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

**Authors: Francesco Bruno, Gaetano Sferrazza**

# OUTLINE


- INTRODUCTION
- APPLICATION CONTEXT
- STATE OF THE ART
- SYSTEM ARCHITECTURE
- EXPERIMENTAL SETUP & RESULTS
- CONCLUSION

# INTRODUCTION

## Olive-SmartScan_AI:

- **What is**: It's deep learning-based System that provides a solution to measure the yield of olive groves.
- **Objective**: Automate the olive count to help a farmer to improve its agricultural productivity and to reduce manual labor in general.
- **How it works:** It takes an olive tree picture and thanks to the trained deep learning model and to the counting algorithm, gives you the number of olive for these two Scenarios:
  - **Olives On-Tree**: Represents the case where olives present on a tree and its crown are counted in the image.
  - **Olives Off-Tree**: Represents the case where olives are counted in the image in contexts other than those strictly related to a possible tree's crown.

# APPLICATION CONTEXT

## Object Counting in Agriculture

- **Crucial Role**: Object counting is essential for tasks like plant monitoring, yield estimation, and disease detection in agriculture.
- **Challenges of Manual Counting**: Historically, manual counting has been labor-intensive, time-consuming, and prone to errors.
- **Advances in Technology**: The development of automatic counting algorithms, leveraging computer vision and machine learning, has significantly improved the efficiency and accuracy of these tasks.

## Challenges in Agricultural Object Detection

- **Complexity**: Agricultural environments pose unique challenges, such as scale variations and occlusions in field images.
- **Object Characteristics**: Target objects like fruits often have simple shapes and blend into the environment, making detection more difficult.

# STATE OF THE ART

**Deep Learning Impact**: Deep learning has revolutionized object detection and counting, offering better generalization and performance.

## Object Counting Methods:

- **Direct Regression**: Predicts object count directly from images.
- **Detection-Based Counting**: Locates and counts objects using detection models.
- **Density Estimation**: Counts objects by predicting and summing density maps.

## Object Detection (CNN-based detectors) :

- **Single-Stage**: Faster and suited for real-time applications (e.g., YOLO, EfficientDet).
- **Two-Stage**: More accurate, better for detailed tasks but slower (e.g., Faster R-CNN, Mask R-CNN)

# UNDERLYING THE SYSTEM

## Detection-Based Counting:

- **Methodology**: Involves detecting objects within images using bounding boxes, then tallying detected instances.
- **Effectiveness**: CNN-based detectors (e.g., YOLO, RetinaNet) have proven highly effective for this approach, especially in complex agricultural environments.

### Detection Component:

- **YOLO**: It was chosen for the object detection component because of its high accuracy and efficiency in identifying olives within the images.

### Counting Component:

- **Custom Algorithm**: We designed a custom algorithm specifically to count the predictions made by the detection model in with the goal of being able to discriminate between two distinct scenarios:
  1. Olive **On-Tree**
  2. Olive **Off-Tree**.

