Olive-SmartScan AI: An Object Detection-Based Counting System for both On-Tree and Off-Tree Olives

Francesco Bruno, Gaetano Sferrazza
f.bruno10@studenti.unipi.it, g.sferrazza@studenti.unipi.it

***Abstract* — With the following Paper, we propose a solution based on a single deep learning model, tested on a dataset of images containing olives concerning the counting of olives in two different scenarios On-Trees and Off-Tree. The System integrates object detection and counting into a single deep network. The detection module is based on a YOLO architecture, while for counting, the approach is based on detecting olives and counting them within the image downstream of the detection step through a specific algorithm. The results are promising, with an average deviation between estimated and visible counts demonstrating a good effectiveness of the model that if further improved could provide greater accuracy. Moreover, additional count-derived statistics are considered to evaluate the accuracy of the model and its potential use in count-based agricultural yield estimates. For images of trees with olives, the actual count of visible olives is used to infer the total number, including that of partially occluded olives. Overall, Olive-SmartScan AI system presents a promising solution for automated counting in the agricultural domain, offering significant potential for reducing manual labor and increasing productivity.**

***Index Terms* — Deep learning, Object Counting, Agriculture, Object Detection, Precision Agriculture, Computer Vision.**

1 Introduction

Object Counting plays a crucial role in various agricultural activities, such as plant monitoring. Historically, this task was performed manually, with all the difficulties that come with it: manual counting is costly, time-consuming, labor-intensive, subjective, and prone to errors. Recent advances in computer vision and machine learning have led to the development of automatic counting algorithms, which have been integrated into numerous applications for plant management and cultivation. These include monitoring plant health by estimating the number of leaves [4], assessing yield potential by counting fruits [5] like olives—which is essential for estimating productivity and optimizing harvest—and detecting diseases and pests [6]. It is important to note that automatic counting can be implemented using traditional computer vision tools within a general feature extraction pipeline, followed by a machine learning algorithm (such as Support Vector Machines (SVM), Random Forest, Gaussian Mixture Models (GMM), etc.). However, the main limitation of these approaches is their limited ability to generalize, mainly due to the significant variations present in the agricultural context.

Another key element supporting the agricultural world and greatly enhancing the counting procedure mentioned earlier is **Object Detection**. Object detection also plays an important role in robotic tasks such as fruit picking [7] and plant spraying [8], where a detection module is required. Fine phenotyping tasks involve examining the traits and characteristics of an object to assess growth, resilience, physiological conditions, or any other observable parameter of a plant [9], also in this case object detection provides a significant contribution. Furthermore, during the past decade, deep learning has gained a predominant role in many computer vision applications and has also made a significant impact in the agricultural sector, leading to substantial progress.

2 Related Works

To provide an overview of the state of the art in the techniques used in our study, *Object Counting* and *Detection*, it is crucial to recognize the growing importance of deep learning-based computer vision in agriculture, particularly for object counting tasks. These methods can be broadly classified into three primary categories: ***direct regression***, ***detection-based counting***, and ***density estimation*** [2]. Each of these approaches offers distinct benefits depending on the agricultural scenario, whether the task involves counting fruits, leaves, or other plant components. *Direct regression* is concerned with predicting the object count directly from the image, *detection-based counting* utilizes object detection models to identify and tally individual items, while *density estimation* involves counting by predicting a density map and summing its values across the image. Focusing on detection-based counting, this method is particularly straightforward, as it entails locating objects within an image and then adding up the detected instances. In this framework, object detection is executed using axis-aligned bounding boxes that enclose each identified object. Convolutional Neural Networks (CNNs) have shown to be highly effective in this area, given their ability to detect and classify multiple instances of objects within an image. Utilizing cutting-edge object detectors allows for count estimates to be directly derived from the results. However, to effectively train an object detector, it is necessary to meticulously collect and annotate images, encasing each object instance within a bounding box. This annotation process, while more complex than merely providing a count per image, yields superior outcomes due to the enhanced supervision. For example, Hong et al. (2021) evaluated various detection-based methods to count and detect pests and insects captured in pheromone traps. This approach also facilitates the extraction of objects for other purposes, such as disease

detection or ripeness evaluation, if required. However, this method demands a greater number of annotations and higher accuracy than other techniques, which can pose a challenge, particularly in dense scenes, common in many agricultural applications, where objects frequently obscure one another. Achieving precise detection in orchard or field settings remains a significant hurdle; however, the introduction of CNNs has considerably enhanced detection performance, addressing various challenges to some extent. CNN-based detectors are generally divided into *two* categories: ***single-stage*** and ***two-stage detectors***. *Single-stage* detectors, such as **YOLO** [11], **RetinaNet** [10], and **EfficientDet** [12], assess hundreds of thousands of potential object locations across an image, classifying them within a unified network. These models are especially well-suited for real-time applications due to their speed and efficiency. Conversely, *two-stage* detectors, like **Faster R-CNN** and **Mask R-CNN**, begin by filtering a smaller set of object candidates from the image, before classifying and refining them in a second network. Although typically slower, two-stage detectors often achieve higher accuracy and are better suited for intricate detection tasks where precision is paramount.

Thus, it is therefore essential to consider the unique challenges presented by detection tasks in agricultural environments, which often exceed those encountered in more conventional object detection scenarios. For example, field images may contain numerous objects with significant scale variations due to different distances between the camera and the objects in a row-based planting system; this situation requires detecting objects at various scales within a single image. Furthermore, target objects, such as olives or other fruits, often have simple shapes with few distinguishing features, making them difficult to differentiate from the surrounding environment.

.3 System Architecture

Considering the various approaches previously analyzed, our architecture defines a system that leverages ***object detection-based counting***. In the first phase, the object of interest is detected using a CNN model, followed by counting its occurrences through an algorithm that discriminates based on the scenarios considered in our study, more precisely ***On-Tree*** and ***Off-Tree*** olives.

