République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université A. MIRA-BEJAIA

جامعة بجاية

Tasdawit n Bgayet

Université de Béjaïa

Faculté des Sciences Exactes

Département d'Informatique


# MÈMOIRE

**En vue de l'obtention du Diplôme de**

**Master en Informatique**

**Domaine :** Mathématiques et Informatique **Filière :** Informatique

**Spécialité :** Intelligence artificielle

**Présenté par**


**Mme.HAMENI Amina**

**M. AGOUD Ayoub**


***Thème***

## Réalisation d'une application basée sur l'intelligence artificielle et un Raspberry Pi pour la détection des plaques d'immatriculation et la vérification de la validité des vignettes des véhicules

**Soutenu le 29 Juin 2025.** **Devant le jury composé de :**

| Nom et Prénom | Grade | | |
|---|---|---|---|
| Mme.KHOULALENE Nadjet | MCB | Université de Béjaia | Président |
| M. MOKTEFI Mohand | MCB | Université de Béjaia | Rapporteur |
| Mme.GASMI Badrina | MCB | Université de Béjaia | Examinatrice |
| M.BEDJOU Khaled | MCB | Université de Béjaia | Examinateur |
| M.SIDER Abderrahmane | MCA | Université de Béjaia | Examinatrice |

**Année Universitaire :** 2024/2025

# TABLE DES MATIÈRES

# LISTE DES FIGURES

# LISTE DES TABLEAUX

# LISTE DES ABRÉVIATIONS ET ACRONYMES

| | |
|---|---|
| ANPR | Automatic Number Plate Recognition |
| ACC | Accuracy |
| SSD | Single Shot MultiBox Detector |
| APT | Advanced Persistent Threat |
| Bi-LSTM | Bidirectional Long Short-Term Memory |
| RFBNe | Receptive Field Block Net |
| C2 | Command and Control (Commande et Contrôle) |
| CIC-IDS 2017 | Canadian Institute for Cybersecurity - Intrusion Detection System 2017 Dataset |
| CNN | Convolutional Neural Network |
| DAPT 2020 | Dataset for Advanced Persistent Threats 2020 |
| DARPA TC | Defense Advanced Research Projects Agency – Transparent Computing |
| DDoS | Distributed Denial of Service |
| DLP | Data Loss Prevention |
| DoS | Denial of Service (Déni de service) |
| FL | Federated Learning |
| FNR | False Negative Rate |
| FPR | False Positive Rate |
| GCN | Graph Convolutional Network |
| GNN | Graph Neural Network |
| GRU | Gated Recurrent Unit |

| | |
|---|---|
| GV-FL | Global Vision Federated Learning |
| HTTP | HyperText Transfer Protocol |
| HTTPS | HyperText Transfer Protocol Secure |
| IA | Intelligence Artificielle |
| IDS | Intrusion Detection System (Système de Détection d'Intrusions) |
| IoT | Internet of Things |
| LSTM | Long Short-Term Memory |
| ML | Machine Learning |
| MLP | Multi-Layer Perceptron |
| MITM | Man-In-The-Middle |
| NLP | Natural Language Processing |
| OpenAPT | Open-source dataset for APT scenarios |
| OS | Operating System |
| RAPID | Robust APT Detection and Investigation using Context-Aware Deep Learning |
| Recall | Rappel (Taux de détection) |
| RNN | Recurrent Neural Network |
| RRCF | Robust Random Cut Forest |
| SDN | Software Defined Network (Réseau Défini par Logiciel) |
| SHAP | SHapley Additive exPlanations |
| SQL | Structured Query Language |
| SQLi | SQL Injection |
| TPR | True Positive Rate |
| URL | Uniform Resource Locator |
| USB | Universal Serial Bus |
| VPN | Virtual Private Network (Réseau privé virtuel) |
| XFedHunter | Explainable Federated Learning Framework for APT Detection in SDN |
| Zero-Day | Vulnérabilité non encore connue ni corrigée au moment de l'attaque |

# INTRODUCTION GÉNÉRALE

Dans un contexte mondial marqué par la transformation numérique et le besoin croissant d'automatisation dans les secteurs de la sécurité et de la gestion des transports, les technologies basées sur l'intelligence artificielle (IA) offrent des solutions innovantes. L'automatisation du contrôle routier, notamment par la numérisation des plaques d'immatriculation et la gestion intelligente des vignettes, s'impose aujourd'hui comme une priorité stratégique. Ce mémoire s'inscrit dans cette dynamique, en présentant la réalisation d'une application basée sur l'IA et le Raspberry Pi pour la détection, la lecture et la gestion des plaques d'immatriculation algériennes dans le cadre du nouveau système de vignette numérique. Le premier chapitre est consacré aux généralités sur les concepts fondamentaux mobilisés dans ce projet. Il aborde l'intelligence artificielle, la vision par ordinateur, le traitement d'image, les réseaux de neurones convolutifs (CNN), les GANs ainsi que l'utilisation du Raspberry Pi comme solution embarquée. Ce cadre théorique constitue la base technologique indispensable pour comprendre les choix méthodologiques et les architectures adoptées dans le développement du système. Le deuxième chapitre présente un état de l'art détaillé des travaux liés à la numérisation de la vignette automobile, ainsi que des systèmes de reconnaissance automatique de plaques d'immatriculation (ANPR). Il examine le cadre réglementaire algérien, en particulier la réforme de la vignette numérique mise en œuvre à partir de 2025, et passe en revue plusieurs méthodes de détection, segmentation et reconnaissance des caractères utilisées dans la littérature, telles que YOLO, ResNet, LSTM, GRU ou encore les approches hybrides. Ce chapitre permet ainsi de situer notre travail dans le paysage scientifique et technologique existant. Le troisième chapitre décrit la méthodologie adoptée pour la réalisation de notre système. Il détaille les principes et évolutions de l'architecture YOLO (de YOLOv1 à YOLOv8), les caractéristiques des plaques d'immatriculation algériennes, la constitution des jeux de données, les processus d'entraînement des modèles YOLOv5n et YOLOv8s, ainsi que leur conversion pour les

plateformes embarquées. Ce chapitre aborde également le développement des trois interfaces applicatives : une application PC, un système embarqué sur Raspberry Pi 4, et une application mobile Android, en précisant les choix technologiques et les structures logicielles retenues. Le quatrième et dernier chapitre est dédié à l'évaluation expérimentale du système proposé. Il présente les résultats obtenus sur les différentes plateformes, les performances du pipeline de détection et de reconnaissance dans des conditions variées (jour, nuit, contre-jour, plaques sales), et compare notre approche à d'autres solutions existantes, notamment celles reposant sur l'OCR. Il met en lumière les points forts de notre approche à double modèle YOLO (v5n + v8s), tout en soulignant les limites identifiées et les perspectives d'amélioration, notamment par l'utilisation de cartes plus puissantes comme la NVIDIA Jetson.

CHAPITRE

# 1

# GÉNERALITÉS

## 1.1 Introduction

L'émergence de l'intelligence artificielle (IA) a profondément transformé notre manière d'interagir avec les systèmes numériques. En particulier, la vision par ordinateur, branche de l'IA, permet aujourd'hui aux machines de voir, comprendre et interpréter des images de manière automatique, ouvrant la voie à une multitude d'applications dans des domaines variés tels que la santé, la sécurité, l'industrie et surtout le transport. Grâce à des avancées significatives dans les domaines du traitement d'images et de l'apprentissage profond (Deep Learning), les ordinateurs peuvent désormais effectuer des tâches complexes comme la reconnaissance d'objets ou la lecture automatique de plaques d'immatriculation, avec une précision et une rapidité remarquables.

Ce chapitre propose un tour d'horizon des notions fondamentales qui sous-tendent ces technologies, en commençant par une présentation de l'intelligence artificielle et de la vision par ordinateur, avant d'aborder les principes du traitement d'images, des réseaux de neurones convolutifs (CNN) et des GANs. Nous mettons également en lumière l'importance de la Raspberry Pi, un ordinateur monocarte à la fois puissant, accessible et parfaitement adapté à la mise en œuvre de solutions intelligentes embarquées, notamment dans le cadre de projets de détection ou de surveillance visuelle.

## 1.2 Intelligence artificiel (IA)

L'intelligence artificielle (IA) est un domaine de l'informatique qui vise à créer des systèmes capables d'imiter certaines fonctions cognitives humaines, telles que l'apprentissage, la résolution de problèmes, la perception, ou encore la prise de décision. Elle repose sur des algorithmes capables de traiter de grandes quantités de données, d'en extraire des modèles et d'ajuster leur comportement en fonction de l'expérience acquise.[1]

## 1.3 Vision par ordinateur(VO)

La vision par ordinateur est un domaine de l'intelligence artificielle (IA) qui utilise le machine Learning et les réseaux neuronaux pour apprendre aux ordinateurs et aux systèmes à dériver des informations significatives à partir d'images numériques, de vidéos et d'autres entrées visuelles. Elle consiste à développer des algorithmes capables d'extraire des informations utiles à partir d'images numériques pour ensuite les exploiter à des fins diverses, comme la reconnaissance d'objets, la détection de formes, le suivi de mouvements, ou encore l'analyse de scènes[2].

## 1.4 Domaines d'application de VO dans le transport

La vision par ordinateur joue un rôle central dans le domaine du transport moderne. En permettant aux systèmes intelligents d'interpréter visuellement l'environnement, elle contribue à l'amélioration de la sécurité, de la navigation autonome, et de la gestion du trafic. Voici deux exemples majeurs d'application :

### 1.4.1 Voiture autonome

Les voitures autonomes utilisent la vision par ordinateur pour percevoir et comprendre leur environnement en temps réel. Grâce à des caméras embarquées, des capteurs et des algorithmes d'apprentissage profond, ces véhicules peuvent[3] :

— détecter les obstacles (piétons, véhicules, panneaux),

— reconnaître les marquages au sol et les feux de signalisation,

— prédire les comportements des autres usagers,

— prendre des décisions de conduite adaptées sans intervention humaine.

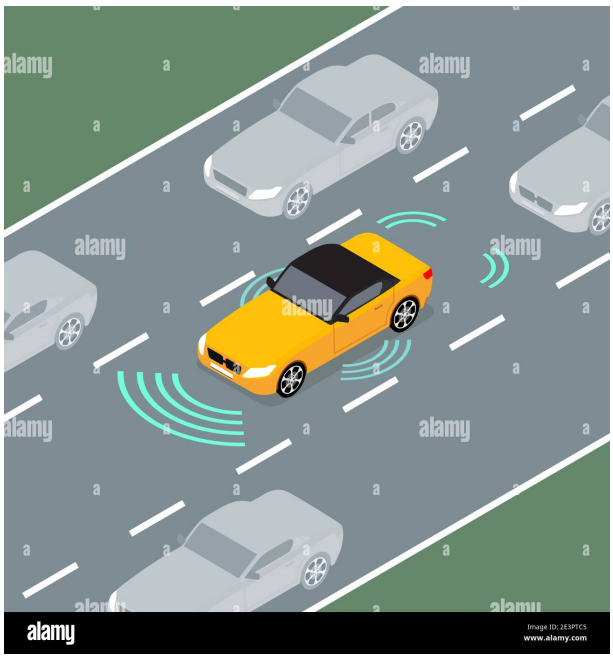


FIGURE 1.1 – Véhicule autonome

### 1.4.2 Sécurité routière

La vision par ordinateur est également un outil puissant pour renforcer la sécurité routière. Elle est utilisée dans plusieurs applications, telles que :

— la détection automatique des infractions (excès de vitesse, non-respect des feux rouges),

— la lecture automatique des plaques d'immatriculation (Automatic License Plate Recognition – ALPR),

— la surveillance du comportement des conducteurs (détection de fatigue ou d'inattention),

— l'analyse du flux de circulation pour prévenir les embouteillages ou les accidents.

Ces systèmes permettent non seulement d'automatiser les tâches de contrôle, mais aussi de collecter des données précieuses pour l'optimisation des infrastructures et des politiques de sécurité.[6]

FIGURE 1.2 – Utilisation de la VO pour la sécurité routière

## 1.5 Généralités sur traitement d'image

### 1.5.1 Définition de l'image

Une image est une représentation visuelle d'un objet, d'une scène ou d'un phénomène, captée ou créée pour transmettre une information perceptible par la vue. Elle peut être naturelle (vue à l'œil nu), capturée (photographie, vidéo), ou générée (dessin, graphique, image de synthèse)[90].

### 1.5.2 Types d'image

Il existe principalement deux types d'images numériques : les images matricielles et les images vectorielles. Chacune d'elles possède des caractéristiques spécifiques adaptées à des usages différents[4].

#### Les images matricielles (ou bitmap)

Une image matricielle est composée d'une grille de points appelés pixels. Chaque pixel possède une valeur représentant une couleur ou une intensité lumineuse. Plus la densité de pixels est élevée, plus l'image est précise, ce qui améliore la résolution. Cependant, une haute résolution implique également une augmentation de la taille du fichier et du temps de traitement. Les images matricielles sont les plus répandues dans le domaine du multimédia. Elles sont utilisées pour l'affichage sur écran, les photographies numériques, les vidéos, ainsi que dans les images obtenues par des appareils comme les caméras numériques ou les scanners.[5]

#### Les images vectorielles

Contrairement aux images matricielles, les images vectorielles ne sont pas constituées de pixels, mais de formes géométriques (lignes, cercles, polygones) définies par des formules mathématiques. Cette structure leur permet d'être agrandies ou réduites sans aucune perte de qualité, ce qui les rend idéales pour les logos, les schémas techniques ou les illustrations. Les images vectorielles occupent généralement moins d'espace mémoire et sont utilisées dans des logiciels de dessin industriel, de publication assistée par ordinateur (PAO), ou encore dans certains outils de traitement de texte.[5]


FIGURE 1.3 – Image matricielle et image vectorielle

### 1.5.3 Caractéristiques d'une image

Une image numérique possède plusieurs caractéristiques qui influencent sa qualité, son apparence, et son traitement par des systèmes informatiques. Ces caractéristiques sont essentielles pour comprendre comment manipuler et analyser une image.

#### Pixels

Le pixel (abréviation de "picture element") est l'unité de base d'une image numérique. Chaque pixel représente un point de l'image et contient des informations sur la couleur ou l'intensité lumineuse. L'ensemble des pixels forme une grille qui constitue l'image. Plus le nombre de pixels est élevé, plus l'image est détaillée.[7]

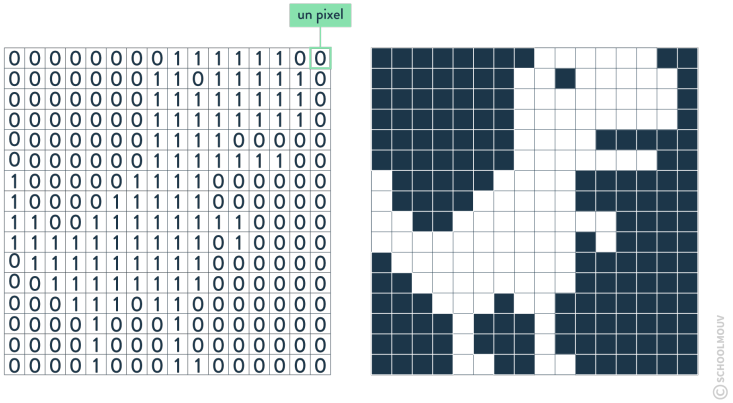


FIGURE 1.4 – Un pixel dans une image

#### Dimensions

Les dimensions d'une image sont exprimées en nombre de pixels selon la largeur et la hauteur. Par exemple, une image de 1920 × 1080 pixels possède 1920 colonnes et 1080 lignes. Ces dimensions déterminent la taille d'affichage de l'image sur un écran.[7]

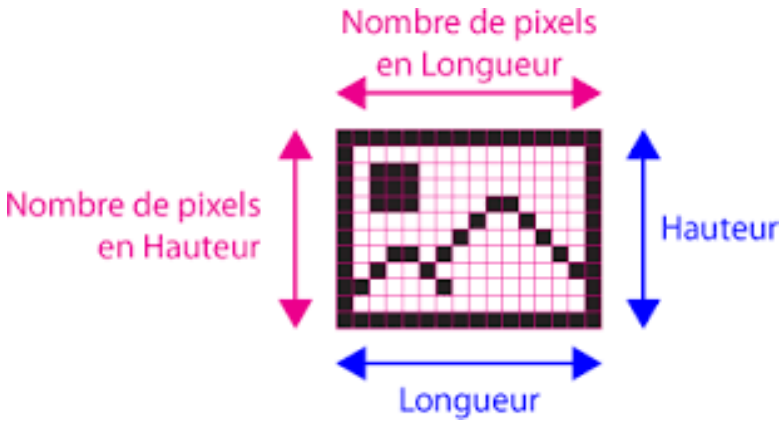


FIGURE 1.5 – Les dimensions d'une image

### Résolution

La résolution représente la densité de pixels dans une image. Elle est généralement exprimée en points par pouce (dpi, dots per inch). Une résolution élevée permet d'obtenir une image plus nette, surtout lors de l'impression. À l'inverse, une faible résolution peut rendre l'image floue ou pixellisée.[7]

### Luminance

La luminance correspond à l'intensité lumineuse d'un pixel. Elle est liée à la perception de la clarté d'une image. En traitement d'image, la luminance est souvent utilisée pour convertir une image en niveaux de gris, car elle reflète la composante lumineuse indépendamment de la couleur.[7]

### Contraste

Le contraste désigne la différence de luminance entre les parties claires et les parties sombres d'une image. Un bon contraste améliore la lisibilité des objets et des détails visuels. Un contraste faible rend l'image terne, tandis qu'un contraste trop élevé peut nuire à la précision des informations.[7]


FIGURE 1.6 – L'effet du contraste sur la clarté de l'image

### Histogramme

L'histogramme d'une image est une représentation graphique qui montre la répartition des niveaux de gris ou des intensités de couleur présents dans une image numérique. Il s'agit d'un outil fondamental en traitement d'image pour analyser le contraste, la luminosité, ou encore la qualité visuelle d'une image. Dans une image en niveaux de gris, l'histogramme affiche le nombre de pixels pour chaque valeur d'intensité, généralement comprise entre 0 (noir) et 255 (blanc). Une concentration des pixels

vers les valeurs basses indique une image sombre, alors qu'une concentration vers les valeurs hautes indique une image claire.[7]

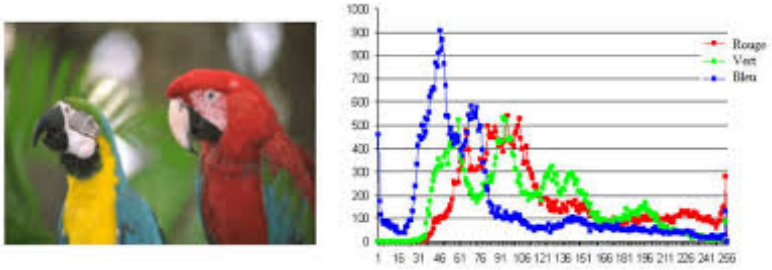


FIGURE 1.7 – L'histogramme d'une image

### 1.5.4 Types d'images selon le codage

#### Image à niveaux de gris

Une image en niveaux de gris est une image composée uniquement de variations de gris, allant du noir au blanc. Chaque pixel possède une valeur d'intensité, généralement codée sur 8 bits (valeurs de 0 à 255). Ce type d'image est fréquemment utilisé en traitement d'image pour simplifier les calculs tout en conservant l'information essentielle [7].

