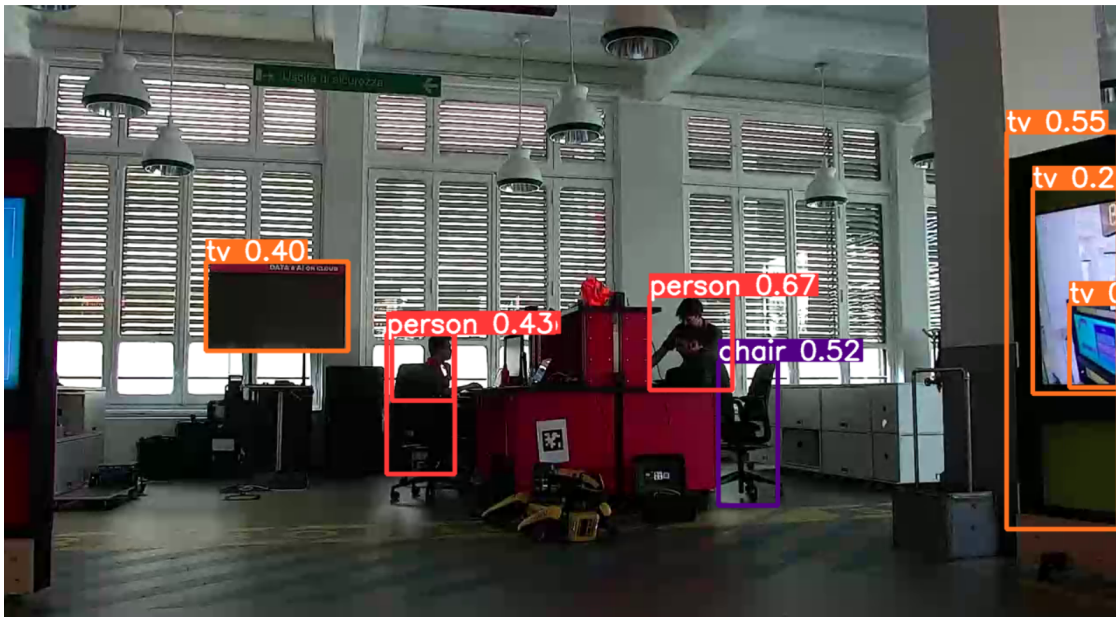

Figure 4.6: Inference result of YOLO detection on MS-COCO dataset

4.3 Use case and possible further researches

Self-aware robot's understanding of the environment is one of the greatest and most exciting challenges of the recent years.
The use of neural networks in robotic field is the way to achieve this goal. Computer vision algorithms allow robots to understand and interact with the surroundings. Thanks to this project Spot is able to recognise all object from MS-COCO dataset, to read licence plates in cars bounding boxes and to anonymize faces, preserving privacy.
However, other computer vision algorithms to enlarge Spot use cases could be added to this Python project, which is particularly scalable, since the only need is to create a new endpoint and call it from the main script 'spot_streaming' which is the HTTP request handler.

Current missions on Spot are pre-registered with fixed waypoints, each corresponding to a specific object. For each waypoint, a custom action is created to execute a specific task. Examples of these tasks include reading a gauge at a waypoint in front of a manometer or detecting leaks from a valve at a waypoint in front of a tap.
Further research can be done on this direction, with the aim is to further increase Spot autonomy. Autonomous navigation could be one of the fields of development, unifying SLAM (Simultaneous Localization and Mapping) algorithm and Spot POV. Thanks to object recognition, the robot could achieve more precise environment knowledge. Spot could recognise objects during its mapping, save their position in the map and interact with them on the basis of the predicted class. The possibilities are nearly infinite.

Chapter 5

Custom dataset creation pipeline

The machine learning solutions explained in Chapter 3 are all trained on datasets which are well-know and fully available on Internet, like MS-COCO dataset, and extendable adding new labelled images, e.g representing exit signals and fire extinguisher.

However, it is not always possible to find these datasets. Maybe the request is to make Spot recognise some custom objects, whose images are rare on Internet. An illustrative example could be a particular type of valve which is used in all the taps of an industrial establishment. Spot could recognise this type of valve and interact with it. Unfortunately, a dataset for this specific purpose may not be available.

For this reason a custom dataset creation pipeline has been developed, using Python. It uses a set of *templates*, which are the objects wanted for the network to recognise, and a set of *patterns*, which are the backgrounds of the images to create.

- A first script called '*generate_dataset.py*' creates a dataset varying templates dimensions, rotating them, adding blur and noises and 'pasting' one or two over a casually chosen background. This code is available both for detection and segmentation datasets. In general the '$config\_generate\_dataset.yaml$' file contains parameters useful for the customization of the dataset.
- A second script called '*training.py*' is used to train and validate a previously created validation and training set using a YOLOv8 model.
- A third script called '*test.py*' uses some images from the test set to output testing metrics.

5.1 Pipeline structure

Figure 5.1 shows a schema of the pipeline. Three main folders constitute the project: *generate_dataset*, *training* and *testing*.

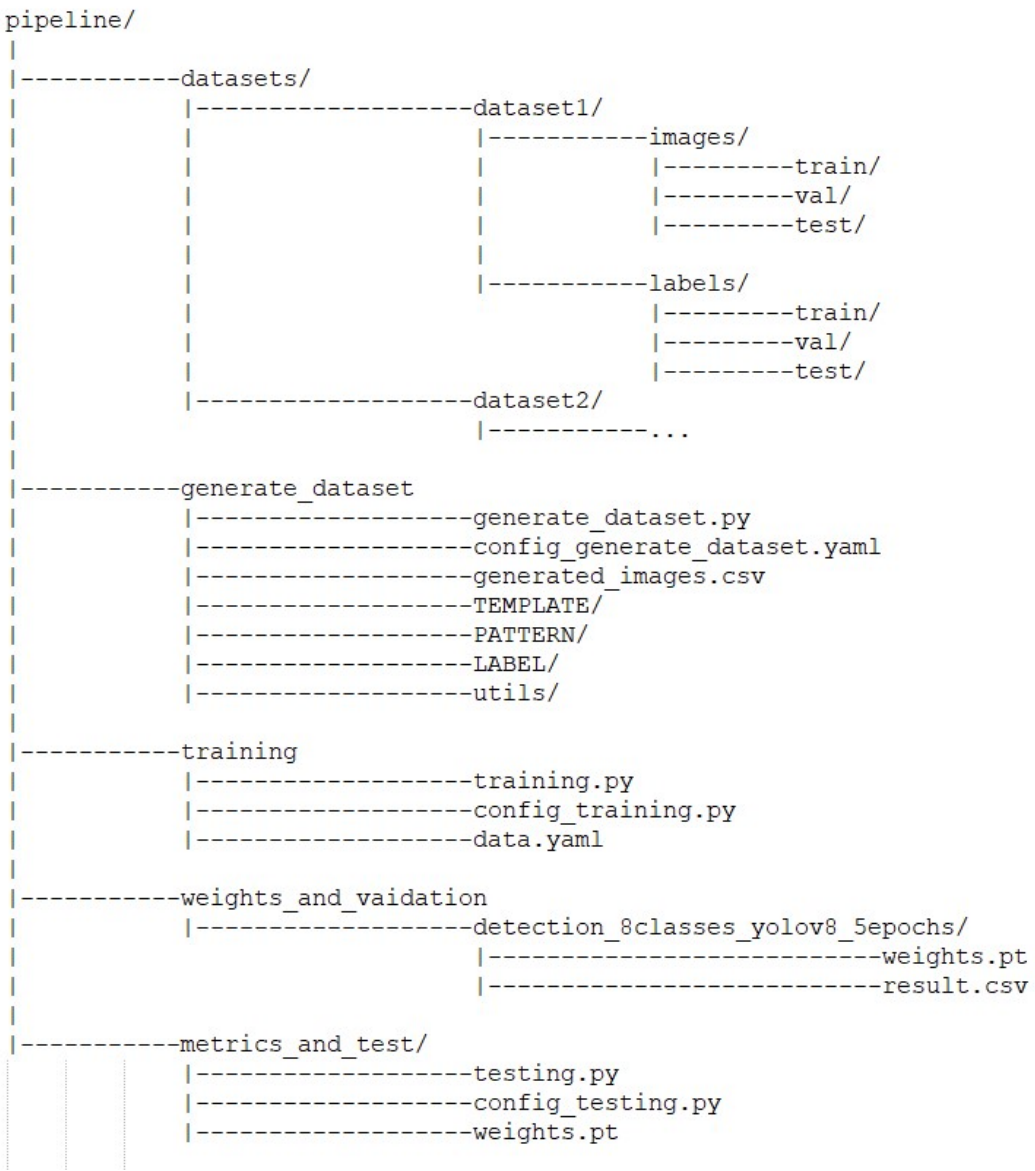

Figure 5.1: Pipeline folder

5.2 The generate dataset folder

Inside the *generate_dataset* folder there are two main files: *generate_dataset.py* and *config_generate_dataset.yaml*, the first is the main structure of this project, it uses other Python files inside the *utils* folder and imports many variables saved inside the second .yaml file, which values an user can 'tune' to create the needed 'custom' dataset ad-hoc for the specific task.

