République Algérienne Démocratique et Populaire Ministère de
l'Enseignement Supérieur et de la Recherche Scientifique
Université M'hamed Bougara - Boumerdès



Faculté des Sciences
Département d'Informatique

Domaine : Mathématiques Informatique
Filière : Informatique
Spécialité : systèmes informatiques

Projet de Fin d'Études pour l'Obtention du Diplôme de
Licence Académique

Thème :

Application de détection des places de parking par analyse vidéo

Réalisé par :

- KHEDIR Youcef
- NOURINE Oussama

Encadré par :

- M. REZKI Mohamed Amine

Devant le jury :

M. MEFTAH Mustapha

Remerciements :

Nous adressons nos sincères remerciements à notre encadreur, Mr REZKI Mohamed Amine, pour son accompagnement, ses conseils précieux, sa disponibilité ainsi que son soutien tout au long de la réalisation de ce projet de fin d'études.

Nous remercions également l'ensemble des enseignants du Département d'Informatique pour la qualité de la formation et des connaissances acquises durant notre parcours universitaire.

Nos remerciements s'adressent aussi aux membres du jury pour l'intérêt porté à notre travail et pour le temps consacré à son évaluation.

Enfin, nous exprimons notre profonde gratitude à nos familles, nos amis et toutes les personnes qui nous ont soutenus et encouragés durant l'élaboration de ce projet.

Résumé

Ce projet consiste à développer une application intelligente de détection et de gestion des places de parking en utilisant le modèle de vision artificielle YOLO v8.

Le système permet d'analyser des images ou des flux vidéo afin d'identifier automatiquement les places occupées et disponibles en temps réel.

Une interface web interactive a été réalisée pour faciliter l'utilisation du système et afficher les résultats de manière claire et rapide.

Le modèle a été entraîné sur un dataset annoté au format YOLO afin d'améliorer la précision de détection.

Les performances du système ont été évaluées à l'aide de métriques comme la loss et la précision de détection.

Ce projet contribue à l'automatisation de la gestion des parkings et offre une solution moderne basée sur l'intelligence artificielle et la vision par ordinateur.

Table des matières :

1. Introduction générale : 5
chapitre 1: Généralité et État de l'Art 6
Introduction 6
1.2 Méthodes classiques de détection 6
1.2.1 Méthode de seuillage (Thresholding) 6
1.2.2 Détection par contours 6
1.3 Méthodes de Machine Learning 7
1.3.1 Haar Cascade 7
1.3.2 HOG + SVM 7
1.4.1 R-CNN 9
1.4.2 Fast R-CNN 9
1.4.3 Faster R-CNN 9
1.4.4 YOLO 10
1.5 Tableau comparatif des méthodes 10
1.5.1 Les Réseaux de Neurones Convolutifs (CNN) 11
1.5.2 Principe de fonctionnement des CNN 11
Conclusion 12
Chapitre 2 : Conception du système de détection des places vides dans un parking 13
2.1 Analyse des besoins 13
2.2 Contraintes du système 13
2.3 Architecture générale du système 13
2.4 Description de la Base de données 14
2.3.1 Format des Annotations YOLO 16
2.5 Entraînement du Modèle 17
2.6 Validation du Modèle 19
Chapitre 3 : Implémentation et Résultats 19
Introduction 19
3.1 Métriques d'Évaluation 19
3.1.1 Loss 19
3.1.2 Précision 20
3.1.3 Recall 20
3.1.4 mAP 20
3.2 Résultats d'entraînement du modèle 21
3.3 Test du modèle sur une image de parking: 21
3.4 Limites du système 22
3.5 : L'interface web du système : 22
3.5.1 Technologies utilisées 23

3.5.2 Architecture générale de l'interface 23
3.5.2.1 Partie Backend 23
3.5.2.2 Partie Frontend 23
3.5.3 Fonctionnement du système 24
3.5.4 Gestion des caméras 24
3.5.5 Mise à jour en temps réel 24
3.5.6 Affichage des résultats 25
3.5.6.1. Capture de l'interface globale (les 4 caméras) : 25
3.5.7 Avantages de l'interface développée 28
Conclusion 28
Conclusion générale 29
Références bibliographiques 30

Liste des figures:

Figure 1 : Illustration des caractéristiques de Haar Cascade. 7
Figure 2 : Extraction des caractéristiques HOG. 8
Figure 3 : Représentation schématique d'un réseau neuronal convolutif à deux couches cachées 9
Figure 4 :Système de détection de places de parking – YOLO v8 14
Figure 5 : Exemple d'image annotée au format YOLO. 16
Figure 6 : structure du fichier data.yamal utilisé 17
Figure 7: code d'entraînement 18
Figure 8: Processus d'entraînement du modèle YOLOv8. 19
Figure 9 : Évolution des métriques d'entraînement et de validation du modèle. 21
Figure 10 : Résultats de détection des véhicules sur une image de parking. 22
Figure 11 : Interface globale du système affichant les quatre caméras du parking 25
Figure 12: Affichage de la Zone 1 avec détection des places libres et occupées 26
Figure 13 : Affichage de la Zone 2 avec résultats de détection en temps réel. 26
Figure 14 : Affichage de la Zone 3 du parking intelligent. 27
Figure 15 : Affichage de la Zone 4 avec statistiques des places disponibles. 27

Liste des tableaux:

Tableau 1 : Comparaison des méthodes 11
Tableau 2 : Répartition du dataset 15

1. Introduction générale :

Avec l'augmentation du nombre de véhicules dans les villes, les parkings deviennent de plus en plus difficiles à gérer. Les conducteurs perdent souvent du temps à chercher une place libre, ce qui provoque des embouteillages, une consommation supplémentaire de carburant et une augmentation de la pollution et du CO2. Pour résoudre ce problème, plusieurs solutions intelligentes ont été développées, parmi lesquelles la détection automatique des véhicules dans les parkings par analyse vidéo.

Ce projet consiste à utiliser une caméra et des techniques de traitement d'image afin de détecter automatiquement les places occupées et les places libres dans un parking. Grâce à cette solution, il devient possible de surveiller le parking en temps réel sans intervention humaine.

Aujourd'hui, les systèmes intelligents basés sur la vision par ordinateur et l'intelligence artificielle sont utilisés dans plusieurs domaines comme la sécurité, la circulation routière, les véhicules autonomes et la surveillance des espaces publics. Dans ce contexte, la détection des véhicules dans les parkings représente une application très importante et utile.

Le projet de détection des véhicules dans un parking par analyse vidéo a pour objectif principal de développer un système capable d'analyser les images capturées par une caméra afin d'identifier automatiquement les véhicules présents dans les places de stationnement.

Le système fonctionne généralement selon les étapes suivantes :

Capture vidéo à partir d'une caméra.

- Traitement des images.
- Détection des véhicules.
- Détermination des places libres et occupées.
- Affichage des résultats en temps réel.

Cette solution permet d'automatiser la gestion des parkings et de réduire les erreurs humaines.

Dans les grands parkings publics ou privés, la gestion manuelle des places de stationnement devient difficile. Les utilisateurs rencontrent plusieurs problèmes tel que la difficulté à trouver une place libre , perte de temps ,embouteillages à l'intérieur du parking ,manque d'informations en temps réel ,besoin d'intervention humaine

Les méthodes classiques basées sur des agents humains ou des capteurs physiques peuvent être coûteuses et difficiles à maintenir. La problématique principale de ce projet est donc :

Comment développer un système intelligent capable de détecter automatiquement les véhicules dans un parking à partir d'une vidéo afin d'identifier les places libres et occupées avec une bonne précision et un faible coût ?

Ce projet vise à développer un système intelligent capable de détecter automatiquement les véhicules dans un parking à partir d'une analyse vidéo. L'objectif principal est d'identifier en temps réel les places libres et occupées afin d'améliorer la gestion des parkings. Ce système permet également de réduire le temps de recherche d'une place de stationnement, de diminuer les coûts liés à la surveillance manuelle et de fournir des informations en temps réel aux utilisateurs. Grâce à l'utilisation des techniques de vision par ordinateur et du modèle YOLOv8, le projet offre une solution moderne, rapide et efficace pour la gestion intelligente des parkings.

chapitre 1: Généralité et État de l'Art

Introduction

La gestion des parkings est devenue un défi important en raison de l'augmentation du nombre de véhicules et des difficultés liées au stationnement. Dans ce contexte, la vision par ordinateur offre des solutions permettant d'automatiser la surveillance et la gestion des places de parking à partir d'images ou de vidéos.

Les anciennes méthodes utilisent le traitement d'image classique comme le seuillage ou la détection de contours. Avec l'évolution de l'intelligence artificielle, des méthodes plus avancées basées sur le Machine Learning et le Deep Learning sont apparues, comme Haar Cascade, HOG + SVM, R-CNN et YOLO. Chaque méthode possède ses propres caractéristiques en termes de précision, vitesse et complexité.