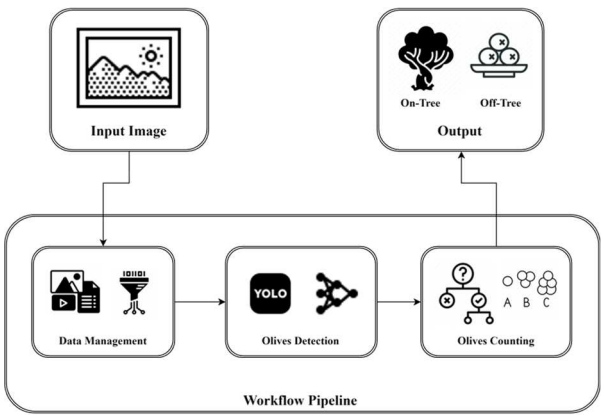

Figure 1: Functional Architecture of the System has been reported.

As can be seen from the image depicting the system architecture, the input to the system is an image containing olives. The system then processes the image and provides the total number of olives present by leveraging a main workflow characterized by three main components: the **Data Management** phase, the **Olives Detection** phase and the final **Olives Counting** phase, which will be explained in more detail in the following chapters.

3.1 Data Management

The **Data Management** module is a crucial component of the system's architecture, serving as the foundation for the subsequent phases of the olive detection and counting process. Receiving images containing olives as input, this module manages and prepares the data through a series of specially developed functions that enable efficient data loading via the ***Data Acquisition*** phase and their manipulation through ***Data Pre-Processing*** operations.

The ***Data Acquisition*** phase is responsible for loading and organizing the images within the system. This allows for the management of data that can vary significantly in characteristics, ensuring that all images are correctly acquired and ready for the following phases. The flexibility and efficiency of this process are essential to ensure that the system can scale and adapt to different datasets and operational contexts, regardless of the size or format of the image files. Once acquired, the data goes through the ***Data Pre-Processing*** phase, a fundamental process to ensure that the images are optimized for analysis by the olive detection module. During this phase, the images may undergo a series of operations, including resizing, normalization, and various other activities aimed at improving the visual quality of the images and reducing the computational complexity of the subsequent steps. Additionally, Data Pre-Processing may include data augmentation techniques, such as rotations, reflections, or brightness variations, to increase the dataset's diversity and make the detection model more robust and generalizable. These steps are crucial for adapting the images to the specific needs of the detection model, ensuring that the relevant features of the olives are well-highlighted, and that the Olives Detection module can operate optimally.

The result of the Data Acquisition and Data Pre-Processing phases is a carefully prepared image dataset, which will then be analyzed by the olive detection module. The quality and accuracy of Data Management are therefore crucial to the overall effectiveness of the system, as well-executed pre-processing allows the detection module to operate with greater precision and reliability.

3.2 Olives Detection

The **Olives Detection** module is a central component of the system’s pipeline, specializing in recognizing and localizing olives within the provided images. This module is based on the implementation of an object detection model of the CNN (Convolutional Neural Network) type, known as YOLO (You Only Look Once). Specifically, the latest version, ***YOLOv8***, has been utilized, which has been further refined and trained

specifically for olive recognition. YOLO was the first *One-Stage-Detector* algorithm in the Deep Learning era. Its uniqueness lies in the fact that it analyzes an entire image in a single step (You Only Look Once) through a convolutional neural network (CNN), rather than dividing it into parts or regions. This approach allows YOLO to be extremely fast and efficient, making it ideal for real-time applications such as video surveillance, autonomous driving, and many other situations where quick and accurate object detection is crucial. Moreover YOLO, unlike other approaches that split the detection process into multiple stages (such as region of interest detection and separate classification), unifies the entire process into a single neural network, making it extremely efficient and fast.

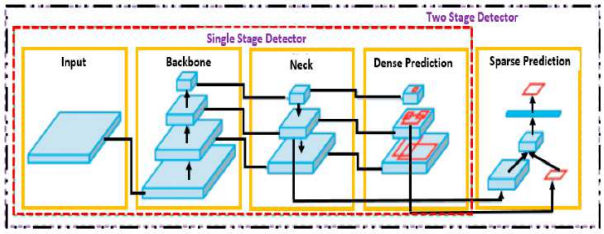

Figure 2: Object detector anatomy
(Source: https://www.mdpi.com/2075-1702/11/7/677)

In the context of this system, ***YOLOv8*** [3,13,14] was chosen due to its significant improvements over previous versions (such as YOLOv5 and YOLOv7) for several reasons:

Architectural Improvements*:* YOLOv8 introduces a more optimized network architecture that increases learning capacity and reduces inference time, making it particularly suitable for applications where a balance between speed and accuracy is required.

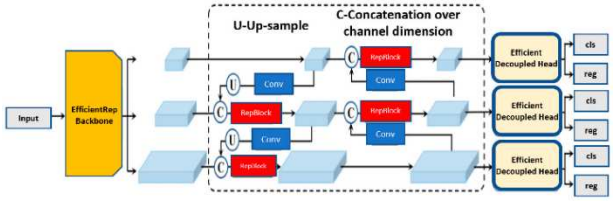

Figure 3: YOLO-v8 model base architecture
(Source: https://www.mdpi.com/2075-1702/11/7/677)

Increased Precision*:* YOLOv8 offers higher precision in detecting small objects like olives, thanks to the use of advanced feature processing techniques and multi-scale resolution, allowing for more accurate capture of minute details.

Flexibility and Adaptability*:* The new version is highly flexible and can be easily adapted to specific scenarios through a fine-tuning process. In our case, the pre-trained version of YOLOv8 was further trained on a customized dataset of images containing olives, ensuring that the model was optimized for detection in agronomic contexts.