The *generate_dataset* folder contains templates, patterns, labels and the main script *generate_dataset.py* which uses these base photos and mixes them to create various datasets.
On the base of the choice made on the configuration file it creates datasets for detection or segmentation.
The segmentation task is quite similar to the detection one, it is a classification and localization of different classes with the additive capacity to identify the contour of the objects, instead of just the rectangular bounding boxes. Since detection bounding box is 'obtainable' from segmentation one, the project logic treats both cases as segmentation (later will be better explained this concept, Figure 5.6), differentiating only the final label form:

- Detection: $class\_id\ x_{center}\ y_{center}\ width\ height$
- Segmentation: $class\_id\ x_1y_1\ x_2y_2\ ...\ x_ny_n$ (template contour coordinates list)

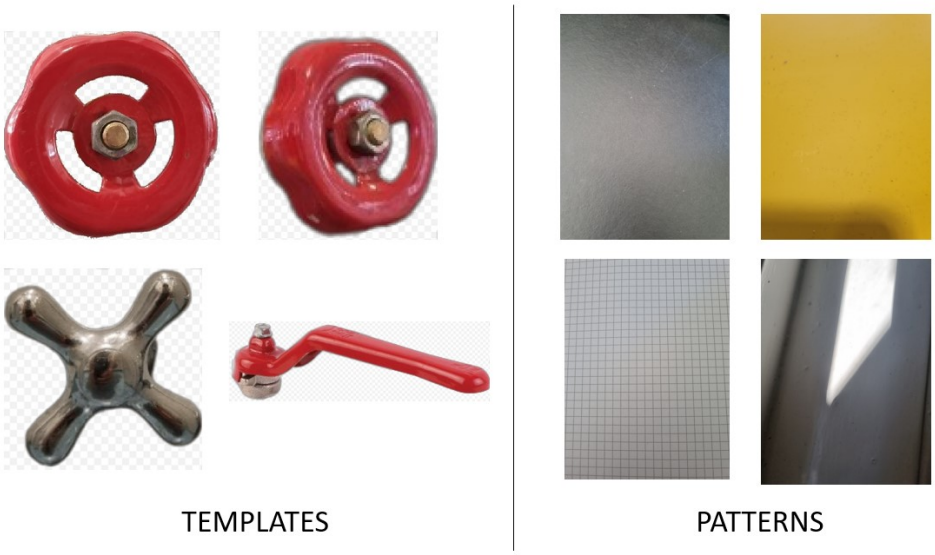

Figure 5.2: Some examples of templates and patterns

As already said, inside the TEMPLATE folder there are the images of all the objects to be recognized from the network, inside PATTERN all the backgrounds (Figure 5.2).

The *templates* can be .png or .jpg files.

- In the first case the image has a transparency or alpha channel used to define which parts of the image are visible, so the object, and which are transparent, so the background of the original photo to cancel. By setting the alpha channel to 0 for the background pixels, it is possible to place the image on top of the *patterns* without having original photo backgrounds on the created image of the dataset.
- In the second case is used a software (like Label Studio) [37] to manually indicate the contours of the desired object. The software creates a .txt file having all the x and y coordinates of the contour which must be put inside the LABEL folder. The pipeline 'reads' this file and discards the background during the dataset creation.

Figure 5.3 clarifies the explained logic.

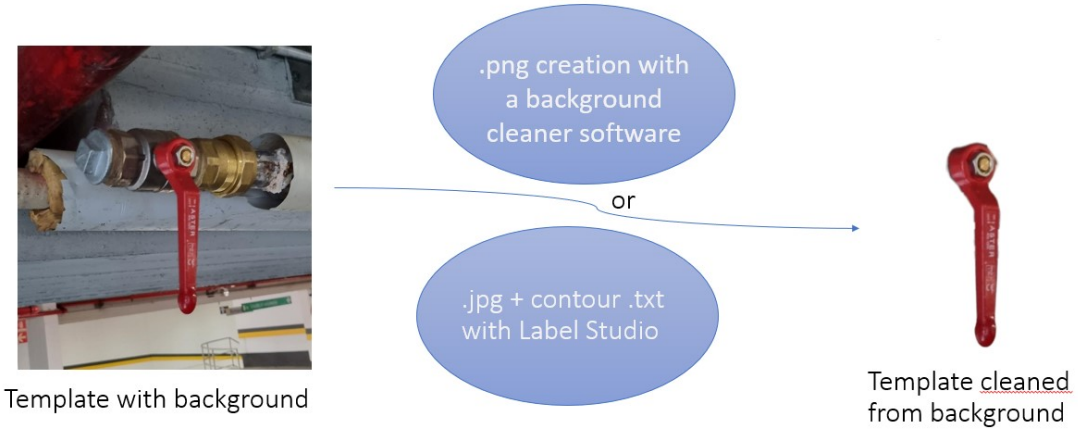

Figure 5.3: Templates cleaning before creation of the dataset

Running *generate_dataset.py* a dataset is created with the name, number of images and classes specified in the configuration file. The TEMPLATE folder must contain the same number of objects as the number of classes specified in the configuration file.
Different transformations are applied on the templates, which can be rotated, varied their dimension and effected by some noises, light and flash. Some of these effects are shown in Figure 5.4.

Figure 5.4: Effects applied on templates

All these transformations are integrated to have the possibility to create nearly infinite types of dataset and to make the more possible realistic the images.

The templates transformed are pasted over the patterns, in order to have a representation of possible real cases. Templates can be 'cutted' over a certain percentage by the edges of the image as in Figure 5.5.

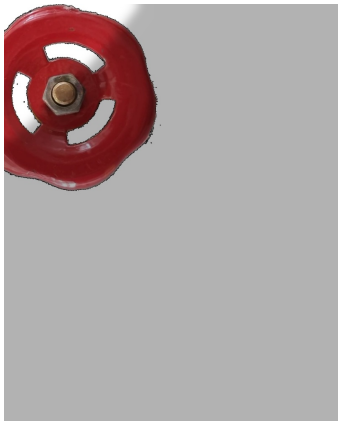

Figure 5.5: Cut on template

The code stores in a vector the pairs of x and y coordinates for each point along the contour of objects in the images. These are basically the segmentation labels, which are saved in the corresponding .txt file. If the dataset is a detection one, before creating the .txt label file the maximum and minimum x and y coordinates are taken so to extrapolate the detection label. Figure 5.6 better explains this logic.

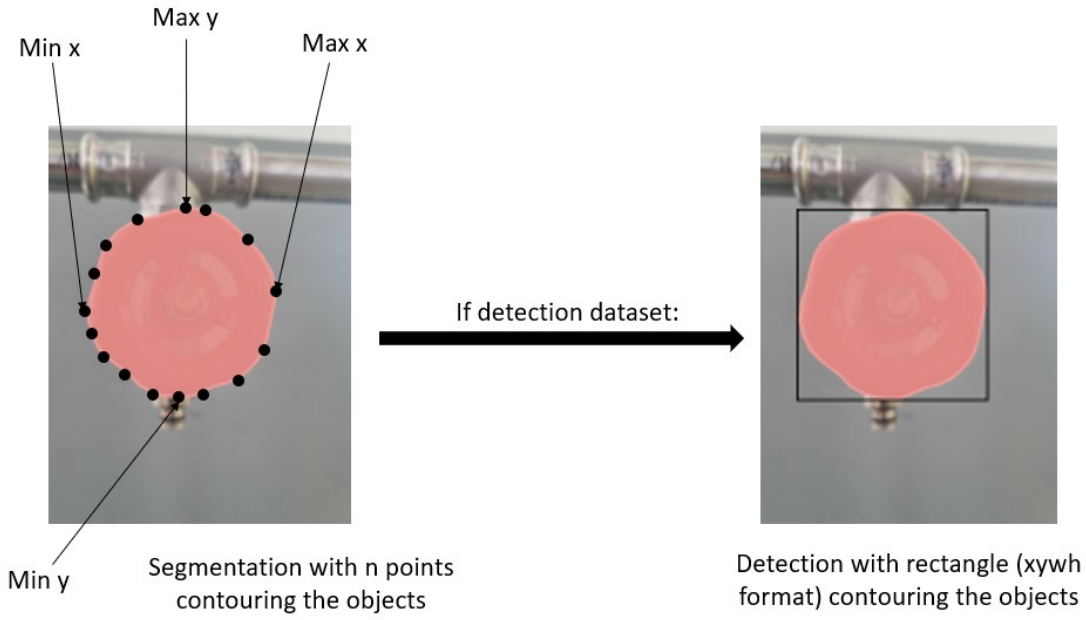

Figure 5.6: Segmentation to detection labels

The significant advantage of synthetic data of this type is that it does not require to be collected on field. To generate a dataset, all that's needed is one or more photo of the object to be recognized and some backgrounds. This results in high accessibility and the capability to generate quickly and easily large volumes of data. However it is convenient to mix these type of data with real data, manually labelled. Some images generated can have a *bias* due to the initial input, e. g. the template could have a low resolution or may not accurately represent reality. The pipeline could overlook certain limit cases, a transformation or noise that can be found in reality, but that is not implemented in the pipeline.
Despite these critical issues, the on-the-fly availability of these data, along with the possibility to create fictitious data which are difficult to find on Internet gives this type of data enormous importance and adaptability to real-world scenarios.