1.2 Méthodes classiques de détection

1.2.1 Méthode de seuillage (Thresholding)

Le seuillage est une méthode simple de traitement d'image qui consiste à transformer une image en noir et blanc selon une valeur seuil.

Les pixels supérieurs au seuil deviennent blancs et les autres deviennent noirs. Cette méthode permet de séparer les objets de l'arrière-plan afin de détecter les véhicules.

1.2.2 Détection par contours

Cette méthode consiste à détecter les contours et les formes des objets présents dans une image afin d'identifier les véhicules. Le principe repose sur la recherche des changements

brusques de luminosité entre les pixels, ce qui permet de mettre en évidence les bords des objets. Grâce à cette technique, le système peut reconnaître la forme des véhicules et les distinguer de l'arrière-plan.

1.3 Méthodes de Machine Learning

1.3.1 Haar Cascade

OpenCV propose la méthode Haar Cascade pour la détection d'objets. Cette méthode utilise des caractéristiques appelées « Haar Features » afin de reconnaître les objets dans une image. Le principe de fonctionnement repose sur l'apprentissage des caractéristiques des véhicules à partir d'un ensemble d'images positives contenant des véhicules et d'images négatives ne contenant pas de véhicules. Grâce à cet apprentissage, le système devient capable de détecter automatiquement les véhicules dans les images.

Cette figure montre le principe des masques Haar utilisés pour détecter les différences de contraste dans une image.

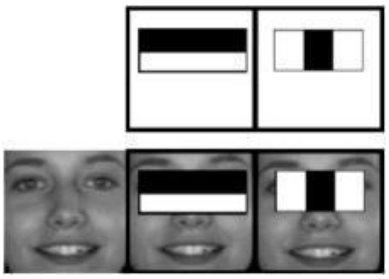


Figure 1 : Illustration des caractéristiques de Haar Cascade.

1.3.2 HOG + SVM

SVM (Support Vector Machine) est un algorithme de classification utilisé pour distinguer les véhicules des autres objets présents dans l'image. Après l'extraction des caractéristiques par la méthode HOG, SVM analyse ces informations afin de déterminer si l'objet détecté correspond à un véhicule ou non. Le principe de SVM repose sur la création d'une frontière de séparation optimale entre différentes classes d'objets. Grâce à cette technique, le système peut effectuer une classification précise et fiable, même lorsque les objets présentent certaines similitudes visuelles. La combinaison de HOG et SVM permet ainsi d'obtenir de bonnes performances dans les systèmes de détection de véhicules.

Cette figure présente l'extraction des contours et des gradients d'une image avec la méthode HOG.

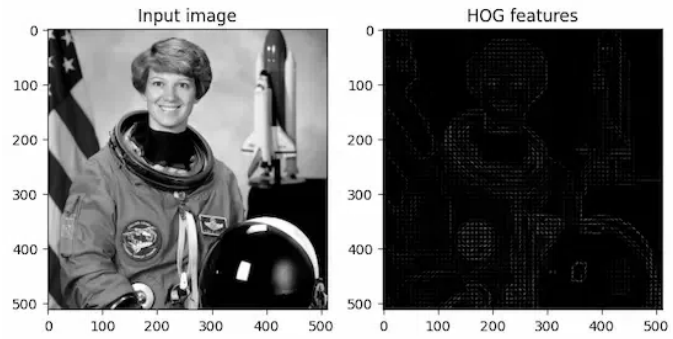


Figure 2 : Extraction des caractéristiques HOG.

1.4 Méthodes basées sur le Deep Learning

Avec le développement rapide de l'intelligence artificielle et l'augmentation de la puissance de calcul des ordinateurs, les méthodes basées sur le Deep Learning sont devenues les plus performantes dans le domaine de la vision par ordinateur et de la détection d'objets. Contrairement aux méthodes classiques de traitement d'image ou de Machine Learning, les techniques de Deep Learning utilisent des réseaux de neurones profonds capables d'apprendre automatiquement les caractéristiques importantes à partir d'un grand nombre de données.

Ces méthodes permettent d'obtenir de meilleures performances en termes de précision, de robustesse, de vitesse de détection, et de capacité de généralisation.

Les modèles Deep Learning sont capables de reconnaître automatiquement des objets complexes dans différentes conditions, variations de lumière, changements d'angles de vue, présence de bruit, objets partiellement cachés, scènes encombrées.

Dans le domaine des parkings intelligents, le Deep Learning permet de détecter automatiquement les véhicules à partir d'images ou de vidéos avec une grande précision, même dans des environnements difficiles.

Le principe général du Deep Learning repose sur l'utilisation de réseaux de neurones artificiels composés de plusieurs couches successives. Chaque couche apprend progressivement des caractéristiques de plus en plus complexes de l'image :

- Premières couches : détectent les contours et les lignes ;
- Couches intermédiaires : reconnaissent les formes ;

- Couches profondes : identifient directement les objets comme les véhicules.

Parmi les architectures les plus utilisées dans la détection d'objets, on trouve les Réseaux de Neurones Convolutifs (CNN), R-CNN, Fast R-CNN, Faster R-CNN, YOLO.

Ces méthodes ont considérablement amélioré les performances des systèmes de détection modernes et sont aujourd'hui largement utilisées dans plusieurs domaines tels que surveillance intelligente; véhicules autonomes; reconnaissance faciale; sécurité; analyse vidéo gestion intelligente des parkings.

1.4.1 R-CNN

R-CNN signifies "Region-Based Convolutional Neural Network".

Cette méthode divise l'image en plusieurs régions puis utilise un réseau de neurones pour détecter les objets.

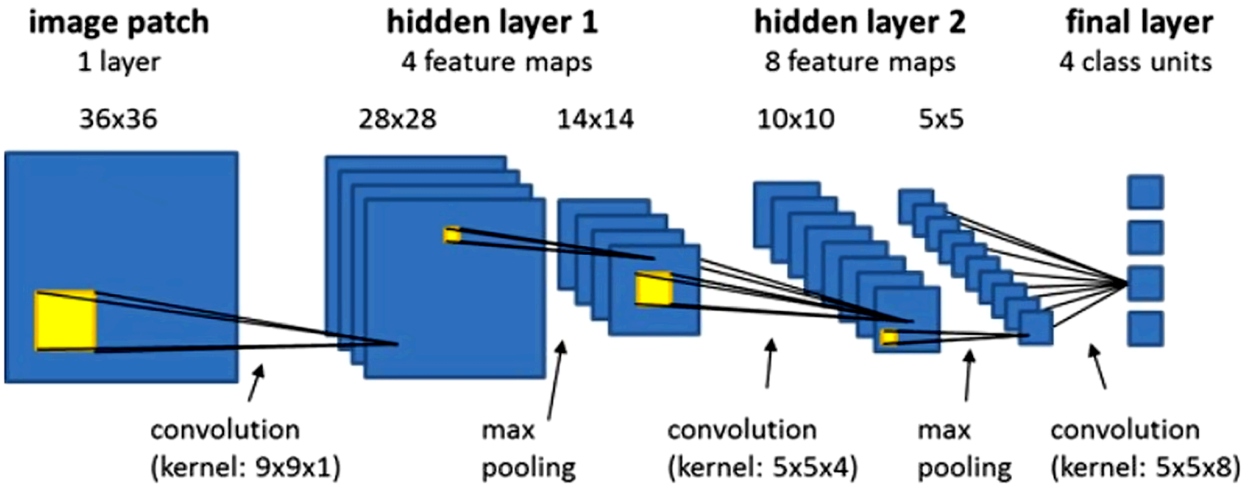


Figure 3 : Représentation schématique d'un réseau neuronal convolutif à deux couches cachées

1.4.2 Fast R-CNN

Fast R-CNN est une amélioration de R-CNN, Au lieu d'analyser chaque région séparément, l'image est traitée une seule fois.

1.4.3 Faster R-CNN

Faster R-CNN améliore encore la vitesse grâce à un système appelé Région Proposal Network (RPN).

1.4.4 YOLO

YOLOv8 est l'une des méthodes les plus populaires de détection d'objets.

YOLO signifie "You Only Look Once", Contrairement aux méthodes R-CNN, YOLO analyse l'image une seule fois pour détecter tous les objets. Le principe est que le modèle divise l'image en plusieurs zones et prédit directement les objets présents.

Dans ce projet, la méthode YOLOv8 a été choisie car elle offre un excellent compromis entre rapidité et précision, ce qui la rend particulièrement adaptée aux applications temps réel basées sur l'analyse vidéo.

