Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in control systems engineering

Camera simulation for YoloV5 training and optimization

Master Candidate

Michele Patero

Student ID 205321

Supervisor

Prof. Carli Ruggero

University of Padova

Academic Year
2023/2024

To my mother
to my father
and to people I love

Abstract

Nowadays, one of the biggest challenges in machine learning and artificial intelligence is the acquisition of labeled data since it usually is a heavy time and money consuming task. The aim of this document is to propose a solution that tries to automatize this process by adapting an already existing robot simulator's rendering software in order to produce synthetic images. Those images will then be used to train the YoloV5 algorithm and it will be verified how well the algorithm performs onto real new images.

Oggigiorno, una delle sfide più grandi nell'ambito del machine learning e dell'intelligenza artificiale è l'acquisizione di dati etichettati, in quanto è usualmente un compito molto impattante in termini di tempo e costi. L'obiettivo di questo elaborato è quello di proporre una soluzione che cerca di automatizzare questo processo adattando un già esistente software di rendering di simulazione robotica per la creazione di immagini sintetiche. Queste immagini verranno poi utilizzate per allenare l'algoritmo YoloV5 e sarà verificata la qualità delle prestazioni su nuove immagini reali.

Contents

List of Figures

List of Tables

List of Code Snippets

List of Acronyms

CSV	Comma Separated Values
NN	Neural Network
RL	Reinforcement Learning
NMS	Non Maximum Suppression
CNN	Convolutional Neural Network
HSV	Hue Saturation Value
UI	User Interface
MVVM	Model View Viewmodel
BB	Boundig Box
ONNX	Open Neural Network Exchange
PoE	Power over Ethernet

1 Introduction

1.1 Brief machine learning historical background

Throughout history, humans have tirelessly endeavored to automate tasks, ranging from the simplest to the most intricate. Over centuries, technological advancements have been devised to streamline and enhance manual labor, progressively automating increasingly complex tasks. As intellectual curiosity grew, there emerged a fascination with creating machines that could emulate the behavior of living beings. An illustrative example is Jacques de Vaucanson's Canard Digérateur, crafted in 1764, a duck-shaped automaton capable of consuming kernels of grain. In the 1950s, scientific interest shifted towards the prospect of computers learning from data. This led to the proposal of algorithms such as early forms of Neural Networks, genetic algorithms, and the renowned perceptron by F. Rosenblatt in 1957. However, due to technical constraints, Neural Networks were temporarily set aside until 1981 when Werbos introduced the Multi-Layer Perceptron with a specific Neural Network Backpropagation technique. In 1995, Support Vector Machines, a pivotal algorithm in Machine Learning, were put forth by Vapnik and Cortes. Presently, with the remarkable advancements in modern computer capabilities, Neural Networks have become the most popular, extensive, and extensively studied category of learning algorithms. Despite this, Support Vector Machines continue to be widely utilized owing to their simplicity, showcasing the ongoing coexistence and relevance of various approaches in the ever-evolving field of automation and machine learning.

1.2 The lack of data problem in Machine Learning

The machine learning field can be categorized into three main classes:

- Supervised learning: the algorithm is trained on a labeled dataset consisting of input-output pairs. The main objective is to find an input-output relation in order to make accurate prediction on new, unseen data. In other words, a set of examples where the correct answers is known is shown to the algorithm. It has to learn and generalize from these examples to make predictions on new, similar instances.
- Unsupervised learning: the algorithm is given input data without output labels. The objective is to find relationships between the data. Unlike supervised learning, there is no predefined target variable for the algorithm to predict but the aim is to group the input data based on their similarities. Unsupervised learning is particularly useful for tasks where the goal is to uncover hidden patterns or groupings within the data, without the need for labeled examples.
- Reinforcement learning (RL): in this class of algorithm there is an agent that learns to make decisions by interacting with an environment. The agent receives feedback in the form of positive or negative rewards based on the actions it takes and on how the environment responds. The objective is for the agent to learn a policy, that maximizes the cumulative reward over time. The learning process involves the agent exploring different actions in various states, receiving feedback and adjusting its strategy to maximize cumulative rewards over time.

Among the three macro area listed above, one issue concerning the supervised learning algorithm is the data scarcity in specific datasets that can lead to multiple problems such as:

- Model Generalization: The effectiveness of models is often measured by their ability to generalize from the training data. Increased exposure to diverse examples enhances the likelihood of achieving this desirable property;
- Unbalanced Datasets: this issue is present when there is a class that is larger in number with respect to another one. This could lead to bad performances by the models since it could become biased toward one class.

In the following list are present some solutions to these problems:

- With Reinforcement Learning the agent is able to build itself a set of example of the form a pair (state, action) to its corresponding reward in order to complete a task without being instructed on how to do it. In general this paradigm can't be used for substituting any supervised learning problem (e.g. image classification is hard or even impossible to be formulated as a RL problem). Moreover this can lead to high cost in therms of time, money and architecture, therefore it can't always be exploited as a solution.
- Data Resample: in case of unbalanced dataset, some of the pairs that input-output that are less present can be duplicated in order to restore the equilibrium among the classes.
- Data Augmentation: this is a technique used in the context of machine learning applied to computer vision to artificially increase the diversity of a training dataset. Instead of collecting new data, data augmentation involves applying various transformations to the existing training samples, creating augmented versions of the original data. These transformations can include rotations, flips, zooms, shifts, crops and changes in brightness or color. By exposing the model to a broader range of augmented data during training, it helps improve the model's ability to generalize and perform well on new, unseen examples.

- Transfer Learning: In training models, if two models are related to some domain, then it is possible to transfer knowledge to improve the results of the target learner. In this case the models does not have to learn from the zero but has already a solid starting point. Moreover a benefit of using transfer learning is the less time spent on the training process and smaller probability of getting stucked in a local minimum.

1.3 Thesis objective

Since the problem of lack of data in Machine Learning described in the paragraph above, this thesis objective was to continue the development of a robot rendering program that could simulate the functions of an industrial camera. The aim of this project was to firstly automatize the process of creating a synthetic dataset of images with their respective labels. Afterwards, this dataset is used to train a chosen object detection algorithm. The algorithm is then tested on a test dataset, made of real images and priorly labelled, to assess the performances of the vision algorithm. Lastly, parameters in the simulation are changed in order to generate more realistic images for a new synthetic dataset on which the vision algorithm will be again trained.

1.4 The starting point

At the beginning of this work I had little to no knowledge of:

- How a relatively big software is structured and the general workflow. Only some basis;

- C# programming although I had some experience with C and very little experience with C++;

- Experience with object-oriented programming;

- Experience with user interface programming and the MVVM paradigm;

While I acknowledge that I am far from mastering any of the aforementioned technologies, I express my gratitude and appreciation to Euclid Labs s.r.l. for providing me with the opportunity to experiment and gain valuable experience in this field.

1.4.1 Light model used

Euclid Labs S.r.l. provided me with software that included a real time rendering component. This component consisted in a 3D virtual environment capable of rendering various entities such as 3D objects, cameras, and lights. The implementation of the Linear Transformed Cosine (LTC) technique was already employed and used to facilitate the accurate rendering of lights and their associated behaviors. Making a step back, let's introduce some lighting basics. Nowadays, the prevailing method for real-time lighting computations is denoted as "per-pixel shading." This approach entails the calculation of light values individually for each pixel displayed on the screen. To achieve this it is usually available:

- P: the pixel position in the 3D world;
- N: the vector corresponding to the normal to the surface where the point P is;
- V: the vector from the point P to the observer;
- L: the vector that connects the point P to the light source;
- A material holds details on how it reacts to incoming light.

Usually what rendering algorithms try to accomplish is to approximate in some way the rendering equation[12]:

$$L_o(P,V) = L_e(P,V) + \int_{\Omega} f_r(P,L,V) L_i(P,L) \langle N,L \rangle dL \tag{1.1}$$

where:

- L_e is the light directly emitted by the point P in the case it belongs to a light source (e.g a lamp or a fire);

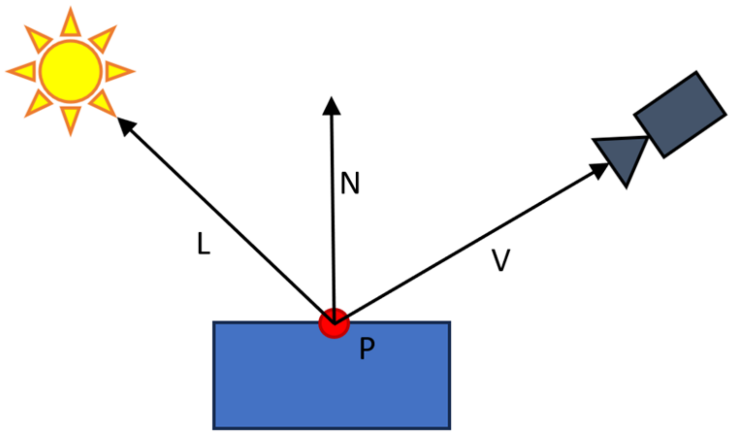


Figure 1.1: Reference model

- The integral expresses the necessity to compute the contribution from each light source by integrating over all directions L on an unit sphere Ω centered around the point P, with normal N as zenith.

- $\langle N, L \rangle$ denotes a clamped cosine function. In simple terms, it means that light from behind the point doesn't really impact the brightness or radiance emitted from that point.

- L_i is the radiance that reaches the point P

- $f_r(P, L, V)$ is the so called Bidirectional Reflectance Distribution Function (BDRF) of the surface where P lays. In the general form, it divides the light into two terms: the diffuse term, representing light reflected from point P in all directions; the specular term, capturing light reflected within a cone surrounding the perfectly reflected light ray.

With the exception of certain scenarios, such as point light sources, it is not always possible to obtain a closed-form solution for the rendering equation within a finite timeframe. Consequently, various models have been proposed to approximate this equation. One of the most simple models available to render light, and also the first one implemented on the software, is the Blinn-Phong model [11]:

$$L_o(P,V) = (K_d\langle N \cdot L\rangle + K_s\langle N \cdot H\rangle^n)Li \tag{1.2}$$

where

- H is the halfway vector defined as $H = \frac{L+V}{\|L+V\|}$
- K_d is the fraction of light diffused
- K_s is the fraction of light specularly reflected
- n is the specular power or the shininess and indicates how close to the behaviour of a mirror is the surface (a perfect mirror would have $n \rightarrow \infty$)

Despite its simplicity and real-time capabilities, the model tends to produce inaccurate results. At the end, the model used in this rendering software was Linearly Transformed Cosine (LTC). The idea behind this technique is this: let's take the Rendering Equation. A BRDF that behaves similarly to the original Phong model is the following:

$$f_r(P,L,V) = \frac{K_d}{\pi} + K_s\frac{n+2}{2\pi}\langle V \cdot R\rangle \tag{1.3}$$

While the first term of the integral can be shown to have a closed solution, the second term has to be solved numerically with computational complexity of $O(n)$ but we want this to be constant. This technique applies an approximation to the specular term in order to transform it into a the integral of a clamped cosine which can be analytically solved [6].

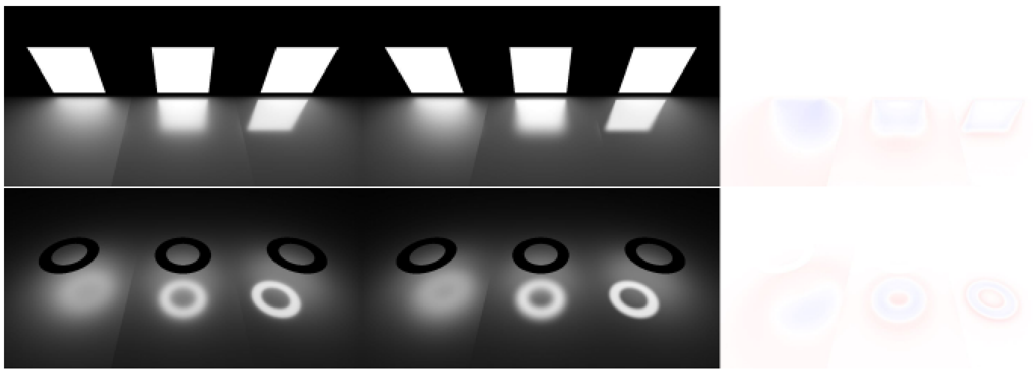


Figure 1.2: LTC rendering algorithm compared to ground truth

1.4.2 YoloV5

To assess the efficacy of the implemented rendering algorithm, the YoloV5 algorithm has been employed. Yolo, which stands for "You Only Look Once," is a member of the family of real-time object detection algorithms. Here it follows a brief description of this algorithm. Yolo was born from the necessity of object detection algorithms that could work in real time application such as the recognition of vehicles, pedestrians, bicycle and other obstacles in autonomous driving tasks. Back in the past, to achieve object detection, a single or multi object classifier algorithm was employed in a sliding window at different scales and position. Another technique used to achieve a similar result was to use a neural network that could propose different bounding boxes and on each bounding box a classifier was run to verify whether an object was present or not. Both these method result to be quite accurate but very slow since multiple steps have to be completed in order to fulfill the task. Yolo stated this problem as a regression problem directly from pixels of an image to the bounding box coordinates and class probabilities. In this way the inference results to be much faster with respect the two methods presented before while reaching a good, although lower, precision. Furthermore, in contrast to previous methods, Yolo reasons globally by processing the entire image through the neural network. Unlike its predecessors, which operated on localized regions defined by sliding windows or proposed regions, Yolo takes a holistic approach. This global perspective allows Yolo to consider contextual information across the entire image during the object detection process, potentially enhancing its ability to recognize complex patterns and relationships. The utilization of the entire image as input sets Yolo apart in terms of its global reasoning capabilities. The first version of Yolo had the following workflow. At the beginning the input image was divided into an SxS grid where S could be considered an hyperparameter. If the centre of an object falls into a grid cell, it is responsible for detecting that object. Each grid cell predicts B bounding boxes in the form of (x, y, w, h, P) where the pair (x, y) identify the centre of the bounding box, (w, h) its width and height and P is the probability that the bounding box contains an object or not. Lastly, each grid cell predicts C class probabilities, namely, given that an object is present in the cell, how likely is that a specific object is contained in a specific cell. To summarize, the output of a Yolo Neural Network is a tensor of dimension $S * S * (B * 5 + C)$. For what concern the Yolo architecture, it is fairly simple: it consist in 24 convolutional layers followed by 2 fully connected layers.

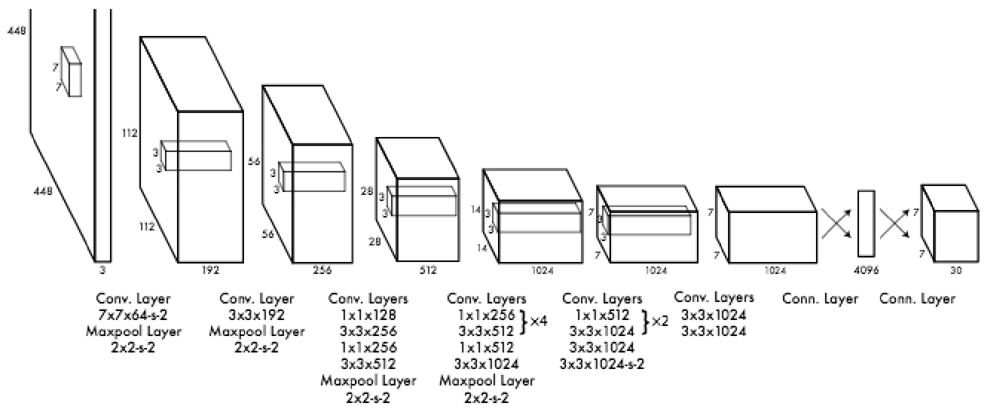
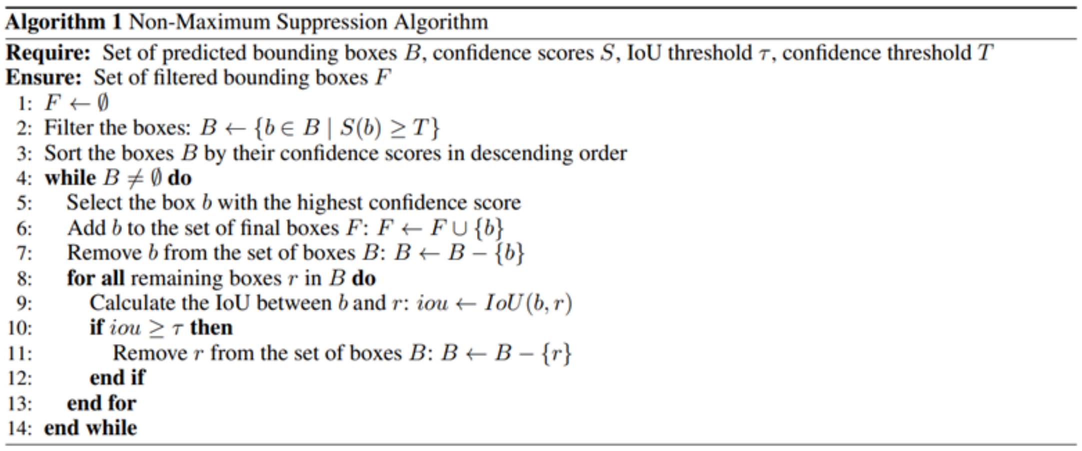


Figure 1.3: YoloV1 CNN architecture

The last component to define for what regards the Yolo Neural Network is its loss function, essential for the parameters training part. In fact, during the training, it is tried to optimize the following loss function:

$$
\begin{aligned}
\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{1.4}
$$

where $\mathbb{1}_i$ denotes if object appears in cell i and $\mathbb{1}_{ij}$ denotes that the jth bounding box predictor in cell i is "responsible" for that prediction. Moreover: the summations multiplied by λ_{coord} can be defined as the *localization loss* namely the part of the loss that tries to reduce the error on the bounding boxes predictions; the third line summations can be defined as the *confidence loss* namely a penalty either if an object is predicted but not present or if an object is not predicted but present; the last term can be defined as *classification loss* for category prediction accuracy [5]. The last step for the Yolo algorithm to run is the Non Maximum Suppression (NMS) technique. It serves as a post-processing method employed in object detection algorithms. Its primary role is to minimize the occurrence of overlapping bounding boxes, thereby enhancing the overall accuracy and precision of object detection results. In the field of object detection algorithms, it's common for multiple bounding boxes to be generated around a single object, each accompanied by varying confidence scores. Non-Maximum Suppression steps in to sift through this abundance of bounding boxes, discarding redundancies and retaining only the most precise ones. This process streamlines the output, ensuring that the final set of bounding boxes represents the most accurate and relevant detections [8].



```
Algorithm 1 Non-Maximum Suppression Algorithm
Require: Set of predicted bounding boxes $B$, confidence scores $S$, IoU threshold $\tau$, confidence threshold $T$
Ensure: Set of filtered bounding boxes $F$
$F \leftarrow \emptyset$
Filter the boxes: $B \leftarrow \{b \in B \mid S(b) \geq T\}$
Sort the boxes $B$ by their confidence scores in descending order
while $B \neq \emptyset$ do
    Select the box $b$ with the highest confidence score
    Add $b$ to the set of final boxes $F$: $F \leftarrow F \cup \{b\}$
    Remove $b$ from the set of boxes $B$: $B \leftarrow B - \{b\}$
    for all remaining boxes $r$ in $B$ do
        Calculate the IoU between $b$ and $r$: $iou \leftarrow IoU(b, r)$
        if $iou \geq \tau$ then
            Remove $r$ from the set of boxes $B$: $B \leftarrow B - \{r\}$
        end if
    end for
end while
```

Figure 1.4: Non Maximum Suppression Pseudo-code

These described above are the fundamentals of the Yolo algorithm. In this project, it is made use of the 5th version of this algorithm (YoloV5) which presents some improvements in terms of efficiency and accuracy. First of all, it uses many improvements used in YoloV4 but developed in PyTorch deep learning framework. It uses several augmentation techniques for better generalization such as random affine, HSV augmentation, random horizontal flip as well as other. YoloV5 offers a range of scaled versions, each tailored to specific applications and hardware demands. These variants include YoloV5n (nano), YoloV5s (small), YoloV5m (medium), YoloV5l (large), and YoloV5x (extra-large). The adjustments in width and depth of the convolution modules allow customization for diverse use cases. For example, YoloV5n and YoloV5s are designed as lightweight models, suitable for resource-constrained devices, whereas YoloV5x prioritizes high performance, albeit with a trade-off in speed.