5.3 The training folder

The training folder contains *training.py* which is used to train a YOLOv8 model by Ultralytics on a dataset created previously and indicated in the *data.yaml* file. Useful parameters like number of epochs, model from which to start the training and typology of the task can be chosen modifying the associated $config\_Training.yaml$.

The Ultralytics *train*() function implements also the validation task, the results of training and validation, so weights, graphs and metrics are uploaded by this code in the *weights_and_validation* folder under another folder with a proper descriptive name.

5.4 The testing folder

The testing folder contains *testing.py* which is used to test the previously trained model. Giving in input the obtained weights, the script uploads a folder with various metrics calculated from the results of inference in the validation set.
In particular the script calculates the confusion matrix, precision-recall and F1 score curves and saves some predicted images.

Chapter 6

Autonomous manipulation with Spot: valve

Spot Arm gives the possibility to interact with the environment thanks to its robotic arm. Spot Enterprise is provided of PTZ and better quality cameras, which take measures and identify objects thanks to implemented computer vision algorithms.

This dualism is the base for the cooperation of the robots. An example is Spot Enterprise sent on a autonomous missions, taking measures on saved waypoints along the path, as the reading of the gauge of a manometer and the detection of valve leakage using DINO from a video taken with Spot CAM [38]. If some problems are detected Spot sends an alert to the operator and eventually triggers a mission for the Spot Arm to resolve the leakage.

In this chapter it is explained the mission where the Spot provided of robotic arm is able to close the valve and resolve the leakage. The code is composed of functions from the Spot SDK to perform the action and a neural network to identify the position of the valve [39].

Spot initially takes a photo of the environment through the camera inside the gripper of the robotic arm, if YOLO identifies the target Spot autonomously searches for grasping, plans the trajectory and performs the movement. To ensure the grasping procedure has been successfully the code prints on screen the gripper open percentage, instead to ensure the knob has rotated it prints the angle before and after the rotation. The algorithm repeats this action until this difference is equal to zero, so the arm tries to close the valve/knob, but the valve is already closed.

6.1 Code explanation

For now assume Spot is manually driven in front of the valve. The code starts with Spot powering on the motors and standing in front of the knob. Then the robot takes a photo from the camera inside the gripper of its robotic arm. This photo is sent as input to the machine learning algorithm, which is a YOLO model that will be explained in details on the next section. The output of the neural network is a set of predictions, usually one (if not one is taken the one with major confidence) which is assumed to be the valve. From this prediction is taken the bounding box in YOLO format (X_{center}, y_{center}, $width$, $height$).

All the subsequent mechanical actions performed by the robot to close the tap are completely managed by Spot SDK.
The next step is to grasp the valve starting from the YOLO output, from which is taken the position of the center of the valve. These values and some constraints are added to the SDK function $manipulation\_api\_pb2.PickObjectInImage()$, which allows to perform the action. In this case the need is to facilitate rotation, two constraint are added in the function:

- Fingertip variable can be set with values between 0, for (default) palm grasp, where the object is pressed against the gripper's palm plate, and 1.0 for fingertip grasp, where the robot tries to pick up the target with just the tip of its fingers. In the case of the valve in the 'Area42' laboratory this parameter is tuned to 0.6 in order to have an optimized grasp.
- To prevent the gripper from reaching the mechanical limit switch prematurely, its joint is rotated by +90 degrees along the x-axis of the gripper reference frame. This adjustment provides a larger rotational range.

Figure 6.1 explains these added constraints.

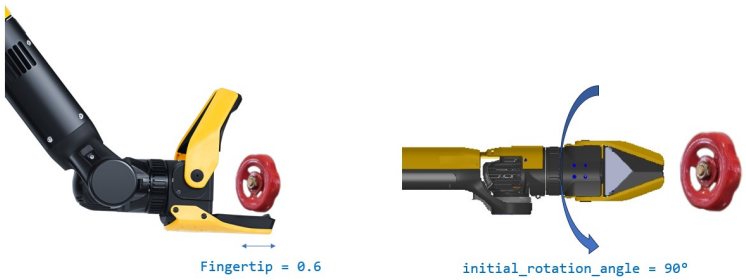

Figure 6.1: The 'turn valve' constraints

After successfully grasping the object, the next action to be executed is the rotation. The Spot SDK offers various functions for this purpose. Since the code repeats this action multiple times until the valve is closed, the critical factor to control is not the angle of rotation but the rotational force. The robot autonomously compensates gravitational effects, so the only force applied is a rotational torque along the x-axis of the gripper, finely tuned to 1.1 Newtons.
This rotation is performed by the wrist joint of the robotic arm, the success of this rotation is determined by the exact value of the wrist joint's rotation. To evaluate the outcome, the difference between the wrist joint's position before and after the rotation is calculated. Two threshold are defined:

- The difference exceeds a certain predefined threshold: it indicates that the grasp is unsuccessful, suggesting that the wrist does not encounter any resistance from the valve during the rotation.
- The difference lays between the two thresholds: it is assumed to be a successfully rotation.
- The difference falls below another predefined threshold: it suggests that the wrist encounters significant resistance from the valve during the rotation. If this lower threshold is met for two consecutive iterations, it is assumed that the valve is closed and the algorithm concludes its operation, powering off the robot.

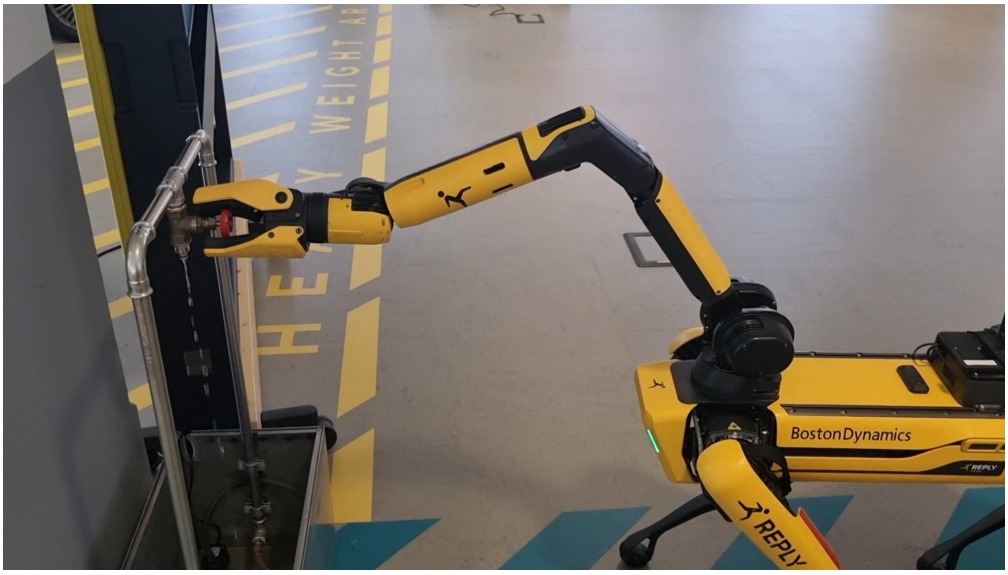

Figure 6.2: Spot performing the closure of the valve in Area42

In Figure 6.3 is shown a clarifying schema [40] of the action.