1.5 Tableau comparatif des méthodes

Méthode	Type	Avantages	Inconvénients
Seuillage	Traitement d'image	Simple et rapide	Sensible à la lumière
Détection par contours	Traitement d'image	Bonne détection des formes	Sensible au bruit
Haar Cascade	Machine Learning	Rapide	Précision limitée
HOG + SVM	Machine Learning	Bonne précision	Plus lent
R-CNN	Deep Learning	Très précis	Très lent
Fast R-CNN	Deep Learning	Plus rapide que R-CNN	Temps réel difficile

Faster R-CNN	Deep Learning	Haute précision	Complexe
YOLOv8	Deep Learning	Très rapide et précis	Besoin d'entraînement

Tableau 1 : Comparaison des méthodes

1.5.1 Les Réseaux de Neurones Convolutifs (CNN)

Les CNN (Convolutional Neural Networks), appelés en français réseaux de neurones convolutifs, sont des modèles d'intelligence artificielle utilisés principalement dans le domaine de la vision par ordinateur et du traitement d'image. Ils permettent aux ordinateurs de reconnaître automatiquement des objets dans les images et les vidéos, comme les voitures, les personnes ou les panneaux de signalisation.

Dans ce projet, les CNN jouent un rôle important car ils permettent de détecter automatiquement les véhicules dans les images du parking.

1.5.2 Principe de fonctionnement des CNN

Le fonctionnement des CNN est inspiré du cerveau humain, plus précisément du système visuel. Le réseau apprend automatiquement les caractéristiques importantes présentes dans une image afin de reconnaître les objets.

Contrairement aux méthodes classiques de traitement d'image, les CNN n'ont pas besoin de programmer manuellement les caractéristiques des objets. Le modèle apprend seul à partir des données d'entraînement.

Le fonctionnement d'un CNN se déroule en plusieurs étapes importantes.

a. Entrée de l'image (Input Layer)

La première étape consiste à envoyer une image au réseau de neurones. L'image est représentée sous forme de pixels. Chaque pixel contient des informations de couleur et de luminosité. Le CNN reçoit donc une matrice de pixels comme entrée.

b. Couche de convolution (Convolution Layer)

La couche de convolution est l'élément principal du CNN. Dans cette étape, le réseau utilise des petits filtres appelés "kernels" qui parcourent l'image afin d'extraire les caractéristiques importantes. Ces filtres permettent de détecter : les contours ; les lignes ; les formes ; les textures ; les objets. Chaque filtre réalise une opération mathématique appelée convolution.Au début, le réseau détecte des éléments simples comme les bords. Ensuite, dans les couches profondes, il peut reconnaître des formes plus complexes.Cette étape produit une nouvelle image appelée "feature map".

c. Fonction d'activation (Activation Function)

Après la convolution, une fonction d'activation est appliquée afin d'ajouter de la non-linéarité au réseau. La fonction la plus utilisée est ReLU (Rectified Linear Unit), elle transforme les valeurs négatives en zéro et conserve les valeurs positives.

d. Couche de pooling (Pooling Layer)

La couche de pooling sert à réduire la taille des données tout en conservant les informations importantes. La méthode la plus utilisée est le Max Pooling, dans laquelle le système sélectionne la valeur maximale à l'intérieur d'une petite zone de l'image.

e. Couche entièrement connectée (Fully Connected Layer)

Après plusieurs couches de convolution et de pooling, les informations extraites sont envoyées vers une couche entièrement connectée. Cette couche fonctionne comme un réseau de neurones classique et a pour rôle d'analyser les caractéristiques extraites afin de prendre la décision finale et de classifier l'objet détecté

f. Phase d'apprentissage (Training)

Pour apprendre correctement, le CNN a besoin d'un grand nombre d'images appelé dataset. Chaque image du dataset contient des annotations indiquant l'objet présent. Pendant la phase d'apprentissage, le réseau analyse les images, compare les résultats obtenus avec les réponses correctes, calcule les erreurs puis ajuste automatiquement ses paramètres afin d'améliorer ses performances. Ce processus est répété plusieurs fois jusqu'à obtenir une bonne précision de détection.

Conclusion

Dans ce chapitre, nous avons présenté les différentes méthodes utilisées pour la détection des véhicules dans les parkings intelligents. Nous avons étudié les approches classiques de

traitement d'image, les méthodes basées sur le Machine Learning ainsi que les techniques modernes de Deep Learning.

L'étude comparative réalisée montre que les méthodes Deep Learning, notamment YOLOv8, offrent les meilleures performances en termes de rapidité et de précision pour les applications de détection en temps réel.

Ces travaux constituent une base importante pour la conception et le développement du système intelligent de gestion de parking présenté dans les chapitres suivants.

Chapitre 2 : Conception du système de détection des places vides dans un parking

2.1 Analyse des besoins

Le système de détection de places de parking a été conçu afin de répondre à un besoin d'automatisation de la gestion des parkings à l'aide de l'intelligence artificielle. L'objectif principal est de détecter en temps réel les véhicules présents dans un parking et de déterminer l'état de chaque place, qu'elle soit libre ou occupée. Pour assurer le bon fonctionnement du système, plusieurs besoins fonctionnels ont été définis, notamment la capture d'un flux vidéo provenant d'une caméra, la détection automatique des véhicules, l'identification des places libres et occupées, l'affichage des résultats en temps réel sur une interface web ainsi que la mise à jour dynamique de l'état des places. En plus des besoins fonctionnels, le système doit également répondre à certains besoins non fonctionnels tels qu'une bonne performance avec un traitement en temps réel et une faible latence, une précision élevée grâce au modèle YOLOv8, une fiabilité assurant un fonctionnement continu, une accessibilité via un navigateur web ainsi qu'une architecture modulaire facilitant la maintenance et l'évolution du système.

2.2 Contraintes du système

Le système dépend fortement du modèle YOLOv8 entraîné. La qualité de la caméra (résolution et angle de vue) influence directement la précision des résultats. De plus, une puissance de calcul suffisante, idéalement avec GPU, est nécessaire pour assurer le traitement en temps réel. Enfin, une connexion réseau stable est requise pour l'accès à l'interface web et la transmission du flux vidéo.

2.3 Architecture générale du système

Le système repose sur une architecture client–serveur basée sur un pipeline de traitement d'images en temps réel.

Le flux de fonctionnement est le suivant :

La caméra capture la vidéo du parking, puis OpenCV assure l'acquisition des images. Ces images sont ensuite envoyées au backend développé en Python, où le modèle YOLOv8 effectue la détection des véhicules. Une logique de traitement permet de déterminer si chaque place est libre ou occupée. Les résultats sont ensuite transmis à l'interface web via un serveur Flask, afin d'être affichés en temps réel à l'utilisateur. Cette architecture permet de séparer clairement les différents modules du système : capture vidéo, traitement par intelligence artificielle et affichage des résultats, ce qui facilite la maintenance et l'évolution du système.

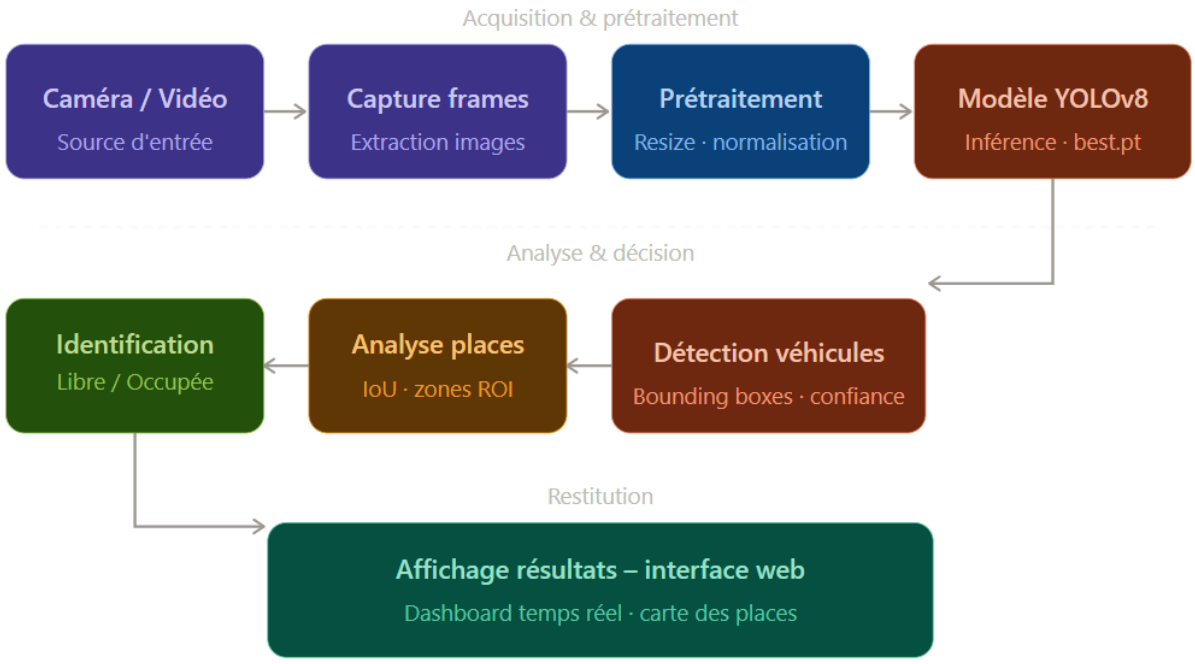


Figure 4 :Système de détection de places de parking – YOLOv8

Cette figure (Figure 4) représente le pipeline de traitement du système, depuis l'acquisition du flux vidéo jusqu'à la détection des objets et la visualisation des résultats sur l'interface web.