It is worth noting that is designed to detect objects at multiple scales simultaneously. In addition, YOLOv8's ability to detect objects comes from the use of anchor boxes. These anchor boxes act as predefined templates for various objects, aiding the algorithm in making informed predictions about object locations and sizes within specific grid cells. By leveraging anchor boxes, YOLOv8 enhances its accuracy and precision, this means it can identify both large and small objects within the same image, making it ideal for our purpose of detecting the tree and the olives for the ***On-Tree*** and ***Off-Tree*** olives detection functionalities. Using YOLOv8 as the foundation for the ***Olives Detection*** module proves to be particularly effective for our needs, thanks to its ability to handle detection with high precision and speed, even in complex scenes where olives may be partially occluded or where there is significant variation in size and lighting.

The final output of this module is a set of predictions that will then be used by the subsequent *Olives Counting* module to count the olives, distinguishing between those on trees and those that have fallen. The synergy between the accurate detection provided by YOLOv8 and the subsequent counting phase allows for a highly efficient and precise system, suitable for various agronomic applications.

3.3 Olives Counting

The **Olives Counting** module represents the final and important phase of the system's pipeline, dedicated to the precise counting of olives present in an image. This module relies on predictions generated by the detection model, implementing a *custom algorithm* that distinguishes between two main scenarios: ***olives on the tree and olives not on the tree***. The module's ability to differentiate between these two scenarios is essential for providing detailed and specific information, which can be used in various agronomic applications, such as yield estimation and crop monitoring. Once the detection model has identified and classified the various entities in the image (e.g., trees, crowns, and olives), the *Olives Counting* module uses these predictions to perform the actual counting of the olives. The ***algorithm*** follows a detailed series of steps:

1. **Model Loading and Inference:** The image is processed by the detection model, which produces a set of predictions in the form of bounding boxes (representing the spatial coordinates of the detected entities) and their respective classes.
2. **Normalization of Bounding Boxes:** The bounding box coordinates are normalized relative to the image size, allowing for more efficient data handling and ensuring that subsequent operations are independent of the image resolution.
3. **Entity Classification:** The bounding boxes are then divided into three main categories: ***trees***, ***crowns***, and ***olives***, based on the classes assigned by the detection model.
4. **Counting Olives on the Tree:** The algorithm checks whether the detected olives are within the crown of a tree. If a crown is associated with a tree, the algorithm increases the count of olives on the tree.
5. **Counting Olives not on the Tree:** The number of olives not associated with any tree is calculated by subtracting the number of olives on the tree from the total number of detected olives.

The final result consists of two distinct values: ***Olive On-Tree***, which represents the number of olives on the trees, and Olive ***Off-Tree***, which represents the number of olives not on the trees. These results provide a detailed view of the olive distribution, essential for in-depth agronomic analysis.

The effectiveness of the **Olives Counting** module, of course, heavily depends on the quality of the *predictions* made by the detection model and the precision of the pre-processing operations performed in the *Data Management* module. For this reason, the obtained results were subjected to a more in-depth accuracy analysis by calculating the counting error, which will be discussed in more detail in the "*Experimental Results"* section of this paper. However, its ability to discriminate between olives on and off the trees represents a significant advancement in the field of computer vision applied to agriculture, enabling a more granular and accurate data collection.

4 Experimental Setup & Results

In the following chapter we provide a detailed description of the entire experimental process conducted to evaluate the effectiveness of the olive detection and counting system. This chapter is divided into **two main sections**: the first, dedicated to the **experimental setup**, outlines the key steps followed to configure the experiments, while the second section reports and analyzes the **results obtained**, using specific metrics to assess the system's quality.

4.1 Setup

In this section, we will explain in detail how we designed and built the customized ***datasets*** necessary to train the olive detection model, next we will describe the training process of the detection model, highlighting the importance of ***fine-tuning*** the pre-trained YOLOv8 version to adapt it to our specific agronomic context, and furthermore we will provide a detailed overview of the development of the ***counting algorithm***, which represents the final step of the process.

4.1.1 *Datasets*

We will now provide a more detailed description of the process of selecting and preparing the datasets used for training and evaluating our olive detection and counting model. To achieve our system's objectives, we started by selecting the most suitable datasets from those available on the https://universe.roboflow.com/ platform. These datasets included over 14,000 images with a resolution of 640x640, containing individual olives, olive tree in crown, olive trees, and trees in general. In addition to the images, the datasets also included *.txt* files containing labels associated with each image, a crucial aspect for training the YOLOv8 model we chose to use.

A crucial part of our work was improving and adapting these datasets to the specific needs of our project. In particular, we focused our efforts on adding additional annotations for the "***Tree***" and "***Crown***" classes, in addition to the existing annotations for the "***Olive***" class. This step was essential to enable our model to differentiate between various scenarios in which olives might appear, with particular attention to detecting olives on trees. The main goal was to minimize potential ambiguities during the detection phase, ensuring that the model could correctly distinguish between the three main classes. The process of annotating and improving the dataset required considerable effort, but it provided us with a significant advantage: the ability to perform a single training session on one well-structured model and dataset. We organized the dataset into specific folders for the train-set, validation-set, and test-set, optimizing the training process and ensuring that the model could recognize all three classes within a given image. This choice also helped reduce the overall training time, which could have increased significantly if we had opted to use multiple neural networks for different purposes.

Additionally, by leveraging a pre-trained YOLO model, we were able to significantly reduce the number of images needed for model training. We selected ***1,635*** images, of which ***1,000*** represented trees and ***635*** represented olives (*472* olives alone and *163* olives with crown). This combination allowed us to conduct targeted experiments with a sufficiently diverse and representative dataset, minimizing the risk of overfitting and improving the model's ability to generalize to new data. At the end of the process, we unified all the images and their associated label files into a single dataset folder called "**full_dataset**." This dataset was then reorganized into specific folders, ready to be used in the training phase cross-validation based.