# YOLOv8 – *ARCHITECTURE OVERVIEW*

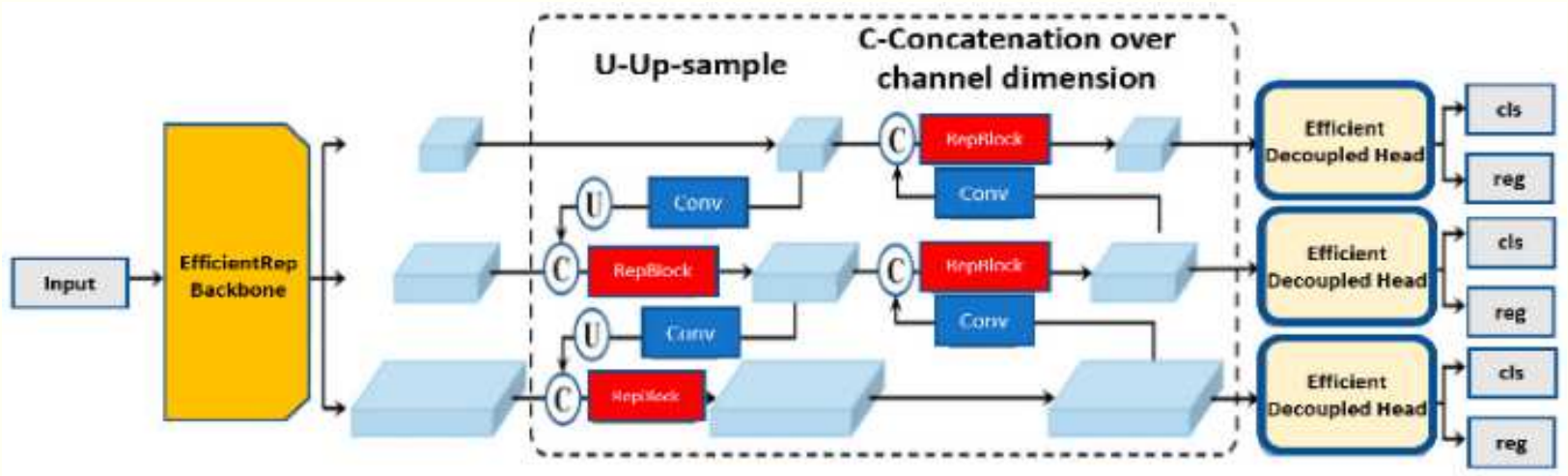


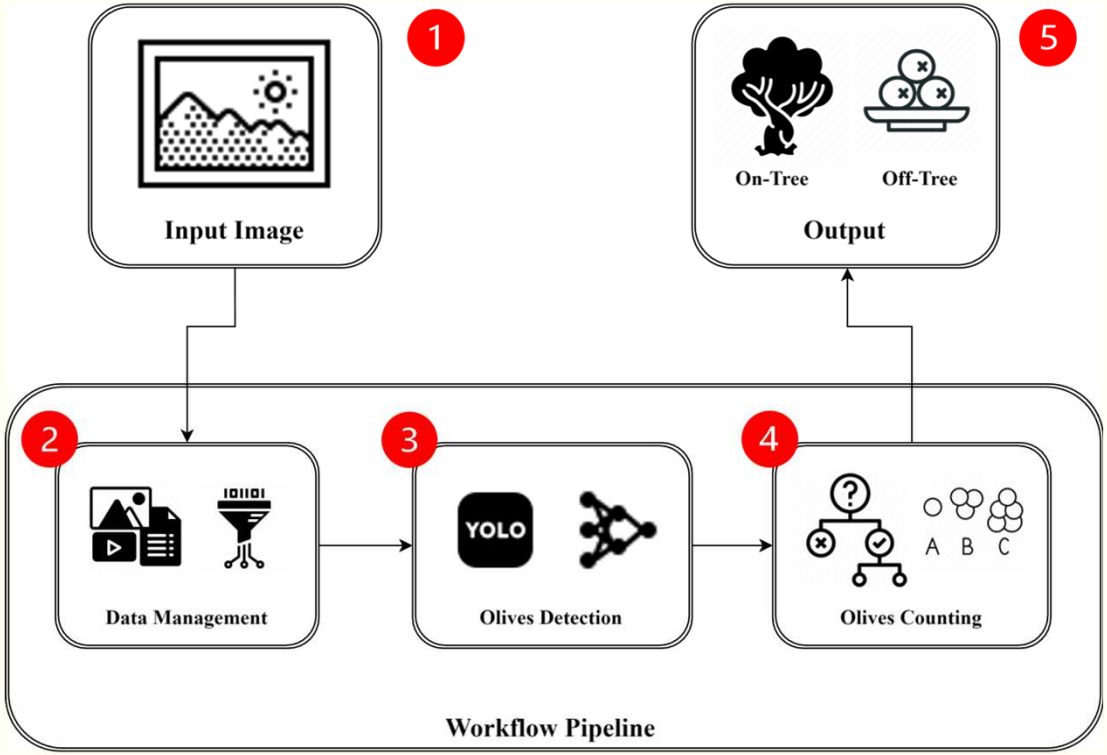

**Table**: YOLOv8 Model Base Architecture.

**Architectural Improvements**: Optimized network architecture that increases learning capacity and reduces inference time.

**Increased Precision**: Higher precision in detecting small objects like olives.

**Flexibility and Adaptability**: Easily adapted to specific scenarios through a fine-tuning process.

# SYSTEM ARCHITECTURE




1. **Input Of The System:**
   1. Images of olive trees.
2. **Data Management Phase:**
   1. Loads, adjusts and organizes the images.
   2. Creates a prepared dataset for the next phase.
3. **Olives Detection Phase:**
   1. Detects olives, crown and the tree in the provided image.
   2. Bounding box-based location.
4. **Olives Counting Phase:**
   1. Two different counting scenarios.
   2. The count is based on *isContained* property.
5. **Output Of The System:**
   1. Calculation of the total number of olives in On-Tree and Off-Tree scenarios.

