

PrimeurVision

Détection de fruits et légumes

Master 2 Statistiques et Informatique
pour la Science des données



Eugénie BARLET
Perrine IBOUROI

Février 2026

Table des matières

1	Introduction	1
2	Constitution du dataset	1
2.1	Annotation des images collectées	1
2.2	Curation interactive du dataset — Streamlit	1
2.3	Augmentation des données	2
2.4	Répartition train / val / test	2
3	Architecture du modèle : YOLOv8	3
3.1	Pourquoi YOLO ?	3
3.2	Variantes testées	3
3.3	Structure de l'architecture	3
4	Apprentissage par transfert	4
4.1	Principe	4
4.2	Stratégie en deux phases	4
5	Hyperparamètres et justifications	4
6	Métriques d'évaluation	5
7	Résultats	5
7.1	Modèle v1 — YOLOv8n (baseline)	5
7.2	Modèle v2 — YOLOv8s (améliorations)	7
7.3	Comparaison et sélection du modèle	9
7.4	Analyse des résultats (YOLOv8n retenu)	10
8	Conclusion	11
	Sources du dataset	12

1 Introduction

La grande distribution et les marchés alimentaires manipulent quotidiennement une grande variété de fruits et légumes. Automatiser leur identification constitue un enjeu pratique concret : gestion des stocks, caisses automatiques, tri qualité. Ce projet s'inscrit dans ce contexte en développant **PrimeurVision**, un système de détection d'objets capable d'identifier six classes de légumes dans des images réelles.

L'objectif est de réaliser un *fine-tuning* d'un modèle pré-entraîné de type YOLOv8 sur un dataset constitué pour l'occasion. L'apprentissage par transfert permet de bénéficier des représentations visuelles générées apprises sur de larges corpus, tout en les spécialisant efficacement sur notre domaine cible avec un volume de données limité.

Le dataset couvre six classes : **carotte, aubergine, citron, pomme de terre, radis et tomate**. L'ensemble du pipeline — construction du dataset, entraînement, évaluation — est documenté dans deux notebooks Jupyter dédiés.

2 Constitution du dataset

La robustesse d'un modèle de détection dépend de la diversité et du volume de son jeu de données. Pour ce projet, nous avons adopté une approche hybride combinant des données pré-existantes et une collecte ciblée.

Le dataset final de 238 images est issu de la fusion de trois sources distinctes :

- **Roboflow** : jeux de données déjà étiquetés pour les classes citron et tomate.
- **LVIS Fruits and Vegetables Dataset** (Kaggle) : dataset issu du corpus LVIS (Large Vocabulary Instance Segmentation) dont nous avons extrait et curé les annotations correspondant à nos six classes.
- **Collecte manuelle** (Google Images et Unsplash) : images collectées sur le web pour compléter les catégories en minorité dans le dataset (radis, aubergines etc.) pour lesquelles la présence dans des datasets existants est plus rare.

L'ensemble a été uniformisé : chaque image est associée à un fichier `.txt` au format YOLO, et les identifiants de classes ont été remappés pour être cohérents entre toutes les sources.

2.1 Annotation des images collectées

Pour les images issues de la collecte manuelle ne disposant pas de labels, une phase d'annotation a été nécessaire afin de les intégrer au dataset global.

Nous avons utilisé la plateforme open-source **CVAT (Computer Vision Annotation Tool)**. Le workflow a été le suivant : définition des labels (*eggplant, carrot, potato, lemon, raddish*), importation des images brutes, et détourage par *bounding boxes*.

L'annotation a été réalisée manuellement via l'interface de CVAT pour garantir une précision optimale. Pour chaque instance, nous avons tracé une boîte englobante définie par quatre points cardinaux. Nous avons ajusté chaque cadre pour affleurer les bords du légume en évitant d'inclure du bruit contextuel (arrière-plan). Ceci permet au modèle de dissocier correctement l'objet de son environnement lors de l'apprentissage.

Les annotations ont été exportées au format **YOLO 1.1**. Ce format traduit les coordonnées des boîtes en valeurs numériques normalisées entre 0 et 1.

Une fois l'exportation terminée, les nouveaux fichiers de labels ont été fusionnés avec ceux des sources Roboflow et LVIS. Le répertoire final est organisé avec deux dossiers miroirs (`/images` et `/labels`), accompagnés du fichier `data.yaml` servant de configuration au modèle.

2.2 Curation interactive du dataset — Streamlit

L'intégration de sources hétérogènes (LVIS, Kaggle, Roboflow, collecte manuelle) implique un risque d'erreurs silencieuses : mauvais remapping des identifiants de classes, boîtes mal placées, images hors contexte. Pour détecter et corriger ces problèmes, nous avons développé une interface de revue interactive avec **Streamlit** (`scripts/review_grid.py`).