2.4 Description de la Base de données

La Base de données utilisée dans ce projet contient des images de parking annotées manuellement afin d'entraîner le modèle à détecter les véhicules ainsi que les places libres.Elle est composée de trois parties principales :

Dossier	Nombre d'images	Rôle
Train	1339 images	Utilisé pour l'apprentissage du modèle
Valid	391 images	Utilisé pour évaluer le modèle pendant l'entraînement
Test	191 images	Utilisé pour tester les performances finales

Tableau 2 : Répartition du dataset

Dataset/

├── train/ → images + labels pour l'entraînement du modèle
├── valid/ → images + labels pour la validation des performances
└── test/ → images + labels pour les tests finaux

Le dataset est organisé en trois parties principales : train, valid et test. Le dossier train contient les images utilisées pour entraîner le modèle YOLOv8 ainsi que leurs annotations stockées dans le dossier labels. Le dossier valid est utilisé pendant l'entraînement afin d'évaluer les performances du modèle et vérifier sa capacité de généralisation. Enfin, le dossier test sert à tester le modèle final sur des données jamais vues auparavant. Chaque sous-dossier images contient les images du parking, tandis que les sous-dossiers labels contiennent les annotations correspondantes au format YOLO. Example: (Figure 5)

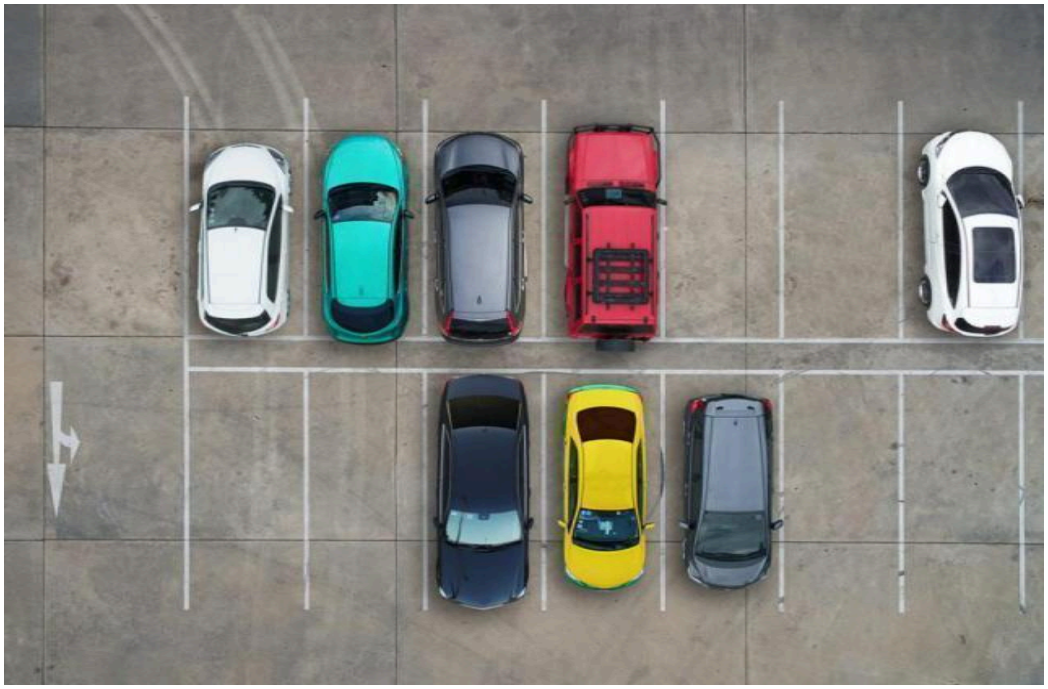


Figure 5 : Exemple d'image annotée au format YOLO.

l'annotation de Figure 5 :

0 0.228125 0.30859375 0.121875 0.3390625
1 0.234375 0.71875 0.1125 0.334375
0 0.35078125 0.30859375 0.1203125 0.3296875
0 0.4703125 0.3109375 0.10625 0.34375
0 0.57578125 0.31953125 0.1140625 0.3296875
1 0.6921875 0.32265625 0.121875 0.3390625
1 0.81171875 0.3171875 0.1140625 0.340625
0 0.92265625 0.3125 0.1109375 0.3375
1 0.34921875 0.71796875 0.1109375 0.3515625
0 0.4625 0.70625 0.125 0.334375
0 0.58203125 0.7109375 0.1109375 0.334375
0 0.69453125 0.71796875 0.1140625 0.3515625
1 0.80625 0.715625 0.11875 0.340625
1 0.92109375 0.71640625 0.1171875 0.3390625

2.3.1 Format des Annotations YOLO

Les annotations sont stockées dans des fichiers texte (.txt) suivant le format standard YOLO. Chaque ligne du fichier représente un objet détecté dans l'image.

Exemple :

1 0.234375 0.71875 0.1125 0.334375

Le format général est le suivant : class x center y_center width height

Les différentes valeurs représentent les informations utilisées par le modèle YOLO pour localiser les objets dans l'image. La valeur class indique la classe de l'objet détecté, x_center représente la coordonnée X du centre du rectangle de détection, tandis que y_center correspond à la coordonnée Y du centre. Les valeurs width et height représentent respectivement la largeur et la hauteur du rectangle englobant l'objet détecté. Toutes ces coordonnées sont normalisées entre 0 et 1 afin d'être adaptées aux dimensions de l'image.

Dans ce projet, deux classes ont été utilisées : car: signifie véhicule détecté

free: signifie place libre

Le fichier data.yaml utilisé pendant l'entraînement est le suivant :

```yaml
path: dataset

train: train/images
val: valid/images
test: test/images

nc: 2
names: ['car', 'free']
```

Figure 6 : structure du fichier data.yamal utilisé

Les paramètres du fichier data.yaml permettent à YOLO de localiser les données du dataset ainsi que les différentes classes utilisées.

2.5 Entraînement du Modèle

Le modèle YOLO a été entraîné sur la base de données contenant les images annotées suivant le processus présenté dans la figure 8.

Pendant l'entraînement, YOLO effectue les étapes suivantes :

- Lecture des images du dossier train.
- Lecture des labels correspondants.
- Extraction des caractéristiques à l'aide du réseau CNN.
- Prédiction des bounding boxes.
- Comparaison avec les annotations réelles.
- Calcul des erreurs (loss).
- Mise à jour des poids du réseau.

Une epoch représente un passage complet sur toutes les images du dataset train.

En utilisant ce code:

```python
from ultralytics import YOLO

# Charger le modèle YOLOv8
model = YOLO("yolov8n.pt")

# Entraînement du modèle
model.train(
    data="data.yaml",
    epochs=50,
    imgsz=640,
    batch=16,
    name="train-3"
)
```

Figure 7: code d'entraînement

Ou la commande suivante:

```
yolo task=detect mode=train model=yolov8n.pt data=data.yaml epochs=50 imgsz=640 batch=16
```

Le modèle a été entraîné sur 50 epochs et est sauvegardé dans le fichier suivant :

runs/detect/train-3/weights/best.pt

Le fichier best.pt représente le meilleur modèle obtenu pendant l'entraînement.
voici la figure qui présente le workflow d'entraînement du modèle, depuis la préparation des données .

