

***Projet Detection de Voitures***

Réalisé par :

IZZEM Loubna ( lhbiiila )

Encadre par :

………………………………….

# **Sommaire :**

1. **Introduction**
2. **Présentation de l’environnement de développement**
3. **Model YOLOv8**
4. **Collecte de données**
5. **Annotation des images**
6. **Préparation du dataset**
7. **Entraînement du modèle YOLOv8**
8. **Validation du modèle**
9. **Tests sur des images et des vidéos**
10. **Conclusion**
11. **Introduction :**

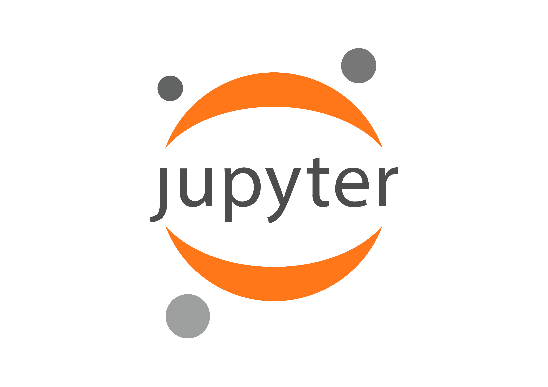
La détection d'objets est un domaine de recherche en vision par ordinateur qui joue un rôle essentiel dans de nombreux domaines d'application, tels que la surveillance, la conduite autonome et la reconnaissance d'activités. Dans ce projet, notre objectif est de développer un système de détection de voitures en utilisant l'algorithme YOLO (You Only Look Once).

La détection de voitures présente des défis uniques en raison de leur diversité de formes, de tailles et de poses. En utilisant YOLO, un réseau de neurones convolutif spécialement conçu pour la détection d'objets en temps réel, nous cherchons à détecter et localiser les voitures dans des images et des vidéos avec précision et efficacité.

Ce projet repose sur plusieurs étapes clés. Tout d'abord, nous avons collecté un ensemble de données contenant des images contenant des voitures à partir de diverses sources en ligne. Ensuite, nous avons annoté ces images en marquant les emplacements des voitures à l'aide de boîtes englobantes.

Une fois les données annotées préparées, nous avons entraîné le modèle YOLO en utilisant une partie des données d'entraînement. Nous avons ajusté les paramètres d'entraînement et surveillé la progression de l'apprentissage pour obtenir un modèle performant.

Pour évaluer la performance de notre modèle, nous avons utilisé une partie des données réservées à la validation et calculé des métriques telles que la précision, le rappel et le score F1. Nous avons également effectué des tests sur des images et des vidéos pour évaluer la capacité de notre modèle à détecter les voitures dans des scénarios réels.

1. **Présentation de l’environnement de développement**

**Jupyter** est un environnement de développement interactif open-source qui permet de créer et de partager des documents contenant du code, des visualisations, des textes explicatifs et des équations mathématiques. Le nom **"Jupyter"** est un acronyme formé à partir des langages de programmation pris en charge : Julia, Python et R. Les documents **Jupyter** sont appelés *"notebooks"* et peuvent être exécutés dans un navigateur Web en utilisant un serveur **Jupyter.**

Il est disponible sur toutes les principales plates-formes, y compris **Windows**, **Mac OS X** et **Linux**. Il peut être installé à l'aide d'un gestionnaire de paquets ou à partir de la distribution Anaconda, qui comprend également d'autres bibliothèques et outils de science des données populaires.

**Jupyter** est un environnement de développement interactif très populaire pour la science des données qui offre une expérience de développement flexible et personnalisable, avec des fonctionnalités de code, de texte, de visualisation et d'équations mathématiques.

1. **YOLOv8 :**

****

YOLO est un réseau de neurones qui traite toute l’image d’un seul coup d’où le nom “You Only Look Once”, ce qui va lui permettre de faire de la détection en temps réel sur des vidéos. Rappelons qu’il traite les images à 30 images par seconde (IPS). Cette détection en temps réel est un véritable atout notamment pour des cas d’usage bien précis comme l’exemple vu de la voiture autonome ou encore de la réalité augmentée.

En effet, après avoir installé une caméra dans une voiture en déplacement et après avoir récupéré les images prise par la caméra, YOLO nous a permis  de faire de la détection d’objets sur la route tels que les feux tricolores, les panneaux de direction, les piétons ainsi que les voitures.

Toutefois, YOLO n’est pas le seul modèle existant capable de faire de la détection. Il en existe plusieurs autres tels que R-CNN, fast R-CNN, faster R-CNN…

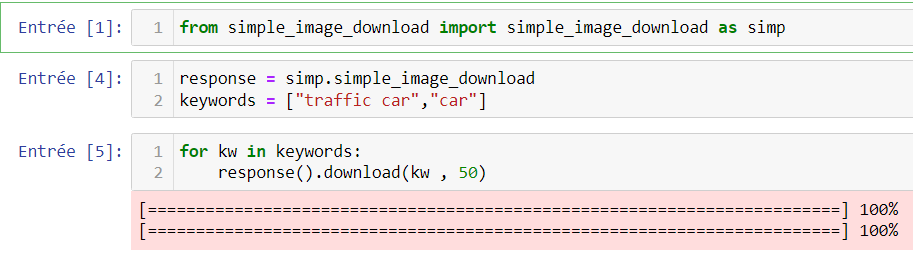
Mais ce qui rend YOLO différent de ces derniers est sa vitesse d’exécution. En effet, les différents modèles du type « R-CNN » vont définir, pour chaque image en entrée (selon des méthodes spécifiques aux différentes versions de R-CNN), un ensemble de régions susceptibles de contenir un objet recherché. Le réseau de neurones va analyser chacune de ces régions séparément et déterminer si elles contiennent un objet ou non, et si oui, la classe correspondante. Pour chaque image passée en entrée, le réseau va donc devoir définir et analyser une multitude de régions, ce qui prend du temps.

