



Université Mohammed V  
École nationale supérieure d'informatique et  
des systèmes  
Département de génie logiciel



Rapport de projet de fin de deuxième année

---

# Techniques basées sur le Deep-learning pour la reconnaissance d'objets en vue de leur suivi sur des images/vidéos aériennes

---

Réalisé par :

**Mehdi CHARIFE**  
**YASSINE OUHADI**

Encadré par :

**Prof. HDIOUD Boutaina**

Année universitaire : 2022-2023

## **Dédicace**

À Nos chers parents en signe de témoignage d'un grand respect et d'une profonde reconnaissance pour les sacrifices et les efforts qu'ils ont déployés pour notre education et notre formation.

À nos chers frères et sœurs.

À tous nos amis et collègues.

À tous nos professeurs.

À toute la famille.

Aucun mot ou expression ne serait valoriser les sacrifices que vous avez consentis pour notre bien-être et notre études.

## **Remerciements**

Nous tenons à exprimer nos sincères remerciements à notre professeur encadrant, Mme. Boutaina HDIOUD, pour son soutien et son accompagnement tout au long de ce projet. Sa disponibilité, ses conseils et ses encouragements ont été précieux pour mener à bien cette réalisation.

Nous souhaitons également remercier l'ensemble de l'équipe pédagogique du département de Génie Logiciel de l'ENSIAS pour la qualité de leur enseignement, leur expertise et leur disponibilité.

Enfin, nous tenons à exprimer notre gratitude envers toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce projet, notamment nos camarades de groupe, ainsi que nos amis et notre famille pour leur soutien et leur encouragement.

## Résumé

Le projet que nous avons réalisé est une application de détection d'objets aériennes basée sur l'algorithme YOLO et utilisant l'algorithme de suivi Deep SORT. Cette application permet aux utilisateurs de télécharger des images ou des vidéos et d'effectuer une détection précise des objets présents dans ces médias.

Dans ce projet, nous avons réalisé une intégration harmonieuse entre l'algorithme YOLO, l'algorithme de suivi Deep SORT, Flask et React pour offrir une expérience utilisateur optimale.

# Table des matières

Introduction générale . . . . .	9
<b>1 Architecture</b>	<b>10</b>
1.1 Introduction . . . . .	10
1.2 Architecture . . . . .	10
1.3 Conclusion . . . . .	11
<b>2 DéTECTEUR d'objets</b>	<b>12</b>
2.1 Introduction . . . . .	12
2.2 Vue d'ensemble de l'algorithme YOLO . . . . .	12
2.3 Entrainement du modèle . . . . .	13
2.4 Évaluation du modèle . . . . .	16
2.5 Conclusion . . . . .	21
<b>3 Suiveur</b>	<b>22</b>
3.1 Introduction . . . . .	22
3.2 Aperçu de l'algorithme DeepSort . . . . .	22
3.3 Implémentation . . . . .	24
3.4 Conclusion . . . . .	27
<b>4 Application web</b>	<b>28</b>
4.1 Environnement du travail . . . . .	28

4.2 Choix des outils de développement . . . . .	29
4.3 Interfaces . . . . .	30
4.4 Conclusion . . . . .	32
Conclusion . . . . .	33

# Table des figures

1.1	Architecture YOLO . . . . .	10
2.1	Architecture YOLO . . . . .	12
2.2	Image issue du aerial airport dataset . . . . .	14
2.3	Google Colab . . . . .	15
2.4	Environnement d'entraînement . . . . .	15
2.5	Courbe Précision-Rappel . . . . .	16
2.6	Courbe Précision-Confidence . . . . .	17
2.7	Courbe Rappel-Confidence . . . . .	18
2.8	Courbe Rappel-Confidence . . . . .	19
2.9	Confusion Matrix . . . . .	20
3.1	Architecture de DeepSort . . . . .	22
3.2	Objets associés par l'algorithme hongrois . . . . .	23
4.1	Visual Studio Code . . . . .	28
4.2	React JS . . . . .	29
4.3	React JS . . . . .	29
4.4	Importer une image ou une vidéo . . . . .	30
4.5	Sélectionnez les objets à détecter . . . . .	30
4.6	Détection d'objets en cour . . . . .	31

4.7 Détection d'objets réussite . . . . .	31
---	----

## Introduction générale

Avec la généralisation de l'utilisation des images et vidéos aériennes, l'analyse du mouvement et la détection d'objets dans ces médias sont devenues des outils indispensables pour de nombreuses applications, telles que la surveillance aérienne, l'analyse environnementale, la cartographie et la navigation autonome.

La détection et le suivi d'objets dans les images et vidéos aériennes présentent des défis uniques en raison de la nature spécifique de ces médias. Les objets peuvent être de petite taille, en mouvement rapide et soumis à des déformations dues aux changements de perspective. De plus, les conditions d'éclairage, les variations de l'arrière-plan et les occlusions peuvent rendre la détection et le suivi plus complexes.

Le but de notre projet est de développer un système de détection et de suivi d'objets en mouvement dans des images et vidéos aériennes. Nous utilisons YOLO pour la détection initiale des objets, qui offre une approche efficace pour la localisation et la classification simultanées des objets dans une image ou une vidéo. De plus, nous utilisons l'algorithme Deep SORT pour le suivi des objets détectés, permettant ainsi de suivre et d'attribuer des identifiants uniques aux objets au fil du temps.

En utilisant la combinaison de l'algorithme YOLO et de l'algorithme Deep SORT, notre système peut détecter et suivre efficacement les objets en mouvement dans les images et vidéos aériennes.

Dans ce rapport, nous présenterons les différentes étapes de la réalisation de notre projet, de la collecte et la préparation des données, à l'entraînement des modèles d'apprentissage en profondeur, en passant par l'implémentation de l'architecture logicielle utilisant Flask API pour le backend et React pour le frontend. Nous discuterons également des performances de notre système, des limitations rencontrées et des perspectives d'amélioration.