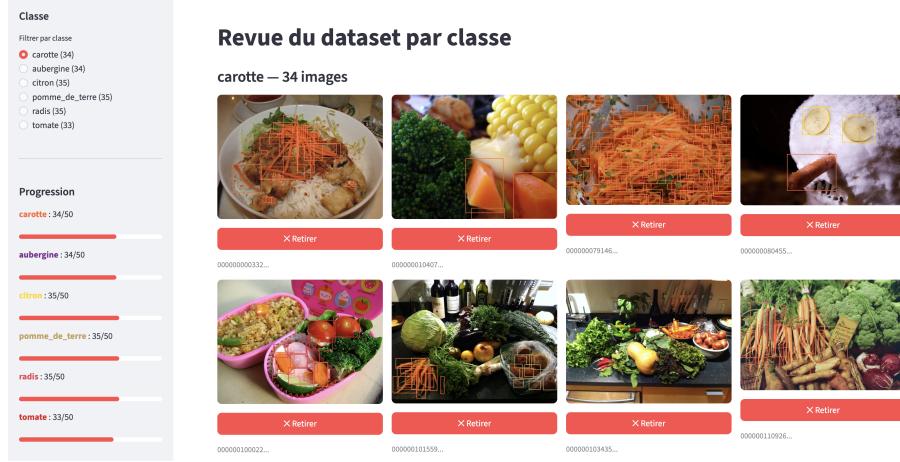


FIGURE 1 – Interface Streamlit de curation du dataset — vue grille par classe avec bounding boxes

L’interface affiche les images organisées **par classe** sous forme de grille. Pour chaque image, les bounding boxes annotées sont dessinées avec la couleur de la classe correspondante. Un bouton *Retirer* permet de supprimer simultanément le fichier image et son label en un clic. Les images contenant plusieurs classes apparaissent dans tous les onglets concernés.

Cet outil a permis de :

1. **Déetecter les erreurs de remapping** — identifier visuellement les images dont les labels ne correspondent pas à ce qui est affiché.
2. **Contrôler la qualité des annotations** — repérer les boîtes trop larges, trop petites ou décalées.
3. **Équilibrer le dataset** — visualiser rapidement les classes sous-représentées et orienter la collecte complémentaire.

2.3 Augmentation des données

Certaines classes présentaient un nombre d’images insuffisant pour garantir un apprentissage robuste, même après avoir ajouté des images individuelles trouvées sur internet. Nous avons donc décidé d’appliquer de l’augmentation de données sur les classes sous-représentées. Étant donné la quantité d’images qu’il fallait transformer, nous avons choisi de réaliser l’augmentation des données via un script Python.

Le script permet d’augmenter la quantité d’images en combinant systématiquement des transformations géométriques (rotation aléatoire entre -40 et $+40$, flip horizontal) et photométriques (luminosité, contraste, saturation), auxquelles s’ajoutent 2 ou 3 transformations supplémentaires tirées aléatoirement parmi : recadrage/zoom, décalage colorimétrique, correction gamma, flou ou accentuation, et bruit. Cette combinaison garantit que chaque image synthétique est visuellement distincte de son originale, tout en restant réaliste.

2.4 Répartition train / val / test

Le dataset a été découpé en trois sous-ensembles distincts. Le jeu d’entraînement sert à ajuster les poids du modèle ; le jeu de validation permet de surveiller la progression à chaque epoch et de détecter un éventuel surapprentissage sans jamais influencer l’apprentissage. À la fin de chaque epoch, le modèle fait des prédictions sur les images de validation. Le jeu de test, mis de côté jusqu’à la fin, fournit une évaluation objective des performances réelles du modèle.

Split	Rôle	Proportion	Nombre d'images
Train	Apprentissage des poids	70%	166
Val	Suivi de l'entraînement	15%	36
Test	Évaluation finale	15%	36

TABLE 1 – Répartition stratifiée du dataset (seed=42)

Le split est stratifié : chaque classe est représentée proportionnellement dans chacun des trois sous-ensembles, soit environ 6 images par classe pour la validation et le test. Le jeu de test n'est jamais utilisé pendant l'entraînement, ce qui garantit que les métriques finales reflètent fidèlement la capacité de généralisation du modèle.

3 Architecture du modèle : YOLOv8

3.1 Pourquoi YOLO ?

Pour la phase de reconnaissance d'objets, nous avons choisi d'utiliser l'architecture YOLO (You Only Look Once). Ce modèle est reconnu pour sa capacité à détecter des objets en une seule analyse de l'image, ce qui le rend particulièrement rapide.

Ce choix s'explique par son bon compromis entre la vitesse d'exécution (compatible avec une utilisation en temps réel), la précision de détection, et la simplicité d'intégration dans un environnement comme Google Colab.

Nous avons retenu YOLOv8 (par Ultralytics), car il s'agit de la version la plus aboutie disponible au moment du projet.

YOLOv8 propose des améliorations par rapport aux versions précédentes, notamment en termes de précision et de stabilité de l'entraînement, tout en conservant une grande rapidité d'inférence.

