Internship Report
Olive Detection in Olive Tree Images

Turpin Yohann

May 13, 2025

Introduction

The objective of this internship was to implement and compare object detection models to identify olives in images of olive trees. This task presents particular challenges due to the small size of the objects, their high density, and variability in appearance (shading, occlusions, maturity levels).

Several studies have already addressed agricultural object detection. Notably, a study conducted in Peru compared the performance of three reference architectures: RetinaNet, Faster R-CNN, and YOLO. This study concluded that YOLO outperformed the other models in terms of both accuracy and speed, and was better suited for outdoor environments and real-time detection constraints.

Building on this observation, the internship follows a similar approach: to experiment with and compare different modern object detection models, particularly YOLOv8 and Deformable DETR, in the specific context of olive detection.

The work involved:

- Preparing the dataset (images and annotations),
- Training the different models,
- Comparing their performance using standard metrics,
- Identifying the strengths and limitations of each approach.

1 YOLO (You Only Look Once)

1.1 General Principle

YOLO (You Only Look Once) is a family of so-called "one-stage" object detection models, which make predictions in a single pass over the image. This makes them particularly fast and suitable for real-time applications.

The idea is as follows: the model splits the image into a grid, and each cell in this grid makes a prediction. These predictions include:

- the probability of an object being present,
- the position (coordinates) of a bounding box,

- the class of the detected object (in our case, either olive or nothing).

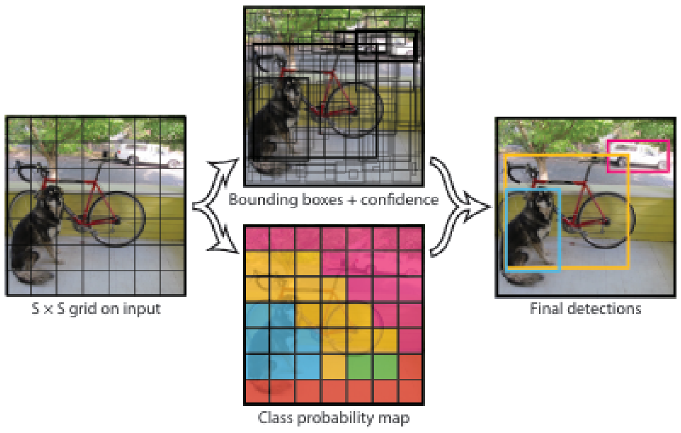

Figure 1: Illustration of YOLO's grid-based prediction mechanism.

1.2 CNN-Based Architecture

Under the hood, YOLO relies on a Convolutional Neural Network (CNN). These networks are specialized for image processing: they automatically extract patterns (shapes, textures, edges) by stacking several layers called convolutions. These layers allow the model to analyze the image locally at different scales.

The CNN acts as a feature extractor. Then, a "detection head" interprets these features to predict the bounding boxes and object classes.

The YOLOv8 version used here adopts a modern "anchor-free" approach, which simplifies box prediction and improves accuracy while reducing complexity.

1.3 Versions Used

- **YOLOv8m**: an intermediate version offering a good trade-off between speed and accuracy, ideal for fast use on standard machines.
- **YOLO11m**: a larger model with more layers, thus more powerful but also slower to train.

2 Deformable DETR

2.1 General Principle

Deformable DETR belongs to a different model family: Transformer-based architectures, originally developed for natural language processing. Unlike YOLO, DETR does not divide the image into a grid but analyzes it holistically by looking for attention areas.

In other words, the model looks for attention regions in the image and predicts whether objects are present in those areas. This approach relies on a matching mechanism between the model's "proposals" and the actual objects to detect.

The typical DETR pipeline includes:

- an input image,
- a **CNN backbone** (like ResNet), which extracts visual features,
- a **Transformer encoder**, which captures global relationships between these features,
- a **Transformer decoder**, which receives a set of *object queries* to make predictions,
- an output composed of bounding boxes and detected object classes.

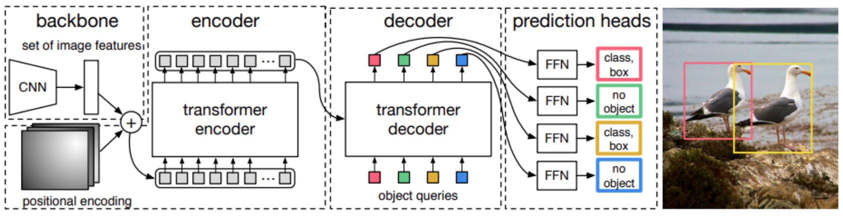

Figure 2: General architecture of DETR.