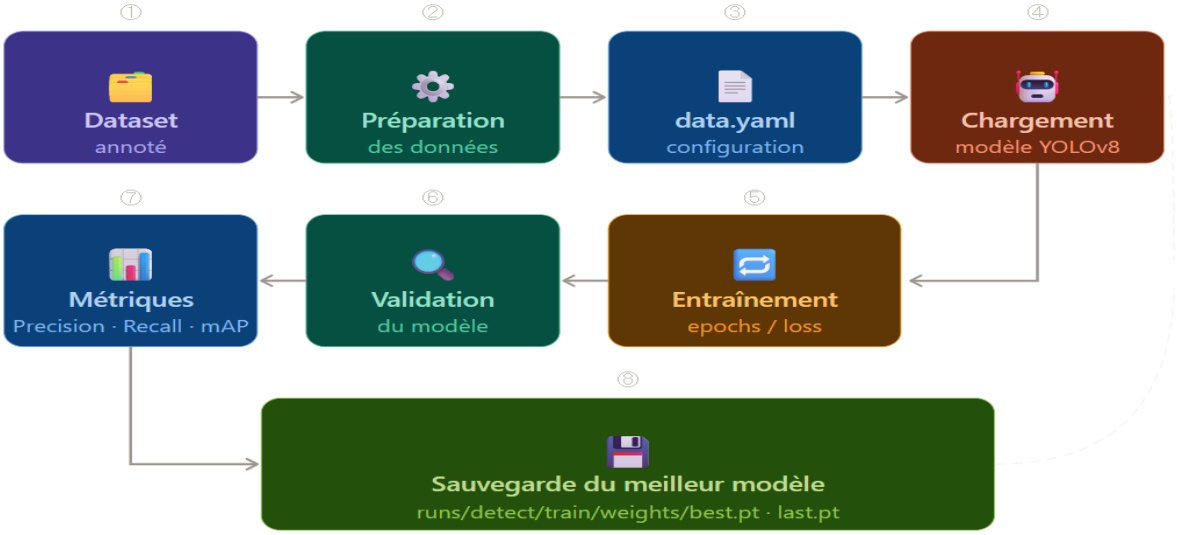


Figure 8: Processus d'entraînement du modèle YOLOv8.

2.6 Validation du Modèle

À la fin de chaque epoch, YOLO utilise le dossier de validation afin d'évaluer les performances du modèle. Pendant cette étape, le modèle ne s'entraîne pas et les poids du réseau ne sont pas modifiés ; seules les performances du modèle sont calculées. Les principales métriques utilisées durant cette phase sont la loss, la précision (precision), le rappel (recall) et le mAP. Cette étape permet de vérifier si le modèle généralise correctement sur des images qui n'ont pas été utilisées pendant la phase d'apprentissage.

Chapitre 3 : Implémentation et Résultats

Introduction

Ce chapitre présente la phase d'implémentation du système de détection des places de parking ainsi que l'analyse des résultats obtenus. Il décrit brièvement les étapes de réalisation du modèle YOLOv8 et son intégration dans l'application. Ensuite, il introduit les principales métriques d'évaluation utilisées pour mesurer les performances du modèle, telles que la loss, la précision, le rappel et le mAP. Enfin, il présente et commente les résultats d'entraînement ainsi que les tests réalisés sur des images réelles de parking afin de vérifier l'efficacité du système.

3.1 Métriques d'Évaluation

3.1.1 Loss

La loss représente l'erreur globale du modèle lors de l'apprentissage. Plus la loss diminue, plus les performances du modèle s'améliorent, ce qui indique qu'il apprend correctement à partir des données. Dans ce projet, plusieurs types de losses sont utilisés. La box_loss correspond à l'erreur liée à la prédiction des bounding boxes, la cls_loss mesure l'erreur de classification des objets détectés, tandis que la dfl_loss représente une erreur de localisation plus avancée permettant d'affiner la précision des coordonnées. Une diminution progressive de ces différentes losses au cours des epochs indique un bon apprentissage et une convergence efficace du modèle.

3.1.2 Précision

La précision représente le pourcentage des objets détectés correctement.

Elle est calculée à partir de la formule suivante :

Précision = TP / (TP + FP)

TP signifie True Positive et FP signifie False Positive

Une précision élevée signifie que le modèle effectue peu de fausses détections.

3.1.3 Recall

Le recall représente le pourcentage des objets réels détectés par le modèle. La formule utilisée est Recall = TP / (TP + FN) avec FN =False Negative

Un recall élevé signifie que le modèle détecte correctement la majorité des véhicules présents dans l'image.

3.1.4 mAP

Le mAP (Mean Average Precision) représente la performance globale du modèle.

Le mAP combine : la précision, le recall et la qualité des bounding boxes.

Le mAP50 utilise un seuil IoU de 50%. Plus la valeur du mAP est élevée, meilleures sont les performances du modèle.

3.2 Résultats d'entraînement du modèle

La figure suivante présente l'évolution des performances du modèle durant les phases d'entraînement et de validation.

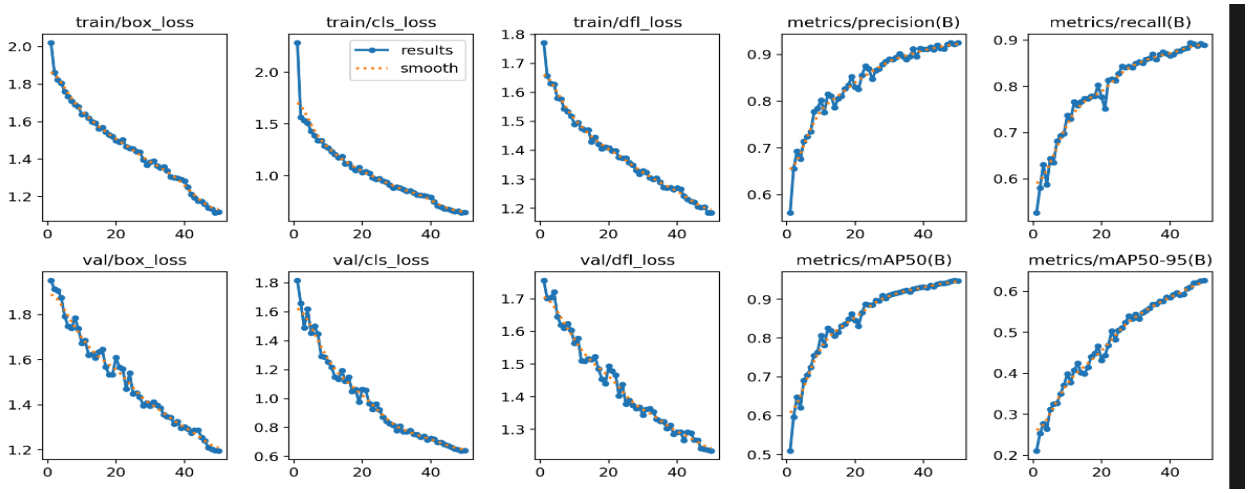


Figure 9 : Évolution des métriques d'entraînement et de validation du modèle.

Les courbes montrent une diminution progressive des pertes d'entraînement et de validation (box_loss, cls_loss et dfl_loss), ce qui indique que le modèle apprend correctement et converge de manière stable au fil des epochs.

Par ailleurs, les métriques de performance augmentent progressivement :

- la précision (Precision) atteint environ 92 % .
- le rappel (Recall) atteint environ 89 % .
- le score mAP50 atteint environ 95 % .
- le score mAP50-95 atteint environ 63 %.

Ces résultats montrent que le modèle obtient de bonnes performances de détection sur les données de validation avec une bonne capacité de généralisation et sans signe important de sur-apprentissage.

3.3 Test du modèle sur une image de parking:

Afin de valider les performances du modèle entraîné, un test a été réalisé sur une image réelle d'un parking. Le système détecte automatiquement les places occupées et les places libres à l'aide de boîtes de détection colorées. Exemple(figure 10):