To create the **cross-validation** folder containing the rounds on which this technique is based, two main tasks were performed considering that we would be dealing with a K-fold Cross-Validation with K=5:

- Splitting the "*full_dataset*," which was pre-shuffled, into five subfolders named ***fold_0***, ***fold_1***, ***fold_2***, ***fold_3*** and ***fold_4***
- Creating five folders named ***round_0***, ***round_1***, ***round_2***, ***round_3*** and ***round_4***, where the images from the subfolders in point 1 were properly placed.

The criteria for selecting the images to reorganize and use the dataset for the previously explained activities are as follows:

1) Criteria for creating ***fold_i***, with 327 images per folder:
    1. Shuffle the dataset from which you want to extract the images to be inserted into each ***fold_i***, in our case "*full_dataset*".
    2. Select the number of images of interest, in our case 327.
    3. Insert the images and their corresponding ***labels.txt*** files into the ***fold_i*** folder.

Note: This process should be repeated for each *dataset_i* and *fold_i* if you have separate datasets of different classes.

At the end, in our case, you will have five ***fold_i*** folders containing 327 images with their respective .txt labels files, so the total for each fold is 654 elements.

2) The five ***ROUND_i*** folders will each contain three subfolders: ***test***, ***train*** and ***val***, where the images from the various ***fold_i*** folders created earlier will need to be appropriately placed. To do this, the following selection criteria should be followed:

*For each round, the test folder will contain the complete dataset of one **fold_i** (327) that changes round by round, while **train** and **val** will respectively distribute **80%** (830) and **20%** (207) of the images present in all **fold_i** (round by round as well) excluding the current round's **fold_i,** which is in the **test** folder.*

Note: Of course, you must not forget to add the .txt files of the labels as well.

To better clarify the structuring of folders, below we will provide you a detailed view of the distribution of data in the various folders *ROUND_0*, *ROUND_1*, *ROUND_2*, *ROUND_3*, *ROUND_4*.

Round	Test Set	Train Set	Validation Set
ROUND_0	fold_0 (100%)	fold_1, fold_2, fold_3, fold_4 (80%)	fold_1, fold_2, fold_3, fold_4 (20%)
ROUND_1	fold_1 (100%)	fold_0, fold_2, fold_3, fold_4 (80%)	fold_0, fold_2, fold_3, fold_4 (20%)
ROUND_2	fold_2 (100%)	fold_1, fold_0, fold_3, fold_4 (80%)	fold_1, fold_0, fold_3, fold_4 (20%)
ROUND_3	fold_3 (100%)	fold_1, fold_2, fold_0, fold_4 (80%)	fold_1, fold_2, fold_0, fold_4 (20%)
ROUND_4	fold_4 (100%)	fold_1, fold_2, fold_3, fold_0 (80%)	fold_1, fold_2, fold_3, fold_0 (20%)

Figure 4: Summary Table for folder structuring for cross-validation technique

- **Test Set** is the subset used for testing in each round.
- **Training Set** consists of 80% of the images from the remaining folds.
- **Validation Set** consists of 20% of the images from the remaining folds.

Just to recap each round uses a different fold as the test set and combines the remaining folds for training and validation, ensuring that each fold gets used in every part of the cross-validation process. This last structure configuration of the dataset obtained, as mentioned earlier, is crucial for the training phase that will be discussed in detail in the next section.

The final phase of dataset construction in our study involved the creation of datasets specifically designed to conduct the system's final evaluations for both detection and counting. In this crucial stage, we generated new datasets composed exclusively of data that the model had never seen during the training and validation process. This approach was essential for testing the system's effectiveness and evaluating its performance in realistic scenarios, ensuring an unbiased assessment of the model's capabilities. We created a dedicated folder called "**evaluation_datasets**" where the data for the two main scenarios of interest in our study were organized:

- **Olive On-Tree**: This dataset contains images representing the first scenario of interest, namely olives on trees. The images in this dataset were selected to represent a variety of environmental conditions and perspectives, allowing us to test the model's robustness in detecting and counting olives on trees.
- **Olive Off-Tree**: The second dataset contains images related to the scenario of olives in contexts other than on trees. In this case as well, the images were carefully selected to cover different situations the model might encounter in a real-world application.

These final datasets were used to conduct the system's concluding tests, providing a detailed evaluation of its performance in the two scenarios of interest. The creation of distinct datasets for these two conditions allowed for a precise analysis of how the model behaves in specific situations, highlighting its generalization abilities and its effectiveness in solving the problem of detecting olives On-Tree and Off-Tree.

4.1.2 *Training*

Below is a detailed description of the training process for the olive detection and counting model, based on the YOLOv8 framework. The chosen model is a ***pre-trained*** version of ***YOLOv8***, which allowed us to leverage the extensive prior knowledge from millions of previously processed images. This made the network highly efficient in recognizing low-level features in the initial layers, while focusing the training on the last layers of the network, specialized in olive recognition.

This fine-tuning approach significantly reduced training time compared to a model trained from scratch, while also improving the model's ability to generalize even with a relatively small amount of data.

Since YOLOv8 offers different versions, we decided to test them all and evaluate which one yielded the best results.

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

Figure 5: Size, Speed, and Accuracy comparison of the five models of Yolov8. (Source: Ultralytics)

The training and validation method we chose to employ is **Cross-Validation**. This technique splits the dataset into ***k subsets*** (or *folds*), using k-1 of them for training and one for validation in each training round, "***crossing***" the various parts of the dataset. In our case, we opted for a ***k-fold*** Cross-Validation with k=5. This means that for each version of YOLOv8, five separate training sessions were conducted, each on a different combination of ***train***, ***test***, and ***validation*** set portions contained in the respective ***round folders***. In total, having tested five different versions of YOLOv8, 25 distinct

training sessions were performed in *Cross-Validation*. For a more detailed analysis of how the dataset was organized to carry out the process described so far, we refer you to the previous section " *4.1.1 Datasets*." This approach was chosen because it is particularly advantageous compared to simply splitting the dataset into *training* and *test* sets, as it provides a more ***robust*** estimate of the model's performance, reduces the risk of ***overfitting***, and improves the model's ability to ***generalize*** to new data, offering a more comprehensive view of its capabilities.

