Rapport d’étude

« Entraînement d’un réseau de neurone YOLO pour la détection de petits objets 3D dans des images »

Exploitation sur carte Raspberry Pi 4

Jean-Luc CHARLES
Consultant IA/Data processing

version 1.0 du 2 décembre 2024

HISTORIQUE DES MODIFICATIONS

Édition	Révision	Date	Modification	Visa
1	0	2024-12-02	Version initiale	

Table des matières

Index des figures

Index des tableaux

1 Contexte de l'étude

Depuis janvier 2023 l'association « la ligue de l'enseignement » coordonne le 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 développer essentiellement le **jalon Niveau 0 pour le mode chasseur trésor** (page 8 à 11).

1.1 Objectifs de l'étude

Compte tenu des besoins fonctionnels exprimés dans le cahier des charges UCIA pour le « jalon Niveau 0 pour le mode chasseur trésor » mentionné ci-dessus, les objectifs de l'étude sont :

1. Développement d'une procédure d'entraînement d'un réseau de la famille YOLO
2. L'exploitation sur carte RPi4 munie d'une caméra RPi2, d'un réseau de neurones de la famille YOLO entraîné à détecter trois types d'objets (sphère, cube, étoile) situés en face du robot portant la caméra, dans un rayon de 5 à 45 cm.

2 Préparation du jeu de données

2.1 Dispositif de prise d'images

Pour favoriser la qualité de l'entraînement du réseau de neurones, les images des objets 3D sont réalisées avec le même environnement que celui qui utilisé à l'exploitation : camera Raspberry Pi V2 connectée à la carte RPi4, comme illustré par la figure 1 :

Figure 1: Carte RPi4 et la "Camera V2" connectée à la carte.

Le programme Python **take_image.py** (code source en annexe et dans les livrables RPi4) développé spécifiquement permet de numéroter automatiquement les images des objets pris en photo par la caméra au **objets3D-nnn.jpg**. On regroupe un maximum d'objets dans chaque image pour obtenir un nombre d'objets suffisant : avec 10 objets par images, une cinquantaine d'images donne 500 objets, ce qui constitue une bonne base d'entraînement :

Figure 2: Configuration des prises d'image des objets 3D.

Les exemples de la figure 3 illustrent le type d'images réalisées :

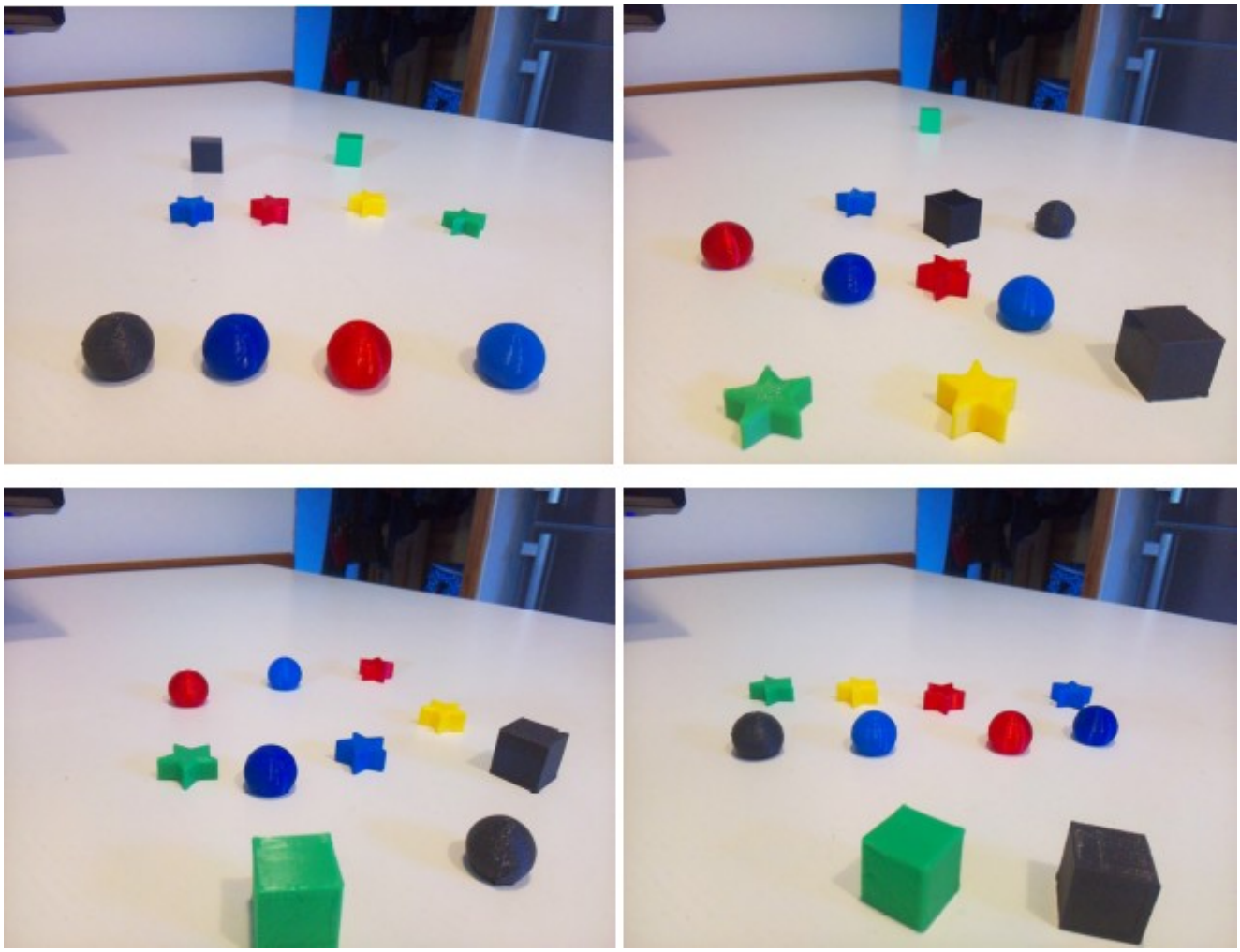

Figure 3: Exemples d'images des objets 3D prises avec la caméra Rasperry Pi V2.

Au total nous avons réalisé 62 images différentes avec les 10 objets disponibles. Les images sont téléchargées vers le site Roboflow pour réaliser l'annotation manuelle.

2.2 Annotation des images sur le site Roboflow

Le site Roboflow propose une interface facile à utiliser dans un navigateur web pour réaliser la tâche d'annotation des images (voir figure 4).

Les images chargées sur le site sont ensuite annotées à la main une par une. Pour chacun des 10 objets contenus 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 classes : star, cube, ball.

Le travail d'annotation est une étape très importante 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.

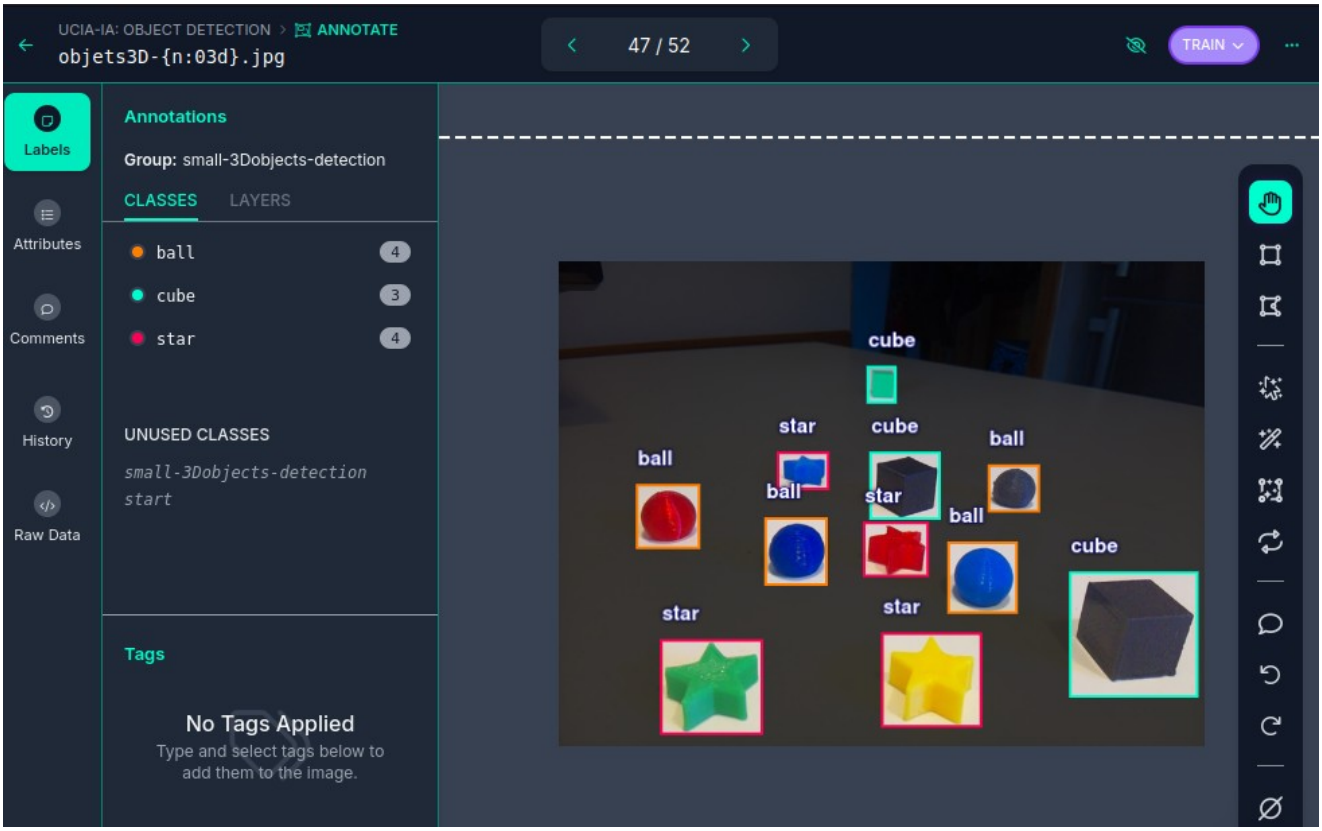

Figure 4: L'interface web du site Roboflow pour annoter les images.