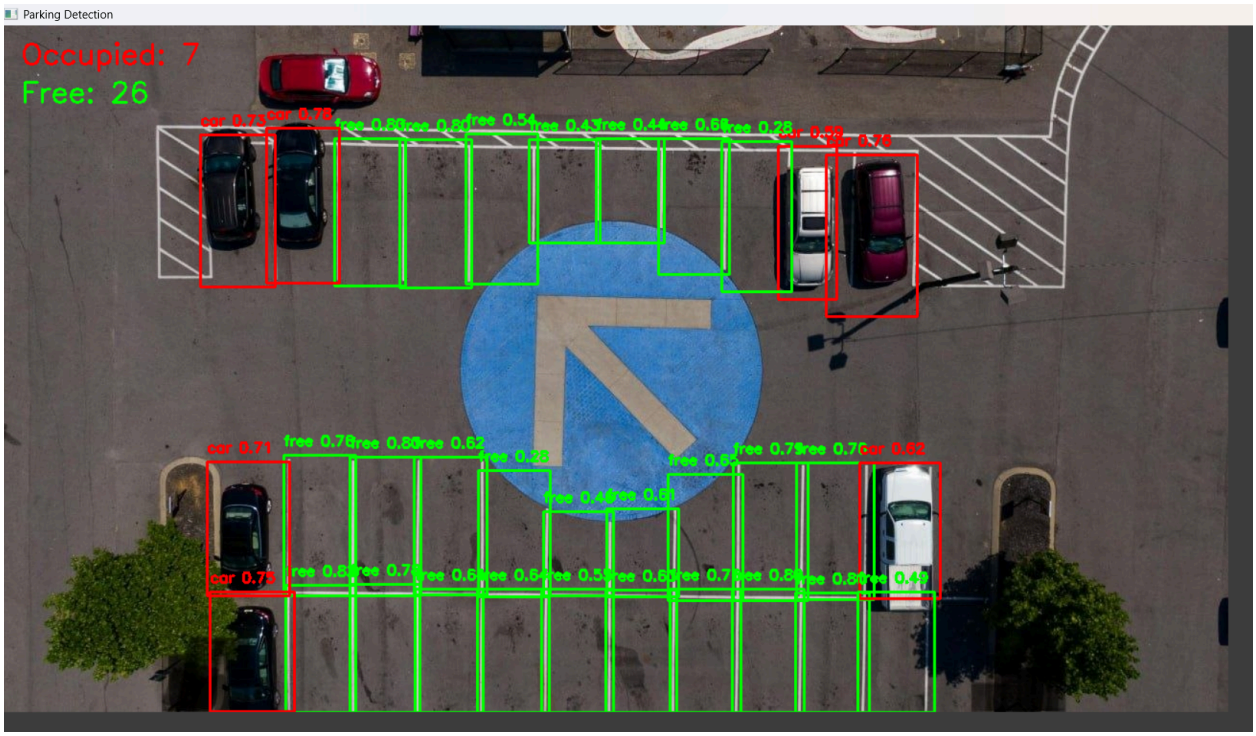


Figure 10 : Résultats de détection des véhicules sur une image de parking.

3.4 Limites du système

Malgré les résultats satisfaisants obtenus, le système présente certaines limites. En effet, une diminution de la précision peut être observée dans certains angles de vue, ce qui affecte la fiabilité de la détection. Le modèle montre également une sensibilité aux variations d'éclairage, notamment dans des conditions de faible luminosité ou de forte exposition. De plus, des erreurs peuvent apparaître dans le cas de parkings très encombrés, où les véhicules se chevauchent ou sont partiellement visibles. Enfin, les performances du système dépendent fortement de la qualité et de la diversité du dataset utilisé pour l'entraînement.

3.5 : L'interface web du système :

Après l'entraînement du modèle de détection basé sur YOLO, une interface web a été développée afin de rendre le système plus interactif et plus facile à utiliser. Cette interface permet de visualiser les vidéos des caméras du parking en temps réel ainsi que l'état des places de stationnement détectées par le modèle d'intelligence artificielle.

L'objectif principal de cette interface est de fournir une surveillance en temps réel des places libres et occupées dans le parking à travers une plateforme simple et moderne.

3.5.1 Technologies utilisées

Plusieurs technologies ont été utilisées pour développer cette interface web :

Python : utilisé pour développer la logique principale du système.
Flask : framework web léger utilisé pour créer le serveur web et gérer les routes de l'application.
OpenCV : utilisé pour lire les vidéos et afficher les résultats de détection.
YOLO : modèle d'intelligence artificielle utilisé pour détecter les véhicules et les places libres.
HTML / CSS / JavaScript : utilisés pour construire l'interface utilisateur.
Bootstrap : utilisé pour améliorer le design et rendre l'interface responsive.
Font Awesome : utilisé pour ajouter des icônes dans l'interface.

3.5.2 Architecture générale de l'interface

L'interface web est composée de deux parties principales :

3.5.2.1 Partie Backend

La partie Backend a été développée avec Flask. Elle assure plusieurs fonctions importantes :

-chargement du modèle YOLO entraîné,
-lecture des vidéos des caméras,
-traitement des images,
-détection des véhicules,
-calcul du nombre de places libres et occupées,
-envoi des résultats à l'interface web.

Le modèle YOLO est chargé avec le fichier :

```
model = YOLO("runs/detect/train-3/weights/best.pt")
```

Cette étape permet au système d'utiliser le modèle entraîné pour effectuer les détections sur les vidéos du parking.

3.5.2.2 Partie Frontend

La partie Frontend représente l'interface visible par l'utilisateur. Elle a été développée avec HTML, CSS et JavaScript.

Cette interface affiche :

- les flux vidéo des caméras,

- les statistiques des places libres et occupées,
- les boutons de sélection des différentes zones du parking.

Le design de l'interface utilise Bootstrap afin d'obtenir une présentation moderne et responsive.

3.5.3 Fonctionnement du système

Le système fonctionne selon les étapes suivantes :

1. Les vidéos des différentes caméras sont chargées dans l'application.
2. Chaque vidéo est analysée par le modèle YOLO.
3. Le modèle détecte les véhicules et les places libres.
4. Les résultats sont affichés directement sur les vidéos sous forme de rectangles et d'étiquettes.
5. Le nombre de places libres et occupées est calculé automatiquement.
6. Les statistiques sont mises à jour en temps réel dans l'interface web.

Le système effectue une détection toutes les deux secondes afin d'améliorer les performances et réduire la charge de calcul.

3.5.4 Gestion des caméras

Dans cette application, quatre vidéos ont été utilisées afin de simuler quatre caméras de surveillance dans un parking réel.

video_sources = ["parking1.mp4", "parking2.mp4", "parking3.mp4", "parking4.mp4"]

Chaque vidéo représente une zone différente du parking : Zone 1, Zone 2, Zone 3, Zone 4.

L'utilisateur peut afficher toutes les caméras simultanément ou sélectionner une seule caméra pour avoir une vue détaillée.

3.5.5 Mise à jour en temps réel

L'interface met automatiquement à jour les statistiques des places disponibles grâce à JavaScript et aux requêtes API.

Le serveur Flask fournit les données sous format JSON grâce à la route :

@app.route('/api/status')

JavaScript récupère ensuite les nouvelles valeurs toutes les 1.5 secondes grâce à la fonction :

setInterval(updateCountersRealTime, 1500);

Cette méthode permet d'obtenir une mise à jour automatique sans recharger la page web.

3.5.6 Affichage des résultats

Le système affiche les résultats directement sur les vidéos, les véhicules détectés sont affichés avec des rectangles rouges et les places libres sont affichées avec des rectangles verts.

Chaque détection contient également le nom de la classe détectée et le niveau de confiance du modèle. Cela permet à l'utilisateur de visualiser clairement les résultats de l'intelligence artificielle.

3.5.6.1. Capture de l'interface globale (les 4 caméras) :

La figure suivante représente l'interface principale de l'application web. Cette interface affiche les quatre zones du parking surveillées par les caméras ainsi que les statistiques des places libres et occupées en temps réel.

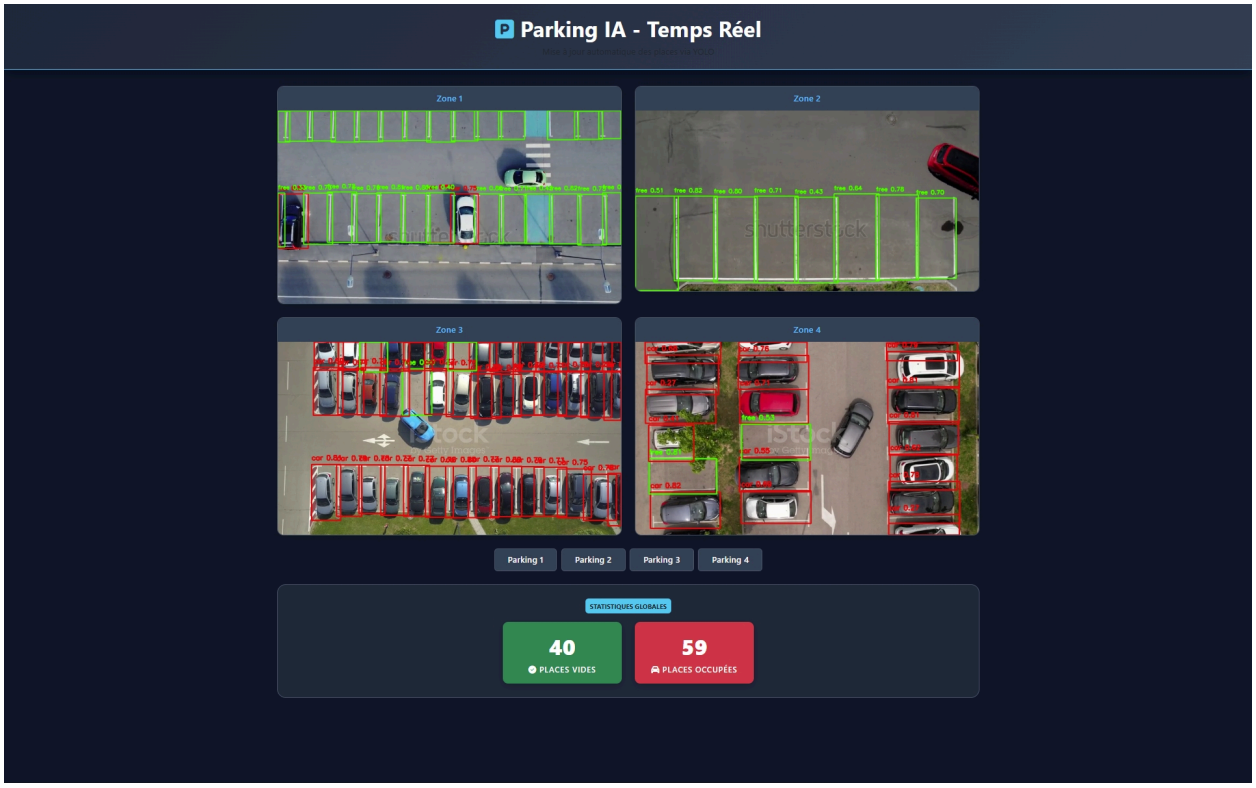


Figure 11 : Interface globale du système affichant les quatre caméras du parking

La figure 12 montre la détection des places libres et occupées dans la Zone 1:

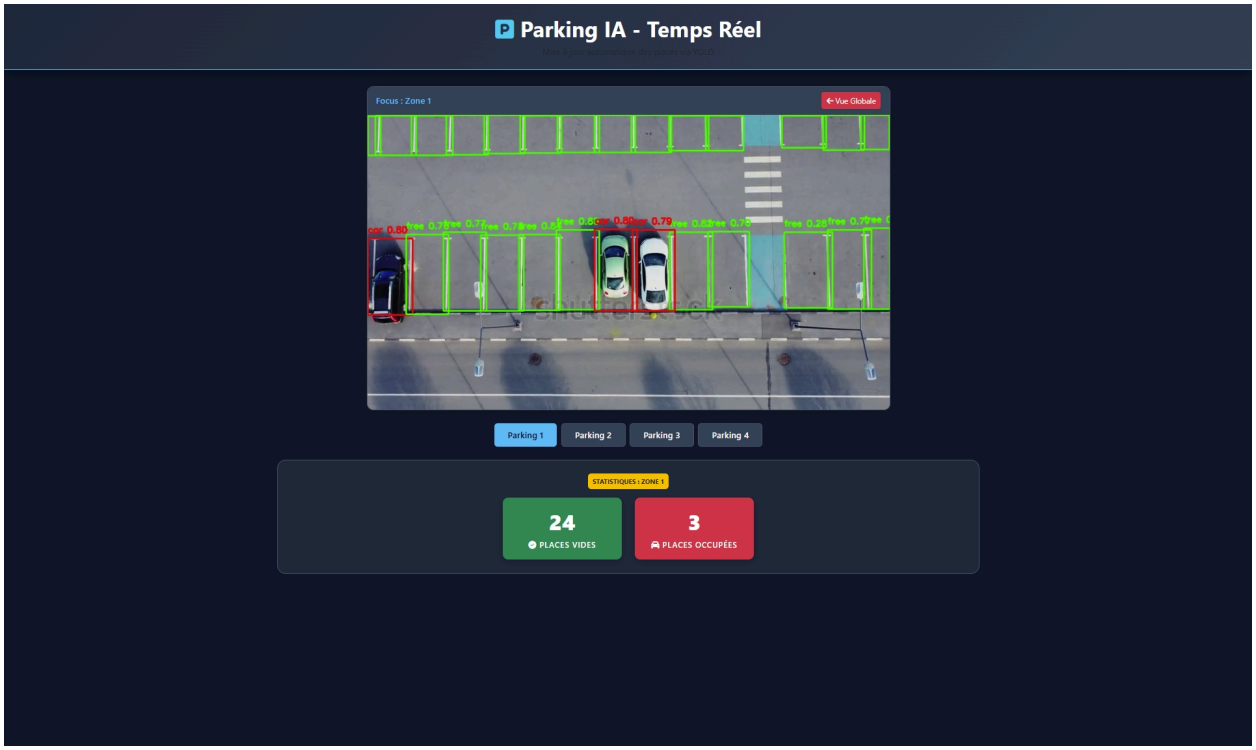


Figure 12: Affichage de la Zone 1 avec détection des places libres et occupées

La figure 13 présente les résultats de détection dans la Zone 2 du parking:

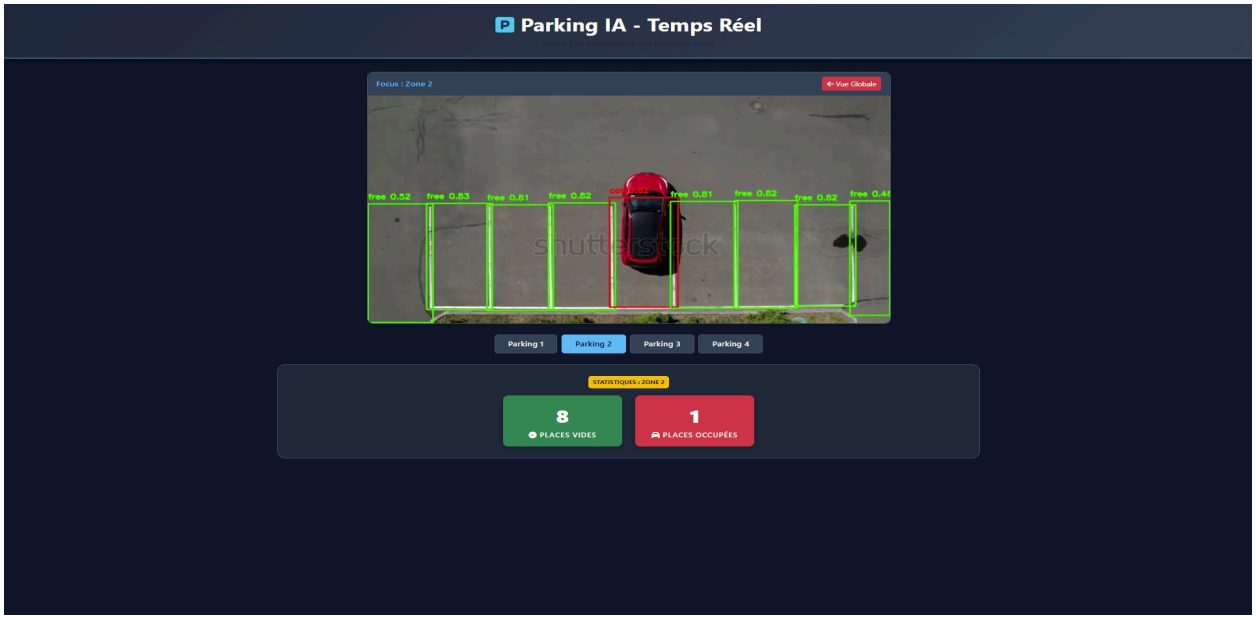


Figure 13 : Affichage de la Zone 2 avec résultats de détection en temps réel.

La figure 14 illustre l'affichage de la Zone 3 surveillée par le système:

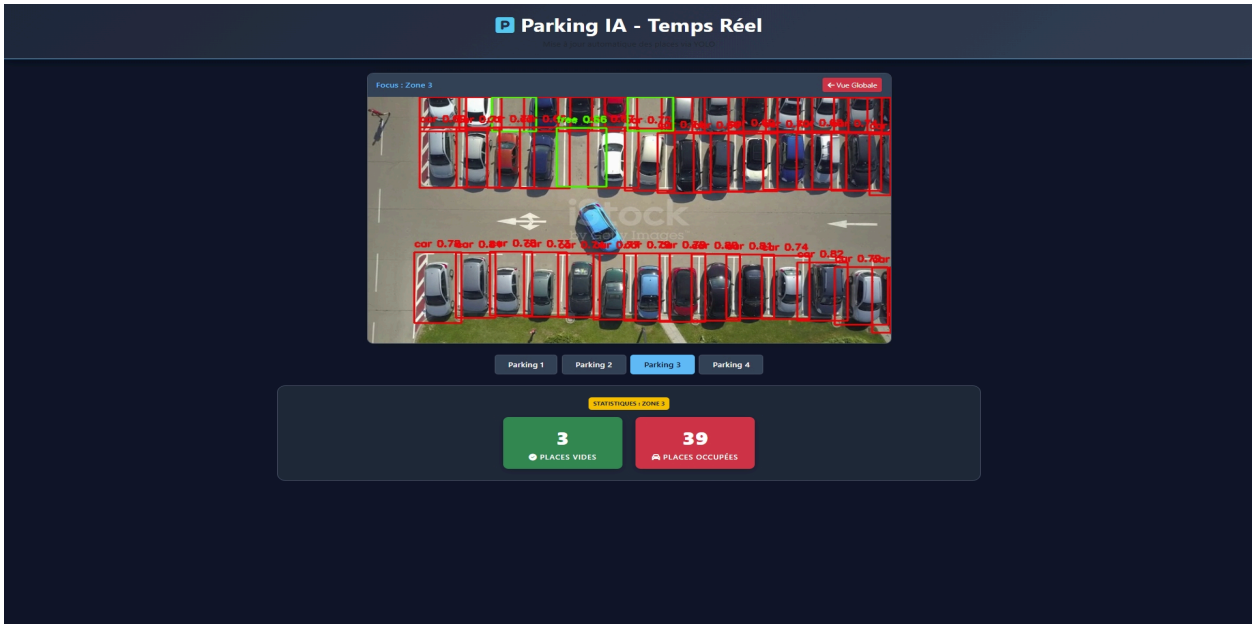


Figure 14 : Affichage de la Zone 3 du parking intelligent.