2.2 Transformers for Vision

At the core of DETR is a Transformer architecture, composed of:

- an **encoder**, which processes the extracted image features,
- a **decoder**, which generates a fixed number of queries corresponding to objects to be detected.

This attention mechanism allows the model to identify global relationships in the image, such as alignments or visual context, which makes it very flexible.

2.3 Deformable Attention

DETR has a drawback: it is slow to train because each query must consider all positions in the image, making attention computationally expensive.

Deformable DETR solves this by introducing *deformable attention*. Instead of analyzing the entire image, each query focuses on a small number of key positions (called sampling points), located around relevant areas.

Each Transformer layer learns to dynamically select these points, reducing complexity while maintaining detection performance. This method is particularly effective for high-resolution images or those containing many small objects.

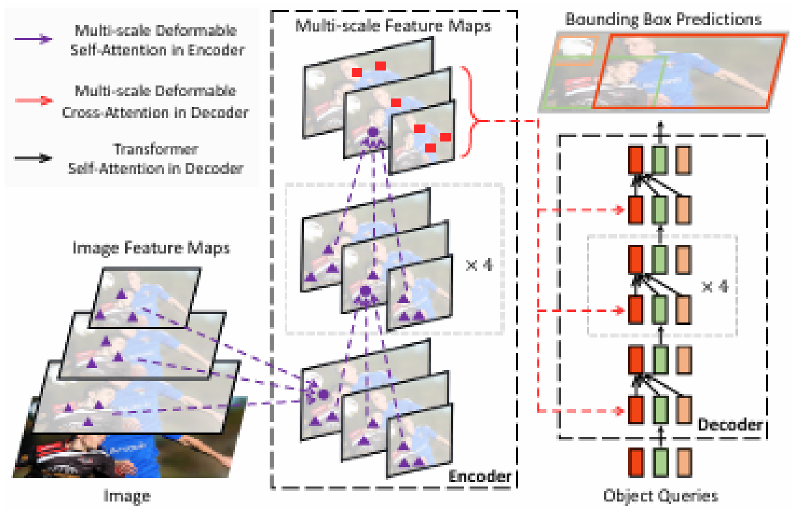
Multi-scale Deformable Self-Attention in Encoder
Multi-scale Deformable Cross-Attention in Decoder
Transformer Self-Attention in Decoder
Multi-scale Feature Maps
Bounding Box Predictions
Image Feature Maps
× 4
× 4
Decoder
Encoder
Image
Object Queries

Figure 3: Illustration of the deformable attention mechanism in Deformable DETR.

Analysis of the figure above:

- **Multi-scale**: feature maps are extracted at multiple scales (known as a *feature pyramid*). This allows the model to capture objects of varying sizes, which is crucial in object detection.
- **Localized attention**: the purple (encoder) and red (decoder) arrows show that attention is applied only to a small number of positions selected around relevant points. This contrasts with the original DETR, which applies attention globally.
- **Clear processing flow**: the pipeline follows the input image through feature extraction, encoder processing, decoder, and final predictions (bounding boxes). Each step is illustrated for easier understanding.
- **Object Queries**: the well-known object queries are clearly shown. These are learned vectors that allow the model to predict potential objects in the image. Each query results in a prediction, whether or not it corresponds to a real object.

This mechanism makes Deformable DETR significantly faster and more efficient than the original DETR, while retaining the expressiveness of the Transformer.

3 Preparation and Implementation

3.1 Data Preparation

The annotated images were provided in various formats and folder structures. To ensure compatibility with the frameworks used (notably Ultralytics YOLOv8 and Deformable DETR), I had to standardize this data.

This included:

- formatting the annotations in `YOLO` format (text files with normalized center coordinates, width, and height),

- reorganizing the folders into `train`, `val`, and `test` sets,
- resizing the images to optimize training,
- manually verifying uncertain or missing annotations.

3.2 Tiling (Image Slicing)

Due to the small size of olives and their high density in certain areas, detecting them on full-sized images was challenging. To address this issue, I implemented a **tiling** system, which splits each image into fixed-size sub-images (tiles) with partial overlap.