# Chapitre 1

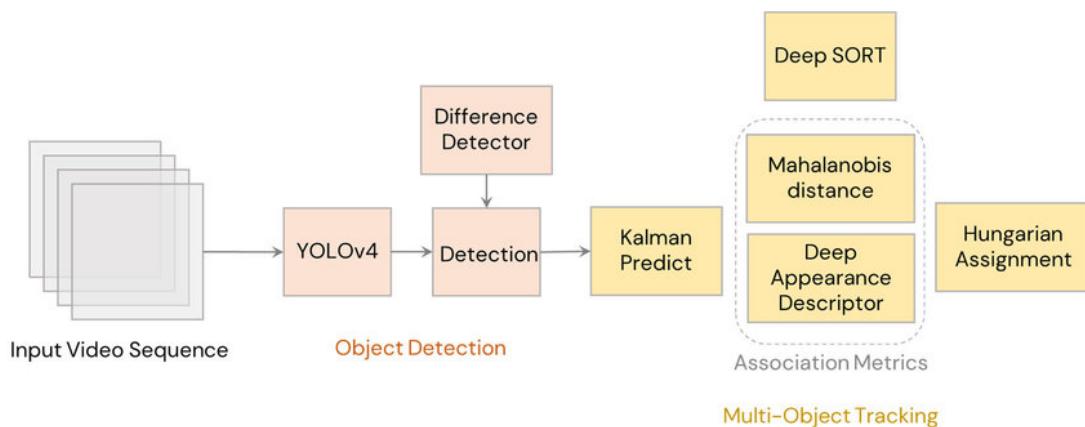
# Architecture

## 1.1 Introduction

La détection et le suivi des objets dans notre projet se basent principalement sur la décomposition des vidéos en frames individuelles et la détection d'objets dans chaque image.

## 1.2 Architecture

Notre système de détection en temps réel est constitué de 4 composantes majeures : un détecteur d'objets (YOLO), un suiveur (DeepSort), un extracteur de caractéristiques, et un segmenteur de vidéo (OpenCV).



**Figure 1.1 – Architecture YOLO**

Lors de son entrée dans le système, la vidéo est segmentée en plusieurs frames utilisant OpenCV. Ces frames font ensuite l'objet d'une boucle dont, à chaque itération :

1. Le détecteur traite le frame courant et produit une liste des détections ;
2. Les détections sont fournis au suiveur de DeepSort
3. Le suiveur fait appel à l'extracteur des caractéristiques pour générer des vecteurs descriptifs de chaque objet détecté.
4. A L'aide de l'algorithme Hongrois, les détections du frame courant sont appariés (matched) aux pistes des frames antérieurs. Si une détection ne correspond à aucun piste, un nouveau est créé. Les pistes qui ne correspondent à aucune détection dans le frame courant correspondent à des objets qui ont quitté la zone englobée par la vidéo.
5. OpenCV est utilisée pour dessiner des rectangles englobant les objets détectés dans le frame courant. La couleur par laquelle un certain frame est dessiné est uniquement déterminée par la piste appariée à la détection courante.

Cette approche permet de détecter et de suivre en temps réel les objets dans la vidéo, fournissant ainsi une analyse précise de leur mouvement et de leur comportement. La combinaison du détecteur d'objets YOLO, du suiveur DeepSort, de l'extracteur de caractéristiques et du segmenteur de vidéo OpenCV offre une architecture puissante pour diverses applications telles que la surveillance, la sécurité, l'analyse de comportement, la réalité augmentée et bien d'autres.

### 1.3 Conclusion

En conclusion, l'architecture mise en place pour notre système de détection en temps réel, combinant le détecteur d'objets YOLO, le suiveur DeepSort, l'extracteur de caractéristiques et le segmenteur de vidéo OpenCV, offre une solution complète et puissante pour la détection et le suivi précis des objets dans les vidéos. Cette architecture modulaire permet une intégration harmonieuse des différentes composantes, permettant ainsi une analyse continue des mouvements et des comportements des objets. Grâce à cette approche, notre système permet d'obtenir des résultats en temps réel, ce qui est essentiel pour les applications nécessitant une réponse instantanée. De plus, cette architecture offre une grande flexibilité et extensibilité, permettant d'adapter facilement le système à différents scénarios et d'incorporer de nouvelles fonctionnalités à l'avenir.

# Chapitre 2

# Détecteur d'objets

## 2.1 Introduction

Afin de pouvoir détecter les objets figurant dans une image donnée, nous avons opté à utiliser le modèle YOLO. Ce dernier est un modèle populaire de détection d'objets connu pour sa rapidité et précision. Il a été initialement introduit par Joseph Redmon et al. en 2016 et a depuis subi plusieurs itérations, la dernière étant YOLO v8.

## 2.2 Vue d'ensemble de l'algorithme YOLO

L'algorithme YOLO prend une image en entrée, puis utilise un réseau neuronal convolutif profond simple pour détecter des objets dans l'image. L'architecture du modèle CNN qui forme la base de YOLO est présentée ci-dessous :

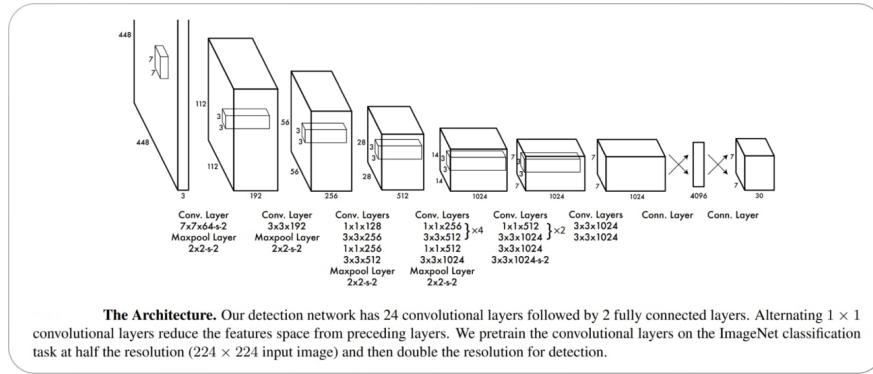


Figure 2.1 – Architecture YOLO

Les vingt premières couches de convolution du modèle sont pré-entraînées à l'aide d'ImageNet en ajoutant une couche de pooling moyenne temporelle et une couche entièrement connectée.

Ensuite, ce modèle pré-entraîné est converti pour effectuer la détection, car des recherches antérieures ont montré que l'ajout de couches de convolution et de connexions à un réseau pré-entraîné améliore les performances. La dernière couche entièrement connectée de YOLO prédit à la fois les probabilités de classe et les coordonnées des boîtes englobantes.

YOLO divise une image d'entrée en une grille  $S \times S$ . Si le centre d'un objet se trouve dans une cellule de la grille, cette cellule est responsable de la détection de cet objet. Chaque cellule de la grille prédit  $B$  boîtes englobantes et des scores de confiance pour ces boîtes. Ces scores de confiance reflètent la confiance du modèle dans le fait que la boîte contient un objet et l'exactitude avec laquelle il prédit la boîte.

YOLO prédit plusieurs boîtes englobantes par cellule de grille. Pendant l'entraînement, nous ne voulons qu'un seul prédicteur de boîte englobante soit responsable de chaque objet. YOLO attribue un prédicteur comme "responsable" de la prédiction d'un objet en se basant sur la prédiction ayant le meilleur IOU (intersection over union) actuel avec la vérité terrain. Cela conduit à une spécialisation entre les prédicteurs de boîtes englobantes. Chaque prédicteur devient meilleur pour prédire certaines tailles, rapports d'aspect ou classes d'objets, améliorant ainsi le score de rappel global.

