

# Détection d'objets - YoloV8

## Plan :

1. Construction d'un dataset de 3 classes : personne, casque et gilet.
2. Entraînement d'un modèle YOLOv8 sur ce dataset.
3. Déploiement du modèle dans une application Flask.

## 1 / Construction du Dataset

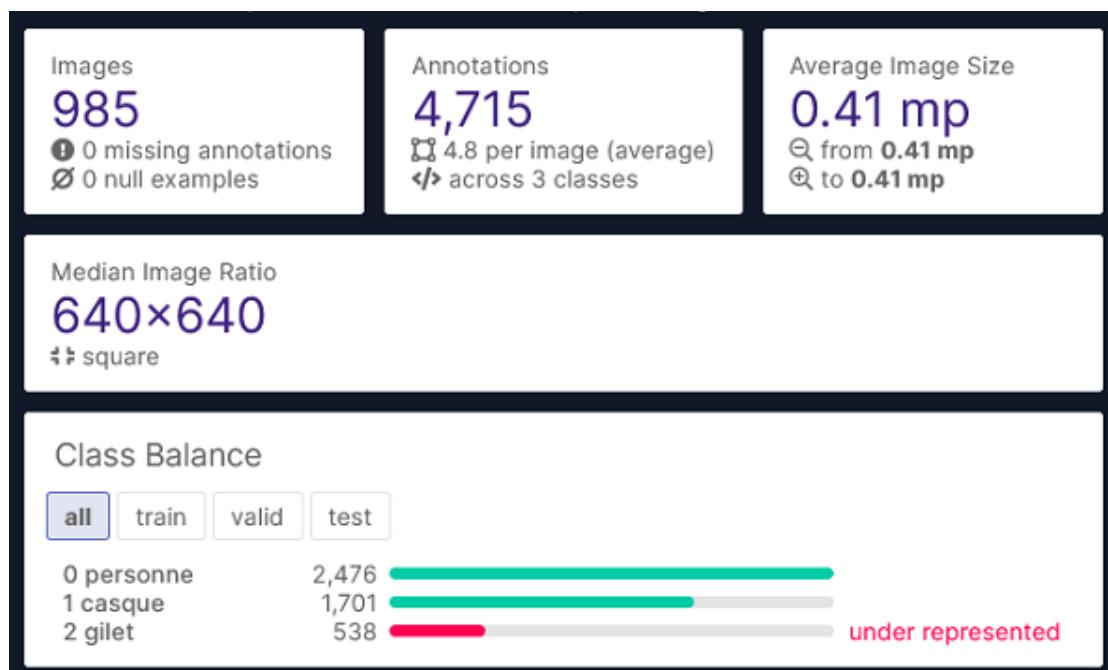
Nous voulons un modèle capable de détecter, sur un flux vidéo, si une personne porte bien un gilet de sécurité ET un casque.

Nous avons décidé de créer un dataset contenant **3 classes** :

- 0 : personne
- 1 : casque
- 2 : gilet

### from scratch

- web scrapping ( 01\_scraping.py / 02\_data\_harvest.py )
- resizing (03\_resize.py - YOLO prend des images en 640x640 )
- labellisation au format YOLOv8 ( pip install label-studio / makesense.ai )
- première version du Dataset déposée sur [Roboflow](#):



- ajout d'images labellisées contenant des gilets pour aider dans la répartition des classes.
- Upload, pre processing, data augmentation et train / eval /test split sur Roboflow
- version finale :
- [https://universe.roboflow.com/yolosafetygear/safety\\_gear\\_simplon/dataset/3](https://universe.roboflow.com/yolosafetygear/safety_gear_simplon/dataset/3)

