Rapport de Stage
Détection d'Olives sur Images d'Oliviers

Turpin Yohann

May 13, 2025

Introduction

Ce stage a pour objectif la mise en œuvre et la comparaison de modèles de détection d'objets pour identifier des olives sur des images d'oliviers. Ce problème pose des défis particuliers, notamment en raison de la petite taille des objets, de leur forte densité, et de la variabilité de leur apparence (ombrage, occlusions, variations de maturité).

Un certain nombre d'études ont déjà été menées sur la détection d'objets agricoles. Notamment, une étude réalisée au Pérou a comparé les performances de trois architectures de référence : RetinaNet, Faster R-CNN et YOLO. Cette étude a conclu que YOLO surpassait les autres modèles en termes de précision et de rapidité, tout en étant plus adapté aux environnements en extérieur et aux contraintes de détection en temps réel.

En s'appuyant sur ce constat, ce stage s'inscrit dans une démarche similaire : expérimenter et comparer différents modèles de détection d'objets modernes, en particulier YOLOv8 et Deformable DETR, dans le contexte spécifique de la détection d'olives.

Le travail a consisté à :

- Préparer les données (images et annotations),
- Entraîner les différents modèles,
- Comparer leurs performances à l'aide de métriques classiques,
- Identifier les points forts et les limites de chaque approche.

1 YOLO (You Only Look Once)

1.1 Principe général

YOLO (You Only Look Once) est une famille de modèles de détection d'objets dits « one-stage », qui effectuent les prédictions en une seule passe sur l'image. Cela les rend particulièrement rapides et adaptés à des usages en temps réel.

Le principe est le suivant : le modèle scinde l'image en une grille, et chaque case de cette grille effectue une prédiction. Ces prédictions comprennent :

- la probabilité de présence d'un objet,
- la position (coordonnées) d'une boîte englobante,

- la classe de l'objet détecté (dans notre cas, une olive ou rien).

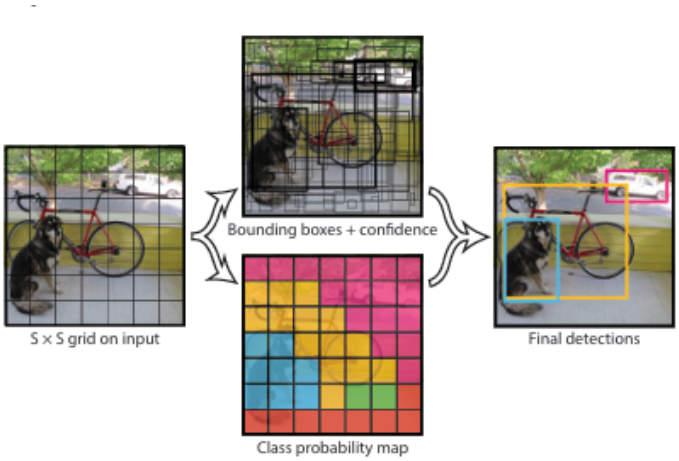

Figure 1: Illustration du découpage en grille dans YOLO.

1.2 Architecture basée sur les CNN

Sous le capot, YOLO repose sur un réseau de neurones convolutifs (CNN). Ces réseaux sont spécialisés dans le traitement des images : ils extraient automatiquement les motifs (formes, textures, contours) en empilant plusieurs couches appelées convolutions. Ces couches permettent d'analyser localement l'image à différentes échelles.

Le CNN agit donc comme un extracteur de caractéristiques. Ensuite, une « tête de détection » interprète ces caractéristiques pour prédire les boîtes et les classes des objets.

La version YOLOv8 utilisée ici adopte une approche moderne dite « anchor-free », qui simplifie la prédiction des boîtes et améliore la précision tout en réduisant la complexité.

1.3 Versions utilisées

- **YOLOv8m** : une version intermédiaire qui offre un bon compromis entre vitesse et précision, idéale pour un usage rapide sur des machines standards.
- **YOLO11m** : un modèle plus large, avec plus de couches, donc plus puissant mais aussi plus lent à entraîner.

2 Deformable DETR

2.1 Principe général

Deformable DETR appartient à une autre famille de modèles : les architectures à base de Transformers, issues initialement du traitement du langage naturel. Contrairement à YOLO, DETR ne découpe pas l'image en grille, mais l'analyse globalement en cherchant des zones d'attention.