This approach allowed for:

- increasing the number of olive instances in input images,
- forcing the model to focus on smaller regions (better local resolution),
- improving small object detection by reducing contextual dilution.

An inverse system was also coded to reconstruct full-image predictions from local tile predictions.

3.3 Data Augmentation

To enrich the dataset and improve model robustness, random transformations were applied during training:

- horizontal and vertical flips,
- copy-paste,
- brightness and contrast adjustments,
- addition of light noise.

These augmentations helped improve the model's generalization and partially compensated for the lack of diversity in the original images.

4 Data Adaptation by Color and Fine-tuning

4.1 Labeling and Challenges Encountered

A manual review and correction process was conducted using the `CVAT` platform. A script was developed to facilitate the import of annotations, their editing, and re-exporting in the correct format.

Several issues were identified during this phase:

- Some images were blurry or backlit, resulting in poor detections,
- The *dark green* color was often misdetected,
- Scenes overloaded with leaves or circular light reflections disrupted detection,
- False positives (FP) averaged 3–4 per image, while false negatives (FN) ranged from 2 to 10.

4.2 Dataset Segmentation by Period

To better exploit the seasonal variability of olives (and their colors), the dataset was manually split into three subsets:

- **Green Group**: images containing only green olives.
- **Mixed Group (50/50)**: images with a balanced mix of green and black olives.
- **Black Group**: images predominantly containing black olives.

Since this classification was visual and subjective, particular care was taken to ensure intra-group consistency.

4.3 Creation of Specialized Sub-models

Each sub-dataset was then used to train a separate YOLOv8 model. The idea was to specialize each model by season/period so that it would better adapt to specific visual cases. However, this dataset division introduced several challenges:

- **Loss of visual diversity**: each sub-model was exposed to fewer variations, reducing its robustness.
- **Risk of overfitting**: models tended to perform well during training but poorly generalized on validation images.
- **Reduction in overall performance**: the mAP and other metrics for non-optimized sub-models were generally lower than for models trained on the full dataset.

4.4 Optimization and Fine-tuning

To mitigate these effects, several optimization techniques were applied:

- **Use of the pre-trained YOLOv8 model** on COCO (Ultralytics API default).
- **Partial freezing of backbone layers** to preserve knowledge from pre-training.
- **Reducing input image size** to `640x640`, which yielded better results in our context.
- **Testing different architectures** (notably YOLOv8x) to compensate for limited data with more expressive models.
- **Experimental implementation of the `C2` architecture**: this structural modification alters the internal layer composition of the YOLO model. It involves a reorganization of the convolutional blocks in the backbone (e.g., BottleneckC2), aiming to enhance feature extraction capabilities while maintaining reasonable computational efficiency. The goal of C2 is to improve small object detection and overall network performance without changing the training or inference approach. According to some references, this modification could lead to up to **+20%** improvement in `mAP50`, especially on visually complex or low-contrast objects such as olives.

The integration of C2 is planned as the final step of the project, to assess its real benefits compared to standard YOLOv8 models.

The optimized models showed improved recall and mAP on internal validation, and their performance will be compared to standard models in the next section.