1. **Collecte de données**

Pour la collecte de données, nous avons opté pour l'utilisation de la bibliothèque simple\_image\_download, qui offre une interface conviviale et facilite le processus de téléchargement d'images à partir de sources en ligne. Nous avons choisi cette bibliothèque en raison de sa simplicité d'utilisation et de sa compatibilité avec Python, ce qui nous a permis d'automatiser le processus de téléchargement des images contenant des voitures.

****

Le dataset téléchargé se compose d'un ensemble d'images contenant des voitures provenant de diverses sources en ligne. Nous avons collecté un total de 100 images de voitures pour notre projet. Ces images ont été téléchargées à partir de différentes sources, telles que des bases de données publiques d'images de voitures, des sites d'annonces automobiles et des forums spécialisés.

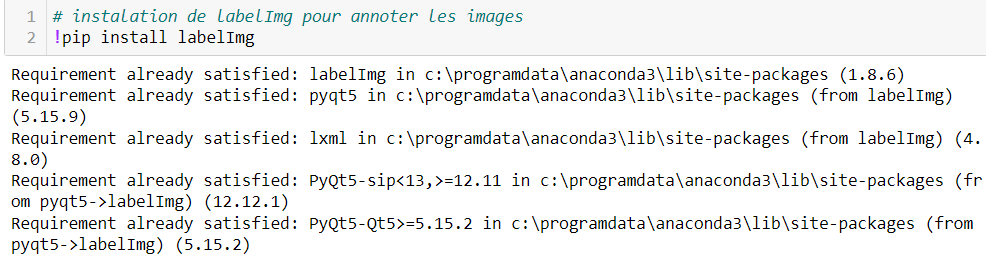
****

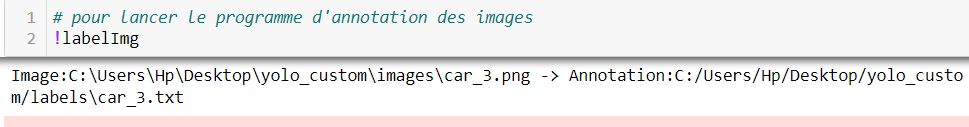
Chaque image dans le dataset est de résolution variée et représente différents environnements et perspectives dans lesquels les voiturespeuvent se trouver. Cette diversité d'images permettra à notre modèle de s'entraîner sur des scénarios réalistes et de généraliser sa capacité à détecter les voitures dans différentes conditions.

En utilisant la bibliothèque simple\_image\_download et en collectant un ensemble diversifié d'images de voitures, nous avons établi une base solide de données pour entraîner et évaluer notre modèle de détection de voitures. Cette étape essentielle nous permettra de poursuivre notre projet avec des données fiables et représentatives des objets que nous souhaitons détecter.