Autrement dit, le modèle cherche dans l'image des zones d'attention et prédit s'il y a un objet à ces emplacements. Ce fonctionnement repose sur un mécanisme d'appariement entre ce que le modèle « propose » et les objets réels à détecter.

Le pipeline typique de DETR inclut :

- une image en entrée,
- un **backbone CNN** (comme ResNet), qui extrait les caractéristiques visuelles (features),
- un **encodeur Transformer**, qui capture les relations globales entre ces caractéristiques,
- un **décodeur Transformer**, qui reçoit un ensemble de *object queries* (requêtes d'objet) pour faire des prédictions,
- une sortie composée de boîtes englobantes et de classes d'objets détectés.

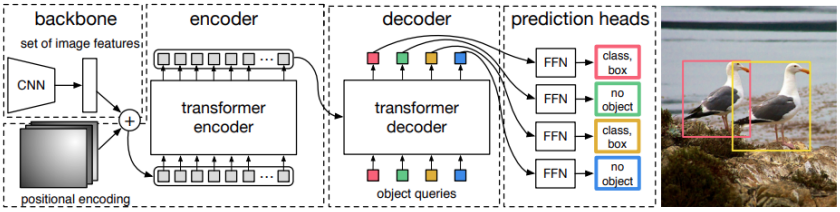

Figure 2: Schéma général du fonctionnement de DETR.

2.2 Transformers pour la vision

Le cœur du modèle DETR est une architecture Transformer, composée :

- d'un **encodeur**, qui traite les caractéristiques extraites de l'image,
- d'un **décodeur**, qui génère un nombre fixe de requêtes (queries) correspondant aux objets à détecter.

Ce mécanisme d'attention permet au modèle d'identifier des relations globales dans l'image, comme des alignements ou des contextes visuels, ce qui le rend très flexible.

2.3 Attention déformable

DETR a cependant un défaut : il est long à entraîner car chaque requête doit prendre en compte toutes les positions de l'image, ce qui rend l'attention coûteuse en temps de calcul.

Deformable DETR corrige ce problème en introduisant une *attention déformable*. Plutôt que de traiter toute l'image, chaque requête n'analyse qu'un petit nombre de positions clés (appelées points d'échantillonnage), localisées autour d'endroits pertinents.

Chaque couche du Transformer apprend à choisir dynamiquement ces points, ce qui réduit la complexité tout en maintenant la performance de détection. Cette méthode est particulièrement efficace pour les images à haute résolution ou contenant beaucoup d'objets petits.

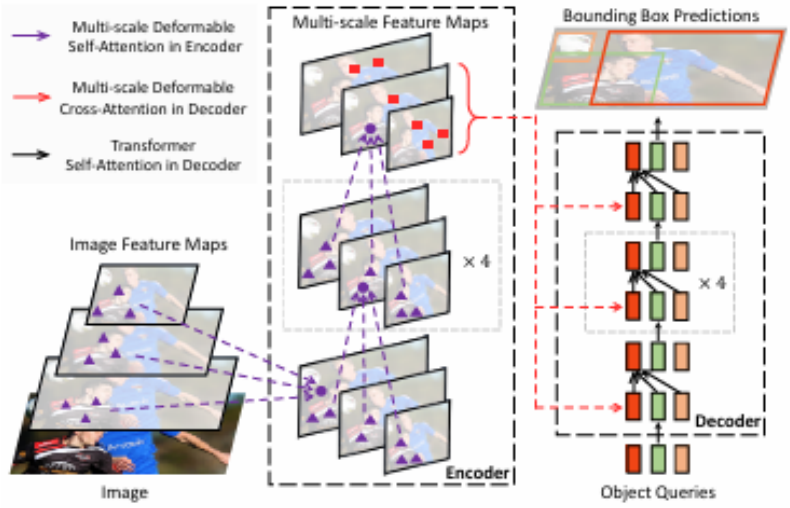

Figure 3: Illustration du mécanisme d'attention déformable dans Deformable DETR.