#### Image en couleurs

Les images en couleurs utilisent généralement le modèle RVB (Rouge, Vert, Bleu), où chaque pixel est représenté par trois composantes de couleur. La combinaison de ces composantes permet de reproduire une grande variété de couleurs. Ce type d'image est utilisé pour la photographie, la vidéo, les interfaces graphiques, etc[7].


FIGURE 1.8 – Image en couleurs

#### Image en noir et blanc

Une image en noir et blanc est une forme binaire de représentation où chaque pixel peut prendre uniquement deux valeurs : noir (0) ou blanc (1). Ce type d'image est utilisé dans certaines applications spécifiques, comme l'analyse de formes, la reconnaissance de texte (OCR) ou la segmentation [7].

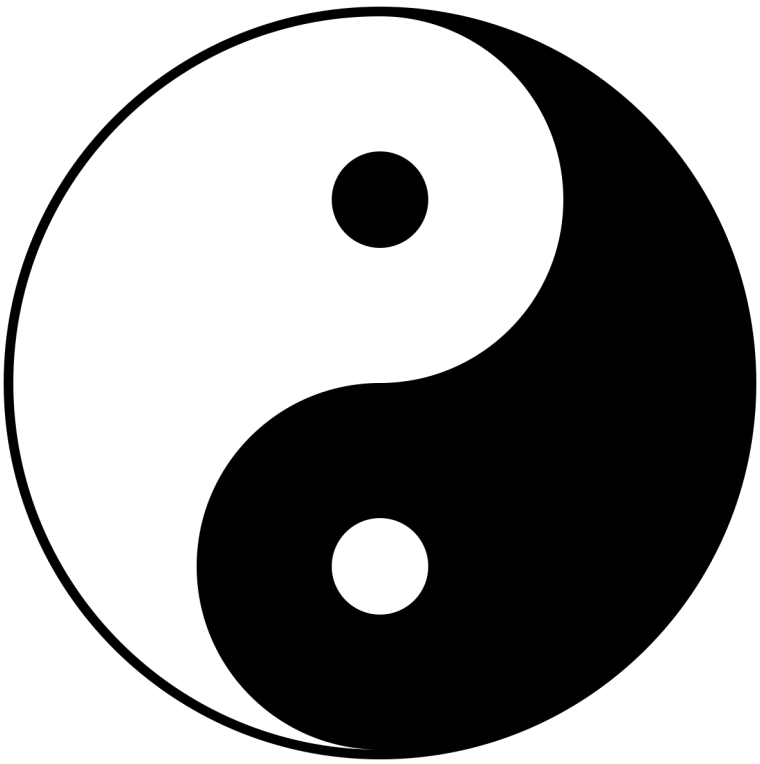

FIGURE 1.9 – Image en noir et blanc

### 1.5.5 Définition de traitement d'image

Le traitement d'images est une discipline relevant de l'informatique et des mathématiques appliquées, qui s'intéresse à l'étude, l'analyse et la transformation des images numériques. Son objectif principal est d'améliorer la qualité des images ou d'en extraire des informations pertinentes pour une utilisation ultérieure. Il s'agit d'un sous-domaine du traitement du signal, spécialement dédié aux données visuelles, comme les images fixes ou les séquences vidéo. Contrairement au traitement des signaux acoustiques ou électriques, le traitement d'image traite des informations bidimensionnelles (ou tridimensionnelles dans certains cas), ce qui implique des méthodes spécifiques d'analyse et de manipulation. Les applications sont nombreuses : imagerie médicale, vidéosurveillance, reconnaissance d'objets, véhicules autonomes, contrôle industriel, etc[8].

### 1.5.6 Techniques de traitement d'images

Le traitement d'images numériques repose sur un ensemble de techniques permettant d'améliorer, d'analyser ou d'interpréter visuellement une image. Ces techniques sont souvent appliquées selon un processus en plusieurs étapes allant du prétraitement à la reconnaissance automatique d'objets. Voici les principales :

### Prétraitement

Le prétraitement est une étape initiale visant à améliorer la qualité de l'image avant toute analyse plus poussée. Il peut inclure :

— La réduction du bruit (ex. : filtrage médian ou gaussien) ;

— L'amélioration du contraste ;

— La normalisation de la luminosité ;

— La mise à l'échelle ou redimensionnement de l'image.

Ces opérations permettent de préparer l'image pour les étapes suivantes, en assurant une qualité visuelle et une uniformité suffisantes.[9]

### Segmentation

La segmentation consiste à diviser une image en régions homogènes ou en objets distincts. L'objectif est d'isoler les parties importantes de l'image (par exemple une vignette, un visage, un texte, etc.).

Les techniques de segmentation peuvent être basées sur :

— La couleur ;

— L'intensité ;

— Les contours ou bords ;

— L'apprentissage automatique (méthodes supervisées ou non supervisées).

C'est une étape cruciale pour permettre la reconnaissance ou l'analyse des objets d'intérêt.[9]

### Détection et reconnaissance d'objet

Cette étape consiste à localiser (détection) et identifier (reconnaissance) des objets spécifiques dans une image. Par exemple : détecter une vignette de véhicule et reconnaître les caractères qu'elle contient.

Les méthodes modernes reposent principalement sur des réseaux de neurones convolutifs (CNN) et des algorithmes d'apprentissage profond comme[9] :

— YOLO (You Only Look Once) ;

— SSD (Single Shot Multibox Detector) ;

— R-CNN (Region-based CNN).

### Augmentation de données

L'augmentation de données est une technique utilisée en apprentissage automatique pour générer de nouvelles images à partir d'un jeu de données existant, afin d'améliorer la robustesse et la généralisation d'un modèle.

Elle consiste à appliquer des transformations comme :

— La rotation ;

— Le redimensionnement ;

— Le recadrage ;

— L'inversion horizontale ou verticale ;

— Les modifications de luminosité ou de contraste.

Cette technique est particulièrement utile dans les projets où le volume de données d'entraînement est limité.[9]

## 1.6 Apprentissage profond et CNN

L'apprentissage profond est un sous-domaine de l'intelligence artificielle qui repose sur l'utilisation de réseaux de neurones artificiels profonds pour apprendre automatiquement des représentations complexes à partir de données. Il a révolutionné le traitement d'image grâce à sa capacité à extraire et reconnaître automatiquement des motifs visuels.[9]

### 1.6.1 Apprentissage automatique (Machine Learning)

Le Machine Learning est une branche de l'intelligence artificielle qui permet aux systèmes d'apprendre à partir de données, sans être explicitement programmés. Il consiste à entraîner un modèle statistique pour effectuer des prédictions ou prendre des décisions basées sur des exemples.[10] Les techniques de machine learning sont souvent classées en trois catégories :

— Apprentissage supervisé (avec des données étiquetées),

— Apprentissage non supervisé (sans étiquettes),

— Apprentissage par renforcement.

### 1.6.2 Apprentissage profond (Deep Learning)

Le Deep Learning est une forme avancée d'apprentissage automatique qui repose sur des réseaux de neurones artificiels profonds. Grâce à leurs nombreuses couches, ces réseaux peuvent apprendre automatiquement des représentations hiérarchiques à partir des données.
En traitement d'image, le deep Learning est très performant pour des tâches comme la classification d'images, la reconnaissance d'objets, la segmentation, etc[11].

### 1.6.3 Réseaux de neurones artificiels ( Artificial Neural Network )

Un réseau de neurones artificiels (ANN) est un modèle inspiré du fonctionnement du cerveau humain. Il est constitué de neurones artificiels organisés en couches (entrée, cachées, sortie), chaque

neurone étant connecté à plusieurs autres.

Chaque connexion possède un poids ajusté pendant la phase d'apprentissage afin d'optimiser les prédictions du modèle.[11]

### 1.6.4 Réseaux de neurones convolutifs (CNN)

Les réseaux de neurones convolutifs (CNN) sont une catégorie de réseaux de neurones spécialement conçue pour le traitement des images. Ils utilisent des opérations de convolution pour extraire automatiquement des caractéristiques locales (bords, textures, formes) à partir d'une image.

Les CNN sont largement utilisés dans :

— La détection d'objets,

— La classification d'images,

— La reconnaissance faciale,

— La lecture automatique de plaques ou vignettes.

Leur architecture repose généralement sur des couches de convolution, de pooling, de normalisation et de couches entièrement connectées.[10]

### 1.6.5 Generative Adversarial Networks (GANs)

Les GANs sont une classe de réseaux de neurones introduite par Ian Goodfellow en 2014. Ils sont composés de deux réseaux : un générateur, qui crée des images, et un discriminateur, qui tente de distinguer les images réelles des images générées.

Les GANs sont utilisés dans :

— La génération d'images réalistes,

— L'augmentation de données,

— La restauration ou super-résolution d'images,

— L'anonymisation ou la modification d'images.

Ils sont particulièrement utiles dans les contextes où les données réelles sont rares ou difficiles à collecter.[11]

## 1.7 Carte raspberry PI

La Raspberry Pi est un ordinateur monocarte de petite taille, comparable à une carte de crédit, conçu pour être connecté à un écran (moniteur ou téléviseur) et utilisé avec un clavier et une souris classiques. Malgré ses dimensions réduites, elle offre une gamme complète de fonctionnalités similaires à celles d'un ordinateur de bureau classique, telles que la navigation Internet, la lecture de vidéos

en haute définition, la gestion de documents (traitement de texte, feuilles de calcul) ou encore les jeux simples. Ce qui rend la Raspberry Pi particulièrement attractive dans le domaine des projets informatiques et embarqués, c'est sa capacité à interagir avec l'environnement externe via ses broches GPIO (General Purpose Input/Output). Cette caractéristique lui permet de servir de base à une grande variété d'applications dans les domaines de la robotique, de la domotique, de l'IoT (Internet des objets) et de l'intelligence artificielle.[12]


FIGURE 1.10 – Raspberry PI

### 1.7.1 Caractéristiques principales du Raspberry Pi 4

Voici les principales spécifications techniques du modèle Raspberry Pi 4, largement utilisé dans les projets embarqués modernes :

**Processeur**

SoC Broadcom BCM2711, quad-core Cortex-A72 (ARMv8) 64 bits, cadencé à 1,5 GHz.

**Mémoire vive (RAM)**

Disponible en versions de 2 Go, 4 Go et 8 Go de SDRAM LPDDR4-3200.

**Connectivité**

— Wi-Fi bi-bande (2,4 GHz et 5 GHz) – norme IEEE 802.11 b/g/n/ac,

— Bluetooth 5.0 / BLE,

— Port Ethernet Gigabit,

— Ports USB 2.0 et USB 3.0.

**Affichage et multimédia**

— Deux ports micro HDMI prenant en charge la double sortie vidéo jusqu'à une résolution 4K,

— Décodage matériel des formats H.265 (4Kp60) et H.264 (1080p60).

**Systèmes d'exploitation pris en charge**

— Raspberry Pi OS (anciennement Raspbian),

— Ubuntu,

— Autres distributions Linux compatibles.

**Domaines d'application**

Domotique, robotique, projets IoT, systèmes embarqués, centres multimédia, jeux rétro, apprentissage de la programmation, vision artificielle, etc.

## 1.8 Conclusion

L'alliance entre l'intelligence artificielle, la vision par ordinateur et les dispositifs matériels comme la Raspberry Pi constitue aujourd'hui une base solide pour le développement de systèmes intelligents capables de percevoir et d'interagir avec leur environnement. Ce chapitre a permis de comprendre comment les différentes briques technologiques –du traitement d'image aux réseaux de neurones convolutifs, en passant par les techniques d'augmentation de données ,s'intègrent pour former des solutions efficaces et performantes dans le domaine de la reconnaissance visuelle.

La Raspberry Pi, par sa compacité, sa modularité et sa compatibilité avec des outils d'IA modernes, se positionne comme une plateforme idéale pour expérimenter et déployer des projets d'analyse d'image en temps réel. Dans la suite de ce travail, ces fondements théoriques serviront à la conception d'une application concrète de lecture automatique de plaques d'immatriculation, illustrant ainsi l'interconnexion entre les concepts explorés et leur mise en pratique dans un contexte réel.

CHAPITRE

# 2

# TRAVAUX CONNEXES

## 2.1 Introduction

Ce chapitre présente un état de l'art approfondi des travaux et systèmes en lien avec la numérisation de la vignette automobile, en particulier dans le contexte algérien. L'objectif est de situer notre travail dans un cadre technologique et réglementaire cohérent en mettant en lumière les fondements, les pratiques existantes et les solutions en cours de déploiement. Nous débutons par l'étude de la numérisation de la vignette, en précisant son contenu, ses modalités d'enregistrement, et les mécanismes d'échange d'informations entre institutions concernées. Ensuite, nous exposons les sanctions prévues en cas de non-conformité, conformément aux textes de loi applicables à partir de 2025, avant de traiter des modalités de présentation de la vignette lors des contrôles routiers.

Nous analysons également l'état actuel de la mise en œuvre en Algérie, notamment via l'application mobile Qassimatouka destinée à l'acquisition de la vignette électronique. Le cœur du chapitre est ensuite consacré au système ANPR/RAPI (Automatic Number Plate Recognition / Reconnaissance Automatique de Plaques d'Immatriculation), qui constitue la base technologique d'un contrôle automatisé, sans contact et en temps réel. Nous abordons sa définition, son évolution, ses principes de fonctionnement, les technologies sous-jacentes, ses domaines d'application, ses avantages, ainsi que ses limites dans un contexte national.

Ce chapitre se conclut par une clarification des objectifs spécifiques du présent travail, ainsi qu'une présentation synthétique des techniques utilisées pour la reconnaissance automatique de plaques d'immatriculation, qui seront détaillées dans les chapitres méthodologiques ultérieurs.

## 2.2 Numérisation de la vignette

### 2.2.1 La vignette

La vignette automobile est un impôt annuel obligatoire que doivent payer tous les propriétaires de véhicules à moteur immatriculés en Algérie. Elle constitue une taxe sur la circulation routière, destinée à financer en partie l'entretien des routes et les infrastructures publiques liées au transport[13].

— Elle est liée à chaque véhicule, en fonction de son type, de sa puissance fiscale (en chevaux) et de son type de carburant (essence ou diesel) [13].

— Elle doit être acquise chaque année, généralement entre le 1er mars et le 31 mars [13].

— Depuis mars 2025, la vignette est entièrement numérisée : elle s'obtient via une plateforme en ligne officielle appelée Qassimatouka [13].

### 2.2.2 Contenu de la vignette numérique

La vignette numérique contient[13] :

— Le numéro d'immatriculation.

— La catégorie du véhicule.

— Le montant payé.

— La date d'acquisition.

— Un QR code ou un identifiant unique vérifiable.


FIGURE 2.1 – Exemple d'une vignette véhicule algérienne

### 2.2.3 Enregistrement et gestion des données

Les informations relatives aux vignettes automobiles (matricule, Nom et prénom de propriétaire...) sont stockées dans une base de données nationale, gérée par la Direction Générale des Impôts (DGI). Cette base de données est interconnectée avec les services du ministère de l'Intérieur, permettant une vérification efficace des informations relatives au véhicule et à son propriétaire.[13]

### 2.2.4 Interconnexion avec d'autres institutions

Le système de vignette numérique est le fruit d'une collaboration entre plusieurs institutions, dont [13] :

— La Direction Générale des Impôts, initiatrice du projet.

— Le ministère des Finances, garant de la transformation numérique du secteur fiscal.

— Le ministère de l'Intérieur et des Collectivités locales, impliqué dans la gestion des données des véhicules.

— Le ministère des Postes et Télécommunications, chargé de l'infrastructure numérique.

— Les institutions bancaires et monétiques, partenaires techniques pour les transactions sécurisées.

## 2.3 Sanctions en cas de non-conformité de la vignette

En Algérie, la loi de finances 2025 a introduit des sanctions spécifiques en cas de non-paiement ou de non-présentation de la vignette automobile numérique. Ces mesures visent à renforcer le respect des obligations fiscales liées à la détention d'un véhicule. [14]

### 2.3.1 Sanctions prévues en 2025

Selon **l'article 58 de la loi de finances 2025**, les sanctions suivantes sont applicables :

#### Saisie temporaire de la carte grise

— En cas de non-présentation de la vignette numérique lors d'un contrôle routier, la grise du véhicule peut être retirée pour une durée de sept (7) jours[15].

— La restitution de la carte grise est conditionnée à la présentation de la vignette et de son justificatif de paiement.[15]

#### Amende équivalente à 50 % du montant de la vignette

Si la vignette n'est pas présentée accompagnée de son reçu de paiement, une amende fiscale correspondant à 50 % du montant de la vignette est appliquée.[15]

## 2.4 Présentation de la vignette lors des contrôles

La Direction Générale des Impôts (DGI) précise que la vignette acquise en ligne n'a pas besoin d'être apposée sur le pare-brise du véhicule. Cependant, elle doit être présentée, soit en format numérique, soit en format papier, lors des contrôles routiers effectués par les agents habilités [13].

## 2.5 Acquisition et téléchargement de la vignette

### 2.5.1 l'application Qassimatouka

Qassimatouka est une application web développée par la DGI dans le cadre de la modernisation des services fiscaux en Algérie. Elle permet aux citoyens d'acquérir leur vignette automobile en ligne, sans avoir à se déplacer. Ce service est le fruit d'une collaboration entre plusieurs institutions, dont le ministère des Finances, le ministère de l'Intérieur et des Collectivités locales, le ministère des Postes et Télécommunications, ainsi que les institutions bancaires et monétiques[16].

### 2.5.2 Acquisition et téléchargement de la vignette

Le processus d'acquisition de la vignette via **Qassimatouka** est conçu pour être simple et rapide[13] :

1. **Accès à la plateforme** :

   Connectez-vous au site qassimatouka.mf.gov.dz depuis un ordinateur ou un smartphone.

2. **Saisie des informations du véhicule** :

   — Entrez le numéro d'immatriculation de votre véhicule.

   — La plateforme récupérera automatiquement les données associées à ce numéro, telles que le genre du véhicule, le nombre de sièges, la puissance, le poids total en charge, l'année de mise en circulation et la wilaya[17].

3. **Saisie des informations du propriétaire** :

   Fournissez les informations personnelles requises, notamment le numéro d'identification nationale, la date et le lieu de naissance[18].

4. **Affichage du tarif de la vignette** :

   La plateforme calculera automatiquement le montant de la vignette en fonction des caractéristiques de votre véhicule.

5. **Paiement sécurisé** :

   Effectuez le paiement en ligne via une carte interbancaire (CIB) ou une carte Edahabia[19].

