

# **Rapport d'étude**

**« Entraînement d'un réseau de neurone YOLO8  
pour la détection d'objets 3D  
posés sur une piste»**

**Exploitation sur carte Raspberry Pi 4**

**Jean-Luc CHARLES**  
**Consultant IA/Data processing**

version 1.0 du 3 mai 2025

## HISTORIQUE DES MODIFICATIONS

Édition	Révision	Date	Modification	Visa
1	0	2025-05-03	Version initiale	

## Table des matières

1 Contexte de l'étude.....	5
1.1 Objectifs de l'étude.....	5
2 Préparation du jeu de données.....	6
2.1 Création des images.....	6
2.2 Annotation des images sur le site Roboflow.....	6
3 Entraînement du réseau de neurones YOLOv8n.....	9
3.1 Environnement Virtuel Python (EVP) sur le PC de calcul.....	9
3.2 Choix des hyper-paramètres d'entraînement.....	10
3.3 Environnement de calcul.....	11
3.4 Fichiers résultats des entraînements du réseau yolov8n.....	12
3.5 Résultats des entraînements.....	13
Early stopping des entraînements.....	14
Performances des réseaux entraînés sur le PC de calcul.....	15
Matrices de confusion.....	16
3.6 Conclusion.....	17
4 Exploitation du réseau YOLO8 sur RPi4.....	18
4.1 Préparation de la carte SD pour la RPi4.....	18
Identifiant SSID unique du WiFi.....	18
4.2 Évaluation des entraînements du réseau yolov8n.....	18
Performances sur RPi4 des réseaux entraînés sur le PC de calcul.....	18
Matrices de confusion.....	19
4.3 Exploitation du réseau YOLO8 depuis le bureau à distance.....	20
Le bureau à distance avec le client RealVNC.....	20
Dossier de travail positionné sur UCIA/UCIA_ObjectDetection-II.....	20
detect_camera-1.py : détection des objets, rendu graphique N&B.....	20
detect_camera-2.py : détection des objets, rendu graphique couleur.....	23
Choix des entraînements du réseau YOLO à exploiter.....	25
4.4 Exploitation du réseau YOLO8 depuis un navigateur distant.....	26
Arrêt ( <i>shutdown</i> ) de la carte RPi4 depuis le navigateur WEB.....	27
5 Conclusions.....	28
6 Glossaire.....	29
7 Annexes.....	30
7.1 Première utilisation des poids du réseau au format NCNN.....	30
7.2 Prise d'images avec la caméra de RPi4.....	31
take_image.py.....	31
7.3 Programmes pour l'entraînement et l'évaluation du réseau yolov8n.....	32
train_YOLOv8.py.....	32
eval_YOLOv8.py.....	34
process_results.py.....	36
7.4 Fichiers résultats.....	39
results_yolov8n_v1.8.txt (obtenu sur le PC de calcul).....	39
results_yolov8n_v1.8.txt (obtenu sur la carte RPi4).....	40
8 Références.....	41

## Index des figures

Figure 1: Les nouveaux objets de l'étude II et la piste.....	5
Figure 2: Les images de la nouvelle étude.....	6
Figure 3: L'interface web du site Roboflow pour annoter les images.....	7
Figure 4: Le jeu de données créé sur Roboflow.....	8
Figure 5: Le réseau de neurones YOLO sur le site web Ultralytics .....	9
Figure 6: Arborescence du projet.....	11
Figure 7: Exemple d'image de validation.....	12
Figure 8: Exemples de statistiques d'entraînement.....	13
Figure 9: Matrice de confusion (batch=40, epochs=300, dataset v1.8).....	16
Figure 10: Matrice de confusion (batch=30, epochs=300, dataset v1.8).....	17
Figure 11: Matrice de confusion sur la RPi4 ( batch=30, epochs=300, dataset v1.8).....	19
Figure 12: RealVnc Viewer : le bureau RPi4 à distance.....	20
Figure 13: detect_camera-1.py : terminal de lancement du programme.....	21
Figure 14: detect_camera-1.py : fenêtre graphique d'affichage des objets détectés.....	22
Figure 15: detect_camera-2.py : Terminal de lancement du programme.....	23
Figure 16: detect_camera-2.py : fenêtre graphique d'affichage des objets détectés.....	24
Figure 17: Les options de lancement des programmes detect_camera_*.py.....	25
Figure 18: Connexion d'un navigateur sur l'URL <a href="http://10.99.99.1:5000/video">http://10.99.99.1:5000/video</a> .....	26
Figure 19: Message du module ultralytics pour la première utilisation du format ncnn.....	30

## Index des tableaux

Tableau 1: Tableau des hyper-paramètres d'entraînement.....	10
Tableau 2: Plages de valeurs des hyper-paramètres utilisés.....	10
Tableau 3: Tableau des paramètres d'entraînement.....	10
Tableau 4: Taille des fichiers binaires des poids du réseau YOLOv8n.....	12
Tableau 5: EarlyStopping des combinaisons d'entraînement.....	14

# 1 Contexte de l'étude

Depuis janvier 2023 l'association « la ligue de l'enseignement » coordonne le projet UCIA (Usages et Consciences des Intelligences Artificielles). Dans le cadre de ce projet, un kit pédagogique doit être créé incluant notamment un robot IA *Open Source* et *Open Hardware* dont l'utilisation doit permettre d'encourager un regard critique sur l'Intelligence Artificielle.

Le document [UCIA\\_Cahier des charges 11-2024.pdf](#) précise le fonctionnement attendu du robot support des fonctionnalités IA. Trois niveaux sont décrits dans le cahier des charges UCIA : dans le cadre de cette étude Il s'agit de consolider le développement des composants logiciels du **jalón Niveau 0 pour le mode chasseur trésor** (page 8 à 11 du CDC UCIA).

## 1.1 Objectifs de l'étude

Cette étude complète les deux études précédentes présentées dans le document [Doc/UCIA-IA-DetectionObj\\_V2.1.pdf](#) sur le dépôt GitHub [UCIA\\_OjectDetection-I](#).

Les objectifs de l'étude sont :

1. Refaire les nouveaux *datasets* entraînement / validation / test pour tenir compte :
  - des nouveaux objets à reconnaître : en plus des trois objets balle, cube et étoile les nouveaux objets 3D sont : triangle, cylindre, hexagone et maison, soit 7 objets différents au total.



Figure 1: Les nouveaux objets de l'étude II et la piste.

Certains objets peuvent facilement être confondus (triangle vu de face et cube vu de face), ce qui peut donner des situations intéressantes pour mettre en évidence des difficultés de classification des objets. Les nouvelles couleurs sont : violet, bleu, rose, cyan, vert.

- de la présence de la piste suivie par le robot Thymio :
  - pour la détection d'objets qualifiant la piste : « virage à gauche », « virage à droite », « piste droite »,
  - pour la détection des 7 objets 3D en situation « hors piste » et « sur la piste ».
  - pour la détection d'objets imprimés sur le papier de la piste : « nid », « cocarde », « passage piéton », « stop » et « parking ».

## 2 Préparation du jeu de données

### 2.1 Création des images

Le programme Python `take_image.py` développé pour l'étude précédente permet de numéroté automatiquement au format `objets3D-nnn.jpg` les images prises par la caméra « grand angle » de la RPi4. On regroupe plusieurs objets dans chaque image pour obtenir un nombre d'objets suffisant. La présente étude utilise 363 images montrant les objets en situation sur la piste :



Figure 2: Les images de la nouvelle étude.

Les images sont ensuite téléchargées sur le site Roboflow pour réaliser l'annotation manuelle.

### 2.2 Annotation des images sur le site Roboflow

Le site Roboflow propose une interface web efficace pour réaliser la tâche d'annotation des images (voir figure 3). Les images chargées sur le site sont annotées à la main une par une. Pour chacun des objets contenu dans chaque image, il faut :

1. Délimiter précisément avec la souris la boîte englobante de l'objet (*bounding box*).
2. Labelliser l'objet délimité en utilisant une des 15 classes :

7 objets 3D : `ball`, `cylinder`, `cube`, `home`, `hexagon`, `star`, `triangle`

8 objets imprimés sur la piste :

`laneLeft` virage à gauche,  
`laneRight` virage à droite,  
`straightLine` piste droite,  
`nest` le nid,

`Stop` panneau STOP,  
`parking` panneau Parking,  
`zebracross` passage piéton,  
`cockate` cocarde.

Le travail d'annotation est une étape importante qui doit être réalisé avec soin pour la qualité de l'entraînement du réseau de neurones. Une fois tous les objets d'une image entourés et labellisés, on peut passer à l'image suivante.

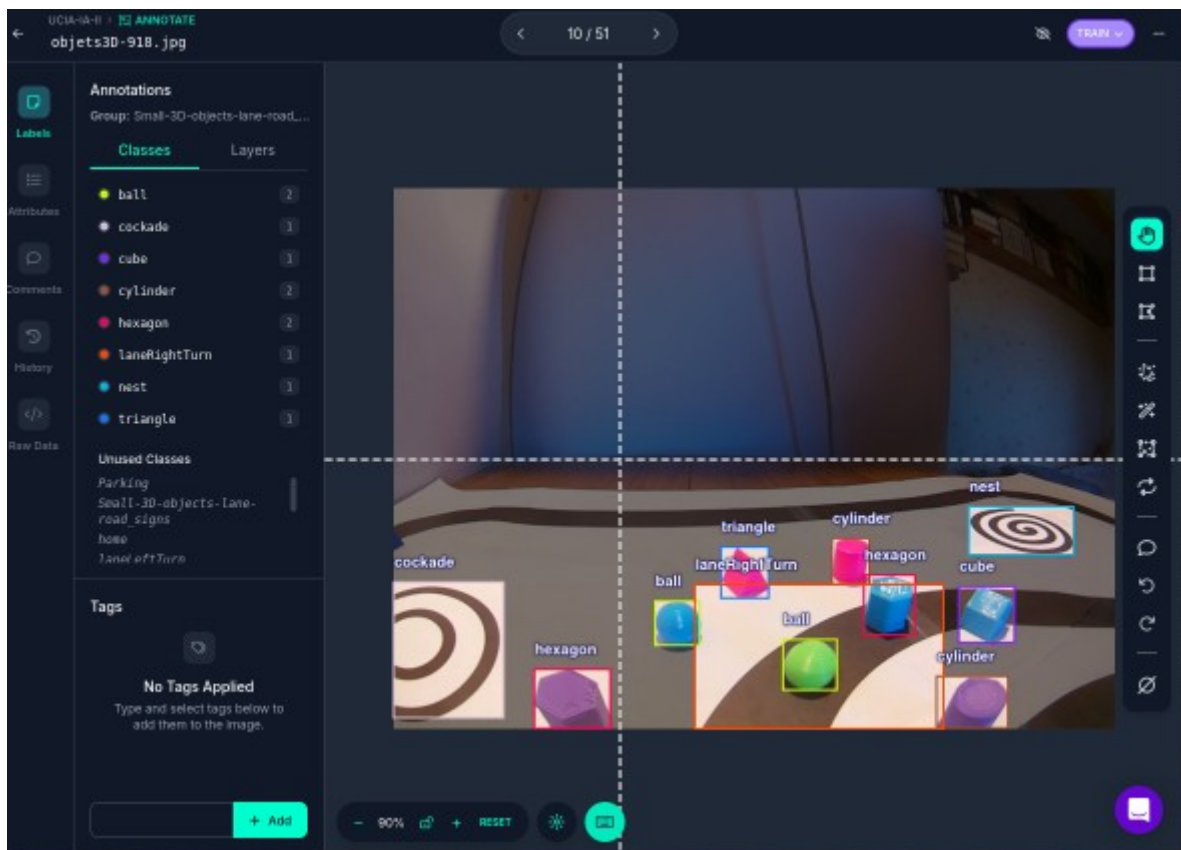


Figure 3: L'interface web du site Roboflow pour annoter les images.

Quand toutes images sont annotées, on peut créer sur le site Roboflow des jeux de données (*datasets*), en répartissant les 363 images annotées en plusieurs jeux :

- **Train dataset** : le jeu d'images pour l'entraînement du réseau de neurones.
- **Validation dataset** : le jeu d'images utilisé pour évaluer les performances du réseau de neurones à différentes étapes de l'entraînement. Ces images ne sont jamais apprises par le réseau de neurones.
- **Test dataset** : un jeu d'images qu'on peut former pour mesurer les performances du réseau entraîné, indépendamment des jeux d'entraînement et de validation.

Avec les 363 images créées, on choisit de mettre :

- 296 images pour l'entraînement,
- 67 images de validation,
- 23 images de test.

Les images du jeu de données sont converties en images **640 x 640 pixels en ton de gris** pour l'entraînement du modèle YOLO.

Plusieurs version des jeux d'images ont été construites pour améliorer progressivement les performances du modèle YOLO entraîné avec ces images. La figure 4 montre la dernière version du jeu de données créés sur le site Roboflow.