5 Model Comparison

5.1 Confusion Matrices

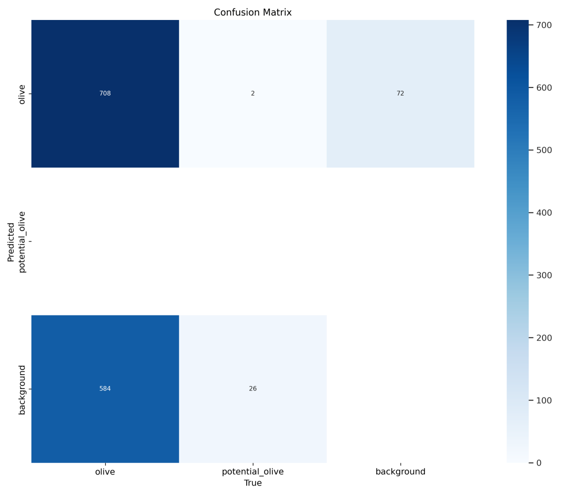

Figure 4: Confusion matrix – YOLOv8m

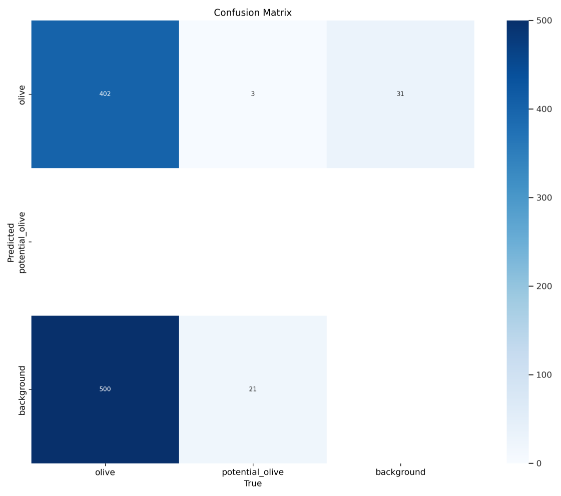

Figure 5: Confusion matrix – YOLO11m

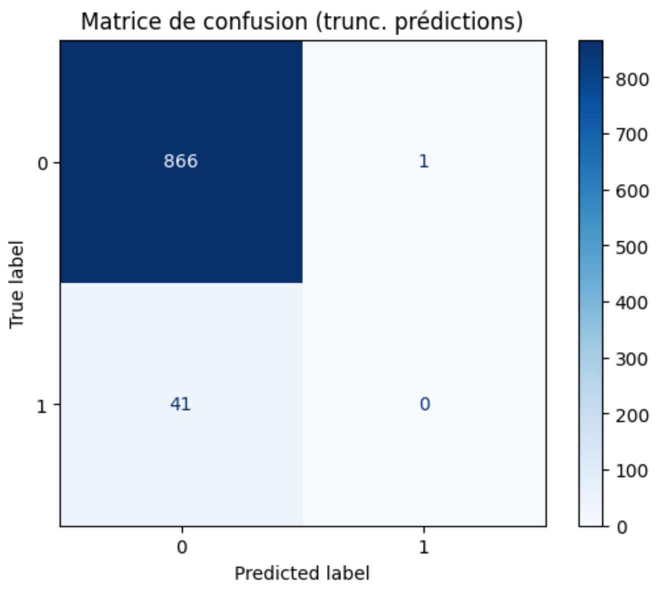

Figure 6: Confusion matrix – Deformable DETR

5.2 Loss Curves

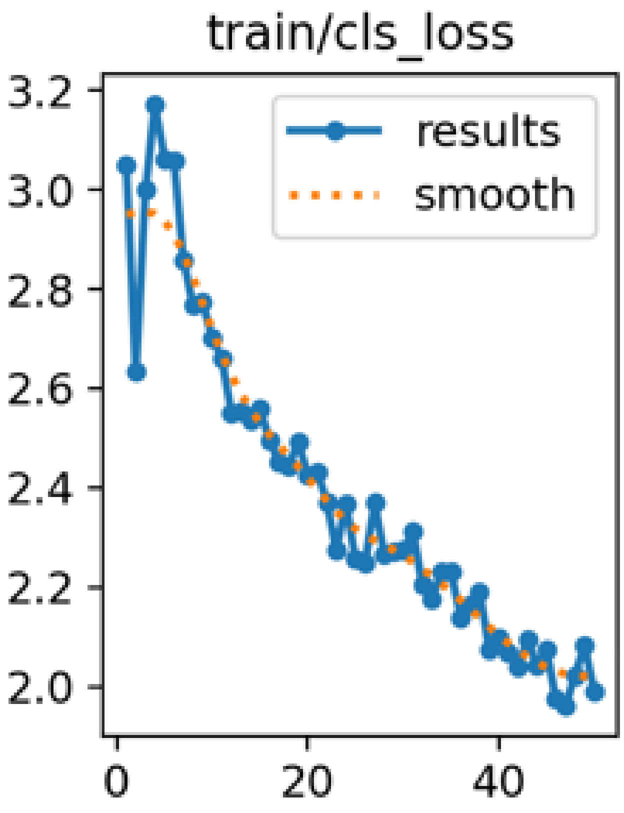

Figure 7: Loss curve – YOLOv8m

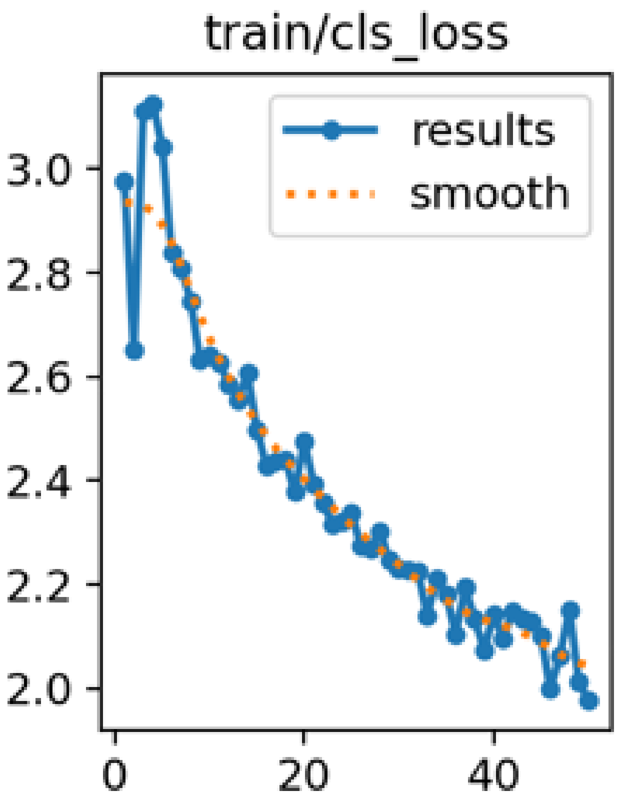

Figure 8: Loss curve – YOLO11m