Analyse de l'illustration ci-dessus :

- **Multi-échelle** : les cartes de caractéristiques sont extraites à plusieurs échelles (on parle de *feature pyramid*). Cela permet au modèle de capturer des objets de tailles variées, ce qui est crucial en détection d'objets.
- **Attention localisée** : les flèches violettes (dans l'encodeur) et rouges (dans le décodeur) montrent que l'attention n'est appliquée que sur un nombre réduit de positions, sélectionnées autour de points jugés pertinents. Cela contraste avec le DETR original qui applique une attention globale sur toute l'image.
- **Flux de traitement clair** : on suit le pipeline depuis l'image d'entrée, passant par l'extraction des features, le traitement par l'encodeur, puis le décodeur, jusqu'aux prédictions finales (boîtes englobantes). Chaque étape est illustrée, ce qui rend la compréhension plus intuitive.
- **Object Queries** : les fameuses requêtes d'objet (*object queries*) sont bien visibles. Ce sont des vecteurs appris qui permettent au modèle de prédire des objets potentiels dans l'image. Chaque requête produit une prédiction, qu'elle corresponde à un objet réel ou non.

Ce mécanisme rend Deformable DETR beaucoup plus rapide et efficace que le DETR original, tout en conservant l'expressivité du Transformer.

3 Préparation et Implémentation

3.1 Préparation des données

Les images annotées m'ont été fournies sous différents formats et structures de dossiers. Pour assurer la compatibilité avec les frameworks utilisés (notamment Ultralytics YOLOv8 et Deformable DETR), j'ai dû uniformiser ces données.

Cela a inclus :

- le formatage des annotations au format `YOLO` (fichiers texte avec centre, largeur, hauteur normalisés),
- la réorganisation des dossiers en ensembles `train`, `val` et `test`,
- le redimensionnement des images pour optimiser l'entraînement,
- la vérification manuelle des annotations douteuses ou manquantes.

3.2 Tiling (découpage d'images)

La petite taille des olives, couplée à leur densité élevée dans certaines zones, rendait difficile leur détection sur des images complètes. Pour pallier ce problème, j'ai mis en place un système de **tiling**, consistant à découper chaque image en sous-images (tuiles) de taille fixe, avec un chevauchement partiel.

Cela a permis :

- d'augmenter le nombre d'exemples d'olives dans les images d'entrée,
- de forcer le modèle à se concentrer sur des zones plus petites (meilleure résolution locale),
- d'améliorer la détection de petits objets en réduisant la dilution contextuelle.

Un système inverse a également été codé pour reconstruire les prédictions sur l'image complète à partir des prédictions locales.

3.3 Augmentation des données

Pour enrichir le dataset et améliorer la robustesse des modèles, des transformations ont été appliquées de manière aléatoire durant l'entraînement :

- flips horizontaux et verticaux,
- copy-paste,
- modification de la luminosité et du contraste,
- ajout de bruit léger.

Ces augmentations ont permis d'améliorer la généralisation du modèle et de compenser partiellement le manque de diversité dans les images de départ.

4 Adaptation des Données par Couleur et Fine-tuning

4.1 Labeling et difficultés rencontrées

Un travail de relecture et de correction manuelle a été effectué via la plateforme `CVAT`. Pour cela, un script a été créé afin de faciliter l'importation des annotations, leur édition, puis leur réexportation dans le bon format.

Plusieurs problèmes ont été identifiés durant cette phase :

- certaines images floues ou prises à contre-jour produisaient de mauvaises détections,

- la couleur *dark green* était souvent mal détectée,
- les scènes trop chargées en feuilles ou avec des reflets lumineux circulaires perturbaient la détection,
- les faux positifs (FP) étaient en moyenne de 3–4 par image, et les faux négatifs (FN) variaient entre 2 et 10.

4.2 Segmentation du dataset par période

Dans le but d'exploiter au mieux la variabilité saisonnière des olives (et de leurs couleurs), le dataset a été séparé manuellement en trois sous-ensembles :

- **Groupe Vert** : images ne contenant que des olives vertes.
- **Groupe Mixte (50/50)** : images avec un mélange équilibré d'olives vertes et noires.
- **Groupe Noir** : images dominées par des olives de couleur noire.

Ce tri étant visuel et subjectif, une vigilance particulière a été apportée pour maintenir une cohérence intra-groupe.