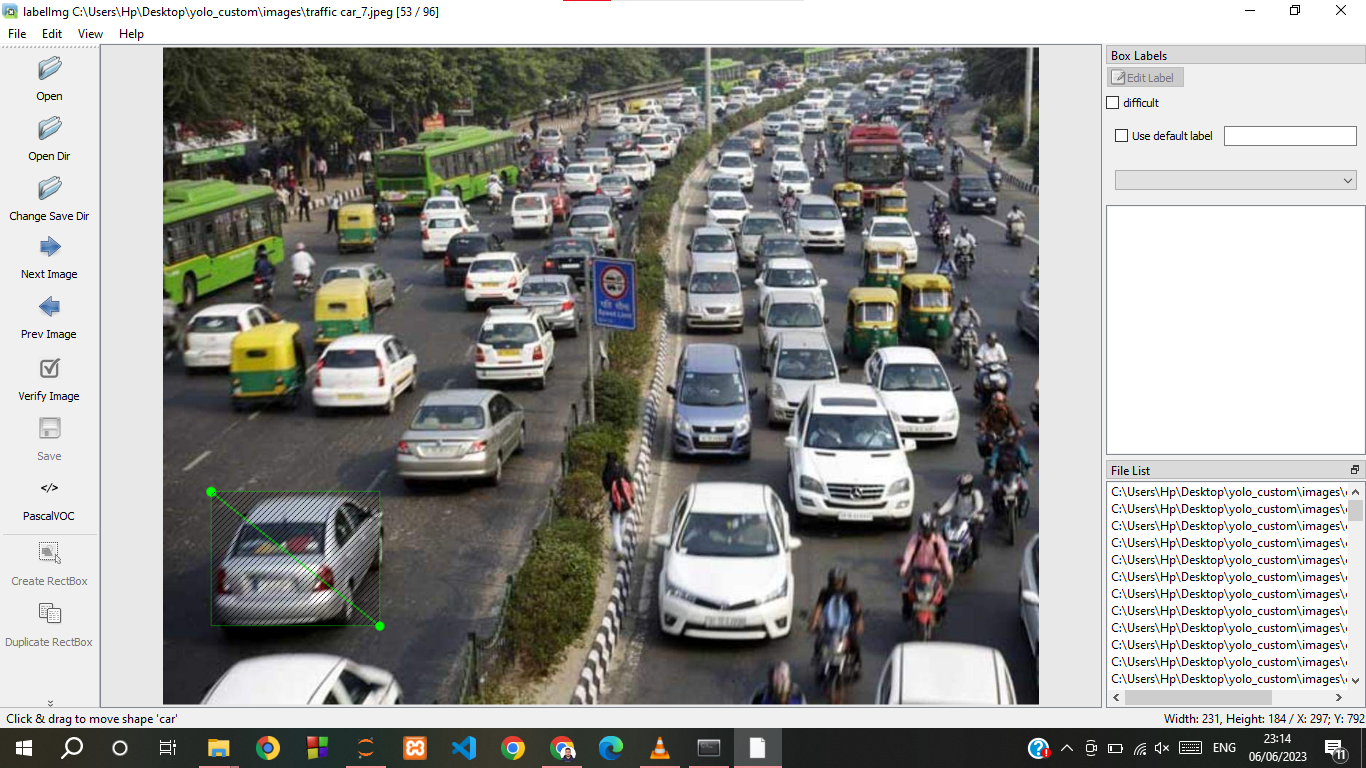
1. **Annotation des images :**

Pour annoter nos images avec des boîtes englobantes autour des voitures, nous avons utilisé l'outil d'annotation d'objets appelé labelImg. Nous avons choisi cet outil en raison de sa simplicité d'utilisation, de sa flexibilité et de sa compatibilité avec différents formats d'annotations.

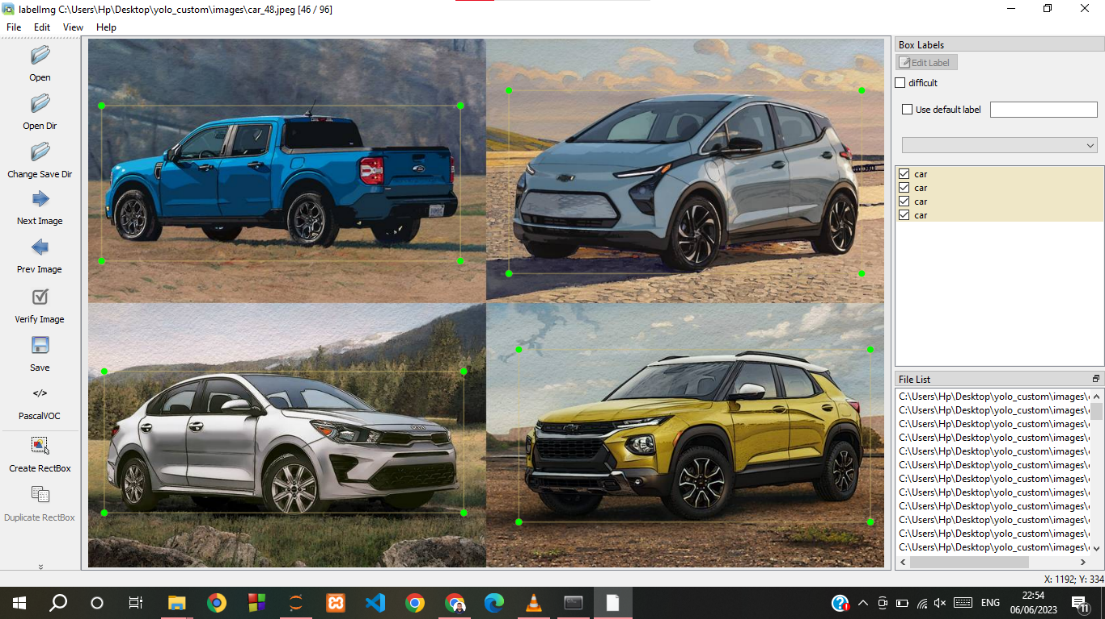




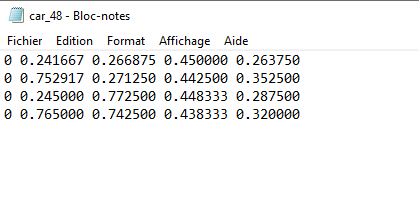
Le processus d'annotation des images s'est déroulé en plusieurs étapes. Tout d'abord, nous avons ouvert une image dans l'interface de labelImg. Ensuite, nous avons utilisé les outils de dessin fournis par l'outil pour créer des boîtes englobantes autour des voitures présentes dans l'image.



Le processus d'annotation a nécessité une attention minutieuse pour s'assurer que les boîtes englobantes couvraient précisément les voitures sans inclure d'autres objets ou arrière-plans. Nous avons veillé à ce que les boîtes englobantes soient bien ajustées et respectent les contours des voitures.



Lors de la création des annotations, nous avons enregistré les coordonnées des boîtes englobantes en spécifiant les positions des coins supérieur gauche et inférieur droit de chaque boîte, ainsi que leur classe correspondante. Ces informations ont été enregistrées dans des fichiers .txt portant le même nom que les images correspondantes.



