



Le contenu de ce document est confidentiel.

*Le contenu de ce document est confidentiel et la copie ou la distribution est interdite.*

Pour toute question, veuillez nous contacter à [contact@datarockstars.ai](mailto:contact@datarockstars.ai)

## Le programme du notebook

### Au programme

- Challenge en full autonomie, un peu comme dans la vrai vie finalement !

Démarrer en exécutant chaque segment de code un à un et en analysant les résultats affichés.

A vous de jouer 😎 🤘

#### Soumettre votre challenge à la fin de chaque module

- Avant de soumettre le lab, pensez à bien **télécharger votre Jupyter notebook** en cliquant sur *Fichier > Download..*
- Seul le notebook `main.ipynb` sera sauvegardé et envoyé pour évaluation à vos professeurs. Si vous avez d'autres notebooks, ils ne seront **pas comptabilisés**.
- Une fois que vous avez soumis votre travail, vous ne pourrez plus le modifier. Vous aurez cependant, la possibilité de faire une nouvelle tentative autant de fois que souhaité.
- Espace disque limité : Les machines sont limitées à 250 Mb à la sauvegarde. Pensez à faire un `rm -rf ./cache >> /dev/null` après chaque pip install

Assurez-vous de vérifier vos fichiers avant de procéder!

## Challenge de Détection d'Objets pour Véhicules Autonomes

### Contexte

Les véhicules autonomes, tels que ceux développés par Tesla, utilisent des technologies avancées de détection d'objets pour naviguer en toute sécurité dans leur environnement. Ces technologies reposent largement sur des réseaux de neurones profonds qui permettent de reconnaître et de localiser divers objets sur la route, tels que d'autres véhicules, piétons, panneaux de signalisation, et bien plus.

 **La détection d'objets** est cruciale pour la sécurité et l'efficacité des véhicules autonomes. Elle permet au véhicule de prendre des décisions éclairées sur le moment d'accélérer, de freiner, ou de tourner, en fonction des objets détectés dans son environnement immédiat.

## Objectif du Challenge

L'objectif de ce challenge est double :

- 1. Charger un modèle pré-entraîné** : Vous commencerez par charger un réseau de neurones déjà entraîné sur un large ensemble de données. Ce modèle comprendra des capacités à identifier divers types d'objets que l'on peut rencontrer sur la route.
- 2. Réaliser l'inférence sur des images** : Une fois le modèle chargé, vous utiliserez ce dernier pour faire des prédictions sur de nouvelles images. Cela vous permettra de visualiser ce que "voit" le réseau de neurones — essentiellement, vous observerez à travers les "yeux" du véhicule autonome.

### Pourquoi est-ce important ?

- Innovation technologique** : Comprendre et améliorer les technologies de détection d'objets peut mener à des avancées significatives dans le domaine des véhicules autonomes.
- Sécurité** : Améliorer la précision de la détection d'objets contribue directement à rendre les véhicules autonomes plus sûrs pour tous les usagers de la route.
- Apprentissage pratique** : En participant à ce challenge, vous acquerrez une expérience pratique dans le domaine de l'apprentissage profond et de la vision par ordinateur, des compétences très recherchées dans l'industrie technologique moderne.

 Préparez-vous à plonger dans le monde fascinant de l'intelligence artificielle et de la conduite autonome. Bonne chance et bon apprentissage !

### Qu'est-ce que YOLO ?

**YOLO (You Only Look Once)** est une méthode de **détection d'objets** de pointe qui accélère et simplifie considérablement le processus d'identification des objets dans les images et les vidéos. Contrairement aux approches traditionnelles, YOLO traite la détection d'objets comme une seule étape, prédisant directement les positions et les catégories des objets.

En faisant cela, il réalise une détection en temps réel sans sacrifier la précision. L'architecture du réseau neuronal de YOLO traite les images rapidement, ce qui le rend précieux pour des applications telles que les voitures autonomes, la surveillance et la robotique.

L'approche unique de YOLO a révolutionné la détection d'objets en la rendant plus rapide et plus accessible tout en maintenant de hautes performances.

### Attention

Toujours rajouter après un pip install : `&& rm -rf ./cache >> /dev/null` pour vider le cache. Les VMs à l'enregistrement sont limités à 250 mb.