# EXPERIMENTAL SETUP – *DATASETS CONSTRUCTION*

## Dataset Selection:

- **Source**: Suitable datasets from the Roboflow platform with over 14,000 images (640x640 resolution) containing individual olives, olive in crown, and general tree images.
- **Compatibility**: Each image was accompanied by a .txt file containing labels, essential for training the YOLOv8 model.

## Organized Dataset Structure:

- **Splitting**: Created specific folders for train-set, validation-set, and test-set, useful for the training and validation process.
- **Advantages**: The well-structured dataset allowed for a good training session, reducing the need for multiple neural networks and shortening training time.

## Dataset Enhancement:

- **Annotation**: Added additional annotations for "Tree" and "Crown" classes alongside existing "Olive" annotations.
- **Advantages**: This step ensured the model could distinguish between different scenarios, particularly focusing on olives on trees.

## Unified and Reorganized Dataset:

- **Reduction**: Reduced the number of images needed by selecting 1,635 images (1,000 trees, 635 olives)
- **Cross-Validation Folder**: Prepared the dataset for k-fold cross-validation with k=5.
- **Evaluation Dataset**: Prepared the dataset to conduct the system's final evaluations for both detection and counting.

# EXPERIMENTAL SETUP – *DATASETS CONSTRUCTION*

| 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%) |

**Table**: Folder structuring for cross-validation technique.

## Criterion for Cross-Validation Folders Creation:

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

# EXPERIMENTAL SETUP – *TRAINING & VALIDATION APPROACH*

## Model Selection and Fine-Tuning:

- **Model Type**: Used a **pre-trained** YOLOv8 model, leveraging extensive prior knowledge from millions of images.
- **Advantages**: Focused training on the last layers for olive recognition, reducing training time and improving generalization.

## Training Parameters:

- **Hyperparameters**: Set at 100 **epochs** with a **batch** size of 64, using the "**Adam**" optimizer
- **Built-in Mechanism** : YOLOv8's automatic best epoch selection saved optimal weights, ensuring the best model performance.

## Cross-Validation Approach:

- **Technique**: Employed **5-fold** Cross-Validation, resulting in **25 training sessions** (5 versions x 5 rounds folds).
- **Advantages**: This method provides a more robust performance estimate and reduces overfitting.

## Computational Resources:

- **High-Performance**: Utilized servers with large memory and computing power from the University of Pisa
- **Advantages**: Enabled dynamic fine-tuning of training parameters, optimizing effectiveness.

# EXPERIMENTAL SETUP – *TRAINING & VALIDATION APPROACH*

**Testing Multiple YOLOv8 Versions:** Evaluated all versions of YOLOv8 to determine the best-performing model for olive detection and counting

| 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 |

**Table**: Models of Yolov8.

**Performance Metrics:** Metrics from the training sessions guided the selection of the best YOLOv8 model, to be discussed in the "Results" section

# EXPERIMENTAL SETUP – *COUNTING METHOD*

**Main Objective:** Accurately distinguish between "**olives on trees**" and "**olives off trees**".

**Bounding-Box Comparison:**

- **Detection-Model Output**: To compare YOLO model's bounding-boxes for the **Olive**, **Tree**, and **Crown** classes, post-prediction.

**"Is Contained" Property:**

- **Custom Function**: To check if **Olive**'s bounding-box is **completely** within the bounding box of the **Tree** or **Crown** class