1.4.3 MVVM paradigm

Model-View-ViewModel (MVVM) is a widely-used architectural pattern in software development that facilitates the separation of concerns between the graphical user interface (GUI or the "view") and the business logic or back-end processes (the "model"). This segregation ensures that the view remains independent of any specific model platform, promoting modular and maintainable code. In MVVM, the viewmodel serves as a crucial component, essentially acting as a value converter. Its primary responsibility is to expose and convert data objects from the model in a way that is easily manageable and presentable for the view. Contrary to its name, the viewmodel often plays a more significant role in managing the model than the view. It takes charge of most, if not all, of the view's display logic. One notable characteristic of the viewmodel is its potential implementation of a mediator pattern. By doing so, it organizes access to the back-end logic, coordinating interactions between the view and the underlying business logic. This mediation helps organize and streamline the flow of data and user interactions, aligning with the specific use cases supported by the view. It's worth mentioning that MVVM has been prominently incorporated into the Windows Presentation Foundation (WPF), a framework by Microsoft for building desktop applications with rich user interfaces. Additionally, the term "Model-View-Binder" is sometimes used interchangeably with MVVM, emphasizing the role of the binder in connecting and synchronizing the model and view components. Overall, MVVM stands as a powerful architectural pattern that enhances code organization, maintainability, and scalability in software development.

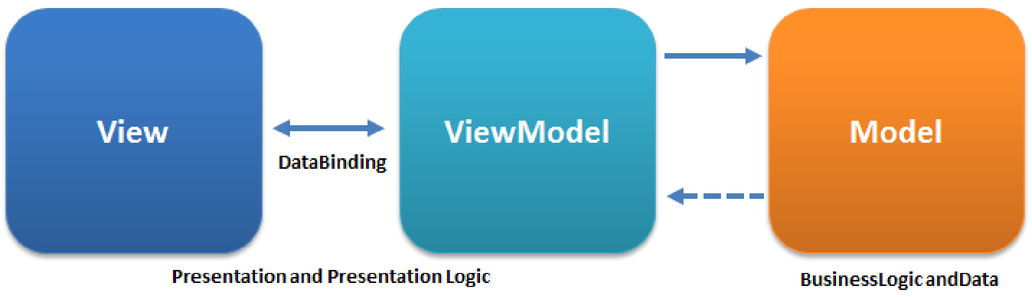


Figure 1.5: MVVM pattern

2 Preliminary works

As previously described in the Chapter 1, I was provided with a software with the rendering component already implemented. The greatest part of my project focused on the preliminary works made in order to then asses the performances of the object detection algorithm trained with synthetic images.

2.1 Software analysis

The journey with this project began with the analysis of the code provided. Providing an in-depth explanation of the code is not possible as a portion of it is proprietary to Euclid Labs. Hence, the forthcoming content will offer a highly simplified overview of the outcome. This software uses libraries from the company MARS program, a simulator for industrial manipulator workcells. In the first execution attempt, the code presented an user interface subdivided into three columns: the left hand side with a selection of actions that the user can access to; in the middle column a window displaying the rendered 3D world; in the right hand side a column some user command to move objects and cameras and to add or remove and manipulate the lights in the scene. The user interface was connected to the computation in the back end through the MVVM paradigm which allows to decouple the UI design with the logic behind.

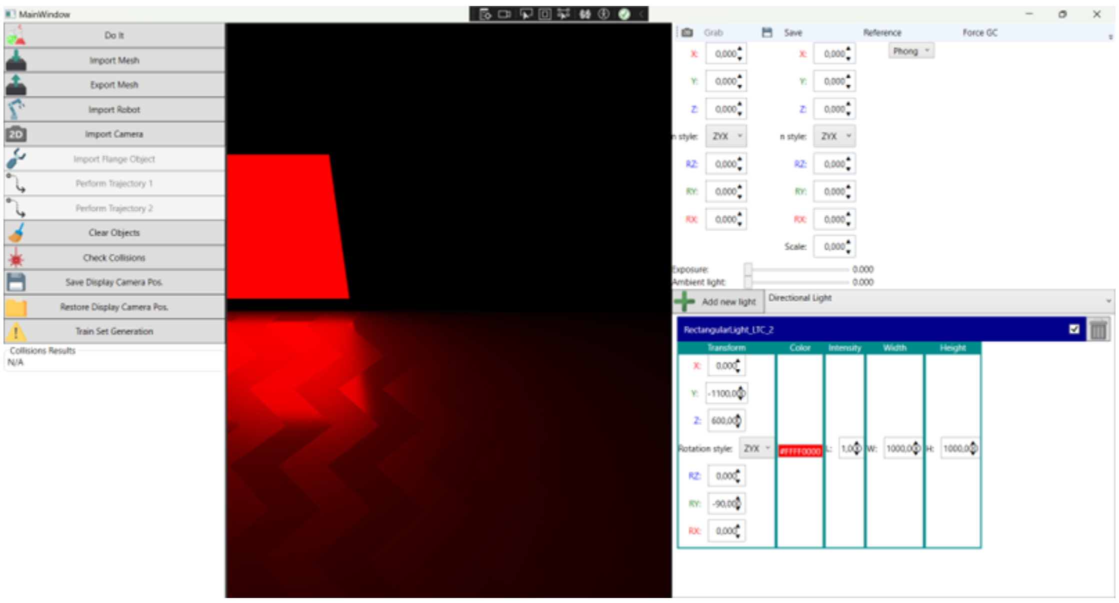


Figure 2.1: Initial software UI

2.2 UI modification

With MVVM paradigm, it was necessary to immediately start to learn the concept of binding. In fact, the very first task assigned was to modify the buttons that allow to change the position and the orientation of an object. Although seeming a simple task, I encountered several challenges when attempting to make it modular and reusable. The workflow followed for implementing this first part was to firstly design the single number textbox with its up and down buttons; the buttons have been defined as *repeat buttons* in order to increase or decrease the number if kept pressed. Moreover, in order to add further flexibility to the user command, it was implemented a mechanism that allow to modify the number by 0.1 if the left shift key was being pressed, 10 if the left control key was being pressed and 100 if both were being pressed. The outcome turned out to be a slightly more user-friendly input where the numbers digited don't superimpose with the arrows. Once accomplished this part, three of these inputs were group

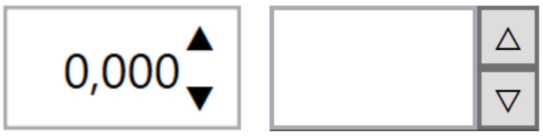


Figure 2.2: Comparison with pre and post single value input

in order to form a vector modifier. It was made use of the dependency property to add property to this custom element such as the input value which could be accepted as *Vector3D*. The final component to upgrade was the matrix modifier,

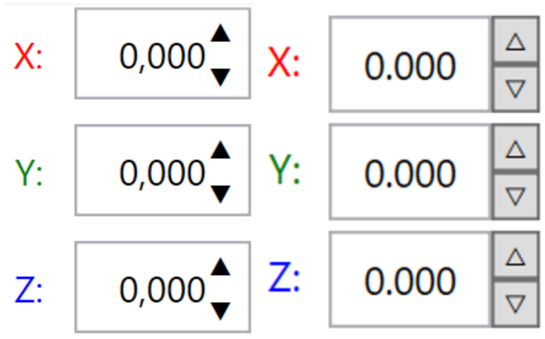


Figure 2.3: Comparison with pre and post vector value input

which proves useful for altering the position and orientation of objects within the scene. In this case it was made use of both the previous custom controls since the position was always defined as a triplet (x, y, z) while the orientation could be defined as different types of Euler angles. It was maintained the original feature which was that when changing the angle triplet, if the rotation was made about the X axis the colour assigned was red, about the Y axis green and about the Z axis blue.

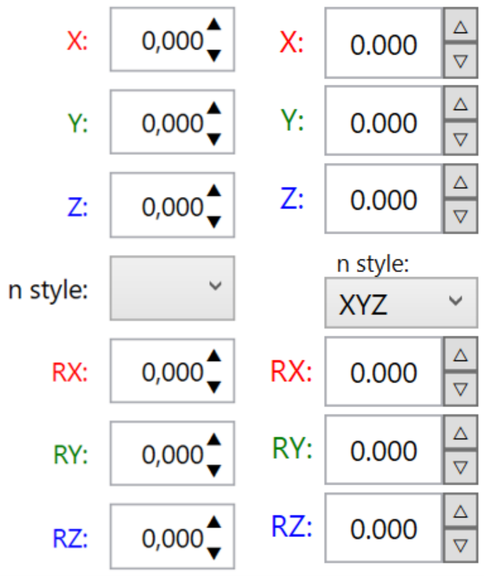


Figure 2.4: Comparison with pre and post matrix value input

2.3 Automatisms

In this section it is presented the automatisms implemented for the generation of the images for the Yolo algorithm training

2.3.1 Auto adjust camera position

Regarding the effective computation, the first task assigned was to create an automatism in order to get some confidence with the manipulation of the objects, camera and lights inside the 3D rendered world. In this particular scenario, the objective was to position the camera directly above the object, regardless its initial position, in such a way that when a picture is taken, the object occupies the entire frame without any empty spaces on the width or height of the image. To achieve this it was needed to know where each point of the mesh composing the 3D object was. A 3D object is essentially composed by a number of points describing the vertices of the object. Those points are then connected by small triangles in order to make the object defined and visible. The more the points that defines the object, the more accurate is at sight but also the more computationally expensive. While exploring the given software,

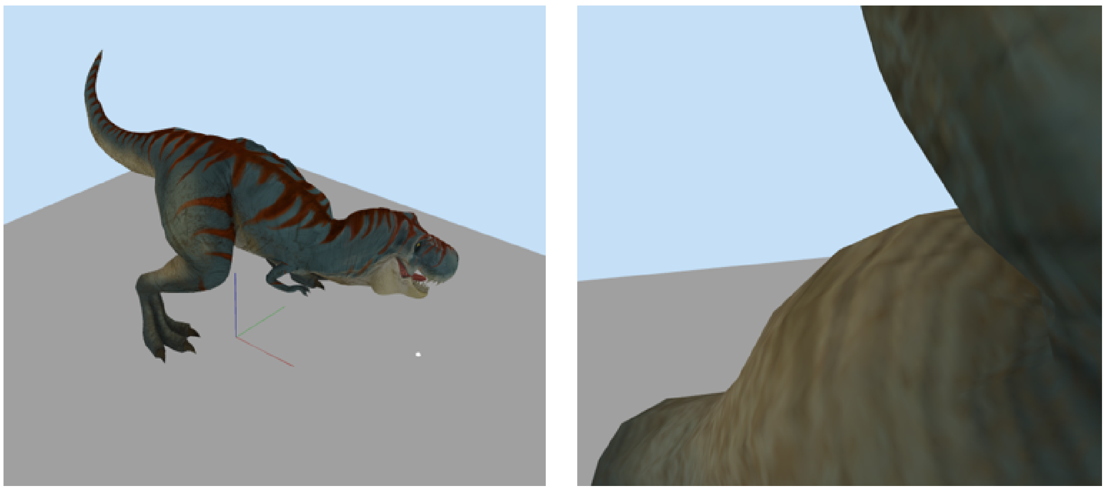


Figure 2.5: Mesh example, on the right it is visible the irregularities

a function utilizing a mesh was discovered, serving as a source of inspiration. The first attempt made was to go through all the mesh region and for each mesh region go through all the vertex. The initial suppositions were made about the starting position of the camera with height equal to $-\infty$ and the object with its centre along the Z axis. It was considered the problem from two different perspectives, one per side, and then the solutions were merged at the end. Let's take one of them as an example: suppose we are considering the first point in

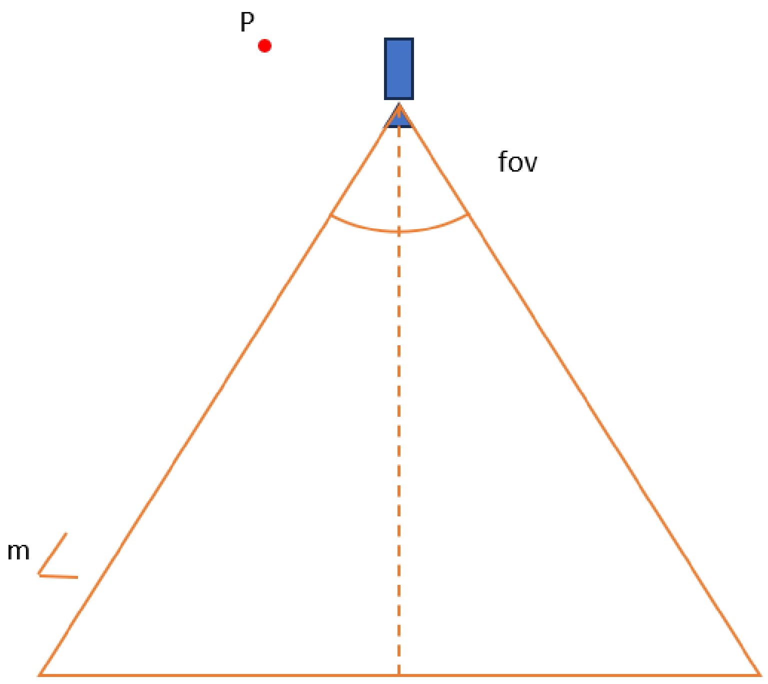


Figure 2.6: Camera positioning scheme

this case. The field of view of the camera can be modeled as two lines with coefficient m and -m where m is given by the equation

$$m = \tan\left(\frac{\pi}{2} - \frac{fov}{2}\right) \tag{2.1}$$

The objective then becomes, given the line equation

$$z = m * x + q \tag{2.2}$$

find the line that is above each point of the mesh, namely, given the family of parallel lines, the line that has the highest y axis intercept. Therefore, for each point, defined by the pair (x, z) or (y, z) it is computed its q value and it is checked whether it is higher or not with respect to the previous maximum; if it is, it is set as the new maximum. This process is execute for both the line with coefficient m and -m hence we will have q_+ and q_- . Once the two lines have been univocally, to find the camera height it can be used the point of intersection of the lines. Moreover, this gives also the position of the camera on the X or Y axis depending on which side we are solving the problem. Given that there will be two terms for Z, the highest one is selected to prevent cropping the object in the image. At the end, since every computation is made with respect to the vertex of the mesh, hence without keeping account of the effective position of the object inside the 3D world, an offset corresponding to the compound transform of the object is added to the camera position previously found. It is possible to say that this algorithm works for the purpose of the project but it does not keep count of the orientation of the object. In fact, if the object is not a sphere and is rotated around any axis, since the computation were made with respect to the mesh, the algorithm does not return what expected(Figure 2.7). This is due to the fact that the mesh could be described as a stencil representing the object. The compound transform then tells to the GPU where to draw the object. To solve this problem,

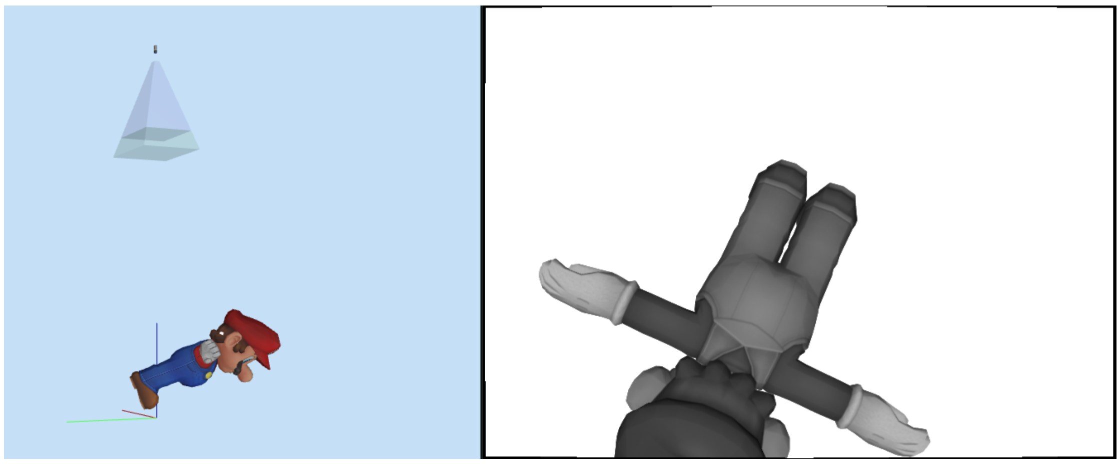


Figure 2.7: With rotated object the outcome is wrong

before doing any of the computation described above, each point was rotated by the rotation matrix representing the object orientation(figure 2.8).

```
private void AutoAdjust(object param)
    {
        double adjustedHegiht;
        double vFoV = (simCamera.GetVerticalFoV()) / 2;
        double hFoV = (simCamera.GetHorizontalFoV()) / 2;
        double q_min_x = double.MinValue;
        double q_max_x = double.MinValue;
        double q_min_y = double.MinValue;
        double q_max_y = double.MinValue;
        double m_x = Math.Tan((MathConstants.PI / 2) - hFoV);
        double m_y = Math.Tan((MathConstants.PI / 2) - vFoV);
        foreach (elMeshRegion meshregion in
        selectedObject.Mesh.MeshRegions)
            foreach (elVertex item in meshregion.VertexBuffer)
            {
                Euclid.MathFun.Vector3D position =
                item.Position;
                Euclid.MathFun.RotationMatrix.Multiply
                (selectedObject.CompoundTransform.Rotation,
                position, ref position);
                double new_q = position.Z - m_x * position.X;
                if (new_q > q_min_x)
                    q_min_x = new_q;
                new_q = position.Z + m_x * position.X;
                if (new_q > q_max_x)
                    q_max_x = new_q;
                new_q = position.Z - m_y * position.Y;
                if (new_q > q_min_y)
                    q_min_y = new_q;
                new_q = position.Z + m_y * position.Y;
                if (new_q > q_max_y)
                    q_max_y = new_q;
            }
        double centreX = (q_min_x - q_max_x) / (-2 * m_x);
        double centreY = (q_min_y - q_max_y) / (-2 * m_y);
        //find z
        double x = (q_max_x - q_min_x) / (2 * m_x);
        double z1 = m_x * x + q_min_x;
        double y = (q_max_y - q_min_y) / (2 * m_y);
        double z2 = m_y * y + q_min_y;
        adjustedHegiht = Math.Max(z1, z2)
        + simCamera.PinholeModelOrigin.Z
        + importedObject.PositionTransform.OriginOffset.Z;
        centreX += selectedObject.OriginOffset.X;
        centreY += selectedObject.OriginOffset.Y;
        CameraPosition = new HomogeneousMatrix(centreX,
        centreY, adjustedHegiht, 0, 0, MathConstants.PI);
```

```
    }
```

Code 2.1: Final camera auto-adjust algorithm

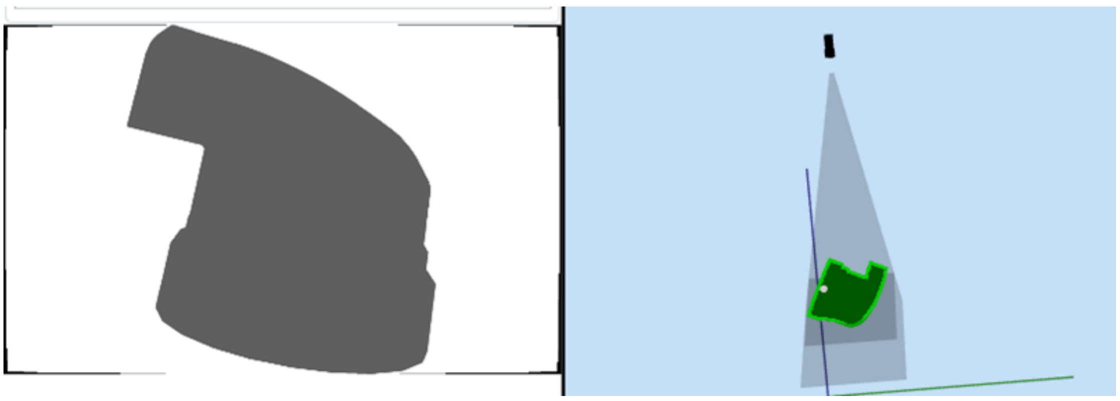


Figure 2.8: Updated algorithm, the object occupies all the image height-wise

2.3.2 Synthetic dataset generation

The very relevant part of this project started here, wherein pertinent information regarding the labeling process was obtained from the YoloV5 algorithm documentation [9]. Therefore in this section it is going to be presented the algorithm with which the dataset was created. First of all it was needed to develop the appropriate user interface part and it was accomplished in the following manner: the data to be prompted is the number of images, that compose the whole dataset, and the test split percentage. Once these input are present and the camera and the object in the 3D world are selected, then the *Generate Image Set* button is enabled.