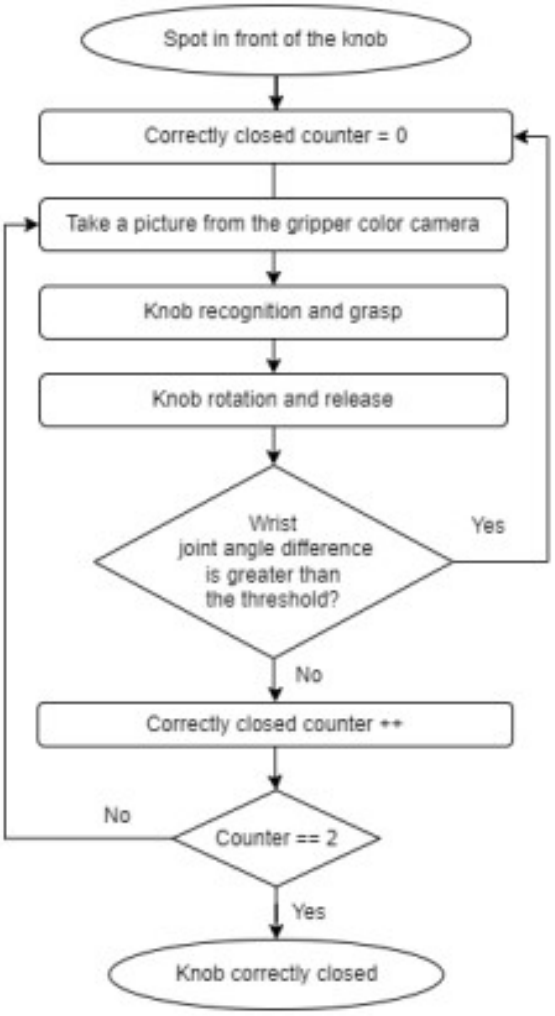

Figure 6.3: Logic of the 'turn knob' code

6.2 Creation of the autonomous mission - Testing

The main use case of Spot is patrolling in industrial plant, in order to substitute humans in repetitive inspections, which could lead to errors because of the lack of attention.
In this context it is important the creation of an autonomous '*mission*', which is a pre-recorded path with saved reference locations (waypoints) where Spot performs a proper action. The '*action*' is a particular task, it can be already integrated as default on Spot, like 'take a photo' with Spot CAM or assume a particular 'body pose', but also some callback functions, so actions created ad hoc by the user for a certain use cases, like the closure of the valve.

Two steps can be involved in the setup of the mission:

- Recording of the mission path through a manual guide of the robot with the tablet. This is done using the Autowalk feature in the app menu shown in Figure 6.4, starting from a '*fiducial*' which is a sort of QR code (Figure 6.4) from which Spot is able to localize and start the mission [41].

  After the recording Spot saves a map, structured as a topological graph of waypoints and edges. Spot tries to repeat exactly the same saved path; however, it is able to apply slight variations to the configured path if it detects obstacles in the path. The mission fails instead and the robot comes back to starting waypoint if there are too many obstacles and it is not able to reach a saved waypoint where it has to perform an action.

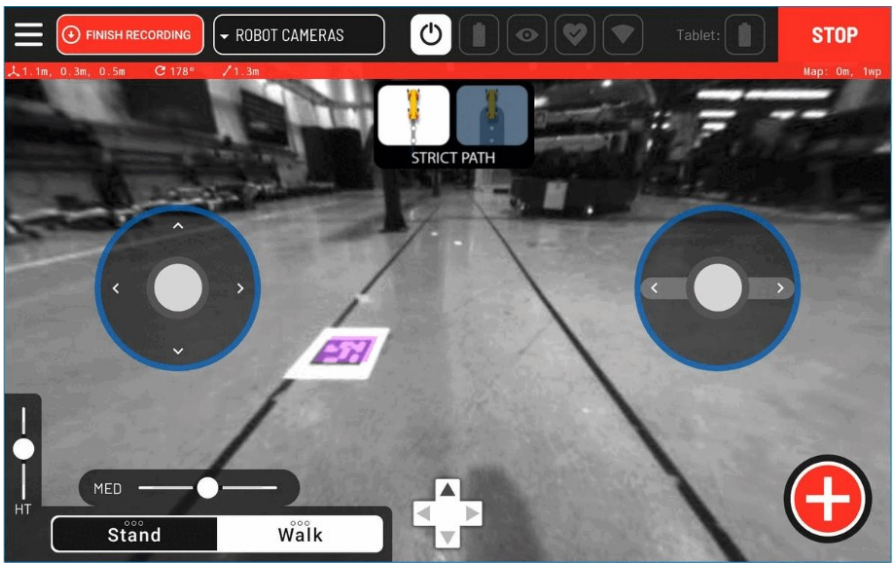

Figure 6.4: Tablet interface for autonomous mission recording with Spot

- In case the default Spot actions are not enough for the use case, it is possible to add callbacks, particular custom actions, as the closure of the valve. Downloading data from Spot tablet, there is a .yaml file where it is possible to add some callbacks in each saved waypoint.

Before the establishment of the mission it is essential to understand the waypoint where to locate the action of valve closure, for this reason some tests (Figure 6.5) are conducted positioning Spot in different locations near the valve in order to understand from which distances and from which angles Spot is able to close successfully the valve.

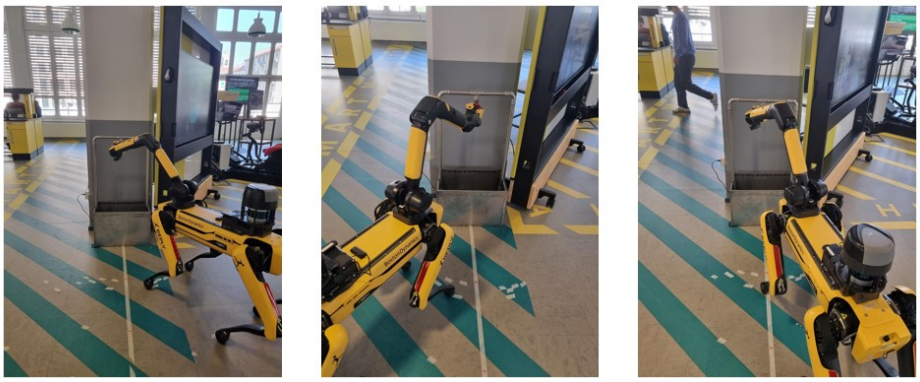

Figure 6.5: Test of the valve closure script

The computer vision algorithm is able to recognise the valve from different distances (max 1.5 meters from the valve) and angles (also 80° with reference to the valve vertical). Explanations of the detection algorithm performance and metrics can be found on the following section.
The main limitation is the operation range of the "grasp()" function in the Spot SDK. Once the valve is identified by the computer vision algorithm, the function allows Spot to walk to and grasp the valve starting from different distances and angles, wrt the valve. The maximum distance is the same of the computer vision algorithm, so 1.5 m, the angle limit range is instead -30°/+30° with reference to the vertical of the valve. As Figure 6.4 shows the action is thought so that the gripper reaches the valve vertically, in order to have a better grasp and to easily rotate the valve, the proper action is guarantted only with these limitations.
In order to add this action on a autonomous mission, a proper waypoint must be chosen. A easy location for its execution is a waypoint in front of the valve, distant 80 cm from it. When the mission is repeated Spot performs the action starting always from that waypoint. However it is necessary to use the computer vision algorithm because the waypoint is not always fixed, there can be errors and uncertainties in the position of the robot.

6.3 Training of neural network process

The code needs to be integrated with a computer vision algorithm allowing Spot to recognise the valve. To perform this task is used the pipeline explainec in Chapter 5. The generation of the images is made starting from three photos of the valve, taken from different angles and put inside the TEMPLATE folder. This dataset contains only one class "area42_knob" and 5000 images. It is divided in training and validation set. For the test set 65 photos of the valve in laboratory are taken and manually labelled with Label Studio in order to ensure that the obtained weights with the synthetic dataset has good inference capabilities also in real-world scenarios.

Figure 6.6: Synthetic dataset of "Area42" valve

The training is done using YOLOv8n model from Ultralytics so to have faster inference time and a smaller number of parameters network so a computationally lighter algorithm. The training is completely done on a Intel Core i7 CPU with 16 GB RAM for 30 epochs, which is a good compromise considering the limited computational resources and the large amount of time needed to process the training. In Figure 6.7 are shown the performance analyzed during this training process.