6. **Téléchargement de la vignette** :

   Une fois le paiement validé, vous pourrez télécharger immédiatement la vignette ainsi que le reçu de paiement.

## 2.6 Le Système ANPR/RAPI

### 2.6.1 Définition

La RAPI (ou LAPI, en anglais ANPR pour Automatic Number Plate Recognition) est un système de vision par ordinateur qui identifie automatiquement les véhicules par la lecture de leurs plaques d'immatriculation[20]. Il s'appuie sur la capture d'images (caméra vidéo ou photo) et l'application d'algorithmes de traitement d'image et de reconnaissance optique de caractères (OCR) pour extraire le numéro d'immatriculation [21]. En pratique, la RAPI détecte d'abord la plaque dans l'image puis convertit le texte visuel en données alphanumériques interprétables. Ce mécanisme sans intervention humaine permet d'intégrer directement l'immatriculation dans une base de données ou un système d'information, facilitant de nombreuses applications de surveillance et de gestion du trafic[21].

### 2.6.2 Historique et évolution

Les premiers travaux sur l'ANPR remontent aux années 1970. En 1976, la division scientifique de la police britannique inventa le concept de reconnaissance de plaques, avec des prototypes opérationnels dès 1979[22]. Les premiers systèmes industriels furent mis en œuvre peu après (EMI Electronics, CRS), et la première arrestation basée sur l'ANPR eut lieu en 1981[22]. Le succès de ces expériences conduisit à une adoption progressive : par exemple, en 2005, l'ANPR permit de résoudre un crime de meurtre [22]. Avec le temps, les avancées matérielles et logicielles ont accéléré la capacité du système. En 2005, des caméras pouvaient déjà lire près d'une plaque par seconde sur des véhicules roulant à 160 km/h [23]. Dans la dernière décennie, l'arrivée des algorithmes d'apprentissage profond a révolutionné l'ANPR : des architectures de réseaux de neurones convolutionnels (CNN) telles que YOLO ou SSD sont désormais couramment employées pour la détection et la lecture de plaques, améliorant sensiblement la précision et la robustesse. [20]

### 2.6.3 Principes de fonctionnement technique

Un système ANPR type suit les grandes étapes suivantes (Figure ci-après)[21] :

• **Acquisition d'image** : Une caméra fixe ou embarquée capture le véhicule (avant ou arrière). On utilise souvent des caméras haute résolution équipées d'un éclairage infrarouge (LED IR) pour assurer une vision claire nuit et jour[23]. L'image est numérisée et, le cas échéant, prétraitée (conversion en niveaux de gris, filtrages, binarisation) pour améliorer le contraste plaque/fond.

• **Détection/extraction de la plaque** : Un algorithme localise la zone de la plaque dans l'image. Traditionnellement on employait des techniques de segmentation par contours ou par histogrammes de projection, mais les systèmes modernes s'appuient sur des détecteurs CNN (Ex : YOLO, SSD...) qui traitent la plaque comme un objet à repérer[24]. Cette étape isole l'identifiant visuel de la plaque

pour traitement ultérieur.

• **Segmentation des caractères** : L'image de la plaque extraite est ensuite segmentée pour isoler chaque caractère (lettre ou chiffre). Des méthodes classiques (connections de pixels, contours de caractères) ou des approches apprises sont utilisées. On obtient un ensemble de glyphes individuels correspondant aux symboles imprimés sur la plaque[21].

• **Reconnaissance optique (OCR)** : Chaque caractère isolé est converti en symbole alpha-numérique via un OCR. Des bibliothèques comme Tesseract ou des réseaux de neurones spécialisés sont couramment utilisées pour cette étape[24]. Le système reconstitue ainsi la chaîne de caractères complète de la plaque.

• **Post-traitement et intégration** : Le texte reconnu est vérifié (application de règles de format, correction d'erreurs probables) puis transmis à un système backend. Les données (numéro de plaque, horodatage, image) sont stockées en base de données pour analyses ultérieures. Un serveur d'applications peut alors déclencher des actions (ouvertures de barrière, relevé de paiement, alerte policière, etc.)[21]. Par exemple, après reconnaissance réussie, l'événement est horodaté et enregistré dans le système central de l'autoroute ou du parking [21].

### 2.6.4 Technologies utilisées

Les ANPR exploitent aujourd'hui des composants logiciels et matériels modernes :

• **Réseaux de neurones (Deep Learning)** : Les détecteurs de plaques et de caractères font souvent appel à des CNN. Par exemple, YOLO (« You Only Look Once ») est fréquemment utilisé pour localiser rapidement la plaque dans l'image, et SSD (Single Shot Multibox Detector) peut être employé pour la segmentation de caractères[24]. Ces modèles CNN sont entraînés sur de grands jeux de données de plaques pour reconnaître différentes mises en forme et conditions d'éclairage.

• **Bibliothèques logicielles** : OpenCV est la bibliothèque standard pour le traitement d'image (filtrage, transformations, détection de contours). Tesseract OCR (open-source) est largement utilisée pour convertir les glyphes isolés en texte[24]. D'autres outils (TensorFlow, PyTorch) peuvent servir à implémenter les modèles d'apprentissage.

• **Matériel** : Des caméras infrarouges dédiées facilitent la capture nocturne en illuminant la plaque sans éblouir le conducteur[23]. Le système peut être déployé sur des plateformes embarquées à faible coût (par exemple un Raspberry Pi 4 couplé à une caméra) pour un traitement en temps réel sur site [20]. À l'autre extrémité, des GPU et des serveurs puissants accélèrent l'entraînement des réseaux de neurones et permettent de traiter des flux vidéo à haute cadence.

### 2.6.5 Domaines d'application

L'ANPR est utilisée dans de nombreux contextes liés à la gestion de véhicules et à la sécurité :

• **Sécurité routière et application de la loi** : contrôle automatisé de la vitesse, détection de circulation sur voies réservées, repérage de véhicules volés ou recherchés.[22]

• **Péages électroniques** : facturation automatique sur autoroute sans arrêt aux barrières. L'ANPR remplace les tickets et transpondeurs en identifiant simplement la plaque de chaque voiture.

• **Parkings intelligents** : contrôle d'accès aux parkings d'entreprise ou publics (ouverture de barrières pour abonnés) et calcul des durées de stationnement sans avoir à saisir manuellement un ticket.[22]

• **Logistique et transport** : suivi de flottes de véhicules et traçabilité des livraisons. Par exemple, une entreprise peut enregistrer automatiquement les entrées/sorties de camions depuis son dépôt grâce à la lecture des plaques.

• **Surveillance urbaine et gestion du trafic** : monitoring du trafic routier (comptage de véhicules, estimation de vitesses moyennes) et contrôle du respect des règles (passage au feu rouge, identification des contrevenants)[21]. L'ANPR intervient également dans des systèmes de vidéosurveillance pour la sécurité publique et la lutte contre la criminalité routière.

### 2.6.6 Avantages du système ANPR

Le principal atout de l'ANPR est son automatisation totale et sa non-intrusivité. Il n'est pas nécessaire de remettre un badge ou un ticket, chaque véhicule est identifié simplement par sa plaque[21].

De plus, l'usage de caméras infrarouges permet un fonctionnement fiable de jour comme de nuit [23]. Les systèmes modernes traitent très rapidement les images, par exemple, des caméras actuelles peuvent analyser plus d'une plaque par seconde même à haute vitesse[23]. L'ANPR génère également de précieuses métadonnées (date/heure, localisation GPS éventuelle) pour le suivi du trafic [21]. Enfin, son coût peut être faible : des solutions « tout-en-un » basées sur Raspberry Pi ou des caméras intelligentes existent sur le marché, offrant un déploiement à moindre frais comparé à des systèmes de télépéage à étiquette RFID [21].

### 2.6.7 Limites et contraintes

Malgré ces avantages, l'ANPR souffre de certaines limitations intrinsèques. Ses performances dépendent fortement de la qualité des images et des conditions d'acquisition, plaques sales, éraflées ou partiellement masquées peuvent échapper au système[25]. Les variations de format (tailles, polices, dispositions selon les pays) compliquent également la reconnaissance, nécessitant souvent un entraînement local des modèles CNN [25]. En outre, des scènes complexes (éblouissement, reflets, zones d'ombres) et le flou de mouvement peuvent dégrader le taux de réussite de la détection[25]. La dépendance au matériel informatique est un inconvénient, pour atteindre un faible taux d'erreur, un ANPR réaliste exige parfois des GPU performants et des caméras spécialisées coûteuses. Enfin, comme toute solution automatisée, l'ANPR n'est pas infaillible, des cas de fausses lectures ou d'erreurs d'identification ont

été rapportés, ce qui impose de prévoir un filtrage ou une supervision humaine pour valider certaines alertes.

## 2.7 Objectifs de travail

Afin de renforcer la numérisation du processus de vérification des vignettes automobiles en Algérie, l'intégration d'un système ANPR/RAPI (Automatic Number Plate Recognition / Reconnaissance Automatique de Plaques d'Immatriculation) vise à atteindre plusieurs objectifs stratégiques :

1. **Automatiser le contrôle routier et la vérification en temps réel** : Le système mis en place permettra de détecter automatiquement les plaques d'immatriculation, de lire leur contenu via des techniques de reconnaissance optique (telles que YOLO ou OCR), et de vérifier en temps réel la validité de la vignette à travers une base de données nationale interconnectée, gérée par la Direction Générale des Impôts (DGI). Cette automatisation réduit l'intervention humaine tout en améliorant la fiabilité du contrôle.

2. **Réduire le temps de contrôle et limiter les fraudes** Grâce à l'automatisation, les contrôles s'effectuent de manière rapide, fluide et précise. Ce système limite considérablement les erreurs humaines et élimine les tentatives de falsification ou de présentation de documents non conformes.

3. **Anticiper les sanctions et informer les usagers** Un mécanisme de notification automatisée (via application mobile) permettra d'alerter les automobilistes avant l'expiration de leur vignette. Cette fonctionnalité vise à réduire les cas d'oubli involontaire et à diminuer les infractions liées au non-renouvellement.

4. **Détecter en temps réel les infractions sans interruption de la circulation**

Le système ANPR assure une surveillance continue et passive de la circulation. Il permet d'identifier instantanément les véhicules circulant sans vignette valide, sans nécessité d'arrêt systématique, contribuant ainsi à la fluidité du trafic tout en assurant une couverture élargie des contrôles.

5. **Garantir l'équité et la transparence dans les sanctions** Chaque infraction détectée est horodatée et enregistrée automatiquement. Cette traçabilité permet aux autorités de disposer de preuves objectives en cas de litige, assurant ainsi une application équitable et transparente des sanctions prévues par la réglementation.

6. **Supprimer l'obligation de présentation physique de la vignette ou du reçu** Le système élimine la nécessité de présenter un justificatif imprimé lors des contrôles. Ainsi, les automobilistes sont protégés contre les pertes de documents physiques ou les sanctions injustifiées dues à l'oubli d'un reçu valide. La vérification se fait directement via le numéro de la plaque connecté à la base de données centrale.

7. **Moderniser la gestion des contrôles routiers en Algérie** Ce dispositif numérique permet une interconnexion fluide entre la DGI, les forces de l'ordre et les usagers. Il crée une boucle de gestion

intelligente favorisant la modernisation de la circulation routière, la transparence administrative, ainsi que la lutte contre la fraude et les infractions.

## 2.8 Techniques et méthode en reconnaissance automatique de la plaque d'immatriculation

### 2.8.1 Méthode basée sur ResNet et LSTM

Naaman Omar et ses collaborateurs en 2022[26] ont développé une approche complète et robuste pour la détection et la reconnaissance des plaques d'immatriculation adaptées aux véhicules circulant dans le nord de l'Irak. Leur système repose sur un pipeline en trois étapes : segmentation de la plaque, segmentation des caractères, puis reconnaissance à l'aide d'un classifieur LSTM, avec des performances très élevées atteignant 98,51% pour les chiffres arabes et 100% pour les villes.

#### Détection et segmentation de la plaque d'immatriculation

La première étape consiste à détecter et segmenter la plaque d'immatriculation à partir d'une image de véhicule. Pour cela, les auteurs utilisent l'architecture SegNet, un réseau de segmentation sémantique basé sur un encodeur-décodeur profond. Cette méthode permet d'étiqueter chaque pixel de l'image pour isoler précisément les trois zones distinctes de la plaque : le numéro, la ville et le pays. Comme la région du pays est identique pour toutes les plaques, elle est éliminée à ce stade.

L'avantage de SegNet est sa capacité à conserver les contours grâce aux indices de max-pooling réutilisés dans le décodeur, ce qui améliore la précision de segmentation même dans des conditions difficiles (pluie, poussière, faible éclairage). Les images utilisées ont été capturées dans diverses conditions environnementales, renforçant la robustesse de la méthode.

#### Segmentation des caractères

Une fois la plaque détectée, une série de techniques classiques de traitement d'image est appliquée pour extraire les caractères arabes :

1. Conversion en niveaux de gris, afin de simplifier l'image.

2. Suppression du bruit, par application d'un filtre de Gauss, pour améliorer la lisibilité des contours.

3. Binarisation avec Otsu, une méthode automatique de seuillage qui sépare le texte du fond.

4. Opérations morphologiques, notamment érosion et dilatation, pour éliminer les petits artefacts et connecter les pixels des caractères.

5. Boîte de délimitation glissante, utilisée pour détecter chaque caractère individuel de la plaque. Cette boîte est ajustée pour encapsuler précisément les chiffres arabes, ce qui est essentiel avant l'étape

de reconnaissance.

Les régions contenant les caractères extraits, ainsi que les noms des villes, sont ensuite redimensionnées en 224×224 pixels, conformément à la taille d'entrée du réseau ResNet-18.

**Reconnaissance des caractères**

L'étape suivante consiste à reconnaître les caractères extraits. Cette tâche est divisée en deux sous-processus :

• Extraction de caractéristiques profondes avec ResNet-18, un réseau neuronal résiduel à 18 couches. Le principe du réseau résiduel est de permettre un entraînement efficace même pour des architectures très profondes, grâce aux connexions de saut (skip connections) qui évitent le problème du gradient qui disparaît.

La couche fc1000 du ResNet génère un vecteur de 1000 dimensions pour chaque caractère ou nom de ville extrait.

• Ces vecteurs sont ensuite introduits dans un réseau LSTM (Long Short-Term Memory), plus précisément une variante bi-directionnelle (BiLSTM), capable de capturer des dépendances dans les deux sens d'une séquence de caractères.

Le modèle LSTM est entraîné avec l'optimiseur ADAM, un taux d'apprentissage initial de 0.0001, et des mini-lots de 25 images. Le taux d'apprentissage est réduit tous les 1000 itérations, et le modèle converge en environ 30 époques.

**Résultats**

L'approche a été testée sur un jeu de données original composé de 600 images de véhicules prises avec des appareils photo reflex dans des conditions variées (neige, poussière, faible luminosité). Les résultats montrent une précision de 98,51% pour la reconnaissance des chiffres arabes et 100% pour la reconnaissance des villes (Erbil, Duhok, Sulemania).

**Analyse critique**

Cette approche présente plusieurs points forts :

Combinaison puissante de SegNet pour la détection, ResNet pour l'extraction et LSTM pour la reconnaissance séquentielle ;

Excellente gestion des caractères arabes, souvent ignorés dans les travaux ANPR ;

Résultats très compétitifs, supérieurs à plusieurs méthodes de l'état de l'art comme celles de Cheang et Shivakumara.

Cependant, l'approche dépend fortement de la qualité de la segmentation sémantique initiale. Toute erreur dans cette étape impacte la suite du processus. En outre, bien que la segmentation des

caractères soit efficace, elle repose encore sur une fenêtre glissante manuelle, pouvant être optimisée par une approche entièrement automatique dans les travaux futurs.

### 2.8.2 Méthode ALPR en conditions réelles avec RFBNet et YOLO

Le système ALPR présenté par Anmol Singhal et Navya Singhal en 2024[27]a été conçu pour des applications réelles et embarquées, notamment dans des véhicules de patrouille équipés de caméras embarquées. Leur approche couvre toutes les étapes de l'ANPR : détection, segmentation, reconnaissance et reconstruction de la plaque, tout en visant une exécution en temps réel sur des dispositifs à faible consommation.

#### Détection de la plaque d'immatriculation

La détection est réalisée avec le RFBNet (Receptive Field Block Net), un détecteur d'objets basé sur SSD, renforcé pour améliorer à la fois la vitesse et la précision grâce à des blocs à champs récepteurs multi-échelles. Le réseau est entraîné avec :

- des images redimensionnées à 512 × 512 pixels,
- des mini-lots de 64 images,
- un seuil IoU de 0.5 pour la suppression des détections redondantes.

Le réseau parvient à une précision de 86,1 % sur un jeu de données original de plus de 16 000 images extraites de vidéos enregistrées sur les routes de Singapour. Cette performance est atteinte avec une vitesse de 56 FPS sur GPU Tesla P4, et de 7,5 FPS sur Jetson TX2, ce qui permet son usage embarqué.

#### Segmentation des caractères

Après détection, la plaque est extraite puis transmise à une étape de segmentation et reconnaissance combinée à l'aide d'un réseau YOLO optimisé. Étant donné les contraintes réelles (flou, faible résolution, obliquité), plusieurs ajustements ont été faits :

- Les plaques de petite taille (moins de 50 pixels) sont filtrées.
- Un réseau YOLO modifié est utilisé avec une taille d'entrée personnalisée :

- 240 × 80 pour les plaques monolignes,

- 288 × 200 pour les plaques bilignes.

- Le nombre de couches de max-pooling est réduit (3 au lieu de 5) pour conserver les détails fins.
- La sortie du réseau est augmentée à 36 × 25, pour permettre une meilleure localisation des petits caractères.

Le réseau détecte chaque caractère individuellement et retourne les coordonnées de leur boîte englobante, utilisées pour trier et organiser les caractères avant la reconnaissance.

**Reconnaissance des caractères**

La reconnaissance des caractères est également effectuée par le réseau YOLO modifié, qui classifie chaque caractère parmi 35 classes (les 10 chiffres et les 26 lettres capitales, avec fusion des classes 0/O). Le système tient aussi compte des erreurs courantes de classification dues à la similarité visuelle :

• 5↔ S, 8 ↔ B, 2 ↔Z, 4 ↔ A, 1 ↔ I/T/L.

Pour corriger ces erreurs, un post-traitement heuristique est appliqué :

• Chaque caractère est replacé selon sa position attendue dans le format des plaques singapouriennes.

• Si un chiffre est détecté dans un emplacement réservé à une lettre, il est remplacé par une lettre similaire (et inversement pour les lettres).

Exemple : Si “5” est détecté en début de plaque, il est remplacé par “S”.

Enfin, une étape de reconstruction de séquence organise les caractères détectés dans le bon ordre, en tenant compte de la disposition (monoligne ou biligne), à partir de leurs coordonnées x et y.