La figure 15 montre la Zone 4 avec les statistiques des places disponibles:

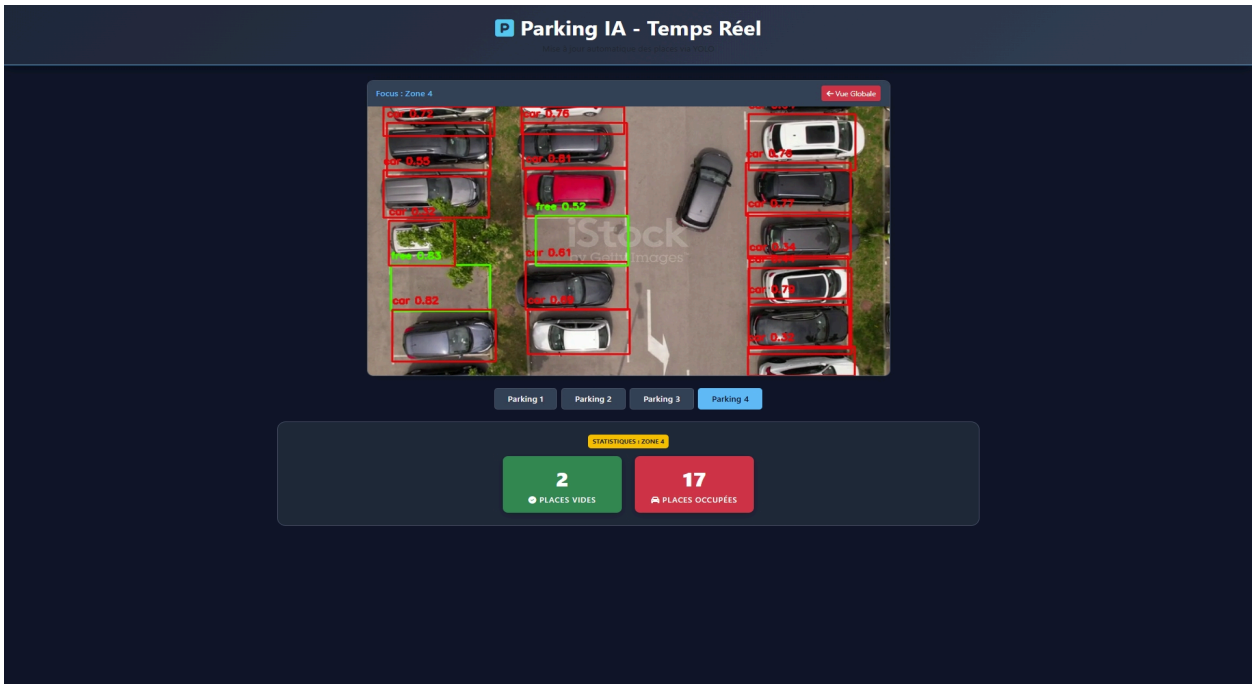


Figure 15 : Affichage de la Zone 4 avec statistiques des places disponibles.

3.5.7 Avantages de l'interface développée

L'interface développée présente plusieurs avantages importants qui améliorent considérablement l'efficacité du système. Elle permet une surveillance du parking en temps réel, offrant ainsi une vision continue de l'état des places disponibles et occupées. Son affichage clair et moderne facilite la compréhension des informations par l'utilisateur. De plus, elle est facile à utiliser, même pour des personnes non techniques, et donne un accès rapide aux statistiques essentielles du parking. L'interface permet également de surveiller plusieurs zones simultanément, ce qui renforce sa capacité de gestion à grande échelle. Enfin, elle contribue à une meilleure organisation et à une gestion plus efficace des places de stationnement, rendant ainsi le projet plus complet et plus proche d'un système réel déployé dans des environnements professionnels.

Conclusion

Ce chapitre a présenté l'implémentation du système, l'évaluation du modèle YOLOv8 ainsi que l'interface web développée. Les résultats obtenus montrent une bonne performance du modèle en termes de précision, de rappel et de mAP, confirmant son efficacité dans la détection des véhicules et la gestion des places de parking.

L'interface web permet également de visualiser en temps réel les flux vidéo et les statistiques des places disponibles et occupées. Grâce à l'utilisation de Flask, YOLO et OpenCV, le système propose une solution intelligente et efficace pour la gestion des parkings.

Conclusion générale

En conclusion, ce projet a permis de concevoir et de développer un système intelligent de détection des places de stationnement basé sur l'analyse vidéo et l'utilisation du modèle YOLOv8. L'objectif principal était d'automatiser la surveillance des places de parking afin d'identifier en temps réel les emplacements libres et occupés.

Les résultats obtenus montrent que l'approche adoptée permet une détection efficace des véhicules et une gestion automatisée de l'occupation des places de stationnement. L'intégration d'une interface web interactive a également contribué à faciliter la visualisation des informations et le suivi de l'état du parking de manière simple et intuitive.

Ce travail met en évidence le potentiel des techniques de vision par ordinateur pour améliorer la gestion des espaces de stationnement et optimiser l'expérience des utilisateurs. Toutefois, certaines contraintes, notamment liées aux conditions d'éclairage, aux variations de l'environnement et aux angles de prise de vue, peuvent influencer les performances du système. Ces limites ouvrent la voie à plusieurs pistes d'amélioration visant à renforcer sa robustesse et sa fiabilité dans des situations réelles.

Dans une perspective future, il serait pertinent d'enrichir la base de données d'entraînement avec des scénarios plus diversifiés, d'expérimenter le système sur des caméras installées dans des environnements réels et d'intégrer de nouvelles fonctionnalités telles qu'une application mobile, un système de réservation des places ou encore des outils d'analyse statistique. Ces évolutions permettraient d'améliorer davantage les performances du système et de contribuer au développement de solutions de parking intelligent plus complètes et plus efficaces.

Références bibliographiques

[1] Redmon, J., Divvala, S., Girshick, R. et Farhadi, A., « You Only Look Once: Unified, Real-Time Object Detection », Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, États-Unis, 2016, pp. 779–788.

[2] Jocher, G., Chaurasia, A. et Qiu, J., Ultralytics YOLOv8 Documentation, Ultralytics, 2023. Disponible : https://github.com/ultralytics/ultralytics

[3] Girshick, R., Donahue, J., Darrell, T. et Malik, J., « Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation », Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, États-Unis, 2014, pp. 580–587.

[4] Dalal, N. et Triggs, B., « Histograms of Oriented Gradients for Human Detection », Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), San Diego, États-Unis, 2005, pp. 886–893.

[5] Viola, P. et Jones, M., « Rapid Object Detection using a Boosted Cascade of Simple Features », Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Kauai, États-Unis, 2001, pp. 511–518.

[6] LeCun, Y., Bengio, Y. et Hinton, G., « Deep Learning », Nature, vol. 521, n° 7553, 2015, pp. 436–444.

[7] Ren, S., He, K., Girshick, R. et Sun, J., « Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks », IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, n° 6, 2017, pp. 1137–1149.

[8] Abadi, M. et al., « TensorFlow: A System for Large-Scale Machine Learning », Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Savannah, États-Unis, 2016, pp. 265–283.

[9] Bradski, G., « The OpenCV Library », Dr. Dobb's Journal of Software Tools, 2000. Disponible : https://opencv.org

[10] Géron, A., Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow, 2e édition, O'Reilly Media, 2019.

[11] PKLot Dataset, Base de données de parkings pour la vision par ordinateur. Disponible : https://web.inf.ufpr.br/vri/databases/parking-lot-databas/

[12] Palazzi, A., Solera, F., Calderara, S., Alletto, S. et Cucchiara, R., « Learning to Map Vehicles into Bird's Eye View », International Conference on Image Analysis and Processing (ICIAP), 2017, pp. 233–243.