Une technique clé utilisée dans les modèles YOLO est la suppression des non-maximaux (NMS). NMS est une étape de post-traitement utilisée pour améliorer la précision et l'efficacité de la détection d'objets. En détection d'objets, il est courant que plusieurs boîtes englobantes soient générées pour un seul objet dans une image. Ces boîtes englobantes peuvent se chevaucher ou être situées à des positions différentes, mais elles représentent toutes le même objet. NMS est utilisé pour identifier et supprimer les boîtes englobantes redondantes ou incorrectes, et pour produire une seule boîte englobante pour chaque objet de l'image.

## 2.3 Entrainement du modèle

Afin d'avoir des résultats optimales pour le cas qui nous occupe, nous avons emprunté un modèle pré-entraîné de YOLO (yolov8n.pt) que nous avons ensuite entraîné sur des captures aériennes. Les images utilisées pour l'entraînement parviennent du dataset d'un *Aerial Airport* et *COCO (Common Objects in COntext)*

### 2.3.1 Ensemble des données

L'ensemble des données sur lequel le modèle a été entrainé se compose d'environ 2365 images avec leurs annotations associées. Pour chaque fichier d'image dans l'ensemble des données, il existe un fichier d'annotation correspondant détaillant les coordonnées des objets dans l'image ainsi que leurs classes et scores. L'exemple suivant illustre la forme de ces fichiers :



**Figure 2.2** – Image issue du aerial airport dataset

---

036.txt

---

```
0 0.1666666666666666 0.1666666666666666 0.06 0.0533333333333334
0 0.4691666666666667 0.4041666666666667 0.0883333333333333 0.08166666666666667
0 0.8908333333333334 0.05916666666666666 0.1083333333333334 0.09166666666666666
0 0.4216666666666667 0.7075 0.08 0.0683333333333333
0 0.3141666666666665 0.790833333333334 0.12166666666666667 0.105
0 0.1191666666666667 0.9291666666666667 0.1416666666666666 0.1183333333333333
```

---

Le fichier 036.txt est composé de deux lignes, chacune correspondant à une detection unique. Chaque detection est exprimé sous la forme de 5 nombres :

1. Identifiant de la classe ;
2. Ordonnée du coin supérieur gauche du conteneur ;
3. Abscisse du coin supérieur gauche du rectangle englobant ;
4. Ordonnée du coin inférieur droit du rectangle englobante ;
5. Ordonnée du coin inférieur droit du rectangle englobante.

### 2.3.2 Environnement d'entraînement

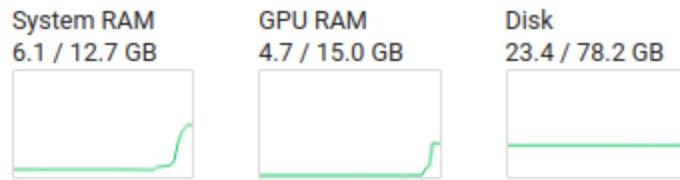
Pour l'entraînement du modèle, nous avons utilisé l'environnement offert par Google Colab. Google Colab est une plateforme en ligne gratuite qui permet d'exécuter du code Python et d'accéder à des ressources matérielles ou des unités de traitement tensoriel pour accélérer les calculs. Cette plateforme offre également la possibilité de collaborer avec d'autres utilisateurs en partageant facilement des notebooks.



**Figure 2.3 –** Google Colab

En utilisant Google Colab, nous avons pu bénéficier de la puissance de calcul des GPU pour accélérer le processus d'entraînement de notre modèle. Cette ressource nous a permis de traiter de grandes quantités de données et de réaliser des calculs intensifs plus rapidement que si nous avions utilisé des ressources locales.

Notre entraînement est caractérisé par 12 GB de RAM, 15 GB de GPU, et 78.2 GB d'espace disque :



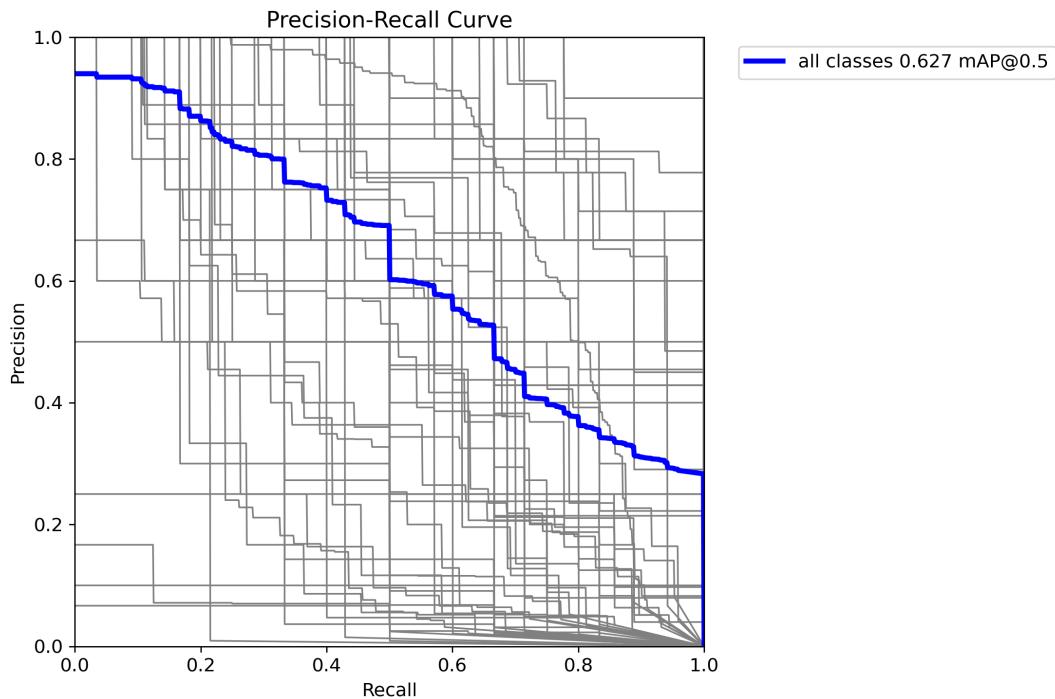
**Figure 2.4 –** Environnement d'entraînement

## 2.4 Évaluation du modèle

Pour évaluer la performance du modèle, YOLO permet de produire un nombre de graphes à la fin d'entraînement. Ces graphes peuvent être utilisés pour avoir une idée générale sur la performance du modèle à l'égard de détection des objets.