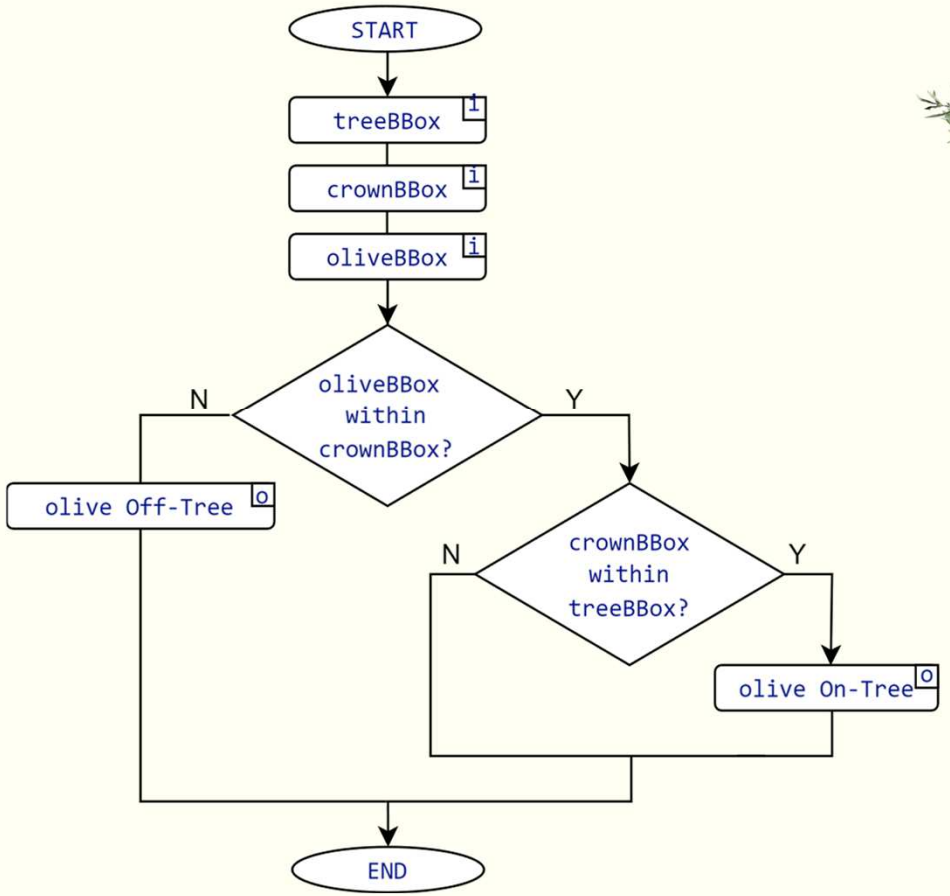
- Let $Box1 = (x1_1', y1_1', x2_1 y2_1)$ and $Box2 = (x1_2', y1_2', x2_2 y2_2)$. The condition for which **Box1** is completely contained within **Box2** can be expressed by the following formula:

$$\boldsymbol{isContained}(Box1, Box2) = (x1_1 \geq x1_2) \wedge (y1_1 \geq y1_2) \wedge (x2_1 \leq x2_2) \wedge (y2_1 \leq y2_2)$$

# EXPERIMENTAL SETUP – *COUNTING METHOD*

## Algorithm Logic (For Each Olive):

- Olive is "on tree" if its bounding box is within the Crown or Tree bounding box.
- If only the Crown is present, the olive is still "on tree".
- If only the Tree is present, the olive is not classified as "on tree".
- Olive is "off tree" if neither Crown nor Tree bounding boxes are present.



## EXPERIMENT RESULTS – *DETECTION-MODEL PERFORMANCE EVALUATION*


**Evaluation Metrics:**

- **Confusion Matrix**: Provides insights into errors (false positives/negatives) and overall model performance across classes.
- **Precision**: Measures the accuracy of positive predictions.
- **Recall**: Assesses the model's ability to identify all positive samples.
- **F1-Score**: Balances precision and recall.
- **mAP@50**: Main metric for detecting accuracy in object detection - (IoU at 0.5 threshold).