**Résultats**

• Précision de détection des plaques (LPD) :

Avec RFBNet : 86

Vitesse : jusqu'à 64 FPS sur GPU Tesla P4 et 7,7 FPS sur Jetson TX2.

• Reconnaissance des caractères (CR) :

Exactitude complète (tous caractères corrects) : 67 %.

Exactitude partielle (tous sauf un caractère correct) : 89 %.

Exactitude partielle (tous sauf deux caractères corrects) : 95 %.

• Performance en temps réel : le système atteint jusqu'à 68 FPS sur Nvidia 1080Ti, montrant sa capacité pour une utilisation embarquée sur des véhicules de patrouille

**Analyse critique**

Ce travail se distingue par plusieurs aspects :

Système end-to-end intégré capable de fonctionner en conditions urbaines réelles, avec fort bruit visuel ;

Optimisations techniques pour adapter YOLO aux contraintes de formes de plaques spécifiques (mono et bilignes) ;

Haute vitesse d'exécution, compatible avec du matériel embarqué peu coûteux ;

Module heuristique intelligent, comblant les limites des modèles CNN avec des règles adaptées au contexte régional.

Cependant, le système est fortement dépendant de la structure locale des plaques (ici Singapour), et le module heuristique ne serait pas directement applicable à d'autres pays sans adaptation. De plus, la reconnaissance atteint 67 % en précision stricte, ce qui pourrait être amélioré avec des modèles spécialisés comme les Spatial Transformer Networks ou des réseaux orientés par parties (Parts-based CNNs), mentionnés comme perspectives.

### 2.8.3 Méthode basée sur YOLOv3, ImageAI et OCR Tesseract

V. Gnanaprakash et ses collègues [28] ont proposé en 2021 une méthode complète pour la reconnaissance automatique des plaques d'immatriculation (ANPR), conçue pour fonctionner en temps réel à partir de vidéos de surveillance routière. Leur approche repose sur une combinaison de YOLOv3 pour la détection, de traitements d'image classiques pour la segmentation, et du moteur OCR Tesseract pour la reconnaissance des caractères. Le système a été optimisé pour être déployé sur des dispositifs embarqués comme la Jetson Nano.

#### Détection de la plaque d'immatriculation

La première étape consiste à détecter les véhicules dans des séquences vidéo extraites de caméras de surveillance. Pour cela, les auteurs utilisent YOLOv3, entraîné sur le dataset Stanford Cars, avec plus de 16 000 images. Chaque image est analysée pour extraire les voitures, qui sont ensuite redécoupées afin de localiser précisément les plaques d'immatriculation. Cette tâche est réalisée à l'aide du framework **ImageAI**, qui simplifie la gestion des modèles de détection et permet une exécution rapide, adaptée aux contraintes du traitement embarqué.

#### Segmentation des caractères

Une fois la plaque extraite, une série de traitements d'image est appliquée pour segmenter les caractères. L'image est d'abord convertie en niveaux de gris, puis un **filtrage bilatéral** est appliqué pour préserver les bords tout en éliminant le bruit. La **détection de contours de Canny** est ensuite utilisée, suivie par une opération de suppression des faux bords par double seuillage et hystérésis. Les **contours détectés** sont analysés pour isoler chaque caractère sous forme de sous-images, prêtes pour la reconnaissance.

#### Reconnaissance des caractères

Les caractères segmentés sont reconnus à l'aide du moteur **OCR Tesseract**, intégré via Anaconda. Ce moteur convertit chaque image de caractère en texte brut. Les résultats sont ensuite automatiquement enregistrés dans un fichier Excel, contenant également les informations temporelles (date et heure de détection), ce qui permet un archivage structuré des plaques reconnues.

**Résultats**

L'approche a été testée sur cinq jeux de données nationaux (Corée, Taïwan, Grèce, USA, Croatie) ainsi que sur un jeu de démonstration multinational regroupant 17 pays.

• Le jeu KarPlate (Corée), composé de plus de 4 000 images annotées automatiquement, a servi de référence.

• Les taux de précision moyens pour la reconnaissance complète des plaques sont les suivants :

- Corée : 99,0 %
- Taïwan : 97,8 %
- Grèce : 98,0 %
- USA : 98,6 %
- Croatie : 97,7 %

• Le temps moyen de reconnaissance complète est de 42 ms par image, permettant u

**Analyse critique**

**Avantages :**

•Méthode end-to-end rapide et efficace, adaptée aux applications embarquées.

• Indépendance au format national, grâce à l'ordonnancement géométrique.

• Traitement unifié de la détection et de la reconnaissance via YOLOv3.

• Ordonnancement spatial des caractères bien pensé, compatible avec plaques bilignes ou exotiques.

• Potentiel pour la reconnaissance multi-plaques sur une même image.

**Limites :**

• Dépendance à une bonne détection initiale par Tiny YOLOv3 : si la plaque est mal détectée, tout le reste échoue.

• Faible généralisation aux alphabets non latins, sauf si le réseau est réentraîné sur ces symboles.

• Risque d'erreurs avec textes parasites (nom du constructeur, ville, drapeaux) s'ils ne sont pas filtrés.

### 2.8.4 Méthode basée sur YOLOv5 amélioré et reconnaissance séquentielle par GRU-CTC

Hengliang Shi et Dongnan Zhao en 2023[29] ont développé une méthode innovante pour la reconnaissance automatique des plaques d'immatriculation, adaptée aux scènes naturelles complexes, incluant les conditions difficiles comme le flou, l'inclinaison ou les occultations partielles. Leur approche repose sur une architecture end-to-end combinant un YOLOv5 amélioré pour la détection et un réseau GRU avec alignement CTC pour la reconnaissance, sans passer par une étape explicite de segmentation des caractères.

**Détection de la plaque d'immatriculation**

La première étape du système repose sur un réseau de détection YOLOv5, connu pour son efficacité et sa légèreté. Les auteurs l'ont modifié pour améliorer la détection dans des scènes complexes, en intégrant plusieurs éléments :

• Mécanisme d'attention L-SE (Local-Spatial Excitation), inséré dans la phase de downsampling du module PANet (Path Aggregation Network). Contrairement au SE standard (Squeeze-and-Excitation), cette version améliore la représentation des caractéristiques spatiales verticales et horizontales, permettant une meilleure localisation des plaques dans l'image, même lorsqu'elles sont inclinées ou partiellement visibles.

• Réduction du nombre de classes prédictives : le modèle ne prédit qu'une seule classe (plaque), ce qui réduit le nombre de paramètres de la couche de sortie (de 255 à 18), limitant les erreurs de classification et accélérant l'inférence.

Grâce à ces optimisations, le détecteur est capable de repérer efficacement des plaques petites, floues ou obliques, avec une précision adaptée à un usage temps réel.

**Segmentation des caractères**

Cette approche n'inclut pas de segmentation explicite des caractères. Contrairement aux pipelines classiques qui isolent chaque caractère avant de les reconnaître, le système exploite une reconnaissance séquentielle directe, rendant inutile cette étape.

**Reconnaissance des caractères**

La reconnaissance repose sur un modèle séquentiel combinant :

• GRU (Gated Recurrent Unit) : une alternative plus légère que LSTM, capable de capturer efficacement la structure temporelle et l'ordre des caractères sans surcharge computationnelle.

• CTC (Connectionist Temporal Classification) : cette technique permet d'aligner automatiquement les séquences d'entrée (carte de caractéristiques visuelles) avec les séquences de sortie (texte de la plaque), même en l'absence de segmentation manuelle. CTC insère automatiquement les décalages nécessaires entre les caractères reconnus.

Ce couplage GRU + CTC permet une reconnaissance fluide et robuste, adaptée à des plaques de formats variés, dans des conditions environnementales non contrôlées.

**Résultats**

Les auteurs ont évalué leur système sur un jeu de données enrichi du CCPD (Chinese City Parking Dataset), comprenant 12 500 images capturées dans des situations réelles (pluie, faible luminosité, angles variés).

• Taux de reconnaissance moyen : 98,98 %, supérieur aux performances obtenues avec LSTM + CTC (98,1 %).

• Robustesse confirmée dans des conditions météorologiques difficiles.

• FPS compatible temps réel, permettant un déploiement dans des systèmes urbains ou industriels.

**Analyse critique**

**Avantages :**

• Pipeline end-to-end sans segmentation manuelle, simplifiant le traitement et réduisant les erreurs d'étape intermédiaire.

• Excellente précision et robustesse, notamment en cas de flou, obliquité ou mauvaise luminosité.

• Modèle GRU + CTC rapide à entraîner et performant, avec une convergence plus rapide que LSTM.

• Mécanisme d'attention L-SE efficace pour améliorer la localisation spatiale de la plaque.

**Limites :**

• Bien que GRU soit plus léger que LSTM, il reste relativement coûteux en ressources pour un déploiement sur des systèmes embarqués très contraints (comme Raspberry Pi).

• Le modèle est entraîné uniquement sur des plaques chinoises ; une adaptation est nécessaire pour des plaques utilisant d'autres alphabets (arabes, latins étendus, cyrilliques, etc.).

• L'approche ne gère pas la correction contextuelle des caractères (pas de post-traitement heuristique).

### 2.8.5 Méthode hybride basée sur le traitement d'image et les réseaux de neurones

Swanand Joshi et ses collaborateurs en 2024 [30] ont proposé une méthode complète et modulaire pour la reconnaissance automatique des plaques d'immatriculation, combinant des techniques classiques de traitement d'image avec des modèles d'apprentissage automatique. L'objectif est de concevoir une architecture flexible, facilement adaptable à différents types de caméras (fixes, mobiles, infrarouges) et de conditions environnementales variées. Leur pipeline se décompose en cinq grandes étapes : acquisition, prétraitement, détection, segmentation, et reconnaissance des caractères.

**Détection de la plaque d'immatriculation**

La détection commence par l'acquisition d'images à partir de divers capteurs visuels. Un prétraitement est ensuite appliqué afin de normaliser l'image et en améliorer la qualité :

• Amélioration de l'image : égalisation d'histogramme, ajustement du contraste, et filtrage du bruit sont utilisés pour améliorer la lisibilité de la plaque.

• Normalisation : redimensionnement, correction d'orientation, et homogénéisation de la luminosité assurent une standardisation des images avant traitement.

• Détection du véhicule : des détecteurs d'objets modernes comme YOLO, Faster R-CNN, ou SSD sont utilisés pour localiser la voiture dans l'image globale. Une fois le véhicule identifié, une région d'intérêt (ROI) est extraite, contenant potentiellement la plaque.

**Segmentation des caractères**

L'extraction de la plaque et des caractères repose principalement sur des méthodes de traitement d'image classiques :

• Segmentation par couleur : distinction de la plaque par rapport au reste du véhicule à l'aide d'un filtre de couleur (typiquement bleu, jaune ou blanc selon le pays).

• Détection des contours : la méthode de Canny est combinée à l'utilisation de caractéristiques Haar-like pour délimiter précisément les bords de la plaque.

• Boîte englobante : création d'une bounding box pour isoler la plaque.

• Analyse des composants connectés : extraction des caractères individuels en repérant les régions connectées dans l'image binaire de la plaque.

• Morphologie mathématique : des opérations comme l'érosion et la dilatation sont appliquées pour corriger les caractères incomplets ou fusionnés.

Cette étape de segmentation est essentielle pour garantir la précision de la reconnaissance en aval.

**Reconnaissance des caractères**

La reconnaissance des caractères segmentés repose sur plusieurs techniques combinées :

• OCR (Reconnaissance Optique de Caractères), avec des méthodes variées :

- Correspondance de motifs (pattern matching)

- Réseaux de neurones convolutifs (CNN) pour l'extraction de caractéristiques et la classification

- Machines à vecteurs de support (SVM), en tant que classifieurs légers

• Les auteurs mentionnent également la possibilité d'utiliser des réseaux récurrents (RNN, LSTM) pour traiter des séquences de caractères, notamment en cas de distorsion ou de lecture partielle.

Le système peut être entraîné sur des jeux de données spécifiques afin de s'adapter aux formats locaux de plaques d'immatriculation.

**Résultats expérimentaux**

L'approche a été testée sur des images issues de scènes variées, incluant des plaques partiellement masquées, inclinées ou peu éclairées. Les performances sont les suivantes :

• Précision globale : supérieure à 95 %, même dans des conditions environnementales complexes.

• Robustesse validée sur des images réelles avec bruits visuels.

• Applications cibles : surveillance urbaine, contrôle d'accès automatisé, péage, gestion de stationnement intelligent.

Le système est compatible avec les architectures de villes intelligentes, grâce à sa modularité et à ses performances stables.

**Analyse critique**

**Avantages :**

• Approche modulaire et explicite, avec des étapes distinctes pour chaque fonction, facilitant le débogage, l'adaptation et l'amélioration.

• Bonne intégration de méthodes classiques et modernes, offrant un compromis entre robustesse, coût de calcul, et précision.

• Excellente précision en conditions difficiles, démontrant la viabilité en environnement urbain réel.

• Adaptabilité à diverses configurations matérielles et besoins applicatifs.

**Limites :**

• Sensibilité à la qualité des images : performances réduites en cas de flou important, mauvaise résolution ou plaques fortement endommagées.

• Nécessite une adaptation aux formats régionaux : ne prend pas en charge les alphabets non latins (arabe, cyrillique, etc.) sans réentraînement.

• Pipeline non end-to-end : l'architecture nécessite une coordination manuelle entre modules, ce qui augmente le temps de traitement global et la complexité d'intégration dans des systèmes compacts.

## 2.9 Travaux de recherche et applications en Algérie pour la détection et la reconnaissance automatique des plaques d'immatriculation

**1. Université Yahia Fares - Médéa (2020)**[91]

• Titre : Reconnaissance automatique des plaques d'immatriculation par réseau CNN.

• Méthode : Détection par contours en niveaux de gris, puis classification CNN.

Plateforme : Raspberry Pi 3, OpenCV, Keras/TensorFlow.

**2. Université Besïa (2024)**[92]

• Titre : Système intelligent de reconnaissance des plaques algériennes par apprentissage profond.

• Méthode : Détection avec YOLO, correction de perspective, segmentation, CNN pour OCR.

**3. Université de M'sila (juillet 2024)**[93]

• Titre : Automated Vehicle Detection and Real-time Number Plate Recognition.

• Méthode : Détection YOLOv5 vs Faster-RCNN, optimisation via NPU embarqué.

**4. Université de Guelma (2023)**[94]

• Titre : Algerian license plate detection and recognition using deep learning.

• Méthode : YOLOv5 pour la détection + CNN pour OCR

Performance : 87% de précision en détection, 93% d'exactitude en reconnaissance.

**Résultat :**

Ces travaux ont réussi à détecter les plaques d'immatriculation, mais le taux de reconnaissance des caractères n'est pas toujours précisé ou reste limité dans certains cas.

## 2.10 Conclusion

L'étude comparative des différentes approches de reconnaissance automatique des plaques d'immatriculation (ANPR) met en évidence la richesse des solutions développées ces dernières années, chacune répondant à des contraintes spécifiques du contexte d'application. Les méthodes basées sur YOLO, dans ses différentes variantes (YOLO modifié, YOLOv3-SPP, YOLOv5), dominent le paysage grâce à leur rapidité, leur précision et leur capacité à s'adapter aux environnements complexes.

Les approches end-to-end, intégrant la détection, la segmentation et la reconnaissance dans un seul pipeline optimisé, montrent une efficacité remarquable en termes de temps de traitement, tout en maintenant de bons taux de reconnaissance, même dans des conditions dégradées (flou, obliquité, faible luminosité). C'est notamment le cas des systèmes combinant YOLO avec GRU-CTC, qui évitent la segmentation explicite, ou des modèles renforcés par des heuristiques intelligents pour corriger les erreurs typiques de classification.

En parallèle, les méthodes hybrides qui intègrent des techniques classiques de traitement d'image avec des modèles de machine learning conservent leur pertinence, notamment dans des contextes à faible ressource ou nécessitant une forte modularité. Toutefois, leur complexité d'intégration et leur dépendance à la qualité d'image en limitent parfois la robustesse.

Il apparaît également que la généralisation à différents formats de plaques (alphabets, structure, disposition) reste un défi non négligeable. Les systèmes les plus performants sont encore fortement dépendants des données d'entraînement spécifiques à une région ou un pays. Des solutions comme les architectures indépendantes du format ou les méthodes de reconstruction géométrique des plaques constituent donc des avancées prometteuses.

Ainsi, le choix d'une approche ANPR dépend fortement des objectifs visés (précision, coût, temps réel, portabilité), et les travaux analysés ouvrent la voie à des combinaisons adaptatives, tirant parti des avantages respectifs de chaque paradigme.

CHAPITRE

# 3

# MÉTHODOLOGIE

## 3.1 Introduction

Dans ce chapitre, nous décrivons la démarche méthodologique adoptée pour la mise en œuvre de notre système de reconnaissance automatique des plaques d'immatriculation (ANPR). L'objectif principal est de concevoir un pipeline efficace capable d'assurer la détection rapide des plaques et la reconnaissance précise des caractères, en s'appuyant sur des modèles de deep learning de la famille YOLO. Après avoir introduit les fondements théoriques de YOLO et justifié le choix des versions YOLOv5n et YOLOv8s pour nos tâches respectives, nous détaillons la constitution et la préparation des jeux de données, les phases d'entraînement des modèles, ainsi que les techniques d'optimisation et de déploiement sur différentes plateformes (PC, Raspberry Pi, Android). Enfin, nous présentons les différentes interfaces applicatives développées, démontrant la portabilité et l'efficacité de notre solution dans des contextes variés.

## 3.2 L'Architecture YOLO : Principes, Évolution et Innovations

### 3.2.1 Introduction à YOLO

#### Contexte et émergence de YOLO

YOLO (You Only Look Once) représente une avancée majeure dans le domaine de la détection d'objets en vision par ordinateur. Contrairement aux approches traditionnelles qui utilisaient des

méthodes en plusieurs étapes, YOLO a introduit un paradigme révolutionnaire de détection en une seule passe. Cette innovation a permis d'atteindre des performances en temps réel tout en maintenant une précision compétitive.

Le premier modèle YOLO a été présenté en 2016 par Joseph Redmon, Santosh Divvala, Ross Girshick et Ali Farhadi dans leur article "You Only Look Once : Unified, Real-Time Object Detection". Cette publication a marqué un tournant dans le domaine de la détection d'objets en proposant une approche unifiée qui traite la détection comme un problème de régression unique, de l'image d'entrée aux coordonnées des boîtes englobantes et aux probabilités de classes [63].