**Indice du classe**

Le processus d'annotation des images a été répété pour chaque image de notre dataset, garantissant ainsi que toutes les voitures étaient correctement marquées. Cette étape d'annotation est cruciale car elle fournit les données d'entraînement nécessaires pour que notre modèle YOLO puisse apprendre à détecter les voitures dans de nouvelles images avec précision.

1. **Préparation du dataset :**

Dans le cadre de la préparation du dataset, nous avons créé un dossier principal nommé "data". Ce dossier contient deux sous-dossiers : "train" pour les données d'entraînement et "val" pour les données de validation. Chacun de ces sous-dossiers est organisé de manière similaire.

Le sous-dossier "train" contient deux dossiers supplémentaires. Le premier dossier, appelé "images", contient les images utilisées pour l'entraînement de notre modèle. Ces images représentent une variété de scènes contenant des voitures. Le deuxième dossier, appelé "annotations", contient les fichiers d'annotations correspondant à chaque image. Ces fichiers d'annotations spécifient les positions des objets dans l'image ainsi que leurs classes.

De même, le sous-dossier "val" suit la même structure que le dossier "train", avec les dossiers "images" et "annotations" contenant respectivement les images de validation et les fichiers d'annotations correspondants.

data

val

train

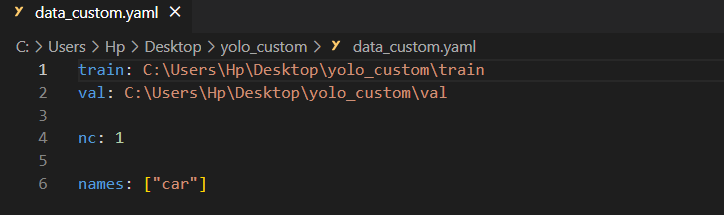
images

labels

labels

images

En plus de la structure du dossier "data", notre modèle a besoin d'un fichier .yaml pour configurer les informations spécifiques au dataset. Dans notre projet, ce fichier est nommé "data\_custom.yaml". Ce fichier contient les informations essentielles, notamment le chemin absolu vers les dossiers "train" et "val", le nombre de classes, et une liste des classes. Dans notre cas, nous avons une seule classe, "car", et donc le paramètre "nc" est défini à 1 . Le fichier "data\_custom.yaml" est crucial pour que notre modèle puisse accéder aux données d'entraînement et de validation et comprendre les informations nécessaires pour la détection des voitures.



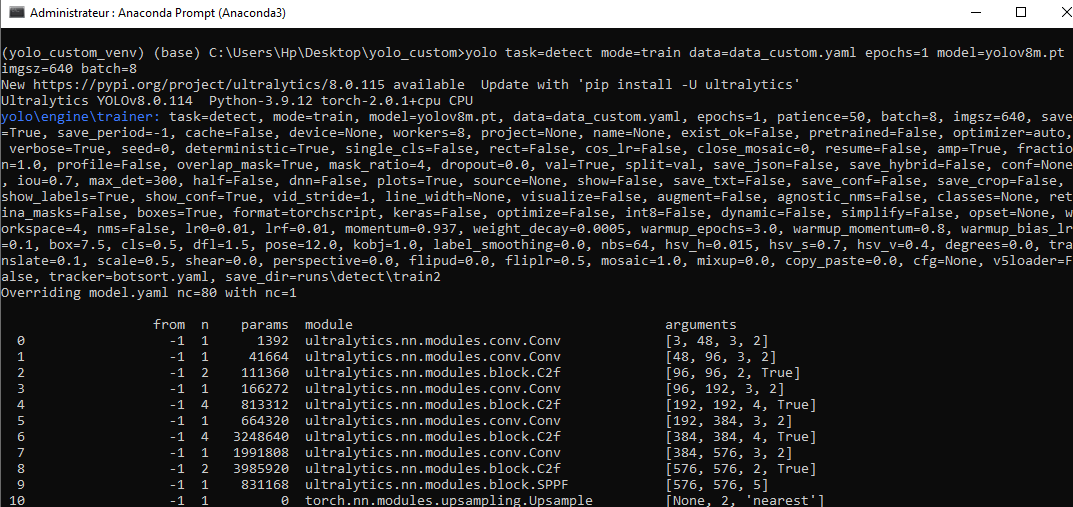
En résumé, la préparation du dataset implique l'organisation des données dans des dossiers spécifiques, contenant les images et les annotations correspondantes. De plus, un fichier .yaml est créé pour configurer les informations du dataset, telles que les chemins des dossiers et les détails sur les classes. Cette préparation soigneuse du dataset garantit une structure cohérente et permet à notre modèle YOLO de traiter efficacement les données d'entraînement et de validation.