UCIA

UCIA-IA-II  
Object Detection

DATA

Upload Data

Annotate

Dataset 386

Versions Train

Analytics

Classes & Tags

MODELS

Models

Visualize

DEPLOY

Deployments

Active Learning

### Versions

+ Create New Version

Versions

296-67-23\_greyscale-v1.0

v14 386 640x640

Stretch to

280-72-15\_greyscale-v1.0

v13 367 640x640

Stretch to

248-84-0\_greyscale-v1.0

v12 332 640x640

Stretch to

247-118-0\_greyscale-v1.0

v10 365 640x640

Stretch to

2025-04-26 9:54am

386 Total Images

View All Images →

Dataset Split

TRAIN SET 77% 296 Images

VALID SET 17% 67 Images

TEST SET 6% 23 Images

Preprocessing

Auto-Orient: Applied

Resize: Stretch to 640x640

Grayscale: Applied

Augmentations

No augmentations were applied.

Figure 4: Le jeu de données créé sur Roboflow.

Une fois annoté, le jeu de données est téléchargé sur le PC de calcul au format **yolov8** pour réaliser l'entraînement du modèle YOLO choisi.



### 3 Entraînement du réseau de neurones YOLOv8n

Le réseau de neurones **YOLO** (*You Only Look Once*) est un modèle populaire de détection d'objets et de segmentation d'images. Lancé en 2015, **YOLO** a rapidement gagné en popularité grâce à sa rapidité et à sa précision. La version **YOLO8** est couramment adoptée comme version optimale de **YOLO**. La version actuelle est **YOLO12**.

Les versions de **YOLO** sont gérées sur le site Ultralytics : <https://docs.ultralytics.com/fr>

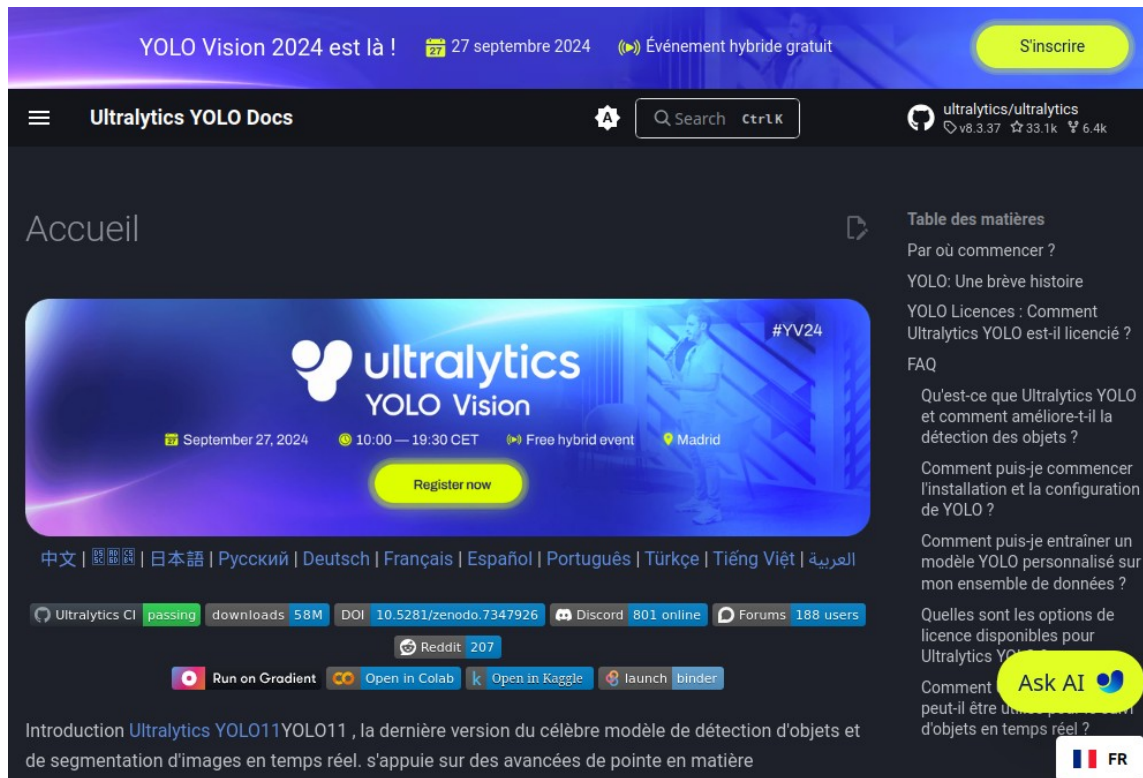


Figure 5: Le réseau de neurones YOLO sur le site web Ultralytics .

Compte tenu des études précédentes, c'est la version **yolov8n** qui est choisie : on obtient des temps d'inférence voisins de 0.5 seconde sur RPi4, avec des performances de détection / classification des objets tout à fait satisfaisantes.

#### 3.1 Environnement Virtuel Python (EVP) sur le PC de calcul

Conformément à l'état de l'art, on utilise un **Environnement Virtuel Python (EVP)**, au sein duquel les modules sont chargés et les programmes d'entraînement sont développés et exploités. Pour l'étude UCIA-II l'EVP est créé avec le gestionnaire **uv**<sup>1</sup> sur le PC de calcul :

```
uv init UCIA_ObjectDetection-II
```

Les modules Python nécessaires à l'entraînement des réseaux sont ajoutés dans le dossier **.venv** créé par **uv** dans le dossier **UCIA\_ObjectDetection-II** :

```
cd UCIA_ObjectDetection-II
```

```
uv add ultralytics
```

L'installation du module **ultralytics** charge tous les autres modules Python nécessaires aux calcul de *machine learning* (**tensorflow**, **torch**, **numpy**, **scipy**...).

<sup>1</sup> <https://docs.astral.sh/uv/>

## 3.2 Choix des hyper-paramètres d'entraînement

De nombreux **hyper-paramètres** influent sur l'entraînement supervisé d'un réseau de neurones : temps de calcul, qualité du réseau entraîné obtenu....

La page [docs.ultralytics.com/fr/modes/train/#train-settings](https://docs.ultralytics.com/fr/modes/train/#train-settings) liste ces hyper-paramètres. Pour cette étude les hyper-paramètres pertinents sont présentés sur le tableau 1.

Tableau 1: Tableau des hyper-paramètres d'entraînement.

Paramètre	Description
epochs	Nombre de répétitions du processus complet d'entraînement pour converger vers le meilleur état de réseau entraîné
batch	Nombre d'images fournies dans un lot d'images d'entraînement
patience	Nombre d'époques à attendre sans amélioration des mesures de validation avant d'arrêter l'entraînement. Permet d'éviter le sur-entraînement en arrêtant le processus lorsque les performances atteignent un plateau.

Le tableau 2 donne les plages de valeur retenues pour les hyper-paramètres du dernier entraînement du réseau **yolov8n**.

Tableau 2: Plages de valeurs des hyper-paramètres utilisés.

Paramètre	Plage de valeurs
epochs	40, 80, 120, 160, 200, 240, 300
batch	4, 8, 16, 20, 30, 40
patience	50

Le tableau 3 donne les valeurs des autres paramètres utilisés pour les entraînements :

Tableau 3: Tableau des paramètres d'entraînement.

Paramètre	Description	Valeur
imgz	Taille des image (en pixels)	640
pretrained	Détermine s'il faut utiliser un modèle pré-entraîné.	True
seed	Fixe la graine des générateurs aléatoire pour garantir la reproductibilité des calculs d'une exécution à l'autre.	1234
workers	Nombre de <i>threads</i> de travail pour le chargement des données.	10

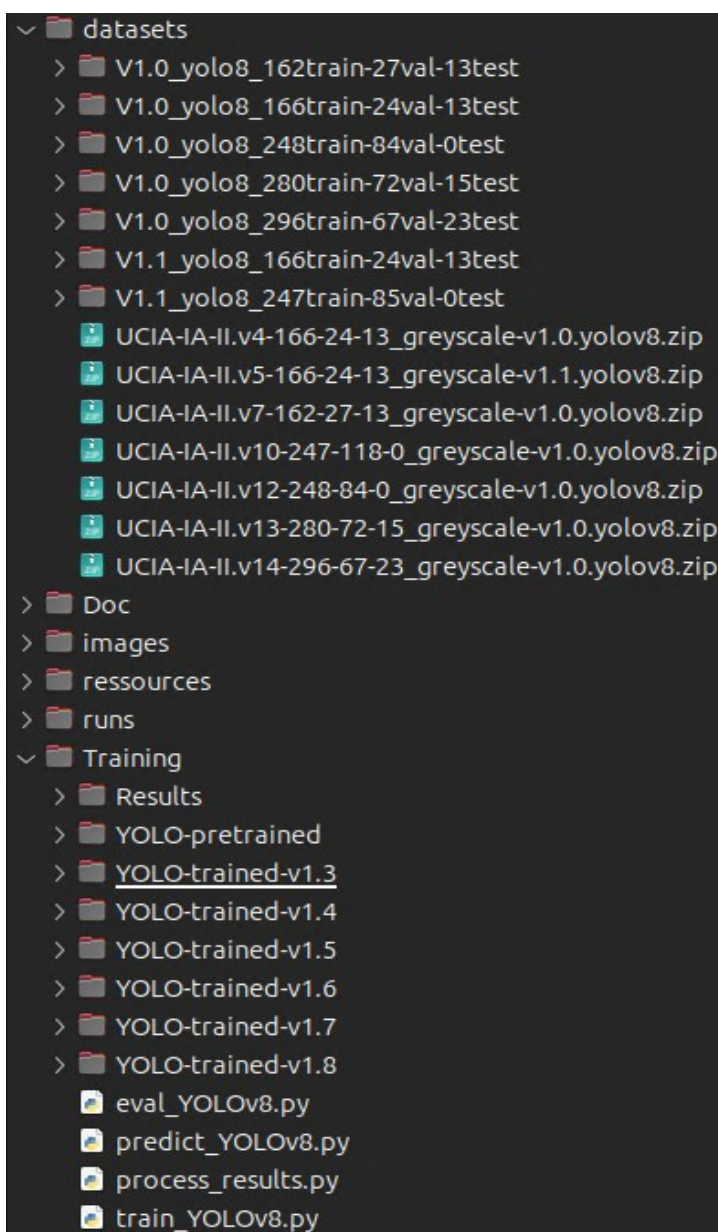
### 3.3 Environnement de calcul

Les plages de valeurs utilisées pour la version finale de l'entraînement du réseau **yolov8n** sont :

- 6 valeurs de l'hyper-paramètre **batch** : (4, 8, 16, 20, 30, 40),
- 8 valeur de l'hyper-paramètre **epoch** : (80, 120, 160, 200, 240, 300),

ce qui donne 6 entraînements, répétés  $80+120+160+200+240+300 = 1100$  fois, soit au total  $6 \times 1100 = 6600$  calculs d'entraînement. Sur un PC portable « core i7 » un entraînement sur ~300 images en ton de gris prend environ ~12 secondes (fonction de la valeur des hyper-paramètres), ce qui donnerait environ 22 h de calcul pour traiter tous les cas retenus. Nous utilisons un PC de calcul sous Ubuntu avec une carte graphique « Nvidia Quadro TRX8000 » pour réduire les temps de calcul à un peu plus de 3 heures.

L'arborescence du développement sur le PC de calcul est représentée sur la figure 6 :



#### datasets

contient les jeux d'images annotées téléchargés depuis le site Roboflow. Plusieurs jeux de données ont été créés pour arriver à une détection correcte des objets, en particulier les objets « virage gauche », « virage droite » et « piste droite » avec des objets 3D présents sur la piste noire.

Les meilleurs résultats ont été obtenus avec le dernier jeu de données du dossier **V1.0-yolo8\_296train-67val-23test**.

#### runs

contient les fichiers (texte, images...) résultats des évaluations des modèles entraînés.

#### Training

contient les fichiers Python pour l'entraînement et le test des réseaux de neurones :

- **Results** : contient les fichiers ASCII d'évaluation des réseaux entraînés.
- **YOLO\_pretrained** : contient le fichier binaire des poids du réseau **yolov8n** pré-entraîné.
- **YOLO-trained-v1.3** à **v1.6** : contient les poids du réseau **yolov8n** entraîné avec les valeurs des méta-paramètres du tableau 1, pour les différents jeux de données de dossier **datasets**.

Figure 6: Arborescence du projet.





- le tracé de la matrice de confusion,
- les indicateurs de performance du réseau entraîné : par exemple le fichier **results.png** (figure 8).