## Installations préalables

```
In [3]: # Mettre à jour la liste des paquets disponibles
!sudo apt-get update >> /dev/null
# Installer la bibliothèque OpenGL nécessaire pour certaines librairies graphiques (OpenCV)
!sudo apt-get install -y libgl1-mesa-glx >> /dev/null
# Supprimer le cache local pour libérer de l'espace disque
!rm -rf ./cache >> /dev/null
```

```
In [4]: # Mettre à jour à nouveau la liste des paquets (après modifications)
!sudo apt-get update >> /dev/null
# Installer apt-utils pour éviter les avertissements de configuration automatique + réinstalle
!sudo apt-get install -y apt-utils libgl1-mesa-glx >> /dev/null
```

## Question 1

Commencez par installer les bibliothèques `ultralytics` et `tqdm` dans votre environnement de notebook.

► 🔎 Voir indice

```
In [13]: # 🖥️ Votre code ici
!pip install ultralytics tqdm opencv-python >> /dev/null      # installe les packages Python ul
!rm -rf ./cache >> /dev/null      # Supprime tous les fichiers du cache pip pour libérer de l'e
```

### 💡 Importation de bibliothèques

**NumPy** : Utilisé pour diverses opérations numériques, essentiel dans le traitement de données.

**Matplotlib et PIL** : Aident à la visualisation des données et au traitement d'images.

**OpenCV (cv2)** : Soutient les tâches avancées de vision par ordinateur, crucial pour le traitement d'image en temps réel.

## Question 2

Préparez votre environnement Python pour travailler avec le traitement d'images. Vous aurez besoin de plusieurs bibliothèques pour charger, afficher et manipuler des images. Quelles bibliothèques devez-vous importer ?

► 🧙 Voir indice

```
In [6]: import numpy as np
import matplotlib.pyplot as plt
from PIL import Image    # Chargement et manipulation d'images

# Traitement d'images avancé
import cv2    # OpenCV pour le traitement et la transformation d'images

# Gestion des fichiers
import os      # Gestion des chemins et fichiers
import random   # Sélection aléatoire de fichiers ou d'indices

# Nettoyage de la sortie
import warnings    # Ignorer certains avertissements non critiques
warnings.filterwarnings("ignore")    # Supprime les warnings inutiles
```

```
# YOLOv8 pour la détection d'objets
from ultralytics import YOLO           # Chargement et utilisation de modèles YOLOv8
```

## 💡 YOLOv8

**YOLOv8** est un ensemble de modèles de réseaux neuronaux. Ces modèles ont été créés et entraînés en utilisant PyTorch et exportés dans des fichiers avec l'extension **.pt**. Dans ce projet, nous utilisons le **yolov8m.pt**, un modèle de taille moyenne pour la détection d'objets.

Tous les modèles YOLOv8 pour la détection d'objets sont déjà pré-entraînés sur le jeu de données COCO, qui est une vaste collection d'images de 80 types différents.

## Question 3

Configurez votre environnement Python pour utiliser le modèle YOLO avec la bibliothèque Ultralytics pour la détection d'objets. Comment prépareriez-vous votre environnement pour charger un modèle spécifique ?

► 🕵️ Voir indice

In [21]:

```
# Importation de la bibliothèque Ultralytics et la classe YOLO
from ultralytics import YOLO
# Charger le modèle YOLOv8 pré-entraîné
model = YOLO("yolov8m.pt")
# On peut installer d'autres modèles : "yolov8n.pt" (nano, plus léger, + rapide, - précis), ...
# Type de fichier : .pt (fichier PyTorch avec les poids pré-entraînés)
```

## Question 4

Supposons que vous souhaitez créer un script pour afficher quatre images choisies au hasard à partir d'un dossier hypothétique nommé `images`, qui contiendrait diverses photos. Vous voulez que ces images soient affichées sans bordures dans un arrangement de grille 2x2. Quelles étapes suivriez-vous pour réaliser ceci en Python, en utilisant des bibliothèques comme `matplotlib` pour la visualisation et `glob` pour la gestion des fichiers ?