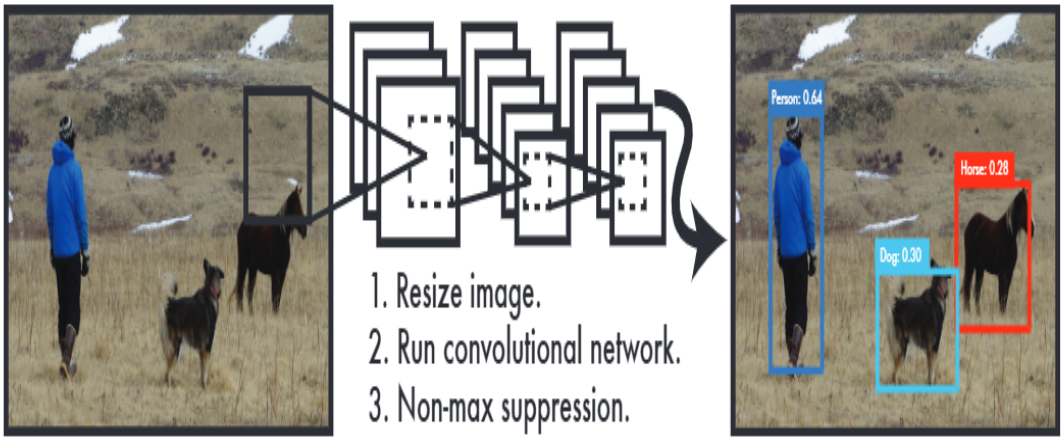


FIGURE 3.1 – Détection YOLO [78]

### Principes fondamentaux de YOLO

Le principe fondamental de YOLO repose sur sa capacité à prédire simultanément les boîtes englobantes et les classes d'objets en une seule passe à travers le réseau neuronal[63]. Cette approche présente plusieurs avantages clés :

1. **Vitesse d'inférence** : En traitant l'image en une seule passe, YOLO atteint des vitesses d'inférence significativement plus élevées que les méthodes à plusieurs étapes [63].
2. **Raisonnement global** : Contrairement aux méthodes basées sur des fenêtres glissantes ou des propositions de régions, YOLO "voit" l'image entière pendant l'entraînement et l'inférence, ce qui lui permet de capturer le contexte global et de réduire les faux positifs [63].
3. **Apprentissage des caractéristiques généralisables** : En considérant l'image complète, YOLO apprend des représentations qui généralisent mieux aux nouvelles images et domaines [63].

4. **Architecture unifiée** : YOLO unifie tous les composants de la détection d'objets en un seul réseau de neurones, simplifiant ainsi l'optimisation de bout en bout [63].

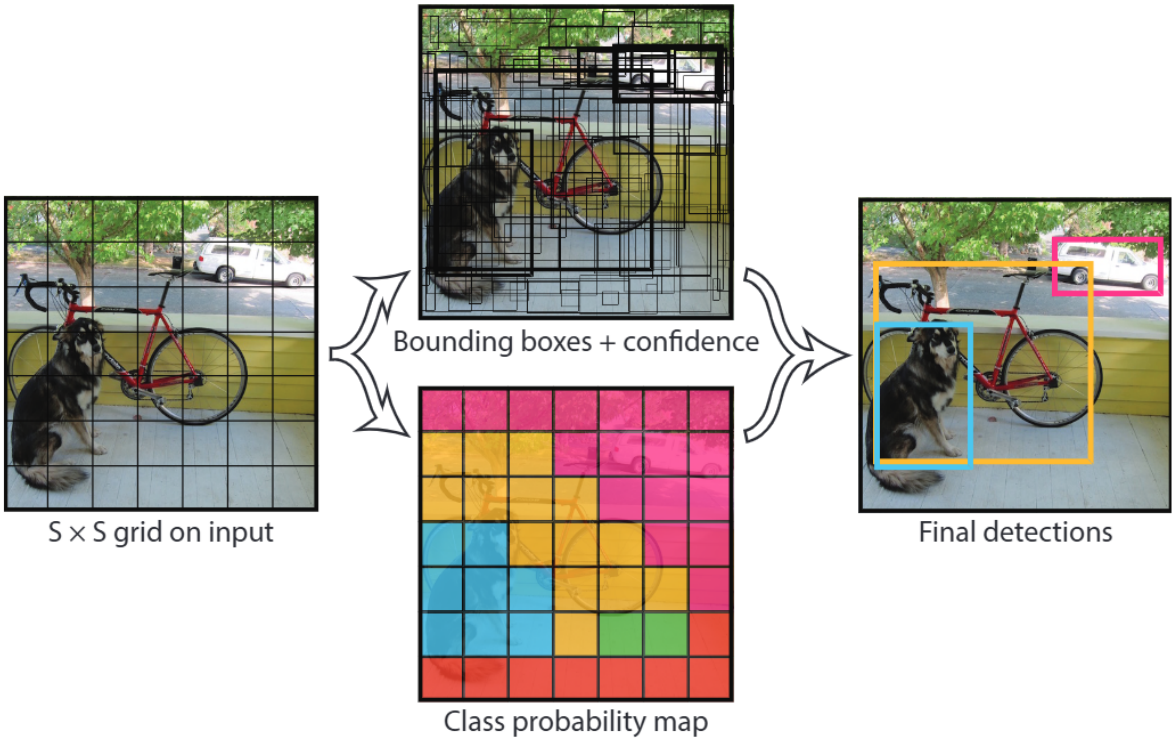


FIGURE 3.2 – Division de l'image en grille et prédiction simultanée des boîtes et classes[79]

### 3.2.2 Architecture de base de YOLO

#### Structure générale

L'architecture originale de YOLOv s'inspire du modèle GoogLeNet et se compose de 24 couches convolutionnelles suivies de 2 couches entièrement connectées [63]. La structure peut être décomposée comme suit :

1. **Backbone (Épine dorsale)** : Série de couches convolutionnelles qui extraient les caractéristiques de l'image d'entrée [63].
2. **Neck (Cou)** : Couches intermédiaires qui agrègent les caractéristiques [63].
3. **Head (Tête)** : Couches finales qui produisent les prédictions de détection [63].

#### Mécanisme de prédiction

Le mécanisme de prédiction de YOLOv1 fonctionne comme suit [63] :

1. L'image d'entrée est redimensionnée à 448×448 pixels et divisée en une grille de S×S cellules (typiquement 7×7) [63].

2. Chaque cellule de la grille est responsable de prédire B boîtes englobantes (typiquement 2) si le centre d'un objet tombe dans cette cellule [63].
3. Pour chaque boîte englobante, le modèle prédit 5 valeurs : les coordonnées du centre (x, y), la largeur (w), la hauteur (h) et un score de confiance [63].
4. Chaque cellule prédit également C probabilités conditionnelles de classes, représentant la probabilité que l'objet détecté appartienne à une classe particulière [63].

La sortie finale du réseau est donc un tenseur de dimension S×S×(B×5+C). Pour YOLOv1 avec S=7, B=2 et C=20 (pour le jeu de données PASCAL VOC), cela donne une sortie de dimension 7×7×30 [63].

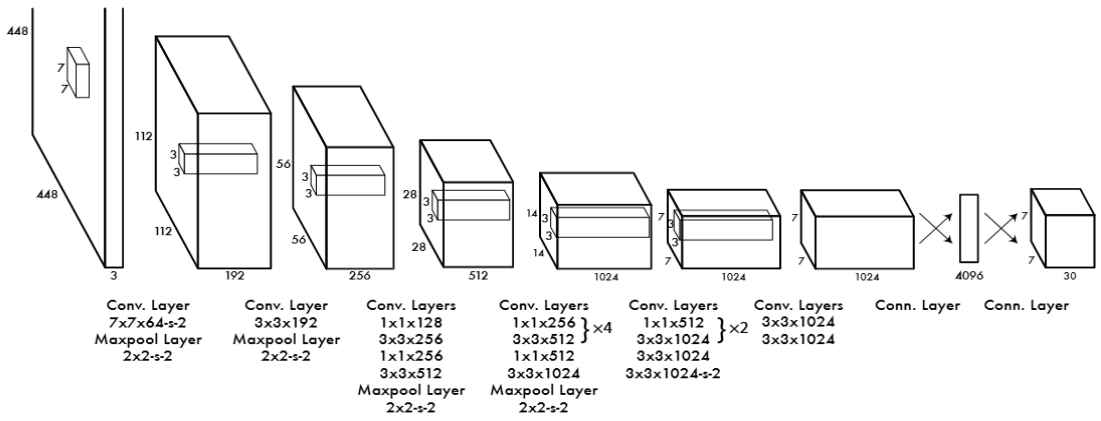


FIGURE 3.3 – Processus de prédiction de YOLOv1[80]

**Fonction de perte**

La fonction de perte de YOLOv1 est une somme pondérée de plusieurs termes [63] :

1. **Erreur de localisation** : Erreur quadratique moyenne sur les coordonnées du centre et les racines carrées des dimensions des boîtes [63].
2. **Erreur de confiance** : Erreur quadratique moyenne sur les scores de confiance, avec des poids différents pour les boîtes avec et sans objets [63].
3. **Erreur de classification** : Erreur quadratique moyenne sur les probabilités conditionnelles de classes [63].

Cette fonction de perte multi-tâche permet d'optimiser simultanément la localisation des objets et leur classification [63].

### 3.2.3 Composants architecturaux clés de YOLO

#### Backbone (Épine dorsale)

Le backbone est responsable de l'extraction des caractéristiques de l'image d'entrée. Son évolution à travers les versions de YOLO illustre la recherche constante d'un meilleur équilibre entre capacité d'extraction de caractéristiques et efficacité computationnelle :

1. **YOLOv1** : Architecture inspirée de GoogLeNet avec 24 couches convolutionnelles[63].
2. **YOLOv2** : Darknet-19, plus léger et efficace[64].
3. **YOLOv3** : Darknet-53 avec connexions résiduelles[65].
4. **YOLOv4** : CSPDarknet53 intégrant des Cross-Stage Partial connections[66][74].
5. **YOLOv5-v8** : Variantes de CSP et C2f avec différentes optimisations[67][71].

#### Neck (Cou)

Le neck est une composante intermédiaire qui agrège les caractéristiques extraites par le backbone à différentes échelles :

1. **YOLOv3** : Introduction du Feature Pyramid Network (FPN) pour la détection multi-échelles[65][72].
2. **YOLOv4** : Path Aggregation Network (PAN) pour une meilleure fusion des caractéristiques[66][73].
3. **YOLOv5** : Combinaison de FPN et PAN avec des blocs CSP[67].
4. **YOLOv8** : Utilisation de modules C2f au lieu du traditionnel FPN[71].

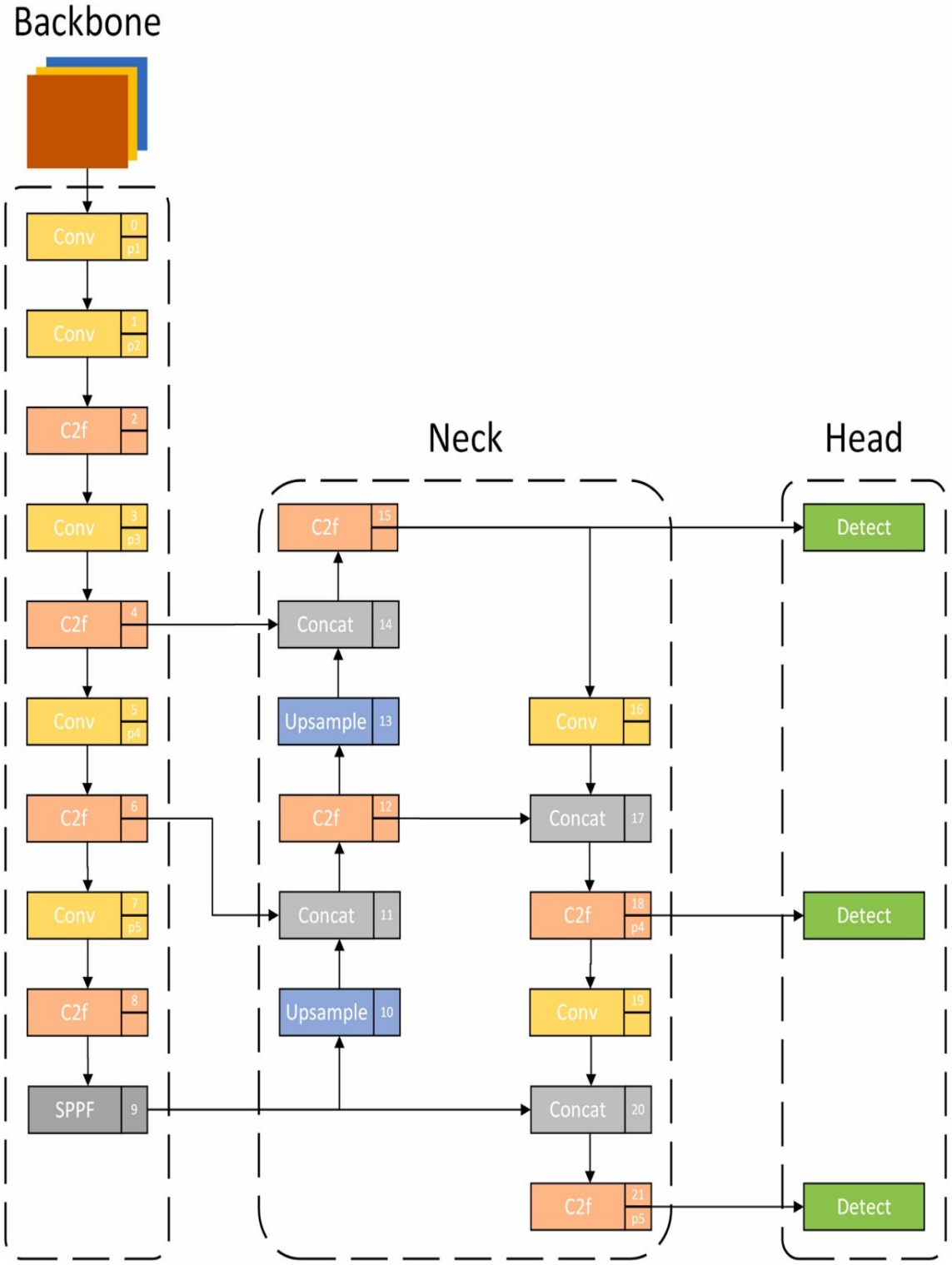


FIGURE 3.4 – Le Neck dans YOLO[85]

**Head (Tête)**

La tête du réseau est responsable de la prédiction finale des boîtes englobantes et des classes :

1. **YOLOv1-v3** : Tête couplée prédisant simultanément les boîtes et les classes[63][64][65].
2. **YOLOv4-v5** : Améliorations progressives de la tête avec une meilleure gestion des échelles[66][67].

3. **YOLOv7-v8** : Tête découplée séparant les tâches de classification et de régression [69][71].

### 3.2.4 Applications et cas d'usage

#### Détection en temps réel

YOLO excelle dans les applications nécessitant une détection en temps réel :

1. **Surveillance vidéo** : Détection de personnes, véhicules et objets dans les flux vidéo[63].
2. **Systèmes d'aide à la conduite** : Détection des véhicules, piétons et obstacles[66].
3. **Robotique** : Perception de l'environnement pour la navigation et l'interaction[71].

#### Détection sur appareils embarqués

Les versions légères de YOLO sont particulièrement adaptées aux appareils à ressources limitées :

1. **Smartphones et tablettes** : Applications mobiles de réalité augmentée et d'analyse d'images[67][71].
2. **Drones** : Détection d'objets pour la navigation et la cartographie [66][71].
3. **Systèmes embarqués** : Intégration dans des appareils IoT et des systèmes de surveillance[67][71].

#### Cas d'usage spécifiques

YOLO a été adapté avec succès à de nombreux domaines spécifiques[71] :

1. **Reconnaissance de plaques d'immatriculation** : Détection et lecture des plaques pour les systèmes de contrôle d'accès.
2. **Analyse médicale** : Détection de structures anatomiques et d'anomalies dans les images médicales.
3. **Agriculture de précision** : Détection de cultures, maladies et insectes nuisibles.
4. **Retail** : Analyse des rayons et du comportement des clients. Références bibliographiques

## 3.3 Les plaques d'immatriculation algériennes

Les plaques d'immatriculation en Algérie sont soumises à une normalisation précise qui facilite leur identification visuelle et automatique. Cette standardisation est essentielle pour des applications comme la reconnaissance automatique des plaques (ANPR), en assurant la cohérence dans le format, les dimensions et la codification des informations.

### 3.3.1 Description du format standardisé

Les plaques algériennes contiennent une séquence de chiffres organisée selon un format logique permettant d'encoder plusieurs informations utiles numéro d'enregistrement, type de véhicule, année de fabrication et lieu d'immatriculation[87]. Le format standard est généralement de type :

NNNNN – XYZ – WW

- NNNNN : Numéro séquentiel attribué au véhicule.
- X : Code indiquant la catégorie du véhicule.
- Y et Z : Deux chiffres représentant l'année de mise en circulation (ex. 22 pour 2022).
- WW : Code de la wilaya (province) d'immatriculation. Ce format est conçu pour être lisible et exploitable facilement par les systèmes de vision par ordinateur.

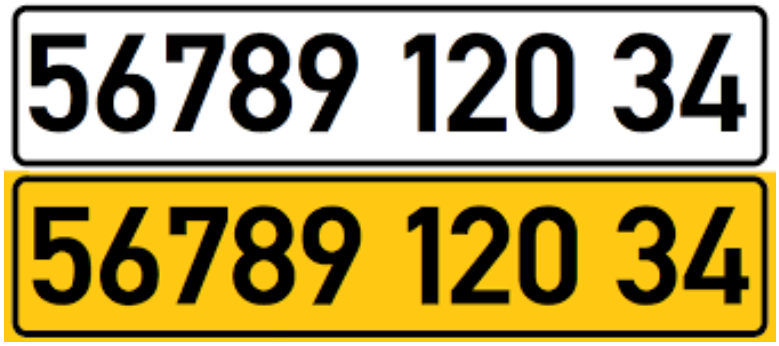


FIGURE 3.5 – plaque d'immatriclulation algérienne

### 3.3.2 Types et catégories de véhicules

Le premier chiffre du bloc central (XYZ) définit la classe du véhicule, selon une codification nationale. Voici les principales catégories :

| Code | Type de véhicule |
|---|---|
| 1 | Véhicule particulier |
| 2 | Camion |
| 3 | Camionnette |
| 4 | Bus |
| 5 | Tracteur routier |
| 6 | Tracteur agricole |
| 7 | Véhicule spécialisé |
| 8 | Remorque |
| 9 | Motocyclette |

TABLE 3.1 – Types des véhicules[87]

### 3.3.3 Défis liés à la détection et reconnaissance dans des conditions réelles

Dans un contexte réel, la détection et la reconnaissance des plaques d'immatriculation présentent plusieurs défis majeurs susceptibles de compromettre la précision du système. Tout d'abord, les variations d'éclairage constituent une contrainte importante une plaque fortement éclairée ou, au contraire, plongée dans l'ombre peut entraîner une perte de contraste, rendant difficile l'extraction des caractères. De plus, les angles de vue obliques, typiques des images capturées en circulation ou dans un stationnement, peuvent provoquer des distorsions géométriques, compliquant la localisation précise de la plaque et la lisibilité des caractères. À cela s'ajoutent les salissures, poussières, boues ou rayures, fréquentes sur les véhicules, qui peuvent partiellement masquer les informations. Les occlusions partielles, provoquées par d'autres objets (barres de toit, véhicules proches, doigts, etc.), représentent également un obstacle courant. Ces conditions dégradées exigent que le système soit capable de généraliser à partir de données imparfaites, ce qui nécessite des modèles robustes, entraînés sur des jeux de données variés et réalistes.