3.2 Variantes testées

Nous avons entraîné et comparé deux variantes pour trouver le modèle qui nous fournirait les meilleurs résultats :

- **YOLOv8n** (*nano*, 3M paramètres) : modèle léger, adapté à notre dataset de taille limitée.
- **YOLOv8s** (*small*, 11M paramètres) : testé avec des hyperparamètres améliorés (augmentation supplémentaire, label smoothing, plus d'epochs) pour évaluer si plus de capacité apporte un gain.

La comparaison des résultats sur le test a permis de sélectionner le meilleur modèle : YOLOv8n (cf. section Résultats).

3.3 Structure de l'architecture

Le modèle YOLOv8n est organisé en trois grandes parties fonctionnelles :

1. **Backbone (extraction de caractéristiques)** : cette partie analyse l'image d'entrée et extrait progressivement des informations visuelles importantes (contours, textures, formes). Le backbone utilisé est pré-entraîné sur le jeu de données COCO, ce qui permet de bénéficier d'un apprentissage préalable sur un grand nombre d'images variées.
2. **Neck (fusion multi-échelle)** : cette étape combine les informations issues de différentes profondeurs du réseau afin de permettre la détection d'objets de tailles variées.
3. **Tête de détection** : partie finale du modèle, elle prédit pour chaque objet détecté sa position dans l'image (boîte englobante) ainsi que sa classe.

Cette organisation en trois blocs permet de séparer l'extraction d'informations visuelles, leur combinaison et la prédiction finale.

4 Apprentissage par transfert

4.1 Principe

Entraîner un réseau de détection depuis zéro nécessite des centaines de milliers d'images. Notre dataset étant de taille limitée, nous avons recours à l'apprentissage par transfert (*transfer learning*) : on part d'un modèle YOLOv8n pré-entraîné sur COCO (330 000 images, 80 classes) et on adapte ses poids à notre tâche spécifique.

Le modèle a déjà appris à reconnaître des formes, des textures et des objets génériques. Il ne reste qu'à spécialiser ces représentations sur nos six classes de légumes — ce qui nécessite bien moins de données et de temps de calcul.

4.2 Stratégie en deux phases

Nous avons adopté une stratégie d'entraînement en deux phases, pour ne pas dégrader les poids pré-entraînés et éviter le surapprentissage :

Phase 1 — Backbone gelé Le backbone (les 10 premières couches) est gelé : ses poids ne sont pas mis à jour. Seule la tête de détection apprend. Cela permet à la tête de s'adapter à nos classes sans dégrader les représentations génériques déjà apprises. Le learning rate peut être plus élevé car seule une petite partie du réseau est mise à jour.

Phase 2 — Fine-tuning complet Toutes les couches sont dégelées. L'ensemble du réseau est affiné avec un learning rate plus faible pour ne pas effacer les poids pré-entraînés. Cette phase permet d'ajuster finement le backbone à nos images spécifiques (éclairage, fond, angle de vue).

5 Hyperparamètres et justifications

Hyperparamètre	Valeur	Justification
IMG_SIZE	640	Taille standard pour YOLOv8 ; compromis entre résolution spatiale et vitesse d'entraînement. Permet la comparaison avec les benchmarks officiels.
BATCH_SIZE	16	Choisi en fonction des contraintes mémoire afin d'éviter les erreurs de dépassement mémoire.
FREEZE_LAYERS	10	Gèle l'ensemble du backbone pré-entraîné durant la phase 1 afin de préserver les représentations génériques apprises sur COCO.
PHASE1_EPOCHS	10	Nombre d'epochs limité car seule la tête de détection est entraînée ; une convergence relativement rapide est attendue.
PHASE1_LR	10^{-2}	Taux d'apprentissage plus élevé car seule la tête (initialisée aléatoirement) est optimisée.
PHASE2_EPOCHS	40	Nombre d'epochs supérieur à la phase 1 car l'ensemble du réseau est entraîné ; un ajustement progressif des poids nécessite davantage d'itérations.
PHASE2_LR	10^{-3}	Taux d'apprentissage réduit pour affiner les poids pré-entraînés sans dégrader les connaissances acquises.

Hyperparamètre	Valeur	Justification
PATIENCE	10 / 15	Early stopping : 10 epochs en phase 1 (convergence rapide attendue), 15 en phase 2 afin de laisser le modèle stabiliser l'optimisation complète.
CONF_THRESHOLD	0.25	Seuil standard pour l'inférence : les prédictions dont la confiance est inférieure sont filtrées.

TABLE 2 – Hyperparamètres utilisés et justification des choix

6 Métriques d'évaluation

Les métriques retenues sont les métriques standard de la détection d'objets. Une détection est considérée correcte si la boîte prédite chevauche suffisamment la boîte réelle. Nous pouvons le mesurer par l'« Intersection over Union » :

$$\text{IoU} = \frac{\text{Aire}(\hat{B} \cap B)}{\text{Aire}(\hat{B} \cup B)} \quad (1)$$

où \hat{B} est la boîte prédite et B la boîte réelle.