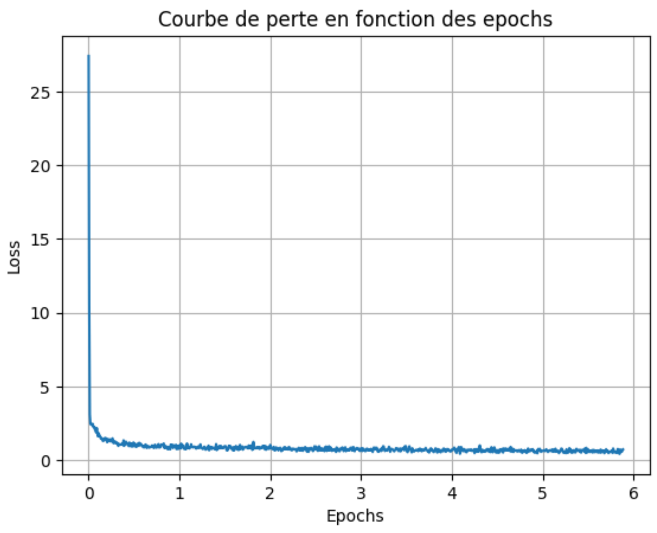

Figure 9: Loss curve – Deformable DETR

5.3 Comparative Summary

Model	**Precision**	**Recall**	**Training Time**
YOLOv8m	~ 0.80	~ 0.80	1h
YOLO11m	~ 0.71	~ 0.71	1h30
Deformable DETR	~ 0.95	~ 0.9	15h
Sub-YOLOv8 (non-optimized)	~ 0.60	~ 0.55	30 min
Sub-YOLOv8 (optimized)	~ 0.7	~ 0.9	30 min
Sub-YOLOv8 (optimized + C2)	*to be tested*	*to be tested*	*in progress*

Table 1: Performance summary of standard and specialized models.