► 🕵️ Voir indice

In [8]:

```
!pip uninstall -y numpy matplotlib opencv-python
```

```
Found existing installation: numpy 1.26.0
Uninstalling numpy-1.26.0:
  Successfully uninstalled numpy-1.26.0
Found existing installation: matplotlib 3.10.7
Uninstalling matplotlib-3.10.7:
  Successfully uninstalled matplotlib-3.10.7
Found existing installation: opencv-python 4.11.0.86
Uninstalling opencv-python-4.11.0.86:
  Successfully uninstalled opencv-python-4.11.0.86
```

In [9]:

```
# Installer une version stable de numpy < 2 pour compatibilité
!pip install numpy==1.26.0 matplotlib opencv-python
!pip cache purge >> /dev/null
```

```

Collecting numpy==1.26.0
  Downloading numpy-1.26.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (58 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.5/58.5 kB 2.7 MB/s eta 0:00:00
Collecting matplotlib
  Downloading matplotlib-3.10.7-cp311-cp311-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (11 kB)
Collecting opencv-python
  Downloading opencv_python-4.12.0.88-cp37-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (19 kB)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (4.61.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (12.0.0)
Requirement already satisfied: pyparsing>=3 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python3.11/site-packages (from matplotlib) (2.9.0.post0)
INFO: pip is looking at multiple versions of opencv-python to determine which version is compatible with other requirements. This could take a while.
  Downloading opencv_python-4.11.0.86-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (20 kB)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.11/site-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
  Downloading numpy-1.26.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.2 MB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.2/18.2 MB 59.8 MB/s eta 0:00:00:00:0100:01
  Downloading matplotlib-3.10.7-cp311-cp311-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (8.7 MB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.7/8.7 MB 55.7 MB/s eta 0:00:00:00:0100:01
  Downloading opencv_python-4.11.0.86-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (63.0 MB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.0/63.0 MB 19.2 MB/s eta 0:00:00:00:0100:01m
Installing collected packages: numpy, opencv-python, matplotlib
Successfully installed matplotlib-3.10.7 numpy-1.26.0 opencv-python-4.11.0.86

```

```

In [17]: # Importations nécessaires
import glob
import random
import matplotlib
print(matplotlib.__version__)
import cv2

# Définir le chemin où les images sont supposées être stockées (dossier images)
root_path = './images/*' # Tous les fichiers du dossier 'images'
# Utiliser la fonction `glob` pour récupérer les chemins de tous les fichiers dans ce dossier
image_paths = glob.glob(root_path) # Cherche tous les fichiers correspondant au pattern ./i
# Utiliser la fonction `sample` de la bibliothèque `random` pour choisir aléatoirement un nom
selected_images = random.sample(image_paths, 4) # Sélection aléatoire de 4 images
# Configurer une figure avec `matplotlib.pyplot`, en spécifiant la taille et le nombre de subplots
fig, axes = plt.subplots(2, 2, figsize=(10, 10))
# Charger et afficher chaque image sélectionnée dans les subplots préparés sans afficher les axes
for ax, img_path in zip(axes.flatten(), selected_images): # Boucle sur les images sélectionnées
    img = Image.open(img_path) # Charger l'image
    ax.imshow(img) # imshow → affiche l'image dans le subplot
    ax.axis('off') # axis('off') → supprime les axes pour que seules les images soient visibles
# Affichage final
plt.tight_layout()
plt.show()

```

```
# Question : Comment configureriez-vous le subplot et la fonction imshow pour obtenir le résultat souhaité ? Pour afficher quatre images choisies au hasard en grille 2x2 sans bordures, je configurerais la grille avec deux lignes et deux colonnes, puis j'utiliserais ax.imshow(image) et ax.axis('off') afin de masquer les axes et obtenir un rendu propre.
```

### 3.10.7



## Chargement du jeu de données pour voitures autonomes

Dans cette section, nous avons chargé le jeu de données d'images **voitures autonomes**, qui est utilisé pour l'entraînement et les tests des systèmes de véhicules autonomes. Ce jeu de données est crucial pour le développement et l'évaluation des performances des algorithmes et modèles de conduite autonome.

Nous avons ensuite sélectionné aléatoirement quelques images pour mettre en œuvre le modèle yolov8 sur elles comme échantillons.