4.3 Création de sous-modèles spécialisés

Chaque sous-dataset a ensuite été utilisé pour entraîner un modèle YOLOv8 distinct. L'idée était de spécialiser un modèle par saison/période, afin qu'il s'adapte mieux à des cas visuels spécifiques. Cependant, cette division du dataset a soulevé plusieurs défis :

- **Perte de diversité visuelle** : chaque sous-modèle a vu moins de cas différents, ce qui a diminué sa robustesse.
- **Risque de surapprentissage (overfitting)** : les modèles ont tendance à bien performer sur l'entraînement, mais à mal généraliser sur les images de validation.
- **Réduction des performances globales** : le mAP, ainsi que les metriques sur les sous-modèles non optimisés était généralement plus bas que celui des modèles entraînés sur l'ensemble complet.

4.4 Optimisation et fine-tuning

Pour contrer ces effets, plusieurs techniques d'optimisation ont été mises en œuvre :

- **Utilisation du modèle pré-entraîné** YOLOv8 sur COCO (défaut de l'API Ultralytics).
- **Gel partiel des couches du backbone** (freezing), pour éviter d'effacer les acquis du pré-entraînement.
- **Réduction de la taille des images d'entrée** à `640x640`, ce qui a produit de meilleurs résultats dans notre contexte.

- **Tests sur des architectures différentes** (YOLOv8x notamment), afin de compenser la faible quantité de données par un modèle plus expressif.

- **Implémentation expérimentale de l'architecture** `C2` : cette modification structurelle touche directement la composition des couches internes du modèle YOLO. Elle repose sur une réorganisation des blocs convolutionnels du backbone (ex. BottleneckC2), visant à renforcer la capacité d'extraction de caractéristiques tout en maintenant une efficacité de calcul raisonnable.

  L'objectif de C2 est d'améliorer la détection des petits objets et les performances générales du réseau sans changer l'approche d'entraînement ou d'inférence. Selon certaines références, cette modification pourrait entraîner une amélioration jusqu'à $+\mathbf{20\%}$ sur le `mAP50`, en particulier sur des objets visuellement complexes ou peu contrastés comme les olives.

L'intégration de C2 est prévue comme étape finale du projet, afin d'en mesurer les bénéfices réels comparés aux modèles standard YOLOv8.

Les modèles optimisés ont montré une amélioration du rappel et du mAP sur les validations internes, et leur performance sera comparée aux modèles standards dans la section suivante.