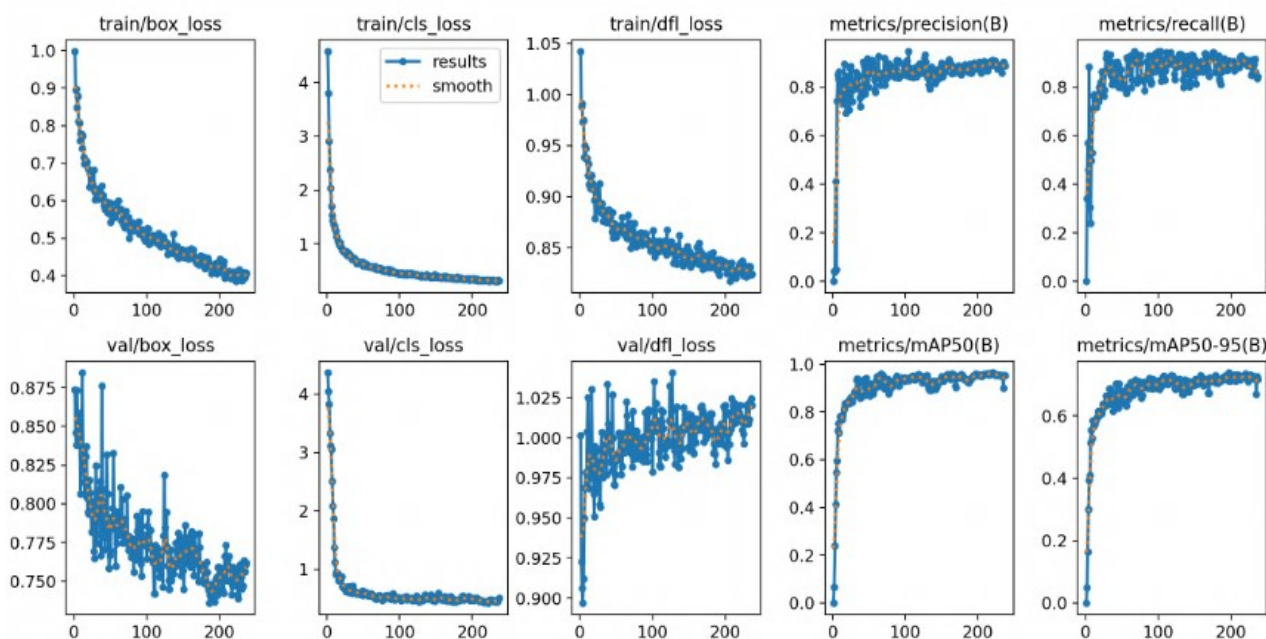


Figure 8: Exemples de statistiques d'entraînement.

### 3.5 Résultats des entraînements

Parmi toutes les combinaisons d'entraînement du réseau **yolov8n** on cherche à identifier celles qui optimisent le rapport « performances / rapidité de calcul ». Le module **ultralytics** fournit des outils pour évaluer des indicateurs de performance des réseaux de neurones entraînés à la détection d'objets (voir la page « Analyse approfondie des mesures de performance » <https://docs.ultralytics.com/fr/guides/yolo-performance-metrics/>).

Le programme **eval\_YOLOv8.py** (développé pour les études précédentes) évalue ces indicateurs pour chacune des combinaisons des hyper-paramètres d'entraînement. Les hyper-paramètres couramment utilisés pour évaluer la détection d'objets sont :

- **prec** (*precision*) : précision des objets détectés, indiquant le nombre de détections correctes.
- **recall** (*recall*) : capacité du réseau à identifier toutes les instances d'objets dans les images.
- **mAP50** : précision moyenne pour un seuil d'intersection sur union (IoU<sup>3</sup>) de 0,5. Mesure la précision d'un réseau détecteur d'objet pour des détections "faciles".
- **mAP50-95** : précision moyenne pour différents seuils IoU allant de 0,50 à 0,95 par pas de 0.5. Mesure la précision d'un réseau détecteur d'objet à différents niveaux de difficulté de détection.
- **fitness** :  $0.1 * \text{mAP50} + 0.9 * \text{mAP50-95}$

Le résultat de l'évaluation est écrit dans un fichier ASCII qui est ensuite traité pour extraire la combinaison des hyper-paramètres donnant les meilleurs indicateurs.

3 *Intersection over Union* (IoU) : L'intersection sur l'union donne la mesure du chevauchement entre les boîtes englobantes prédite et vraie. Elle joue un rôle fondamental dans l'évaluation de la précision de la localisation des objets. Voir [https://en.wikipedia.org/wiki/Jaccard\\_index](https://en.wikipedia.org/wiki/Jaccard_index).

## Early stopping des entraînements

Le phénomène *EarlyStopping* se produit lorsque la répétition d'un entraînement ne procure plus d'amélioration du réseau de neurones pendant un nombre d'itérations atteignant la valeur de l'hyper-paramètre **patience** (cf tableau 1 page 10).

Le message affiché en cas d'Early Stopping est du type :

EarlyStopping: Training stopped early as no improvement observed in last 50 epochs. Best results observed at epoch **N**, best model saved as best.pt.

Dans ce cas c'est l'état du réseau entraîné à l'epoch **N** qui est enregistré dans le fichier des poids **best.pt**. Avec l'entraînement **v1.8** qui correspond au *dataset* du dossier **V1.0-yolo8\_296train-67val-23test**, la valeur de **patience** utilisée pour toutes les combinaison des hyper-paramètres **batch** et **apochs** est de 50.

Le tableau 5 montre les événements *EarlyStopping* observés pour la combinaison des entraînements effectués.

Tableau 5: *EarlyStopping* des combinaisons d'entraînement.

batch	epochs	EarlyStop at	Best at
4	80		80
	120		120
	160		160
	200	127	77
	240	87	37
	300	215	161
8	80		80
	120	76	26
	160		160
	200	173	123
	240	192	146
	300	269	149
16	80		80
	120		120
	160		160
	200		200
	240	202	152
	300	233	183
20	80		80
	120	119	69
	160	155	105
	200	200	150
	240	230	180
	300	121	71
30	80		80
	120		120
	160		160
	200	117	67
	240	227	177
	300	231	181
40	80		80
	120		120
	160	97	47
	200	184	134
	240	151	101
	300	242	192

## Performances des réseaux entraînés sur le PC de calcul

Le détail des validations des différents entraînements peut être trouvé sur le dépôt GitHub du projet, avec les fichiers ASCII du dossier **Training/Results**.

Pour l'entraînement final avec les données annotées du dossier **V1.0-yolo8\_296train-67val-23test**, l'évaluation correspond au fichier **results\_yolov8n-v1.8.txt**, visible en annexe page 39.

On regarde alors les colonnes des indicateurs **recall**, **mAP50-95** et **fitness** du fichier résultat **results\_yolov8n\_v1.8.txt**, qui sont les plus significatifs pour quantifier les performances pour la détection d'objets.

Pour synthétiser les résultats nous avons développé dans l'étude précédente le programme Python **process\_results.py** : il lit les données du fichier **results\_yolov8n\_v1.8.txt** et les trie par ordre décroissant :

- de la colonne **mAP50**,
- des colonnes **recall** et **mAP50-95**,
- puis de la colonne **fitness**.

On affiche à chaque fois les 4 premières lignes qui montrent les meilleures combinaisons d'entraînement.

En triant avec la colonne **mAP50-95**, on obtient :

```
*****
* Sort by 'mAP50'
*****
Max values -> "mAP50": 0.962
#meta-params recall mAP50 mAP50-95 fitness
36 batch-40_epo-300 0.940 0.962 0.825 0.838
30 batch-30_epo-300 0.948 0.962 0.822 0.836
29 batch-30_epo-240 0.946 0.955 0.815 0.829
21 batch-20_epo-160 0.868 0.900 0.767 0.780
```

En triant avec les colonnes **recall** et **mAP50-95**, on obtient :

```
*****
* Sort by 'recall' & 'mAP50-95'
*****
Max values -> "max_recall": 0.948, "max_mAP50-95": 0.822
#meta-params recall mAP50 mAP50-95 fitness
30 batch-30_epo-300 0.948 0.962 0.822 0.836
29 batch-30_epo-240 0.946 0.955 0.815 0.829
36 batch-40_epo-300 0.940 0.962 0.825 0.838
1 batch-04_epo-080 0.888 0.887 0.727 0.743
```

Et en triant avec la colonne **fitness**, on obtient :

```
*****
* Sort by 'fitness'
*****
Max values -> "fitness": 0.838
#meta-params recall mAP50 mAP50-95 fitness
36 batch-40_epo-300 0.940 0.962 0.825 0.838
30 batch-30_epo-300 0.948 0.962 0.822 0.836
29 batch-30_epo-240 0.946 0.955 0.815 0.829
21 batch-20_epo-160 0.868 0.900 0.767 0.780
```

Les configurations d'entraînement qui donnent les meilleurs résultats avec les images de validation sont : **batch-40\_epo-300** et **batch-30\_epo-300**.

## Matrices de confusion

La matrice de confusion permet également de quantifier les performances du réseau de neurones entraîné.

Avec l'entraînement **v1.8** qui correspond au *dataset* du dossier **V1.0-yolo8\_296train-67val-23test**, les figures 9 et 10 montrent la matrice de confusion obtenue avec le programme **eval\_YOLOv8.py** pour les configurations d'entraînement respectives : **batch-40\_epo-300** et **batch-30\_epo-300**.

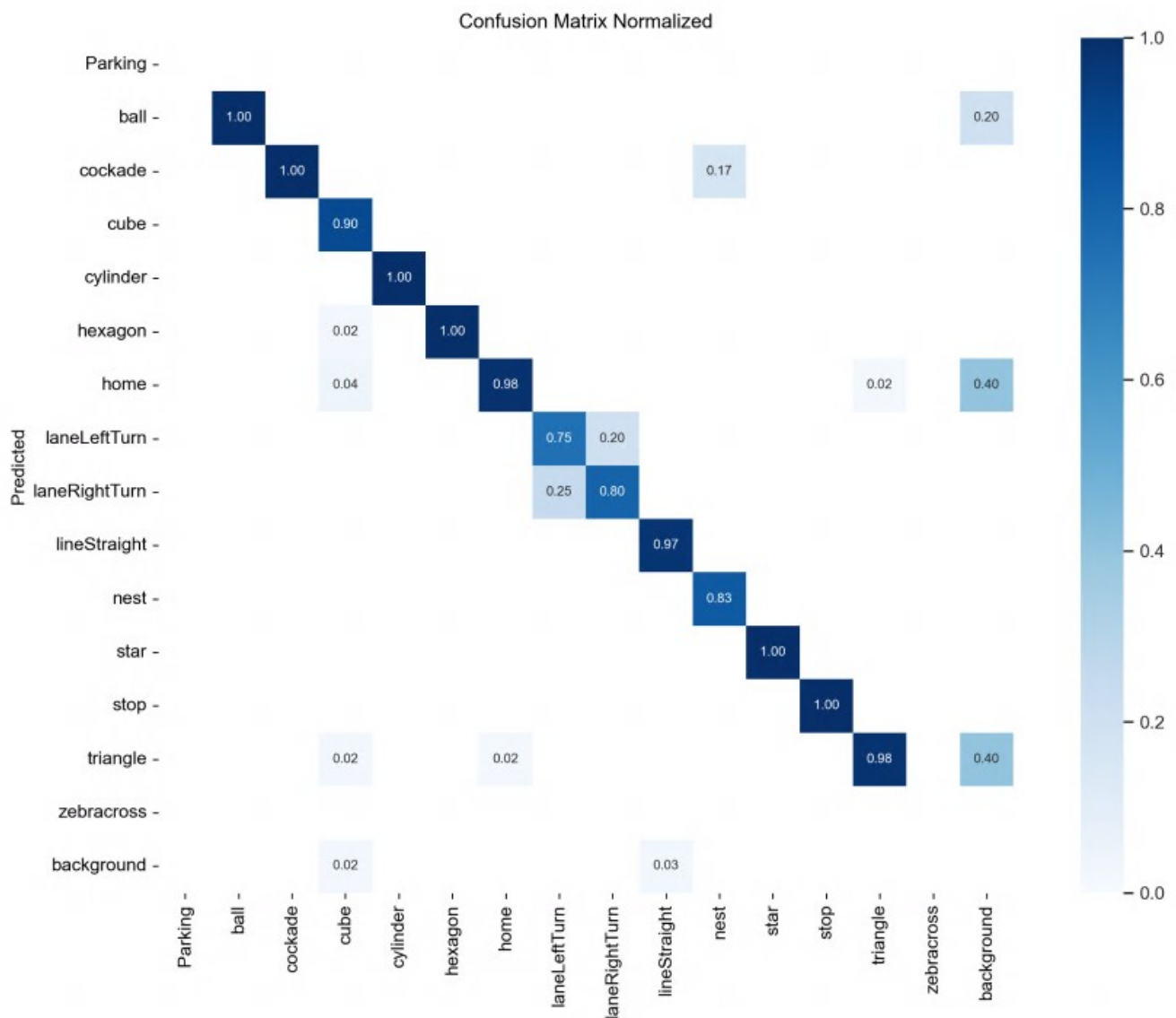


Figure 9: Matrice de confusion (batch=40, epochs=300, dataset v1.8).