## Question 5 : Fill the blanks

In [23]:

```
import time
from tqdm import tqdm # nécessaire pour la barre de progression
from ultralytics import YOLO

# Charger le modèle pré-entraîné YOLOv8 (ex : yolov8m)
yolo_model = YOLO("yolov8m.pt")

# Charger les chemins des images
image_paths = glob.glob('./images/*') # tous les fichiers du dossier images
```

```

# Sélectionner aléatoirement un sous-ensemble
num_samples = min(4, len(image_paths)) # ici on prend max 4 images
selected_paths = random.sample(image_paths, num_samples)

# Charger les images dans une liste
images = [Image.open(p) for p in selected_paths]

# Fonction du module Python time qui retourne le temps actuel en secondes depuis le « epoch »
start_time = time.time()

# Spécifier Le nombre d'échantillons à traiter : range(num_samples), chaque nombre correspond
for i in tqdm(range(num_samples), desc="Traitement des images"): # Crée une séquence de n
    # Utiliser le modèle YOLO pour les prédictions : ce modèle est pré-entraîné pour la détec
    yolo_outputs = yolo_model.predict(images[i]) # images[i] correspond à l'image spécifique
    output = yolo_outputs[0] # On stocke le résultat dans yolo_outputs, une liste ou un obj
    box = output.boxes # toutes les boîtes englobantes détectées.
    names = output.names # noms des classes (ex. « voiture », « piéton », « chien », etc.). 

    # Commencez une boucle pour traiter chaque boîte englobante détectée.
    for j in range(len(box)):
        labels = names[box.cls[j].item()] # box.cls[j] : l'indice de classe pour l'objet j
        # .item() : convertit le tenseur PyTorch en un entier Python
        coordinates = box.xyxy[j].tolist() # coordonnées de la boîte englobante sous forme
        confidence = np.round(box.conf[j].item(), 2) # box.conf[j] → le score de confiance
        # .item() → convertit le tenseur en float # confidence contient la probabilité que
        print(f'Objet {j + 1} est : {labels}')
        print(f'Les coordonnées sont : {coordinates}')
        print(f'La confiance est : {confidence}')
        print('-----')

    # Ajoutez l'image traitée, convertie au bon format de couleur, à La liste 'images'.
    images.append(output.plot()[:, :, ::-1]) # méthode .plot() : Dessine les boîtes englobant
    # [:, :, ::-1] : Cette notation inverse l'ordre des canaux de couleur
    # En Python/OpenCV : OpenCV utilise BGR (Bleu, Vert, Rouge), Matplotlib utilise RGB (Rouge
    # [:, :, ::-1] convertit donc l'image de BGR en RGB pour que l'affichage avec Matplotlib s
    # images.append(...) : Ajoute cette image annotée à la liste images, pour éventuellement l

# Marquer la fin du chronométrage :
end_time = time.time()
# Calculez le temps total en soustrayant 'start_time' de 'end_time'.
total_time = end_time - start_time
# Affichez le temps total d'exécution.
print(f"Temps total d'exécution : {total_time:.2f} secondes") # Utile pour mesurer les performances

```

Traitement des images: 0% | 0/4 [00:00<?, ?it/s]

0: 416x640 4 cars, 1 traffic light, 1869.5ms

Speed: 90.4ms preprocess, 1869.5ms inference, 48.1ms postprocess per image at shape (1, 3, 416, 640)

Traitement des images: 25% | ■ | 1/4 [00:02<00:08, 2.94s/it]

```
Objet 1 est : car
Les coordonnées sont : [121.32820129394531, 139.78839111328125, 188.18898010253906, 158.681854
24804688]
La confiance est : 0.7
-----
Objet 2 est : car
Les coordonnées sont : [252.7035369873047, 145.61402893066406, 278.5704650878906, 164.70693969
726562]
La confiance est : 0.68
-----
Objet 3 est : car
Les coordonnées sont : [190.71791076660156, 147.99998474121094, 203.3033447265625, 158.2729797
3632812]
La confiance est : 0.63
-----
Objet 4 est : traffic light
Les coordonnées sont : [310.6459655761719, 108.39473724365234, 319.9971618652344, 126.46537017
822266]
La confiance est : 0.57
-----
Objet 5 est : car
Les coordonnées sont : [0.14186668395996094, 131.70941162109375, 92.75495910644531, 173.958297
7294922]
La confiance est : 0.57
-----

0: 416x640 3 cars, 1 truck, 1 traffic light, 1265.8ms
Speed: 4.2ms preprocess, 1265.8ms inference, 2.1ms postprocess per image at shape (1, 3, 416,
640)

Traitement des images: 50%|██████ | 2/4 [00:04<00:03, 1.97s/it]

Objet 1 est : truck
Les coordonnées sont : [0.2530860900878906, 124.64215850830078, 155.1016387939453, 181.3293609
6191406]
La confiance est : 0.82
-----
Objet 2 est : car
Les coordonnées sont : [254.25201416015625, 145.7113037109375, 278.4407958984375, 164.29664611
816406]
La confiance est : 0.64
-----
Objet 3 est : traffic light
Les coordonnées sont : [310.71856689453125, 108.1783676147461, 320.3019104003906, 126.13085174
560547]
La confiance est : 0.45
-----
Objet 4 est : car
Les coordonnées sont : [195.63748168945312, 147.3741455078125, 214.84222412109375, 154.9672546
3867188]
La confiance est : 0.3
-----
Objet 5 est : car
Les coordonnées sont : [198.80023193359375, 148.2707061767578, 214.5295867919922, 154.35971069
335938]
La confiance est : 0.3
-----

0: 416x640 2 persons, 5 cars, 5 traffic lights, 2571.2ms
Speed: 3.8ms preprocess, 2571.2ms inference, 9.9ms postprocess per image at shape (1, 3, 416,
640)

Traitement des images: 75%|███████ | 3/4 [00:06<00:02, 2.26s/it]
```