- **mAP@50** : moyenne de la précision sur l'ensemble des classes avec un seuil IoU fixé à 0.5. Il s'agit de la métrique principale en détection d'objets. Elle permet de vérifier que les objets sont correctement localisés et classifiés.
- **mAP@50-95** : moyenne des mAP calculées pour plusieurs seuils IoU compris entre 0.5 et 0.95. Cette métrique est plus exigeante car elle demande une localisation plus précise des boîtes englobantes.
- **Précision** : proportion de détections correctes parmi toutes les prédictions du modèle. Une faible précision signifie qu'il y a beaucoup de fausses détections.
- **Recall** : proportion d'objets réellement présents qui ont été détectés par le modèle. Un recall faible indique que certains objets ne sont pas détectés.
- **AP@50 par classe** : score individuel pour chaque catégorie. Cette mesure permet d'identifier les classes pour lesquelles le modèle rencontre le plus de difficultés.

7 Résultats

Nous avons entraîné deux modèles : un YOLOv8n (baseline) et un YOLOv8s (avec hyperparamètres améliorés). Le tableau 6 présente une comparaison directe sur le jeu de test.

7.1 Modèle v1 — YOLOv8n (baseline)

Split	mAP@50	mAP@50-95	Précision	Recall
Validation	0.4935	0.3241	0.5331	0.4619
Test	0.4552	0.3112	0.5015	0.4313

TABLE 3 – Métriques v1 (YOLOv8n)

Le modèle obtient une mAP@50 de 0.46 sur le test, proche de celle obtenue en validation (0.49), ce qui indique une bonne capacité de généralisation sans surapprentissage. La précision (0.50) et le recall (0.43) restent modérés : le modèle détecte correctement environ la moitié des objets présents, mais en rate encore une part significative.

Classe	AP@50 (test)	Recall estimé
Carotte	0.3301	~33%
Aubergine	0.4823	~9%
Citron	0.5321	~58%
Pomme de terre	0.6951	~83%
Radis	0.2806	~18%
Tomate	0.4109	~59%
mAP@50	0.4552	

TABLE 4 – AP@50 et recall estimé par classe sur le test (v1)

Les performances sont très hétérogènes selon les classes. La pomme de terre (0.70) et le citron (0.53) sont bien détectés, grâce à leurs caractéristiques visuelles distinctives. À l'inverse, le radis (0.28) et la carotte (0.33) posent plus de difficultés.

Les figures 2 et 3 présentent les courbes d'entraînement et la matrice de confusion du modèle v1.

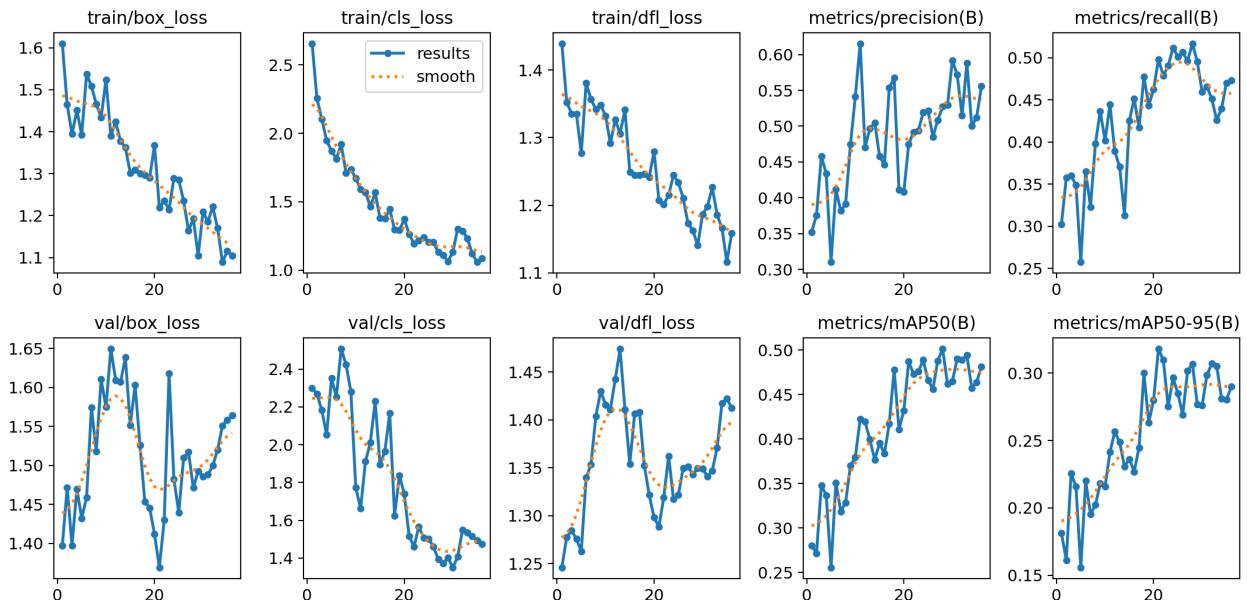


FIGURE 2 – Courbes d'entraînement — YOLOv8n (phase 2)

Observons les courbes box (position de la boîte), cls (classification) et dfl (précision des bordures) :