On peut constater que les détections sont bien sur la diagonale, avec des pourcentages proches de 100 %, sauf pour les classes « laneLeftTurn » et « laneRightTurn » qui n'atteignent que 75 % et 80 %.



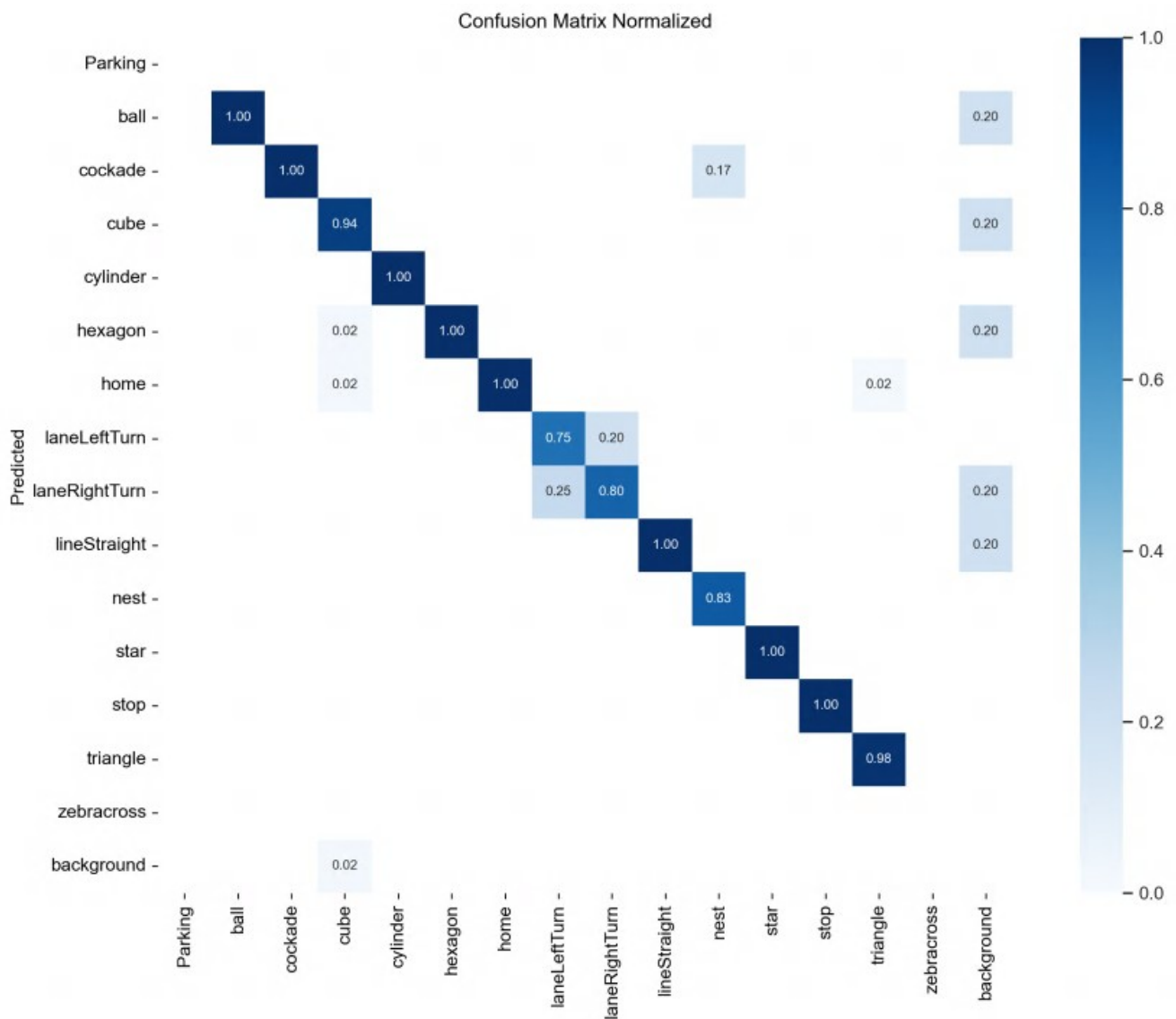


Figure 10: Matrice de confusion (batch=30, epochs=300, dataset v1.8).

On peut constater les détections sont bien sur la diagonale, avec des pourcentages proches de 100 %, sauf pour les classes « laneLeftTurn » et « laneRightTurn » qui n'atteignent que 75 % et 80 %.

### 3.6 Conclusion

Les configurations d'entraînement qui donnent les meilleurs résultats avec les images de validation sont : **batch-40\_epo-300** et **batch-30\_epo-300**.

Pour être rigoureux, il reste à faire la même évaluation sur la RPi4, avec les fichiers de poids au format **NCNN**. En effet ce format résulte d'une « dégradation » de l'architecture du réseau **yolov8n**, de utilisation d'entiers (**int**) au lieu de flottants (**float**)... pour occuper moins de place en RAM et être calculé plus rapidement.

Le paragraphe 4.2 présente l'évaluation des des différents entraînements obtenus sur RPi4.

## 4 Exploitation du réseau YOLO8 sur RPi4

### 4.1 Préparation de la carte SD pour la RPi4

L'installation du système d'exploitation « Raspberry PI OS (64bits) » sur la carte micro-SD est présentée pages 19-20 du document [UCIA-IA-DetectionObj\\_V2.1.pdf](#) de l'étude précédente (dépôt GiHb [UCIA\\_ObjectDetection-I](#), dossier [Doc](#)).

#### Identifiant SSID unique du WiFi

Pour obtenir un identifiant WiFi unique propre à chaque carte RPi4, le fichier `/usr/bin/rpi-access-point` de la carte micro-SD est modifié pour ajouter les 3 derniers octets de l'adresse MAC à l'identifiant SSID.

Exemple : avec une carte RPi4 d'adresse MAC `e4:5f:01:6d:71:96`, le SSID du pont d'accès émis par la carte devient : `RPi4-UCIA_6d-71-94`.

### 4.2 Évaluation des entraînements du réseau yolov8n

#### Performances sur RPi4 des réseaux entraînés sur le PC de calcul

Le format **NCNN** diffère du format **pytorch** utilisé sur le PC de calcul pour enregistrer les poids du réseau entraîné, d'où l'intérêt de refaire sur la carte RPi4 l'évaluation des entraînements du réseau **yolov8n**. C'est le même programme `eval_YOLOv8.py`, qui est utilisé pour créer le fichier `results_yolov8n-v1.8.txt` (présent sur la carte micro-SD et visible en annexe page 40). Le programme `process_results.py` permet de synthétiser les résultats :

En triant avec la colonne **mAP50-95**, on obtient :

```
*****
* Sort by 'mAP50'
*****
Max values -> "max_mAP50": 0.956
#meta-params recall mAP50 mAP50-95 fitness
29 batch-30_epo-300 0.943 0.956 0.808 0.823
21 batch-20_epo-200 0.877 0.898 0.751 0.765
18 batch-20_epo-080 0.857 0.897 0.738 0.754
3 batch-04_epo-200 0.864 0.894 0.731 0.747
```

En triant avec les colonnes **recall** et **mAP50-95**, on obtient :

```
*****
* Sort by 'recall' & 'mAP50-95'
*****
Max values -> "max_recall": 0.943, "max_mAP50-95": 0.808
#meta-params recall mAP50 mAP50-95 fitness
29 batch-30_epo-300 0.943 0.956 0.808 0.823
12 batch-16_epo-080 0.882 0.888 0.734 0.750
34 batch-40_epo-240 0.881 0.890 0.742 0.756
21 batch-20_epo-200 0.877 0.898 0.751 0.765
```

Et en triant avec la colonne **fitness**, on obtient :

```
*****
* Sort by 'fitness'
*****
Max values -> "fitness": 0.823
#meta-params recall mAP50 mAP50-95 fitness
29 batch-30_epo-300 0.943 0.956 0.808 0.823
20 batch-20_epo-160 0.852 0.888 0.757 0.770
21 batch-20_epo-200 0.877 0.898 0.751 0.765
17 batch-16_epo-300 0.871 0.892 0.748 0.762
```

C'est la combinaison **batch-30\_epo-300** qui donne les meilleurs résultats d'évaluation sur la RPi4 : c'est cette configuration qui est mise par défaut pour tous les programmes Python d'exploitation du réseau **yolov8n**.

On peut noter que c'est une des 2 combinaisons obtenues pour l'évaluation sur le PC de calcul (cf page 15).

### Matrices de confusion

Avec l'entraînement **v1.8** qui correspond au *dataset* du dossier **V1.0-yolo8\_296train-67val-23test**, la figure 11 montre la matrice de confusion obtenue avec le programme **eval\_YOLOv8.py** pour la configuration d'entraînement **batch-30\_epo-300**.

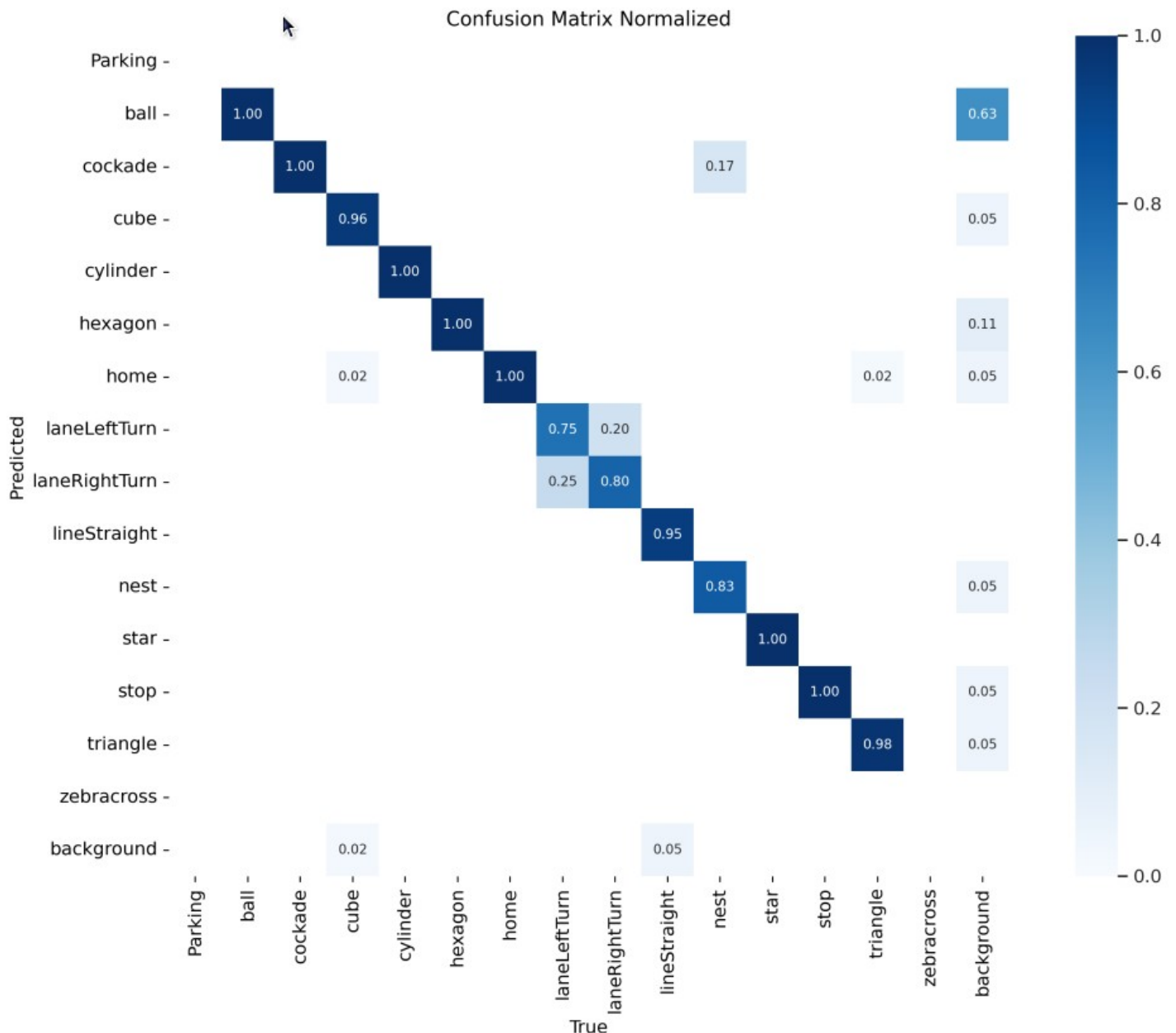


Figure 11: Matrice de confusion sur la RPi4 ( batch=30, epochs=300, dataset v1.8).

Comme sur le PC de calcul, on observe que la majorité des détections est bien sur la diagonale, avec des pourcentages proches de 100 %, sauf pour les classes « laneLeftTurn » et « laneRightTurn » qui n'atteignent que 75 % et 80 %.