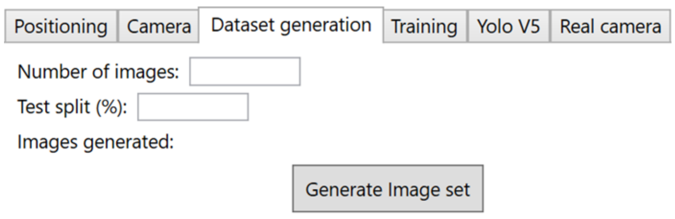


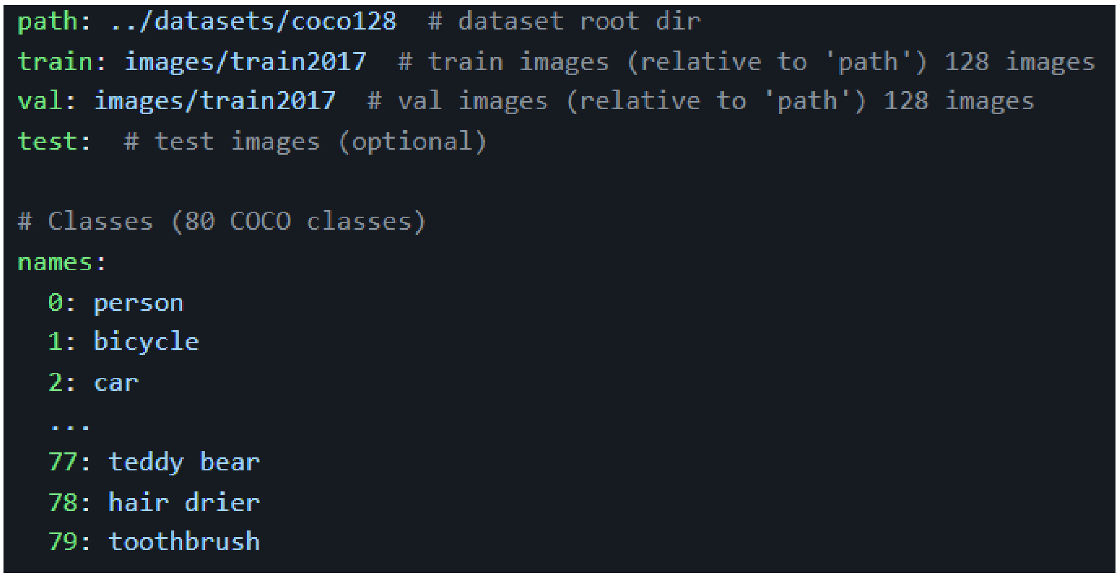
Figure 2.9: Dataset Generation tab

After pressing the button, a new folder labeled "custom#" is generated within the dataset directory, with # representing the incremented last folder number.

Subsequently, additional folders are created, including "train" and "validation" directories. Within each of these, subdirectories named "images" and "labels" are set up. After this, it is required to create a YAML file. This file is required for YoloV5 in order to know where the dataset is located and which classes of object it will predict. The YAML file is composed in the following way:

- path to dataset folder
- path to training images relative to the dataset directory
- path to validation images relative to the dataset directory
- path to test images relative to the dataset directory
- class name dictionary

In figure 2.10 is an example taken from the Ultralytics github repository. Once



```
path: ../datasets/coco128  # dataset root dir
train: images/train2017  # train images (relative to 'path') 128 images
val: images/train2017  # val images (relative to 'path') 128 images
test:  # test images (optional)

# Classes (80 COCO classes)
names:
  0: person
  1: bicycle
  2: car
  ...
  77: teddy bear
  78: hair drier
  79: toothbrush
```

Figure 2.10: Example of Yaml file

the various subdirectories and the YAML file are created, it is time to generate the training images and their corresponding label files. As it will be explained in the next section, in order to train the YoloV5 algorithm it is required to already have the images and labels divided into training and validation set. Hence, the choice made was to create an array containing the range of numbers from 0 to the number of image generated, shuffle it with the Fisher-Yates algorithm and take only the first M indexes where M is the number of validation images selected before.

```
static void Shuffle<T>(T[] array)
    {
        Random random = new Random();
        // Start from the end of the array and swap each
        //element with a random one before it
        for (int i = array.Length - 1; i > 0; i--)
        {
            int j = random.Next(0, i + 1);
            // Swap array[i] and array[j]
            T temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }
```

Code 2.2: Fisher-Yates algorithm

Therefore, since a for loop is created in order to generate N images, each time that the i-th index, which is increased at every iteration, is contained in the array above mentioned, the results are saved into the validation folder. Now let's tackle the effective image generation code. In this part it was needed to decide how the images were taken since there is not an effective rule. The decision was to make the object move from left to right while centered on the height of the image for half of the number of images taken and from bottom to top while centered on the width of the image for the other half. To add a bit more of generalization in the images, a Gaussian noise is injected to the position in the axis perpendicular to the one on which the object moves. Moreover, the object is casually rotated along its Z axis in order to view it from different viewpoints. Lastly, on the image side, it was decided to resize pictures to a dimension of 640∗640 pixel which could be accepted as input of the YoloV5 Neural Network. In figure 2.11 some example of images generated. For each image generated, a txt file containing the labels had to be produced. The labels for YoloV5 have to be in the form of (L, x, y, w, h) where L is the class of the object contained inside the bounding box; x and y are correspondingly the center on the horizontal axis and vertical axis of the bounding box; w and h are the width and height of the bounding box. All the measures but L must be normalized with respect to the images size hence each of those numbers will be bounded between 0 and 1 (figure 2.12). The first attempt for accomplish the labelling task was to use

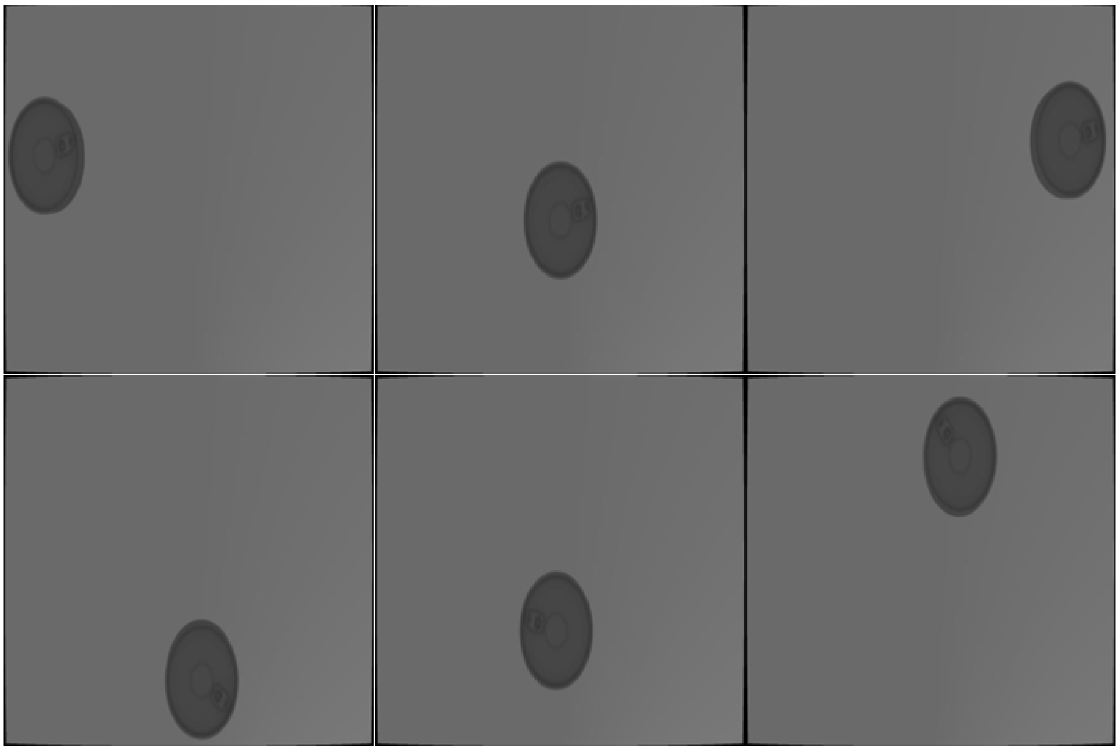
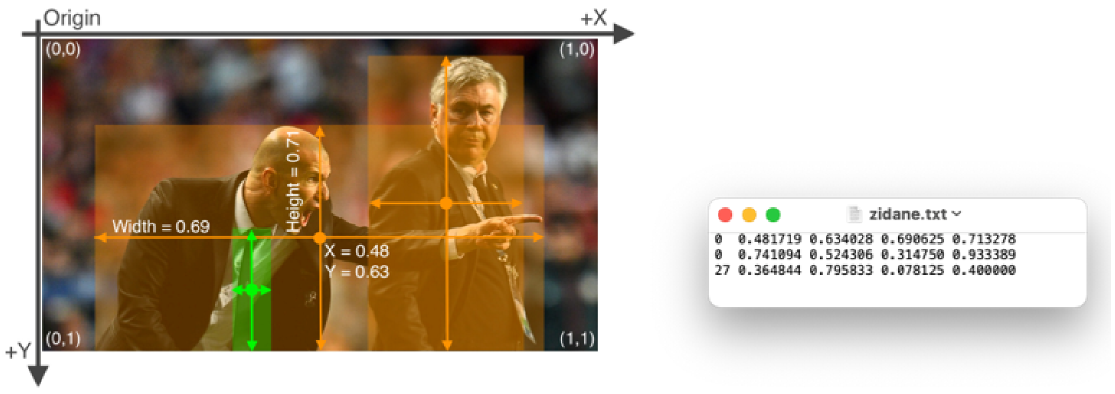


Figure 2.11: Example of images generated

the bounding box of the object. With this, a 3D vector for both the maximum and the minimum point of the bounding box were taken into consideration. Given that the object has been approximated as a rectangular parallelepiped and positioned in a way that only one face is directly aligned with the camera axis, it is necessary to solely consider the aspects of the face that is facing the camera (figure 2.13). This approach simplifies calculations and facilitates a more straightforward representation of the object. Hence, to compute the label of an object it is considered as image plane a rectangle posed at the height of the upward face of the parallelepiped. The x and y coordinate of centre of this figure are the same as the one of the camera. Its width and height is computed with some trigonometric calculations namely given that it is known from the manufacturer the horizontal and vertical field of view and the distance between the camera origin and the parallelepiped face, the width and height of the plane becomes:

$$\begin{aligned} width &= [(CameraHeight - PlaneHeight) * \tan(HorizontalFoV/2)] * 2 \\ height &= [(CameraHeight - PlaneHeight) * \tan(VerticalFoV/2)] * 2 \end{aligned} \tag{2.3}$$



```
0  0.481719 0.634028 0.690625 0.713278
0  0.741094 0.524306 0.314750 0.933389
27 0.364844 0.795833 0.078125 0.400000
```

Figure 2.12: Example of labelling file

Once the measurements have been obtained, it becomes feasible to perform reasoning solely in relation to the plane under consideration. By establishing the bottom left corner as the origin for the bounding box points, normalizing the measures involves subtracting the coordinates of the bottom left corner from both points. Subsequently, the x-coordinate is divided by the width of the image plane, and the y-coordinate by its height. With normalized coordinates in place, they can then be utilized to calculate pixel coordinates by multiplying

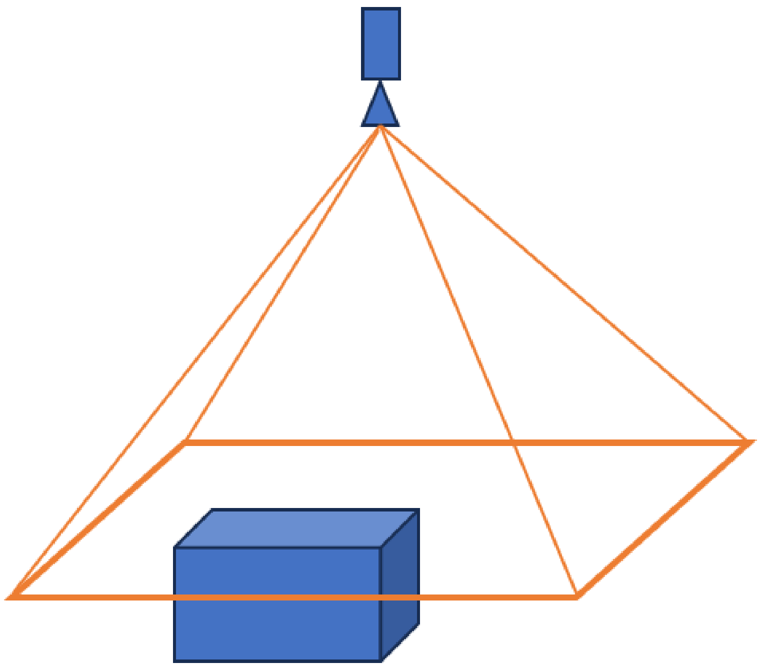


Figure 2.13: First attempt for automatizing Bounding Boxes placement scheme

them for the width and the height of image. Upon implementing the first algorithm, an imperfection in bounding box placement was observed. It was noticeable that when the object was near the center of the camera origin, the bounding boxes were well-placed. However, upon reaching the extremities of the camera field, the bounding boxes appeared to cut part of the object, as depicted in Figure 2.14. This is caused by the fact that the parallelepiped when

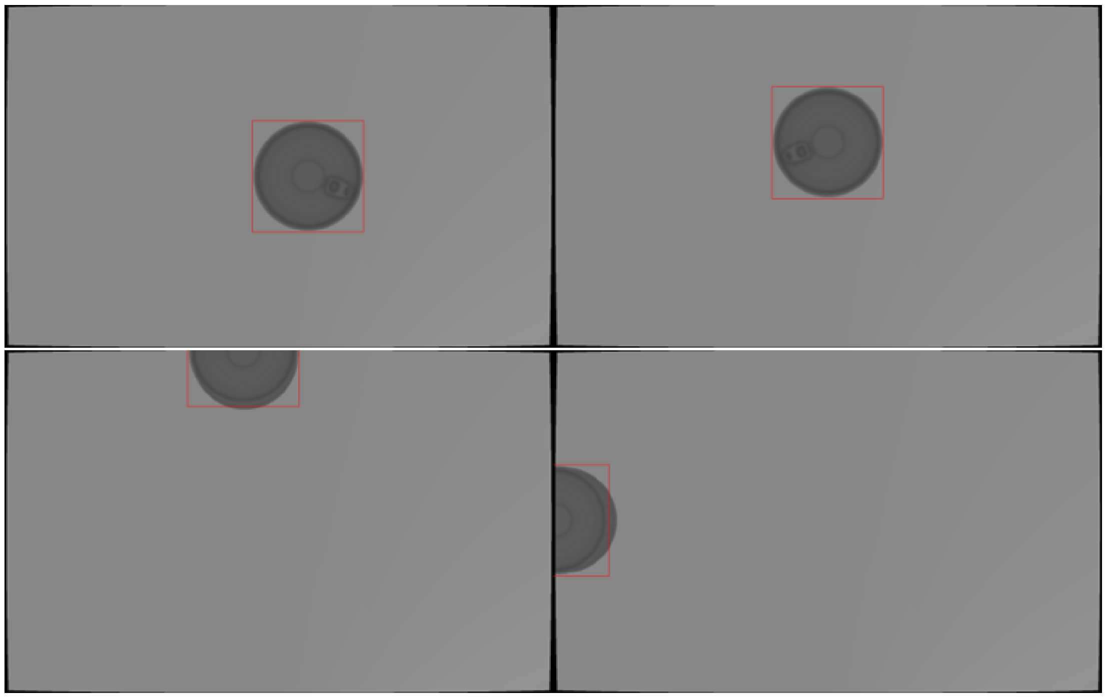


Figure 2.14: The top images have object well bounded while the bottom two are cut toward the centre

in the extremities of the camera field shows also one lateral side that needs to be taken into account (e.g if the object is on the left side of the image plane then the right side of the figure is not covered by the front side). This would be more evident the more the object depth is. In fact, taking the reasoning to an extreme case, if the object taken into consideration was flat then this phenomena would not appear. Therefore, in order to avoid this flaw, this algorithm was improved by taking into consideration the relative position on the xy plane of the object with respect to the camera. In other words, taking as a reference the image in the figure 2.15, if the object is on the upper left side of the camera centre, the vertex used to compute the bounding boxes are the top left corner of the front face and the bottom right corner of the rear face; in the case in which the object is in the bottom left side of the camera centre, the vertex used to compute the bounding boxes are the bottom left corner of the front face and the top right corner of the rear face; the other two cases are analogue to the previous ones and are respectively when the object is on the bottom right corner and in the top right corner of the camera centre. Moreover, in this case,