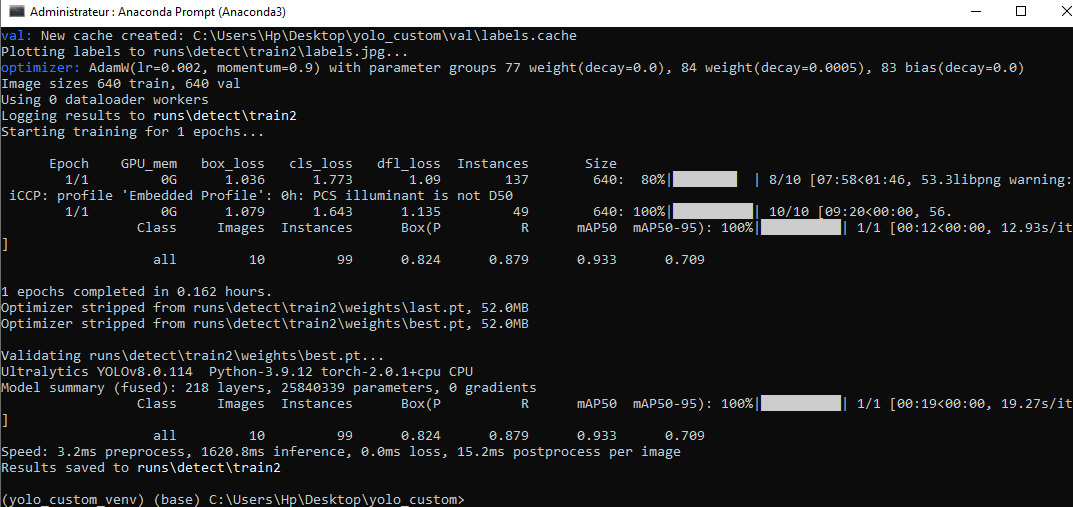
1. **Entraînement et Validation du modèle YOLO :**

Pour l'entraînement du modèle YOLO, nous avons utilisé la bibliothèque Ultralytics, qui facilite l'implémentation et l'entraînement de modèles de détection d'objets. Après avoir importé cette bibliothèque dans notre projet, nous avons chargé le modèle pré-entraîné yolov8n.pt, qui est une version du modèle YOLO utilisée pour notre tâche de détection de voitures.



Ensuite, nous avons procédé à l'entraînement du modèle sur notre propre dataset en utilisant la fonction "train" fournie par la bibliothèque Ultralytics. Dans les paramètres de cette fonction, nous avons spécifié le chemin du fichier data\_custom.yaml, qui contient les informations sur notre dataset personnalisé, y compris les chemins vers les dossiers d'entraînement et de validation.

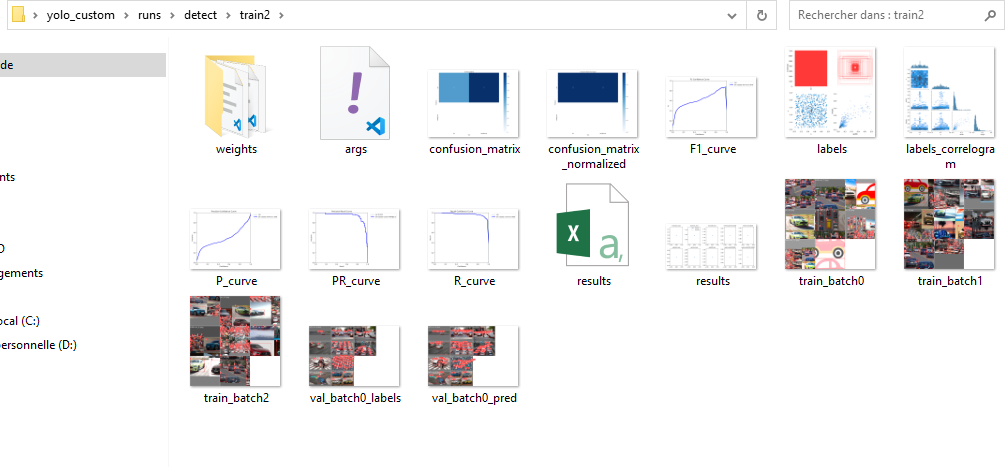
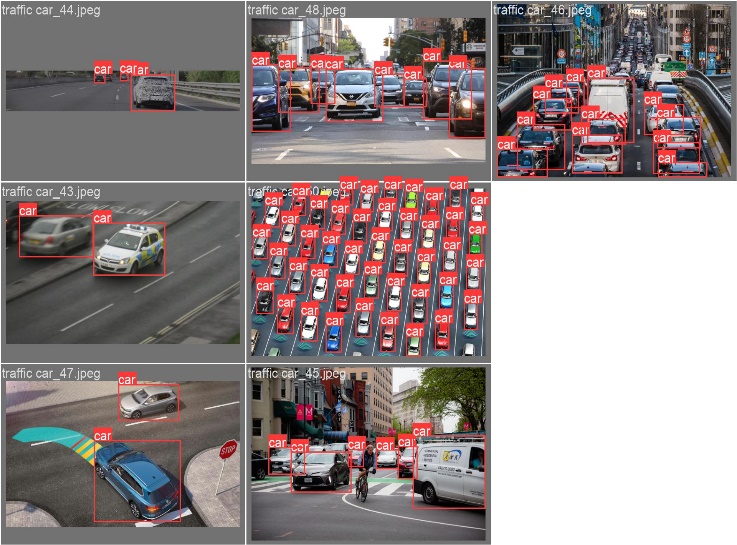