5 Comparaison des Modèles

5.1 Matrices de Confusion

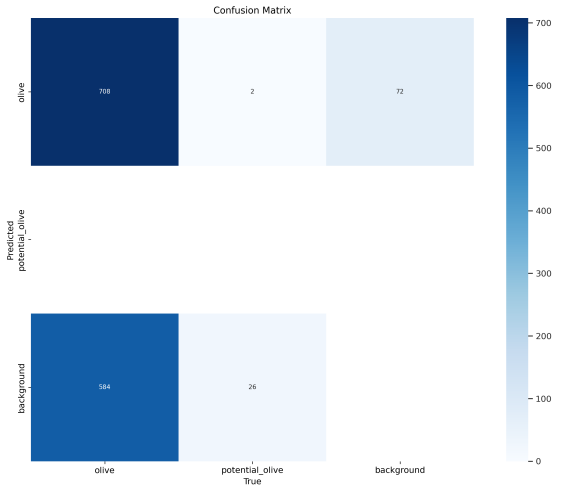

Figure 4: Matrice de confusion – YOLOv8m

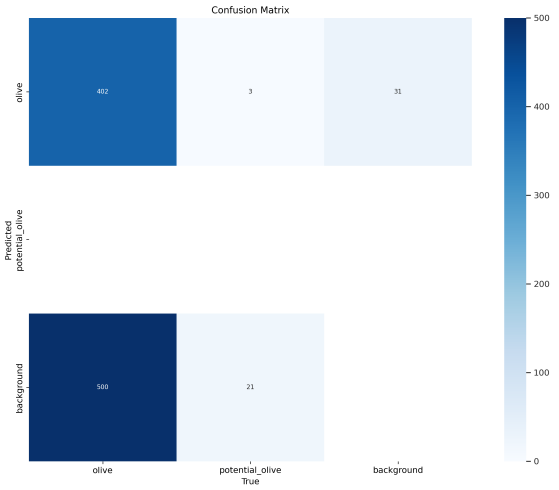

Figure 5: Matrice de confusion – YOLO11m

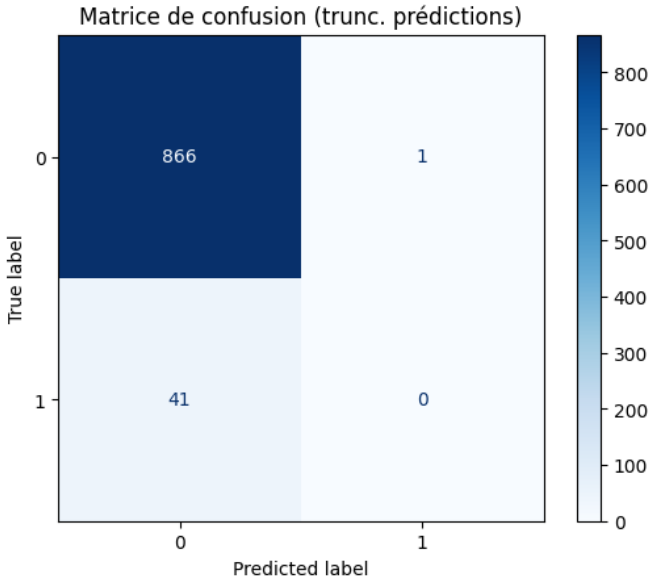

Figure 6: Matrice de confusion – Deformable DETR

5.2 Courbes de Loss

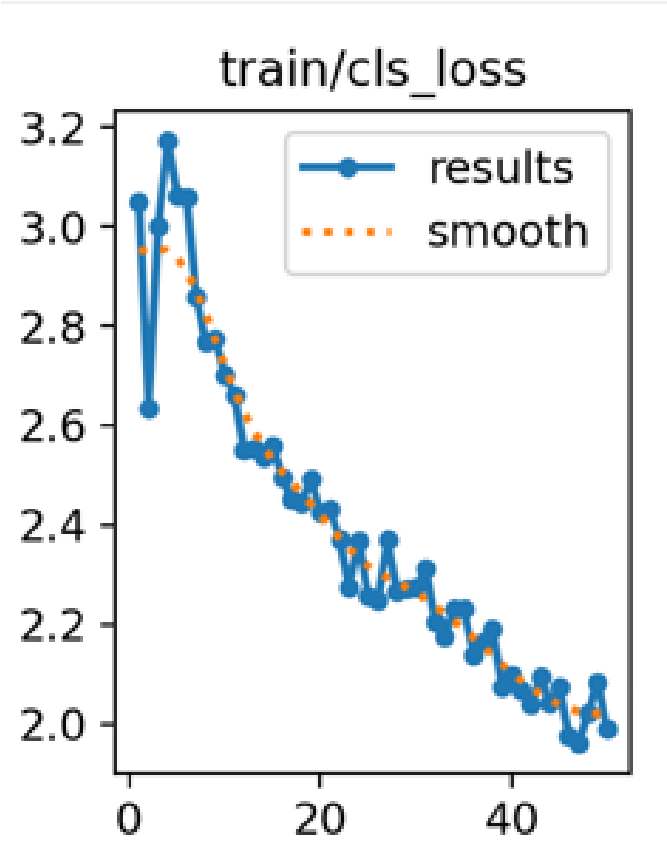

Figure 7: Courbe de loss – YOLOv8m

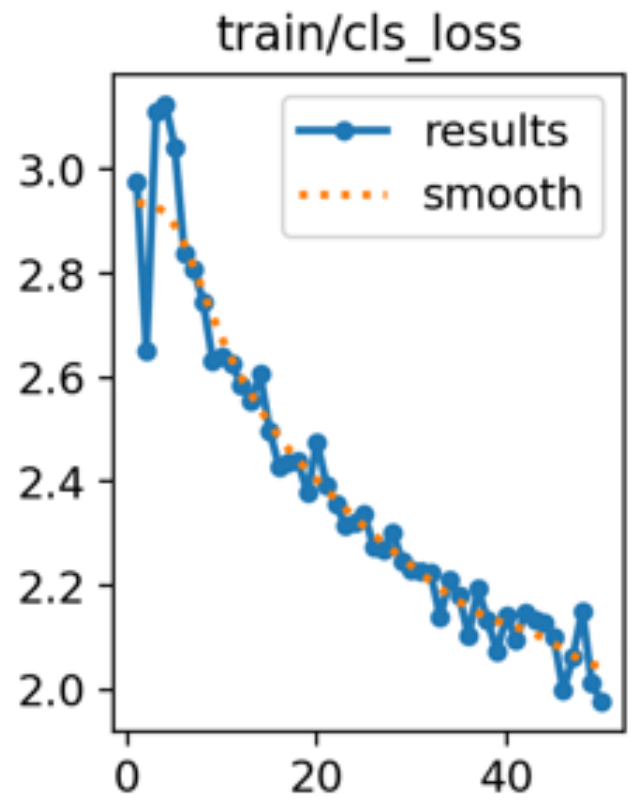

Figure 8: Courbe de loss – YOLO11m

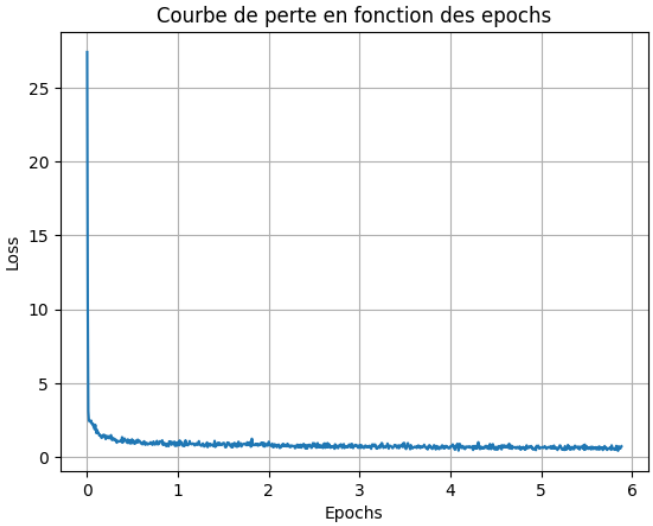

Figure 9: Courbe de loss – Deformable DETR