## 3.4 Constitution et préparation des jeux de données

### 3.4.1 Jeu de données pour la détection de plaques

Pour l'entraînement du modèle YOLOv5n destiné à la détection des plaques d'immatriculation, deux sources principales de données ont été exploitées :

• **Données fournies par un enseignant d'Oran :** Un jeu de 3 400 images annotées a été mis à disposition par Mr.BENSOUILAH Mouad un enseignant universitaire basé à Oran. Ces images présentent des véhicules dans différents contextes et angles de vue, ce qui constitue une base solide et diversifiée pour l'entraînement. Cependant, les annotations fournies n'étaient pas au format requis par YOLO. Afin de résoudre ce problème, Nous avons développé un script Python personnalisé pour convertir les annotations existantes au format YOLO, qui consiste en des fichiers texte contenant les coordonnées normalisées des boîtes englobantes selon la structure suivante [34] :

```
<classe> <x_center> <y_center> <width> <height>
```

• **Images capturées manuellement :** En complément, 300 images supplémentaires ont été prises manuellement à l'aide de téléphones portables, dans des environnements réels et variés, afin de renforcer l'adaptabilité du modèle aux conditions locales (types de plaques, angles, luminosité). Ces images ont été annotées manuellement à l'aide de l'outil ImageLabel, en suivant le format YOLO.

Le dataset a ensuite été combiné, vérifié et nettoyé pour éliminer les doublons et corriger d'éventuelles erreurs d'annotation, pour un total de 3695 images.

### 3.4.2 Jeu de données pour la détection et reconnaissance des chiffres

#### Sources des données

Le jeu de données utilisé pour l'entraînement du modèle YOLOv8s pour la détection et la reconnaissance de chiffres est constitué de deux sources principales :

• **Dataset de plaques d'immatriculation** : Un ensemble de 1000 images de plaques d'immatriculation, fournies par Mr.BENSOUILAH Mouad [34].Ces images ont été capturées sous différents angles et conditions (bruit, pluie, variations d'éclairage, etc.) pour refléter des scénarios réels, la figure suivant donnée des exemples sur ces images.

FIGURE 3.6 – Exemple de dataset pour la détection/reconnaissance de chiffres LPAD

• **Dataset Kaggle** : Un ensemble de 34 466 images extraites du dataset License Plate Digits Classification Dataset disponible sur le dataset [35]. Ce dataset est organisé en 10 dossiers, nommés de 0 à 9,chaque dossier contient des images représentant un chiffre unique occupant toute la surface de l'image, la figure suivant donnée des exemples sur ces images.

FIGURE 3.7 – Exemple de dataset pour la détection/reconnaissance de chiffres Kaggle

**Volume du dataset**

Le volume total du jeu de données est le suivant :

• **Plaques d'immatriculation** : Les 1000 images de plaques contiennent entre 10 et 11 chiffres par plaque, ce qui équivaut à environ 10 000 chiffres annotés.

• **Images Kaggle** : 34 466 images, chacune correspondant à un chiffre unique (0 à 9).

Ainsi, le dataset combiné comprend un total de 34 466 images de chiffres 1000 images de plaque matricule annoter , formant un ensemble 44 466 images robuste pour l'entraînement et l'évaluation du modèle.

**Stratégie d'annotation**

La stratégie d'annotation diffère selon la source des données :

• **Plaques d'immatriculation** : Les 1000 images ont été organisées en 10 dossiers pour optimiser le temps de travail et la gestion des annotations. Chaque image a été annotée manuellement à l'aide de l'outil LabelImg, qui génère des fichiers d'annotation au format YOLO (contenant l'identifiant de classe et les coordonnées des boîtes englobantes). Cependant, chaque dossier a généré un fichier classes.txt avec un ordre de classes différent, ce qui a nécessité une standardisation. Nous avons développé un

script Python (convertClasses.py) afin d'harmoniser les annotations en réorganisant les indices de classes selon un ordre standard (0 à 9). Le script lit les fichiers d'annotation, remappe les identifiants de classes et génère un nouveau fichier classes.txt cohérent.

• **Images Kaggle** : Les images de ce dataset représentent chacune un chiffre occupant la totalité de l'image, rendant l'annotation manuelle avec LabelImg inutile. Nous avons créé un script Python (creeAnnotation.py) pour générer automatiquement les annotations au format YOLO. Pour chaque image, le script attribue l'identifiant de classe correspondant au nom du dossier (par exemple, classe 9 pour le dossier 9) et définit une boîte englobante couvrant toute l'image (coordonnées normalisées : centre à (0.5, 0.5), largeur et hauteur à 1.0). Les annotations sont sauvegardées dans un dossier dédié.

## 3.5 Entraînement des modèles

Cette section détaille le processus d'entraînement des deux modèles YOLO qui constituent le cœur de notre système de détection de plaques d'immatriculation, YOLOv5n pour la localisation des plaques et YOLOv8s pour la détection et l'identification des chiffres.

### 3.5.1 Métriques d'Évaluation pour les Modèles

#### Concepts Fondamentaux

L'évaluation d'un modèle de détection d'objets repose sur la comparaison entre les prédictions du modèle et la vérité terrain (les annotations manuelles). Plusieurs concepts sont essentiels pour comprendre les métriques :

- **Vrai Positif (True Positive - TP)** : Détection correcte d'un objet réellement présent, avec une superposition suffisante (définie par l'IoU) entre la boîte prédite et la boîte vérité[37].
- **Faux Positif (False Positive - FP)** : Détection incorrecte (objet détecté là où il n'y en a pas, ou mauvaise classification)[37].
- **Faux Négatif (False Negative - FN)** : Objet réel présent mais non détecté par le modèle[37].
- **Intersection over Union (IoU)** : L'Intersection over Union (IoU) est une métrique fréquemment utilisée en vision par ordinateur et en détection d'objets. Elle permet de mesurer le degré de chevauchement entre deux boîtes englobantes : celle correspondant à la vérité terrain (ground truth) et celle prédite par l'algorithme. L'IoU prend une valeur comprise entre 0 et 1, où une valeur proche de 1 signifie que les deux boîtes sont presque parfaitement superposées, indiquant ainsi une détection très précise Un seuil (ex : 0.5) est souvent utilisé pour déterminer si une détection est un TP. voiyer la figure 3.8[38] .

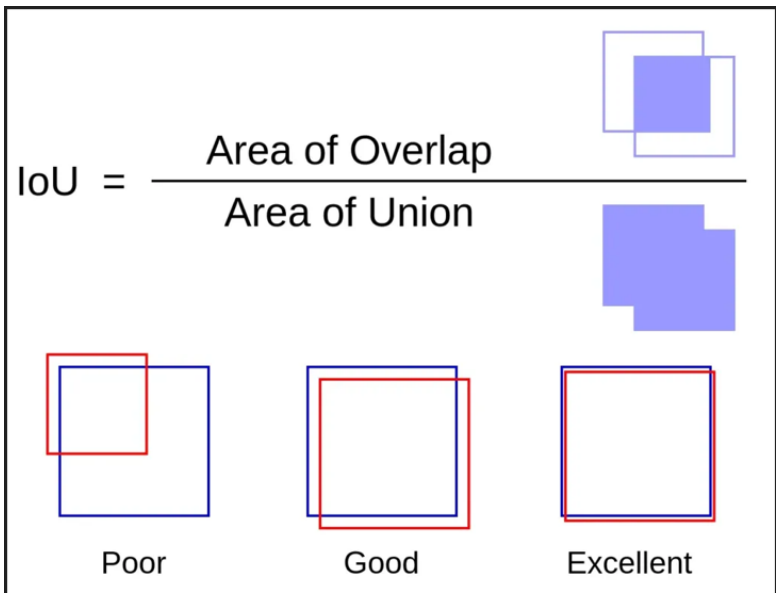


FIGURE 3.8 – Un diagramme représentant l'intersection sur l'union (IoU)[38].

**Métriques d'Évaluation Principales**

1. **Précision (Precision - P)**

   • **Définition** : Quelle est la proportion de détections correctes parmi toutes les détections effectuées par le modèle ? [39]

   • **Formule** : P = TP / (TP + FP)

   • **Interprétation** : Une haute précision indique que le modèle fait peu de fausses détections.

2. **Rappel (Recall - R)**

   • **Définition** : Quelle est la proportion d'objets réels correctement détectés par le modèle parmi tous les objets réellement présents ? [39]

   • **Formule** : R = TP / (TP + FN)

   • **Interprétation** : Un haut rappel indique que le modèle manque peu d'objets présents.

3. **Score F1 (F1-Score)**

   • **Définition** : Moyenne harmonique de la Précision et du Rappel, offrant un équilibre entre les deux.[39]

   • **Formule** : F1 = 2 * (P * R) / (P + R)

   • **Interprétation** : Utile pour obtenir une mesure unique combinant précision et rappel.

4. **Courbe Précision-Rappel (Precision-Recall Curve)**

   • **Définition** : Graphique traçant la Précision en fonction du Rappel pour différents seuils de confiance de détection [40].

   • **Interprétation** : Une courbe proche du coin supérieur droit indique une haute performance globale. L'aire sous cette courbe (AUC-PR) est une mesure synthétique.

5. **Précision Moyenne (Average Precision - AP)**

   • **Définition** : Mesure résumant la courbe Précision-Rappel en une seule valeur pour une classe spécifique. Elle correspond approximativement à l'aire sous la courbe PR[41].

   • **Interprétation** : Plus l'AP est élevée (proche de 1), meilleur est le modèle pour cette classe.

6. **Précision Moyenne (Mean Average Precision - mAP)**

   • **Définition** : Moyenne des AP calculées sur toutes les classes d'objets que le modèle doit détecter[37] [42] .

   • **Interprétation** : C'est la métrique standard la plus courante pour évaluer et comparer globalement les modèles de détection d'objets multi-classes

   • **Variantes courantes** :

   a) **mAP@0.5 (ou mAP50)** : Calculée avec un seuil d'IoU de 0.5 pour considérer une détection comme TP.

   b) **mAP@0.5 :0.95 (ou mAP COCO)** : Calculée en moyennant les mAP obtenues pour des seuils d'IoU allant de 0.5 à 0.95 (par pas de 0.05). C'est une métrique plus stricte, exigeant une localisation plus précise [43] .

**Métriques d'Entraînement (Losses)**

Ces valeurs suivent la progression de l'apprentissage mais ne sont pas des métriques d'évaluation finales :

a) **Box Loss** : Erreur de localisation des boîtes.

b) **Cls Loss** : Erreur de classification.

c) **Obj Loss (YOLOv5)** : Erreur sur la confiance de présence d'objet.

d) **DFL Loss (YOLOv8)** : Distribution Focal Loss, aide à affiner la localisation.

• **Interprétation** : Ces pertes doivent idéalement diminuer sur les ensembles d'entraînement et de validation. Une augmentation de la perte de validation peut indiquer un surapprentissage.

L'évaluation d'un modèle YOLO implique l'analyse de plusieurs métriques complémentaires. La mAP (en particulier mAP@0.5 et mAP@0.5 :0.95) fournit une mesure globale essentielle, tandis que la Précision, le Rappel, le Score F1, et la Matrice de Confusion permettent une analyse plus fine des performances et des types d'erreurs pour chaque classe spécifique.

### 3.5.2 Environnement d'entraînement

L'entraînement des modèles a été réalisé sur la plateforme Kaggle, reconnue pour sa flexibilité et l'accès à des ressources de calcul haut performance. Nous avons exploité un environnement configuré avec les spécifications suivantes :

#### Matériel

Deux accélérateurs GPU NVIDIA Tesla T4, permettant de paralléliser et d'accélérer significativement les phases d'entraînement gourmandes en calcul.

#### Librairies logicielles clés

**PyTorch** : Framework d'apprentissage profond principal utilisé par les librairies YOLO[88].

**Ultralytics YOLO** : Librairie open-source fournissant les implémentations de YOLOv5 et YOLOv8, ainsi que les scripts et outils nécessaires à l'entraînement, la validation et l'export des modèles. L'utilisation de Kaggle et de GPU dédiés a permis de mener des entraînements sur des jeux de données conséquents dans des délais raisonnables, tout en bénéficiant d'un environnement reproductible [89].

### 3.5.3 Entraînement du détecteur de plaques (YOLOv5n)

Le premier modèle de notre pipeline est chargé de détecter la zone de la plaque d'immatriculation dans l'image d'entrée. Pour cette tâche, nous avons privilégié le modèle YOLOv5n en raison de sa légèreté et de sa rapidité, des atouts essentiels pour une exécution en temps réel sur des plateformes aux ressources limitées.

#### Jeu de données

Un total de 3700 images contenant des plaques d'immatriculation algériennes a été collecté et préparé. Ce jeu de données a été divisé en trois sous-ensembles selon une répartition standard : 70% pour l'entraînement (2590 images), 15% pour la validation (555 images) et 15% pour le test (555 images). Cette répartition assure une évaluation fiable de la capacité du modèle à généraliser sur de nouvelles données.

#### Prétraitements et Augmentation des données

Pour améliorer la robustesse et la capacité de généralisation du modèle, plusieurs techniques d'augmentation de données ont été appliquées lors de l'entraînement, définies dans le fichier d'hyperparamètres hyp.yaml. Celles-ci incluent notamment :

- Variations de teinte, saturation et valeur (HSV) :

  hsv_h : 0.015

  hsv_s : 0.7

  hsv_v : 0.4
- Rotations aléatoires légères : degrees : 5.0.
- Translations : translate : 0.1.
- Mise à l'échelle : scale : 0.5.

- Retournement horizontal : fliplr : 0.5.
- Mosaïque (combinant 4 images en une) : mosaic : 1.0.
- Mixup (mélange d'images et de labels) : mixup : 0.2.

Ces augmentations simulent différentes conditions de prise de vue et aident le modèle à mieux gérer les variations d'éclairage, les légers changements d'angle et les différentes tailles de plaques, comme illustré par les exemples de lots d'entraînement dans la figure suivant :


FIGURE 3.9 – les exemples de lots d'entraînement

### Configuration de l'entraînement

L'entraînement a été lancé en utilisant le script train.py de la librairie YOLOv5, avec les paramètres principaux dans les deux figures suivantes :

```
!python train.py \
  --img 512 \
  --batch 128 \
  --epochs 200 \
  --data /kaggle/working/data.yaml \
  --weights /kaggle/input/yolov5n/best.pt \
  --project /kaggle/working/YOLOv5_plate_resul \
  --name yolov5n_plate \
  --hyp /kaggle/working/hyp.yaml \
  --device 0,1 \
  --cache \
  --patience 30 \
  --workers 4
```

FIGURE 3.10 – Le scripte d'entraînement YOLOv5

- `--img 512` : Taille des images d'entrée redimensionnées pour l'entraînement.
- `--batch 128` : Taille du lot d'images traitées simultanément, répartie sur les deux GPU.
- `--epochs 200` : Nombre maximal d'époques (passages complets sur le jeu de données d'entraînement).
- `--data data.yaml` : Fichier décrivant les chemins vers les ensembles d'entraînement, validation et test, ainsi que le nombre de classes (1, "matricule").
- `--weights yolov5n.pt` : Utilisation des poids pré-entraînés du modèle YOLOv5n comme point de départ (transfer learning), ce qui accélère la convergence et améliore les performances.
- `--hyp hyp.yaml` : Fichier contenant les hyperparamètres spécifiques à l'entraînement et à l'augmentation.
- `--device 0,1` : Utilisation des deux GPU disponibles.
- `--cache` : Mise en cache des images en RAM pour accélérer la lecture.
- `--patience 30` : Arrêt anticipé de l'entraînement si aucune amélioration de la métrique principale (mAP@0.5) n'est observée sur l'ensemble de validation pendant 30 époques consécutives.

FIGURE 3.11 – Les paramètres principaux d'entraînement YOLOv5

`--weights yolov5n.pt` : on a utilisé un yolov5n[36] Le modèle a déjà appris à détecter des caractères structurés (lettres/chiffres) sur des surfaces variées (plaques) Il a déjà appris des caractéristiques visuelles utiles (formes, contrastes, alignements), Le dataset contient 8 823 images de plaques d'immatriculation de divers pays, bien que la majorité semble provenir de régions où les plaques sont en alphabet latin, ce qui est similaire à notre tâche.