Thanks to collaboration with the ***University of Pisa***, we had access to high-performance computational resources, including servers with large memory capacity and computing power. These resources allowed us to adopt a dynamic and flexible approach to managing the fine-tuning of training parameters, maximizing the effectiveness of the process and various experiments. The chosen training parameters include 100 ***epochs*** and a ***batch-size*** of 64. The choice of a relatively large batch size helped improve the model's ability to generalize, while the optimizer used was "***Adam***," the default in YOLOv8. Additionally, we took advantage of YOLOv8's ***built-in*** automatic best epoch selection mechanism during training, which continuously monitors overfitting signals and saves the best weights for the optimal epoch. The validation metrics calculated on these weights provided a solid foundation for evaluating the model's performance and proceeding with further development phases.

At the end of these experiments, metrics were calculated from the data obtained in various training sessions, which were useful for correctly selecting the best model among the five, which will be detailed in the next section "*Results*."

4.1.3 *Counting Method*

During the development phase of the algorithm for counting olives based on the predictions generated by the detection model, our primary goal was to accurately distinguish between olives on trees and those not on trees. Starting from the fact that ***YOLO*** detects the classes ***Olive***, ***Tree***, and ***Crown*** by highlighting each with a ***bounding box***, we designed an ***algorithm*** that leverages this information for targeted olive counting. Specifically, the algorithm we developed is based on a systematic comparison of the bounding boxes generated by the model. The key principle of our approach is the identification of bounding boxes belonging to the ***Olive*** class, followed by evaluating whether they are ***contained*** within the bounding boxes of the ***Crown*** or ***Tree*** classes. This process relies on a property we defined as "***isContained***," which is essential for determining whether a detected olive is on a tree or not. To implement this functionality, we developed a custom function that checks if one bounding box is completely contained within another. The function, called ***is_contained***, operates by checking if the coordinates of the olive's bounding box are within the boundaries of the Crown or Tree bounding boxes. It was designed following this line of reasoning:

Let $Box1 = (x1_1, y1_1, x2_1, y2_1)$ and $Box2\ (x1_2, y1_2, x2_2, y2_2)$. The condition for which ***Box1*** is contained within ***Box2*** can be expressed by the following formula:

(1)

$$\boldsymbol{isContained}(Box1, Box2) = (x1_1 \geq x1_2) \wedge (y1_1 \geq y1_2) \wedge$$

$$\wedge (x2_1 \leq x2_2) \wedge (y2_1 \leq y2_2)$$

Where:

- $\boldsymbol{x1_1 \geq x1_2}$**:** The x-coordinate of the top-left corner of Box1 is greater than or equal to the x-coordinate of the top-left corner of Box2.
- $\boldsymbol{y1_1 \geq y1_2}$**:** The y-coordinate of the top-left corner of Box1 is greater than or equal to the y-coordinate of the top-left corner of Box2.
- $\boldsymbol{x2_1 \leq x2_2}$: The x-coordinate of the bottom-right corner of Box1 is less than or equal to the x-coordinate of the bottom-right corner of Box2.
- $\boldsymbol{y2_1 \leq y2_2}$: The y-coordinate of the bottom-right corner of Box1 is less than or equal to the y-coordinate of the bottom-right corner of Box2.

If all four of these conditions are true, then ***Box1*** is ***completely*** contained within ***Box2***. The algorithm then proceeds to evaluate, for each olive, the instances associated with the *Olive* class by checking whether the corresponding *bounding box* is *contained* within the *Crown* or *Tree* bounding box. If the Olive's bounding box is contained within the Crown's bounding box, it is classified as "***olive on tree***." Similarly, if the latter it is contained within the Tree's bounding box, it is also classified as "***olive on tree***." However, if only the Crown's bounding box is present and not the Tree's, the olive is still considered "***on tree***." Conversely, if the Crown's bounding box is missing but the Tree's is present, the olive is ***not*** classified as "*on tree*" because it's not possible to confirm whether the olive is truly on the tree or on the ground. If both Crown and Tree's bounding boxes are absent, the olive is classified as "***off-tree***".

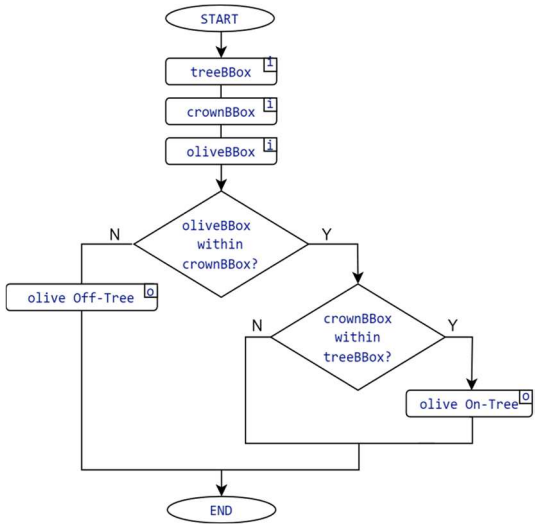

Figure 6: Functional Flowchart of the algorithm that counts olives and discriminates between On-Tree and Off-Tree olives

This mechanism of comparing bounding boxes, and particularly the "***isContained***" property, is crucial for achieving our goal of accurately counting olives on trees.

We applied these counting operations to all the versions of the YOLOv8 model that we trained, with the aim of identifying which model was the most accurate in counting olives in the context that interests us the most, specifically "***olives on tree***."

The results are presented in the "*4.2 Results*" section, which is discussed in detail in the following chapter.