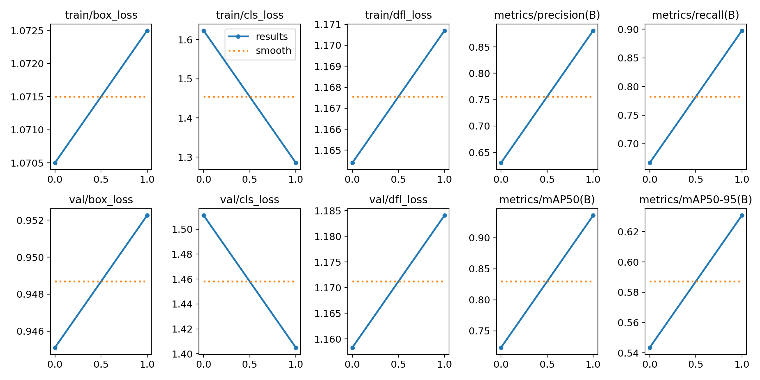
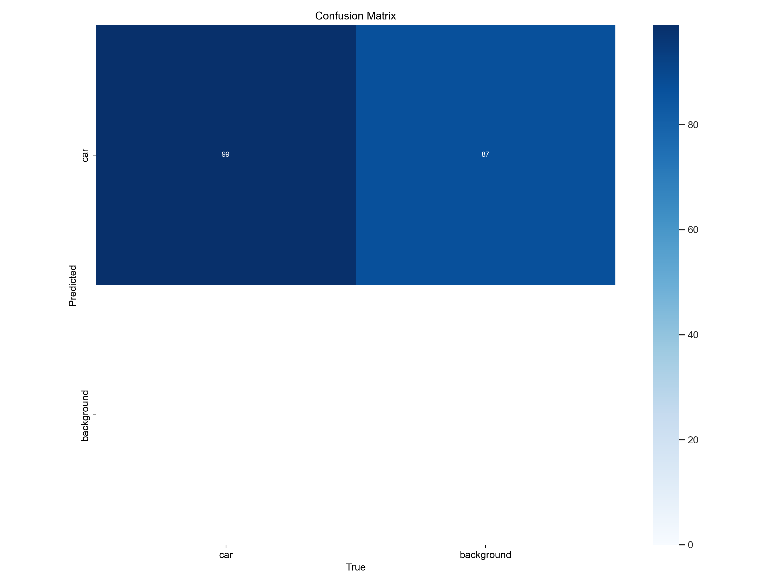


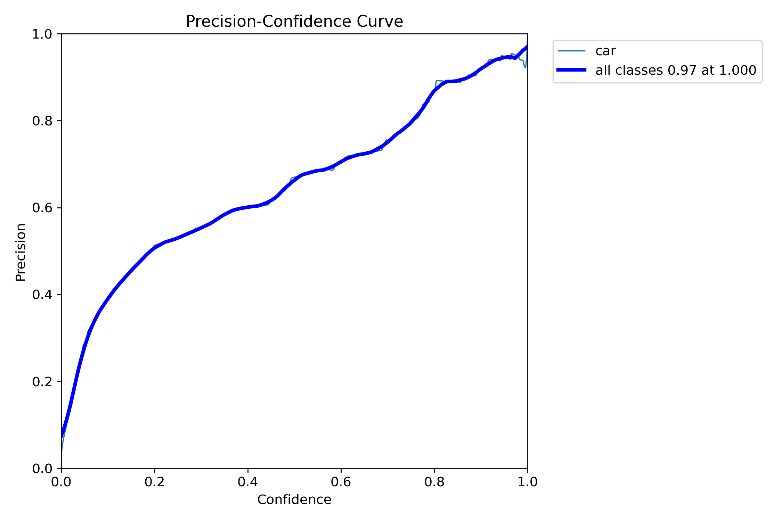
Pendant l'entraînement, nous avons eu la possibilité de fixer d'autres paramètres tels que le nombre d'époques (epochs), qui détermine le nombre de fois que le modèle parcourt l'ensemble des données d'entraînement. Ces paramètres peuvent être ajustés en fonction des besoins spécifiques de notre projet.

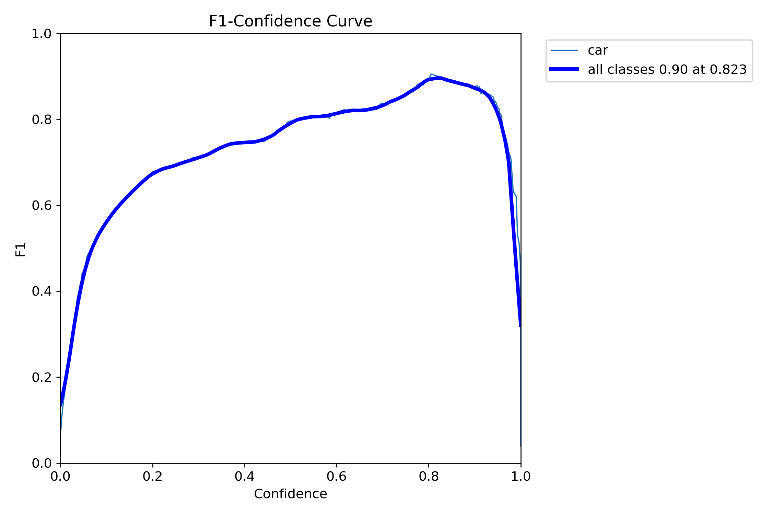
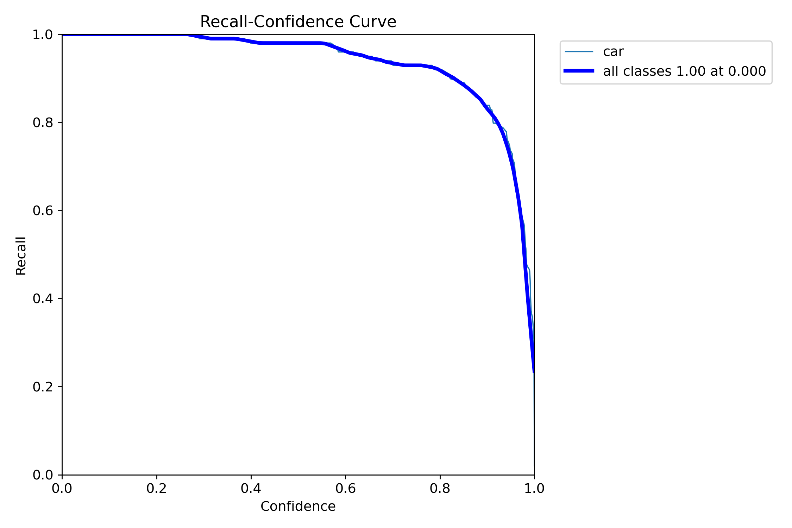
Après l'entraînement, l'algorithme YOLO effectue automatiquement la validation du modèle sur les données de validation mentionnées dans le fichier data\_custom.yaml. Il évalue les performances du modèle en termes de précision, de rappel et de score F1, ce qui nous permet d'évaluer son efficacité dans la détection des voitures.