### Suivi de l'entraînement et Résultats

L'entraînement a été suivi via les métriques standards. L'analyse du fichier yolov5n_results.csv et des courbes dans figure3.12 :

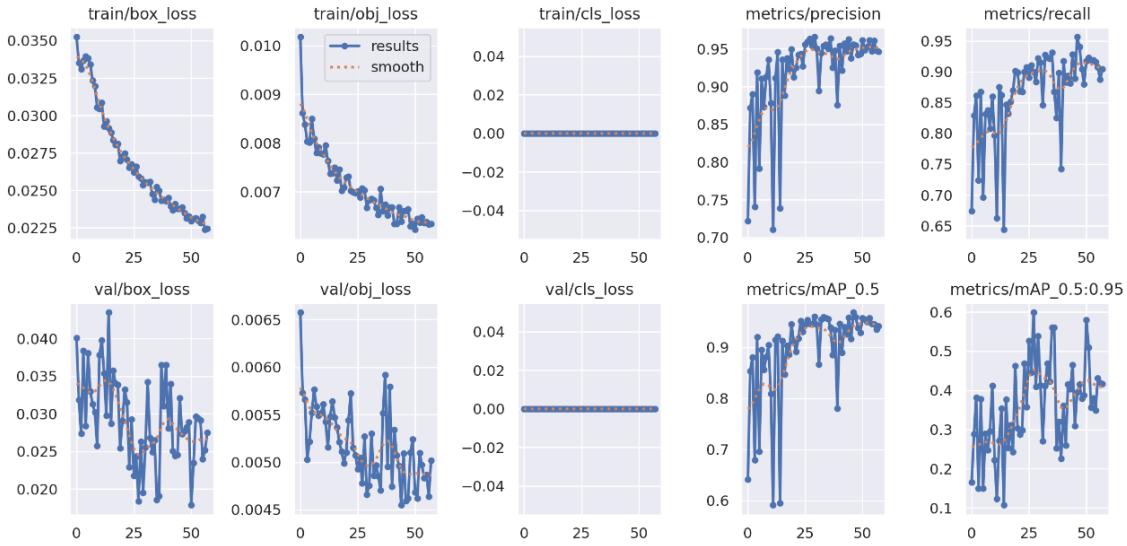


FIGURE 3.12 – Courbes

Montre une convergence rapide des fonctions de perte (box_loss, obj_loss, cls_loss) pour l'ensemble d'entraînement et de validation. Les métriques de performance sur l'ensemble de validation indiquent que le modèle atteint un pic de performance mAP@0.5 de 0.9683 et un pic de mAP@0.5 :0.95 de 0.59893 autour de la 27ème époque, avant de légèrement fluctuer. La précision et le rappel dans les figures 3.13et 3.14 :

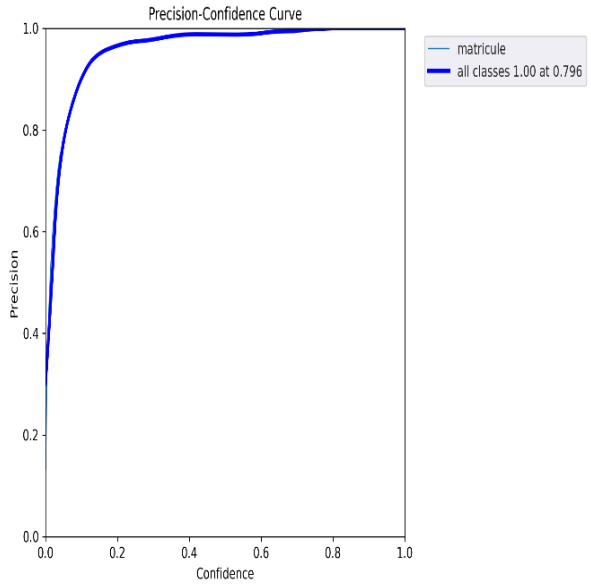


FIGURE 3.13 – La précision

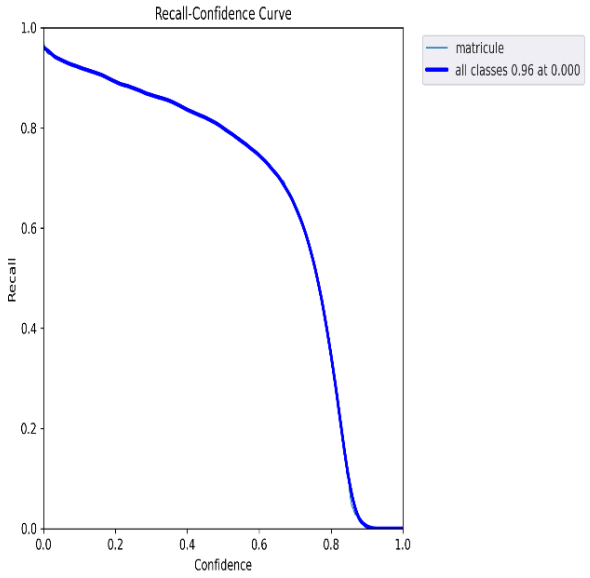


FIGURE 3.14 – Le rappel

montrent également de bonnes performances, avec une courbe Précision-Rappel dans la figure 3.15 :

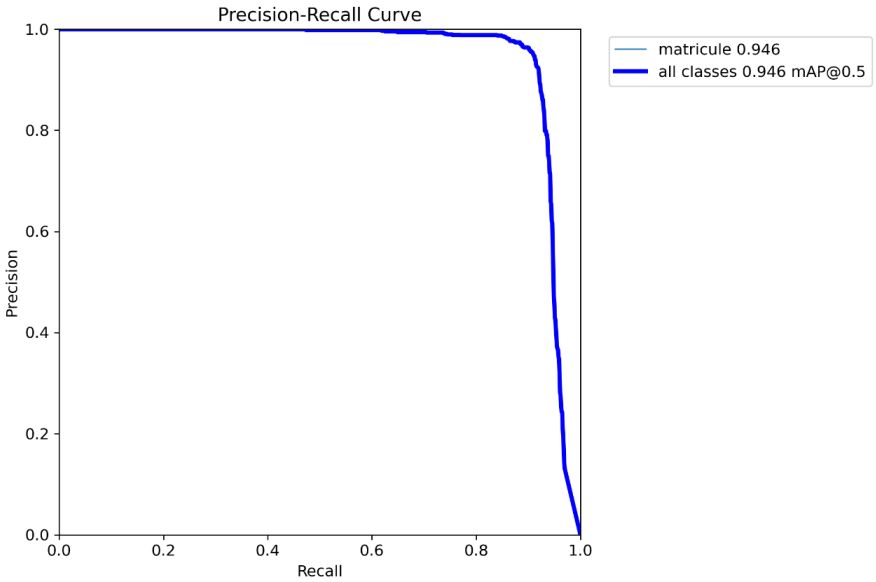


FIGURE 3.15 – la courbe Précision-Rappel

Indiquant une aire sous la courbe élevée, confirmant la capacité du modèle à bien détecter les plaques avec différents niveaux de confiance. La courbe F1 dans la figure 3.16 :

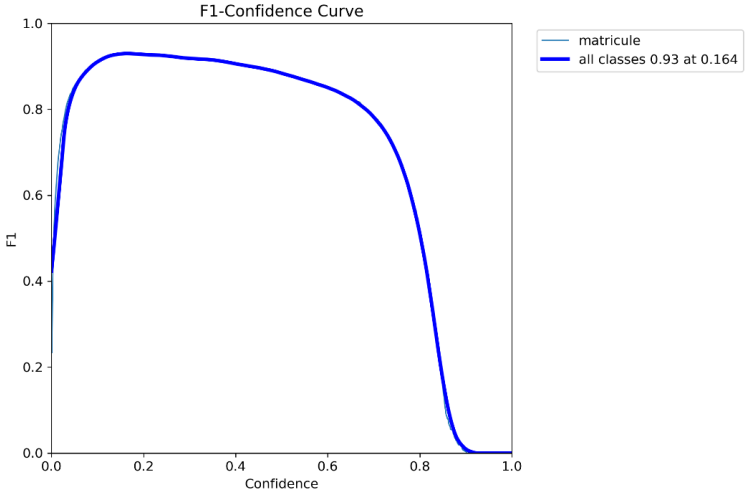


FIGURE 3.16 – La courbe F1

Atteint un score élevé, témoignant d'un bon équilibre entre précision et rappel. L'entraînement s'est arrêté prématurément grâce au mécanisme de patience, sélectionnant le modèle offrant le meilleur compromis performance/généralisation. Les exemples de prédictions sur l'ensemble de validation sont présentés dans la figure 3.17 et comparés à ceux de la figure 3.18 :


FIGURE 3.17 – Visualisation d'un lot des étiquettes du jeu de validation


FIGURE 3.18 – Visualisation d'un lot des prédictions du jeu de validation

De même, une comparaison entre les figures 3.19et 3.20 :

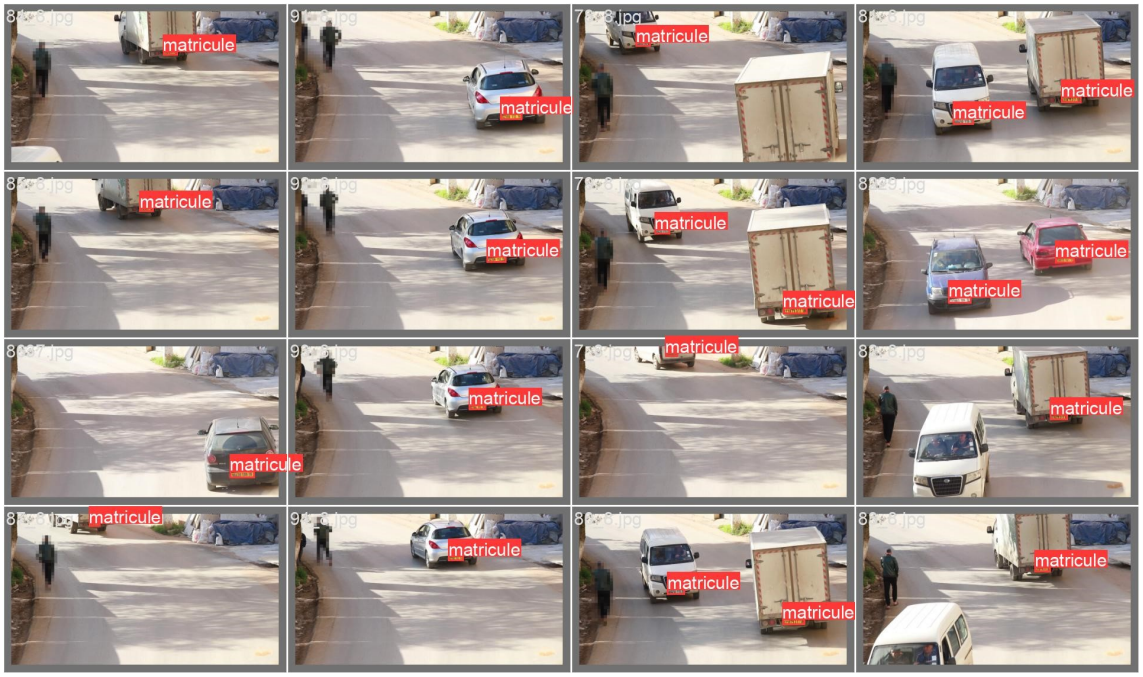


FIGURE 3.19 – Visualisation d'un lot des étiquettes du jeu de validation

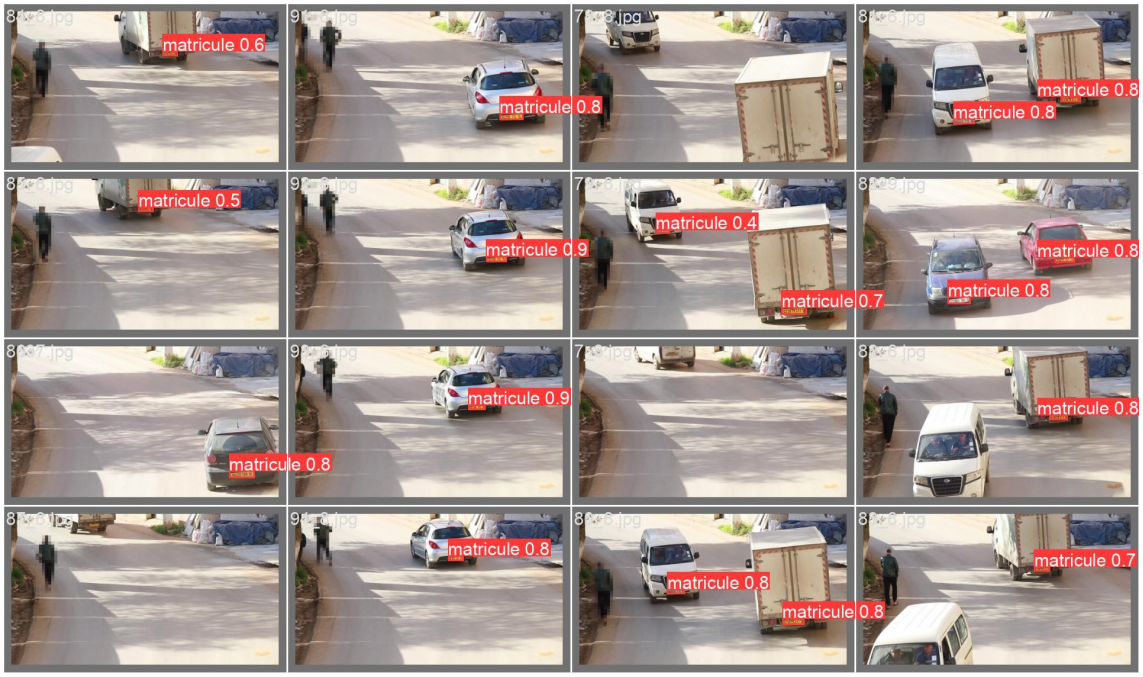


FIGURE 3.20 – Visualisation d'un lot des prédictions du jeu de validation

La comparaison montre que les plaques d'immatriculation très éloignées, ayant une faible résolution et peu d'informations visuelles, le modèle échoue à la détecter. À l'inverse, lorsque la plaque est nette et bien définie, la détection est fiable, ce qui souligne l'importance de la qualité et la résolution des données pour la performance du modèle.

### 3.5.4 Entraînement du détecteur de chiffres (YOLOv8s)

Le second modèle est responsable de la tâche plus fine de détection et d'identification de chacun des 10 chiffres (0 à 9) présents sur la plaque préalablement localisée. Pour cette tâche nécessitant une plus grande précision de classification, nous avons opté pour YOLOv8s.

#### Jeu de données

Un jeu de données beaucoup plus conséquent, totalisant 44 466 images de chiffres extraits de plaques d'immatriculation, a été constitué (70% train, 15% val et 15% test). La répartition des classes et les dimensions des boîtes englobantes sont visualisées dans lafigure 3.21 La grande diversité de ce jeu de données est un facteur clé pour la robustesse.

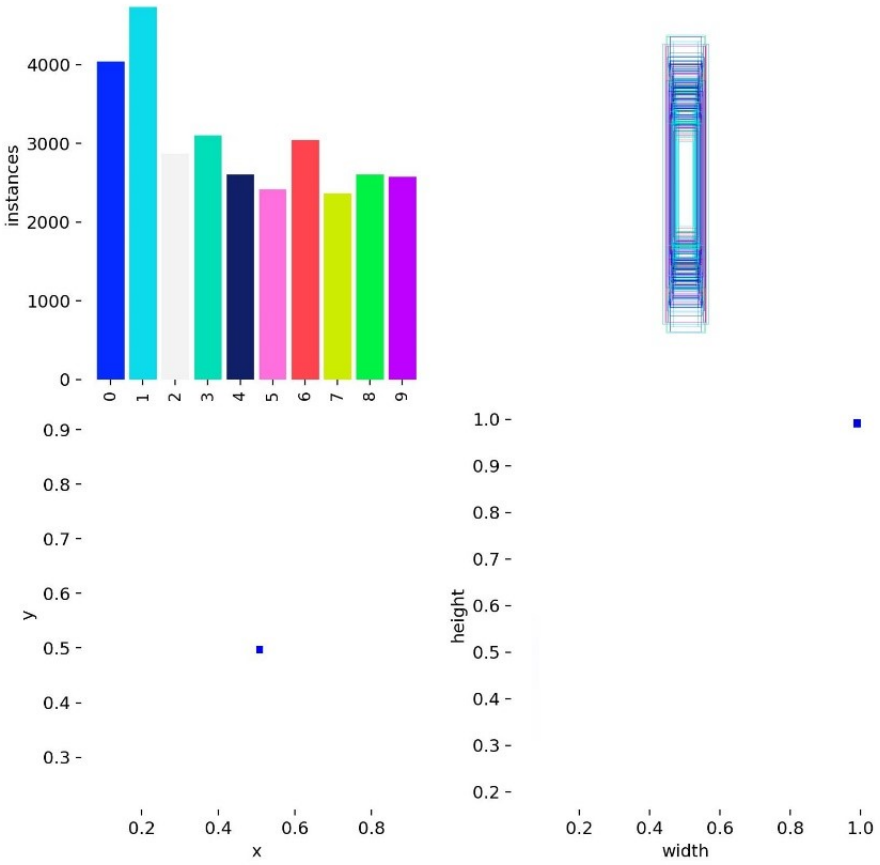


FIGURE 3.21 – Labels

#### Vérification des annotations

Un script spécifique a été utilisé pour vérifier la correspondance entre images et annotations.

### Prétraitements et Augmentation des données

L'augmentation a été ajustée pour préserver la lisibilité des chiffres (peu ou pas de rotations, translations, mises à l'échelle, cisaillements), tout en appliquant des variations HSV (hsv_h : 0.015, hsv_s : 0.7, hsv_v : 0.4), le retournement horizontal (fliplr : 0.5) et la mosaïque (mosaic : 1.0). Les exemples de lots d'entraînement dans la figure 3.22 montrent l'effet de ces augmentations.


FIGURE 3.22 – Exemples de lots d'entraînement

### Configuration de l'entraînement

L'entraînement a été effectué avec la librairie Ultralytics pour YOLOv8, en utilisant les poids pré-entraînés et les paramètres définis dans le scripte figure 3.23et le fichier hyp.yaml :

```python
from ultralytics import YOLO
import yaml

# Charger le modèle pré-entraîné
model = YOLO("/kaggle/input/yolov8ss/best.pt") # utilisation des poids pré_entrainés du modele YOLOv8s comme
                                               #point de départ (transfer learning)

# Charger le fichier d'hyperparamètres
with open("/kaggle/working/hyp.yaml","r") as f:
    hyp_dict=yaml.safe_load(f)

# Lancer l'entraînement
model.train(
    data="/kaggle/working/dataset_pour_la_segmentation_reconnaissance_chiffre_matricule/images/data.yaml",
    epochs=200,
    imgsz=320,
    batch=256,
    device="0,1",
    optimizer="SGD",
    project="/kaggle/working/YOLOv8_plate_result",
    name="yolov8s_digit",
    workers=4,
    patience=30,
    resume=False,             # Ne pas reprendre une session interrompue
    cache="ram",
    single_cls=False,
    **hyp_dict
)
```

FIGURE 3.23 – scripte d'entrainnement

```
*   `imgsz=320` : Taille d'image adaptée aux petits objets.
*   `batch=256` : Taille de lot importante.
*   `data data.yaml` : Fichier décrivant les chemins et les 10 classes (0-9).
*   `optimizer="SGD"` : Optimiseur Stochastic Gradient Descent.
```

FIGURE 3.24 – commentaire d'entrainnement

model = YOLO("/kaggle/input/yolov8ss/best.pt") : on 'a utiliser un yolov8s [44]Le modèle a déjà appris à détecter de chiffres manuscrits Il a déjà appris des caractéristiques visuelles utiles (formes, contrastes, alignements), Le dataset contient 10 Images de chiffres manuscrits isolés (de 0 à 9) ,taille des images 500x500 pixels, en niveaux de gris, ce qui est similaire à notre tâche.