5.3 Synthèse comparative

Modèle	Précision	Rappel	Temps d'entraînement
YOLOv8m	~ 0.80	~ 0.80	1h
YOLO11m	~ 0.71	~ 0.71	1h30
Deformable DETR	~ 0.95	~ 0.9	25h
Sous-YOLOv8 (non optimisé)	~ 0.60	~ 0.55	30 min
Sous-YOLOv8 (optimisé)	~ 0.7	~ 0.9	30min
Sous-YOLOv8 (optimisé + C2)	*à tester*	*à tester*	*en cours*

Table 1: Résumé des performances comparées des modèles standards et spécialisés.

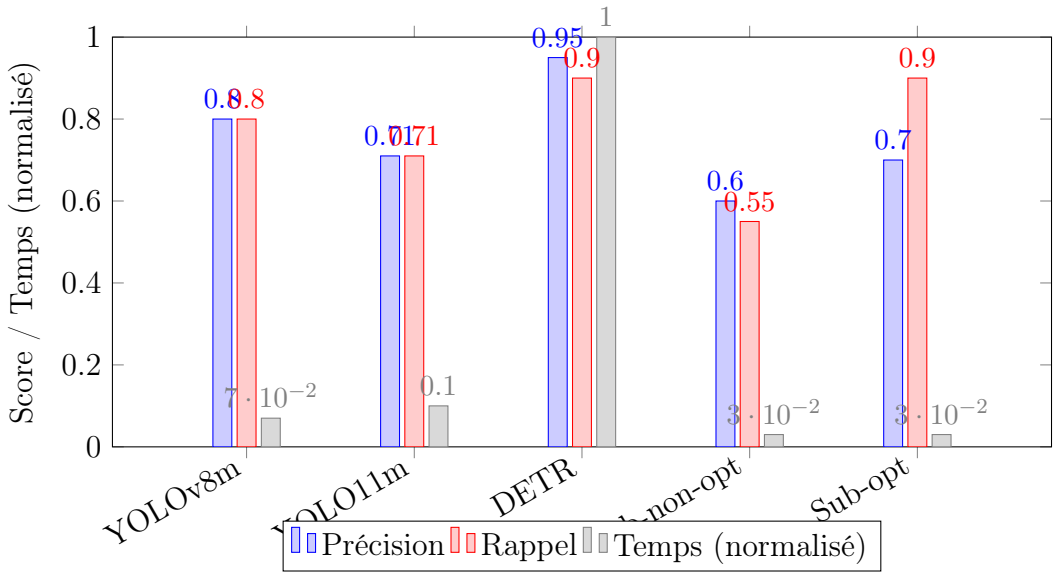

Figure 10: Comparaison des modèles : précision, rappel et temps d'entraînement (temps normalisé sur DETR = 1)