- **Losses sur les données d'entraînement (box, cls, dfl)** : les trois courbes diminuent régulièrement tout au long de la phase 2, ce qui indique que le modèle apprend correctement sur le dataset d'entraînement.
- **Losses sur les données de validation** : le comportement est plus instable. La val/box_loss et la val/dfl_loss augmentent légèrement en fin de phase 2, ce qui traduit un début de surapprentissage, tandis que la val/cls_loss décroît de façon stable, montrant que la classification continue de s'améliorer.
- **Précision et recall** : les deux métriques progressent globalement, mais présentent une forte variabilité d'une epoch à l'autre, typique d'un dataset de validation de petite taille (36 images). Le recall (0.50 en fin d'entraînement) reste inférieur à la précision, ce qui indique que le modèle ne détecte pas encore tous les objets présents.
- **mAP@50 et mAP@50-95** : ces métriques augmentent tout au long de la phase 2 sans atteindre de plateau net, ce qui suggère que le modèle n'avait pas totalement convergé à la fin des 40 epochs. Un

léger allongement du nombre d'epochs ou de la patience aurait probablement permis d'améliorer encore les performances.

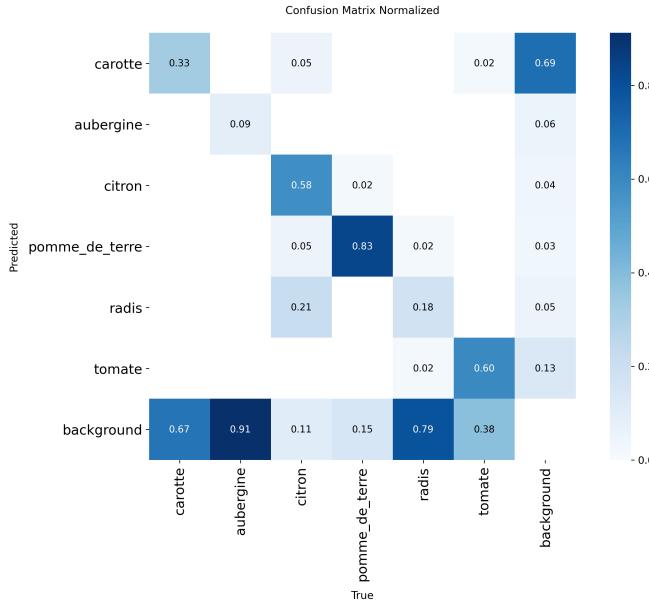


FIGURE 3 – Matrice de confusion normalisée — YOLOv8n (test)

La matrice de confusion normalisée (figure 3) illustre les deux types d'erreurs dominants du modèle. D'une part, la ligne *background* est très chargée : la grande majorité des instances de carotte (~67%), d'aubergine (~91%) et de radis (~79%) sont prédites comme arrière-plan, ce qui explique les recalls très faibles de ces classes (~33%, ~9% et ~18% respectivement). D'autre part, la colonne *background* révèle un taux significatif de faux positifs pour la carotte : le modèle détecte des carottes là où il n'y en a pas, probablement en confondant des formes allongées en arrière-plan. La pomme de terre et la tomate affichent en revanche une diagonale nette, confirmant leur bonne détection.

Au-delà des erreurs vers l'arrière-plan, la matrice révèle également des confusions inter-classes. La plus notable concerne le citron : 21 % de ses instances sont prédites comme des radis, ces deux classes partageant une silhouette ronde à taille comparable. Par ailleurs, 5 % des instances de citron sont prédites comme des carottes, vraisemblablement en raison d'une teinte jaune-orangée commune. Dans le même sens, la pomme de terre et le citron s'échangent ponctuellement (2 à 5 %), leur couleur claire et leur silhouette sphérique induisant le modèle en erreur. En revanche, la carotte et l'aubergine ne génèrent quasiment aucune confusion inter-classe : leurs instances sont directement manquées (prédites comme fond) plutôt que mal classifiées, ce qui suggère que ces classes sont tout simplement trop peu représentées pour que le modèle ait appris leurs caractéristiques visuelles distinctives.

7.2 Modèle v2 — YOLOv8s (améliorations)

Pour la version 2, nous avons utilisé YOLOv8s (11 millions de paramètres) avec des hyperparamètres optimisés : la phase 2 a été étendue à 80 epochs avec une patience de 20, des augmentations supplémentaires ont été appliquées (rotation ± 15 , flip vertical, mixup) et un label smoothing de 0.1 a été introduit pour limiter la sur-confiance du modèle sur des annotations éventuellement bruitées.

Split	mAP@50	mAP@50-95	Précision	Recall
Validation	0.5650	0.3652	0.6142	0.5265
Test	0.3546	0.1982	0.3307	0.4579

TABLE 5 – Métriques v2 (YOLOv8s)

Le modèle YOLOv8s progresse nettement en validation (+7 pts mAP@50 par rapport au YOLOv8n), mais chute fortement sur le test (0.35 contre 0.46). Cet écart important entre validation et test est le signe d'un surapprentissage : le modèle a appris à performer sur les 36 images de validation, sans généraliser au test.