### 4.3 Exploitation du réseau YOLO8 depuis le bureau à distance

#### Le bureau à distance avec le client RealVNC

Le service de « bureau à distance » étant activé au lancement de la RPi4, il suffit de télécharger la version gratuite de l'application cliente « RealVNC Viewer » sur le site [www.realvnc.com](http://www.realvnc.com) et de l'installer sur un ordinateur portable ou sur un smartphone.

La configuration est simple :

- le SSID WiFi est celui qui est ajouté à la liste des SSID existants après avoir démarré la carte RPi4, il commence nécessairement par **RPi4-UCIA\_** suivi des 3 octets de l'adresse MAC de la carte RPi4
- l'adresse du serveur VNC est celle de la RPi4 : **10.99.99.1**
- le compte à utiliser **ucia** avec le mot de passe **poppy!station**.

On obtient alors sur le périphérique distant (ordinateur, smartphone...) une fenêtre graphique qui permet d'utiliser à distance le bureau de la RPi4 (voir figure 12).

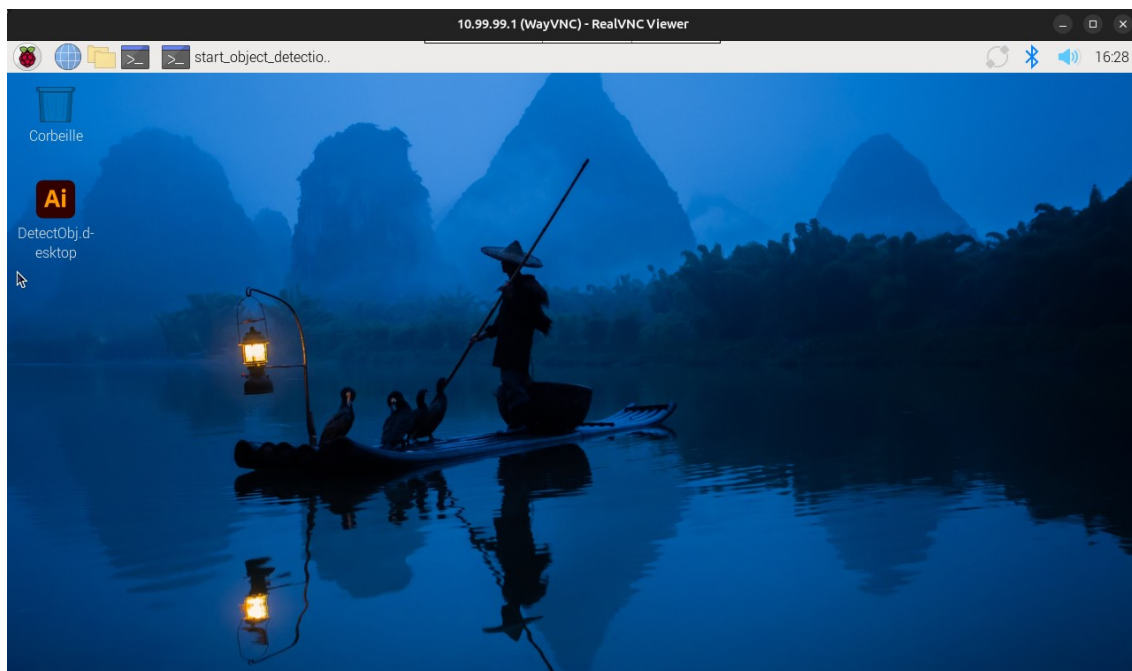


Figure 12: RealVnc Viewer : le bureau RPi4 à distance.

#### Dossier de travail positionné sur UCIA/UCIA\_ObjectDetection-II

Tous les programmes Python d'exploitation du réseau **yolov8n** sont dans le dossier **UCIA/UCIA\_ObjectDetection-II** du compte **ucia**. Le fichier **/home/ucia/.bashrc** est modifié pour se positionner automatiquement dans ce dossier. Au lancement du terminal, on peut donc vérifier le dossier de travail :

```
(vision) ucia@raspberrypi ~/UCIA/UCIA_ObjectDetection-II $
```

#### detect\_camera-1.py : détection des objets, rendu graphique N&B

On rappelle que les images fournies au réseau de neurones sont converties en ton de gris : l'information de couleur est absente, seule la position dans l'image et la forme des objets

sont apprises. Le programme `detect_camera-1.py`<sup>4</sup> permet détecter des objets dans les images de la caméra de la RPi4 avec une configuration d'entraînement du réseau `yolov8n` qui peut être choisie.

Le programme affiche en temps réel :

- Dans le terminal de lancement : le nombre et le nom de chaque classe d'objet détecté (les noms des classes sont en anglais) et les temps de *pre-processing*, d'*inference* et de *post-processing* nécessaires au traitement de chaque image (cf figure 10).
- Dans la **fenêtre graphique** : l'image de la caméra en ton de gris avec le tracé des boîtes englobantes, les nom des classes et les confiances de détection (cf figure 14).

Le lancement du programme se fait dans le terminal positionné dans le répertoire `UCIA/UCIA_ObjectDetection` :

```
(vision) ucia@raspberrypi:~/UCIA/UCIA_ObjectDetection $ python detect_camera-1.py
```

L'entraînement par défaut est celui de la configuration `batch-30_epo-300`.

On peut quitter le programme :

- soit en tapant la touche [Q] dans la fenêtre graphique,
- soit en tapant la séquence de touches « Ctrl + C » dans le terminal.

```

ucia@raspberrypi: ~/UCIA/UCIA_ObjectDetection-II
Fichier  Édition  Onglets  Aide
(vision) ucia@raspberrypi:~/UCIA/UCIA_ObjectDetection-II $ python detect_camera-1.py
[0:07:34.565748209] [2074]  INFO Camera camera_manager.cpp:325 libcamera v0.3.2+99-1230f78d
[0:07:34.594440116] [2080]  WARN RPiSdn sdn.cpp:40 Using legacy SDN tuning - please consider moving
SDN inside rpi.denoise
[0:07:34.597420764] [2080]  INFO RPi vc4.cpp:447 Registered camera /base/soc/i2c0mux/i2c@1/ov5647@3
6 to Unicam device /dev/media1 and ISP device /dev/media2
[0:07:34.597558005] [2080]  INFO RPi pipeline_base.cpp:1120 Using configuration file '/usr/share/li
bcamera/pipeline/rpi/vc4/rpi_apps.yaml'
[0:07:34.604617116] [2074]  INFO Camera camera.cpp:1197 configuring streams: (0) 640x640-RGB888 (1)
1296x972-SGBRG10_CSI2P
[0:07:34.605280709] [2080]  INFO RPi vc4.cpp:622 Sensor: /base/soc/i2c0mux/i2c@1/ov5647@36 - Select
ed sensor format: 1296x972-SGBRG10_1X10 - Selected unicam format: 1296x972-pGAA
Loading Training/YOLO-trained-v1.8/UCIA-II-YOLOv8n/batch-30_epo-300/weights/best_ncnn_model for NCN
N inference...

0: 640x640 1 ball, 2 cylinders, 1 home, 1 lineStraight, 1 nest, 2 stars, 2 triangles, 412.7ms
Speed: 13.1ms preprocess, 412.7ms inference, 6.0ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 1 ball, 2 cylinders, 1 home, 1 lineStraight, 1 nest, 2 stars, 2 triangles, 804.9ms
Speed: 12.3ms preprocess, 804.9ms inference, 4.1ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 1 ball, 2 cylinders, 1 home, 1 lineStraight, 1 nest, 2 stars, 2 triangles, 460.8ms
Speed: 18.8ms preprocess, 460.8ms inference, 3.3ms postprocess per image at shape (1, 3, 640, 640)

```

Figure 13: `detect_camera-1.py` : terminal de lancement du programme.

<sup>4</sup> Inspiré de <https://docs.ultralytics.com/fr/guides/raspberry-pi/#inference-with-camera>





Figure 14: `detect_camera-1.py` : fenêtre graphique d'affichage des objets détectés.

## detect\_camera-2.py : détection des objets, rendu graphique couleur

Fonctionne comme le précédent, avec un rendu couleur. Il affiche en temps réel :

- Dans le terminal : en plus des informations affichées par le programme **detect\_camera-1.py**, pour chaque objet détecté (par ordre de confiance décroissant, cf figure 15) :
  - le numéro de la classe de l'objet (entre 0 et 14),
  - la confiance de la détection de l'objet (entre 0 et 1.),
  - les coordonnées x1, y2, x2, y1 des coins « bas-gauche » et « haut-droit » de la boîte englobante de l'objet,
  - les moyennes des composantes R, G, B des pixels contenus dans un carré de 14x14 pixels au centre de la boîte englobante.
- Dans la **fenêtre graphique couleur** : le tracé des boîtes englobantes avec le nom de l'objet (en français par défaut) et la confiance de détection (cf figure 16).

Le lancement se se fait dans le terminal, depuis le dossier **UCIA/UCIA\_ObjectDetection/** :

```
(vision) ucia@raspberrypi:~/UCIA/UCIA_ObjectDetection $ python detect_camera-2.py
```

```

ucia@raspberrypi: ~/UCIA/UCIA_ObjectDetection-II
Fichier  Édition  Onglets  Aide
idlex    x    ucia@raspb... x
(vision) ucia@raspberrypi:~/UCIA/UCIA_ObjectDetection-II $ python3 detect_camera-2.py
[0:04:39.625058925] [1943]  INFO Camera camera_manager.cpp:325 libcamera v0.3.2+99-1230f78d
[0:04:39.653775610] [1949]  WARN RPiSdn sdn.cpp:40 Using legacy SDN tuning - please consider moving SDN
inside rpi.denoise
[0:04:39.656963647] [1949]  INFO RPI vc4.cpp:447 Registered camera /base/soc/i2c0mux/i2c@1/ov5647@36 to
Unicam device /dev/media0 and ISP device /dev/media2
[0:04:39.657318647] [1949]  INFO RPI pipeline_base.cpp:1120 Using configuration file '/usr/share/libcam
era/pipeline/rpi/vc4/rpi_apps.yaml'
[0:04:39.664902017] [1943]  INFO Camera camera.cpp:1197 configuring streams: (0) 640x640-RGB888 (1) 129
6x972-SGBRG10_CSI2P
[0:04:39.665318239] [1949]  INFO RPI vc4.cpp:622 Sensor: /base/soc/i2c0mux/i2c@1/ov5647@36 - Selected s
ensor format: 1296x972-SGBRG10_1X10 - Selected unicam format: 1296x972-pGAA
Loading Training/YOLO-trained-v1.8/UCIA-II-YOLOv8n/batch-30_epo-300/weights/best_ncnn_model for NCNN in
ference...

0: 640x640 1 ball, 2 cylinders, 1 home, 2 lineStraights, 1 nest, 2 stars, 2 triangles, 487.5ms
Speed: 96.7ms preprocess, 487.5ms inference, 161.0ms postprocess per image at shape (1, 3, 640, 640)
13 0.96 88 594 160 512 128 215 100
13 0.95 478 501 526 436 19 145 211
4 0.91 178 544 236 470 2 56 159
11 0.91 295 594 368 529 242 61 142
6 0.91 184 465 238 410 17 154 203
10 0.91 381 429 545 370 144 130 134
1 0.85 441 603 505 535 0 61 176
9 0.80 236 640 387 470 231 49 145
11 0.78 289 482 337 440 95 89 182
4 0.70 397 524 453 455 65 55 164
9 0.55 234 639 389 520 186 22 100
  
```

Figure 15: detect\_camera-2.py : Terminal de lancement du programme.