5.4 Comparaison des sous-modèles spécialisés optimisées

Les trois sous-modèles entraînés sur des groupes spécifiques d'images (selon la couleur dominante des olives) ont montré des performances différentes. Le tableau suivant résume leurs résultats :

Sous-modèle	Couleur dominante	Précision	Rappel	mAP50
YOLOv8 – Vert	Olives vertes uniquement	soon	soon	soon
YOLOv8 – 50/50	Mélange vert/noir	0.70	0.9	0.8
YOLOv8 – Noir	Olives noires dominantes	0.7	0.8	0.66

Table 2: Comparaison des performances des sous-modèles spécialisés optimisés par couleur.

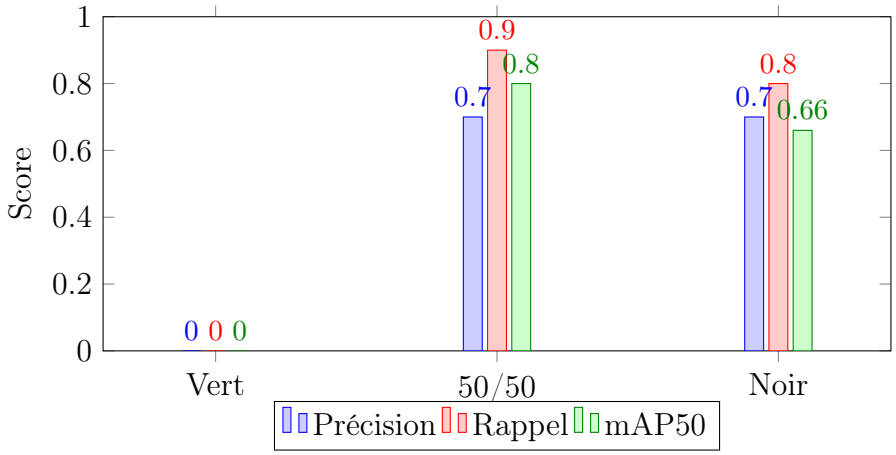

Figure 11: Comparaison graphique des scores des sous-modèles spécialisés optimisés.

Analyse : On remarque que le modèle entraîné sur les olives vertes a donné les meilleurs résultats, probablement grâce à un contraste plus marqué entre les olives et le fond dans ces images. Le groupe "50/50", avec une plus grande hétérogénéité visuelle, a posé davantage de difficultés au modèle. Le modèle "Noir", quant à lui, montre des performances intermédiaires mais stables.

Ces résultats confirment que la spécialisation par couleur peut être bénéfique, à condition que chaque groupe dispose d'un volume suffisant de données bien annotées.

5.5 Commentaires

On observe que YOLOv8m reste le modèle le plus équilibré en termes de précision, rappel et temps d'entraînement. YOLO11m, plus lourd, n'a pas montré de gain significatif sur ce cas spécifique, probablement à cause de la taille relativement limitée du dataset.

Le modèle Deformable DETR a été pénalisé par sa lenteur d'entraînement et sa dépendance à une plus grande quantité de données pour atteindre son plein potentiel. Il reste cependant un candidat intéressant pour des cas plus complexes ou à plus grande échelle.

Les sous-modèles YOLOv8 spécialisés par couleur ont initialement souffert de surapprentissage, faute de diversité suffisante dans leurs sous-ensembles. L'ajout d'optimisations (gel de couches, fine-tuning, redimensionnement) a significativement amélioré leurs performances.

Un dernier test est en cours d'intégration, utilisant une technique appelée **C2** (par exemple SAHI ou autre algorithme d'inférence optimisée pour les petits objets), avec l'objectif d'augmenter encore la détection des olives peu visibles ou partiellement cachées.

Conclusion

Ce stage a permis de confronter des architectures très différentes dans un cadre réaliste. YOLOv8m a été retenu comme le plus adapté pour la détection d'olives, grâce à son efficacité, sa rapidité, et sa simplicité de mise en œuvre. Des perspectives d'amélioration incluent l'exploration de techniques d'optimisation des données via des techniques comme SAHI ou le masquage de fond.