### 2.4.1 Courbe Précision-Rappel

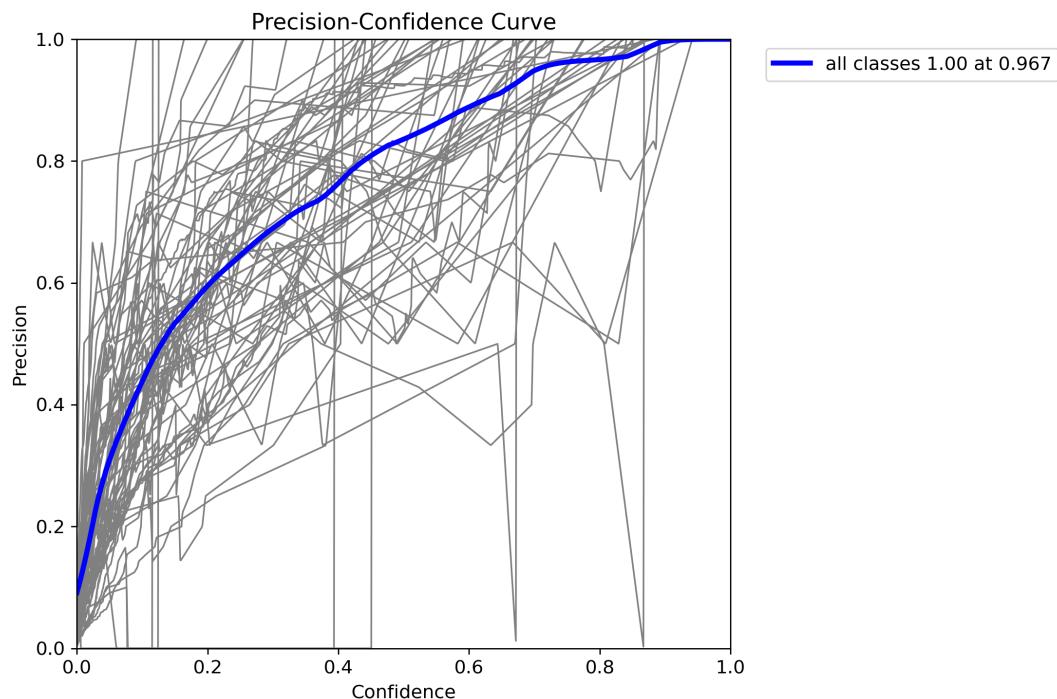
La courbe précision-rappel est un outil d'évaluation utilisé pour évaluer les performances d'un système ou d'une méthode de classification. Elle représente graphiquement la relation entre la précision et le rappel d'un système de classification binaire à différents seuils de décision.



**Figure 2.5 –** Courbe Précision-Rappel

## 2.4.2 Courbe Précision-Confidence

La courbe précision-confiance est un outil d'analyse utilisé pour évaluer la qualité des prédictions faites par un modèle ou un algorithme de classification. Cette courbe représente graphiquement la relation entre la précision et le niveau de confiance associé aux prédictions effectuées par le modèle.

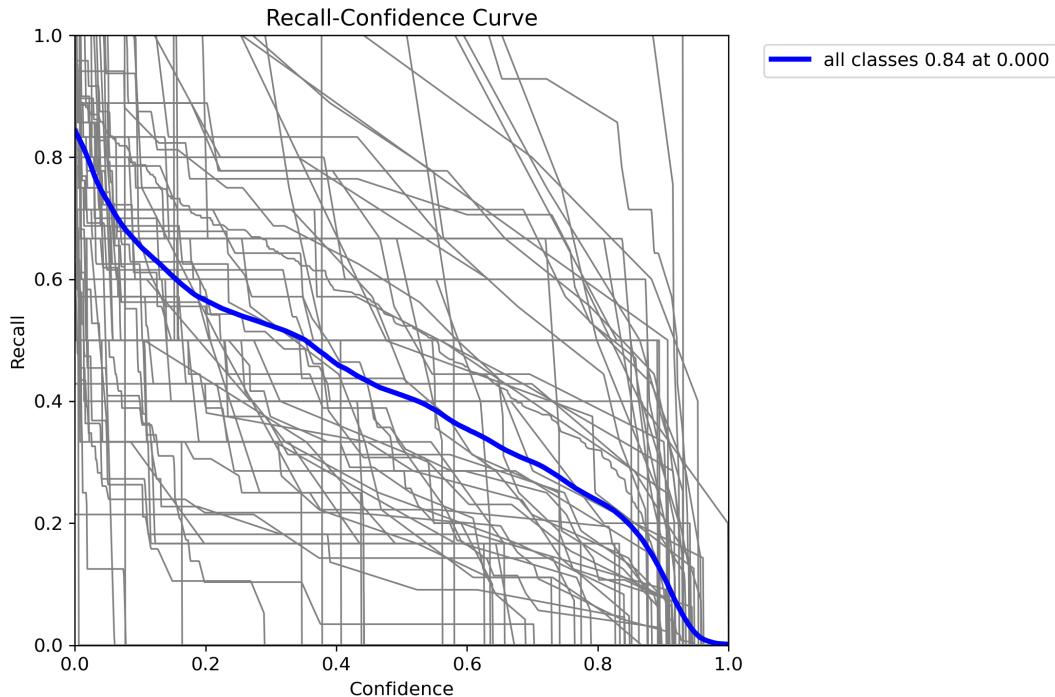


**Figure 2.6 – Courbe Précision-Confidence**

En traçant la courbe précision-confiance, on peut observer comment la précision évolue en fonction du niveau de confiance choisi. Cela permet d'analyser la fiabilité du modèle à différents niveaux de confiance et d'identifier le seuil de confiance optimal en fonction des objectifs spécifiques de la classification.