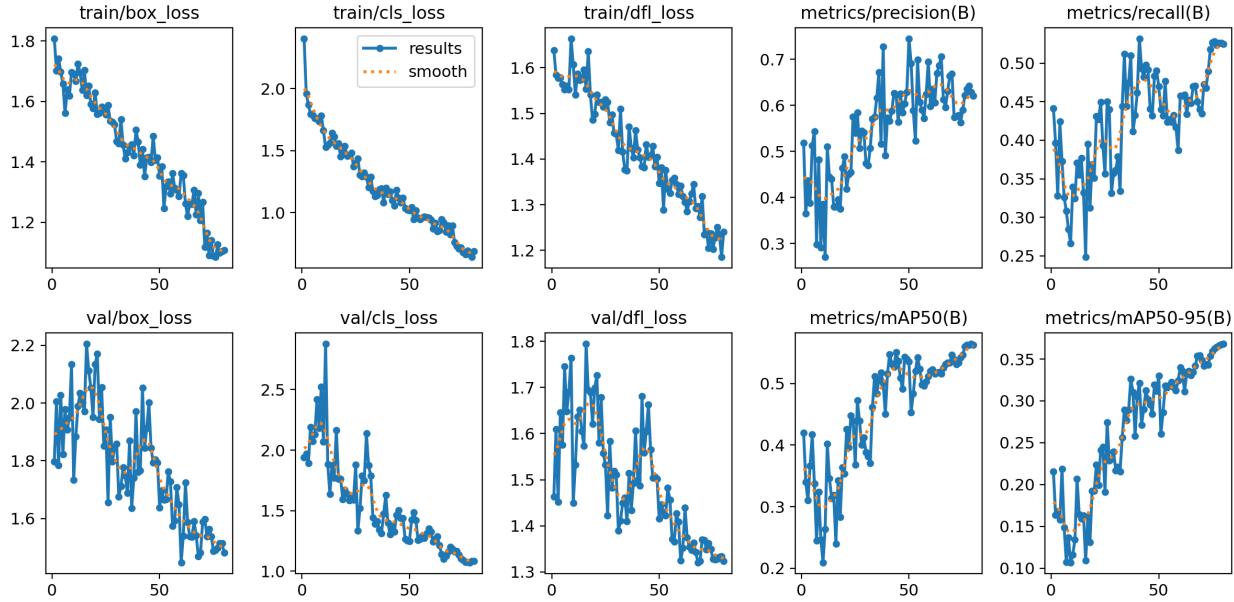


FIGURE 4 – Courbes d'entraînement — YOLOv8s (phase 2)

- **Losses sur les données d'entraînement** : les trois courbes diminuent sur l'ensemble des 80 epochs, ce qui montre que le modèle continue d'apprendre efficacement sur le dataset d'entraînement, de façon plus prolongée que dans la version 1.
- **Losses sur les données de validation** : le comportement diffère nettement. La val/box_loss et la val/dfl_loss diminuent jusqu'aux alentours de l'epoch 40, puis augmentent progressivement — signe de surapprentissage. Le modèle améliore encore ses performances sur le train, mais commence à se dégrader sur la validation.
- **Précision et recall** : la précision oscille, traduisant une instabilité des prédictions sur un jeu de validation très restreint. Le recall progresse de manière plus régulière, mais reste affecté par cette variabilité.
- **mAP@50 et mAP@50-95** : les deux métriques augmentent jusqu'à environ l'epoch 50, puis atteignent un plateau et montrent parfois une légère régression. Le fait que ce plateau soit atteint bien avant la fin des 80 epochs confirme que le modèle commence à surapprendre au-delà d'un certain point.

En résumé, les courbes de la version 2 illustrent clairement un phénomène de surapprentissage : les performances en validation s'améliorent initialement, puis se dégradent malgré la poursuite de l'optimisation sur les données d'entraînement.

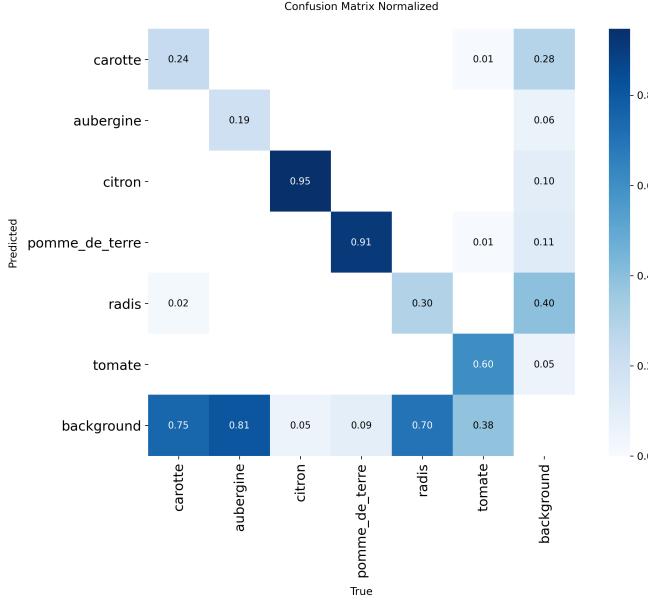


FIGURE 5 – Matrice de confusion normalisée — YOLOv8s (test)