Objet 1 est : person  
Les coordonnées sont : [364.7071838378906, 114.37841033935547, 404.2037658691406, 213.20385742  
1875]  
La confiance est : 0.85  
-----  
Objet 2 est : person  
Les coordonnées sont : [396.4638977050781, 113.4627685546875, 425.7748718261719, 216.517913818  
35938]  
La confiance est : 0.84  
-----  
Objet 3 est : car  
Les coordonnées sont : [254.68519592285156, 140.00685119628906, 277.1495666503906, 157.2225189  
2089844]  
La confiance est : 0.78  
-----  
Objet 4 est : car  
Les coordonnées sont : [432.5591125488281, 131.4907989501953, 479.57080078125, 153.91659545898  
438]  
La confiance est : 0.62  
-----  
Objet 5 est : traffic light  
Les coordonnées sont : [307.0335693359375, 105.38248443603516, 315.18035888671875, 121.3931808  
4716797]  
La confiance est : 0.49  
-----  
Objet 6 est : traffic light  
Les coordonnées sont : [45.18324279785156, 114.0329360961914, 53.68509292602539, 136.116958618  
16406]  
La confiance est : 0.43  
-----  
Objet 7 est : traffic light  
Les coordonnées sont : [307.5189514160156, 105.54972839355469, 313.5484924316406, 116.47602081  
298828]  
La confiance est : 0.36  
-----  
Objet 8 est : car  
Les coordonnées sont : [246.24728393554688, 141.2423095703125, 260.889404296875, 153.025833129  
8828]  
La confiance est : 0.35  
-----  
Objet 9 est : car  
Les coordonnées sont : [147.97210693359375, 138.6293487548828, 170.50408935546875, 147.1740570  
0683594]  
La confiance est : 0.34  
-----  
Objet 10 est : traffic light  
Les coordonnées sont : [45.908409118652344, 114.27578735351562, 53.03371810913086, 129.9474334  
716797]  
La confiance est : 0.33  
-----  
Objet 11 est : traffic light  
Les coordonnées sont : [451.3141784667969, 62.76543426513672, 459.9576721191406, 83.8662796020  
5078]  
La confiance est : 0.32  
-----  
Objet 12 est : car  
Les coordonnées sont : [196.557861328125, 142.08311462402344, 207.19290161132812, 148.97727966  
308594]  
La confiance est : 0.28  
-----

0: 416x640 3 cars, 2 trucks, 2029.2ms

Speed: 8.0ms preprocess, 2029.2ms inference, 2.6ms postprocess per image at shape (1, 3, 416,  
640)

Traitemen des images: 100% |██████████| 4/4 [00:08<00:00, 2.23s/it]

```
Objet 1 est : truck
Les coordonnées sont : [326.26202392578125, 103.027587890625, 408.81243896484375, 182.24711608
88672]
La confiance est : 0.85
-----
Objet 2 est : car
Les coordonnées sont : [302.0107727050781, 142.7921600341797, 319.9681091308594, 165.455581665
03906]
La confiance est : 0.73
-----
Objet 3 est : truck
Les coordonnées sont : [50.24087905883789, 136.24417114257812, 97.32408905029297, 160.63189697
265625]
La confiance est : 0.32
-----
Objet 4 est : car
Les coordonnées sont : [49.879188537597656, 136.23251342773438, 97.4120864868164, 161.63624572
753906]
La confiance est : 0.32
-----
Objet 5 est : car
Les coordonnées sont : [318.3772277832031, 137.8667449951172, 331.524169921875, 169.2758789062
5]
La confiance est : 0.25
-----
Temps total d'exécution : 8.92 secondes
```