### 2.4.3 Courbe Rappel-Confiance

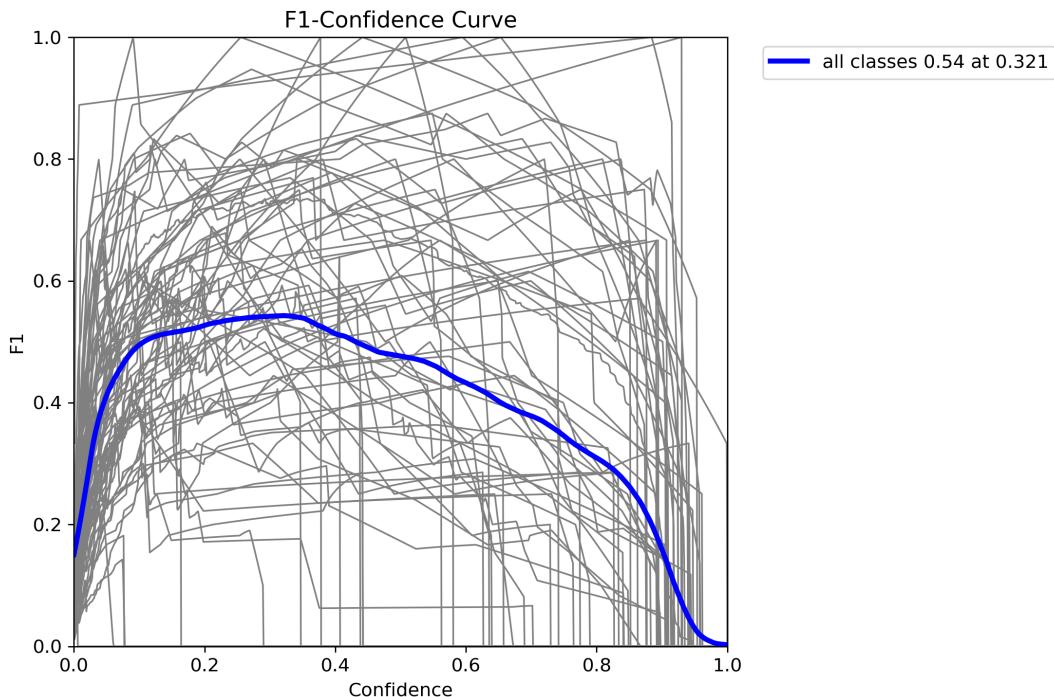
La courbe rappel-confiance est un outil d'analyse utilisé pour évaluer la sensibilité d'un modèle ou d'un algorithme de classification par rapport à différents niveaux de confiance associés aux prédictions. Cette courbe représente graphiquement la relation entre le rappel et le niveau de confiance des prédictions effectuées par le modèle.



**Figure 2.7 – Courbe Rappel-Confidence**

#### 2.4.4 Courbe F1-Confiance

La courbe F1-confiance permet d'évaluer les performances de l'algorithme à différents niveaux de confiance, ce qui permet de choisir un seuil approprié en fonction du score F1 souhaité. Elle offre des informations sur le compromis entre la précision et le rappel du modèle à différents niveaux de confiance.



**Figure 2.8 – Courbe Rappel-Confidence**

#### 2.4.5 Précision moyenne globale

La précision moyenne globale est une mesure couramment utilisée pour évaluer la performance d'un modèle ou d'un système de classification ou de détection. Elle peut être calculée utilisant le graphe précision-rappel (P-R).

Model summary fused: 168 layers, 3151904 parameters, 0 gradients						
Class	Images	Instances	BoxP	R	mAP50	mAP50–95: 100% 5/5
all	68	1018	0.837	0.699	0.797	0.361

## 2.4.6 Confusion Matrix

La matrice de confusion est un outil puissant pour évaluer la performance d'un modèle en comparant les prédictions faites par celui-ci avec les valeurs réelles des données. Elle fournit une représentation claire des résultats de classification et permet de calculer différentes métriques d'évaluation pour mesurer la précision et la fiabilité du modèle.

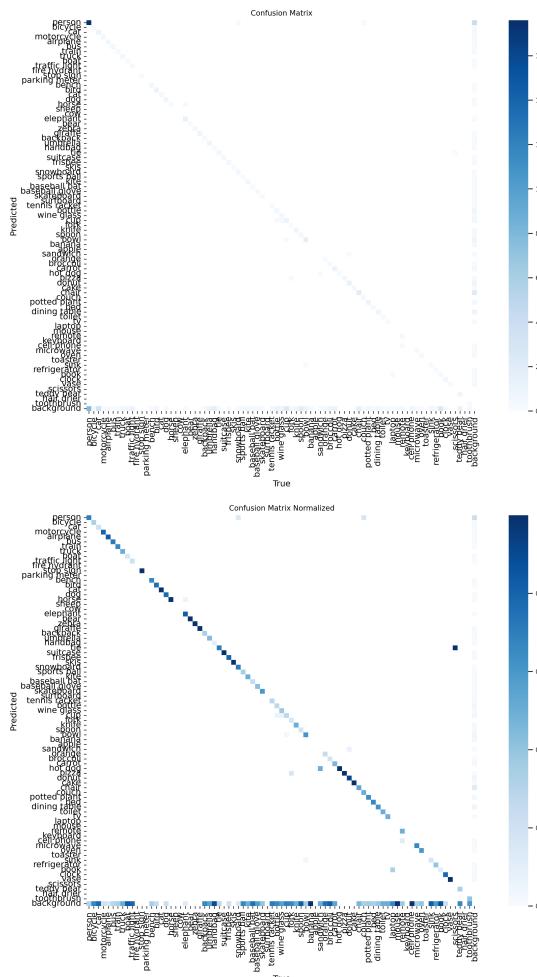


Figure 2.9 – Confusion Matrix

## **2.5 Conclusion**

Dans ce chapitre, nous avons examiné en détail le fonctionnement du détecteur d'objets YOLO. Ce détecteur nous a permis de localiser et de classifier les objets présents dans une image.

# Chapitre 3

# Suiveur

## 3.1 Introduction

Dans ce chapitre, nous nous concentrerons sur la partie suivante du processus de vision par ordinateur : le suivi en temps réel des objets détectés.

## 3.2 Aperçu de l'algorithme DeepSort

Pour réaliser le suivi des objets, nous avons utilisé l'algorithme DeepSort. C'est un algorithme de suivi des objets basé sur des réseaux de neurones profonds. Il associe des identités aux objets détectés en utilisant des informations de similarité spatiale et de caractéristiques visuelles. L'algorithme utilise un réseau de neurones pour extraire des descripteurs significatifs des objets détectés, qui sont ensuite utilisés pour calculer des scores de similarité et associer les objets entre les images successives.

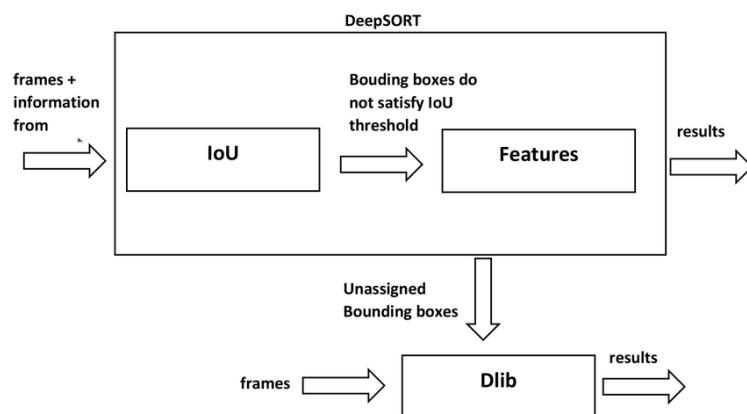
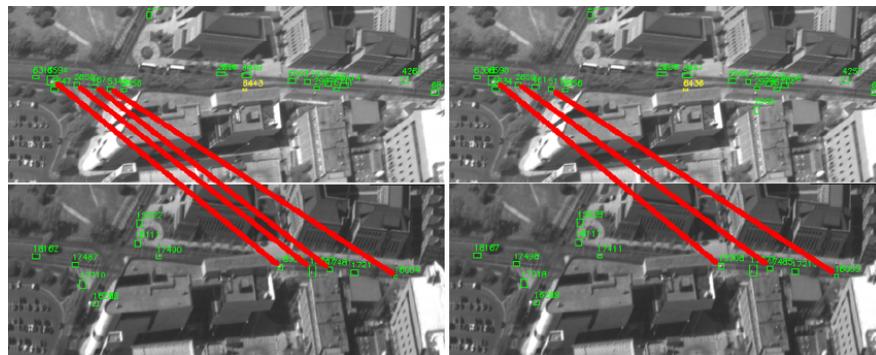