In Figure 6.8 are shown the plot of the metrics trend. Although Ultralytics hasn't published an official paper to explain each Loss used, it is possible to make some reasoning along the obtained results.

epoch,	train/box_loss,	train/cls_loss,	train/dfl_loss,	metrics/precision(B),	metrics/recall(B),	metrics/mAP50(B),	metrics/mAP50-95(B),	val/box_loss,	val/cls_loss,	val/dfl_loss,	lr/pg0,	lr/pg1,	lr/pg2
1,	2.5894,	2.9367,	3.0802,	0.9571,	0.928,	0.93599,	0.66142,	1.2561,	1.5295,	1.6888,	0.000664,	0.000664,	0.000664
2,	1.1408,	1.0273,	1.5168,	0.99224,	0.994,	0.99438,	0.81849,	0.7079,	0.86866,	1.1146,	0.0010672,	0.0010672,	0.0010672
3,	0.82152,	0.67401,	1.232,	0.99578,	0.996,	0.995,	0.90742,	0.48776,	0.45631,	0.9448,	0.0012064,	0.0012064,	0.0012064
4,	0.68853,	0.53012,	1.1206,	0.99582,	1,	0.995,	0.90932,	0.4413,	0.40927,	0.90054,	0.000812,	0.000812,	0.000812
5,	0.62308,	0.46293,	1.0777,	1,	0.99527,	0.995,	0.92125,	0.41442,	0.32801,	0.88615,	0.000812,	0.000812,	0.000812
.....													
26,	0.37081,	0.26233,	0.91799,	0.99897,	1,	0.995,	0.98005,	0.25874,	0.20128,	0.80998,	0.000664,	0.000664,	0.000664
27,	0.40292,	0.28612,	0.93101,	0.998,	0.99987,	0.995,	0.93947,	0.34839,	0.26006,	0.83882,	0.0010672,	0.0010672	0.0010672
28,	0.40283,	0.28712,	0.93095,	0.99985,	1,	0.995,	0.98637,	0.26783,	0.23026,	0.81735,	0.0012064,	0.0012064,	0.0012064
29,	0.37439,	0.26765,	0.91566,	0.99989,	1,	0.995,	0.96988,	0.28356,	0.21351,	0.80926,	0.000812,	0.000812,	0.000812
30,	0.37439,	0.26765,	0.91566,	0.99989,	1,	0.995,	0.96988,	0.28356,	0.21351,	0.80926,	0.000812,	0.000812,	0.000813

Figure 6.7: Training performance

The YOLOv8 loss is the combination of three different losses which graphs are shown in the Figure 6.8 [42].

- bounding box loss: measures how "tight" the predicted bounding boxes are to the ground truth object. Usually it is implemented using regression loss, L1, smoothL1. These losses give the error between the predicted and ground truth bounding box: nearest is the value of the function to zero, higher is the network capability to predict correct and precise bounding boxes. The gain in the total loss function is 7.5.

- classification loss: quantifies how good the network predicts the class of the object in each detected bounding box, so measures how the model adapts to training data labels. The considered classes are two in this implementation: "area42_knob" and "background". Generally it is implemented with a cross-entropy loss, the function:

$$\mathcal{L}_{cls} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(p_{i,c})$$

  with N n° of dataset images, C n° of classes, y_i, c binary variable and p_i, c predicted probability indicating if i example belong to class c. The objective is to minimize this difference, so it should tend to zero to have good classification performance. The gain in the total loss function is 0.5.

- distribution focal loss: is particularly useful when the distribution of the classes is very imbalanced, which is not properly this case because the dataset has only one class. This loss gives more importance to the rare classes (with less images on the dataset) during training, making the model pay more attention to them. Basically dfl allows the model to make more balanced predictions in dataset which are not. The gain in the total loss function is 1.5.

Training losses are promisingly, all having a decreasing behaviour, however plateau seems not to be reached already, probably some other epochs of training can be done, but the performance are already good for the use case and the possibility to reach overfitting must be avoided.

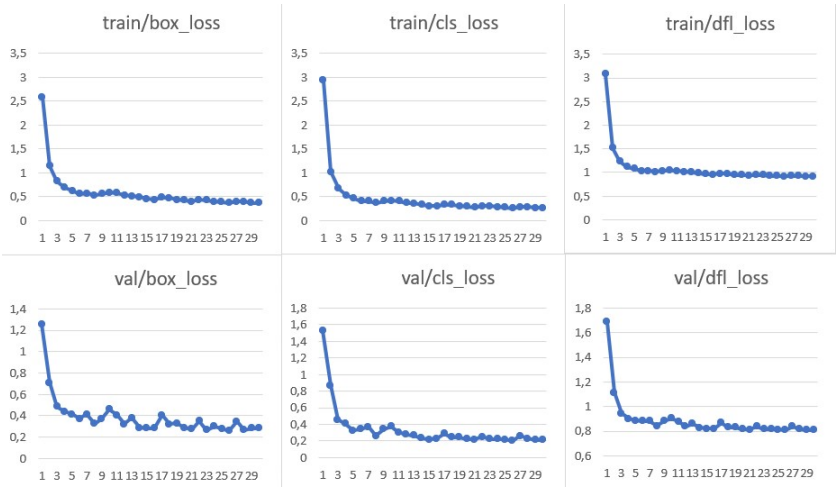

Figure 6.8: Training and validation set losses

In Figure 6.8 are also shown the performance obtained with the validation set, important to underline is that the set of images is still synthetic but different from the one used during the training process. Also in this case the loss values decrease quickly, reaching promising values.

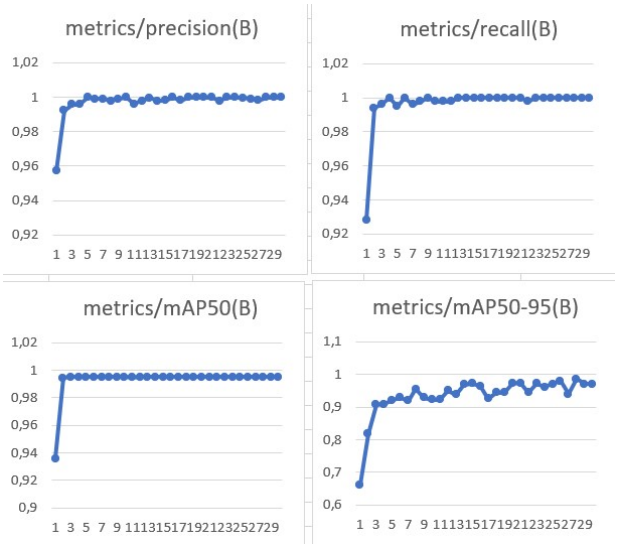

Figure 6.9: Precision, recall and mean average precision values

In Figure 6.9 it is possible to see the values obtained for precision, recall and mean average precision during the training. In particular mAP50 calculates the precision putting a threshold of IoU equal to 0.5, instead mAP50-95 uses different threshold of IoU, starting from 0.5 and arriving to 0.95 with a step of 0.05 (so 0.5, 0.55, 0.6, 0.65, ...), giving a more detailed evaluation of the model.
All these metrics reach values near to 1, certifying good performance of the model.

The testing process is instead done over a set of 65 photos taken from real world cases, covering different angles and backgrounds with which Spot could interact.

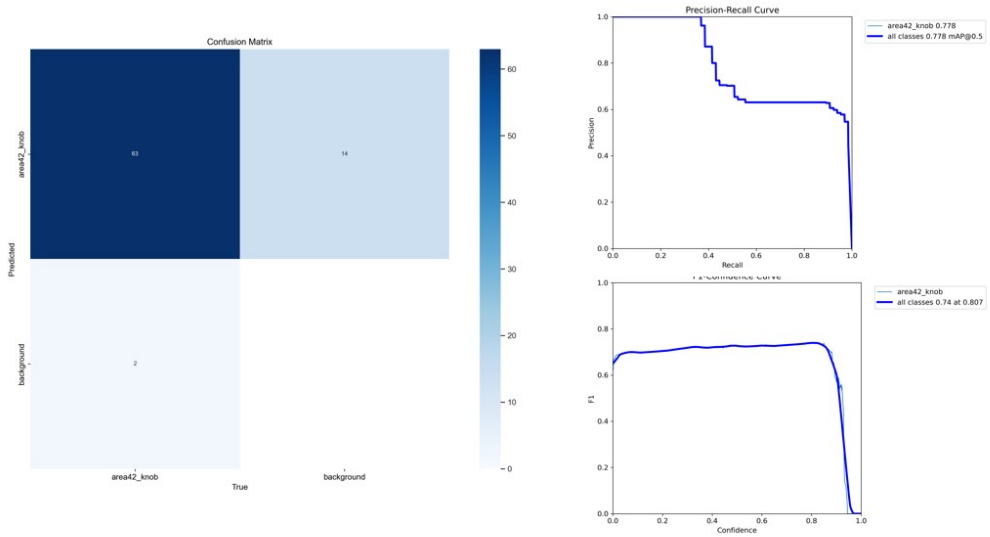

Figure 6.10: Confusion matrix, PR-confidence and F1-confidence graphs

Figure 6.10 shows on the left the confusion matrix, even if the neural network predicts only one class, the background is 'treated' as a class. This is useful to point out that there are some false positive, some background objects that are predicted by the network as 'area42 valves'. The attempt to resolve this issue will be made in the next section, which discusses fine-tuning on the model.
In the same Figure 6.10 are shown on the right the precision recall graph, which is a function of the model's confidence threshold. The same thing is the F1 score graph that indicates the variation of the F1 score on the base of the threshold. As already explained in Chapter 2, higher values of confidence threshold involves a minor number of predicted classes, slower values instead a major number of prediction, affecting the values of precision, recall and mAP. Using this two graph is possible to tune a good value of this threshold in order to maximize the model performance, in this case a confidence threshold of 0.6-0.8 is a good compromise.
In Figure 6.11 there are the result of inference on the test set, the image on the

bottom center is an example of the problem of false positive highlighted with the confusion matrix.