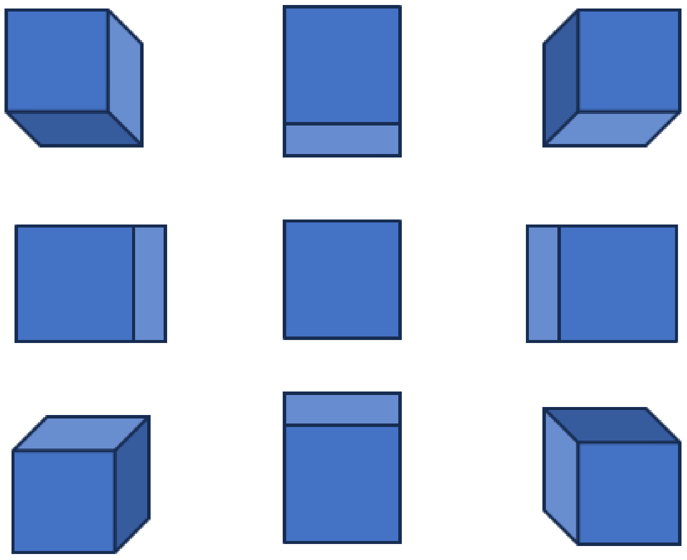


Figure 2.15: Perspective effect

since the points considered are laying on 2 different planes perpendicular to the camera axis, the reasoning exploited in the previous algorithm has to be extended to the point taken into consideration, namely to normalize the point coordinates in the xy plane the equation 2.3 has to be used for each of the points. The results given by this algorithm seemed better than the previous algorithm and are shown in figure 2.16. During testing, it became apparent that the algorithm encountered difficulties when applied to objects lacking symmetry with respect to the z-axis. The issue stemmed from the algorithm's reliance on a bounding box that was specific to the object's mesh, rather than being adaptable to the object's orientation in the 3D world. This limitation was evident in cases where objects, lacking z-axis symmetry, underwent rotation. For instance, the algorithm performed well with objects possessing z-axis symmetry, as a square bounding box could consistently encapsulate them, regardless of orientation, such as the case presented above where the tuna can has a cylindrical shape, hence if rotated around the z axis, the original bounding box remains still valid. Rotating a bounding box could not be an option since the output would not be an aligned rectangle. Also, using the vertices of the rotate bounding box was not

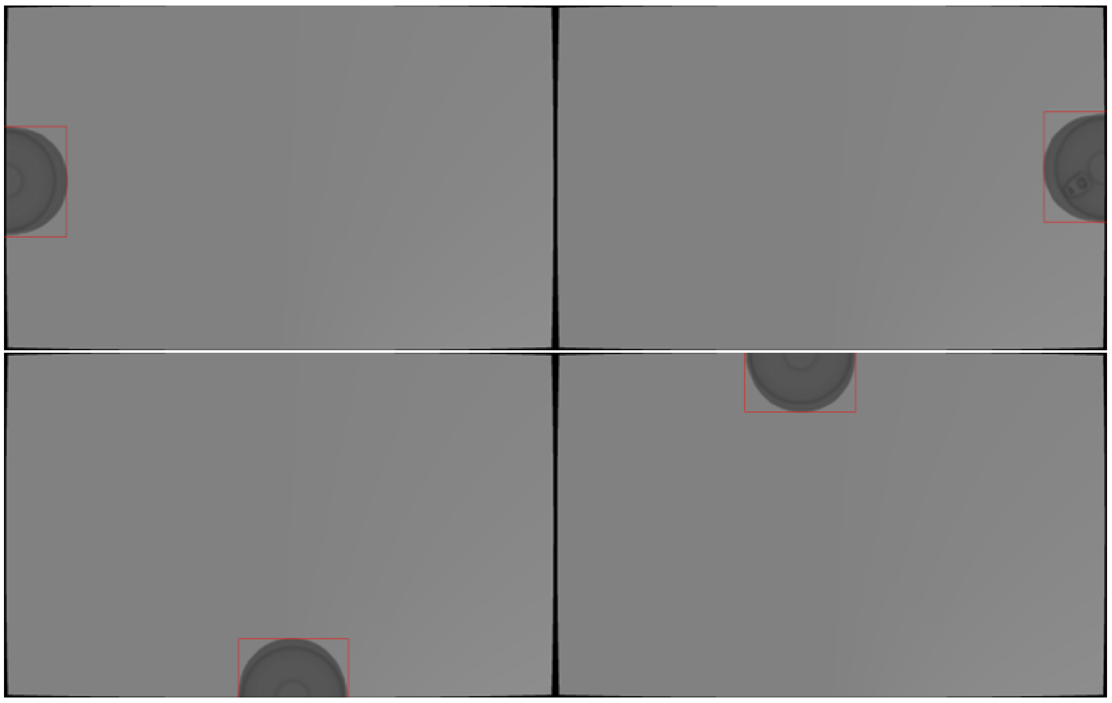


Figure 2.16: With this algorithm the BBs are corrected also at the extremities

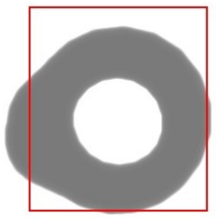


Figure 2.17: Flaw in the second algorithm

a solution since in this case the risk was to have wide spaces between the inner side of the BBs and the object. The last method resorted to was to evaluate each single point on the mesh, rotated and translated by the compound transform of the object and then evaluated with respect to the other points when normalized.

With this comparison, it is possible to find directly the points with minimum and maximum coordinates along the normalized x and y plane. Once these, which represent the normalized coordinate of the bounding box in the image plane, are present, the only remaining thing to do is to multiply them by the width and height of the image to obtain the BB in pixel coordinates.

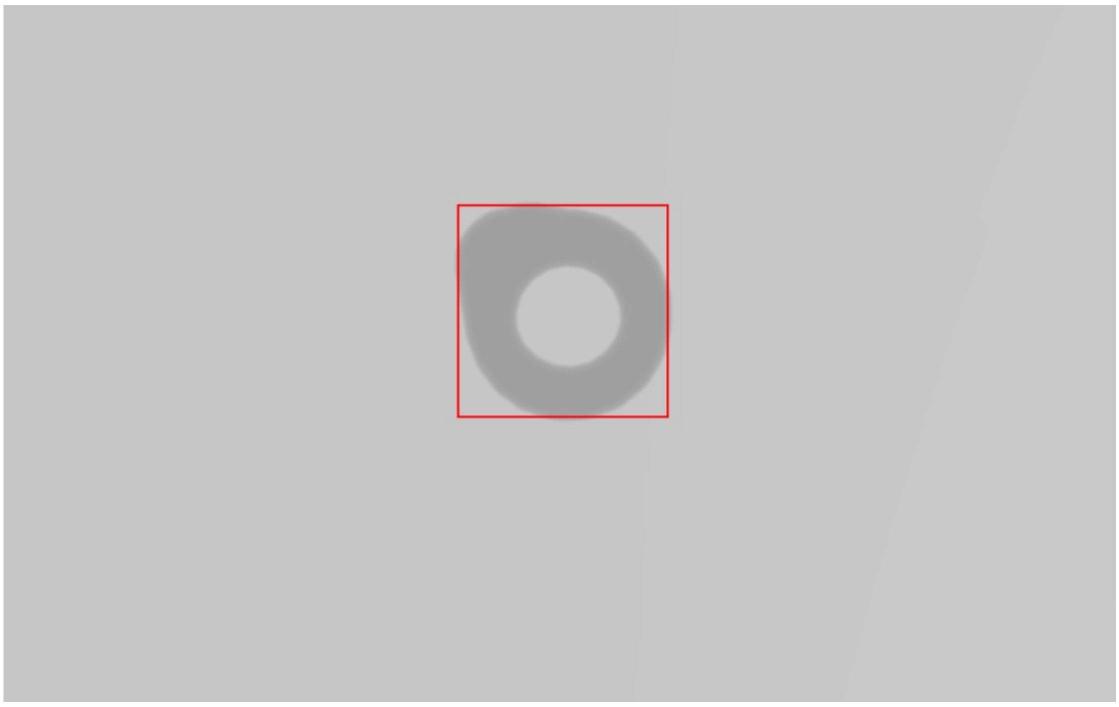


Figure 2.18: Final algorithm for BB labelling

2.3.3 Yolo training

In this part it is presented the methodology employed for training the algorithm. The YoloV5 algorithm was released a couple of months after YoloV4 in 2020 by Glen Jocher, founder and CEO of Ultralytics. Hence all the documentation necessary for the implementation of this algorithm was found on the Ultralytics website. First thing to do was to clone the git repository on the desired folder. After this it was required to install all the libraries necessary for the scripts to operate such as numpy, matplotlib, pytorch and many more. To do so, a requirement.txt file was provided in the cloned repository. In order to install the precise version of all the libraries and not to have conflicts between various versions of the same library already installed and other libraries, it was made use of a virtual environment. A virtual environment in the context of installing libraries typically refers to a self-contained and isolated workspace within a computer system. This type of environment allows users to manage and organize their project-specific dependencies, libraries, and packages separately from the system-wide installations. The primary purpose is to create a controlled environment, ensuring that the required software components for a particular project do not interfere with the global system configuration. Using tools like virtualenv (for Python) or venv, developers can create these isolated environments, encapsulating the specific versions of libraries and dependencies needed for a given project. This practice enhances project reproducibility, as the virtual environment can be easily shared, and other users can recreate the exact software environment to run the project seamlessly. Additionally, virtual environments contribute to better dependency management, enabling developers to work on multiple projects with potentially conflicting requirements without conflicts. Overall, virtual environments play a crucial role in maintaining a clean and well-organized development workflow, promoting consistency and ease of collaboration in software projects. To achieve what previously described, it was needed to run the terminal from a desired folder and creating and activating the virtual environment with the following commands:

```
python -m venv myVirtualEnv
venv\Scripts\activate
```

After successfully creating and activating the virtual environment (you can verify activation by checking if the command line prompt starts with the virtual environment name, e.g., (myVirtualEnv)), it can be proceeded with cloning the repository and installing the necessary requirements. Use the following commands [10]:

```
git clone https://github.com/ultralytics/yolov5
cd yolov5
pip install -r requirements.txt
```

Once this is done, it was possible to use the python script train.py in order to start the training of the algorithm. To make use of this script the terminal was employed. In fact, with the virtual environment still activated, it was possible to enter the following command:

```
python train.py --img 640 --epochs 3 --data data.yaml --weights
    yolov5s.pt
```

where:

- - - img 640 is the dimension in pixel of the input image;
- - - epochs 3 is the number of the epochs on which the Neural Network is trained;
- - - data data.yaml is the YAML file described in one of the previous sections;
- - - weights yolov5s.pt is the version of YoloV5 that it is decided to use (small). Other versions are YoloV5n (nano), YoloV5m (medium), YoloV5l (large), YoloV5x (xLarge). Moreover this allow to train the network with pretrained weights. These weights are trained on the COCO dataset.

Additional configuration options, such as batch size, optimizer selection, and the output folder for saving training results, can be incorporated into the setup. After configuring these parameters, integration with the C# code involves using the System.Diagnostics.Process class. Instantiate the Process object, providing the file name (e.g., cmd.exe), the working directory, and any other necessary options to update the user interface. Once the Process object is configured, it can be started, and the previously described commands, including virtual environment activation and initiation of the training script, can be executed. During the training process, a significant challenge arose as the available computer memory proved insufficient, leading to frequent system collapses. Initially, it was hypothesized that the default batch size of 16 might be the culprit. However, even after reducing the batch size, the issue persisted. In the quest for a solution, a crucial insight emerged from a GitHub discussion pertaining to the workers parameter. The workers parameter determines the number of CPU cores utilized during training. It was discovered that adjusting this parameter to a value of 1 resolved the memory problem, albeit at the cost of a considerable slowdown in the training process. This compromise, while effective in mitigating memory constraints, highlighted the delicate balance between computational efficiency and memory usage, necessitating careful consideration and optimization of training parameters for optimal performance. Once the training process is finished the script outputs a series of files among which are present 2 files named last.pt and best.pt. These are the pytorch files containing the NN weights at respectively the last iteration and the best iteration, which is chosen among the ones with best fitness to the validation dataset. The fitness to a certain dataset is evaluated as a weighted sum of the network Precision, Recall, mAP@.5 and mAP@.95 on the validation dataset. These metrics are defined with the following equations:

$$\begin{gathered} Precision = \frac{Tp}{Tp + Fp} \quad Recall = \frac{Tp}{Tp + Fn} \\ mAP = \frac{1}{|classes|} \sum_{c \in classes} \frac{\#TP(c)}{\#TP(c) + \#FP(c)} \end{gathered} \tag{2.4}$$

where Tp are the true positive predictions, Tn are the true negatives, Fp are the false positives and Fn are the false negatives. To determine if a prediction is, for example a true positive, it is made us of the Intersection over Union metric (typically indicated as IoU). The IoU between a set A, representing the pixels of the proposed object region, and a set B, representing the pixels of the true object region, is calculated as:

$$IoU(A, B) = \frac{A \cap B}{A \cup B} \tag{2.5}$$

Typically, an IoU value greater than 0.5 is considered indicative of a successful detection, while an IoU value less than or equal to 0.5 is considered indicative of a failed detection. Hence a prediction is considered a true positive if a proposal was made for class c and there actually was an object of class c. Therefore mAP@.5, which is the mean Average Precision at 0.5, is calculated by setting the IoU threshold at 0.5 [1] [2].

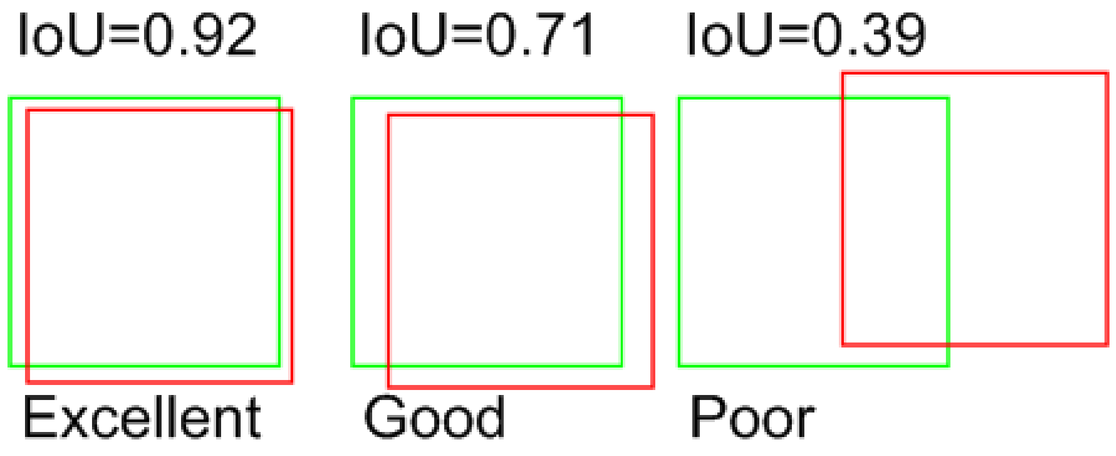


Figure 2.19: Example of Intersection over Union

Another precision to be made is that, depending on the algorithm used, validation and test dataset could change in meaning. In this case, the validation dataset is primarily employed to calculate metrics at the end of each epoch, with no direct impact on the model weights evolution while the test dataset is used in order to test the neural network onto new unseen data.

2.4 Model deployment

Once trained, the model has to be deployed. Model deployment in machine learning is the process of integrating the model into an existing production environment where it can take in an input and return an output. The goal is to make the predictions from the trained machine learning model available to others. In order to accomplish this task, it was needed a way to use the model with the C# environment. By researching on the internet, it was found a small library on a github repository which allows to load the trained model and use it for the inference task [3]. The library makes use of the Microsoft.ML.OnnxRuntime library. Hence the model needs to be converted from the original pytorch format to the ONNX one. This is simply achieved by utilizing the *export.py* script already present in the YoloV5 repository downloaded at the beginning of this project. By utilizing the system described in the previous chapter for invoking the shell command, the script was invoked with the following command

```
    python export.py --weights /Path/to/NeuralNetwork/best.pt
  --include onnx
```

To use the now format correct network, the library required to create a custom class that inherit from a given class called YoloModel. This class is an abstract class with some public Get only properties such as *Width*, *Height*, *Depth*, *Labels*, *Confidence* and so on and so forth. Locating documentation for certain property definitions proved challenging, leading to reliance on comments from the issues GitHub section and insights gathered from discussions on Stack Overflow. Following is the class definition that was formulated in the end:

```
public class YoloCustomModel : YoloModel
    {
        public override int Width => 640;
        public override int Height => 640;
        public override int Depth => 3;
        public override int Dimensions => 7;
        public override float[] Strides => new float[] { 8, 16,
                            32 };
        public override float[][][] Anchors => new float[][][]
        {
            new float[][] { new float[] { 10, 13 }, new float[]
            { 16, 30 }, new float[] { 33, 23 } },
            new float[][] { new float[] { 30, 61 }, new float[]
            { 62, 45 }, new float[] { 59, 119 } },
            new float[][] { new float[] {116, 90 }, new float[]
            { 156, 198 }, new float[] { 373, 326 } }
        };
        public override int[] Shapes => new int[] {80, 40, 20};
        public override float Confidence => 0.20f;
        public override float MulConfidence => 0.25f;
        public override float Overlap => 0.45f;
        public override string[] OutputNames => new string[]
        { "output0" };
        public override bool UseDetect => true;
        public override string Weights => customWeights;
        public static string customWeights;
        public YoloCustomModel()
        {
            base.Labels = new List<YoloLabel>
            {
                new YoloLabel
                {
                    Id = 0,
                    Name = "Pezzo in metallo"
                },
                new YoloLabel
                {
                    Id = 1,
                    Name = "Interruttore"
                }
            };
        }
```

Code 2.3: Custom Yolo model class definition

It can be observed that the customWeights field is not overridden. This is intentional, as it was introduced in relation to the base class to enable the definition of the weights path after the model instantiation. The rationale behind this design is rooted in the flexibility it provides, allowing users to specify the path to the Yolo model dynamically. The original guidance from the publisher advised manually crafting a class that already incorporates the specific path to the Yolo model. This approach facilitates reusability, enabling the same class to be employed for different models with distinct weight file locations. After having defined this, it could be instantiated the YoloScorer object from the library cited above and to perform the inference task it could be invoked its Predict method. It follows an example of the inference on a single image:

```
public void PredictOnSingleImage()
        {
            using (System.Drawing.Image image = System.Drawing.
            Image.FromFile(ImagePath))
            {
                YoloCustomModel.customWeights = NNPath;
                YoloScorer<YoloCustomModel> scorer = new
                YoloScorer<YoloCustomModel>();
                var predictions = scorer.Predict(image);
                Bitmap bitmap = new Bitmap(image);
                for (int i = 0; i<predictions.Count; i++)
                {
                    Rectangle rectangle = new Rectangle((int)
                    predictions[i].Rectangle.X, (int)
                    predictions[i].Rectangle.Y, (int)
                    predictions[i].Rectangle.Width, (int)
                    predictions[i].Rectangle.Height);
                    Yolov5Image = DrawRectangle(bitmap,
                    rectangle);
                }
            }
        }
```