Figure 3.1 – Architecture de DeepSort

La première étape de l'algorithme consiste à extraire des descripteurs à partir des régions d'intérêt (ROI) des objets détectés par le modèle de YOLO. Ces descripteurs sont obtenus en faisant passer les ROI à travers un réseau de neurones pré-entraîné, tel qu'un réseau de neurones convolutifs (CNN). Le réseau de neurones apprend à extraire des caractéristiques discriminantes des objets, qui sont ensuite utilisées pour représenter les objets dans un espace de descripteurs.

Une fois que les descripteurs sont extraits, DeepSort utilise un algorithme de suivi basé sur le filtrage de Kalman pour prédire la position des objets dans les images suivantes. Le filtrage de Kalman est une technique utilisée pour estimer la position et la vitesse d'un objet en utilisant des observations passées et un modèle dynamique de l'objet. Cela permet de prédire la position probable des objets et d'associer les objets détectés dans les images suivantes avec les objets déjà suivis.

Ensuite, DeepSort calcule les scores de similarité entre les descripteurs des objets détectés dans l'image courante et ceux des objets déjà suivis. Ces scores de similarité sont généralement calculés à l'aide de mesures de distance, telles que la distance euclidienne ou la distance cosinus. Les objets avec les scores de similarité les plus élevés sont associés aux objets déjà suivis.



**Figure 3.2 – Objets associés par l'algorithme hongrois**

Cependant, il peut y avoir des cas où plusieurs objets détectés correspondent à un même objet suivi ou lorsque des objets détectés sont des faux positifs. Pour résoudre ce problème, DeepSort utilise un algorithme d'association de données appelé l'algorithme d'association de données de Joint Probabilistic Data Association (JPDA). Cet algorithme permet de gérer les associations ambiguës et les faux positifs en utilisant des modèles probabilistes pour estimer les probabilités d'association entre les objets détectés et les objets suivis.

### 3.3 Implémentation

Pour utiliser l'algorithme DeepSort, nous avons cloné leur répertoire disponible sur Github. Ce dernier comporte les classes et les fonctions nécessaires pour implémenter notre suiveur. Dans la suite nous allons présenter les parties du code écrites pour cette implémentation :

#### 3.3.1 Module utilisées

```
'''===== DeepSort ====='''
from deep_sort.deep_sort.tracker import Tracker
from deep_sort.deep_sort.nn_matching import NearestNeighborDistanceMetric
from deep_sort.tools.generate_detections import create_box_encoder
from deep_sort.deep_sort.detection import Detection

'''===== YOLO ====='''
from ultralytics import YOLO

'''===== OpenCV ====='''
import cv2

'''===== Utilités ====='''
import numpy as np
import random
import os
```

#### 3.3.2 Initialisation du tracker de DeepSort

```
similarity_metric = NearestNeighborDistanceMetric("cosine", 0.4, None)
tracker = Tracker(similarity_metric)
```

La première ligne initialise la mesure utilisé pour calculer la similarité entre les vecteurs des caractéristique de deux objets détectés. Dans notre cas, nous avons choisi d'utiliser la similarité cosinus pour effectuer les comparaison. Cette dernière est ensuite utilisée comme argument pour initialiser le suiveur.

### 3.3.3 Initialisation de l'extracteur des caractéristiques

Afin de pouvoir comparer les objets détectés dans les différents frames d'une vidéo, DeepSort fait recours à un modèle qui permet d'extraire les caractéristiques des objets détectés. Ces caractéristiques sont organisées dans des vecteurs multidimensionnelles.

```
feature_extractor = create_box_encoder("./deep_sort/mars-small128.pb",
                                         batch_size=1)
```

### 3.3.4 Initialisation du Lecteur et Scribeur de OpenCV

```
cap = cv2.VideoCapture(file_path)
ret, frame = cap.read()

inter_file_path = "outcoming/inter_" + filename
ret_file_path = "outcoming/" + filename
inter_file = open(inter_file_path, "x")
inter_file.close()
cap_out = cv2.VideoWriter(inter_file_path,
                          cv2.VideoWriter_fourcc(*'mp4v'),
                          cap.get(cv2.CAP_PROP_FPS),
                          (frame.shape[1], frame.shape[0]))
)
```

Le code utilise la bibliothèque OpenCV pour ouvrir un fichier vidéo à partir du chemin spécifié et lire le premier cadre. Ensuite, il crée un fichier intermédiaire vide et prépare un objet "VideoWriter" d'OpenCV pour écrire les images de sortie dans ce fichier intermédiaire

### 3.3.5 Traitement itératif des détections d'objets

```
while ret:
    for results in resultss:

        # Generating a list of Detection(s) for the current frame
        bboxes, features, scores = ([] for k in range(3))
        for bbox_wrapper in results.bboxes.data.tolist():
            min_x, min_y, max_x, max_y, score, class_id = bbox_wrapper

            tlwh = (min_x, min_y, max_x - min_x, max_y - min_y)
            bboxes.append(np.asarray(tlwh))
            scores.append(score)
        features = feature_extractor(frame, bboxes)

        detections = []
        for k in range(len(bboxes)):
            detections.append(Detection(bboxes[k], scores[k], features[k]))

        tracker.predict() # Feeding the Detection(s) to DeepSort's Tracker
        tracker.update(detections)

        # Adding the assigned (colored) detections to the frame
        for track in tracker.tracks:
            if track.track_id not in color_tracks:
                color = (random.randint(0, 255),
                          random.randint(0, 255),
                          random.randint(0, 255))
            )
            while color in color_tracks.values():
                color = (random.randint(0, 255),
                          random.randint(0, 255),
                          random.randint(0, 255))
            )
            color_tracks[track.track_id] = color

            cv2.rectangle(frame, (int(track.to_tlbr()[0]), int(track.to_tlbr()[1])),
                         (int(track.to_tlbr()[2]), int(track.to_tlbr([3])), color_tracks[track.track_id], 3
            )

        # Adding the updated frame to the output video
        cap_out.write(frame)

        # Getting the next frame
        ret, frame = cap.read()
        resultss = model(frame)
```

La boucle décrite effectue les opérations suivantes de manière itérative dans le contexte de la détection des objets :

1. Elle génère une liste de détections pour chaque image du flux vidéo en utilisant les résultats obtenus.
2. Ces détections sont ensuite fournies à un système de suivi (Tracker) appelé DeepSort pour prédiction et mise à jour.
3. Les détections attribuées (colorées) sont ajoutées à chaque image pour visualisation.
4. Enfin, l'image mise à jour est enregistrée dans une vidéo de sortie.