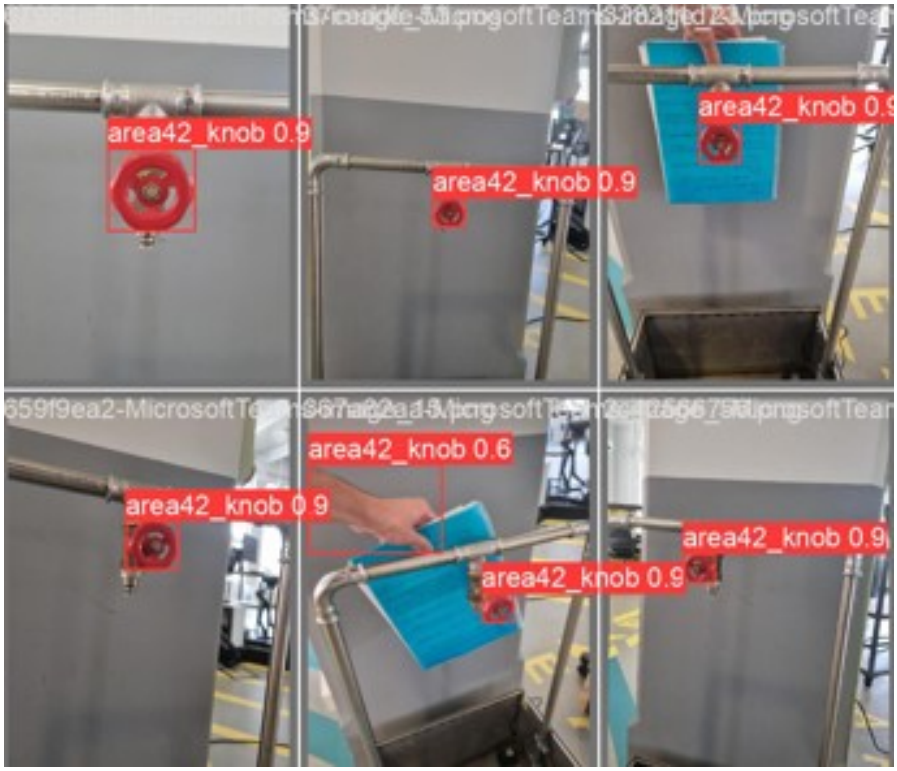

Figure 6.11: Prediction obtained on the test set

In Figure 6.12 are shown some predictions obtained during the execution of the mission. As expected the neural network is able to correctly estimate the position of the valve from images taken from different distances and angles.

Figure 6.12: Photos taken by Spot robotic arm during the script execution

6.4 Fine-tuning process

Fine-tuning consists in the re-training of a already trained model on new data. This technique is applicable on the entire network or only in same layers that are not 'frozen', so their weights aren't kept invariant [10].

A fine-tuning process is applied to improve the results obtained with the training process of the synthetic dataset of valves. Some limitations like the discrete number of false positives, required further refinement to improve model performance.

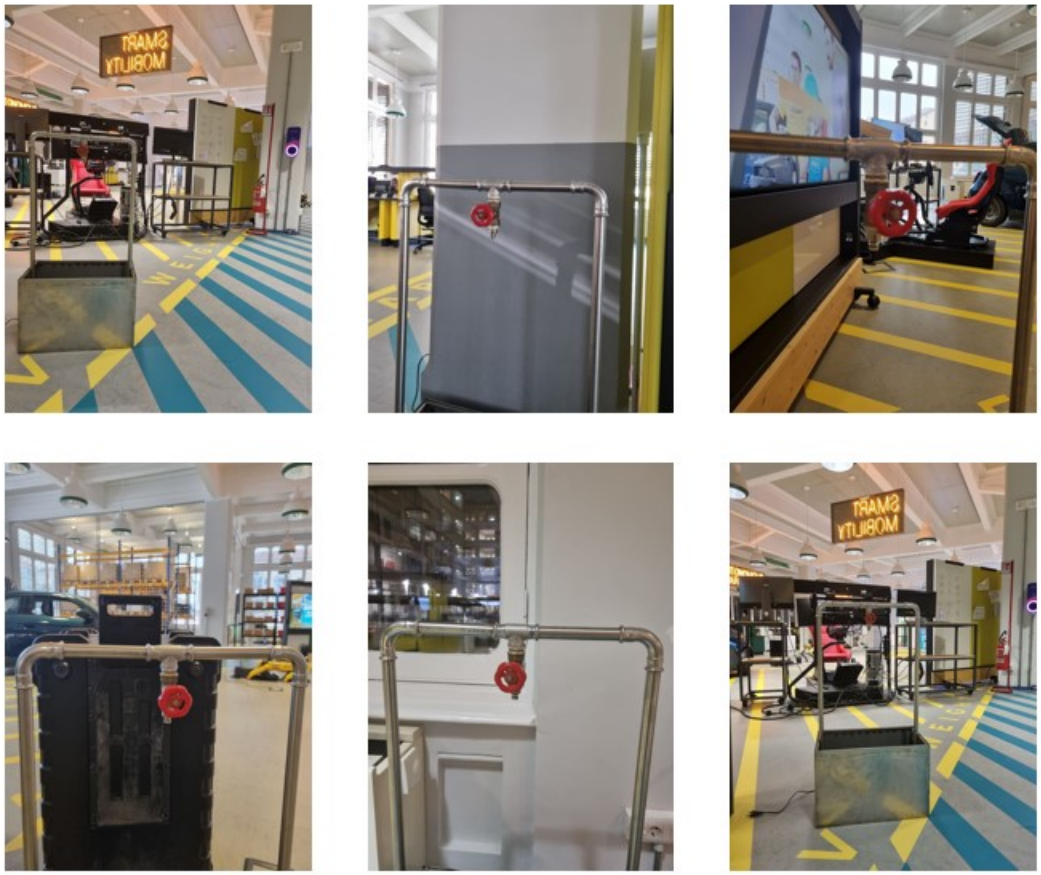

Figure 6.13: Images of valves from Area42 laboratory

In the laboratory 700 images of the valve are taken and manually labelled using 'Label Studio' tool. The images are divided into 600 images for the training set and 100 for the validation set. The test set is the same of the previous dataset of 65 real images.
Starting from the weights obtained with the training of the synthetic dataset, 10

epochs of training are executed. In Figure 6.14 the values of the training and validation losses decrease and reach a plateau.

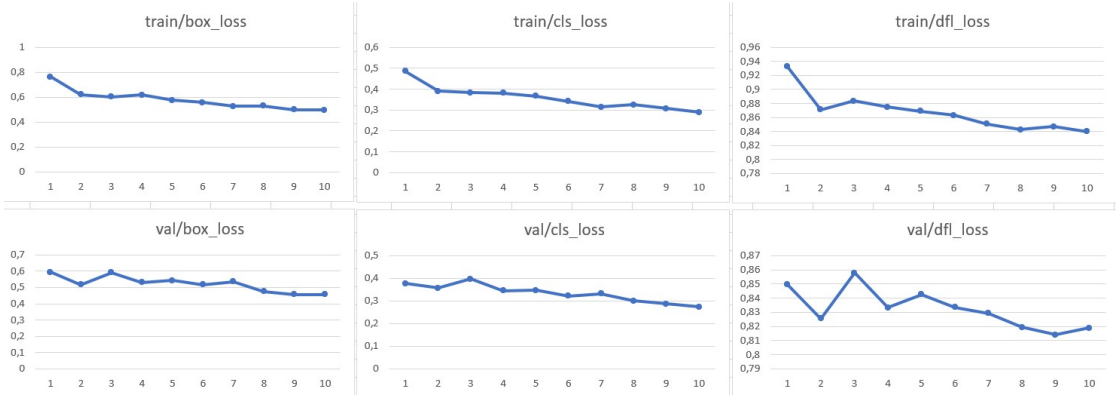

Figure 6.14: Fine-tuning training metrics

In Figure 6.15 is possible to see values of precision, recall and mean average precision with a good growing trend near to 1.