4.2 Results

We will present the metrics we identified and used to evaluate the system's performance. These metrics were chosen to offer a comprehensive view of the system's effectiveness in both the detection, through ***Confusion Matrix***, ***Precision***, ***Recall***, ***F1-Score*** and the final ***mAP@50*** *(mean Average Precision*) accuracy and counting phases through a metric specially selected during the experimentation called ***MSE*** that stands for "*Mean Squared Error*". The results obtained will be analyzed and compared to demonstrate the robustness and reliability of our approach, providing a solid foundation for future applications and developments in the field of computer vision applied to agriculture.

4.2.1 *Detection-Model Performance Evaluation Metrics*

The metrics used for evaluating the model's performance are *Confusion Matrix*, *Precision*, *Recall*, *F1-Score*, and *mean Average Precision* (*mAP@50*). These metrics were calculated using the internal mechanisms provided by the YOLOv8 framework, which, at the end of the training phase, allows the model to be put into validation mode to obtain the necessary results. Among all the metrics considered, we focused on ***mAP@50*** to select the best model, as it is a well-established indicator of the model's ability to accurately detect the objects of interest, in our case, the olives.

The ***Confusion Matrix*** is particularly useful because it allows for the calculation of other evaluation metrics, such as precision, recall, accuracy, and F1-Score previously mentioned, providing a detailed view of the model's performance on each class. It helps easily identify where the model makes errors, whether these are more frequent in *false positives* or *false negatives* and offers an overall picture of the model's predictive abilities. It is represented by a table where the ***rows*** correspond to the *actual* classes (real observations) and the ***columns*** represent the classes *predicted* by the model. This setup allows for a comparison between the model's predictions and the actual values, enabling an analysis of how the model performs on each class present in the dataset. The metrics ***Precision***, that measures the percentage of positive predictions that are accurate, ***Recall***, that measure of how well a classifier identifies all positive samples and ***F1-Score***, that balances precision and recall for the positive class, for all classes are based on ***TP*** (True Positive), ***TN*** (True Negative), ***FP*** (False Positive), and ***FN*** (False Negative), as defined in Eq\s. 2 to 4. A good F1 score indicates that the model has low false positives and low false negatives.

(2)

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

(3)

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

(4)

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

These metrics then allow for the construction of a ***precision-recall*** curve, from which the ***Average Precision*** for each class can be obtained, associated with a specific ***IoU*** (*Intersection over Union*) threshold, specifically the notion of ***AP@50*** when considering the threshold equal to 0.5.

In order to provide a global accuracy metric, we will resort to the ***mean Average Precision*** (*mAP*), as earlier said, simply calculated as the mean value of AP among all classes as defined in Eq. 5.

(5)

$$mAP = \frac{1}{N}\sum_{h=1}^{N} AP^{h}$$

Where N is the number of classes to be detected, also, in this case, we will use the notion of ***mAP@50*** considering an IoU threshold equal to 0.5. The letter ***h*** present above the ***AP*** within the summation is not a power but only indicates how that AP relates to the ***h-th*** class.

As mentioned earlier, the experimentation was based on the ***Cross-Validation*** technique. The evaluation process for each model involved performing cross-validation over 5 rounds. After the training of each model in every round a list of metrics was generated for the best model among all epochs and with the model set to validation mode we extracted the ***mAP@50*** for each class: ***Olive***, ***Crown***, and ***Tree***.

To ensure that our results were statistically significant, we calculated the ***average*** *mAP@50* across all 5 rounds for each model and computed the corresponding ***standard deviation***. These results were then compiled into a comprehensive table. This process was repeated for all version of the models tested, allowing us to identify and select the model with the best combination of *mAP@50* and *standard deviation* for our project. This rigorous approach ensured that our selection was based on a robust and reliable performance metric.

Below is the final table containing mAP@50 and standard deviation for each model of the YOLO v8 framework.

MODEL		CLASSES		
		Tree	*Crown*	*Olive*
YOLOv8	*Nano*	85.86 ± 2.17	99.36 ± 0.23	71.54 ± 4.05
YOLOv8	*Small*	86.04 ± 2.64	99.4 ± 0.20	71.82 ± 4.54
YOLOv8	*Medium*	86.26 ± 2.14	98.9 ± 0.67	71.04 ± 4.45
YOLOv8	*Large*	85.70 ± 2.02	99.3 ± 0.13	71.22 ± 3.88
YOLOv8	*XLarge*	86.54 ± 2.06	98.82 ± 0.49	71.6 ± 4.61

Figure 7: Summary Table of the results obtained by evaluating the detection performance of the models tested for Olive, Crown, and Tree classes with relative *mAP@50 ± standard deviation* in Cross-Validation

As can be seen from the table above, all models achieved good results in terms of precision for the ***Tree*** and ***Crown*** classes. For this reason, in selecting the best model for our purposes, we decided to focus more on the model that reported the highest value for the ***Olive*** class. Consequently, the most overall performant model is ***YOLO v8 Small***, with an accuracy of *86.04 ± 2.64* for the ***Tree*** class, *99.4 ± 0.20* for the ***Crown*** class, and *71.82 ± 4.54* for the ***Olive*** class.

For completeness, we also present the values of all the metrics calculated for the chosen model below.

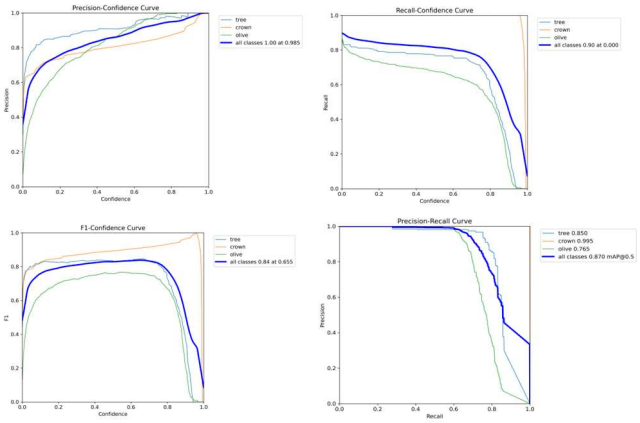

Figure 8: Summary Tables for *Precision*, *Recall* and *F1-Score* Classification Metrics and *Precision-Recall* Curve for YOLO v8 Small model for Olives detection in Cross-Validation