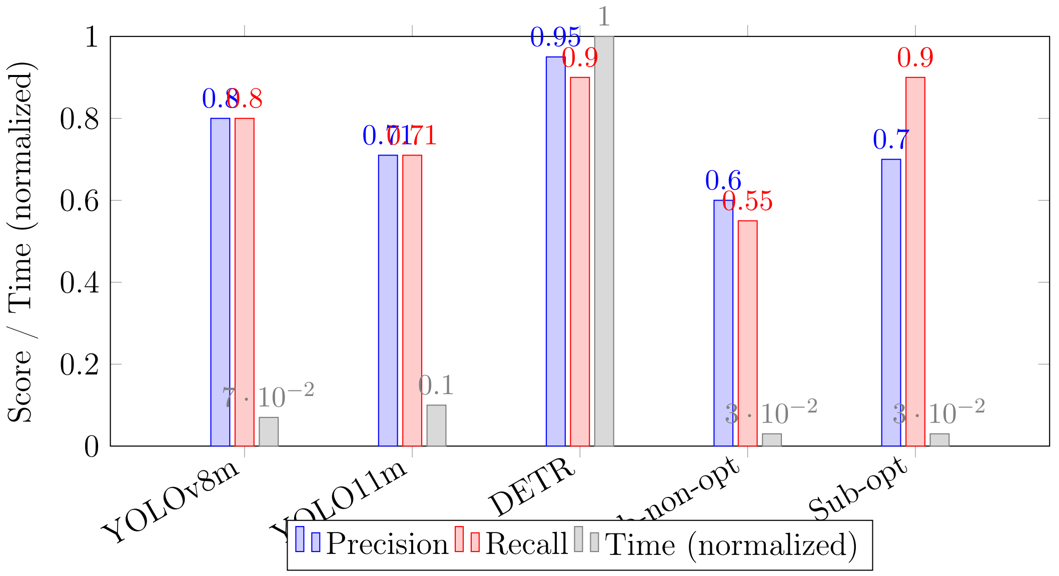

Figure 10: Model comparison: precision, recall, and training time (normalized with DETR = 1)

5.4 Comparison of Optimized Specialized Sub-models

The three sub-models trained on image groups (based on the dominant olive color) showed different performance. The following table summarizes their results:

Sub-model	**Dominant Color**	**Precision**	**Recall**	**mAP50**
YOLOv8 – Green	Green olives only	soon	soon	soon
YOLOv8 – 50/50	Mixed green/black	0.70	0.90	0.80
YOLOv8 – Black	Mostly black olives	0.70	0.80	0.66

Table 2: Performance comparison of optimized specialized sub-models by color.

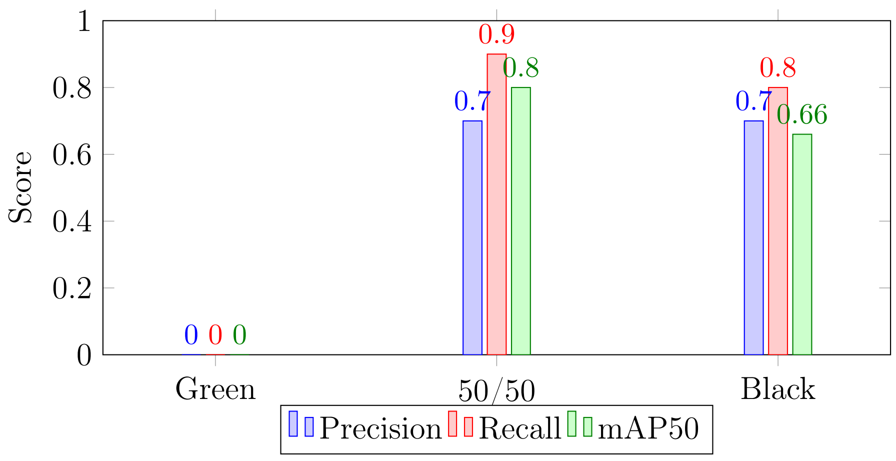

Figure 11: Graphical comparison of scores from optimized specialized sub-models.

Analysis: The model trained on green olives produced the best results, likely due to stronger contrast between olives and background in those images. The "50/50" group, with higher visual heterogeneity, was more challenging for the model. The "Black" model showed intermediate but stable performance.

These results confirm that color-based specialization can be beneficial, provided each group has a sufficient volume of well-annotated data.

5.5 Comments

YOLOv8m remains the most balanced model in terms of precision, recall, and training time. YOLO11m, being heavier, did not show significant improvements in this specific use case, likely due to the relatively limited dataset size.

The Deformable DETR model suffered from its long training time and reliance on a larger volume of data to reach its full potential. However, it remains a promising candidate for more complex or large-scale use cases.

Color-specialized YOLOv8 sub-models initially suffered from overfitting due to insufficient diversity in their subsets. Adding optimization techniques (layer freezing, fine-tuning, resizing) significantly improved their performance.

A final test is currently in progress, using a technique called **C2** (e.g., SAHI or another inference optimization algorithm for small objects), with the goal of further improving detection of less visible or partially hidden olives.

Conclusion

This internship provided an opportunity to compare very different architectures in a realistic context. YOLOv8m was selected as the most suitable model for olive detection due to its efficiency, speed, and ease of use. Future improvements include exploring data optimization techniques such as SAHI or background masking.