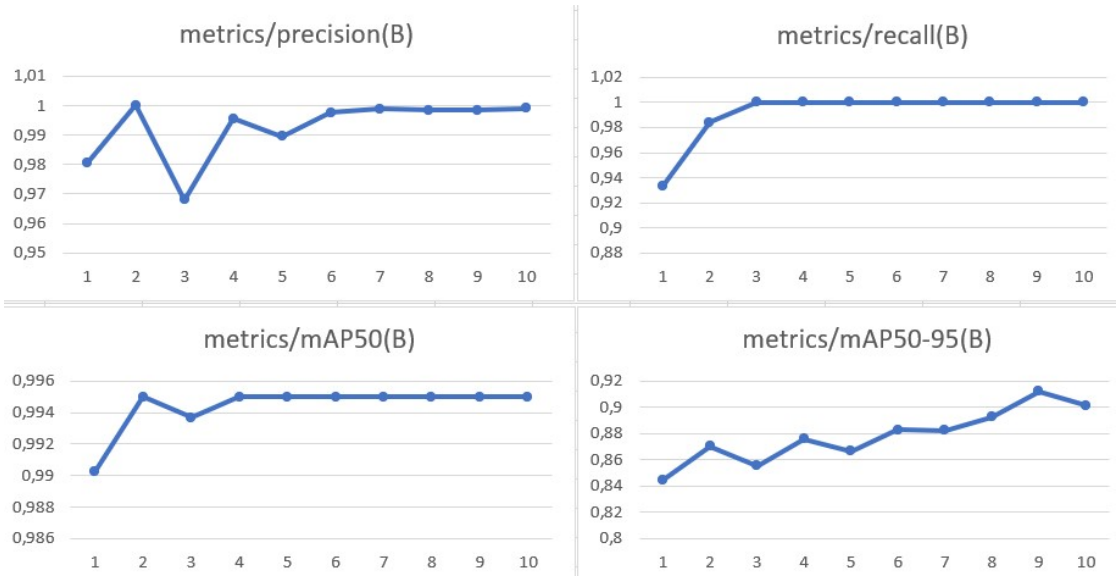

Figure 6.15: Fine-tuning precision, recall and mAP metrics

An essential step is to test the network, the same 65 images taken in the laboratory and manually labelled are used. In order to compare the values of precision, recall and mean average precision it is necessary to choose a fixed value of confidence threshold and to use the same for the testing of the model before and after this last training, since varying this value also the metrics vary with it. In this case it

is used the default value of 0.001 of Ultralytics YOLO, causing a greater value of recall with respect to precision.
The result obtained with the test set are shown in the table below:

Training	Precision	Recall	mAP50	mAP50-95
30 epochs (no fine-tuning)	0.626	0.908	0.778	0.658
40 epochs (fine-tuning with real images)	0.669	1	0.803	0.754

Table 6.1: Test set results

The table also shows an higher value of precision. Moreover, Figure 6.16 has on the left the confusion matrix, differently from the previous training this time there are almost no more false positive. Fine-tuning has contributed to these substantial improvements on the prediction performance of the network.
Also mAP50 and mAP50-95 state an improvement of the network inference capabilities.

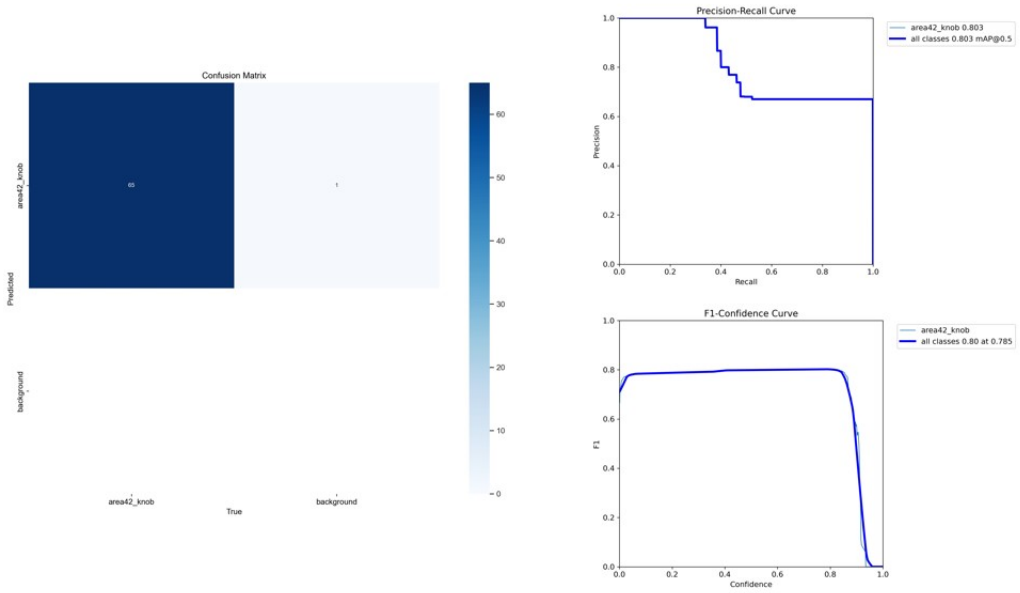

Figure 6.16: Fine-tuning validation metrics

On the right of Figure 6.16 there are the precision-recall curve and F1score graph, a good value for the confidence threshold to maximise the metrics is 0.6-0.8 as in the previous case.

Figure 6.17 shows some results of inference from the test set, the second image is an example of the sensitivity of the model, giving a minor confidence to a red seat behind the red valve.

Figure 6.17: Fine-tuning inference image

6.5 Turn valve results

The algorithm is tested over a total of 150 experiments, each performing the valve recognition and rotation (for at least 3 times to ensure the valve is closed). Statistical conclusions can be taken:

N° of experiments	**Success**	**AI failure**	**SDK failure**
>150x3	85%	5%	10%

Table 6.2: Turn valve experiments

Failure causes:

- AI failure: in this case the neural network does not find the valve, maybe cause it is too far or because of illumination on the camera of the arm which causes problem to image resolution and so to the computer vision algorithm. Usually the case where two valve are detected even if there is only one does not cause failures because only the prediction with major confidence is taken, that is almost always the correct valve position.
- SDK failure: this problem indicates that the arm approaches the valve from a wrong direction, for example, from the top instead of from the front. This is a inner problem of the BD SDK that, unless the function receives the correct coordinates of the valve, computes a inaccurate movement to reach it [43].

Chapter 7

Conclusion and future research

Boston Dynamics©' robot 'Spot' is one of the most advanced robotic platform nowadays. The possibility to control all its components, from the movements to the large number of cameras streaming, makes it suitable for computer vision research. The objective of this thesis was to integrate Spot with the capability to autonomously navigate in industrial plants and co-operate with humans. Spot is able to substitute humans in repetitive inspections, which could lead to errors because of the lack of attention. Moreover, it is able to resolve daily leakages or to execute simple tasks, such as the closure of a valve or the pick-and-place of objects from an area to another of the plant. The possibility to mount different sensors on its back makes it suitable for a wide number of tasks, from industrial patrolling to human-robot cooperation.

The first effort is directed in giving the robot the possibility to have auto comprehension of the environment. Using the three proposed neural networks Spot is able to recognise faces and anonymize them to preserve privacy, to recognise the 80 classes of the MS-COCO dataset and to identify exit signals and fire extinguishers.

The second objective is the end-to-end creation of the 'closing valve mission'. Firstly, the creation of a pipeline to generate a synthetic dataset of valves and the training of a neural network used to detect them. Then, the utilization of proper functions from Spot SDK to approach and grasp the valve. Finally, the application of a smart logic allowing the robot to overpass its physical limits (joints end of run) and autonomously understand whether the valve is closed after n rotations.

Further researches can be carried out, utilizing the pipeline to create a dataset of lever valves (as the one at the right bottom of Figure 5.2), providing Spot with directional detection, allowing it to recognise the direction from which to grasp the valve and implementing the proper rotation to close it in case of leakage.

An imminent next step is the activation of Spot Arm for the 'closing valve' mission when Spot Enterprise detects a leakage. This process is orchestrated through cloud. When Spot Enterprise during its missions identifies the valve leakage, sends an alert to a cloud service which then sends a message to Spot Arm, triggering the mission of closure. The Spot Arm mission will be managed as a ROS (robot operating system) 'service'.

Integrating additional AI capabilities beyond computer vision represents an wise strategy to extend the horizons of Spot's functionalities. A future project is the integration of a LLM (large language modeling) allowing natural language interaction with Spot. Analysing an input text the LLM is able to translate the various robot skills, as 'navigate to a waypoint' or 'perform this specific action', in simple natural language commands. The logic is the same of the closing valve project: a previously recorded map with saved waypoints and a computer vision algorithm to give Spot comprehension and autonomous decision making. Associating a name to each waypoint it would became possible to send Spot on a specific location or to perform a specific task. Saying: "Hey Spot, are you ready to solve the leakage? Please go in front of the dripping valve and close it", the mission would automatically start.