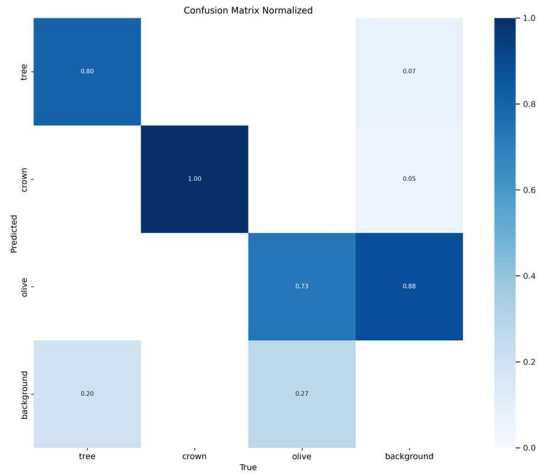

Figure 9: Confusion Matrix for *YOLO v8 Small* model for Olives detection in Cross-Validation

4.2.1 *Counting Performance Evaluation Metrics*

As previously mentioned, the accuracy of the counting process is heavily influenced by the detection phase performed by the model earlier in the pipeline. However, we deemed it essential to carefully and statistically evaluate the system's counting abilities by considering any potential errors that may arise during the counting phase following the model's predictions, utilizing a dedicated algorithm.

To assess the effectiveness of the olive counting performed by our system, we required a statistical metric that would allow us to quantify the error by comparing the ***predicted*** values with the ***actual*** number of olives present in the images. We chose to rely on the ***Mean Squared Error*** (***MSE***) metric, which is widely used in the field of machine learning to measure the average squared errors by calculating the quadratic mean difference between the estimated values and the actual values, according to the following formula:

(6)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y'_i)^2$$

Where Y_i is the ***actual true*** value of the olives in the image and Y'_i is the ***predicted*** value.

We calculated the ***MSE*** during the counting phase for each individual image in the dataset created for testing purposes. To conduct this evaluation, we constructed a specialized dataset by selecting 130 images, a mix of olives on the tree and olives not on the tree; for more details on the structure of this dataset you can refer to the relevant section of this paper "*4.1.1 Datasets*". These were then divided into two separate datasets—one for olives ***On-Tree*** and another for olives ***Off-Tree***—allowing us to conduct distinct analyses and evaluation for the two scenarios considered in our project.

Upon completing the various tests, we documented the results in a dedicated table, which enabled us to observe which model performed better in the counting phase. It is important to note that while ***MSE*** provides a global measure of error, it does not reveal the ***distribution of errors***. Examining individual image errors can be useful to identify outliers that may be inflating the MSE. Additionally, it is crucial to consider the MSE value in relation to the "***Scale of Data***." For instance, a large MSE value might initially suggest a high mean squared error, but to evaluate it properly, one must consider that if the number of olives per image typically ranges in the hundreds or thousands, an MSE in the hundreds may not be excessively high. However, if the number of olives is usually much lower, say in the tens, this MSE value would suggest that the model is making significant errors. For further clarity in data interpretation, we decided to generate histogram charts to better understand the results and ***error distribution***. Alongside MSE, we also calculated the corresponding ***Root Mean Square Deviation*** (***RMSD***), which represents the standard deviation.

Below are reported the results obtained for the two different evaluations done per single scenario

MODEL		SCENARIOS	
		Olives On-Tree	***Olives Off-Tree***
YOLOv8	***Nano***	1341,46 ± 36,62	72,76 ± 8,53
YOLOv8	***Small***	1060,68 ± 32,56	164,2 ± 12,81
YOLOv8	***Medium***	1089,06 ± 33	123,26 ± 11,10
YOLOv8	***Large***	961,67 ± 31,01	42,66 ± 6,53
YOLOv8	***XLarge***	1092,03 ± 33,04	108,26 ± 10,4

Figure 10: Summary Table of the results obtained by evaluating the counting performance of the system tested for olives On-Tree and Off-Tree Scenarios with relative *MSE* ± *standard deviation* (RMSD)