## Question 6 : Affichez les prédictions en utilisant YoloV8

```
In [24]: # valeurs appropriées pour définir la taille de la figure matplotlib.
plt.figure(figsize=(12, 8)) # Taille suffisamment grande pour afficher 4 images
# Commencez une boucle pour afficher chaque image dans 'images'.
for i, img in enumerate(images):
    # Définir la position et le nombre de sous-graphiques (2 Lignes, 2 colonnes).
    plt.subplot(2, 2, i + 1)
    # Utilisez 'imshow' pour afficher 'img'.
    plt.imshow(img) # affiche l'image brute
    # Désactivez les axes pour une meilleure visibilité des images.
    plt.axis('off')
# Ajustez automatiquement les paramètres du subplot pour donner un espace spécifié.
plt.tight_layout()
# Affichez la figure avec tous les sous-graphiques.
plt.show()
```

```
ValueError
```

```
Traceback (most recent call last)
```

```
Cell In[24], line 6
```

```
    3 # Commencez une boucle pour afficher chaque image dans 'images'.
    4 for i, img in enumerate(images):
    5     # Définir la position et le nombre de sous-graphiques (2 lignes, 2 colonnes).
--> 6     plt.subplot(2, 2, i + 1)
    7     # Utilisez 'imshow' pour afficher 'img'.
    8     plt.imshow(img)      # affiche l'image brute
```

```
File /opt/conda/lib/python3.11/site-packages/matplotlib/pyplot.py:1551, in subplot(*args, **kwargs)
```

```
1548 fig = gcf()
1550 # First, search for an existing subplot with a matching spec.
-> 1551 key = SubplotSpec._from_subplot_args(fig, args)
1553 for ax in fig.axes:
1554     # If we found an Axes at the position, we can reuse it if the user passed no
1555     # kwargs or if the Axes class and kwargs are identical.
1556     if (ax.get_subplotspec() == key
1557         and (kwargs == {}
1558             or (ax._projection_init
1559                 == fig._process_projection_requirements(**kwargs))):
```

```
File /opt/conda/lib/python3.11/site-packages/matplotlib/gridspec.py:589, in SubplotSpec._from_subplot_args.figure, args)
```

```
587 else:
588     if not isinstance(num, Integral) or num < 1 or num > rows*cols:
--> 589         raise ValueError(
590             f"num must be an integer with 1 <= num <= {rows*cols}, "
591             f"not {num!r}"
592         )
593     i = j = num
594 return gs[i-1:j]
```

```
ValueError: num must be an integer with 1 <= num <= 4, not 5
```



```
In [28]: # Code YOLOv8 : Affichage des prédictions annotées
```

```
plt.figure(figsize=(12, 8))
```

```

for i, img in enumerate(images):
    # Obtenir les prédictions YOLO pour chaque image
    results = yolo_model.predict(img)

    # YOLOv8 renvoie une image annotée via results[0].plot()
    annotated_img = results[0].plot()

    # Affichage dans une grille 2x2
    plt.subplot(2, 2, i + 1)
    plt.imshow(annotated_img)
    plt.axis('off')

plt.tight_layout()
plt.show()

```

0: 416x640 4 cars, 1 traffic light, 1356.0ms

Speed: 13.7ms preprocess, 1356.0ms inference, 2.3ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 3 cars, 1 truck, 1 traffic light, 1176.6ms

Speed: 5.7ms preprocess, 1176.6ms inference, 2.0ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 2 persons, 5 cars, 5 traffic lights, 1124.1ms

Speed: 3.1ms preprocess, 1124.1ms inference, 2.7ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 3 cars, 2 trucks, 1230.4ms

Speed: 2.8ms preprocess, 1230.4ms inference, 1.9ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 (no detections), 1263.1ms