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

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

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

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

| 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 |

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

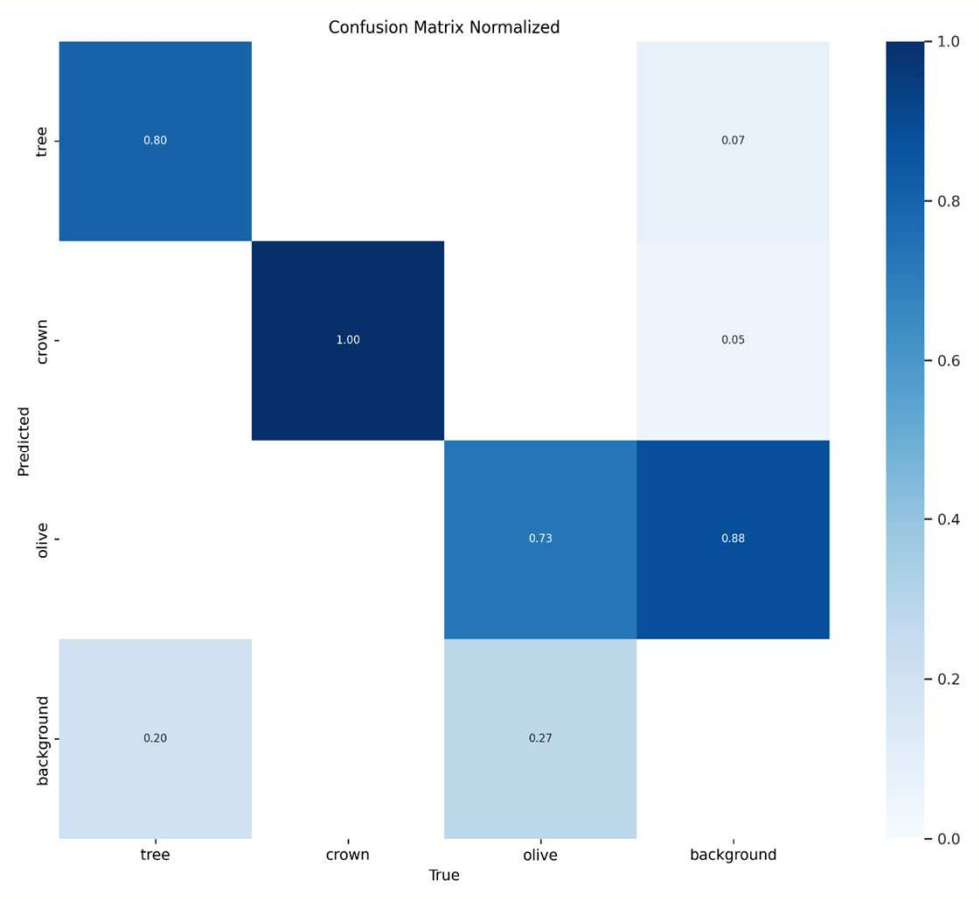



**Table**: Confusion Matrix

# EXPERIMENT RESULTS – *DETECTION-MODEL PERFORMANCE EVALUATION*

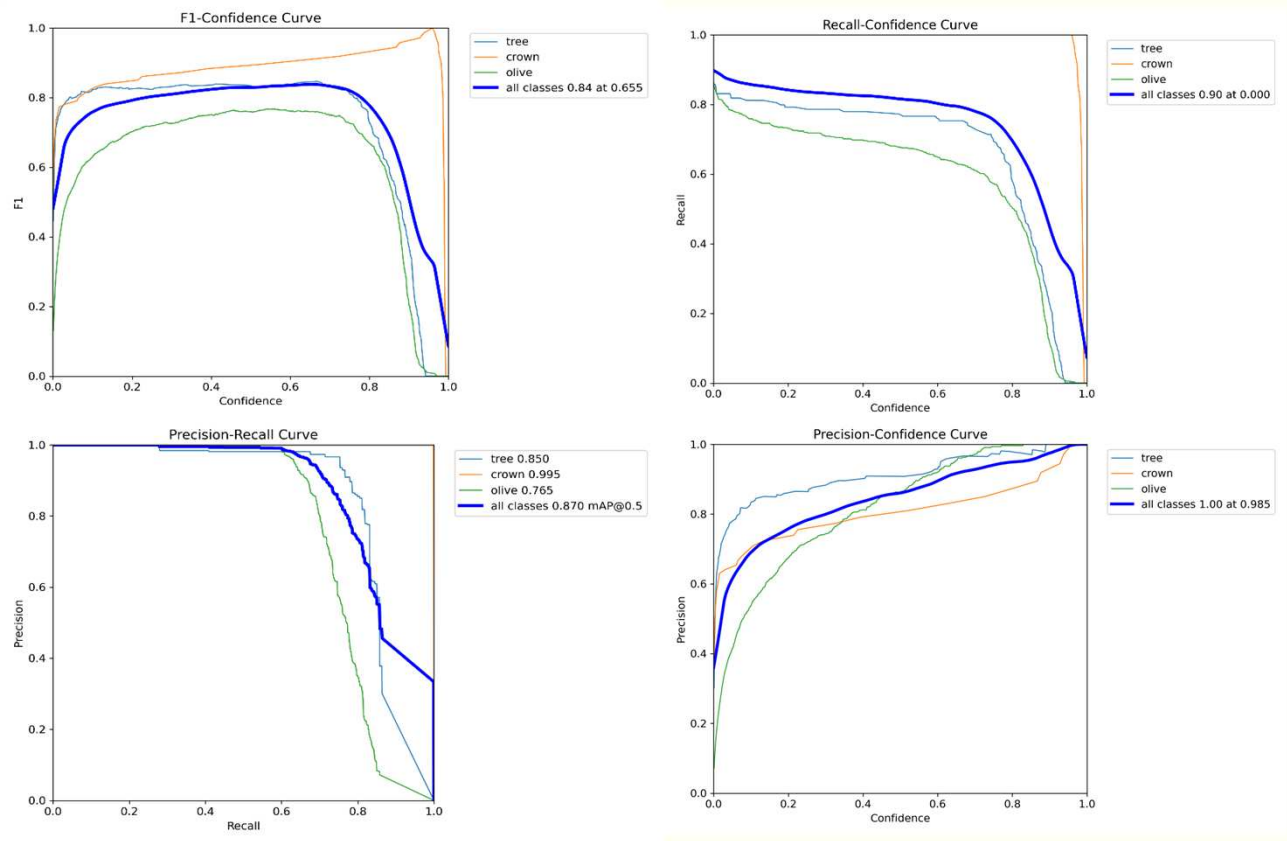



**Table Graph**: Precision, Recall and F1-Score Precision-Recall Curve

# EXPERIMENT RESULTS – *COUNTING PERFORMANCE EVALUATION*

## Evaluation Metrics:

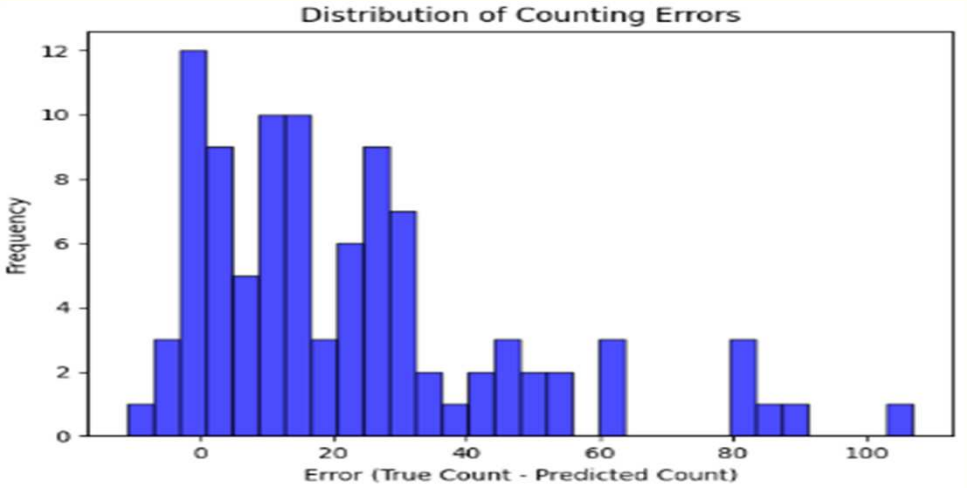
- **Mean Squared Error**: To quantify the error by comparing the *predicted* values with the *ground-true.*

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

- **Error distribution**: To observe the error frequency of the model in the counting phase

| 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 |

**Table**: (average) MSE ± standard deviation (RMSD)




**Histogram**: YoloV8 Small Counting Error Distribution (On-Tree Scenario)

# CONCLUSION

- **Promising Results**:
  - The system effectively **detects and counts olives**, closely imitating expert field procedures with good precision.
- **Areas for Improvement**:
  - **Dataset Expansion**: Increasing the **size** and **diversity** of the training data to enhance accuracy and reliability
  - **Detection Phase**: Use a **Two-Stage** detector model for better precision.
  - **Counting Phase**:
    - Use a **Direct Regression-based** neural network to reduce error impact from the detection phase.
    - Additional metrics like Absolute Error or **Mean Absolute Percentage Error (MAPE)** to gain deeper insights.
- **Future Enhancements**:
  - Future works could involve training a custom detection model to assess the system's feasibility in mobile application contexts.


THANKS
FOR
YOUR
ATTENTION

# REFERENCES

[1]. Olive-SmartScan_AI: An Object Detection-Based Counting System for both On-Tree and Off-Tree Olives; Francesco Bruno, Gaetano Sferrazza; https://github.com/g-sferr/Olive-SmartScan_AI/tree/main/docs