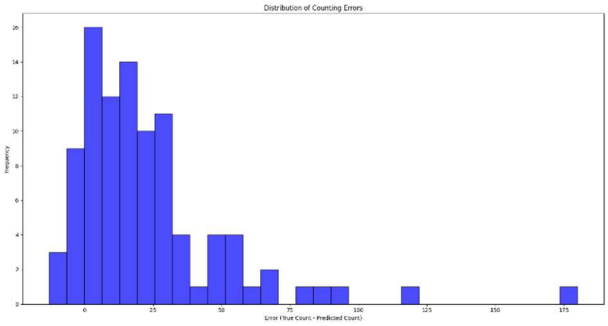

Figure 11: Histogram for YoloV8 Nano

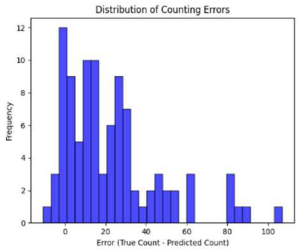

Figure 11.1: Histogram for YoloV8 Small

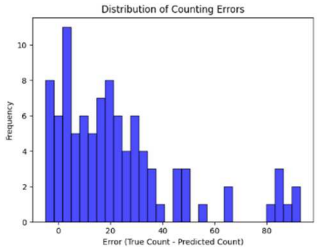

Figure 11.2: Histogram for YoloV8 Medium

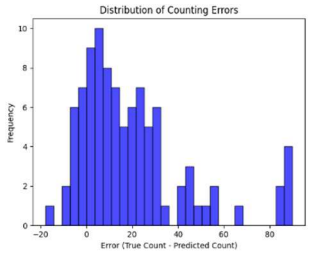

Figure 11.3: Histogram for YoloV8 Large

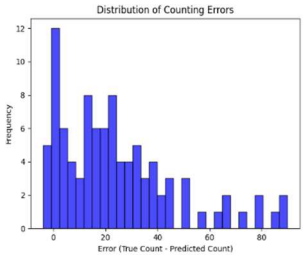

Figure 11.4: Histogram for YoloV8 XLarge

The histograms presented above, derived during the performance evaluation phase of the counting system tested for olives in both ***On-Tree*** and ***Off-Tree*** scenarios but we reported here ***on the tree ones*** because more interest for our purposes, illustrate the distribution of errors in the counting phase across all YOLO v8 model versions. As shown, the ***horizontal axis*** represents the ***error*** values, while the ***vertical axis*** indicates the ***frequency*** of these errors. As we partially expected before conducting the various analyses—considering that counting accuracy is influenced by the detection phase—***YOLO v8 Small*** emerges as the best-performing model in the counting phase as well.

This observation is particularly evident in its corresponding histogram, ***Figure 11.1: Histogram for YoloV8 Small***, where the error distribution is generally more balanced within the range of non-zero values of the errors. Specifically, the histogram shows that the model exhibits a ***lower*** *frequency of high error values per image and a* ***higher*** *frequency of errors close to zero per image*. This balanced distribution underscores YOLO v8 Small's superior ability to minimize counting errors, suggesting that it handles the complexities of both the ***detection*** and ***counting*** phases more effectively than the other models tested.

Such results confirm that ***YOLO v8 Small*** is particularly well-suited for tasks requiring both high detection accuracy and reliable object counting, olives in our case, making it the most robust option among the tested models for this specific application in precision agriculture.

5 Conclusion

In conclusion, considering the objective of this study, which was to develop a system capable of using machine learning to estimate yield by imitating the procedure followed by experts in the field—based on observing and interpreting visible olives directly on site—the results obtained are promising. They indicate that the developed system is effective in detecting and counting olives. However, there are several areas where the system's performance could be further improved.

A key area for improvement involves increasing the ***dataset*** used during the training phase. By ***expanding*** the amount and diversity of training data, the model could achieve more robust generalization capabilities, thereby enhancing accuracy and reliability in both detection and counting phases.

Furthermore, opting to use ***2-stage models*** as an alternative for the ***detection*** phase in the pipeline could represent a significant advancement. 2-stage models, which divide the detection process into two distinct phases (region of interest detection and subsequent classification), often offer greater precision than single-stage models like YOLOv8. This strategy could significantly improve the quality of predictions, particularly for the olive class, which proved to be the most challenging in our study.

Regarding the ***counting*** phase, an additional improvement could come from developing a network based on ***direct regression***, capable of directly estimating the number of objects in an image, thereby reducing the impact of errors made during the detection phase. An approach based on cropped images during the detection phase could also provide more specific and accurate inputs to the counting network. Moreover, considering the use of different metrics to evaluate errors during the counting phase could offer a deeper understanding of the system's performance. For example, alongside *Mean Squared Error* (*MSE*), adopting metrics such as ***Absolute Error*** or ***Mean Absolute Percentage Error*** (***MAPE***) could provide further insights into improving the model and better addressing systematic errors or data variability.

In summary, although the results achieved are already significant, there are several promising avenues to further enhance our system. Focusing on these aspects could lead to an even more effective and reliable model, paving the way for new applications in the field of computer vision applied to agriculture.

References

[1] Object detection in agricultural contexts: A multiple resolution benchmark and comparison to human, Omer Wosner, Guy Farjon, Aharon Bar-Hillel, Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer Sheva, Israel

[2] AgroCounters—A repository for counting objects in images in the agricultural domain by using deep-learning algorithms: Framework and evaluation, Guy Farjon, Yael Edan, Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer Sheva, Israel

[3] YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection, Muhammad Hussain, Department of Computer Science, School of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfield HD1 3DH, UK

[4] Setyawan, R.A., Basuki, A., Wey, C.Y., 2020. Machine vision-based urban farming growth monitoring system. In: 2020 10th Electrical Power, Electronics, Communications, Controls and Informatics Seminar. EECCIS, IEEE, pp. 183–187.

[5] Bargoti, S., Underwood, J., 2017. Deep fruit detection in orchards. In: 2017 IEEE International Conference on Robotics and Automation. ICRA, IEEE, pp. 3626–3633

[6] Liu, J., Wang, X., 2021. Plant diseases and pests detection based on deep learning: a review. Plant Methods 17, 1–18.

[7] Santos, T.T., de Souza, L.L., dos Santos, A.A., Avila, S., 2020. Grape detection, segmentation, and tracking using deep neural networks and three-dimensional association. Comput. Electron. Agric. 170, 105247

[8] Berenstein, R., Shahar, O.B., Shapiro, A., Edan, Y., 2010. Grape clusters and foliage detection algorithms for autonomous selective vineyard sprayer. Intel. Serv. Robot. 3, 233–243

[9] Costa, C., Schurr, U., Loreto, F., Menesatti, P., Carpentier, S., 2019. Plant phenotyping research trends, a science mapping approach. Front. Plant Sci. 9, 19

[10] Lin, T.Y., Goyal, P., Girshick, R., He, K., Doll´ar, P., 2017b. Focal loss for dense object detection. In: Proceedings of the IEEE international conference on computer vision, pp. 2980–2988.

[11] Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779–788

[12] Tan, M., Pang, R., Le, Q.V., 2019. Efficientdet: Scalable and efficient object detection. arXiv preprint arXiv:1911.09070

[13] https://medium.com/@m.tahailyas786/principles-of-yolo-v8-3c605eab9bcc

[14] https://github.com/ultralytics/ultralytics

[15] ***Olive-SmartScan_AI GitHub Repository***: https://github.com/g-sferr/Olive-SmartScan_AI/tree/main