## 2/ Entraînement du modèle

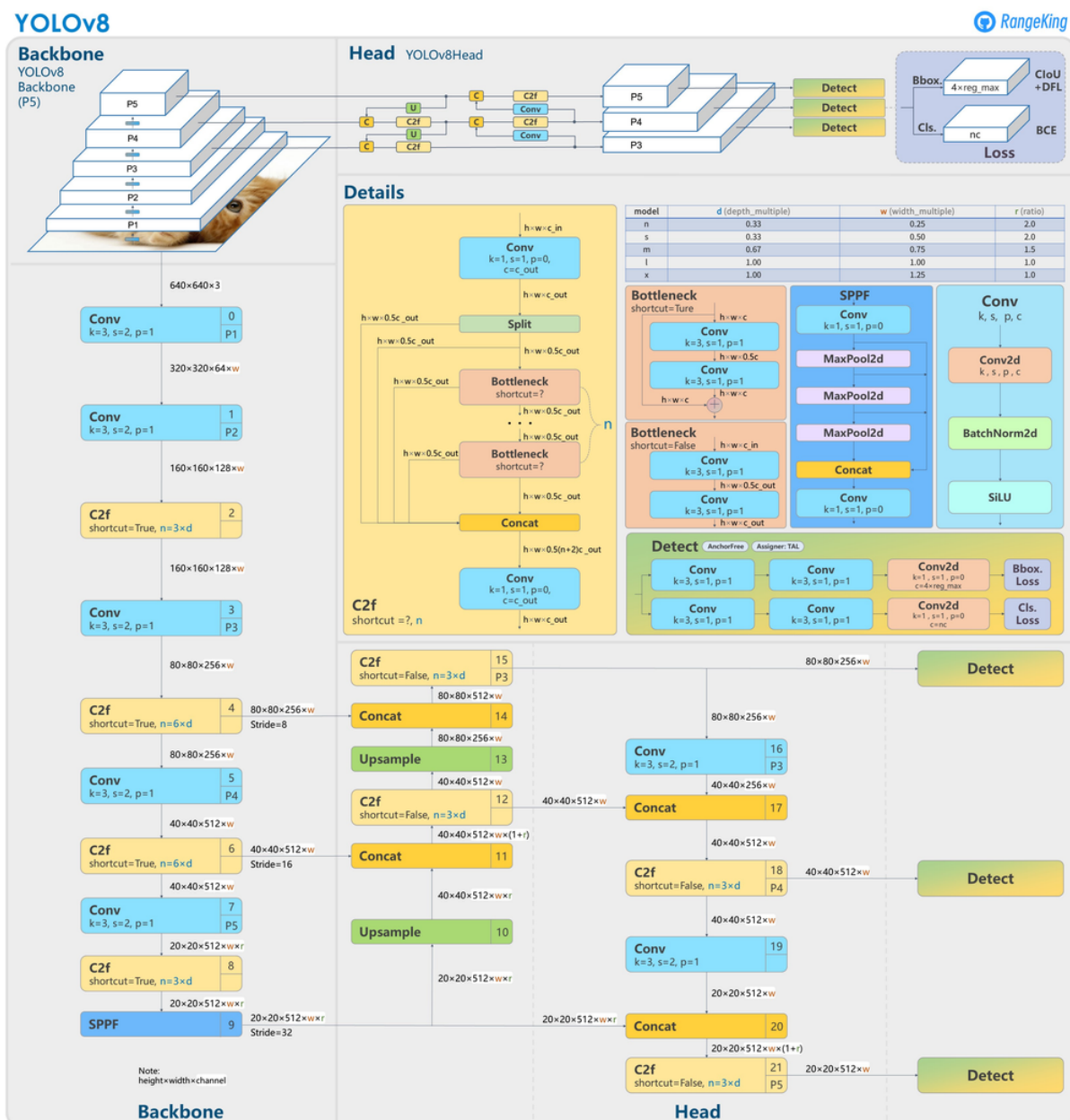
### choix du modèle : YOLOv8 d'Ultralytics

YOLOv8 est disponible en tant que modèle gratuit et open source avec Python.