Quand toutes images sont annotées, on peut créer sur le site Roboflow des jeux de données (*datasets*), en répartissant les 62 images annotées en plusieurs jeu de données :

- ***Train set*** : le jeu d'images qui va servir à l'entraînement du réseau de neurones.
- ***Validation set*** : le jeu d'images qui permet d'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 set*** : un jeu d'images qu'on peut former pour mesurer les performances du réseau entraîné, indépendamment des jeux d'entraînement et de validation. Dans le cadre de l'étude on choisit de privilégier les jeux d'entraînement et de validation : des images de test supplémentaires seront crées lors de la mesure sur carte RPi4 des performances du réseau entraîné.

Avec les 62 images crées, on choisit de mettre :

- 52 images pour l'entraînement, soit 520 objets.
- 10 images (100 objets) pour les validations utilisées pendant l'entraînement du réseau.

La figure 5 montre les jeux de données créés sur le site Roboflow.

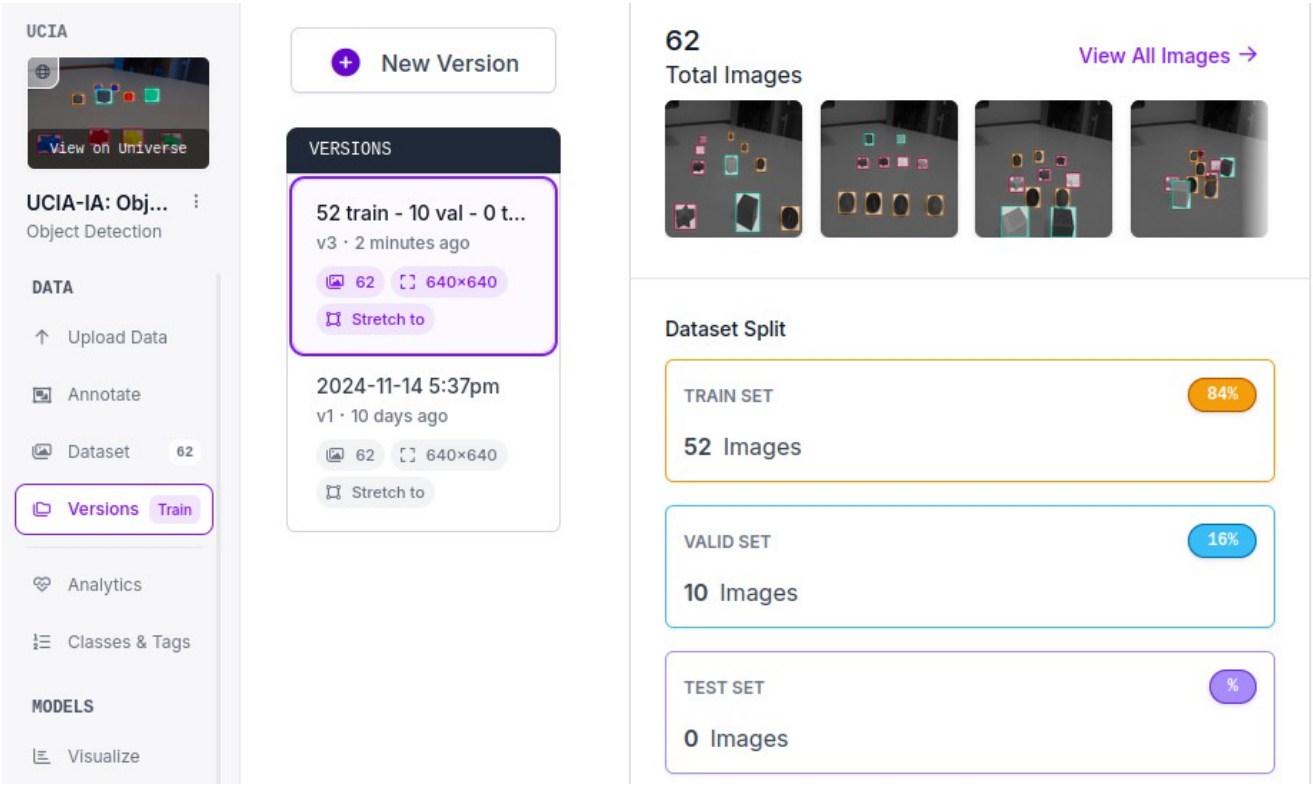

Figure 5: Les jeux de données créés sur Roboflow.

Une fois les jeux de données constitués sur le site Roboflow, ils sont téléchargés sur le PC d'entraînement aux formats **yolov8** et **yolo11** pour réaliser l'entraînement des réseaux de neurones correspondants.

3 Entraînement du réseau de neurones YOLO

Le réseau de neurones YOLO (*You Only Look Once*) est un modèle populaire de détection d'objets et de segmentation d'images. Il a été développé par Joseph Redmon et Ali Farhadi à l'Université de Washington. 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 YOLO11.

Les différentes versions du réseau YOLO sont gérées sur le site WEB Ultralytics :

https://docs.ultralytics.com/fr

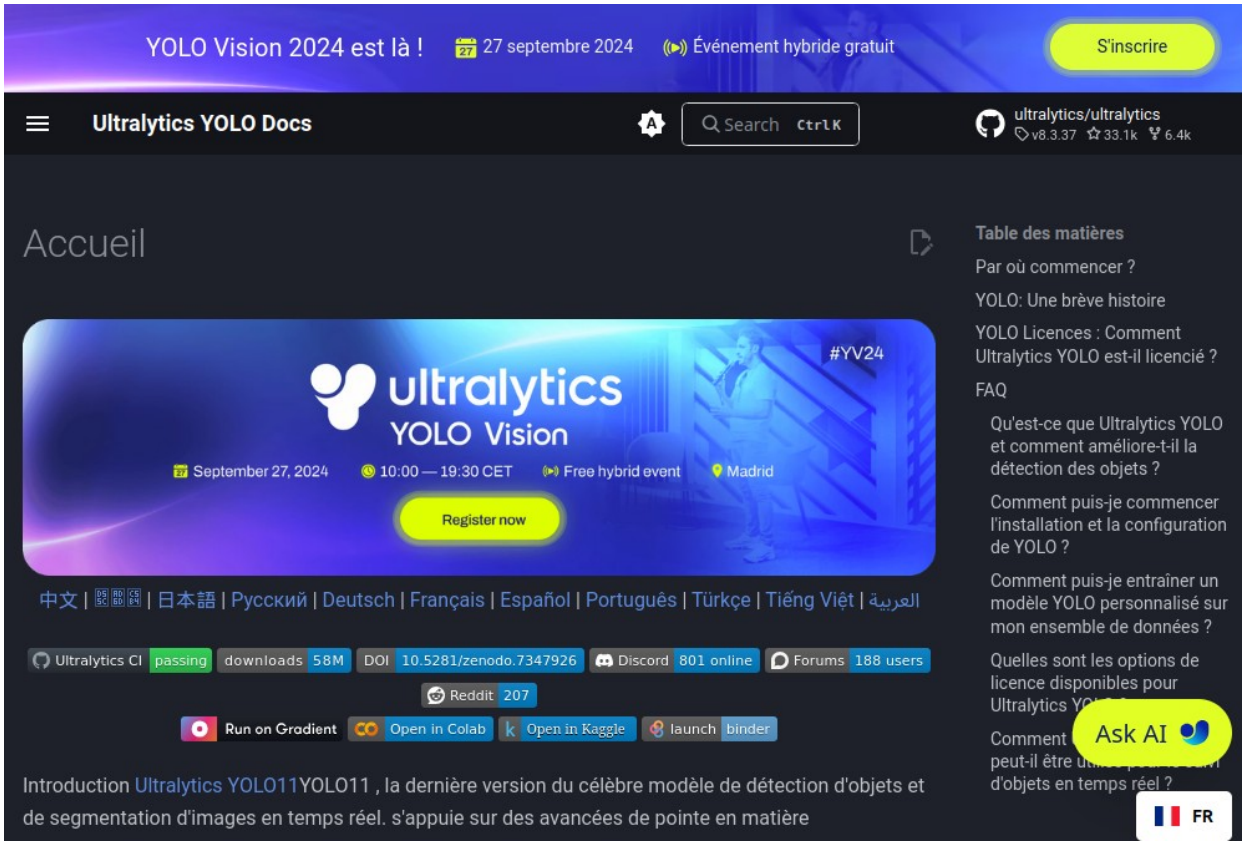

Figure 6: Le réseau de neurones YOLO sur le site web Ultralytics .

Extraits de la page d'accueil du site Ultralytics :

Qu'est-ce que Ultralytics YOLO et comment améliore-t-il la détection des objets ?

Ultralytics YOLO est la dernière avancée de la célèbre série YOLO (You Only Look Once) pour la détection d'objets et la segmentation d'images en temps réel. Elle s'appuie sur les versions précédentes en introduisant de nouvelles fonctionnalités et des améliorations pour accroître les performances, la flexibilité et l'efficacité. YOLO prend en charge diverses tâches d'IA visionnaire telles que la détection, la segmentation, l'estimation de la pose, le suivi et la classification. Son architecture de pointe garantit une vitesse et une précision supérieures, ce qui la rend adaptée à diverses applications, y compris les appareils périphériques et les API en nuage.

3.1 Choix des versions du réseau de neurones

Les versions 8 et 11 du réseau YOLO sont proposées en 4 architectures de complexité croissante :

- YOLO...n → nano pour les tâches petites et légères.
- YOLO...s → small mise à niveau de nano, meilleure précision.
- YOLO...m → medium pour une utilisation à usage général.
- YOLO...l → large meilleure précision au prix d'un calcul plus lourd.
- YOLO...x → Extra-large pour une précision et des performances maximales.

Le figure 7[1] détaille quelques caractéristiques des 5 architectures du réseau YOLO11 :

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	params (M)	FLOPs (B)
YOLO11n	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
YOLO11s	640	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
YOLO11m	640	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
YOLO11l	640	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9
YOLO11x	640	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9	194.9

Figure 7: Différentes versions du réseau YOLO11.