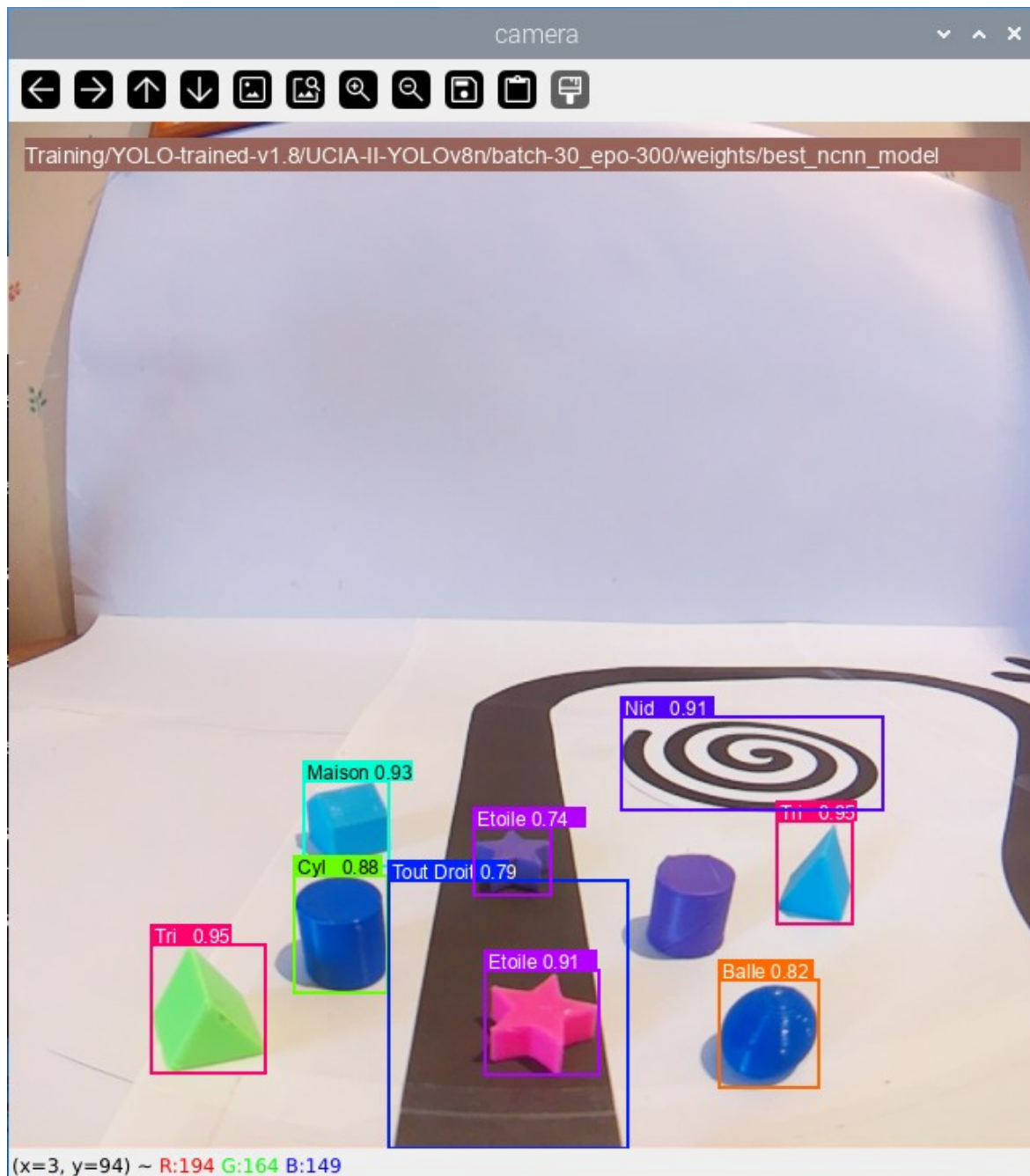


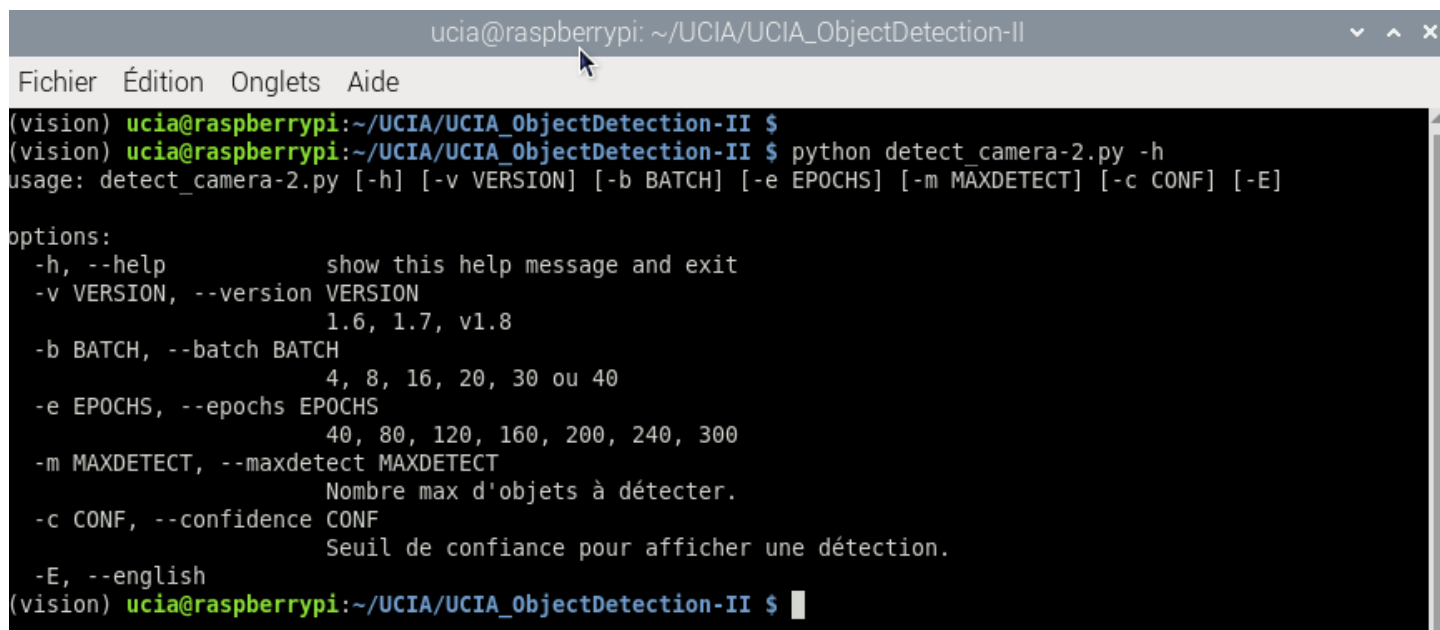
Figure 16: detect\_camera-2.py : fenêtre graphique d'affichage des objets détectés.



## Choix des entraînements du réseau YOLO8 à exploiter

Tous les programmes Python `detect_camera_n.py` avec `n = 1, 2, 3` ou `4` peuvent être lancés avec des options permettant de choisir un entraînement particulier du réseau `yolov8n`.

L'option `-h` (*help*) affiche l'aide sur l'utilisation du programme :



```

ucia@raspberrypi: ~/UCIA/UCIA_ObjectDetection-II
Fichier  Édition  Onglets  Aide
(vision) ucia@raspberrypi:~/UCIA/UCIA_ObjectDetection-II $
(vision) ucia@raspberrypi:~/UCIA/UCIA_ObjectDetection-II $ python detect_camera-2.py -h
usage: detect_camera-2.py [-h] [-v VERSION] [-b BATCH] [-e EPOCHS] [-m MAXDETECT] [-c CONF] [-E]
options:
  -h, --help            show this help message and exit
  -v VERSION, --version VERSION
                        1.6, 1.7, v1.8
  -b BATCH, --batch BATCH
                        4, 8, 16, 20, 30 ou 40
  -e EPOCHS, --epochs EPOCHS
                        40, 80, 120, 160, 200, 240, 300
  -m MAXDETECT, --maxdetect MAXDETECT
                        Nombre max d'objets à détecter.
  -c CONF, --confidence CONF
                        Seuil de confiance pour afficher une détection.
  -E, --english
(vision) ucia@raspberrypi:~/UCIA/UCIA_ObjectDetection-II $

```

Figure 17: Les options de lancement des programmes `detect_camera_*.py`.

Les options possibles sont :

- h** affiche l'aide.
- v VERSION** choix d'une version d'entraînement parmi **1.6**, **1.7**, **1.8**.  
Valeur par défaut : **1.8** (correspond au dernier jeu d'images).
- b BATCH** choix du méta-paramètre **batch** parmi **4**, **8**, **16**, **20**, **30**, **40**.  
Valeur par défaut : **30**.
- e EPOCHS** choix du méta-paramètre **epochs** **40**, **80**, **120**, **160**, **200**, **240** et **300**.  
Valeur par défaut : **300**.
- m MAXDETECT** choix du nombre maximum de détections d'objet à afficher (entre 1 et 20). Valeur par défaut : **15**.
- c CONF** choix du seuil de confiance à partir duquel afficher la détection de l'objet, entre 0 et 1. Valeur par défaut : **0.5**.
- E** pour afficher les noms des classes d'objets en anglais.  
Par défaut c'est le français qui est utilisé.

Par exemple pour choisir le `yolov8n` entraîné avec les images de la version **1.8**, **batch** égal à **16**, **epoch** égal à **100**, **10** objets détectés au maximum avec un niveau de confiance meilleur que **0.7** :

```

(vision) ucia@raspberrypi:~/UCIA/UCIA_ObjectDetection $ python detect_camera-2.py
-v 1.8 -b 16 -e 100 -m 6 -c 0.7

```

## 4.4 Exploitation du réseau YOLO8 depuis un navigateur distant

Au démarrage de la RPi4, c'est le programme `detect_camera_3.py` qui est exécuté : il lance un serveur WEB sur l'adresse <http://10.99.99.1:5000> en attente de connexion d'un navigateur sur cette adresse. Il n'est pas nécessaire de brancher un écran / clavier / souris ou de lancer un bureau à distance.

Dès qu'un client distant connecté au WiFi de la RPi4 (ordi portable, smartphone, tablette...) lance un navigateur WEB sur l'URL <http://10.99.99.1:5000/video>, le programme `detect_camera_3.py` lance la détection des objets et les images couleurs s'affichent en temps réel dans le navigateur :

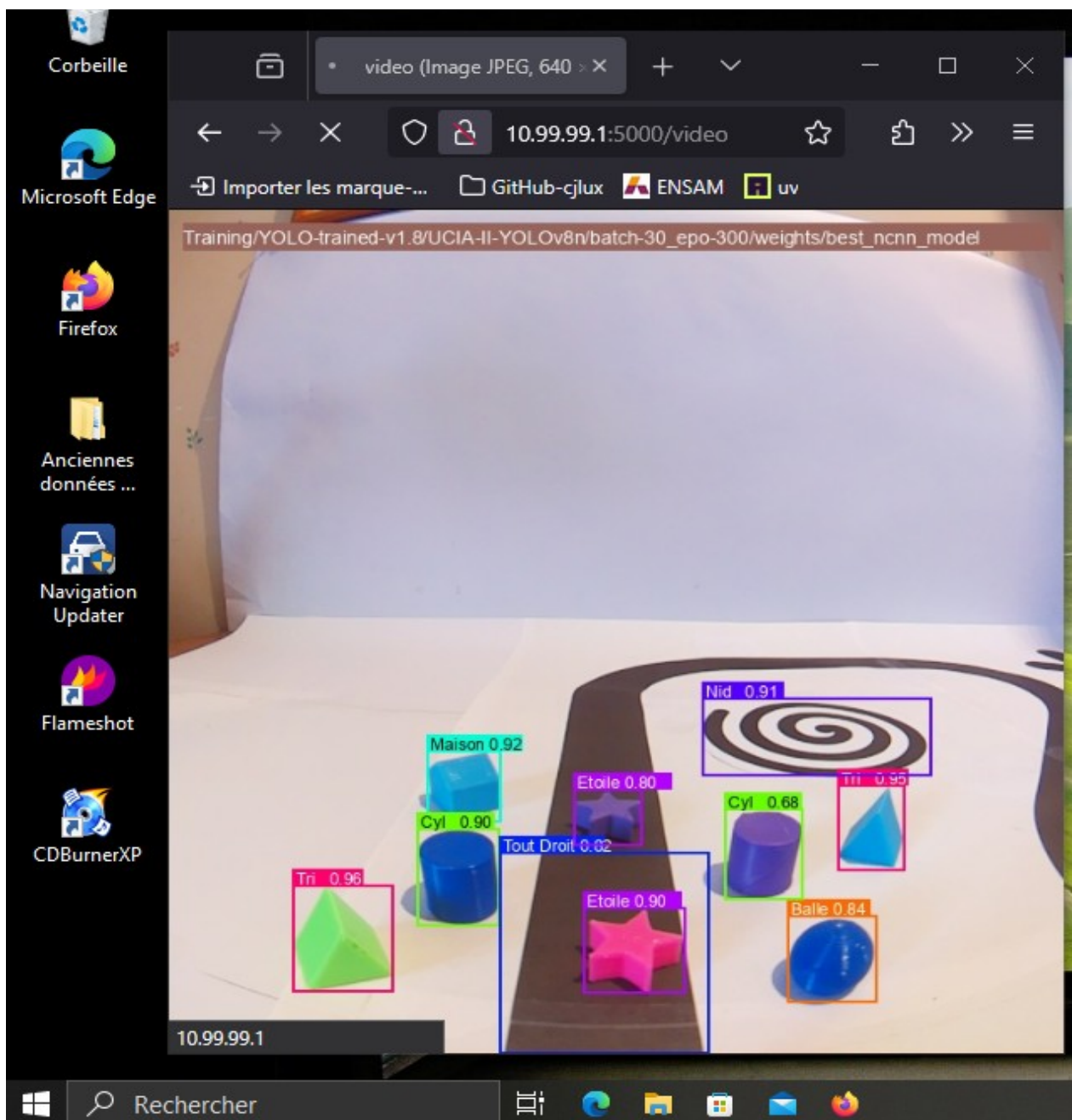


Figure 18: Connexion d'un navigateur sur l'URL <http://10.99.99.1:5000/video>.