La bibliothèque fournit des modèles pré-entraînés pour une variété de tâches de détection et de segmentation d'objets. Ces modèles peuvent être utilisés directement ou ils peuvent être ajustés sur un ensemble de données personnalisé.

La version 8 est sorti en 2023

## Architecture :



**YOLOv8** est un réseau neuronal convolutionnel (CNN). Sa structure est basée sur l'utilisation de filtres convolutifs pour extraire des caractéristiques d'images, divisant l'image en une grille pour détecter des objets, prédisant plusieurs boîtes englobantes par cellule de grille et utilisant la suppression non maximale pour améliorer la précision de la détection. Les **avantages** des CNN comprennent une connectivité partielle qui réduit le nombre de poids nécessaires, et le partage de poids permettant à un seul groupe de poids d'apprendre l'ensemble de l'entrée, réduisant ainsi le temps d'entraînement. Cependant, les CNN ont également des **inconvénients**, tels que le besoin d'un grand nombre de données d'entraînement pour apprendre à détecter efficacement les objets, et la difficulté d'interprétabilité due aux représentations abstraites apprises par les CNN.

## Les différents modèles :

Performance						
	Detection (COCO)	Detection (Open Images V7)	Segmentation (COCO)	Classification (ImageNet)	Pose (COCO)	
Model	size (pixels)	mAP <sup>val</sup> <sub>50-95</sub>	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

Nous avons choisit d'entraîner le modèle YOLOv8m puisqu'il semble offrir le meilleur rapport précision / vitesse d'exécution.

---

Nous avons ensuite entraîné un modèle YOLOv8n pour le déployer dans une application web légère qui tourne en local sur un petit CPU.

## Entraînement :

- fait sur Google colab pour bénéficier de la puissante de calcul nécessaire

<https://colab.research.google.com/drive/1haVoMxOHWRqEHvxCUKGXXVGnhcuMjPEo>

→ 05\_train\_yolov8.ipynb

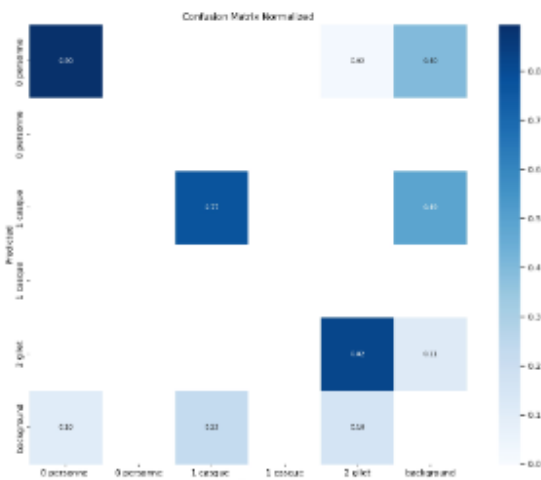
( mAP50 = 0.826 )

La **mAP50** est une métrique d'évaluation largement utilisée dans les tâches de détection d'objets et signifie "**Mean Average Precision**" à un seuil IoU (Intersection over Union) de 50 %.

La mAP50 mesure la précision du modèle pour détecter et localiser des objets dans une image.

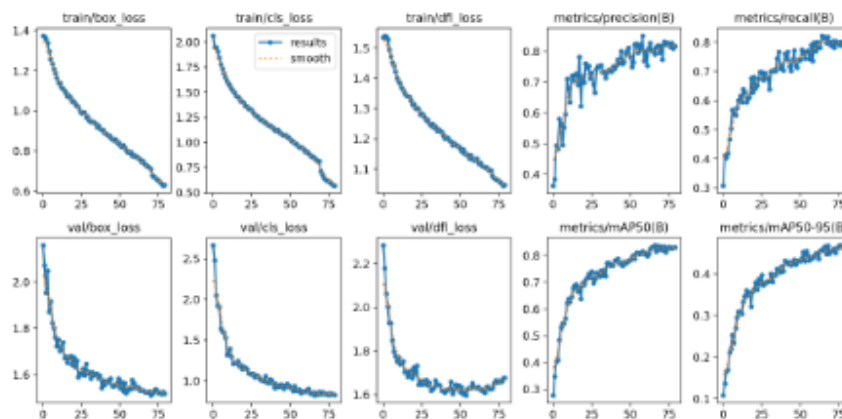
- La mAP50 varie de 0 à 1, où 1 indique une performance parfaite.
- Une mAP50 de 0,826 signifie qu'en moyenne, le modèle obtient une précision de 82,6% dans la détection et la localisation des objets dans l'ensemble de test en utilisant un seuil IoU de 50 %.