Code 2.4: Inference on single image

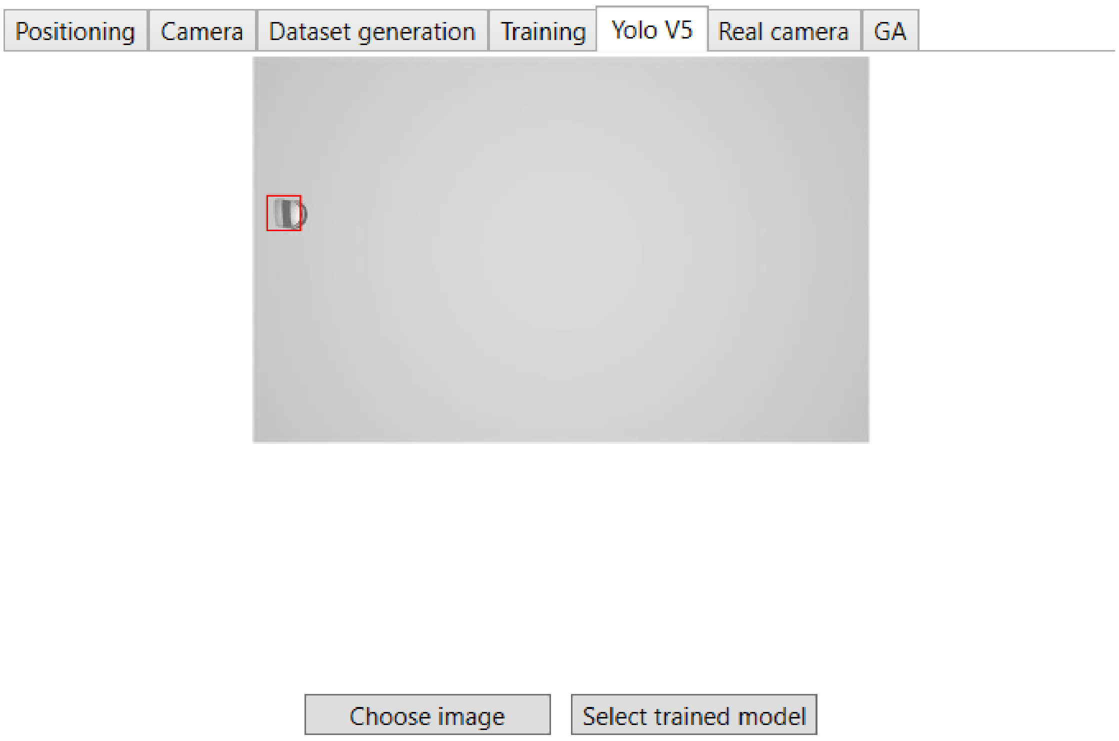


Figure 2.20: Prediction example

2.5 Real camera and real image dataset

The last part concerning the preliminary work revolved around the acquisition of the real images and their labelling.

2.5.1 Image acquisition

In order to acquire the needed images it was provided from Euclid Labs s.r.l. an industrial camera produced by Basler, in specific the *Basler ace 2 a2A1920-51gmBAS* model. Its specifications could be found in its datasheet but the choice of the used camera was not critical for this project for it has only to be made present to the software in order to correctly simulate the camera. For what concerns the optic, it was used the *6mm Focal Length Lens, 1" Sensor Format, #13-755* from Edmund Optics. Also in this case the choice of the optic was not critical. It is crucial to pay attention to the specifications of both the camera body and optics to ensure a more accurate and realistic simulation of the resulting image. For the purpose of connecting the camera to the computer, on which the software runs, it was needed to make use of a power over Ethernet unit, usually denominated as PoE. It consists in a unit that is able in the same time to transmit data between the camera and the computer and to power the camera. As the name implies, it uses an Ethernet cable to connect the twos. Once the setup has been completed, it was time to follow the instruction that the constructor provided. Basler allows to download 2 main software to use their camera: Basler Pylon IP Configurator and Basler Pylon. The first is used in order to view and manipulate the IP address of the camera. This results to be essential since to connect the two unit it is needed to change the computer IP address to 192.168.0.10 and afterward check the status of the camera on the IP Configurator. After the connection has been established it can be opened the Basler Pylon software. This results to be extremely convenient in order to test the camera to check whether there are some spots on the grabbed image meaning that some dirt has deposited onto the lens or onto the sensor itself. Once everything has been checked, then, it was proceeded to acquire the code for the purpose to use the camera directly by this thesis software. Basler proposes a series of samples in order to operate its cameras with C# based software. There is a specific example used for grabbing 10 images that was adapted in order to grab a single image and multiple images at an interval of n seconds where n was specified by the user. Here it follows the first algorithm (the latter is basically the same but with a for loop and a wait between each cycle):

```
public void GrabRealImageCommandHandler()
    {
        using (Camera camera = new Camera())
        {
            // Set the acquisition mode to free running
            // continuous acquisition when the camera is opened
            camera.CameraOpened += Basler.Pylon.Configuration.
            AcquireContinuous;
            // Open the connection to the camera device.
            camera.Open();
            camera.Parameters[PLCamera.ExposureTime].SetValue
            (CameraExposure, FloatValueCorrection.ClipToRange);
            // The parameter MaxNumBuffer can be used to
            // control the amount of buffers
            // allocated for grabbing. The default value of
            //this parameter is 10.
            camera.Parameters[PLCameraInstance.MaxNumBuffer].
            SetValue(5);
            // Start grabbing.
            camera.StreamGrabber.Start();
            // Wait for an image and then retrieve it.
            //A timeout of 5000 ms is used.
            IGrabResult grabResult = camera.StreamGrabber.
            RetrieveResult(5000,
            TimeoutHandling.ThrowException);
            using (grabResult)
            {
                // Image grabbed successfully?
                if (grabResult.GrabSucceeded)
                {
                    RealImage = new Bitmap((int)grabResult.
                    Width, (int)grabResult.Height, System.
                    Drawing.Imaging.PixelFormat.Format24bppRgb)
                    ;
                    BitmapData bmpData = RealImage.LockBits(new
                    Rectangle(0, 0, RealImage.Width,
                    RealImage.Height), ImageLockMode.WriteOnly,
                    RealImage.PixelFormat);
                    // Assuming buffer contains 8-bit grayscale
                    values
                    byte[] buffer = grabResult.PixelData as
                    byte[];
                    int stride = bmpData.Stride;
                    for (int y = 0; y < RealImage.Height; y++)
                    {
                        for (int x = 0; x < RealImage.Width;
                        x++)
                        {
                            byte intensity = buffer[y *
                            RealImage.Width + x];
                            Color color = Color.FromArgb
                            (intensity, intensity, intensity);
                            Marshal.WriteByte(bmpData.Scan0, y
                            * stride + x * 3, color.B);
                            Marshal.WriteByte(bmpData.Scan0, y
                                  * stride + x * 3 + 1, color.G);
                                  Marshal.WriteByte(bmpData.Scan0, y
                                  * stride + x * 3 + 2, color.R);
                             }
                        }
                        RealImage.UnlockBits(bmpData);
                   }
                   else
                   {
                        Console.WriteLine("Error: {0} {1}",
                        grabResult.ErrorCode,
                        grabResult.ErrorDescription);
                   }
              }
              // Stop grabbing.
              camera.StreamGrabber.Stop();
              // Close the connection to the camera device.
              camera.Close();
         }
    }
```

Code 2.5: Code snippet for grabbing images with the camera

The code above is used to acquire an image from the camera. In the user interface it is placed a button for taking single images and multiple images. The first one is useful in order to set the exposure to have brighter or darker images and to prepare the setup for taking picture. The second is used after for creating a specific dataset of images. After pressing the second button, in fact, it is asked to choose a folder where to save the current dataset and at intervals of some seconds a chosen number of picture are taken.

2.5.2 Image labelling

After having generated the real image dataset it was time to also create the associated label txt file to build the ground truth for the experiments. To achieve this, the YoloV5 GitHub repository suggested to make use of the Roboflow online software [7]. Basically, it is a website developed on purpose for creating dataset for image classification and object detection for Yolo algorithms. In the context of this project this was used to utilize its labelling function. The software

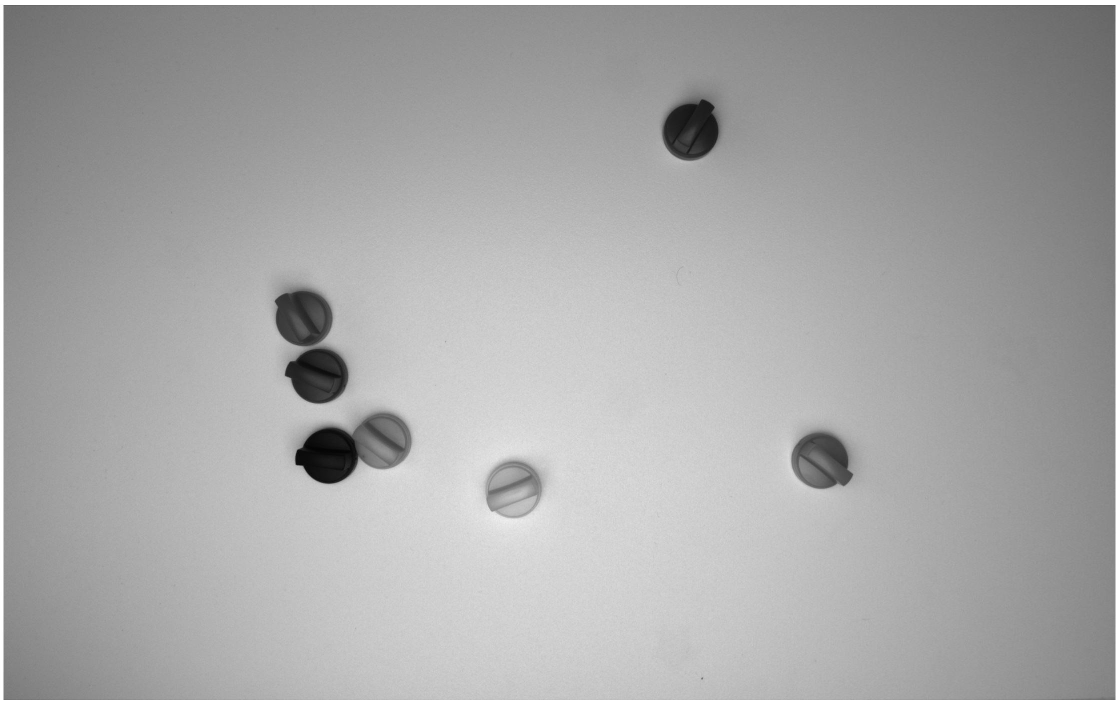


Figure 2.21: Example of image taken

requires to upload the custom images and, for creating the annotation, onto each image drag a square of the dimension of the object in question. It is possible to select also different labels for each object but in this case the object considered was just one. Before generating the labels file, Roboflow allows some type of image augmentation and resize. In this first phase only the resize of the image was chosen. Hence the output were 640*640 pixel images and their associated txt file.

2.6 Other consideration on the preliminary work

Before proceeding it is mandatory to mention some aspects that were modified in order to obtain better results by the software. By speaking with Mr. Polesel, CEO of Euclid Labs s.r.l., it was noticed that using squared images obtained by resizing the images could lead to the lost of proportions in object that don't present a specific form of symmetry. This could cause loss of information, for example a rectangular object does not maintain its "rectangularness" if, when the image is shrunk, the sides become comparable. To avoid this issues, the solution presented was to enlarge the image size so that the height of the image

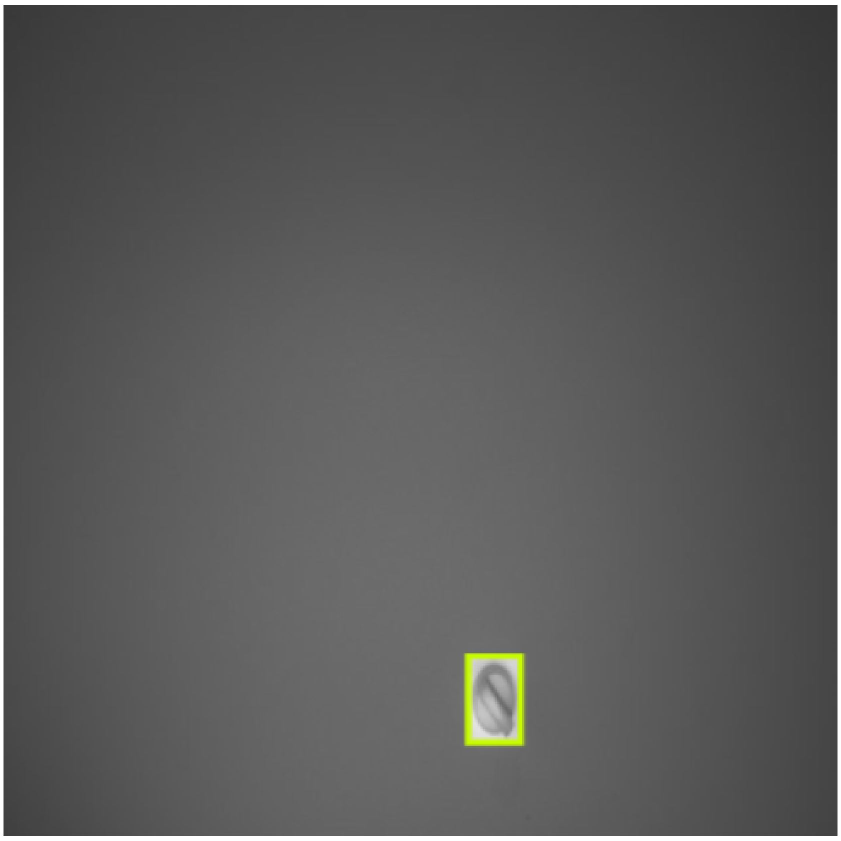


Figure 2.22: Image labelling

matched its width of 1920 pixel. To fill the new portion then it was decided to set all the new pixel to 255, hence making them white. Lastly all the images had to be resized to a dimension of 640 pixel per dimension. An example of this outcome is showed in the image 2.23. Secondly, it was evident from the beginning that the computational power of the computer on which this project was developed was too low to perform the training of YoloV5 for hundreds of epoch. A solution was found using the Google Colab script made available by Roboflow. It basically allowed to use NVIDIA T4 Tensor Core GPUs that made the training process extremely faster compared to the previous method. The new code, implemented in Python, essentially starts with installing the necessary libraries onto the virtual environment, as it was done also in this project. Afterwards, it connects the environments to the user google drive in order to get the access to the images and their respective labels. Once that part is completed, it runs the train.py python script to start the training of the YoloV5 network. A small modification of the YAML file has been made to correct the position of the training test and validation folder location for the use of the script on Google

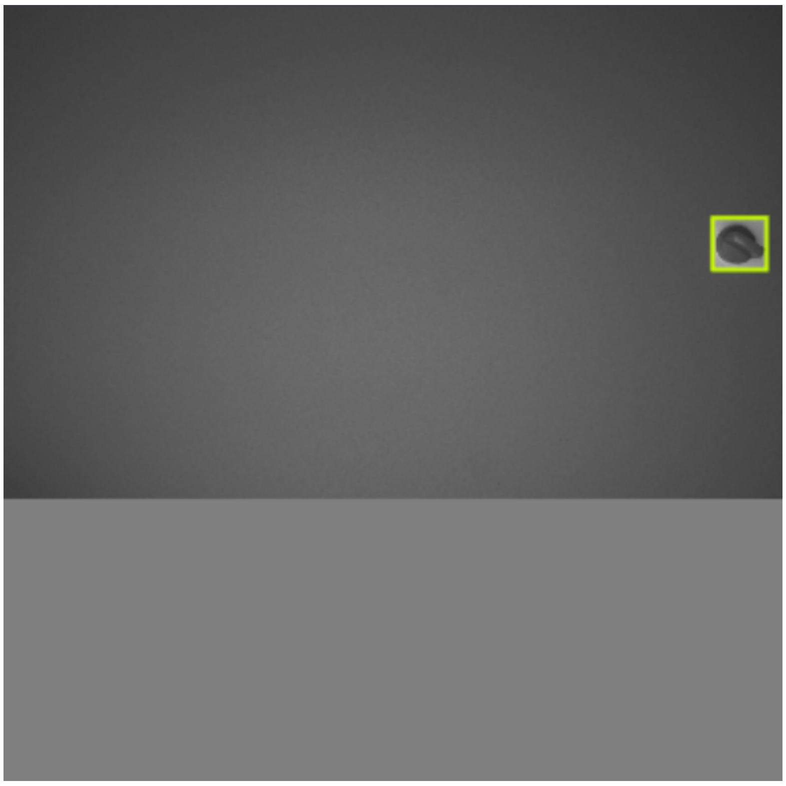


Figure 2.23: New image for YoloV5 training

Colab. Then some plots are shown to the user in order to have a first glance to the quality of the trained network. These plots contain information about the loss function on both train and validation dataset, and some metrics computed onto the validation dataset (an example in figure 2.24). Lastly it is tested on a unseen test dataset of real images to assess the network capability of generalization.

```
# clone YOLOv5 repository
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
!git reset --hard 064365d8683fd002e9ad789c1e91fa3d021b44f0
# install dependencies as necessary
!pip install -qr requirements.txt  # install dependencies (
    ignore errors)
import torch
from IPython.display import Image, clear_output  # to display
    images
from utils.downloads import attempt_download  # to download
    models/datasets
# clear_output()
```

```
print('Setup complete. Using torch %s %s' % (torch.__version__,
    torch.cuda.get_device_properties(0) if torch.cuda.
   is_available() else 'CPU'))
from google.colab import drive
# Mount Google Drive to the '/content/drive/' directory
drive.mount('/content/drive')

generation=0
custom=1
dataset=f"/content/drive/MyDrive/Tesi/Gen{generation}/custom{
   custom}"
%cd /content/yolov5/
!python train.py --img 640 --batch 32 --epochs {epochs} --data
   {dataset}/data.yaml --cfg /content/yolov5/models/yolov5s.
   yaml --weights '' --name yolov5s_results_Gen{generation}
   _custom{custom}  --cache
```

Code 2.6: Python code for training YoloV5

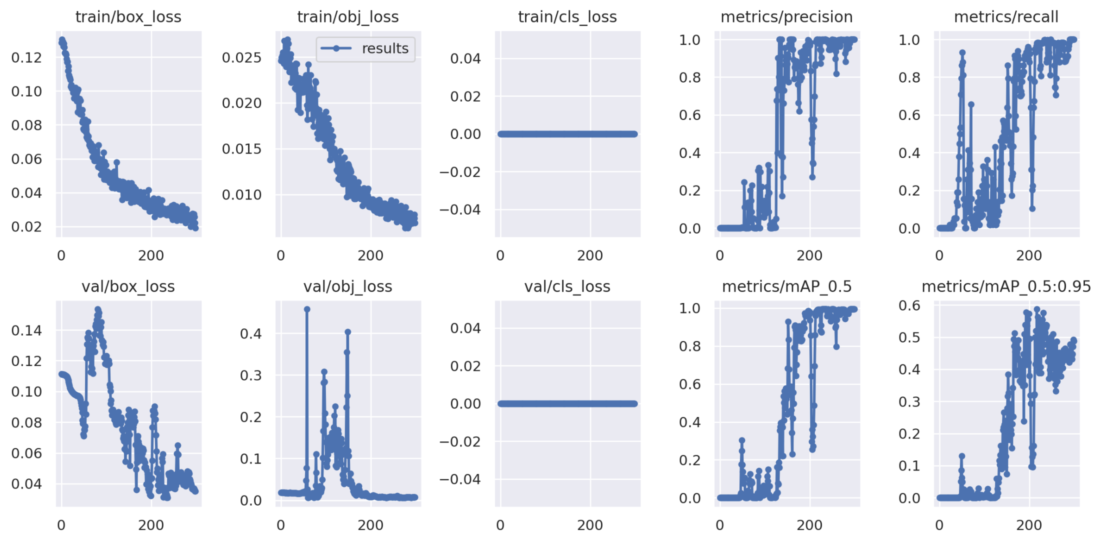


Figure 2.24: Output graphics of training process

```
%cd /content/yolov5/
custom=1
!python val.py --weights runs/train/yolov5s_results_Gen{
   generation}_custom{custom}/weights/best.pt --img 640 --data
   {dataset}/data.yaml --task test
```

Code 2.7: Python code for validating trained model

3 Tests, results and future works

In this chapter it is presented the results of the training and how the parameters of the simulation were changed in order to obtain better results when training the networks with synthetic images.