Speed: 6.0ms preprocess, 1263.1ms inference, 1.1ms postprocess per image at shape (1, 3, 416, 640)

```

ValueError                                         Traceback (most recent call last)
Cell In[28], line 13
    10 annotated_img = results[0].plot()
    11 # Affichage dans une grille 2x2
--> 13 plt.subplot(2, 2, i + 1)
    14 plt.imshow(annotated_img)
    15 plt.axis('off')

File /opt/conda/lib/python3.11/site-packages/matplotlib/pyplot.py:1551, in subplot(*args, **kwargs)
 1548 fig = gcf()
 1550 # First, search for an existing subplot with a matching spec.
-> 1551 key = SubplotSpec._from_subplot_args(fig, args)
 1553 for ax in fig.axes:
 1554     # If we found an Axes at the position, we can reuse it if the user passed no
 1555     # kwargs or if the Axes class and kwargs are identical.
 1556     if (ax.get_subplotspec() == key
 1557         and (kwargs == {}
 1558             or (ax._projection_init
 1559                 == fig._process_projection_requirements(**kwargs)))):

```

File /opt/conda/lib/python3.11/site-packages/matplotlib/gridspec.py:589, in SubplotSpec.\_from\_subplot\_args(figure, args)

```

 587 else:
 588     if not isinstance(num, Integral) or num < 1 or num > rows*cols:
--> 589         raise ValueError(
 590             f"num must be an integer with 1 <= num <= {rows*cols}, "
 591             f"not {num!r}"
 592         )
 593     i = j = num
 594 return gs[i-1:j]

```

ValueError: num must be an integer with 1 <= num <= 4, not 5



In [25]: # Limiter l'affichage à 4 images  
plt.figure(figsize=(12, 8))

for i, img in enumerate(images[:4]): # <= Limite à 4 images

```
plt.subplot(2, 2, i + 1)
plt.imshow(img)
plt.axis('off')

plt.tight_layout()
plt.show()
```



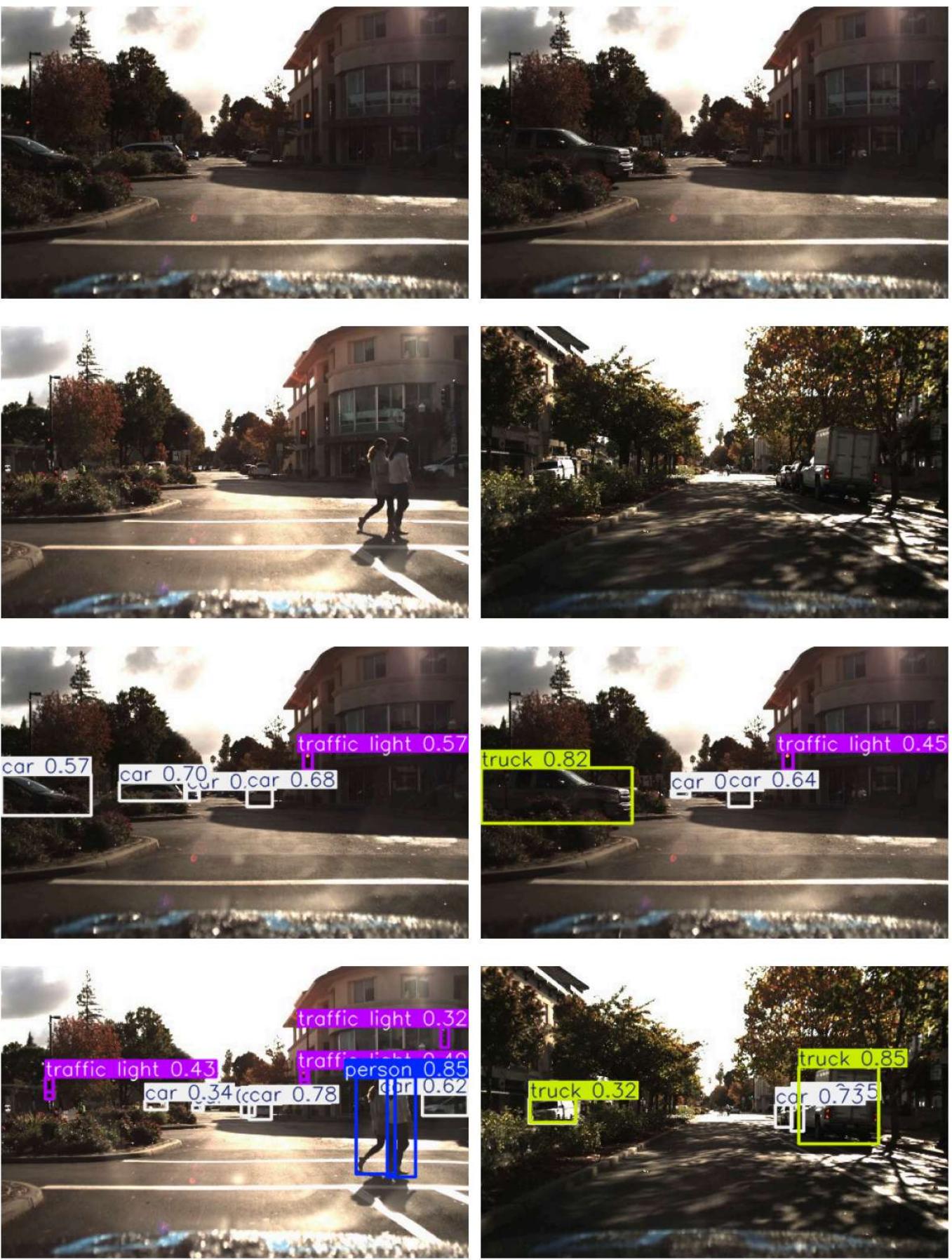
In [26]: # Résolution : Créer automatiquement la bonne grille selon le nombre d'images  
import math

```
n = len(images)
cols = 2
rows = math.ceil(n / cols)

plt.figure(figsize=(12, rows * 4))

for i, img in enumerate(images):
    plt.subplot(rows, cols, i + 1)
    plt.imshow(img)
    plt.axis('off')

plt.tight_layout()
plt.show()
```



```
In [27]: # Conclusion : Affichage des images + prédictions YOLOv8 (automatique)
```

```
import math
import matplotlib.pyplot as plt

# Nombre total d'images
n = len(images)

# Grille automatique (2 colonnes)
cols = 2
rows = math.ceil(n / cols)
```

```

plt.figure(figsize=(12, rows * 4))

for i, img in enumerate(images):

    # --- Prédictions YOLO ---
    yolo_outputs = yolo_model.predict(img)
    output = yolo_outputs[0]
    box = output.boxes

    # --- SubPlot ---
    plt.subplot(rows, cols, i + 1)

    # Affichage de l'image + bbox YOLO
    output.plot()  # YOLO dessine directement les box sur l'image
    plt.imshow(output.plot()[:, :, :-1])  # conversion BGR → RGB
    plt.axis("off")

plt.tight_layout()
plt.show()