Cette boucle permet de traiter successivement chaque image du flux vidéo, d'effectuer la détection des objets, de les suivre et de générer une sortie visuelle annotée avec les détections colorées.

### **3.4 Conclusion**

En conclusion, l'utilisation de l'algorithme DeepSort a été une étape cruciale dans notre projet, nous permettant d'accomplir avec succès le suivi en temps réel des objets détectés. Ce chapitre nous a permis de comprendre les principes fondamentaux de l'algorithme.

# Chapitre 4

# Application web

Dans ce chapitre, nous avons réalisé une API qui nous permet de détecter les objets en utilisant des requêtes et des réponses. Nous avons également développé une interface web qui répond à nos besoins. Pour ce faire, nous avons utilisé des outils front-end et back-end pour la réalisation de cette application.

## 4.1 Environnement du travail

Pour la réalisation de l'application web, nous avons utilisé Visual Studio Code, un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS. Visual Studio Code offre de nombreuses fonctionnalités pratiques telles que le débogage, la mise en évidence de la syntaxe, la complétion intelligente du code, les snippets, la refactorisation du code, et une intégration native avec Git.



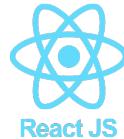
**Figure 4.1** – Visual Studio Code

## 4.2 Choix des outils de développement

Dans notre projet, nous avons mis en place une architecture client-serveur. Le serveur, basé sur Flask, agit en tant qu'API pour la détection d'objets à l'aide du modèle YOLO. Il reçoit les requêtes des clients et renvoie les résultats de détection. Le client, quant à lui, est l'interface utilisateur développée en React JS. Il communique avec le serveur en utilisant des requêtes HTTP pour envoyer les images ou les vidéos à analyser et pour recevoir les résultats de détection.

### 4.2.1 Front End

Nous avons utilisé React JS pour créer l'interface utilisateur de notre application. Nous avons choisi en raison de sa flexibilité, de sa performance et de sa richesse en fonctionnalités. En utilisant React JS.



**Figure 4.2 – React JS**

### 4.2.2 Back End

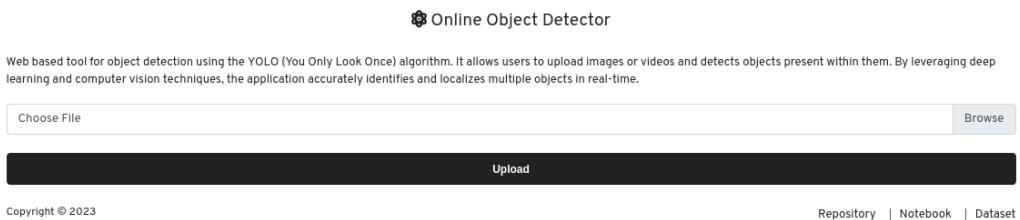
Flask est un framework open source pour le développement de microservices en Java. Il permet de créer rapidement et facilement des applications Java autonomes et productives. Nous avons utilisé le framework Flask pour créer une API permettant d'intégrer notre modèle YOLO de détection d'objets dans une application web conviviale. Flask est un framework léger et flexible en Python, idéal pour le développement d'applications web.