Acknowledgements

Desidero ringraziare il relatore di questa tesi, il Professor Fabrizio Lamberti. Le sue stimolanti lezioni hanno suscitato il mio interesse per il mondo del machine learning e sono grato per l'opportunità che mi ha concesso di lavorare questi mesi presso il laboratorio 'Area42' di Reply. Desidero inoltre ringraziare il co-relatore della tesi, il Dottor Oscar Pistamiglio, che ho avuto il privilegio di conoscere e apprezzare durante la mia esperienza presso il laboratorio. Un grazie particolare va ai miei tutor aziendali, Mattia e Giacomo, per i loro preziosi consigli. Vorrei anche esprimere la mia gratitudine a tutti i colleghi di Sprint Reply e a Silvia.

Ringrazio la mia famiglia, mamma per il suo sostegno e amore incondizionato, papà, per la fiducia sempre dimostratami e i sorrisi. Ringrazio i miei nonni, Anna Maria e Antonio, spero di rendervi fieri, Luciana e Salvatore, le stelle che mi guidano.

Ringrazio tutti gli zii, cugini, Vittoria, Matteo e Sergio.

Ringrazio gli amici tutti, da chi conosco da 20 anni a chi solo da un mese. Tutti voi qui presenti, per il sostegno, le serate, i sorrisi, vi voglio bene.

Grazie Lorenzo, sei fonte di ispirazione e stimolo, grazie Federica, Alice, Davide, Tommaso, Andrea e Gabriele siete sostegno e ascolto, grazie Martina, Lorenzo e Marco per i bei momenti passati insieme. Grazie Elisa, Stefano e Kevin, siamo di natura molti affini, e grazie Tiziano, Fabrizio, Stefano, Davide e Simone per i mesi spensierati trascorsi e le risate.

Grazie a tutti per aver reso questa esperienza formativa così arricchente e stimolante e per avermi sostenuto.

Bibliography

[1] *Spot characteristics*. URL: https://bostondynamics.com/products/spot/ (cit. on pp. 1, 2).

[2] *Spot CAM+ characteristics*. 2023. URL: https://support.bostondynamics.com/s/article/Spot-CAM-Spot-CAM-Spot-CAM-IR (cit. on p. 3).

[3] *Spot EAP characteristics*. 2022. URL: https://support.bostondynamics.com/s/article/Spot-Enhanced-Autonomy-Package-EAP (cit. on p. 3).

[4] *Spot Arm characteristics*. URL: https://bostondynamics.com/products/spot/arm/ (cit. on p. 4).

[5] *Spot connectivity*. 2022. URL: https://support.bostondynamics.com/s/article/Spot-network-setup (cit. on p. 5).

[6] *Spot controller*. 2022. URL: https://support.bostondynamics.com/s/article/Spot-controller (cit. on p. 5).

[7] *Spot charger*. 2023. URL: https://support.bostondynamics.com/s/article/Introduction-to-the-Spot-battery-and-charger (cit. on p. 6).

[8] *Spot specifications*. 2022. URL: https://support.bostondynamics.com/s/article/Robot-specifications (cit. on pp. 7, 39).

[9] *What is machine learning??* 2017. URL: https://theconversation.com/what-is-machine-learning-76759#:~:text=In%201959%2C%20Arthur%20Samuel%2C%20a,or%20to%20make%20accurate%20predictions. (cit. on p. 11).

[10] Fabrizio Lamberti. «Machine learning for vision and multimedia». In: (2022) (cit. on pp. 11, 13, 15, 16, 26, 27, 29, 33, 63).

[11] *What is Computer Vision?* 2022. URL: https://www.ibm.com/topics/computer-vision (cit. on p. 11).

[12] *What is image classification? Basics you need to know*. 2023. URL: https://www.superannotate.com/blog/image-classification-basics (cit. on p. 14).

[13] *Haar Cascade Classifier vs Histogram of Oriented Gradients(HOG).* 2020. URL: https://medium.com/@goutam0157/haar-cascade-classifier-vs-histogram-of-oriented-gradients-hog-6f4373ca239b (cit. on p. 14).

[14] Pierre Sermanet David Eigen Xiang Zhang Michael Mathieu Rob Fergus Yann LeCun. «OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks». In: (Feb. 2014) (cit. on p. 16).

[15] Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik. «Rich feature hierarchies for accurate object detection and semantic segmentation». In: (2014) (cit. on p. 17).

[16] Ross Girshick Jeff Donahue Student Member IEEE Trevor Darrell Member IEEE and Jitendra Malik Fellow IEEE. «Region-based Convolutional Networks for Accurate Object Detection and Segmentation». In: (2015) (cit. on p. 17).

[17] Shaoqing Ren Kaiming He Ross Girshick and Jian Sun. «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks». In: (Jan. 2016) (cit. on p. 17).

[18] Joseph Redmon Santosh Divvala Ross Girshick Ali Farhadi. «You Only Look Once: Unified, Real-Time Object Detection». In: (May 2016) (cit. on p. 18).

[19] *What is Mean Average Precision (mAP), how to calculate it, and why is it important for evaluating models' performance?* URL: https://www.v7labs.com/blog/mean-average-precision#precision-recall-curve-breakdown (cit. on p. 20).

[20] *What is Average Precision in Object Detection Localization Algorithms and how to calculate it?* URL: https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b (cit. on p. 21).

[21] *What is gradient descent?* URL: https://www.ibm.com/topics/gradient-descent (cit. on p. 25).

[22] *What is underfitting?* URL: https://www.ibm.com/it-it/topics/underfitting (cit. on p. 26).

[23] *What is overfitting?* URL: https://www.ibm.com/it-it/topics/overfitting (cit. on p. 27).

[24] *OpenCV python library.* URL: https://opencv.org/ (cit. on pp. 29, 30).

[25] *RetinaFace GitHub repository.* URL: https://github.com/serengil/retinaface (cit. on p. 29).

[26] Jiankang Deng Jia Guo Evangelos Ververas Irene Kotsia Stefanos Zafeiriou. «RetinaFace: Single-shot Multi-level Face Localisation in the Wild». In: (2020) (cit. on pp. 29, 30).

[27] Marco Tulio Ribeiro Sameer Singh Carlos Guestrin. «Anchors: High-Precision Model-Agnostic Explanations». In: (2018) (cit. on p. 30).

[28] *Coco dataset*. URL: https://cocodataset.org/#home (cit. on p. 32).

[29] *Ultralytics official site*. URL: https://docs.ultralytics.com/ (cit. on p. 32).

[30] *Ultralytics YOLOv5 model structure*. URL: https://github.com/ultralytics/ultralytics/issues/189 (cit. on p. 33).

[31] *Ultralytics GitHub repository*. URL: https://github.com/ultralytics/ultralytics (cit. on pp. 33, 36).

[32] *docTR GitHub repository*. URL: https://github.com/mindee/doctr (cit. on p. 34).

[33] *docTR official site*. URL: https://www.mindee.com/product/doctr (cit. on p. 35).

[34] *Boston Dynamics' Spot SDK documentation*. URL: https://dev.bostondynamics.com/ (cit. on pp. 41, 42).

[35] *REST*. URL: https://www.w3.org/2001/sw/wiki/REST (cit. on p. 41).

[36] *Spot CORE characteristics*. 2022. URL: https://support.bostondynamics.com/s/article/Spot-CORE-payload-reference (cit. on p. 42).

[37] *Label Studio software*. URL: https://labelstud.io/ (cit. on p. 48).

[38] Mathilde Caron Hugo Touvron Ishan Misra1 Herve Jegou Julien Mairal Piotr Bojanowski Armand Joulin. «Emerging Properties in Self-Supervised Vision Transformers». In: (May 2021) (cit. on p. 52).

[39] *Boston Dynamics' Spot SDK GitHub repository*. URL: https://github.com/boston-dynamics/spot-sdk/blob/master/docs/python/understanding_spot_programming.md (cit. on p. 52).

[40] Tiziano Allara. *Boston Dynamics SPOT and Microsoft Teams to automate inspections*. 2023 (cit. on p. 55).

[41] *Spot Autowalk mission*. URL: https://support.bostondynamics.com/s/article/Getting-Started-with-Autowalk (cit. on p. 56).

[42] *Ultralytics YOLOv8 losses*. URL: https://docs.ultralytics.com/modes/val/ (cit. on p. 59).

[43] *Computer vision problems*. URL: https://www.chooch.com/blog/5-common-problems-with-computer-vision-and-their-solutions/ (cit. on p. 67).