## Arrêt (*shutdown*) de la carte RPi4 depuis le navigateur WEB

L'arrêt de la carte RPi4 (*shutdown*) peut être obtenu en ouvrant avec le navigateur l'URL <http://10.99.99.1:5000/halt> .

Une fois que la LED verte ne clignote plus sur la RPi4, on peut débrancher/éteindre son alimentation en toute sécurité.

## 5 Conclusions

Cette étude étend l'étude précédente pour l'entraînement du réseau **yolov8n** à détecter les 15 classes d'objets retenues :

- 7 objets 3D : **ball**, **cylinder**, **cube**, **home**, **hexagon**, **star**, **triangle**

- 8 objets imprimés sur la piste :

<b>laneLeft</b>	piste tourne à gauche,	<b>Stop</b>	panneau STOP,
<b>laneRight</b>	piste tourne à droite,	<b>parking</b>	panneau Parking,
<b>straightLine</b>	piste droite,	<b>zebracross</b>	passage piéton,
<b>nest</b>	nid,	<b>cockate</b>	cocarde.

Un travail particulier a du être fait sur les images avec les classes :

- « piste tourne à gauche »,
- « piste tourne à droite »,
- « piste droite »

avec ou sans objet 3D posé sur la piste, afin d'obtenir une détection / classification satisfaisante de ces objets.

Les inférences du réseau **yolov8n** pour la détection de ces 15 classes d'objets reste proches de la demi-seconde, ce qui permet au programme de pilotage du robot Thymio de traiter les détections du réseau de neurones pour piloter le comportement du robot selon des algorithmes variés.

Tous les fichiers et documents de l'étude sont sur le nouveau dépôt GitHub :

[https://github.com/cjlux/UCIA\\_ObjectDetection-II](https://github.com/cjlux/UCIA_ObjectDetection-II)

## 6 Glossaire

Nom anglais	Nom français	signification
<i>Epoch</i>	Époque	Une itération de l'entraînement du réseau de neurones sur l'ensemble complet des données.
<i>Batch</i>	Lot	Sous -ensemble du jeu complet des données fourni pour l'entraînement du réseau de neurones.. Dans cette étude : un paquet d'images fournies pour un entraînement du réseau de neurones
<i>batch size</i>	Taille de lot	Méta-paramètre qui fixe la taille du lot fourni au réseau de neurones. Dans notre étude c'est le nombre d'images fournies à chaque entraînement. Nota : les programmes Python du modules
<i>Overfitting</i>	Sur-entraînement	C'est un défaut de l'entraînement, où le réseau est sur-entraîné avec les données d'entraînement, et par suite il devient moins performant pour faire des déductions correctes sur de nouvelles images qu'il n'a jamais vues.
<i>Patience</i>	Patience	Nombre d'époques à attendre sans amélioration des mesures de validation avant d'arrêter l'entraînement. Permet d'éviter l' <i>overfitting</i> en arrêtant l'entraînement lorsque les performances atteignent un plateau.
<i>Précision</i>	Precision	Dans le contexte de la détection d'objets, désigne le pourcentage d'objets correctement détecté
<i>Rappel</i>	Recall	La capacité du modèle à identifier toutes les instances d'objets dans les images.

## 7 Annexes

### 7.1 Première utilisation des poids du réseau au format NCNN

Lorsqu'on utilise pour la première fois un fichier au format **NCNN** sur la RPi4, le module **ultralytics** affiche le message de la figure 19:

```
(vision) ucia@raspberrypi:~/UCIA/UCIA_ObjectDetection $ python eval.py
model loaded in 0.8 ms
image pre-processing: 55.0 ms
Loading YOLO-trained/UCIA-YOLOv8n/batch-08_pat-100_epo-080/weights/best_ncnn_model for NCNN inference...
requirements: Ultralytics requirement ['git+https://github.com/Tencent/ncnn.git'] not found, attempting AutoUpdate...
Running command git clone --filter=blob:none --quiet https://github.com/Tencent/ncnn.git /tmp/pip-req-build-km5uft8k
Running command git submodule update --init --recursive -q
```

*Figure 19: Message du module ultralytics pour la première utilisation du format ncnn.*

En clair, le message est :

requirements: Ultralytics requirement ['git+https://github.com/Tencent/ncnn.git'] not found, attempting AutoUpdate...

Running command git clone --filter=blob:none --quiet https://github.com/Tencent/ncnn.git /tmp/pip-req-build-km5uft8k

Running command git submodule update --init --recursive -q

Le chargement et la compilation du module prennent un bon quart d'heure sur RPi4...

## 7.2 Prise d'images avec la caméra de RPi4

### take\_image.py

```
#####  
#   Jean-Luc.Charles@mailo.com  
#   2024/11/21 - v1.0  
#####  
  
from picamera2 import Picamera2, Preview  
import sys, time  
  
picam2 = Picamera2()  
picam2.preview_configuration.main.size = (800, 600)  
picam2.configure("preview")  
picam2.start_preview(Preview.QTGL, width=800, height=600)  
picam2.start()  
  
n = 1  
rep = input("numéro image pour démarrer [Q:quit] ? ")  
  
if rep.lower() == 'q':  
    picam2.stop()  
    sys.exit()  
else:  
    n = int(rep)  
  
while True:  
    rep = input(f"ENTER -> image suivante {n:03d} [Q:quit] ...")  
    if rep.lower() == 'q':  
        break  
  
    picam2.capture_file(f"objets3D-{n:03d}.jpg")  
    time.sleep(1)  
    n += 1  
  
picam2.stop()
```

## 7.3 Programmes pour l'entraînement et l'évaluation du réseau yolov8n

### train\_YOLOv8.py

```
#####
#   Jean-Luc.Charles@mailo.com
#   2025/04/14 - v1.1
#####

from pathlib import Path

from ultralytics import YOLO
from time import sleep

#
# this program must be run from the UCIA_ObjectDetection directory
#

def main(VER:str, force_NCNN=False):

    BATCH = {'v1.3': (4, 8, 16, 32),
              'v1.4': (4, 8, 16, 20, 30, 40),
              'v1.5': (4, 8, 16, 20, 30, 40),
              'v1.6': (4, 8, 16, 20, 30, 40),
              'v1.7': (4, 8, 16, 20, 30, 40),
              'v1.8': (4, 8, 16, 20, 30, 40),
              }

    EPOCH = {'v1.3': (40, 80, 120, 160, 200),
             'v1.4': (80, 120, 160, 200, 240, 300),
             'v1.5': (80, 120, 160, 200, 240, 300, 400, 500),
             'v1.6': (80, 120, 160, 200, 240, 300),
             'v1.7': (80, 120, 160, 200, 240, 300),
             'v1.8': (80, 120, 160, 200, 240, 300),
             }

    PATIENCE = {'v1.3': 100, 'v1.4': 100, 'v1.5': 100, 'v1.6': 40, 'v1.7': 40,
                'v1.8': 50,}

    model_dir = Path('./Training/YOLO-pretrained')

    data_path = {'v1.3': './datasets/V1.1_yolo8_166train-24val-13test/data.yaml',
                 'v1.4': './datasets/V1.0_yolo8_162train-27val-13test/data.yaml',
                 'v1.5': './datasets/V1.1_yolo8_247train-85val-0test/data.yaml',
                 'v1.6': './datasets/V1.0_yolo8_248train-84val-0test/data.yaml',
                 'v1.7': './datasets/V1.0_yolo8_280train-72val-15test/data.yaml',
                 'v1.8': './datasets/V1.0_yolo8_296train-67val-23test/data.yaml',
                 }

    yolo = 'YOLOv8n'
    yolo_weights = f'{yolo.lower()}.pt'

    for batch in BATCH[VER]:
        for epoch in EPOCH[VER]:
            project = f'Training/YOLO-trained-{VER}/UCIA-II-{yolo}'
            name = f'batch-{batch:02d}_epo-{epoch:03d}'
            best = Path(project, name, 'weights', 'best.pt')
            print(f'{best}')
```



```

if not best.exists():
    model = YOLO(model_dir / yolo_weights) # load a pretrained model
    model.train(data=data_path[VER],
                epochs=epoch,
                imgsz=640,
                batch=batch,
                patience=PATIENCE[VER],
                cache=False,
                workers=10, # no parallelisation for loading data
                project=project,
                name=name,
                exist_ok=True,
                pretrained=True,
                optimizer='auto',
                seed=1234,
                deterministic=True, # force using deterministic algorithms
                overlap_mask=False) # whether object masks should be merged into
                                   # a single mask for training or kept separate

    print(f'looking for <best_ncnn_model>... ')
    best_ncnn = Path(project, name, 'weights', 'best_ncnn_model')
    if not best_ncnn.exists() or force_NCNN:
        print('\t exporting <best.pt> to <best_ncnn_model>...', end="")
        model = YOLO(best) # load the best custom trained model
        model.export(format="ncnn", imgsz=640, half=True, data=data_path[VER])
        print(" done.")

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('-v', '--version', action="store", dest='version',
                        required=True, help="dataset version: 1.0, 1.1...")
    parser.add_argument('--NCNN', action='store_true', dest='NCNN', default=False)
    args = parser.parse_args()
    version = f'v{args.version}'
    NCNN = args.NCNN

    main(version, force_NCNN=NCNN)

```

## eval\_YOLOv8.py

```
#####
#   Jean-Luc.Charles@mailo.com
#   2025/04/28 - v1.2
#####

from pathlib import Path
from ultralytics import YOLO
from time import sleep, strftime
import sys

def main(VER, timeStamp=''):

    BATCH = {'v1.3': (4, 8, 16, 32),
              'v1.4': (4, 8, 16, 20, 30, 40),
              'v1.5': (4, 8, 16, 20, 30, 40),
              'v1.6': (4, 8, 16, 20, 30, 40),
              'v1.7': (4, 8, 16, 20, 30, 40),
              'v1.8': (4, 8, 16, 20, 30, 40),
              }

    EPOCH = {'v1.3': (40, 80, 120, 160, 200),
              'v1.4': (80, 120, 160, 200, 240, 300),
              'v1.5': (80, 120, 160, 200, 240, 300, 400, 500),
              'v1.6': (80, 120, 160, 200, 240, 300),
              'v1.7': (80, 120, 160, 200, 240, 300),
              'v1.8': (80, 120, 160, 200, 240, 300),
              }

    data_path = {'v1.3': "./datasets/V1.1_yolo8_166train-24val-13test/data.yaml",
                  'v1.4': "./datasets/V1.0_yolo8_162train-27val-13test/data.yaml",
                  'v1.5': "./datasets/V1.1_yolo8_247train-85val-0test/data.yaml",
                  'v1.6': "./datasets/V1.0_yolo8_248train-84val-0test/data.yaml",
                  'v1.7': "./datasets/V1.0_yolo8_280train-72val-15test/data.yaml",
                  'v1.8': "./datasets/V1.0_yolo8_296train-67val-23test/data.yaml",
                  }

    yolo = 'YOLOv8n'

    header = '#meta-params\tpre[ms]\tinf[ms]\tloss[ms]\tpost[ms]\t'
    header += 'prec\trecall\tmAP50\tmAP50-95\tfitness\n'
    header += '#pre:preprocessing; inf:inference; post:postprocessing; prec:precision\n#\n'

    results_dir = Path('./Training/Results')

    if not results_dir.exists():
        results_dir.mkdir()

    file = f'results_yolov8n-{VER}'
    if timeStamp: file += f'_{timeStamp}'
    results_file = Path(results_dir, f'{file}.txt')

    F_out = open(results_file, "w", encoding="utf8")
    F_out.write(header)
```

```

for batch in BATCH[VER]:
    for epoch in EPOCH[VER]:
        project_dir = f'Training/YOLO-trained-{VER}/UCIA-II-{yolo}'
        name = f'batch-{batch:02d}_epo-{epoch:03d}'
        best = Path(project_dir, name, 'weights', 'best.pt')
        print(best)

    if best.exists():
        model = YOLO(best) # load the trained model
        # Validate the model
        metrics = model.val(batch=batch,
                            imgsz=640,
                            data=data_path[VER],
                            workers=0,
                            conf=0.3,
                            max_det=15,
                            save_crop=True,
                            plots=True,
                            name=f'{VER}_{name}')

        F_out.write(f'{name}')
        for key in metrics.speed:
            F_out.write(f'\t{metrics.speed[key]:.2f}')
        for key in metrics.results_dict:
            F_out.write(f'\t{metrics.results_dict[key]:.3f}')

        F_out.write('\n')

    del model

F_out.close()

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('-v', '--version', action="store", dest='version',
                        required=True, help="dataset version: 1.0, 1.1...")
    parser.add_argument('-t', '--timestamp', action='store_true', dest='time_stamp', default=False)

    args = parser.parse_args()
    version = f'v{args.version}'
    time_stamp = args.time_stamp

    timeStamp = f'{strftime("%Y-%m-%d_%H-%M-%S")}' if time_stamp else None

    main(version, timeStamp)