Vu la simplicité des objets à détecter, on choisit de se limiter aux architecture (n) et (s) susceptibles de fournir un bon score en un temps raisonnable sur une RPi4. Les architecture plus complexes (m), (l) et (x) risquent de pénaliser sensiblement le temps d'inférence, sans apporter d'amélioration notoire dans la précision de la reconnaissance des objets.

Au final les réseaux candidats retenus pour l'étude sont :

- **yolov8n**, **yolov8s**, **yolo11n** et **yolo11s**.

3.2 Création de l'Environnement Virtuel Python (EVP)

L'état de l'art pour entraîner un réseau de neurones avec les modules Python consiste à créer un **Environnement Virtuel Python** (EVP), au sein duquel les modules Python sont chargés et les programmes d'entraînement sont développés et exploités.

L'EVP ucia est créé avec le module Python **venv** sur le PC d'entraînement :

```
python3 -m venv .vision
```

L'EVP ucia est ensuite activé puis les modules Python nécessaires à l'entraînement des réseaux sont ajoutés dans l'EVP :

```
source .vision activate
pip install ultralytics, onnx, onnxruntime, pathlib
```

1 Source : https://learnopencv.com/yolo11/

Le module **ultralytics** utilise le mécanisme de dépendance des modules Python pour charger à son tour tous les autres modules Python nécessaires au *machine learning* (**tensorflow**, **torch**, **numpy**, **scipy**, **matplotib**…).

3.3 Choix des méta-paramètres d'entraînement

L'entraînement supervisé d'un réseau de neurones est un processus complexe grandement facilité par l'utilisation de modules Python comme **torch** ou **tensorflow**, encapsulés dans le module **ultralytics**. De nombreux paramètres influent sur l'entraînement, sa vitesse, la qualité du réseau entraîné obtenu….

La page modes/train/#resuming-interrupted-trainings du site Ultralytics liste ces paramètres.

Pour les besoins de l'étude, nous retenons les méta-paramètres et les plages de valeurs présentés sur le tableau 1 :

Tableau 1: Tableau des plages de valeurs des méta-paramètres d'entraînement.

Paramètre	Description	Plage de valeurs
epochs	Nombrer de répétitions du processus complet d'entraînement pour converger vers le meilleur état de réseau entraîné	20, 40, 60, 80
batch	Nombre d'images fournies dans un lot d'images	8, 19, 16, 20, 30

Le tableau 2 donne les valeurs des autres paramètres utilisés pour les entraînements :

Tableau 2: Tableau des paramètres d'entraînement.

Paramètre	Description	Valeur
imgz	Taille des image (en pixels)	640
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.	100
pretrained	Détermine s'il faut utiliser un modèle pré-entraîné.	True
seed	Définit la graine aléatoire pour l'entraînement pour garantir la reproductibilité des résultats d'une exécution à l'autre avec les mêmes configurations.	1234
workers	Nombre de threads de travail pour le chargement des données.	0

Nota : des valeurs du paramètre workers autre que 0 conduisent à des anomalies de l'entraînement sur le PC d'entraînement.

4 Entraînement des réseaux YOLO, résultats.

4.1 Environnement de calcul

La combinaison des cas d'entraînement est la suivante :

- 4 réseaux : **yolov8n**, **yolov8s**, **yolov11n** et **yolo11s**,
- 7 valeurs du paramètre batch (2, 4, 8, 10, 16, 20 et 30),
- 5 valeur du paramètre epochs (20, 40, 60, 80, 100)

ce qui donne 4 x 7 = 28 entraînements, répétés 20 + 40 + 60 + 80 +100 = 300 fois, soit au total 8400 calculs d'entraînement. Sur un PC portable core i7 un entraînement prend en moyenne ~ 7 secondes (fonction de la valeur des méta-paramètres), ce qui donnerait plus de 16 h de calculs pour traiter tous les cas retenus. Nous avons utilisé un PC Ubuntu avec carte graphique Nvidia permettant de réduire les temps de calcul à environ 2 h.

L'arborescence du développement sur le PC de calcul est représentée sur la figure 8 :

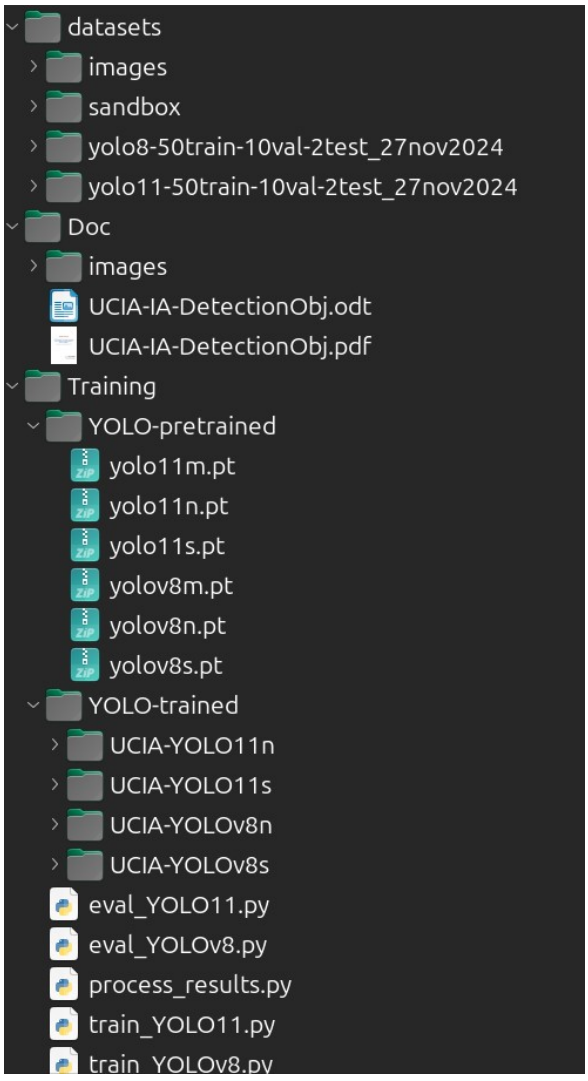

Figure 8: Arborescence du projet.

- Dossier **datasets** :
  contient les jeux d'images annotées, téléchargés depuis le site Roboflow, comme expliqué chapitre 2.2 page 7.
- Dossier **Training** :
  contient les fichiers Python pour l'entraînement et le test des réseaux entraînés, aisni que les principaux sous-dossiers :
  - Dossier **YOLO_pretrained** :
    contient les fichiers binaires au format **pytorch** des poids des réseaux YOLO pré-entraînés à la détection d'objets avec le jeu de données coco.
  - Dossier **YOLO-trained** :
    contient l'arborescence des poids des réseaux **yolov8vn**, **yolov8vn** et **yolo11s**, **yolo11s** entraînés avec les valeurs des méta-paramètres du tableau 1.

Le nommage du dossier des résultats pour chacune des combinaisons d'entraînement suit la règle de nommage :

Training/YOLO-trained/UCIA-YOLOvvv/batch-BB_epo-EEE

avec :

- **vvv** : la version du réseau YOLO parmi (**v8n**, **v8s**, **11n**, **11s**)
- **BB** : la valeur du méta-paramètre batchzize, parmi (**08**, **10, 16**, **20**, **30**)
- **EEE** : la valeur du méta-paramètre epochs, parmi (**020**, **040**, **060**, **080**, **100**).

Exemple :
Training/YOLO-trained/UCIA-YOLO11n/batch-08_epo-020 correspond à l'entraînement d'un réseau **yolo11n** avec **batch=8** et **epochs=20**.

Les entraînements sont réalisés avec les 2 fichiers Python **train_YOLOv8.py** et **train_YOLO11.py** développés spécifiquement pour cette étude, dont le code source est donné en annexe et fait partie des livrables de l'étude.

À la fin de chaque entraînement, trois fichiers des poids du réseau entraîné sont écrits dans le dossier :

- **best.pt** : les poids du réseau entraîné, au format binaire du module **pytorch**,
- **best.onnx** : les poids du réseau entraîné exportés au format ONNX[2] optimisé pour RPi4
- **best_ncnn_model/best.ncnn** : les poids du réseau entraîné exportés au format NCNN[3] optimisé pour RPi4.

4.2 Résultats des entraînements

Parmi les 100 combinaisons des réseaux YOLO et des méta-paramètes on cherche celles présentant un rapport « performances / rapidité de calcul » favorable à une bonne exécution sur la carte RPi4.

Le module **ultralytics** donne la possibilité d'évaluer les principales métriques de performance de détection des réseaux entraînés. Les programmes Python **eval_TOLOv8.py** et **eval_YOLO11.py** développés spécifiquement pour cette étude mettent en œuvre ces possibilités et nous permettent de comparer les 100 versions d'entraînement des réseaux YOLO.

Le détail des résultats est donné par les quatre fichiers **results_yolov8n.txt**, **results_yolov8s.txt**, **results_yolo11n.txt** et **results_yolo11s.txt** qui figurent sur les pages 15 et 17.

L'analye des temps d'inférence et de la précision des réseaux entraînés et les conclusion sont présentées page 19.