La matrice de confusion normalisée du modèle v2 (figure 5) révèle un profil très contrasté. Deux classes se distinguent nettement : le citron ($\sim 95\%$ de recall) et la pomme de terre ($\sim 91\%$), dont la quasi-totalité des instances sont correctement localisées et classifiées. À l’opposé, la carotte ($\sim 24\%$) et l’aubergine ($\sim 19\%$) restent très mal détectées — leur ligne *background* est chargée à respectivement 75 % et 81 %, confirmant que le modèle les rate massivement. Le radis progresse légèrement par rapport au v1 ($\sim 30\%$ contre $\sim 18\%$), mais 70 % de ses instances sont encore prédictes comme arrière-plan.

Au niveau des faux positifs, le radis se démarque : 40 % de ses prédictions sont des faux positifs (colonne *background*), soit le taux le plus élevé parmi toutes les classes. La carotte présente également un taux notable de 28 %, le modèle confondant des formes allongées en arrière-plan avec cette classe. En revanche, les confusions inter-classes sont quasi inexistantes : seules de légères confusions apparaissent entre la tomate et la carotte (1 %), et entre la tomate et la pomme de terre (1 %). Le modèle v2 a donc simplifié ses erreurs — presque tout se résume à des detections manquées vers l’arrière-plan — mais au prix d’un recall très inégal selon les classes, signe typique d’un surapprentissage sur les exemples de validation les plus représentatifs.

7.3 Comparaison et sélection du modèle

Modèle	mAP@50 val	mAP@50 test	Précision	Recall
v1 — YOLOv8n	0.4935	0.4552	0.5015	0.4313
v2 — YOLOv8s	0.5650	0.3546	0.3307	0.4579

TABLE 6 – Comparaison v1 vs v2 sur validation et test

Bien que le v2 obtienne de meilleures métriques de validation, il se dégrade significativement sur le test. Cet écart révèle un surapprentissage : avec seulement 166 images d’ entraînement, YOLOv8s (11 millions de paramètres) est trop grand et mémorise les patterns de validation. YOLOv8n reste donc mieux adapté à la taille de notre dataset.

7.4 Analyse des résultats (YOLOv8n retenu)

Cas de succès

La pomme de terre (AP@50 = 0.70, recall ~83%) et le citron (AP@50 = 0.53, recall ~58%) sont les classes les mieux détectées. Ces éléments présentent des caractéristiques visuelles distinctives : forme ronde et couleur jaune vif pour le citron, texture particulière pour la pomme de terre, qui facilitent l'apprentissage.

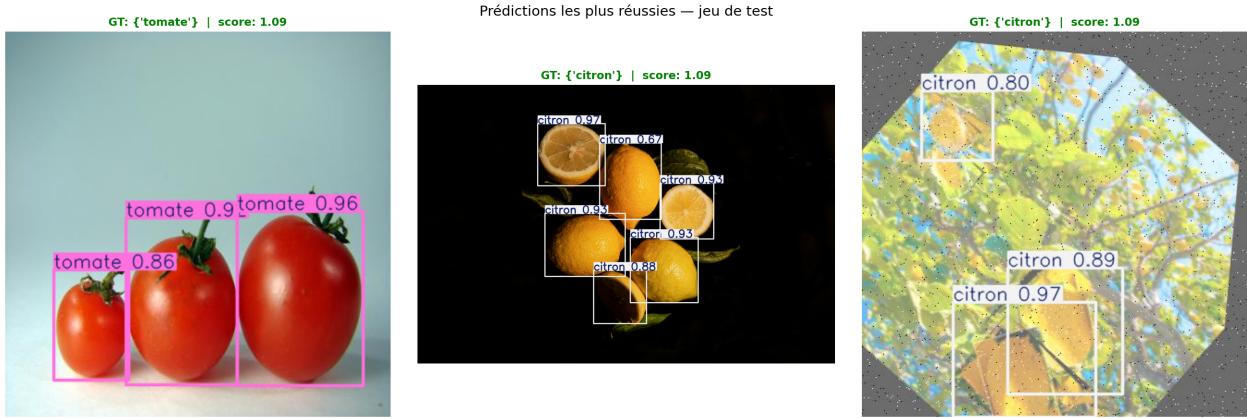


FIGURE 6 – Les trois prédictions les mieux réussies sur le jeu de test (YOLOv8n, conf ≥ 0.25)

La figure 6 illustre trois images où le modèle parvient à localiser et classifier correctement l'ensemble des instances présentes (score = 1.09). Les deux premières — tomates sur fond blanc neutre, citrons sur fond noir — bénéficient d'un objet bien isolé et d'un contraste élevé, conditions idéales pour la détection. La troisième montre que le modèle reste robuste face à une augmentation agressive (rotation, teinte modifiée), avec des scores de confiance élevés (>0.80) malgré la dégradation visuelle.