3.1 Genetic algorithm

3.1.1 General Genetic Algorithm

This section explores the application of Genetic Algorithms (GAs) as a powerful optimization technique. Genetic Algorithms draw inspiration from the principles of natural selection and genetics, providing a versatile and robust methodology for solving complex optimization problems. Genetic Algorithms have emerged as a popular optimization method due to their ability to mimic the evolutionary processes observed in nature. Inspired by Darwinian principles, GAs harness the power of genetic recombination, mutation, and selection to iteratively evolve a population of potential solutions towards optimal or near-optimal solutions. Traditional optimization methods may face challenges when dealing with complex, nonlinear, and multi-dimensional problem spaces. Genetic Algorithms offer a promising alternative by providing a flexible and adaptive approach to search for solutions in such challenging domains. In the context of Genetic Algorithms, a solution to the optimization problem is encoded as a chromosome, which consists of genes representing different parameters or variables. The structure of chromosomes is crucial in determining the potential solutions within the population. A GA maintains a population of candidate solutions. Through successive generations, these solutions undergo genetic operations such as crossover and mutation, creating diversity and exploration in the search space. The fitness function evaluates the performance of each solution in the population, assigning a numerical value that represents the solution's quality. Genetic Algorithms utilize the fitness values to guide the evolutionary process, favoring solutions that contribute positively to the objective. As said,

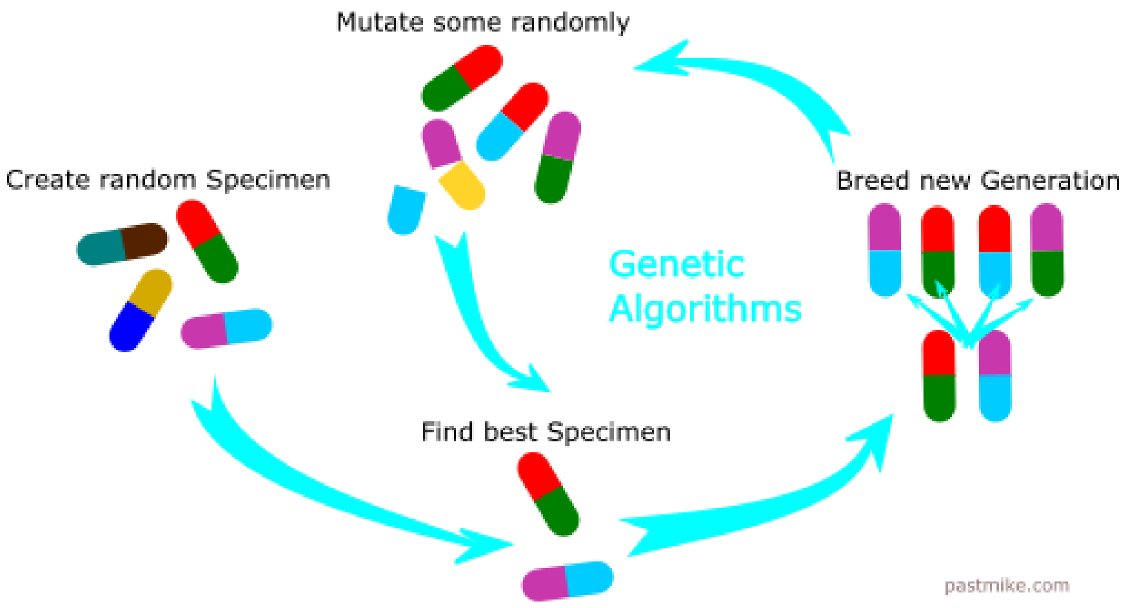


Figure 3.1: Genetic algorithm scheme

two different genetic operation are performed on each individual:

- Crossover, or recombination, involves exchanging genetic information between parent solutions to produce offspring. This operation introduces exploration by combining promising features from different solutions, potentially yielding superior offspring;
- Mutation introduces random changes to individual genes within a chromosome. This operation promotes exploration by introducing small, random modifications to the solutions, preventing premature convergence to suboptimal solutions.

The optimization process begins with the creation of an initial population of solutions. Each solution is represented by a chromosome, and the population is evaluated using the fitness function. Solutions are selected to serve as parents for the next generation based on their fitness values. Higher fitness increases the likelihood of being selected, emulating the natural selection process. The selected solutions undergo genetic operations (crossover and mutation) to produce offspring. The offspring forms the next generation of the population. The optimization process continues for a predefined number of generations or until a convergence criterion is met. The algorithm terminates when a satisfactory solution or a predefined stopping condition is achieved.

3.1.2 Implementation of a genetic algorithm in the software

The problem that needed to be addressed was how to optimize the simulation parameter in order to generate images that could be used to efficiently train a computer vision algorithm such as YoloV5. Therefore the last step to accomplish was to find a method that allowed the modification of the parameters in a sensible manner. This could be stated as an optimization problem where it was needed

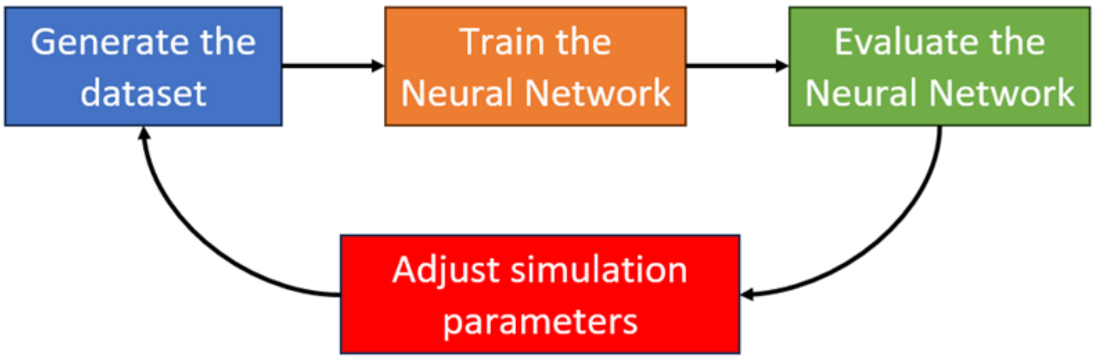


Figure 3.2: General workflow for the experiments

to define the parameters to change and a score function to evaluate the dataset generated with a specific set of parameters. For what concerns the parameters chosen, they are the following:

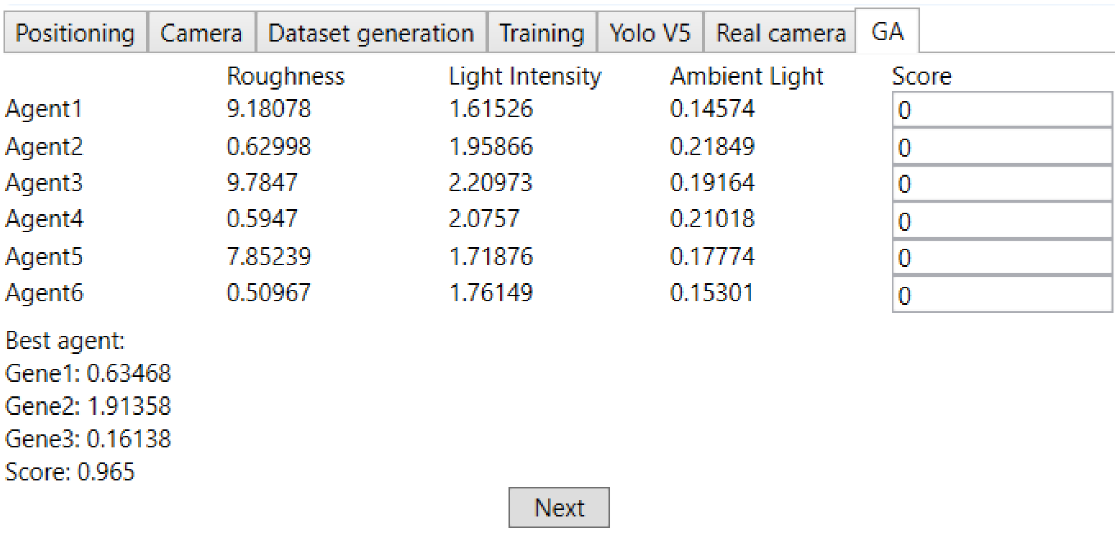
- Object specular power/roughness: the property of the object that express its behaviour when reflecting light. As explained in the first part of this thesis the higher the specular power the more the object act as a perfect mirror;

- Light intensity: this parameter is connected to the irradiance of the lamp used when the real dataset was acquired. Since there is not a real connection between the power of the real light and the light intensity used on the simulation, this could be used as parameter to calibrate the scene;
- Ambient light: this parameter is fundamental since the real dataset was taken in a space where other light source were present such as windows and other lights. Though they were less relevant compared to the main light used, they need to be taken into account in the simulation with the ambient light parameter. This parameter can take a value between 0 and 1 and it basically create a directional light of different intensity.

Hence each and every chromosome was composed of the genes listed above. The tests were organized in the following manner:

1. Given the chromosome generate the synthetic dataset with images and labels;
2. Train the network with the generated dataset as train set and a portion of the real dataset as validation set;
3. Evaluate the algorithm best performance on the rest of the real dataset.

The first thing to do was to produce the code that allowed to replicate the genetic algorithm in C#. To accomplish this, a class named Agent was created. Its constructor did not accept any inputs and set the genes as a random number chosen between a maximum and a minimum. This class presents three public fields, one for each gene, a breed method and a mutate method. The breed method is called when it is necessary to generate a children from 2 agent parents. Hence, for each gene it is generated a random number between 0 and 1 and if the result is lower than 0.5 then the children gene is inherited from the first chromosome, if it is higher than 0.5 then it is the other way around. After the inheritance of the parents genes, it was time for using the mutation method. It consists into apply a small change to one of the genes of the chromosome. In order to choose which gene has to change, it was generated a integer random number belonging to the set {0, 1, 2}. If the outcome was 0 then the roughness was changed; if it was 1 the ambient light; if 2 the light intensity. Once chosen the gene to mutate, it was increased or decreased by a random percentage bounded between -20% and +20%. It was decided to instantiate six objects of that class. This number was chosen as a trade off between diversity and time constraint since it was expected a huge time to train each neural network with each dataset. Once the new generation of chromosomes was created, a new dataset of synthetic images was generated. This process exploited the functions and methods defined in the chapter 2 in order to automatically to automatically generate the labels file for the images. For each dataset it was chosen to generate 50 images of the piece in question. This was a rule of thumb but this number can be decreased or increased based on the complexity of the piece photographed. Obviously, also in this case, the more the training images, the higher the time to train the models. On the side of the user interface, the GUI presented itself in this manner:



	Roughness	Light Intensity	Ambient Light	Score
Agent1	9.18078	1.61526	0.14574	0
Agent2	0.62998	1.95866	0.21849	0
Agent3	9.7847	2.20973	0.19164	0
Agent4	0.5947	2.0757	0.21018	0
Agent5	7.85239	1.71876	0.17774	0
Agent6	0.50967	1.76149	0.15301	0

Best agent:
Gene1: 0.63468
Gene2: 1.91358
Gene3: 0.16138
Score: 0.965

Figure 3.3: Genetic algorithm UI

hence for each agent were presented its relative genes and had to be prompted manually the individual score. Once started the user interface presents all its fields empty. It is needed to press the *Next* button in order to generate the dataset. Moreover, before pressing the *Next* button it is needed to select the object which is needed for the dataset generation. This is due to the fact that

the scene could be composed of multiple object such as the object of interest, a plane that emulates the plane on which it is placed or a conveyor belt in the case the object detection algorithm is used in an automatic machine. In the specific case of this project, in fact, for simplicity of use, when starting the program the scene was initialized with the reference object, a light having the dimensions of the real one used for acquiring the real dataset and a plane with specular power calibrated with the one of the real experiment.
Upon pressing the *Next* button for the first time, six datasets are generated to allow the training of the YoloV5 algorithm. These datasets play a crucial role in the subsequent training process. Once the training completes, the scores corresponding to the performance of each Agent are prompted in the designated score section of the user interface. Subsequently, when the *Next* button is pressed again, the system initiates the back-end process. This involves ranking the Agents based on their respective scores. The top two Agents in this ranking are then selected for breeding, giving rise to the generation of new offspring. It's important to note that no specific terminal condition is defined within the system. Instead, the decision to conclude the evolutionary process is left to the user's discretion. The user is empowered to determine when they believe further iterations would yield marginal or no improvements. This flexible approach allows users to exercise their judgment and bring the evolutionary process to a close when they deem it appropriate. In summary, the iterative cycle involves training the YoloV5 algorithm, evaluating Agent performance, and then selectively breeding Agents for subsequent generations. The absence of a predetermined terminal condition empowers users to decide when to conclude the evolutionary process based on their assessment of the system's performance and the potential for further improvement.

3.2 Technical issues

Throughout the development phase of this specific software module, various challenges were encountered, each requiring resolution with the available knowledge and resources at hand.

3.2.1 Random number generator

First of all it was noticed that whenever it was tried to generate a random number, often the outcomes resulted to be the same. In fact, to draw a random number from a uniform distribution between two number, it was used the Random class which requires to instantiate a Random object usually called random or rnd [4]. Once instantiated, it can be called the method *Next(Int32, Int32)* in order to get a random integer that is within a specified range or the method *NextDouble()* that returns a random floating-point number that is greater than or equal to 0.0, and less than 1.0. The first method was used to draw a number from the set {0,1,2} in order to decide which gene to mutate while the second method was employed to draw a number from 0 to 1 for choosing from which parent a gene has to be inherit and how much a gene had to be increased or decreased. The Random class, though, represents a pseudo-random number generator, which is an algorithm that produces a sequence of numbers, starting from a certain number called *seed*, that meet certain statistical requirements for randomness. Hence, as intuition leads to assume, each generated number is in a certain way connected to the previous and the next. The problem was indeed attributable to the use of this class. Searching on the internet, an user responding to a question on stackoverflow explained that when instantiated, Random class takes seed values from your CPU clock which is very much predictable. The correct usage of this class, hence, was to define a single random object and to use it as needed. In this way sampling from the pseudo random variable guaranteed to get result different one from another. To ensure a greater randomness, though, it was implemented another method: the .Net framework makes available the RNGCryptoServiceProvider class. This class implements a cryptographic Random Number Generator (RNG) using the implementation provided by the cryptographic service provider (CSP). This class differs from the previous by the choice of the seed for the pseudo random sequence. In fact, the class uses OS entropy to generate seeds. OS entropy is a random value which is generated using sound, mouse click, and keyboard timings, thermal temp etc. Below, an example of the usage of this class for generating a random Int 32 number with range {-2 147 483 648; 2 147 483 647 }

```
using (RNGCryptoServiceProvider rg = new
   RNGCryptoServiceProvider())
{
    byte[] rno = new byte[4];
    rg.GetBytes(rno);
    int randomvalue = BitConverter.ToInt32(rno, 0);
}
```

Code 3.1: Random Number Generator

The code in 3.1 was then taken and modified in order to produce a double that could take values in the interval [0,1] using the following method of the Agent class:

```
    static double GetRandomDouble()
        {
            using (RNGCryptoServiceProvider rg = new
            RNGCryptoServiceProvider())
            {
                byte[] rno = new byte[4];
                rg.GetBytes(rno);
                int randomvalue = BitConverter.ToInt32(rno, 0);
                randomvalue = Math.Abs(randomvalue % 10001);
                double randomvaluedouble =
                (double)randomvalue / 10000D;
                return randomvaluedouble;
            }
        }
```

Code 3.2: Double random generator

In the course of enhancing the evolutionary algorithm, a deliberate decision was made to focus on the method for generating random double numbers exclusively. This choice was driven by the concurrent modification of the mutate class, which was specifically tailored to mutate each gene within every chromosome each time a breeding operation occurred between two chromosomes. The rationale behind this modification was rooted in the desire to enhance the exploration parameter within the exploitation versus exploration dilemma. Given the limited number of agents available, a comprehensive approach was adopted to systematically mutate genes during each breeding iteration. The strategy employed involved the meticulous tracking of the best-performing agent throughout the evolutionary process. This strategic oversight allowed for the retention of the best agent's genetic makeup. Consequently, when newly bred agents yielded suboptimal results in comparison to the established best performer, the superior genetic configuration could be preserved. This adaptive mechanism ensured that the evolutionary algorithm retained a robust ability to navigate the complex landscape of the optimization problem, striking a balance between exploiting existing solutions and exploring potential improvements. The synergy between the refined random number generation method and the comprehensive gene mutation approach contributed to the algorithm's adaptability and efficacy in addressing the challenges posed by the limited pool of available agents.