```

0: 416x640 4 cars, 1 traffic light, 1223.9ms

Speed: 9.9ms preprocess, 1223.9ms inference, 2.2ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 3 cars, 1 truck, 1 traffic light, 1152.4ms

Speed: 2.9ms preprocess, 1152.4ms inference, 2.9ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 2 persons, 5 cars, 5 traffic lights, 1240.9ms

Speed: 3.1ms preprocess, 1240.9ms inference, 2.7ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 3 cars, 2 trucks, 1190.1ms

Speed: 2.7ms preprocess, 1190.1ms inference, 2.1ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 (no detections), 1169.4ms

Speed: 6.1ms preprocess, 1169.4ms inference, 1.4ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 1 car, 1 truck, 1264.7ms

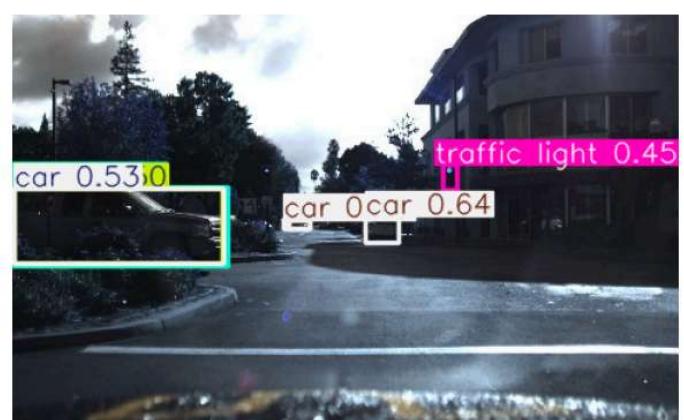
Speed: 5.3ms preprocess, 1264.7ms inference, 1.6ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 1 person, 1265.5ms

Speed: 6.9ms preprocess, 1265.5ms inference, 1.5ms postprocess per image at shape (1, 3, 416, 640)

0: 416x640 1 person, 1556.6ms

Speed: 6.7ms preprocess, 1556.6ms inference, 1.6ms postprocess per image at shape (1, 3, 416, 640)



In [ ]: