Projet de Détection d'Objets avec YOLO et OpenCV

# Introduction

La détection d'objets est une technique clé dans le domaine de la vision par ordinateur, qui consiste à localiser et identifier des objets spécifiques dans une image ou une vidéo.  
Les progrès dans les modèles d'apprentissage automatique et de vision par ordinateur, tels que YOLO (You Only Look Once), permettent désormais de réaliser cette tâche avec une grande précision et en temps réel.  
Dans ce projet, nous avons utilisé YOLOv8, l'une des dernières versions de ce modèle, pour détecter des objets dans des vidéos capturées en temps réel ou enregistrées.  
Nous avons intégré ce modèle avec OpenCV pour gérer le traitement des images et l'affichage des résultats.

# Objectif du Projet

L'objectif de ce projet est de développer une application capable de détecter et d'annoter des objets dans une vidéo en temps réel ou à partir d'une vidéo enregistrée.  
L'application doit être capable de traiter les images, d'identifier les objets présents, et de dessiner des boîtes de délimitation autour de chaque objet détecté avec un label correspondant à sa classe et à son niveau de confiance.  
En outre, nous avons apporté des améliorations au projet de base pour optimiser l'affichage des résultats et améliorer l'expérience utilisateur.

# Étapes du Projet

## 1. Installation des Bibliothèques Nécessaires

Le projet repose sur deux bibliothèques principales : OpenCV et Ultralytics (qui contient YOLO). OpenCV est utilisé pour manipuler les images et vidéos, tandis que la bibliothèque Ultralytics permet de charger et d'utiliser le modèle YOLO.  
  
Les bibliothèques peuvent être installées via pip avec les commandes suivantes :  
bash  
pip install opencv-python  
pip install ultralytics  
  
Après l'installation, nous créons un fichier Python (main.py) et nous téléchargeons les vidéos à utiliser pour les tests.

## 2. Chargement du Modèle YOLO

Le modèle YOLO que nous utilisons est YOLOv8, qui est une version plus récente du modèle, offrant une meilleure précision tout en étant efficace en termes de vitesse.   
Nous utilisons les poids pré-entraînés contenus dans le fichier yolov8s.pt qui sont téléchargés automatiquement lors de l'exécution du programme.  
  
python  
yolo = YOLO('yolov8s.pt')  
  
Ce modèle est conçu pour détecter des objets parmi 80 classes différentes, comme les personnes, les voitures, les camions, etc.  
Le modèle est particulièrement efficace pour les tâches de détection en temps réel.

## 3. Choix de la Source Vidéo

L'un des principaux défis de ce projet est de permettre à l'utilisateur de choisir entre l'utilisation de la caméra de son ordinateur pour une détection en temps réel, ou l'utilisation d'une vidéo enregistrée.  
Le code ci-dessous montre comment nous avons implémenté cette fonctionnalité :  
  
python  
use\_camera = False # Choisir True pour la caméra, False pour une vidéo  
video\_path = 'path\_to\_your\_video.mp4' # Chemin vers la vidéo  
if use\_camera:  
 videoCap = cv2.VideoCapture(0)  
else:  
 videoCap = cv2.VideoCapture(video\_path)  
  
En fonction de l'option choisie, le programme capture soit des images de la caméra, soit des images provenant d'une vidéo préenregistrée.

## 4. Redimensionnement et Traitement des Images

Une fois que la source vidéo est choisie, chaque image est extraite et redimensionnée pour être mieux affichée à l'utilisateur. Cela garantit que l'image n'est ni trop grande ni trop petite pour la fenêtre d'affichage.  
Nous avons utilisé la fonction cv2.resize() pour ajuster la taille des images comme suit :  
python  
frame = cv2.resize(frame, (800, 600))  
  
Cela permet un meilleur contrôle de l'affichage et une expérience utilisateur plus agréable.

## 5. Fonction de Détection d'Objets

Pour chaque image extraite de la vidéo, nous appliquons YOLO pour détecter les objets. YOLO traite l'image en une seule passe, ce qui le rend très rapide pour la détection en temps réel.  
Le modèle renvoie les coordonnées des boîtes englobantes (bounding boxes) ainsi que les classes des objets détectés et leur niveau de confiance.  
Nous avons utilisé un seuil de confiance de 40% pour décider si un objet devait être annoté ou non :  
python  
if box.conf[0] > 0.4:  
 # Annoter l'objet  
  
Cette partie du code permet de s'assurer que seuls les objets détectés avec une confiance suffisante sont annotés.

## 6. Gestion des Couleurs pour les Boîtes

Pour rendre le projet plus visuellement attractif, nous avons implémenté une fonction permettant d'assigner une couleur différente à chaque classe d'objet.  
La fonction getColours() génère des couleurs en se basant sur les indices des classes des objets, en ajoutant des variations pour chaque classe :  
python  
def getColours(cls\_num):  
 base\_colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255)]  
 color\_index = cls\_num % len(base\_colors)  
 increments = [(1, -2, 1), (-2, 1, -1), (1, -1, 2)]  
 color = [base\_colors[color\_index][i] + increments[color\_index][i] \*   
 (cls\_num // len(base\_colors)) % 256 for i in range(3)]  
 return tuple(color)  
  
Chaque objet détecté est entouré d'une boîte colorée pour le rendre plus facilement identifiable.

## 7. Boucle Principale pour la Détection en Temps Réel

La boucle principale du programme est chargée de capturer chaque image, de la traiter, et d'afficher les résultats à l'utilisateur.  
La boucle continue tant que l'utilisateur ne presse pas la touche 'q', ce qui permet un flux constant de détection en temps réel ou sur une vidéo enregistrée.  
Voici un extrait du code de la boucle principale :  
python  
while True:  
 ret, frame = videoCap.read()  
 if not ret:  
 print("Fin de la vidéo ou erreur de lecture")  
 break  
 frame = cv2.resize(frame, (800, 600))   
 results = yolo.track(frame, stream=True)  
 # Code de détection et d'affichage...  
 cv2.imshow('frame', frame)  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break

## 8. Affichage et Annotation des Objets

Pour chaque objet détecté, une boîte de délimitation est dessinée autour de l'objet, et le nom de la classe ainsi que le niveau de confiance sont affichés au-dessus de la boîte.  
Nous utilisons la fonction cv2.putText() pour annoter l'image avec le nom de l'objet et son score de confiance :  
python  
cv2.putText(frame, f'{classes\_names[int(box.cls[0])]} {box.conf[0]:.2f}', (x1, y1), cv2.FONT\_HERSHEY\_SIMPLEX, 1, colour, 2)  
  
L'image annotée est ensuite affichée dans une fenêtre.

## 9. Libération des Ressources

Une fois la vidéo terminée ou si l'utilisateur quitte la détection en temps réel, il est important de libérer les ressources allouées au processus.  
Nous fermons la capture vidéo et détruisons toutes les fenêtres ouvertes avec les commandes suivantes :  
python  
videoCap.release()  
cv2.destroyAllWindows()

# Conclusion

Grâce à ce projet, nous avons démontré comment utiliser YOLOv8 pour la détection d'objets dans des vidéos en temps réel ou enregistrées.   
Nous avons appris à capturer et traiter des vidéos à l'aide d'OpenCV, et à afficher des résultats de détection précis et en temps réel.   
Les modifications que nous avons apportées, telles que le redimensionnement de l'image et l'ajout de couleurs spécifiques pour chaque classe d'objets, ont permis d'améliorer l'interface utilisateur et de rendre les résultats plus compréhensibles.  
Ce projet constitue une excellente introduction à l'utilisation de modèles d'apprentissage profond pour des applications de vision par ordinateur.