3.2.2 Async methods

The dataset generation process employed a sequential approach, starting with the definition of genetic information for each agent, encompassing parameters such as roughness, light intensity, and ambient light. Subsequently, these parameters were applied to the scene, initiating a sequence of steps involving the positioning of an object with random orientation on the z-axis. A simulated camera captured images of the scene at each step, with the object being successively moved to different positions. The problem encountered, though, was that if the code was made running synchronously, when changing the parameters of the simulation or the position of the object, it would have left no time to the GPU to process the request that another change had to be made. Moreover the user interface would have not update itself since the main thread was being occupied by the algorithm working in the back end. In order to address this problem, it was made use of the Task.Run() method. This method, intrinsic to the Task Parallel Library (TPL) in C#, serves as a pivotal construct for asynchronous task execution within the programming paradigm. Enabling the concurrent execution of operations, it is specifically designed to offload computationally or I/O-intensive tasks from the main thread, fostering parallelism and responsiveness. Within the context of the broader Task Parallel Library, a Task represents an asynchronous unit of work, encapsulating operations that can be executed independently. The ThreadPool, an essential element of the runtime environment, manages a pool of worker threads, minimizing the overhead associated with thread creation and promoting efficient multitasking. Task.Run() encapsulates a succinct mechanism for initiating and dispatching a new Task on the ThreadPool. Its usage involves providing a delegate, often expressed through an anonymous function or a lambda expression, delineating the code to be executed asynchronously. This method facilitates the concurrent execution of the specified code block, allowing the main thread to proceed with its tasks without being obstructed by the asynchronous operation. Without knowing about how to operate with this tool, a *datasetGenerated* boolean variable was declared. It was set false before the Task.Run() method and inside it, after having generated the dataset, was converted to true. After the Task.Run() was a while(!datasetGenerated); loop. The problem was that sometimes this code wasn't able to escape the while loop. After some researches it was found out that asynchronous programming made also use of the *await* and *async* keywords. The async modifier, when applied to a method, signals the compiler to transform the method into an asynchronous operation. This enables the method to execute asynchronously, freeing the calling thread to concurrently perform other tasks. An *async* method typically returns a Task or Task<T>, indicating the ongoing execution of the asynchronous operation. The *await* keyword, employed within an async method, acts as a suspension point. It allows the asynchronous method to pause its execution until the awaited task is complete. Crucially, during this pause, the calling thread remains unblocked, contributing to a responsive user interface and efficient resource utilization. The previous code then was changed and the dataset generation method was converted to an async function while the await keyword was used each time the Task.Run() method occurred.

```
async public Task GenerateDataset()
    {
        await Task.Run(() =>
        {
            GenerateDatasetFromAgent(Agent1);
        });
        .
        .
        .
    }
```

Code 3.3: Use of the Task.Run() method and async and await keyword

3.3 Test and results

After having completed all the previous steps it was time to train the Neural Network onto the dataset generated. Hence for each generation the dataset have been uploaded onto google drive in order to make them available for google colab and sequentially the neural network have been trained for each agent. Each neural network had been trained for 300 epochs and each took from 15 to 20 minutes to complete. For the first generation the following results have been obtained:

Table 3.1: First generation results

Gen 1	Roughness	Light intensity	Ambient light	mAP@0.5
Agent1	0.611	2.4211	0.66808	0.939
Agent2	2.065	2.2166	0.28464	0.612
Agent3	8.823	1.6977	0.1528	0.985
Agent4	1.409	2.2803	0.13632	0.938
Agent5	1.723	1.9915	0.4012	0.962
Agent6	1.575	2.1336	0.12424	0.888

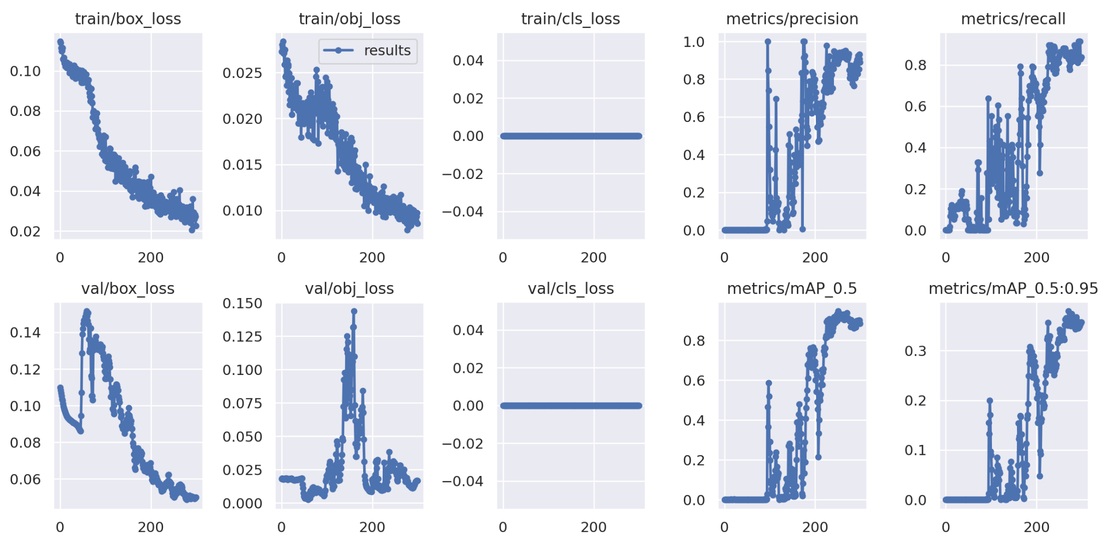


Figure 3.4: Training Generation 1, Agent 1

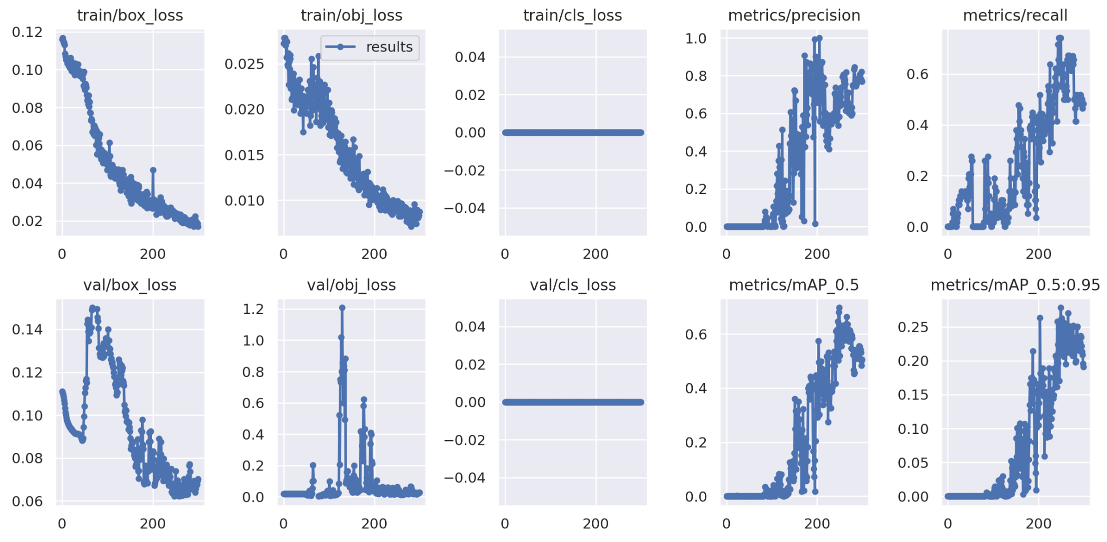


Figure 3.5: Training Generation 1, Agent 2

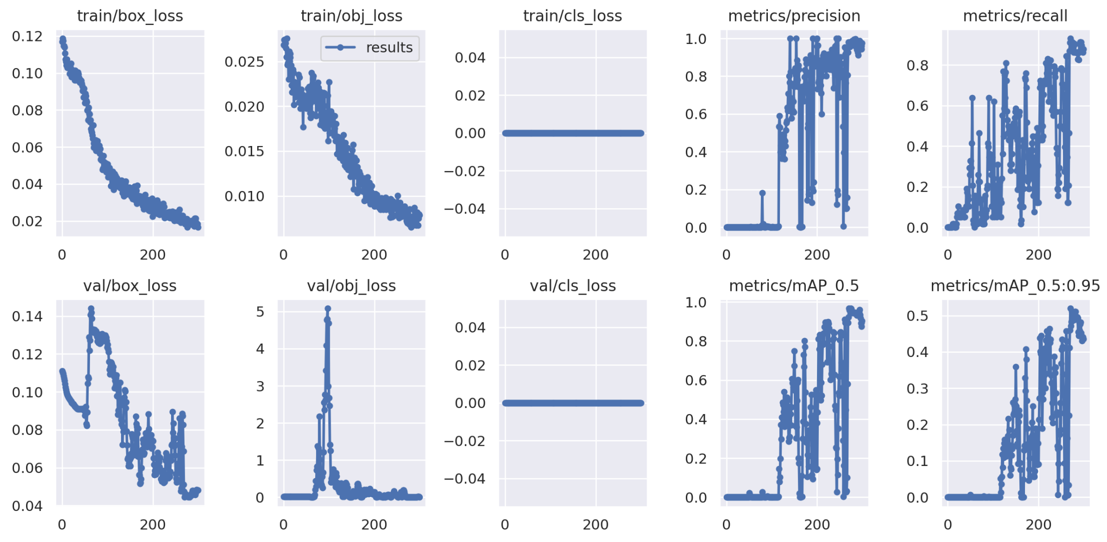


Figure 3.6: Training Generation 1, Agent 3

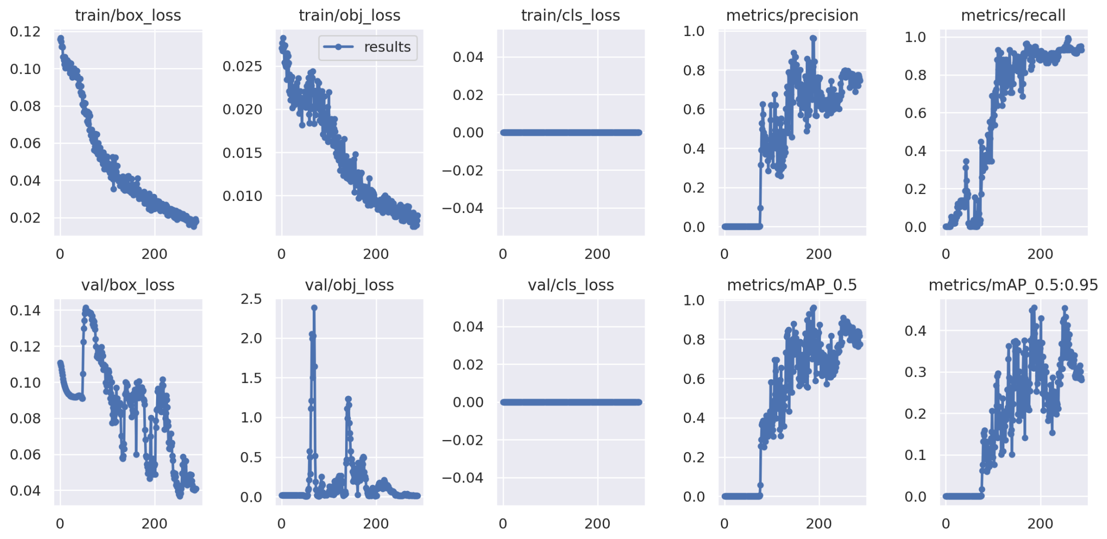


Figure 3.7: Training Generation 1, Agent 4

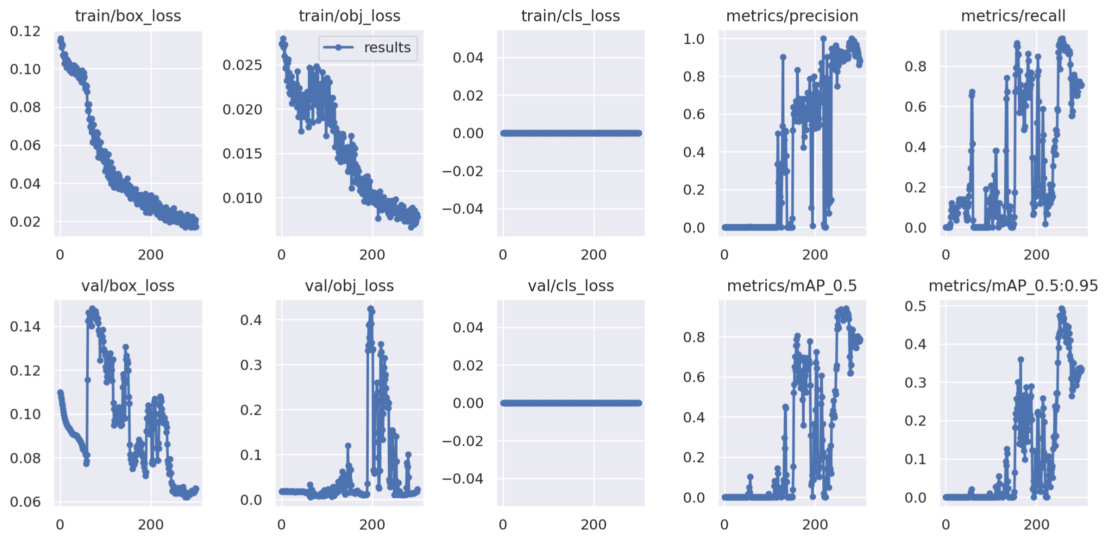


Figure 3.8: Training Generation 1, Agent 5

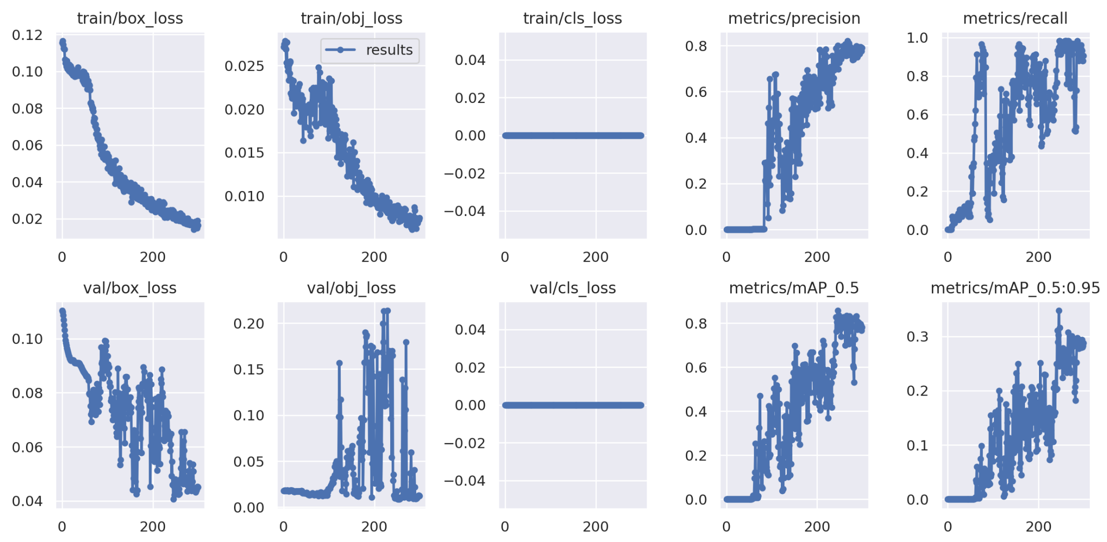


Figure 3.9: Training Generation 1, Agent 6

From the results in the table above, the two agents chosen as parents were Agent 3 and Agent 5. The genetic algorithm then produced 6 new agents starting from the new parents. Another wanted peculiarity was that, for what regards the graph, as the training loss decreased also the mAP@0.5 and mAP@0.95 increased. In a practical scenario, the absence of a validation dataset comprising real images necessitates the determination of the best.pt weight file based on the synthetic validation set's optimal performance. The aim in fact was for this part was to calibrate the scene with proper parameters. In the tables 3.2, 3.3, 3.4 are the next generation results and in figure 3.10, 3.11. 3.12, 3.13, 3.14, 3.15 the results of the training of the last generation.

Table 3.2: Second generation results

Gen 2	**Roughness**	**Light intensity**	**Ambient light**	**mAP@0.5**
Agent1	10.34479	1.61336	0.72512	0.286
Agent2	9.09616	1.51143	0.16777	0.944
Agent3	0.52155	1.7159	0.79608	0.093
Agent4	7.17204	1.44298	0.17908	0.217
Agent5	0.60345	1.87562	0.18187	0.987
Agent6	0.65895	1.99305	0.56175	0.498

Table 3.3: Third generation results

Gen 3	Roughness	Light intensity	Ambient light	mAP@0.5
Agent1	10.02215	1.90923	0.13445	0.683
Agent2	0.63468	1.91358	0.16138	0.965
Agent3	0.55822	1.21604	0.17715	0.952
Agent4	9.3312	1.6726	0.18751	0.96
Agent5	9.42144	1.54214	0.15194	0.167
Agent6	7.50506	2.18562	0.18302	0.885

Table 3.4: Fourth generation results

Gen 4	Roughness	Light intensity	Ambient light	mAP@0.5
Agent1	0.5635	1.52862	0.15404	0.994
Agent2	10.34457	1.74894	0.19796	0.995
Agent3	9.8177	1.75663	0.15361	0.992
Agent4	9.60069	1.65487	0.16286	0.994
Agent5	0.54895	1.98919	0.1316	0.995
Agent6	0.61206	1.59392	0.1843	0.942