```

**process\_results.py**

```
#####
#   Jean-Luc.Charles@mailo.com
#   2025/04/28 - v1.3
#####

import pandas as pd
from pathlib import Path

def main(VER:str, timeStamp:str=None):

    results_dir = Path('./Training/Results')
    if timeStamp:
        out_file = Path(results_dir, f"processed_res-{VER}_{timeStamp}.txt")
    else:
        out_file = Path(results_dir, f"processed_res-{VER}.txt")

    print(f'{out_file=}')
    with open(out_file, "w", encoding="utf8") as stream_out:

        if timeStamp:
            txt_file = Path(results_dir, f'results_yolov8n-{VER}_{timeStamp}.txt')
        else:
            txt_file = Path(results_dir, f'results_yolov8n-{VER}.txt')

        mess = f'Input File <{txt_file}>\n'
        print(mess)
        stream_out.write(mess)

        #####
        mess = '\n' + 50*' ' + "\n* Sort by 'mAP50'\n" + 50*' '
        print(mess)
        stream_out.write(mess+'\n')

        # read CSV file with panda:
        df = pd.read_csv(txt_file, sep='\t', header=0, skiprows=[1])
        # now sort rows by descending order of column "fitnes":
        df = df.sort_values(by=["mAP50"], ascending=False)
        # the first values in column "fitness" is the max values
        max_mAP50 = df['mAP50'].values[0]

        mess = f'\tMax values -> "mAP50": {max_mAP50}'
        print(mess)
        stream_out.write(mess+'\n')

        # selected significant columns
        df1 = df[['#meta-params', 'recall', 'mAP50', 'mAP50-95', 'fitness']]

        # print the first 4 rows:
        mess = df1.head(4)
        print(mess)
        stream_out.write(str(mess)+'\n')

        #####
        mess = '\n' + 50*' ' + "\n* Sort by 'recall' & 'mAP50-95'\n" + 50*' '
        print(mess)
        stream_out.write(mess+'\n')

        # read CSV file with panda:
        df = pd.read_csv(txt_file, sep='\t', header=0, skiprows=[1])
        # sort rows by descending order of columns "recall", "mAP50-95":
        df = df.sort_values(by=["recall", "mAP50-95", ], ascending=False)
        # the first values in columns "recall" and "mAP50-95 are the max values:
        max_mAP50_90 = df['mAP50-95'].values[0]
        max_recall = df['recall'].values[0]

        mess = f'\tMax values -> "max_recall": {max_recall}, "max_mAP50-90":
{max_mAP50_90}'
        print(mess)
        stream_out.write(mess+'\n')

        # selected significant columns
        df2 = df[['#meta-params', 'recall', 'mAP50', 'mAP50-95', 'fitness']]

        # print the first 4 rows:
```

```
#####
mess = '\n' + 50*' ' + "\n* Sort by 'fitness'\n" + 50*' '
print(mess)
stream_out.write(mess+'\n')

# read CSV file with panda:
df = pd.read_csv(txt_file, sep='\t', header=0, skiprows=[1])
# now sort rows by descending order of column "fitnes":
df = df.sort_values(by=["fitness"], ascending=False)
# the first values in column "fitness" is the max values
max_fitness = df['fitness'].values[0]

mess = f'\tMax values -> "fitness": {max_fitness}'
print(mess)
stream_out.write(mess+'\n')

# selected significant columns
df3 = df[['#meta-params', 'recall', 'mAP50', 'mAP50-95', 'fitness']]

# print the first 4 rows:
mess = df3.head(4)
print(mess)
stream_out.write(str(mess)+'\n')

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('-v', '--version', action="store", dest='version',
                        required=True, help="dataset version: 1.0, 1.1...")
    parser.add_argument('-t', '--timestamp', action="store", dest='timestamp',
                        required=False, help="time stamp to select an input data file")

    args = parser.parse_args()
    version = f'v{args.version}'
    time_stamp = args.timestamp

    print(f'{args=}')
    main(version, timeStamp=time_stamp)
```

## 7.4 Fichiers résultats

**results\_yolov8n\_v1.8.txt** (obtenu sur le PC de calcul)

```
#meta-params      pre[ms]      inf[ms]      loss[ms]      post[ms]  prec  recall
      mAP50 mAP50-95      fitness
#pre:preprocessing; inf:inference; post:postprocessing; prec:precision
#
batch-04_epo-080  0.31  2.64  0.00  1.82  0.861 0.888 0.887 0.727 0.743
batch-04_epo-120  0.26  2.11  0.00  0.72  0.881 0.860 0.880 0.730 0.745
batch-04_epo-160  0.28  2.29  0.00  0.56  0.881 0.886 0.892 0.741 0.756
batch-04_epo-200  0.26  2.25  0.00  0.58  0.876 0.884 0.892 0.743 0.758
batch-04_epo-240  0.26  2.04  0.00  0.75  0.858 0.863 0.879 0.712 0.729
batch-04_epo-300  0.26  2.23  0.00  0.76  0.874 0.870 0.883 0.736 0.751
batch-08_epo-080  0.31  1.77  0.03  1.48  0.884 0.852 0.884 0.740 0.754
batch-08_epo-120  0.31  1.79  0.00  1.21  0.858 0.864 0.883 0.727 0.742
batch-08_epo-160  0.30  1.78  0.00  1.30  0.879 0.859 0.885 0.748 0.762
batch-08_epo-200  0.28  1.77  0.00  1.37  0.910 0.862 0.887 0.742 0.756
batch-08_epo-240  0.30  1.79  0.01  1.56  0.860 0.861 0.883 0.742 0.756
batch-08_epo-300  0.29  1.79  0.00  1.00  0.875 0.849 0.876 0.733 0.748
batch-16_epo-080  0.31  1.71  0.01  2.40  0.892 0.884 0.893 0.741 0.756
batch-16_epo-120  0.34  1.61  0.01  2.37  0.873 0.869 0.882 0.749 0.762
batch-16_epo-160  0.34  1.62  0.00  2.40  0.876 0.859 0.886 0.739 0.754
batch-16_epo-200  0.30  1.60  0.02  2.56  0.888 0.884 0.890 0.747 0.761
batch-16_epo-240  0.33  1.70  0.00  2.23  0.894 0.860 0.885 0.756 0.769
batch-16_epo-300  0.30  1.69  0.00  2.43  0.884 0.862 0.889 0.746 0.760
batch-20_epo-080  0.69  2.11  0.00  2.54  0.886 0.871 0.892 0.751 0.765
batch-20_epo-120  0.65  1.96  0.00  1.78  0.889 0.882 0.897 0.751 0.765
batch-20_epo-160  0.69  1.58  0.01  1.93  0.904 0.868 0.900 0.767 0.780
batch-20_epo-200  0.65  1.70  0.00  2.57  0.900 0.884 0.898 0.746 0.761
batch-20_epo-240  0.78  1.71  0.00  1.75  0.888 0.886 0.896 0.750 0.765
batch-20_epo-300  0.68  1.62  0.00  2.16  0.880 0.868 0.887 0.746 0.761
batch-30_epo-080  0.92  2.26  0.00  0.83  0.903 0.870 0.888 0.742 0.756
batch-30_epo-120  0.85  2.11  0.02  1.09  0.899 0.848 0.883 0.741 0.755
batch-30_epo-160  0.94  2.20  0.00  0.96  0.899 0.860 0.885 0.749 0.762
batch-30_epo-200  0.86  1.59  0.00  1.23  0.860 0.822 0.860 0.736 0.748
batch-30_epo-240  0.91  2.19  0.00  0.97  0.935 0.946 0.955 0.815 0.829
batch-30_epo-300  1.00  2.14  0.00  0.97  0.936 0.948 0.962 0.822 0.836
batch-40_epo-080  0.38  1.65  0.00  0.27  0.903 0.865 0.883 0.737 0.752
batch-40_epo-120  0.54  1.59  0.00  0.27  0.886 0.863 0.885 0.749 0.762
batch-40_epo-160  0.52  1.54  0.00  0.27  0.879 0.859 0.879 0.726 0.742
batch-40_epo-200  0.51  1.58  0.00  0.28  0.884 0.882 0.894 0.757 0.771
batch-40_epo-240  0.53  1.58  0.00  0.28  0.865 0.879 0.888 0.745 0.760
batch-40_epo-300  0.39  1.58  0.00  0.27  0.941 0.940 0.962 0.825 0.838
```

## results\_yolov8n\_v1.8.txt (obtenu sur la carte RPi4)

#meta-params	pre[ms]	inf[ms]	loss[ms]	post[ms]	prec	recall
mAP50 mAP50-95 fitness						
#pre:preprocessing; inf:inference; post:postprocessing; prec:precision						
batch-04_epo-080	10.14	413.20	0.00	2.69	0.867	0.881
batch-04_epo-120	5.70	424.16	0.00	2.47	0.874	0.885
batch-04_epo-160	5.89	425.22	0.00	2.65	0.869	0.873
batch-04_epo-200	5.63	406.04	0.00	2.86	0.892	0.864
batch-04_epo-240	5.88	429.48	0.00	2.74	0.869	0.858
batch-04_epo-300	5.79	407.63	0.00	2.74	0.876	0.859
batch-08_epo-080	5.35	406.59	0.00	2.84	0.865	0.841
batch-08_epo-120	5.95	424.77	0.00	2.62	0.886	0.845
batch-08_epo-160	5.44	423.44	0.00	2.83	0.877	0.865
batch-08_epo-200	5.38	418.56	0.00	2.97	0.900	0.866
batch-08_epo-240	5.54	422.85	0.00	2.68	0.877	0.848
batch-08_epo-300	5.52	409.17	0.00	2.69	0.873	0.839
batch-16_epo-080	5.88	428.55	0.00	2.82	0.866	0.882
batch-16_epo-120	5.63	408.64	0.00	2.75	0.867	0.867
batch-16_epo-160	5.41	406.72	0.00	2.63	0.881	0.860
batch-16_epo-200	5.96	416.81	0.00	2.99	0.881	0.846
batch-16_epo-240	6.01	409.32	0.00	2.83	0.897	0.854
batch-16_epo-300	5.83	420.60	0.00	2.93	0.888	0.871
batch-20_epo-080	5.61	404.58	0.00	2.91	0.885	0.857
batch-20_epo-120	6.29	401.40	0.00	2.95	0.878	0.856
batch-20_epo-160	6.31	413.45	0.00	3.05	0.886	0.852
batch-20_epo-200	6.17	429.94	0.00	2.88	0.902	0.877
batch-20_epo-240	5.59	405.28	0.00	2.73	0.879	0.857
batch-20_epo-300	5.37	420.90	0.00	2.71	0.878	0.867
batch-30_epo-080	5.60	427.07	0.00	2.70	0.881	0.866
batch-30_epo-120	6.61	404.49	0.00	3.11	0.884	0.856
batch-30_epo-160	6.05	430.39	0.00	3.13	0.894	0.862
batch-30_epo-200	6.38	413.55	0.00	3.38	0.846	0.824
batch-30_epo-240	5.60	408.91	0.00	2.84	0.887	0.859
batch-30_epo-300	5.62	408.50	0.00	3.04	0.896	0.943
batch-40_epo-080	5.62	418.51	0.00	2.64	0.875	0.864
batch-40_epo-120	5.67	416.49	0.00	2.86	0.879	0.851
batch-40_epo-160	5.53	407.39	0.00	2.84	0.866	0.826
batch-40_epo-200	6.04	419.85	0.00	3.10	0.875	0.854
batch-40_epo-240	5.74	406.13	0.00	3.03	0.834	0.881
batch-40_epo-300	6.21	417.11	0.00	3.11	0.881	0.847



## 8 Références

UCIA Cahier des charges : Robot ROSA avec Intelligence Artificielle OpenSource et OpenHardware

Page du site roboflow pour l'accès public au jeu de données de l'étude :

<https://app.roboflow.com/ucia/ucia-ia-ii/14>

Article Wikipédia sur les indicateurs de précision :

[https://fr.wikipedia.org/wiki/Pr%C3%A9cision\\_et\\_rappel](https://fr.wikipedia.org/wiki/Pr%C3%A9cision_et_rappel)

« Analyse approfondie des mesures de performance », site Ultralytics

<https://docs.ultralytics.com/fr/guides/yolo-performance-metrics/>

« The Complete Guide to Object Detection Evaluation Metrics: From IoU to mAP and More »

<https://medium.com/@prathameshamrutkar3/the-complete-guide-to-object-detection-evaluation-metrics-from-iou-to-map-and-more-1a23c0ea3c9d>