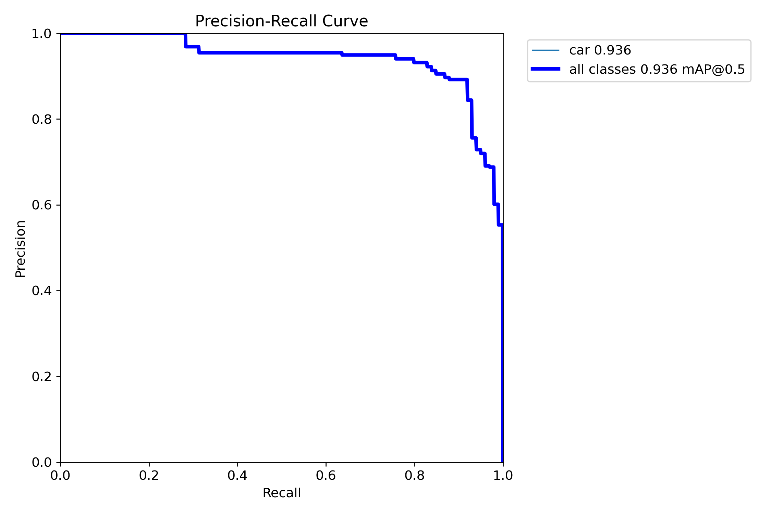
Tous les résultats de l'entraînement et de la validation, y compris les métriques de performance, les courbes d'apprentissage et les images avec les boîtes englobantes prédites, sont stockés dans un dossier spécifique appelé "runs". Ce dossier contient des sous-dossiers correspondant à chaque exécution d'entraînement, ce qui nous permet de suivre et d'analyser les résultats obtenus à différentes étapes du projet.







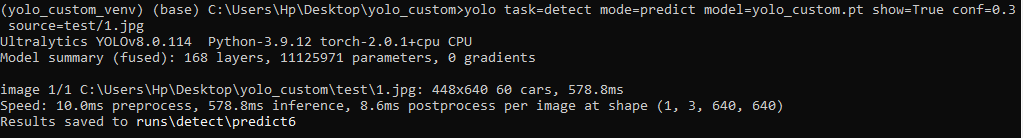




Grâce à l'utilisation de la bibliothèque Ultralytics et au processus d'entraînement bien défini, nous avons pu entraîner notre modèle YOLO sur notre propre dataset et évaluer ses performances sur les données de validation. Cette étape essentielle nous permet d'améliorer et d'ajuster notre modèle pour obtenir des résultats de détection de voitures plus précis et fiables.

1. **Tests sur des images et des vidéos :**

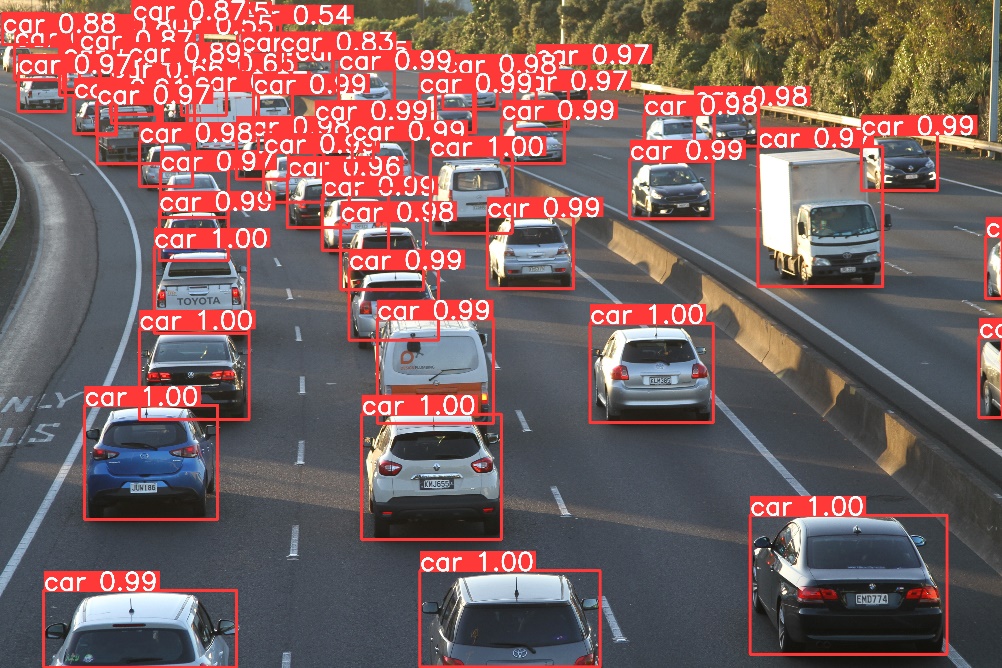
Une fois que notre modèle YOLO a été entraîné, nous avons procédé à des tests sur des images et des vidéos pour évaluer sa capacité à détecter les voitures. Pour effectuer ces tests, nous avons utilisé la fonction "predict" qui est fournie par la bibliothèque Ultralytics. Lorsque nous appliquons cette fonction à une image ou une vidéo, elle retourne une variable appelée "results" qui contient une liste de boîtes englobantes représentant les objets détectés.