2 Format ONNX : https://docs.ultralytics.com/fr/integrations/onnx/
3 Format NCNN : https://docs.ultralytics.com/fr/integrations/ncnn/

Fichier **results_yolov8n.txt** :

#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-02_epo-020	0.27	2.92	0.00	3.16	0.993	0.987	0.995	0.823	0.840
batch-02_epo-040	0.27	2.91	0.00	2.18	0.997	0.976	0.990	0.841	0.856
batch-02_epo-060	0.26	3.13	0.00	2.57	0.995	0.993	0.995	0.840	0.856
batch-02_epo-080	0.26	3.14	0.00	2.46	0.994	0.999	0.995	0.840	0.855
batch-02_epo-100	0.26	3.15	0.00	2.53	0.995	0.999	0.995	0.838	0.854
batch-04_epo-020	0.56	2.92	0.00	2.55	0.987	0.994	0.994	0.818	0.835
batch-04_epo-040	0.28	2.01	0.00	2.04	0.994	1.000	0.995	0.830	0.847
batch-04_epo-060	0.27	2.00	0.00	1.86	0.992	0.998	0.995	0.839	0.854
batch-04_epo-080	0.29	2.31	0.00	1.70	0.996	0.999	0.995	0.838	0.854
batch-04_epo-100	0.27	2.00	0.00	2.06	0.996	1.000	0.995	0.839	0.855
batch-08_epo-020	1.17	1.87	0.00	0.89	0.993	0.997	0.995	0.825	0.842
batch-08_epo-040	1.19	1.87	0.00	0.86	0.986	0.984	0.993	0.833	0.849
batch-08_epo-060	0.61	1.87	0.00	0.69	0.995	0.989	0.995	0.843	0.858
batch-08_epo-080	1.18	1.87	0.00	0.59	0.997	1.000	0.995	0.844	0.859
batch-08_epo-100	1.17	1.87	0.00	0.57	0.996	1.000	0.995	0.839	0.855
batch-10_epo-020	0.22	1.75	0.00	0.69	0.990	0.981	0.995	0.816	0.834
batch-10_epo-040	0.22	1.68	0.00	0.69	0.989	0.990	0.995	0.832	0.848
batch-10_epo-060	0.22	1.69	0.00	0.70	0.997	0.989	0.995	0.834	0.850
batch-10_epo-080	0.22	1.68	0.00	0.57	0.997	0.989	0.995	0.845	0.860
batch-10_epo-100	0.22	1.68	0.00	0.54	0.997	1.000	0.995	0.837	0.853
batch-16_epo-020	0.22	1.68	0.00	0.68	0.968	0.735	0.962	0.786	0.804
batch-16_epo-040	0.22	1.68	0.00	0.66	0.990	0.988	0.995	0.836	0.852
batch-16_epo-060	0.22	1.69	0.00	0.55	0.990	0.993	0.995	0.838	0.854
batch-16_epo-080	0.22	1.69	0.00	0.53	0.997	0.987	0.995	0.835	0.851
batch-16_epo-100	0.22	1.68	0.00	0.52	0.997	0.988	0.995	0.835	0.851
batch-20_epo-020	0.22	1.69	0.00	0.69	1.000	0.356	0.874	0.732	0.746
batch-20_epo-040	0.22	1.68	0.00	0.68	0.991	0.988	0.995	0.832	0.848
batch-20_epo-060	0.22	1.68	0.00	0.52	0.996	0.980	0.995	0.834	0.850
batch-20_epo-080	0.22	1.68	0.00	0.51	0.998	0.990	0.995	0.844	0.859
batch-20_epo-100	0.22	1.68	0.00	0.50	0.995	0.998	0.995	0.831	0.848
batch-30_epo-020	0.22	1.68	0.00	0.64	1.000	0.107	0.840	0.686	0.701
batch-30_epo-040	0.22	1.68	0.00	0.55	1.000	0.752	0.989	0.826	0.842
batch-30_epo-060	0.22	1.68	0.00	0.54	0.986	0.994	0.995	0.825	0.842
batch-30_epo-080	0.22	1.69	0.00	0.50	0.985	0.991	0.995	0.838	0.853
batch-30_epo-100	0.22	1.68	0.00	0.48	0.998	0.994	0.995	0.833	0.850

*Tableau 3: Évaluation des entraînements du réseau **yolov8n**.*

Fichier **results_yolov8s.txt** :

#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-02_epo-020	0.26	4.93	0.00	5.00	0.946	0.947	0.977	0.776	0.796
batch-02_epo-040	0.26	4.96	0.00	2.06	0.978	0.946	0.983	0.800	0.818
batch-02_epo-060	0.26	4.91	0.00	1.91	0.979	0.982	0.992	0.821	0.838
batch-02_epo-080	0.26	5.01	0.00	1.97	0.995	0.977	0.994	0.842	0.857
batch-02_epo-100	0.27	4.93	0.00	2.03	0.998	0.992	0.994	0.840	0.856
batch-04_epo-020	0.27	4.60	0.00	1.06	0.977	0.997	0.994	0.813	0.831
batch-04_epo-040	0.31	4.65	0.00	0.79	0.998	1.000	0.995	0.826	0.843
batch-04_epo-060	0.28	4.54	0.00	0.94	0.998	0.985	0.995	0.850	0.864
batch-04_epo-080	0.28	4.54	0.00	0.95	0.997	0.992	0.995	0.841	0.857
batch-04_epo-100	0.28	4.60	0.00	0.84	0.987	0.992	0.995	0.849	0.864
batch-08_epo-020	0.62	4.42	0.00	0.42	0.994	0.999	0.995	0.823	0.840
batch-08_epo-040	1.19	4.42	0.00	0.36	0.989	1.000	0.995	0.833	0.849
batch-08_epo-060	1.18	4.43	0.00	0.35	0.993	0.999	0.995	0.852	0.866
batch-08_epo-080	1.18	4.43	0.00	0.35	0.996	0.993	0.995	0.849	0.863
batch-08_epo-100	1.19	4.43	0.00	0.36	0.993	0.999	0.995	0.847	0.861
batch-10_epo-020	0.22	4.05	0.00	0.56	0.977	0.997	0.994	0.823	0.840
batch-10_epo-040	0.22	3.99	0.00	0.33	0.988	0.998	0.995	0.826	0.843
batch-10_epo-060	0.22	3.99	0.00	0.33	0.997	0.999	0.995	0.839	0.855
batch-10_epo-080	0.22	3.99	0.00	0.33	0.998	1.000	0.995	0.840	0.856
batch-10_epo-100	0.22	3.99	0.00	0.33	0.997	1.000	0.995	0.846	0.861
batch-16_epo-020	0.22	3.99	0.00	0.56	0.998	0.988	0.995	0.818	0.836
batch-16_epo-040	0.22	3.99	0.00	0.33	0.997	1.000	0.995	0.838	0.853
batch-16_epo-060	0.22	3.98	0.00	0.32	0.997	1.000	0.995	0.838	0.854
batch-16_epo-080	0.22	3.99	0.00	0.33	0.996	0.998	0.995	0.836	0.852
batch-16_epo-100	0.22	3.99	0.00	0.32	0.997	1.000	0.995	0.851	0.865
batch-20_epo-020	0.22	3.98	0.00	2.39	0.996	0.999	0.995	0.817	0.835
batch-20_epo-040	0.22	3.84	0.00	0.33	0.992	0.999	0.995	0.832	0.848
batch-20_epo-060	0.22	4.00	0.00	0.33	0.998	1.000	0.995	0.845	0.860
batch-20_epo-080	0.22	3.99	0.00	0.32	0.996	1.000	0.995	0.840	0.856
batch-20_epo-100	0.22	3.99	0.00	0.32	0.997	1.000	0.995	0.850	0.865
batch-30_epo-020	0.22	3.99	0.00	3.27	0.975	0.963	0.981	0.805	0.823
batch-30_epo-040	0.22	3.64	0.00	0.38	0.997	0.997	0.995	0.826	0.843
batch-30_epo-060	0.22	4.01	0.00	0.33	0.995	0.999	0.995	0.838	0.854
batch-30_epo-080	0.22	3.99	0.00	0.32	0.997	1.000	0.995	0.844	0.859
batch-30_epo-100	0.22	4.00	0.00	0.32	0.997	1.000	0.995	0.853	0.867

*Tableau 4: Évaluation des entraînement du réseau **yolov8s**.*

Fichier **results_yolo11n.txt**

#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-02_epo-020	0.27	5.41	0.00	2.31	0.997	0.959	0.992	0.815	0.833
batch-02_epo-040	0.27	4.88	0.00	1.99	0.982	0.981	0.983	0.827	0.843
batch-02_epo-060	0.27	4.85	0.00	1.91	0.987	0.983	0.993	0.821	0.838
batch-02_epo-080	0.32	5.93	0.00	1.38	0.990	0.983	0.994	0.839	0.855
batch-02_epo-100	0.27	5.74	0.00	1.02	0.997	0.996	0.995	0.834	0.850
batch-04_epo-020	0.27	4.34	0.00	1.87	0.985	0.984	0.985	0.801	0.820
batch-04_epo-040	0.27	3.65	0.00	1.29	0.988	0.992	0.995	0.831	0.847
batch-04_epo-060	0.29	3.57	0.00	1.48	0.997	0.990	0.995	0.836	0.852
batch-04_epo-080	0.43	3.73	0.00	1.60	0.994	0.991	0.995	0.830	0.847
batch-04_epo-100	0.27	3.44	0.00	0.97	0.995	0.999	0.995	0.840	0.855
batch-08_epo-020	1.18	1.95	0.00	0.81	0.980	0.973	0.983	0.808	0.825
batch-08_epo-040	1.19	1.96	0.00	0.90	0.989	0.976	0.988	0.821	0.838
batch-08_epo-060	1.20	1.95	0.00	0.83	0.989	0.993	0.993	0.832	0.848
batch-08_epo-080	1.18	1.95	0.00	0.67	0.979	0.998	0.995	0.833	0.849
batch-08_epo-100	1.15	1.96	0.00	0.52	0.989	0.981	0.994	0.841	0.856
batch-10_epo-020	0.22	2.05	0.00	0.76	0.977	0.926	0.979	0.806	0.824
batch-10_epo-040	0.22	1.68	0.00	0.74	0.984	0.976	0.982	0.812	0.829
batch-10_epo-060	0.22	1.68	0.00	1.02	0.978	0.998	0.995	0.841	0.857
batch-10_epo-080	0.22	1.68	0.00	0.64	0.997	0.980	0.994	0.835	0.851
batch-10_epo-100	0.22	1.68	0.00	0.57	0.997	0.992	0.995	0.832	0.848
batch-16_epo-020	0.22	1.68	0.00	0.97	1.000	0.456	0.941	0.776	0.793
batch-16_epo-040	0.22	1.68	0.00	0.65	0.974	0.980	0.980	0.825	0.840
batch-16_epo-060	0.22	1.68	0.00	0.93	0.986	0.992	0.994	0.834	0.850
batch-16_epo-080	0.22	1.68	0.00	0.61	0.991	0.992	0.995	0.845	0.860
batch-16_epo-100	0.22	1.68	0.00	0.52	0.994	0.989	0.995	0.833	0.850
batch-20_epo-020	0.22	1.69	0.00	0.91	1.000	0.249	0.722	0.576	0.591
batch-20_epo-040	0.22	1.68	0.00	0.89	0.991	0.946	0.980	0.820	0.836
batch-20_epo-060	0.22	1.68	0.00	0.63	0.988	0.993	0.995	0.847	0.862
batch-20_epo-080	0.22	1.68	0.00	0.61	0.998	0.982	0.995	0.835	0.851
batch-20_epo-100	0.22	1.68	0.00	0.47	0.997	0.990	0.995	0.836	0.852
batch-30_epo-020	0.22	1.68	0.00	1.41	0.032	0.883	0.637	0.507	0.520
batch-30_epo-040	0.22	1.68	0.00	0.64	1.000	0.286	0.957	0.790	0.807
batch-30_epo-060	0.22	1.68	0.00	0.59	0.990	0.914	0.984	0.824	0.840
batch-30_epo-080	0.22	1.68	0.00	0.56	0.977	0.991	0.994	0.831	0.847
batch-30_epo-100	0.22	1.69	0.00	0.47	0.985	0.997	0.994	0.831	0.847

*Tableau 5: Évaluation des entraînement du réseau **yolov11n**.*

Fichier **results_yolo11s.txt**

#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-02_epo-020	0.26	6.24	0.00	1.56	0.969	0.971	0.981	0.788	0.807
batch-02_epo-040	0.27	6.59	0.00	1.73	0.977	0.982	0.993	0.823	0.840
batch-02_epo-060	0.27	6.45	0.00	0.83	0.996	0.981	0.995	0.828	0.845
batch-02_epo-080	0.27	6.60	0.00	0.69	0.998	0.993	0.995	0.842	0.858
batch-02_epo-100	0.27	6.38	0.00	0.88	0.995	0.999	0.995	0.838	0.854
batch-04_epo-020	0.27	4.93	0.00	0.94	0.995	0.999	0.995	0.822	0.839
batch-04_epo-040	0.28	4.59	0.00	1.01	0.992	0.981	0.994	0.837	0.853
batch-04_epo-060	0.29	5.02	0.00	0.69	0.997	1.000	0.995	0.841	0.856
batch-04_epo-080	0.27	4.66	0.00	1.05	0.995	1.000	0.995	0.843	0.859
batch-04_epo-100	0.28	4.96	0.00	0.70	0.997	0.988	0.995	0.856	0.870
batch-08_epo-020	1.18	4.29	0.00	0.42	0.993	0.999	0.995	0.823	0.840
batch-08_epo-040	1.18	4.28	0.00	0.36	0.995	0.991	0.995	0.833	0.850
batch-08_epo-060	0.59	4.28	0.00	0.35	0.998	1.000	0.995	0.853	0.867
batch-08_epo-080	1.19	4.23	0.00	0.35	0.994	1.000	0.995	0.856	0.870
batch-08_epo-100	1.19	4.29	0.00	0.35	0.998	1.000	0.995	0.856	0.870
batch-10_epo-020	0.22	3.88	0.00	0.50	0.998	0.993	0.995	0.827	0.844
batch-10_epo-040	0.22	3.81	0.00	0.33	0.997	0.999	0.995	0.833	0.849
batch-10_epo-060	0.22	3.82	0.00	0.33	0.997	1.000	0.995	0.845	0.860
batch-10_epo-080	0.22	3.83	0.00	0.33	0.997	1.000	0.995	0.848	0.863
batch-10_epo-100	0.22	3.82	0.00	0.32	0.998	1.000	0.995	0.846	0.861
batch-16_epo-020	0.22	3.81	0.00	0.50	0.991	0.979	0.995	0.818	0.836
batch-16_epo-040	0.22	3.81	0.00	0.33	0.994	0.999	0.995	0.834	0.850
batch-16_epo-060	0.22	3.82	0.00	0.33	0.997	1.000	0.995	0.846	0.861
batch-16_epo-080	0.22	3.81	0.00	0.32	0.997	1.000	0.995	0.843	0.859
batch-16_epo-100	0.22	3.82	0.00	0.32	0.998	1.000	0.995	0.847	0.861
batch-20_epo-020	0.22	3.82	0.00	0.61	0.995	0.992	0.995	0.823	0.840
batch-20_epo-040	0.22	3.82	0.00	0.33	0.984	0.999	0.995	0.844	0.859
batch-20_epo-060	0.22	3.82	0.00	0.32	0.997	1.000	0.995	0.847	0.862
batch-20_epo-080	0.22	3.83	0.00	0.32	0.997	1.000	0.995	0.847	0.862
batch-20_epo-100	0.22	3.82	0.00	0.32	0.997	1.000	0.995	0.855	0.869
batch-30_epo-020	0.22	3.81	0.00	1.01	0.993	0.994	0.995	0.800	0.820
batch-30_epo-040	0.22	3.85	0.00	0.35	0.997	1.000	0.995	0.835	0.851
batch-30_epo-060	0.22	3.83	0.00	0.33	0.988	0.989	0.995	0.841	0.856
batch-30_epo-080	0.22	3.82	0.00	0.33	0.997	1.000	0.995	0.843	0.858
batch-30_epo-100	0.22	3.82	0.00	0.33	0.996	1.000	0.995	0.843	0.858

*Tableau 6: Évaluation des entraînement du réseau **yolo11s**.*

Temps d'inférence

Le tableau 7 synthétise les temps moyen d'inférence sur le PC d'entraînement des 4 réseaux testés :

*Tableau 7: Temps moyen d'inférence des réseaux **yolov8** et **yolo11**.*

Réseau	*Temps moyen d'inférence[ms] batch:8*	*Temps moyen d'inférence[ms] batch:10, 16, 20, 30*
yolov8n	1.9	1.7
yolo11n	1.9	1.7
yolov8s	4.4	4
yolo11s	4.4	3.8

On constate par ailleurs deux tendances :

- Les temps moyens d'inférence pour les entraînement avec batch=8 sont légèrement supérieurs à ceux obtenus pour les autres valeurs de batch (10, 16, 20 et 30).
- Il y a peu de différence globalement entre les temps d'inférence **yolov8n** et **yolo11n** d'une part , et **yolov8s** et **yolo11s** d'autre part.

Précision des réseaux entraînés

On s'attache ici à regarder les colonnes des métriques recall, mAP50-95 et fitness dans les quatre fichiers résultats **results_yolov8n.txt**, **results_yolov8s.txt**, **results_yolo11n.txt** et **results_yolo11s.txt**, qui sont les plus significatives pour quantifier les performances des réseaux entraînés à la détection d'objets.

Les tendances observées sont les suivantes :

- Les entraînements avec epoch=20 donnent souvent lieu à des valeurs faibles pour les trois métriques.
- Les entraînements avec epoch=80 et epoch=100 donnent souvent des valeurs élevées pour les trois métriques.

Pour synthétiser les résultats nous avons écrit le programme Python **process_results.py** (code source en annexe page 34 et dans les livrables) : on lit les 4 fichiers avec la fonction read_cvsv du module **pandas**, puis on trie les *dataframes* obtenus par ordre décroissant des colonnes recall et mAP50-95, puis par ordre décroissant de la colonne fitness .

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

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

```
File <results_yolov8n.txt>
        Max values -> "max_recall": 1.0, "max_mAP50-90": 0.844
        #meta-params  recall  mAP50-95  fitness
batch-08_epo-080     1.0     0.844    0.859
batch-04_epo-100     1.0     0.839    0.855
batch-08_epo-100     1.0     0.839    0.855
batch-10_epo-100     1.0     0.837    0.853

File <results_yolo11n.txt>
        Max values -> "max_recall": 0.999, "max_mAP50-90": 0.84
        #meta-params  recall  mAP50-95  fitness
batch-04_epo-100   0.999     0.840    0.855
batch-10_epo-060   0.998     0.841    0.857
batch-08_epo-080   0.998     0.833    0.849
batch-30_epo-100   0.997     0.831    0.847

File <results_yolov8s.txt>
        Max values -> "max_recall": 1.0, "max_mAP50-90": 0.853
        #meta-params  recall  mAP50-95  fitness
batch-30_epo-100     1.0     0.853    0.867
batch-16_epo-100     1.0     0.851    0.865
batch-20_epo-100     1.0     0.850    0.865
batch-10_epo-100     1.0     0.846    0.861

File <results_yolo11s.txt>
        Max values -> "max_recall": 1.0, "max_mAP50-90": 0.856
        #meta-params  recall  mAP50-95  fitness
batch-08_epo-080     1.0     0.856    0.870
batch-08_epo-100     1.0     0.856    0.870
batch-20_epo-100     1.0     0.855    0.869
batch-08_epo-060     1.0     0.853    0.867
```

En triant avec la colonne fitness, on obtient :

```
File <results_yolov8n.txt>
        Max values -> "fitness": 0.86
        #meta-params  recall  mAP50-95  fitness
batch-10_epo-080   0.989     0.845    0.860
batch-08_epo-080   1.000     0.844    0.859
batch-20_epo-080   0.990     0.844    0.859
batch-08_epo-060   0.989     0.843    0.858

File <results_yolo11n.txt>
        Max values -> "fitness": 0.862
        #meta-params  recall  mAP50-95  fitness
batch-20_epo-060   0.993     0.847    0.862
batch-16_epo-080   0.992     0.845    0.860
batch-10_epo-060   0.998     0.841    0.857
batch-08_epo-100   0.981     0.841    0.856

File <results_yolov8s.txt>
        Max values -> "fitness": 0.867
        #meta-params  recall  mAP50-95  fitness
batch-30_epo-100   1.000     0.853    0.867
batch-08_epo-060   0.999     0.852    0.866
batch-16_epo-100   1.000     0.851    0.865
batch-20_epo-100   1.000     0.850    0.865

File <results_yolo11s.txt>
        Max values -> "fitness": 0.87
        #meta-params  recall  mAP50-95  fitness
batch-08_epo-080   1.000     0.856    0.870
batch-08_epo-100   1.000     0.856    0.870
batch-04_epo-100   0.988     0.856    0.870
batch-20_epo-100   1.000     0.855    0.869
```

On en déduit les configurations d'entraînement qui donnent les meilleurs résultats sur les images de validation :

- **yolov8n** : **batch-08_epo-080** ou **batch-10_epo-080**
- **yolo11n** : **batch-04_epo-100** ou **batch-20_epo-060**
- **yolov8s** : **batch-30_epo-100**
- **yolo11s** : **batch-08_epo-080**

4.3 Conclusion

Il faut garder présent à l'esprit que ces résultats sont obtenus avec le jeu d'images de validation, et que en vraie situation d'utilisation 'RPi4 montée sur le robot mobile, couleur de la table, éclairage…) il peut s'avérer que ce soit d'autres combinaisons d'entraînement qui donnent des performances optimales.

Au vu des résultats obtenus précédemment, nous pouvons tester sur RPi4 les réseaux **yolov8n** et **yolo11n** (temps d'inférence les plus faibles) dans les configurations les plus performantes :

- **yolov8n : batch-08_epo-080 ou batch-10_epo-080**
- **yolo11n : batch-04_epo-100 ou batch-20_epo-060**

5 Exploitation sur RPi4 des réseaux YOLO entraînés

5.1 Préparation de la carte SD pour la RPi4

Il est important d'utiliser une carte SD rapide pour installer le système d'exploitation « Rasperry PI OS (64bits)» de la RPi4 afin d'optimiser ses performances. Nous avons utilisé une carte micro SD 64 GB de la marque TEAM, de classe A1.

Le système d'exploitation est installé sur la carte grâce au logiciel « Raspberry PI imager[4] » qui propose une interface graphique efficace pour flasher des cartes micro SD. La version du système d'exploitation installée est datée du 2024-11-19 (cf figure 9).

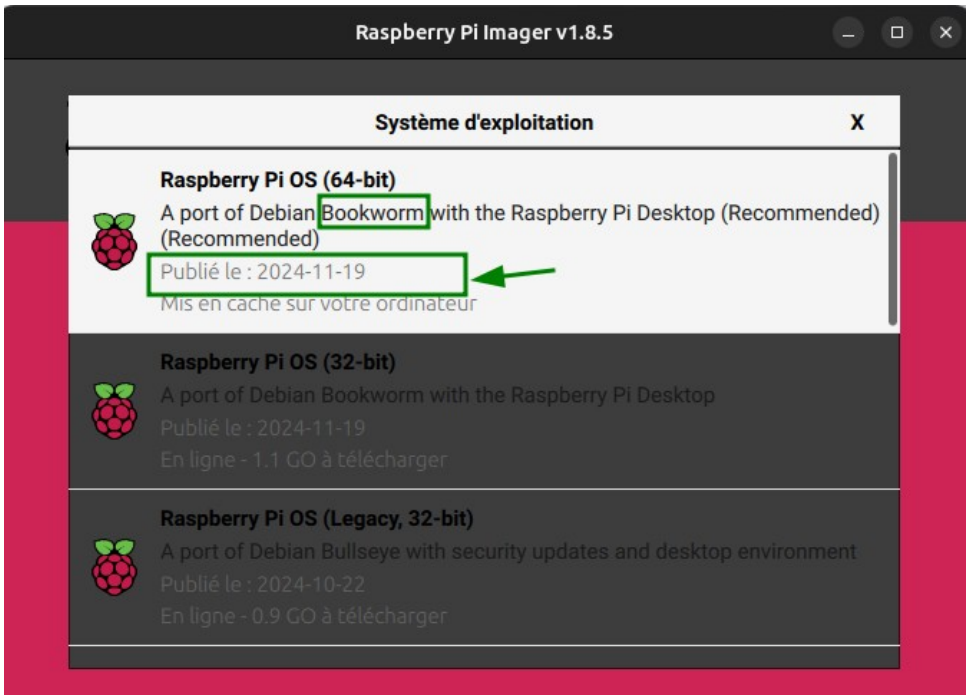

Figure 9: Flash de la carte micro SD pour la PRi4.

5.2 Configuration de la carte RPi4

Au premier démarrage de la RPi4 l'utilisateur **ucia** est créé, avec le mot de passe **poppy! station**, le système d'exploitation est mis à jour, puis le système procède à un redémarrage.

Création de l'environnement virtuel Python vision

Après le deuxième démarrage, l'environnement d'exploitation des réseaux entraînés est configuré :

```
ucia@raspberrypi:~ $ mkdir UCIA
ucia@raspberrypi:~ $ cd UCIA
ucia@raspberrypi:~/UCIA $ python -m venv --system-site-packages vision
ucia@raspberrypi:~/UCIA $ source vision/bin/activate
(vision) ucia@raspberrypi:~/UCIA $
```

4 https://www.raspberrypi.com/software/

Installation des modules Python dans l'EVP ucia activé

```
(vision) ucia@raspberrypi:~/UCIA $ pip install ultralytics
...
(vision) ucia@raspberrypi:~/UCIA $ pip install onnx
...
(vision) ucia@raspberrypi:~/UCIA $ pip onnxruntime
...
```

L'installation du module **ultralytics** est assez longue car il installe par le biais des dépendances un grand nombre de modules dont certains sont très volumineux (**pytorch**, **tensorflow**…). Au final, le dossier **~/UCIA/vision** prend environ 1.2 GiO.

5.3 Exploitation du réseau YOLO sur RPi4

Le fichier **~/.bashrc** est modifié pour activer automatiquement l'EVP vision au lancement d'un terminal.

Programme de détection des objets

Le programme **inf_camera-1.py**[5] permet de choisir un réseau **yolo** entraîné et de faire des inférences sur les images du flux vidéo de la caméra de la RPi4. Il affiche en temps réel :

- dans le terminal de lancement : les objets détectés et les temps de *pre-processing*, *inference* et *post-processing* nécessaires au traitement de chaque image par le réseau entraîné,
- dans une fenêtre graphique : le tracé des boîtes englobantes ainsi que du nom de la classe de l'objet et la confiance de détection (cf figure 11).

Le lancement du programme se fait dans le terminal :

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

Les figures 10 et 11 illustrent les fenêtres correspondantes.

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.

5 Inspiré de https://docs.ultralytics.com/fr/guides/raspberry-pi/#inference-with-camera

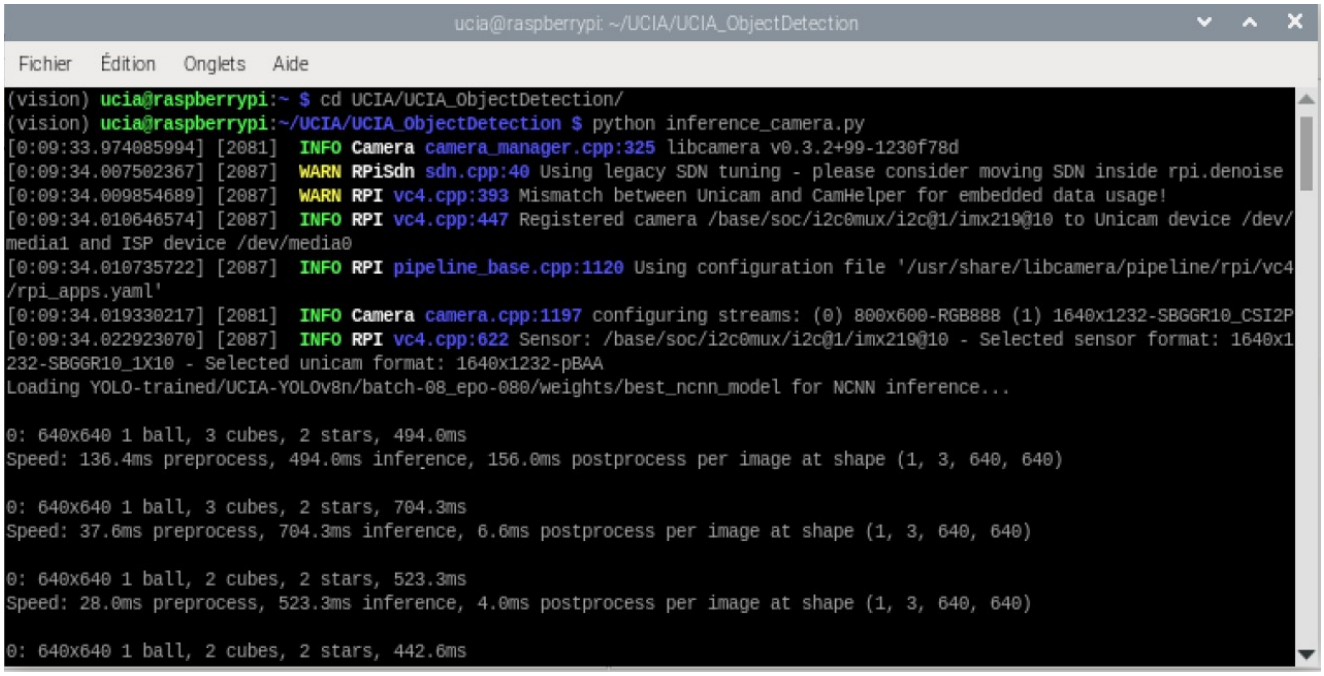

Figure 10: Terminal de lancement du programme inf_camera-1.py.

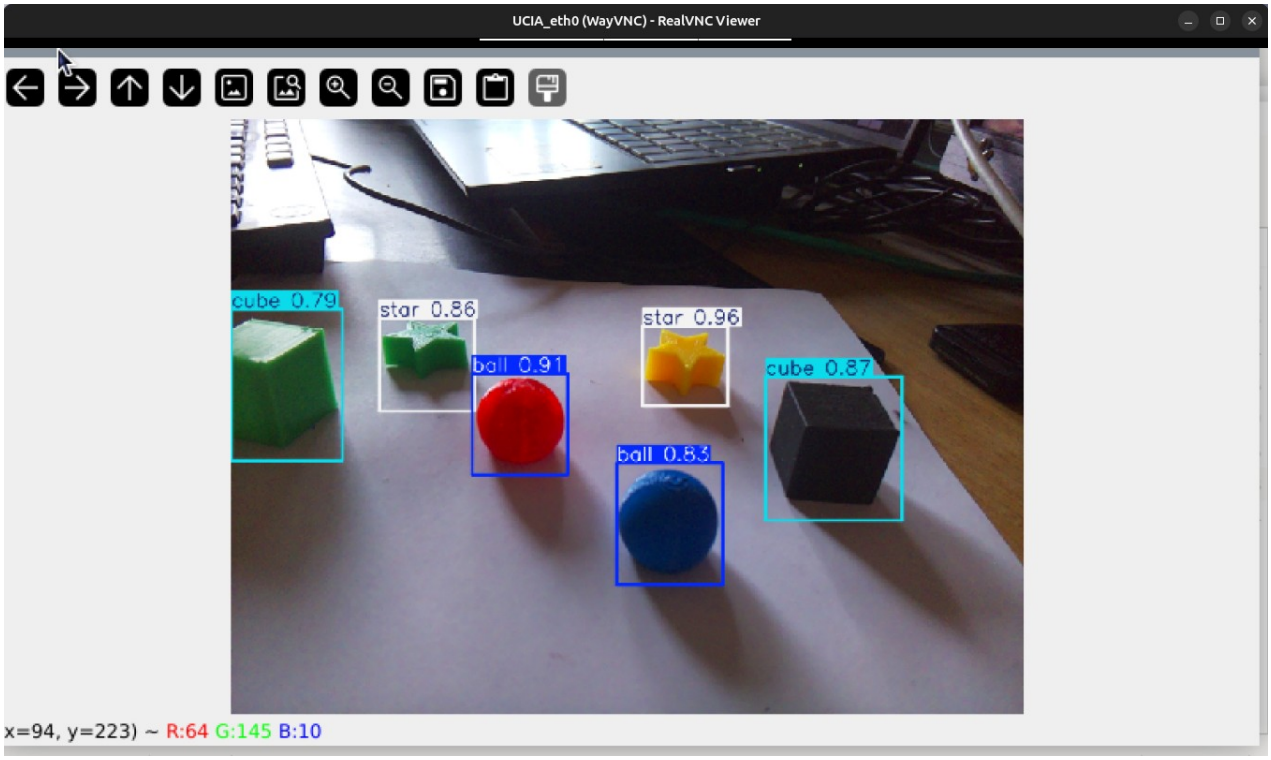

Figure 11: Fenêtre graphique d'affichage des objets détectés.

Fichiers de poids du réseau entraîné 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 12:

Figure 12: 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…

Programme de détection des objets et des couleurs

On rappelle ici que les images d'entraînement du réseau de neurones sont converties en ton de gris : l'information de couleur est absente, seule la forme est apprise. En suivant un algorithme naïf de traitement des pixels contenus dans la boîte englobante d'un objet, on peut déduire la couleur de l'objet contenu dans la boîte.

Le programme Python **inf_camera-2.py**, spécifiquement développé pour cette étude, permet de choisir un réseau **yolo** entraîné, de faire des inférences sur les images de la caméra de la RPi4. Il affiche en temps réel :

- dans le terminal :
  - le nom de l'objet, la confiance de détection, la couleur déduite des pixels contenus dans la boîte englobante de l'objet,
  - les coordonnées (x1, y2, x2, y1) des coins bas-gauche et haut-droit de la boite englobante.
- dans une fenêtre graphique : le tracé des boîtes englobantes ainsi que du nom de la classe de l'objet et la confiance de détection (cf figure 14).

Le lancement du programme se fait dans le terminal :

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

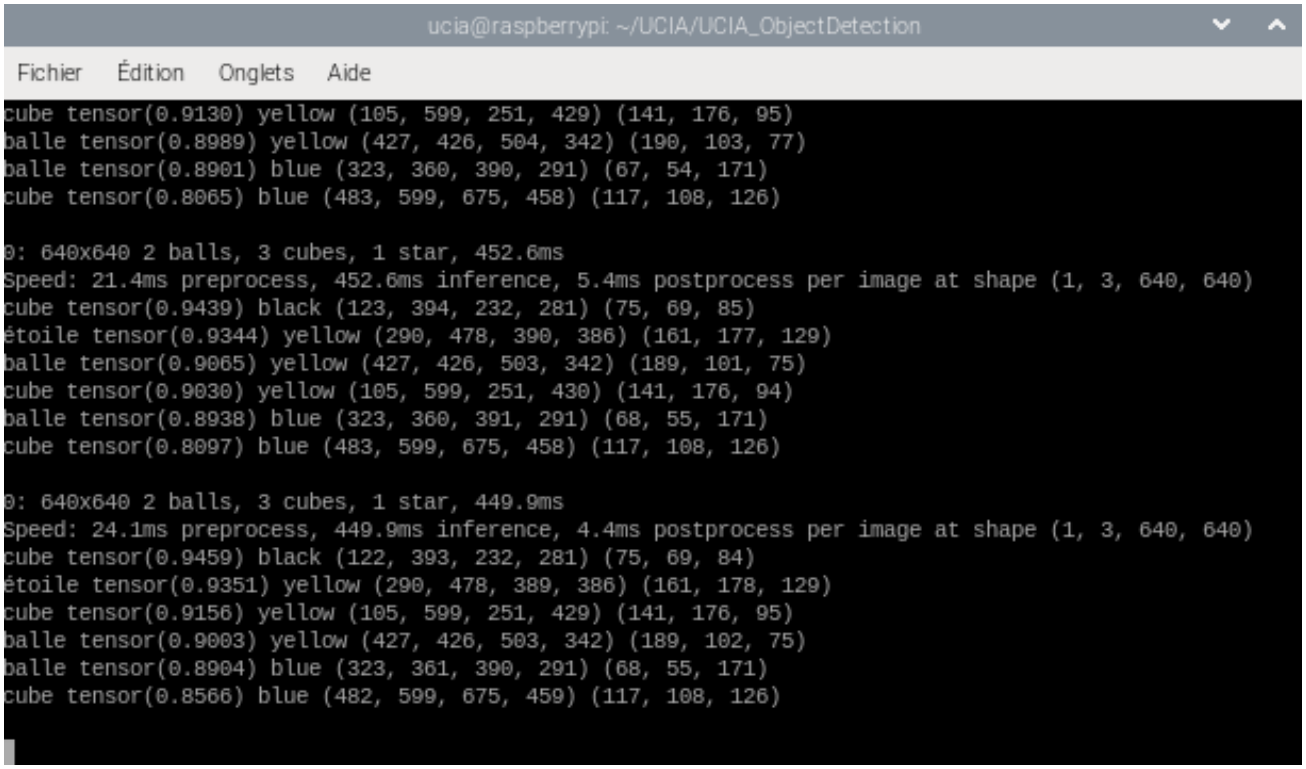

Figure 13: Terminal de lancement du programme inf_camera-2.py.

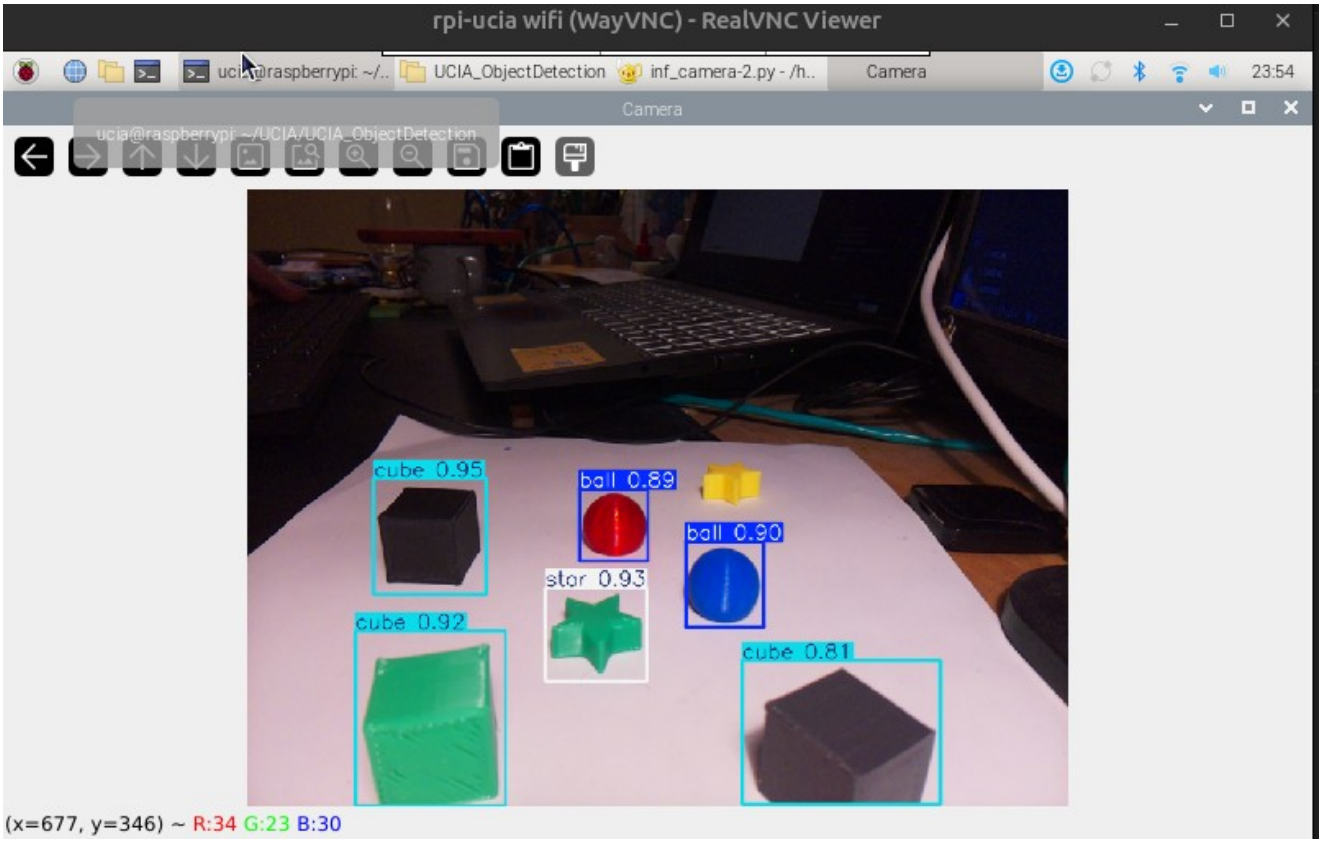

Figure 14: Fenêtre graphique d'affichage des objets détectés.