Erreurs courantes

Le radis (AP@50 = 0.28) et la carotte (AP@50 = 0.33) sont les classes les plus problématiques. L'analyse de la matrice de confusion (figure 3) révèle deux types d'erreurs principaux :

- **Faux négatifs** : le modèle rate beaucoup d'objets, notamment les radis (petits objets peu distinctifs, ~18% de recall) et les aubergines (~9% de recall, visuellement complexes sur fond sombre) — la quasi-totalité des instances de ces deux classes sont manquées.
- **Faux positifs sur la carotte** : le modèle génère un nombre significatif de détections fantômes, confondant des formes allongées en arrière-plan avec des carottes.

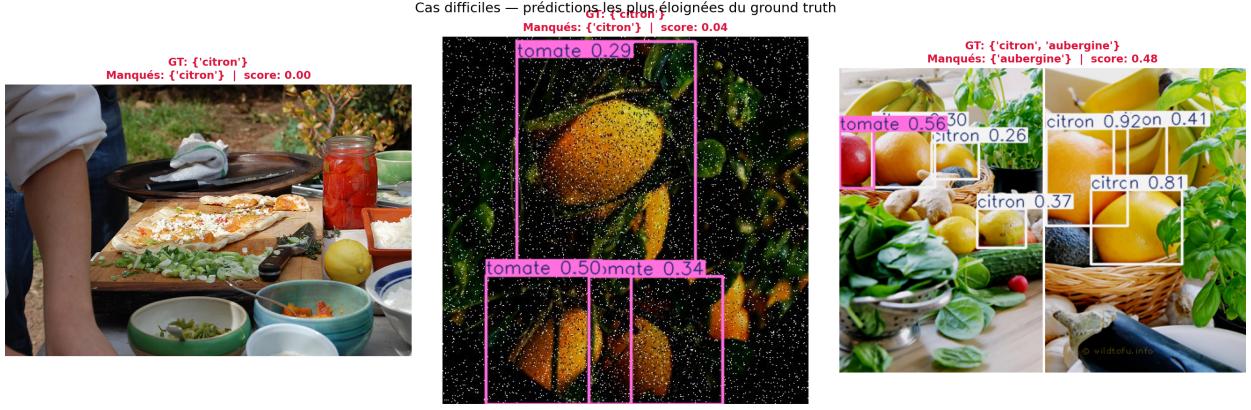


FIGURE 7 – Les trois prédictions les plus difficiles sur le jeu de test — classes manquées ou score de confiance très faible (YOLOv8n)

La figure 7 présente trois profils d'échec distincts. La première image illustre un faux négatif total (score = 0.00) : un citron partiellement visible dans une scène de cuisine très chargée n'est pas détecté, le modèle étant perturbé par la richesse de l'arrière-plan. La deuxième révèle une confusion de classe (score = 0.04) : sur une image fortement bruitée, les objets sont prédits comme des tomates (conf. 0.29–0.50) plutôt que des citrons — deux classes à silhouette ronde similaire dont la distinction devient difficile après augmentation agressive. La troisième montre une détection partielle (score = 0.48) : dans une scène multi-classes, les citrons sont correctement identifiés mais l'aubergine est totalement manquée, ce qui illustre la difficulté persistante du modèle sur cette classe sous-représentée (~9% de recall).

Ces résultats s'expliquent principalement par la taille limitée du dataset (~28 images par classe pour l'entraînement) et l'hétérogénéité des sources (LVIS, Kaggle, Roboflow, collecte manuelle).

8 Conclusion

Ce projet a permis de mettre en place un pipeline complet de détection de légumes, allant de la constitution du dataset à l'évaluation de deux variantes de YOLOv8. L'approche de transfert learning en deux phases sur YOLOv8n a permis d'atteindre un mAP@50 de 0.46 sur le jeu de test, malgré un dataset relativement petit de seulement 166 images.

Nos analyses montrent que la taille du modèle doit être adaptée à celle du dataset : un modèle trop volumineux sur un petit jeu de données risque le surapprentissage.

Pour améliorer significativement les performances, il serait nécessaire d'augmenter le dataset à au moins 600 images, en ciblant particulièrement les classes les moins représentées comme le radis, l'aubergine et la carotte. Il serait également important d'homogénéiser les annotations, car la qualité variable des sources hétérogènes pourrait pénaliser l'apprentissage.

Enfin, le modèle YOLOv8s devrait être testé uniquement une fois que le dataset est suffisamment large, idéalement au-delà de 400 images, afin d'éviter le surapprentissage.

Sources du dataset

- **Fruit and Vegetable** — Roboflow Universe
<https://universe.roboflow.com/cse299/fruit-and-vegetable/>
- **LVIS Fruits and Vegetables Dataset** — Kaggle
<https://www.kaggle.com/datasets/henningheyen/lvis-fruits-and-vegetables-dataset>