Voilà l’image **test/1.jpg** :



Pour visualiser les résultats, nous avons utilisé la commande "print(results[0].boxes.boxes)". Cette commande nous a permis d'afficher les coordonnées des boîtes englobantes détectées, ainsi que leur score de confiance et leur classe. Chaque boîte englobante est représentée par une liste au format [x1, y1, x2, y2, score, label], où (x1, y1) et (x2, y2) sont les coordonnées des coins supérieur gauche et inférieur droit de la boîte englobante, respectivement.



Grâce à ces tests sur les images et les vidéos, nous avons pu observer les performances de notre modèle dans la détection des voitures. Les boîtes englobantes fournies par la fonction "predict" nous ont permis de localiser précisément les objets détectés dans chaque image ou chaque trame vidéo, et les scores de confiance associés nous ont donné une indication de la fiabilité de chaque détection. Ces résultats nous ont permis de quantifier la performance de notre modèle et d'identifier les domaines où des améliorations peuvent être apportées.

1. **Conclusion :**

En conclusion, notre projet de détection de voitures avec YOLO a produit des résultats prometteurs. Lors de l'évaluation de notre modèle, nous avons constaté une bonne précision de détection, avec des boîtes englobantes correctement positionnées autour des voitures dans les images et les vidéos testées. Le modèle a été capable d'identifier avec précision les voitures présentes, ce qui constitue une étape importante dans l'application de la détection d'objets dans des scénarios réels.

En ce qui concerne les performances, nous avons également observé une vitesse de traitement satisfaisante, ce qui rend le modèle utilisable en temps réel dans certains cas. Cependant, il convient de noter que la vitesse peut varier en fonction de la puissance de calcul disponible et de la taille du modèle utilisé.

Malgré ces résultats encourageants, notre modèle présente certaines limitations. Il peut y avoir des situations où la détection des voitures peut échouer, notamment en présence d'occlusions, de faible luminosité ou de perspectives complexes. De plus, la détection peut également être influencée par la qualité du dataset utilisé, y compris le nombre d'images annotées et la diversité des scènes représentées.

Ce projet nous a permis d'acquérir une expérience précieuse dans plusieurs aspects de la détection d'objets. Nous avons appris à collecter des données pertinentes en utilisant la bibliothèque simple\_image\_download, à les annoter avec l'outil labelImg, et à préparer notre dataset pour l'entraînement du modèle YOLO. Nous avons également acquis des compétences en utilisant la bibliothèque Ultralytics pour l'entraînement, la validation et les tests du modèle.

En ce qui concerne les perspectives futures, il existe plusieurs pistes que nous pourrions explorer pour améliorer davantage notre modèle de détection de voitures. Tout d'abord, l'augmentation des données peut être envisagée pour accroître la diversité des scènes et des situations de conduite représentées dans le dataset. Cela pourrait aider à renforcer la capacité du modèle à généraliser et à détecter les voitures dans des conditions variées.

De plus, l'optimisation des hyperparamètres peut être effectuée pour trouver la combinaison optimale de paramètres d'entraînement, tels que le taux d'apprentissage, la taille des mini-lots, etc. Cela peut aider à améliorer les performances du modèle en termes de précision et de temps de traitement.

Enfin, il serait intéressant d'explorer d'autres architectures de détection d'objets, en dehors de YOLO, pour comparer leurs performances et déterminer si une architecture différente peut mieux répondre aux besoins spécifiques de notre projet de détection de voitures.

Dans l'ensemble, ce projet nous a permis de plonger dans le domaine passionnant de la détection d'objets avec YOLO et d'obtenir des résultats prometteurs dans la détection de voitures. Les compétences et les connaissances acquises au cours de ce projet nous serviront de base solide pour poursuivre nos explorations et nos améliorations futures dans ce domaine en constante évolution.

1. **Références :**

<https://docs.ultralytics.com/>

<https://inside-machinelearning.com/yolov8/>

<https://www.youtube.com/watch?v=Ixt7YpMRg5I&list=PL1FZnkj4ad1PFJTjW4mWpHZhzgJinkNV0>

https://www.youtube.com/watch?v=m9fH9OWn8YM&list=PL1FZnkj4ad1M1em9-NCa9GDC5TGX70ztv