## Matrice de confusion normalisée :



Malgrès un manque de gilet dans notre dataset, le modèle s'en sort plutôt bien.

Par contre, il a tendance a confondre les casque avec le fond de l'image.



best\_model/  
weights/  
results.png

YOLOv8 produit automatiquement des graphiques qui permettent de suivre l'évolution de l'apprentissage et d'évaluer les performance du modèle entraîné.

Ces infos se trouvent dans le dossier 'best\_model'

## Déployer le modèle dans un web app Flask :

→ app.py

Screenshot de l'app :

# Safety Gear Detection

with Ultralytics YOLOv8.

## Object Detection Results

### Real-time prediction of safety vest detection

Start Webcam Feed

Stop Webcam Feed



0 personne - 0.78

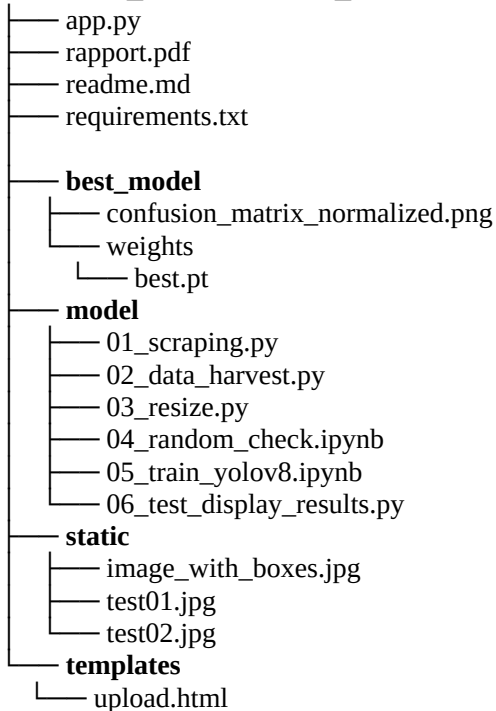
1 casque - 0.75

2 gilet - 0.72



Powered by [Hatchi-Kin](#)

## OBJECT\_DETECTION\_YOLOV8




## Conclusion :

La documentation de la bibliothèque python YOLO est vraiment très bien faite, le modèle très simple à utiliser avec ses valeurs par défaut. Les performances du modèle fine-tuned sur notre propre dataset sont plutôt bonnes ( mAP50 = 0.826, pour la version m et 0,799 pour la version s )

Et un utilisateur peut tester les capacités du modèle via sa webcam grâce à l'application flask.

## Reste à implémenter / idées d'amélioration :

- ~~Trouver le moyen d'afficher le flux vidéo directement dans le navigateur~~ 
- Améliorer le Dataset. Dans l'idéal, il nous faudrait plus d'images de gilets. Il nous faudrait aussi des angles différents pour aider le modèle même dans des configurations plus rares (contre plongée etc.). Il faudrait aussi revoir les labels ('0 personne' et '0 personne')
- Script python pour afficher les boites entourant les personnes en rouge si ces boites ne contiennent pas un casque ET un gilet, et toutes les autres boites en bleu.
- Une autre méthode pour savoir si l'équipement est complet sans forcément avoir à regarder l'image.
- Sauvegarder dans un fichier séparé (.txt / .json?) les dates/heures de detections non conformes