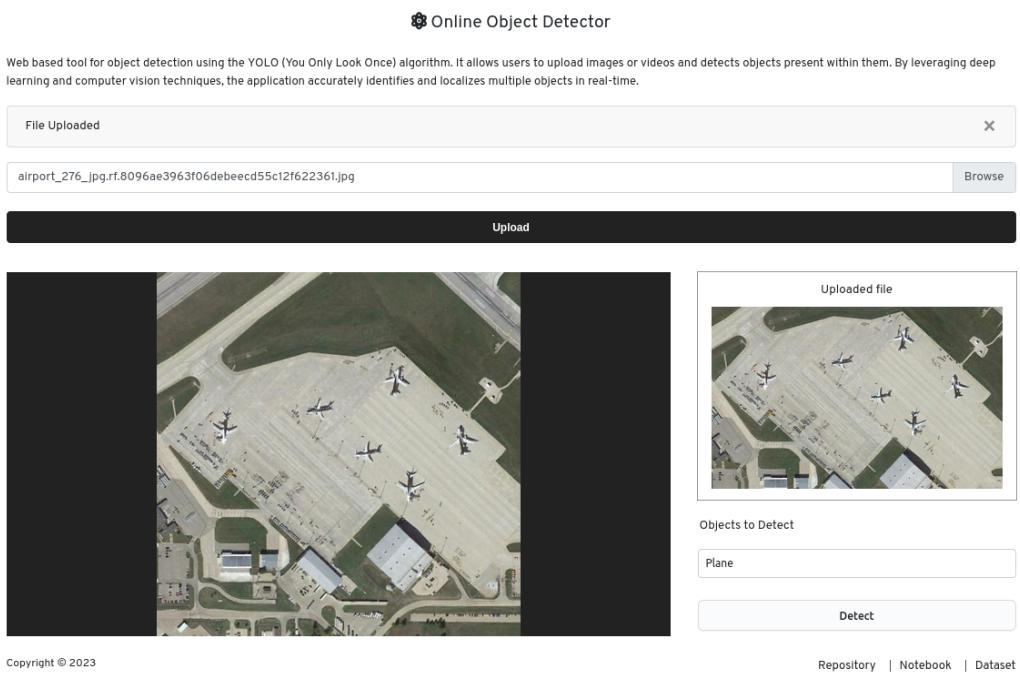
**Figure 4.3 – React JS**

## 4.3 Interfaces

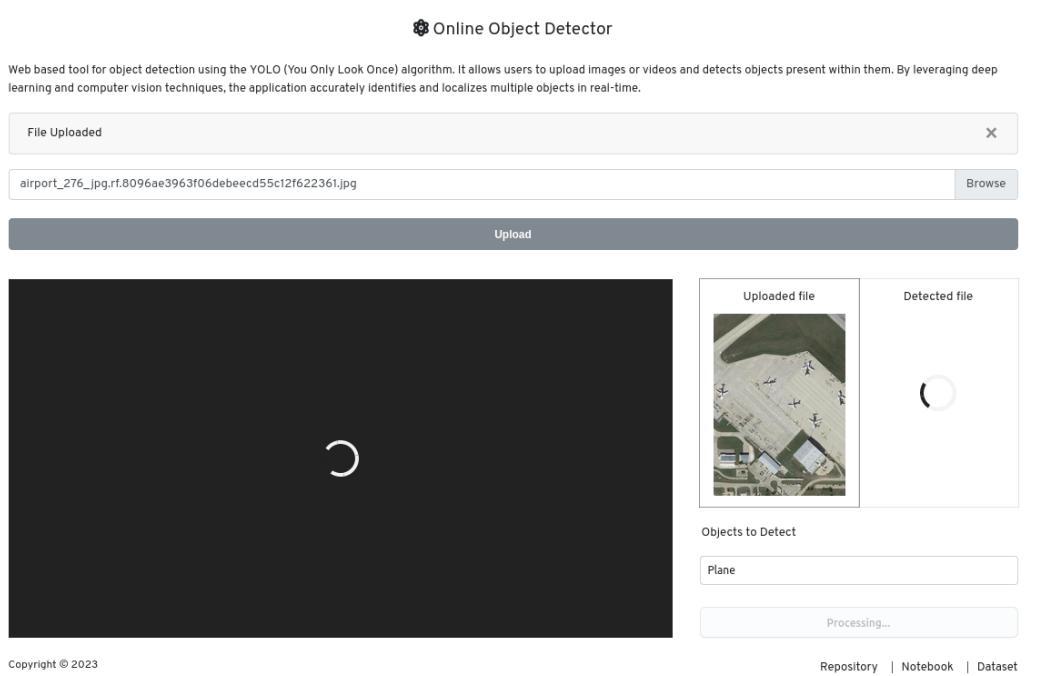
L'interface web offre à l'utilisateur la possibilité d'importer des fichiers d'images ou de vidéos pour la détection des objets. De plus, nous avons inclus une fonctionnalité permettant à l'utilisateur de spécifier les objets qu'il souhaite détecter. Une fois que l'utilisateur a soumis les fichiers et les paramètres de détection, l'interface web traite les données en utilisant le modèle YOLO et génère un nouveau fichier avec les résultats de détection.



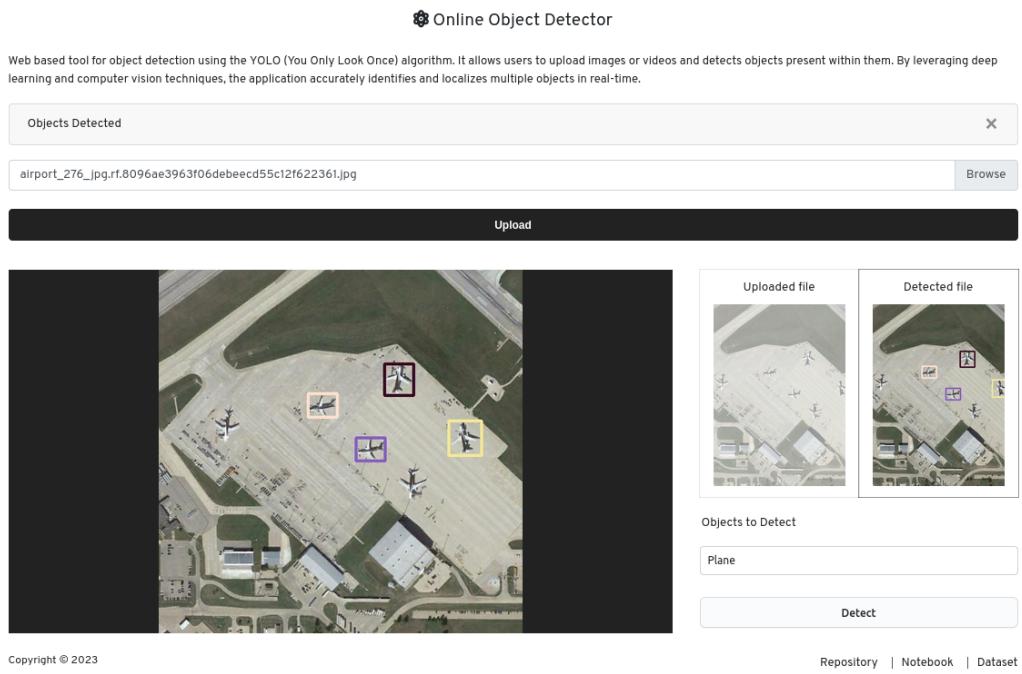
**Figure 4.4 – Importer une image ou une vidéo**



**Figure 4.5 – Sélectionnez les objets à détecter**



**Figure 4.6 – Détection d’objets en cours**



**Figure 4.7 – Détection d’objets réussite**

## **4.4 Conclusion**

Dans ce chapitre, nous avons montré l'architecture et les technologies utilisées pour la réalisation de l'interface web pour notre projet.

## Conclusion

En conclusion, ce projet nous a permis de développer un système de détection et de suivi d'objets en mouvement dans des images et vidéos aériennes en utilisant les algorithmes YOLO et Deep SORT. Nous avons réalisé une interface conviviale et simple à utiliser, permettant aux utilisateurs d'importer des fichiers d'images ou de vidéos et de spécifier les objets à détecter. Le système a démontré son efficacité dans la détection et le suivi des objets, malgré les défis spécifiques liés aux médias aériens tels que la petite taille des objets, les mouvements rapides et les déformations.

Nous avons utilisé des technologies modernes telles que Flask pour le backend de l'API et React pour le frontend de l'interface utilisateur. De plus, nous avons travaillé en équipe, en synchronisant nos commits grâce à l'utilisation de GitHub, ce qui nous a permis de collaborer de manière efficace.

Par ailleurs, nous avons découvert l'utilisation du Deep Learning comme une technologie puissante pour la détection et le suivi des objets. Cela a nécessité une compréhension approfondie des concepts et des techniques associées, ainsi que des ajustements pour adapter les modèles aux spécificités de notre projet.

En résumé, ce projet a été une occasion précieuse pour mettre en pratique nos compétences en génie logiciel et explorer les technologies modernes de détection d'objets. Nous sommes satisfaits des résultats obtenus, mais nous reconnaissions également les perspectives d'amélioration, notamment en termes d'optimisation des performances du système et d'exploration de nouvelles fonctionnalités.

# **Webographie**

REPOSITORY GITHUB

NOTEBOOK COLLAB

DATASET DRIVE