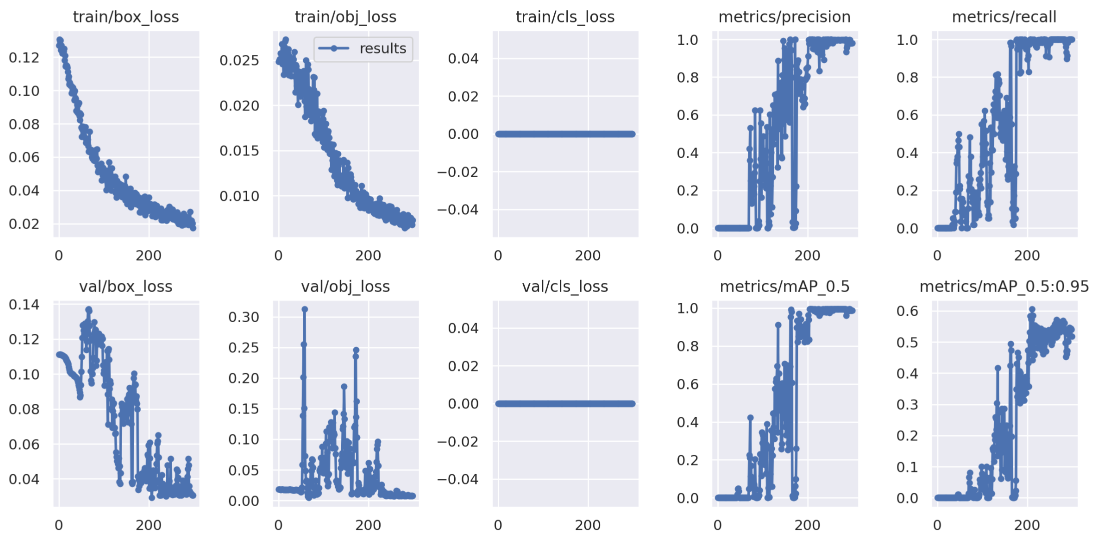


Figure 3.10: Training Generation 4, Agent 1

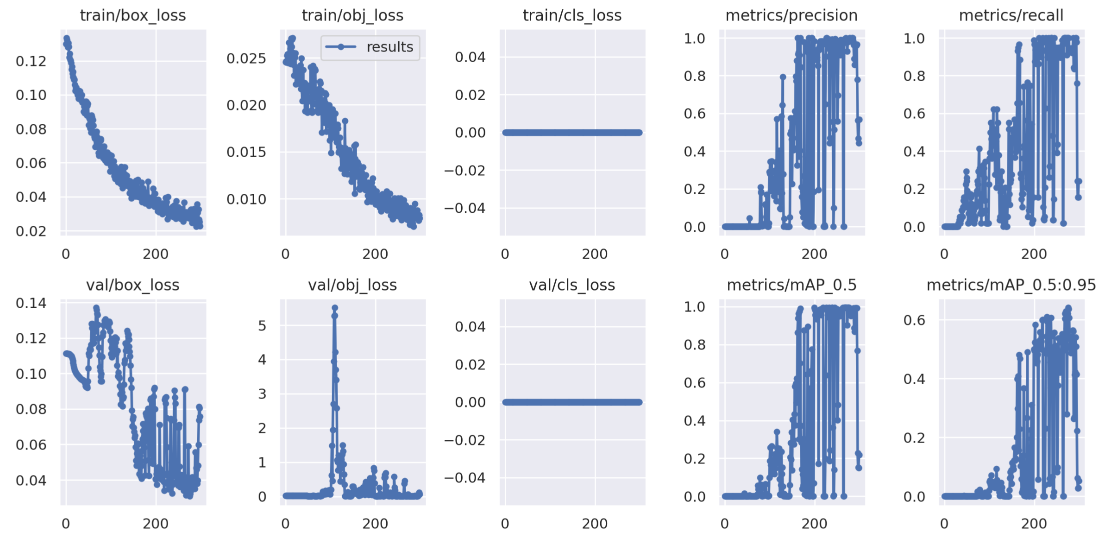


Figure 3.11: Training Generation 4, Agent 2

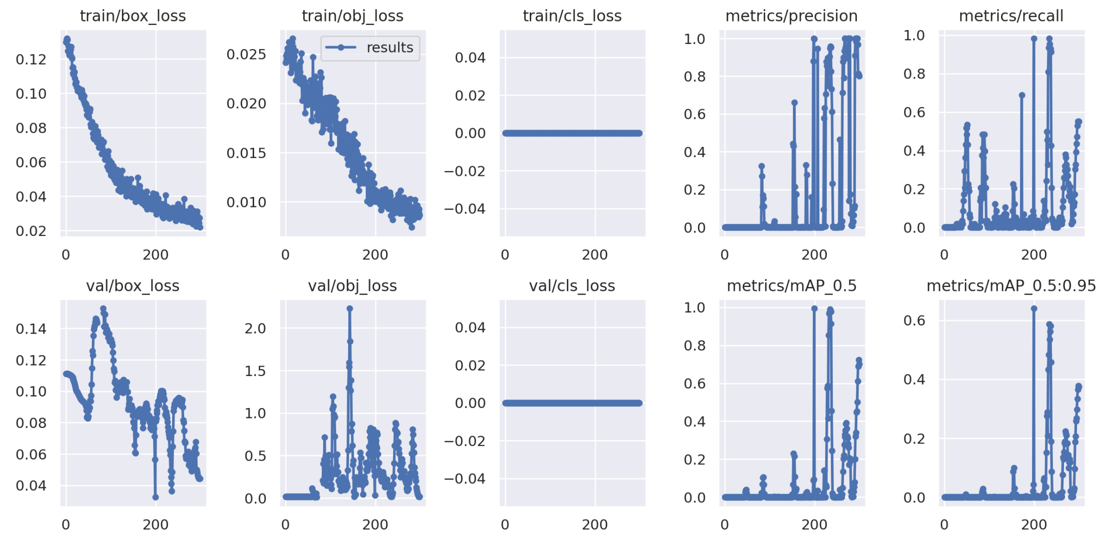


Figure 3.12: Training Generation 4, Agent 3

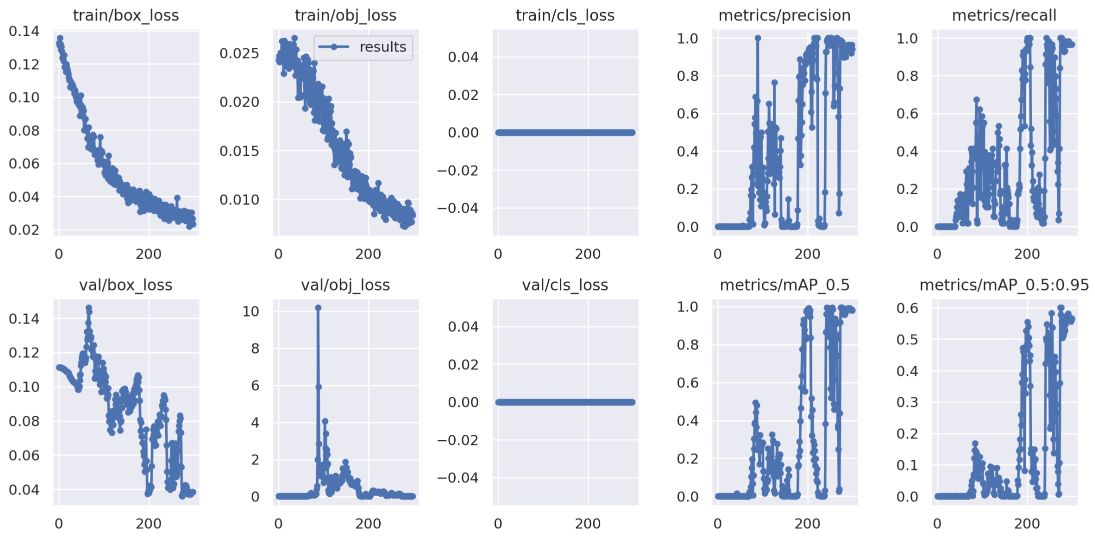


Figure 3.13: Training Generation 4, Agent 4

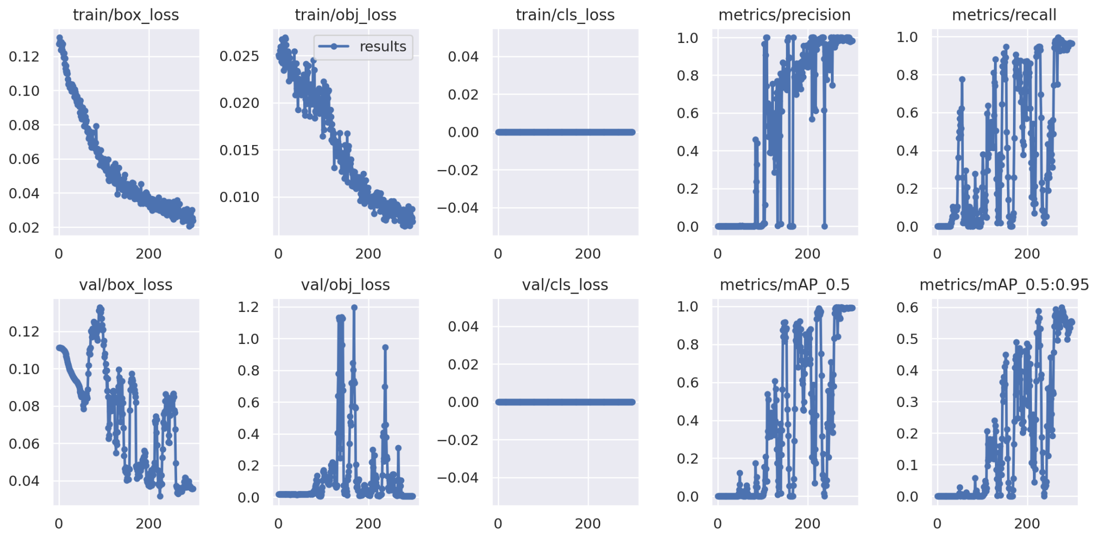


Figure 3.14: Training Generation 4, Agent 5

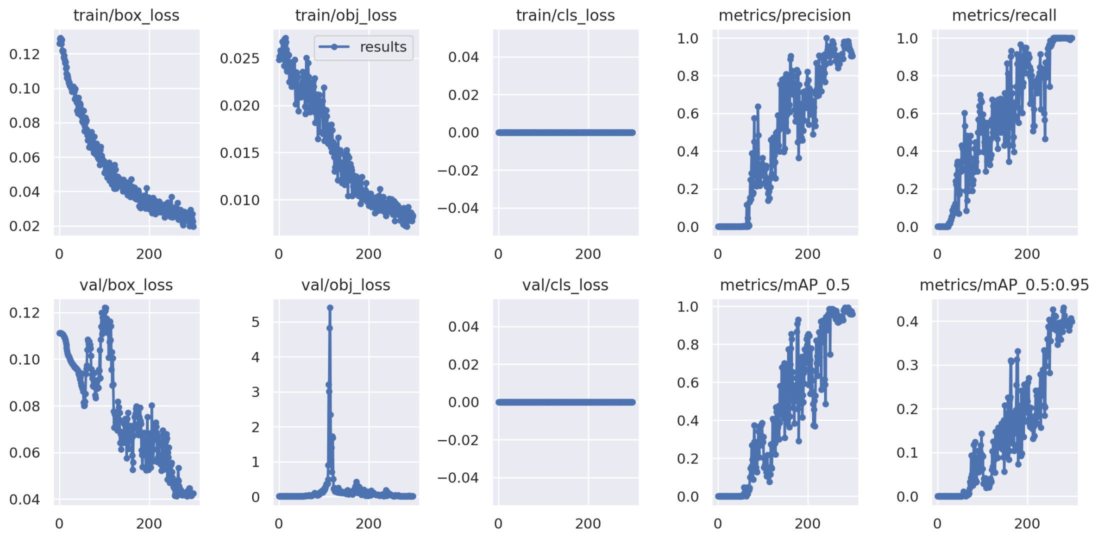


Figure 3.15: Training Generation 4, Agent 6

One thing to notice is that, although the train losses in general decrease smoothly, the validation losses tends to flutter, hence the metrics tends to have a great variance. This is due to the fact that by decreasing the train losses the model tends to generalize less and hence there could be some epochs in which the model tends to overfit.

After the previous tests were completed, it was decided to try with another method. In this case the Neural Network was still trained with synthetic images but for what concerns the metrics it was decided to use also a synthetic validation dataset. The train validation split in this case was set to 70% train and 30% validation with 70 images for training and 30 for validation. This is a more real scenario since everything in this case was simulation made. After that the network was trained, it was tested onto a real dataset of 100 labelled real images to evaluate its quality. The results obtained were quite surprising since even from the first generation of images the performances of some neural networks were exceptionally good. In the table 3.5 are presented the parameters with the score of the NN with best weights for each dataset on the real data:

Table 3.5: First generation results with new method

Gen 1	Roughness	Light intensity	Ambient l.	mAP@0.5	mAP@0.95
Agent1	3.838	1.8612	0.18626	0.995	0.593
Agent2	5.295	2.8414	0.21320	0.912	0.507
Agent3	6.564	1.3628	0.26082	0.933	0.457
Agent4	9.452	2.6716	0.16424	0.979	0.452
Agent5	2.301	1.1222	0.09778	0.995	0.584
Agent6	2.560	1.6042	0.09532	0.989	0.553

3.4 Comments on the result and future works

The results of the rendering algorithm used to train the YoloV5 network in this setup are astonishing with a mAP@0.5 over 0.99 in the best case. It could be seen that in some cases such as the agents 1 and 5 from the first generation of the second method, there is a balance between the light intensity of the lamp and the general ambient light for obtaining good results. In fact in the first case it is present a higher light intensity with lower ambient light while in the second case it is the vice versa. From the tests with the first method it is evident that the roughness plays a minor role with respect to the lighting and this is reasonable. Overall, the results of the every test were very good and for the first method it is shown that genetic algorithm could be a valid optimization method for this job. Due to time and spatial constraints it was not possible to test the networks in more complex environment hence it is possible to suppose that the NN could likely behave well in controlled and simple scenarios like the one used in this project.

In terms of future work, it is imperative to subject the algorithm to more complex scenarios to comprehensively evaluate its efficacy. Furthermore, expanding the project to facilitate model training directly on the machine where the software is deployed, rather than relying on Google Colab, is essential. Unfortunately, the hardware demands proved too burdensome for my personal computer, resulting in impractical training times.

Addressing the code is also paramount. Given my limited experience with large-scale coding projects, a thorough review is necessary. Workflow and structural aspects require optimization to enhance efficiency and maintainability. This

might entail revisiting design patterns, modularization, and ensuring adherence to coding best practices.
In summary, future efforts should focus on rigorous testing in diverse scenarios, transitioning to local model training, and refining the codebase through comprehensive review and optimization. These endeavors are vital for the continued advancement and success of the project.

References

[1] DataScience. *what does the notation map 0.5 0.95 mean*. 2017. URL: https://datascience.stackexchange.com/questions/16797/what-does-the-notation-map-5-95-mean.

[2] Jonathan Hui. *map mean average precision for object detection*. 2018. URL: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173.

[3] kkarakhainko. *Yolov5Net*. 2021. URL: https://github.com/techwingslab/yolov5-net.

[4] Microsoft. *System.Random*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.random?view=net-8.0.

[5] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[6] De Zen Riccardo. "Image acquisition systems simulation techniques for computer vision algorithms' training and validation". MA thesis. Università degli studi di Padova, 2022.

[7] Roboflow. *Roboflow*. 2020. URL: %https://roboflow.com/.

[8] Juan Terven and Diana Cordova-Esparza. "A comprehensive review of YOLO: From YOLOv1 to YOLOv8 and beyond". In: *arXiv preprint arXiv:2304.00501* (2023).

[9] Ultralytics. *YOLOv5 custom train*. 2024. URL: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data.

[10] Ultralytics. *YOLOv5 custom train - train on custom data*. 2024. URL: https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#train-on-custom-data.

[11] Wikipedia. *Blinn-Phong model*. 2024. URL: https://en.wikipedia.org/wiki/Blinn%E2%80%93Phong_reflection_model.

[12] Wikipedia. *Rendering Equation*. 2024. URL: https://en.wikipedia.org/wiki/Rendering_equation.

Acknowledgments

Questa sezione la scriverò in italiano per tutti i miei cari che mi sono stati affianco in questo quarto di secolo. Inizio dai miei genitori, mia mamma e mio papà.

Cara mamma, ti ringrazio infinitamente per il bene che mi vuoi anche se a volte non ci capiamo, anche se a volte litighiamo sei sempre stata una persona di fiducia. Spesso siamo di opinioni diverse, ai due lati estremi, e questo ci sta. Sappiamo però che alla fine della giornata, quando andiamo a dormire, ognuno può contare sull'altro. Nonostante i nostri spigoli sappiamo anche posizionarci nel modo giusto per abbracciarci senza farci del male ma stando sereni ed in pace.

Caro papà, ti ringrazio infinitamente perchè hai continuato a combattere anche contro ogni probabilità ed alla fine ci siamo rincontrati. Quando penso a quando mi sono trovato quel libro come regalo mi sembra ieri ed invece sono passati anni. Il rapporto è saltato da genitore-figlio ad amico-amico e per me non ci poteva essere cosa migliore. Ogni volta che mi vedi, anche se sono passate settimane o mesi, mi sento sempre accolto calorosamente e mi fa immenso piacere sapere che ho un'altra parte di famiglia che mi attende e mi accoglie con piacere.

Cari Matilde e Daniele, da quando siete andati ad abitare insieme il rapporto è evoluto ma ogni sabato se riesco ci tengo ad essere a casa per mangiare la pizza con voi. Vi auguro il meglio, che abbiate delle carriere di successo e delle vite felici.

Cari Carlo, Samantha e Brunalba, quando arrivo a Peschiera non mancate mai di farmi sentire sereno ed accolto. Mi sento ascoltato quando arrivo lì e spero di far percepire lo stesso anche io. Vi auguro anche a voi una vita serena e piena di soddisfazioni

Cara Irene, non saprei nemmeno da dove cominciare. Potrei cominciare da quando ti ho vista la prima volta in aula Ae il mio secondo giorno di università con quegli occhiali che ti facevano gli occhi grandissimi, occhi di cui mi sono innamorato dal primo secondo. Potrei iniziare da quando mi sono seduto dietro di te. Non conoscevo nessuno e volevo fare amicizia e te, leggendomi nel pensiero, ti sei girata a salutarmi col tuo sorriso che non neghi a nessuno. Potrei iniziare da quando abbiamo iniziato a scherzare come se ci conoscessimo da una vita o da quando Ema ci ha chiesto "da quanto è che vi conoscete? sembra che vi conosciate da anni". Potrei inizare da tutte queste cose qua ma preferisco ringraziarti per tutto quello che siamo diventati. Una spalla l'uno per l'altra e viceversa. Hai colmato un vuoto che sentivo da tempo e con te ho capito veramente cosa significa amare ma soprattutto essere amati. Spero di essere in grado di farti sentire come tu fai sentire me. Ti ringrazio per essermi vicino ogni volta che le cose si fanno difficili. Ti voglio dire che la soluzione ad un problema la troveremo sempre se stiamo insieme.

Caro Stassi, ti ringrazio per essermi stato vicino tutti questi anni. Siamo sempre stati come il giorno e la notte, ovviamente te la notte con le uscite in Veronetta e le discoteche, io il giorno con le passeggiate in montagna e le gite fuori porta in generale. Condividiamo pochi interessi in comune ma condividiamo un affetto reciproco creato nel tempo che ci fa restare amici a distanza di tempo e di spazio. Ti auguro di raggiungere i traguardi che ti poni e di vivere una vità in serenità.

Caro Samuele, non mi sarei mai aspettato di legare così tanto in così poco tempo con una persona eppure sei riuscito a farmi ricredere. Ti reputo una delle persone più buone che conosca ed un amico fidato. So sempre che se ho bisogno di parlare tu ci sei con i tuoi consigli dettati molto dalla tua empatia che ammiro. Ti auguro di diventare un'ottimo infermiere ma so già che sarà così.

Caro Gianni, sei l'amico con cui ho il legame mentale più forte. So che ogni volta che ci vediamo nasceranno dei pensieri profondi a prescindere dal tema, brutto o bello che sia. Mi hai fatto scoprire molto fuori dalla mia comfort zone e di questo te ne sono grato. Ti auguro di riuscire a vincere contro i tuoi demoni perchè ti meriti una vita serena. Sappi che ci sono ogni qualvolta che ne avrai

bisogno.

Cara Anastasia, nonostante tutte le difficoltà sei ancora a combattere. Sappi che ci sono ogni volta che avrai bisogno ed in qualche modo cercherò di darti una mano. Ti augoro di trovare la serenità che cerchi e di stare bene.

Cara Valentina, siamo sempre stati le due facce della stessa medaglia quando si trattava di rapporti. C'è sempre stao il rispetto però e questo è quello che mi è piaciuto di più. Ti sei sempre interessata a me e mi ascoltavi quando parlavo. Spero di averti fatto percepire lo stesso. Ti auguro una vita felice ma so già che sarà così

Caro Guarni, sei la persona più pazza che conosca e, anche se a volte un po' spigoloso, rimani una delle persone più buone che abbia mai incontrato. Sono molto orgoglioso dei percorsi che hai intrapreso e spero che tu riesca a trovare quello che cerchi e a raggiungere i tuoi obiettivi.

Cari Giovanni, Giacomo, Valentina e Emanuele, nonostante il percorso difficile, con voi è stato bello. Mi mancheranno infinitamente le pause ed i pranzi con voi. Sebbene questa laurea sia stato complicato non avete mai mancato di farmi sorridere e ridere. Non potevo chiedere dei compagni di univerisità migliori. Vi auguro il meglio.