6 Conclusions

Avec les configurations d'entraînement choisies, on obtient des temps d'inférence sur RPi4 tout à fait convenables :

yolov8n_batch-08_epo-080

format onnx : environ 0.6 s
format ncnn : environ 0.5 s

yolo11n_batch-04_epo-100

format onnx : environ 0.6
format ncnn : environ 0.5 s

Cette étude montre qu'on peut entraîner un réseau YOLO à détecter les petits objets 3D du cahier des charges UCIA et que leur exploitation sur RPi4 donne des temps d'inférence inférieurs à la secondes.

À noter l'importance de placer les objets sur un fond banc de préférence, avec un bon éclairage.

Pour les performances de détection des objets, il conviendra de faire des essais dans les conditions réelles d'exploitation de la carte RPi4 pour trouver la combinaison d'entraînement qui donne les meilleurs résultats.

Dans l 'état actuel de l'étude , la détection des couleurs des objets ne fonctionne de façon robuste que pour les couleurs primaires Rouge, Vert et Bleu.

7 Glossaire

Nom anglais	*Nom français*	*signification*
Epoch	Époque	Une itération de l'entraînement du réseau de neurones sur l'ensemble complet des données.
Batch	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
Batch size	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.
Overfitting	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.
Patience	Patience	Nombre d'époques à attendre sans amélioration des mesures de validation avant d'arrêter l'entraînement. Permet d'éviter *l'overfitting* en arrêtant l'entraînement lorsque les performances atteignent un plateau.
Précision	Precision	Dans le contexte de la détection d'objets, désigne le pourcentage d'objets correctement détecté
Rappel	Recall	La capacité du modèle à identifier toutes les instances d'objets dans les images.

8 Annexes

8.1 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("numÃ©ro image pour dÃ©marrer [Q:quit] ? ")

if rep.lower() == 'q'
        picam2.stop()
        sys.exit()
else:
        n = int(rep)

while True:
        rep = input("ENTER -> image suivante [Q:quit] ...")
        if rep.lower() == 'q':
                break

        picam2.capture_file(f"objets3D-{n:03d}.jpg")
        time.sleep(1)
        n += 1

picam2.stop()
```

8.2 Programmes Python d'entraînement

train_YOLOv8.py

```
########################################
#   Jean-Luc.Charles@mailo.com
#   2024/11/14 – v1.0
########################################

from pathlib import Path
from ultralytics import YOLO
from time import sleep

BATCH = (8, 10, 16, 20, 32)
PATIENCE = (50, 100)
EPOCH = (20, 40, 60, 80)
YOLO_SIZE = ('n', 's')

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

data_path = "./datasets/yolo8-52train-10val-0test_24nov2024/data.yaml"
for size in YOLO_SIZE:

    yolo = 'YOLOv8' + size
    yolo_weights = yolo.lower() + '.pt'

    for batch in BATCH:
        for patience in PATIENCE:
            for epoch in EPOCH:
                project = f'Training/YOLO-trained/UCIA-{yolo}'
                name = f'batch-{batch:02d}_pat-{patience:03d}_epo-{epoch:03d}'
                best = Path(project, name, 'weights', 'best.pt')
                if best.exists():
                    print(f'File <{best} exists; looking for <best.onnx>... ')
                    best_onnx = Path(project, name, 'weights', 'best.onnx')
                    if not best_onnx.exists():
                        print('\t exporting <best.pt> to <best.onnx>...', end="")
                        model = YOLO(best)  # load a custom trained model
                        model.export(format="onnx", int8=True, data=data_path)
                        print(" done.")
                        del model
                    continue


                # Load the model
                model = YOLO(model_dir / yolo_weights)  # load a pretrained model
                model.train(data=data_path,
                            epochs=epoch,
                            imgsz=640,
                            batch=batch,
                            patience=patience,
                            cache=False,
                            workers=0,
                            project=project,
                            name=name,
                            exist_ok=True,
                            pretrained=True,
                            optimizer='auto',
                            seed=1234)

                sleep(1)
                del model
```

train_YOLO11.py

```
########################################
#   Jean-Luc.Charles@mailo.com
#   2024/11/14 – v1.0
########################################

from pathlib import Path
from ultralytics import YOLO
from time import sleep

BATCH = (8, 10, 16, 20, 32)
PATIENCE = (50, 100)
EPOCH = (20, 40, 60, 80)
YOLO_SIZE = ('n', 's')

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

data_path = "./datasets/yolo11-52train-10val-0test_24nov2024/data.yaml"
for size in YOLO_SIZE:

    yolo = 'YOLO11' + size
    yolo_weights = yolo.lower() + '.pt'

    for batch in BATCH:
        for patience in PATIENCE:
            for epoch in EPOCH:
                project = f'Training/YOLO-trained/UCIA-{yolo}'
                name = f'batch-{batch:02d}_pat-{patience:03d}_epo-{epoch:03d}'
                best = Path(project, name, 'weights', 'best.pt')
                if best.exists():
                    print(f'File <{best} exists; looking for <best.onnx>... ')
                    best_onnx = Path(project, name, 'weights', 'best.onnx')
                    if not best_onnx.exists():
                        print('\t exporting <best.pt> to <best.onnx>...', end="")
                        model = YOLO(best)  # load a custom trained model
                        model.export(format="onnx", int8=True, data=data_path)
                        print(" done.")
                        del model
                    continue

                # Load the model
                model = YOLO(model_dir / yolo_weights)  # load a pretrained model
                model.train(data=data_path,
                            epochs=epoch,
                            imgsz=640,
                            batch=batch,
                            patience=patience,
                            cache=False,
                            workers=0,
                            project=project,
                            name=name,
                            exist_ok=True,
                            pretrained=True,
                            optimizer='auto',
                            seed=1234)

                sleep(1)
                del model
```

eval_YOLOv8n.py

```
########################################
#   Jean-Luc.Charles@mailo.com
#   2024/11/14 - v1.0
########################################

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

data_path = "./datasets/yolo8-50train-10val-2test_27nov2024/data.yaml"

BATCH = (2, 4, 8, 10, 16, 20, 30)
EPOCH = (20, 40, 60, 80, 100)
YOLO_SIZE = ('n', 's')

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

for size in YOLO_SIZE:
    yolo = 'YOLOv8' + size
    yolo_weights = yolo.lower() + '.pt'

    # load any network for the first time:, because there i son overhead in computing the first time
    model = YOLO(f"Training/YOLO-trained/UCIA-YOLOv8{size}/batch-08_epo-020/weights/best.pt")
    metrics = model.val(batch=8, imgsz=640, data=data_path, workers=0)

    results_file = f"results_yolov8{size}.txt"
    F_out = open(results_file, "w", encoding="utf8")
    F_out.write(header)

    for batch in BATCH:
        for epoch in EPOCH:
            project = f'Training/YOLO-trained/UCIA-{yolo}'
            name = f'batch-{batch:02d}_epo-{epoch:03d}'

            best = Path(project, name, 'weights', 'best.pt')
            print(best)

            if best.exists():
                model = YOLO(best)  # load a pretrained model
                # Validate the model
                metrics = model.val(batch=batch, imgsz=640, data=data_path, workers=0)
                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()
```

eval_YOLOv11.py

```
########################################
#   Jean-Luc.Charles@mailo.com
#   2024/11/14 - v1.0
########################################

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

data_path = "./datasets/yolo11-50train-10val-2test_27nov2024/data.yaml"

BATCH = (2, 4, 8, 10, 16, 20, 30)
EPOCH = (20, 40, 60, 80, 100)
YOLO_SIZE = ('n', 's')

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

for size in YOLO_SIZE:
    yolo = 'YOLO11' + size
    yolo_weights = yolo.lower() + '.pt'

    # load any network for the first time:, because there i son overhead in computing the first time
    model = YOLO(f"Training/YOLO-trained/UCIA-YOLO11{size}/batch-08_epo-020/weights/best.pt")
    metrics = model.val(batch=8, imgsz=640, data=data_path, workers=0)

    results_file = f"results_yolo11{size}.txt"
    F_out = open(results_file, "w", encoding="utf8")
    F_out.write(header)

    for batch in BATCH:
        for epoch in EPOCH:
            project = f'Training/YOLO-trained/UCIA-{yolo}'
            name = f'batch-{batch:02d}_epo-{epoch:03d}'

            best = Path(project, name, 'weights', 'best.pt')
            print(best)

            if best.exists():
                model = YOLO(best)  # load a pretrained model
                # Validate the model
                metrics = model.val(batch=batch, imgsz=640, data=data_path, workers=0)
                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]:5.3f}')

                F_out.write('\n')
            del model
F_out.close()
```

process_results.py

```
########################################
#   Jean-Luc.Charles@mailo.com
#   2024/11/14 - v1.0
########################################

import pandas as pd

print(50*'*', "\n* Sort by 'recall' & 'mAP50-95'\n", 50*'*', sep='')

for version in ('v8n', '11n', 'v8s', '11s'):
    txt_file = f'results_yolo{version}.txt'
    print(f'\nFile <{txt_file}>', end="")

    # 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]
    print(f'\n\Max values -> "max_recall": {max_recall}, "max_mAP50-90": {max_mAP50_90}\n')

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

    # print the first 4 rows:
    print(df1.head(4))


print(50*'*', "\n\n* Sort by 'fitness'\n", 50*'*', sep='')

for version in ('v8n', '11n', 'v8s', '11s'):
    txt_file = f'results_yolo{version}.txt'
    print(f'\nFile <{txt_file}>', end="")

    # 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]
    print(f'\n\tMax values -> "fitness": {max_fitness}\n')

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

    # print the first 4 rows:
    print(df2.head(4))
```

9 Références

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

Page du site roboflow pour l'accès publique au jeu de données de l'étude : https://universe.roboflow.com/ucia/ucia-ia-object-detection/dataset/2

« Ultralytics YOLO11: Faster Than You Can Imagine! », Ankan Ghosh, October 8, 2024 https://learnopencv.com/yolo11/