- **Suivi de l'entraînement et Résultats** : L'analyse des résultats (yolov8s_results.csv, yolov8s_results.png) figure3.25 :

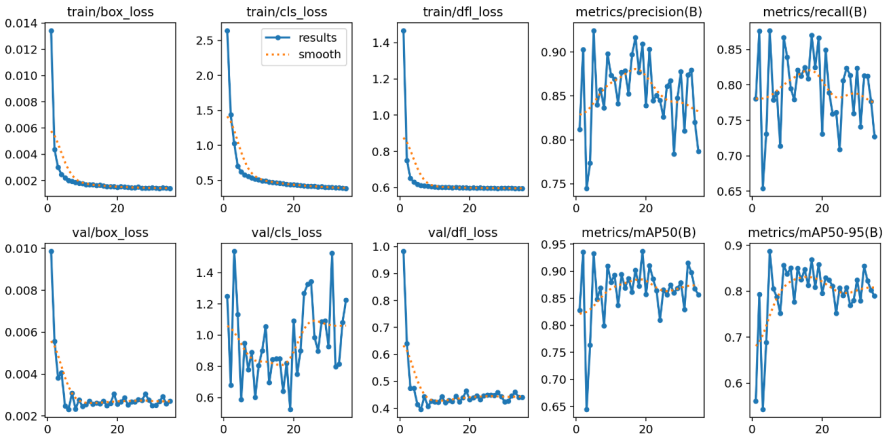


FIGURE 3.25 – yolov8s-results

Montre une bonne convergence des pertes (box_loss, cls_loss, dfl_loss). Les performances sur l'ensemble de validation sont excellentes : le modèle atteint une mAP@0.5 de 0.93687 et une mAP@0.5 :0.95 de 0.88701 (atteinte à l'époque 5). Ces très bons scores, en particulier la mAP50-95 qui est exigeante sur la localisation, sont attribuables à la grande taille et à la diversité du jeu de données, notamment l'ensemble de validation qui contient des images variées représentatives des conditions réelles, forçant le modèle à bien généraliser. Les courbes de Précision, Rappel, F1 et PR yolov8s_P_curve figure3.26, yolov8s_R_curve figure 3.27, yolov8s_F1_curve figure 3.28 yolov8s_PR_curve figure 3.29 :

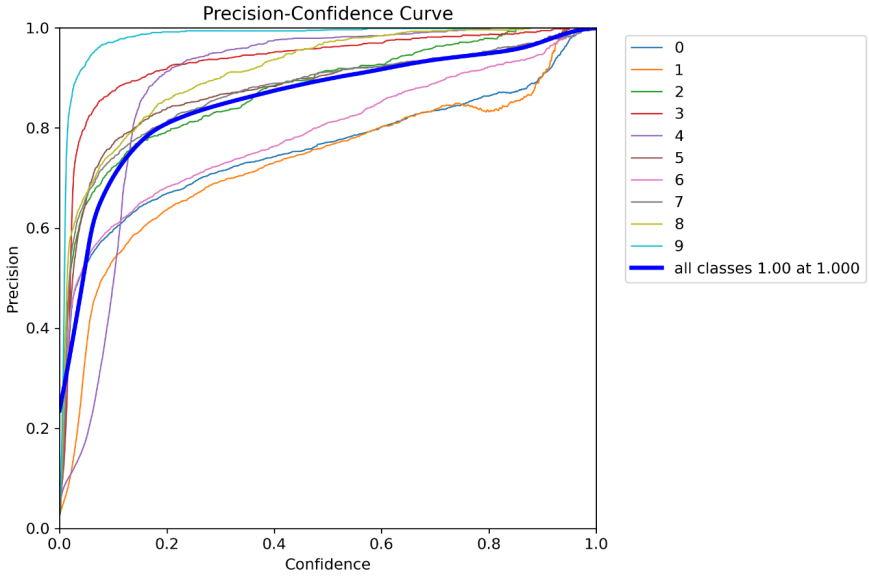


FIGURE 3.26 – yolov8s_P_curve

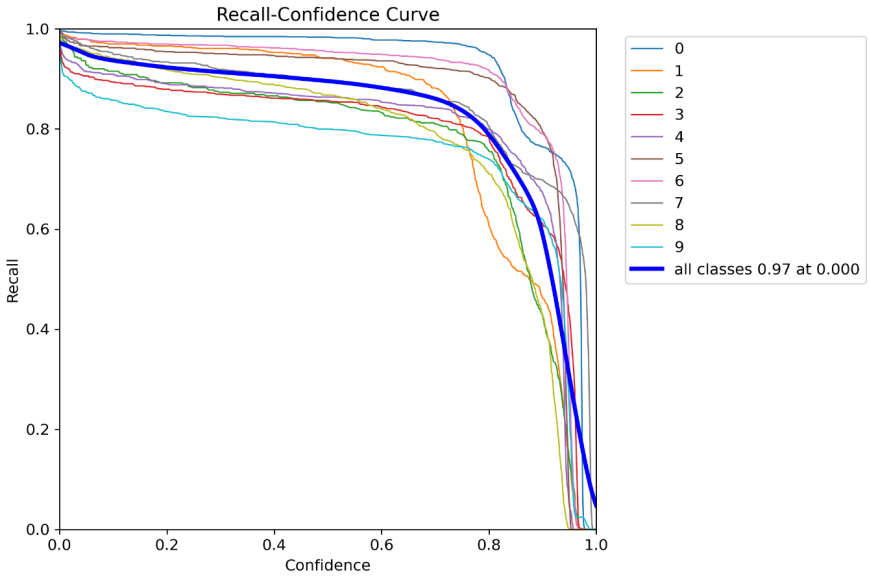


FIGURE 3.27 – yolov8s_R_curve

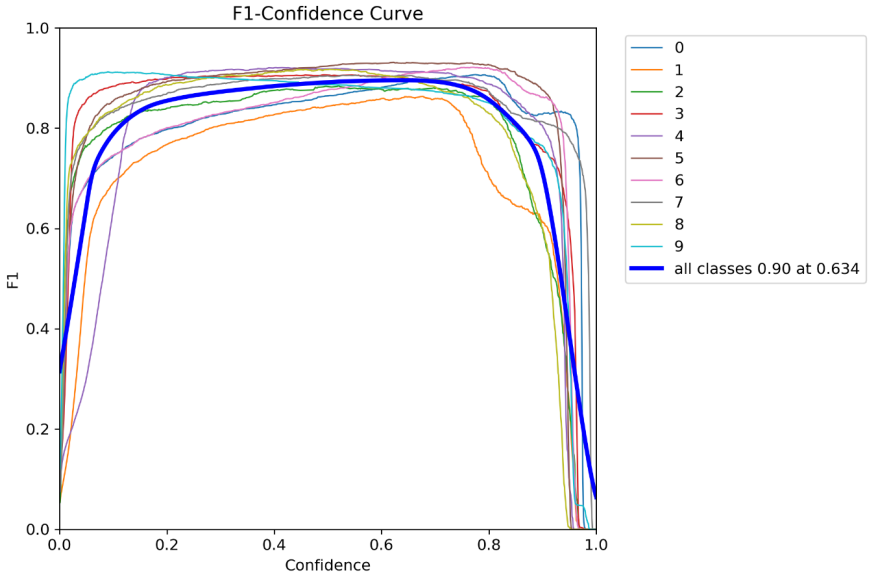


FIGURE 3.28 – yolov8s_F1_curve

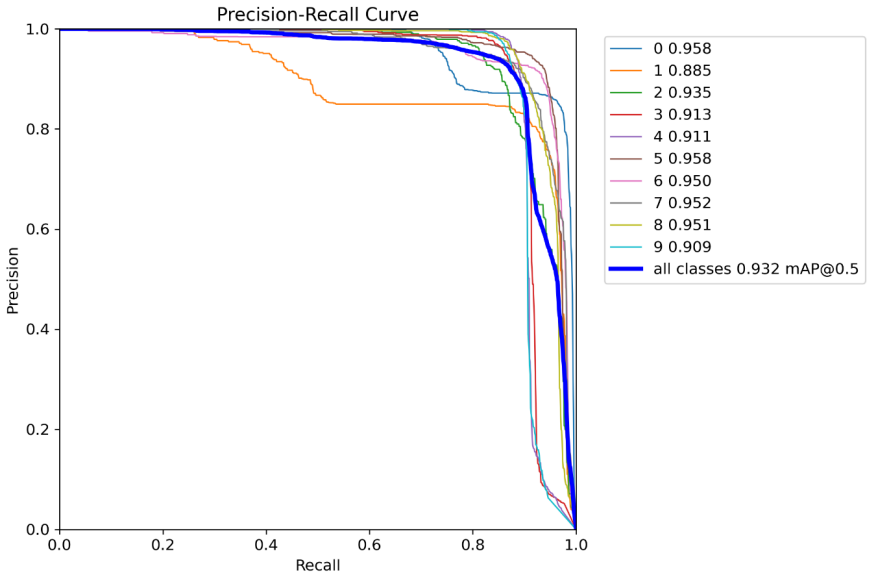


FIGURE 3.29 – yolov8s_PR_curve

confirment ces excellentes performances. La matrice de confusion normalisée yolov8s_confusion_matrix_normaliz
figure 3.30 :


FIGURE 3.30 – yolov8s_confusion_matrix_normalized

Est particulièrement intéressante : elle montre que la plupart des chiffres sont très bien reconnus (diagonale proche de 1). Les confusions les plus notables, bien que faibles, semblent apparaître entre '0' et '9' 11%, '1' et '7' 14%, ou encore '6' et '8' 7%, ce qui est cohérent avec les similarités visuelles entre ces chiffres. Les exemples de prédictions peuvent être comparés en observant les figures suivantes (yolov8s_val_batch0_pred figure 3.31) comparée à (yolov8s_val_batch0_labels figure 3.32) et (yolov8s_val_batch1_pred figure 3.33) comparée à (yolov8s_val_batch1_labels figure 3.34) :

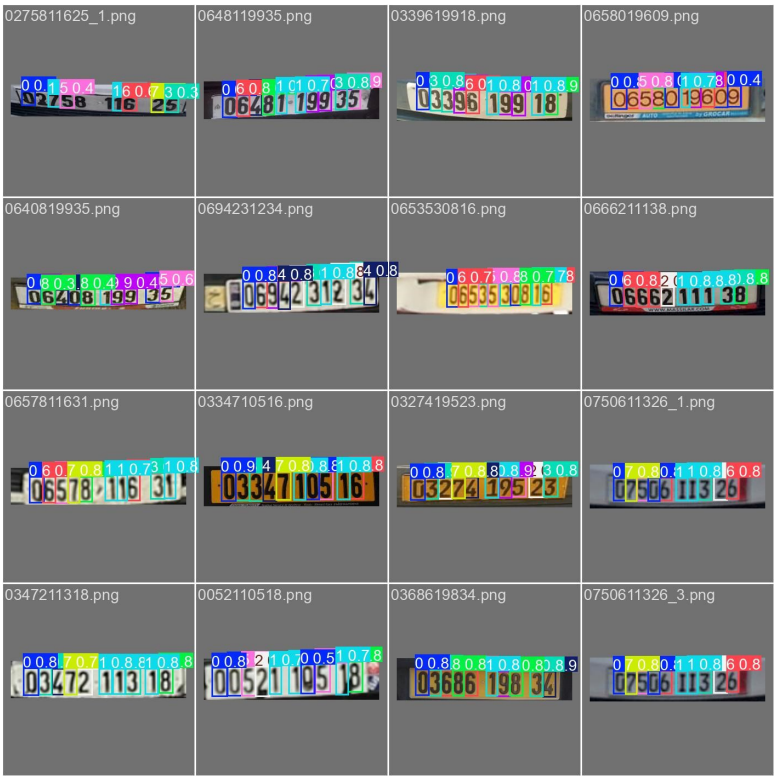


FIGURE 3.31 – yolov8s_val_batch0_pred


FIGURE 3.32 – yolov8s_val_batch0_labels

FIGURE 3.33 – yolov8s_val_batch1_pred

FIGURE 3.34 – yolov8s_val_batch1_labels

Confirment la haute précision du modèle dans la détection et l'identification des chiffres sur les images de validation. les deux modèles ont été entraînés avec succès sur des jeux de données dédiés et volumineux, en utilisant des configurations et des techniques d'augmentation adaptées à chaque tâche spécifique. L'analyse détaillée des métriques et des visualisations confirme l'atteinte d'excellentes performances, notamment pour le modèle YOLOv8s sur la tâche complexe de reconnaissance de chiffres, validant ainsi l'approche méthodologique choisie.

## 3.6 Optimisation et déploiement des modèles

Une fois les modèles YOLOv5n et YOLOv8s entraînés avec succès, une étape cruciale consiste à les optimiser et à les convertir dans des formats adaptés aux plateformes cibles (PC, Raspberry Pi 4, smartphone Android) afin de permettre leur déploiement effectif. Cette phase vise à concilier les exigences souvent contradictoires de vitesse d'inférence, de taille mémoire et de compatibilité matérielle et logicielle.

### 3.6.1 Objectifs de l'optimisation

L'objectif principal de l'optimisation est de rendre les modèles exécutables efficacement sur les différentes plateformes, chacune ayant ses propres contraintes :

• **Vitesse d'inférence** : Essentielle pour le traitement en temps réel des flux vidéo, particulièrement sur les appareils aux ressources limitées comme le Raspberry Pi et les smartphones.

•**Taille mémoire** : Les modèles doivent être suffisamment compacts pour être stockés et chargés en mémoire sur les appareils cibles, notamment les smartphones Android où l'espace peut être restreint.

•**Compatibilité** : Les formats de modèles doivent être compatibles avec les frameworks d'inférence disponibles et optimisés pour chaque plateforme (ex : OpenVINO pour les GPU Intel/ARM et CPU

Intel/ARM sur PC/RPi, TensorFlow Lite pour Android).

Les tailles des modèles obtenus avant et après conversion illustrent l'impact de ces optimisations et les choix effectués :

**1. YOLOv5n (détection plaque) :**

best.pt (PyTorch Float32) : 3.60 Mo.

- best-fp16.tflite (TensorFlow Lite Float16) : 3.45 Mo (Réduction de taille modeste mais gain de vitesse attendu sur matériel compatible)

**2.YOLOv8s (détection chiffres) :**

- best.pt (PyTorch Float32) : 21.9 Mo
- best.onnx (ONNX Float32) : 43.6 Mo (Augmentation due à la structure du format ONNX)
- best_float32.tflite (TensorFlow Lite Float32) : 43.6 Mo (Taille similaire à ONNX, pas de réduction car pas de quantification)

### 3.6.2 Conversion de YOLOv5n pour l'embarqué

Pour le modèle YOLOv5n, responsable de la détection rapide des plaques, la priorité a été donnée à la vitesse d'inférence pour minimiser la latence de la première étape du pipeline.

• **Export vers TFLite :** Le modèle a été converti au format TensorFlow Lite (TFLite), un framework spécifiquement conçu pour l'inférence sur appareils mobiles et embarqués [45]. Le script convert_yolov5n.py figure 3.35automatise cette conversion en utilisant la fonctionnalité d'export intégrée à la librairie YOLOv5 (export.py).

```
# Extrait simplifié de la logique de convert_yolov5n.py
# Utilisation du script export.py de YOLOv5
subprocess.Popen([
    "python3",
    "path/to/yolov5/export.py",
    "--weights", "path/to/yolov5n/best.pt",
    "--imgsz", 512,
    "--include", "tflite",
    "--half", # Activation de la quantification FP16
    # ... autres options ...
])
```

FIGURE 3.35 – converti yolov5n au format TensorFlow Lite

• **Quantification Float16 (FP16) :** Lors de la conversion TFLite, une quantification post-entraînement en float16 a été appliquée. Cette technique représente les poids du modèle avec des nombres flottants de 16 bits au lieu des 32 bits standards (float32). L'avantage principal est une réduction de la taille du modèle et surtout une accélération significative de l'inférence sur le matériel compatible (GPU mobiles récents, certains DSPs)[46][47]. Bien que la quantification FP16 puisse

entraîner une légère perte de précision par rapport au FP32 [46], ce compromis est délibérément choisi pour la détection de plaques. Notre priorité ici est la vitesse : nous voulons que cette première étape soit la plus rapide possible, même au prix d'une infime baisse de précision, afin de consacrer davantage de temps de calcul à l'étape suivante de détection des chiffres, qui est intrinsèquement plus complexe et exigeante en précision.

### 3.6.3 Conversion de YOLOv8s pour l'embarqué

Pour le modèle YOLOv8s, chargé de la détection et de la classification précise des 10 chiffres, la stratégie d'optimisation est différente, visant un équilibre entre précision maximale et compatibilité multi-plateforme.

- **Export vers ONNX et TFLite :** Le modèle a été exporté vers deux formats pour s'adapter aux différents environnements de déploiement :
- **1.ONNX (Open Neural Network Exchange) :** Ce format ouvert standard favorise l'interopérabilité entre les frameworks [48][49] Il a été choisi pour le déploiement sur PC et Raspberry Pi, car il permet une exécution optimisée via le framework OpenVINO, performant sur les architectures CPU/GPU Intel et ARM, assurant ainsi la compatibilité sur des machines sans GPU NVIDIA dédié.
- **2.TFLite (TensorFlow Lite) :** Ce format est incontournable pour le déploiement sur la plateforme Android. [45][50] Le script convert_yolov8s.py figure 3.36utilise la méthode export() de la librairie Ultralytics pour générer le format TFLite et ONNX.

```python
# Extrait de convert_yolov8s.py (pour TFLite)
from ultralytics import YOLO

model = YOLO("best.pt")

model.export(
    format="tflite",
    imgsz=320,
    half=False, # Maintien en FP32 pour la précision
    # ... autres options ...
)
# Une commande similaire serait utilisée pour exporter en format="onnx"
```

FIGURE 3.36 – converti yolov5n au format TensorFlow Lite/ONNX

- **Maintien en Float32 (FP32)** : Contrairement à YOLOv5n, aucune quantification agressive (ni FP16, ni INT8) n'a été appliquée lors de la conversion de YOLOv8s. Le modèle a été conservé en précision flottante standard (float32). La justification principale est la priorité absolue donnée à la précision pour la tâche critique de reconnaissance des 10 chiffres. Une erreur, même minime, sur un seul chiffre invalide le numéro de plaque complet. Conserver la précision float32 minimise le risque de

dégradation des performances de classification potentiellement induit par la quantification [50]. Bien que cela résulte en une taille de modèle plus importante (43.6 Mo pour TFLite/ONNX) et une vitesse d'inférence potentiellement plus lente qu'une version FP16, ce choix représente un équilibre réfléchi : nous maximisons la précision là où elle est indispensable (reconnaissance des chiffres), tout en assurant la compatibilité via les formats ONNX (pour PC/RPi avec OpenVINO) et TFLite (pour Android).

### 3.6.4 Frameworks d'inférence utilisés

Le choix des formats de modèle (TFLite, ONNX) et des plateformes cibles a conduit à l'utilisation de frameworks d'inférence spécifiques, optimisés pour chaque environnement :

• **OpenVINO™ Toolkit (PC et Raspberry Pi)** : Développé par Intel, OpenVINO est utilisé pour exécuter le modèle YOLOv8s au format ONNX sur les plateformes PC (CPU/GPU Intel) et Raspberry Pi (CPU ARM) [51]. Il optimise l'inférence pour ces architectures matérielles, permettant d'obtenir de bonnes performances même sans GPU dédié.

• **TensorFlow Lite (Android)** : C'est le framework standard pour l'inférence sur Android [46]. Il est utilisé dans l'application mobile (voir YoloDetector.java) pour charger et exécuter les deux modèles au format .tflite : best-fp16.tflite pour YOLOv5n et best_float32.tflite pour YOLOv8s, en bénéficiant des optimisations spécifiques à l'écosystème Android.

En combinant des stratégies d'optimisation différenciées (FP16 pour la vitesse sur YOLOv5n, FP32 pour la précision sur YOLOv8s) et des formats d'export adaptés (TFLite, ONNX) avec les frameworks d'inférence optimisés pour chaque plateforme (TensorFlow Lite, OpenVINO), nous avons pu déployer notre pipeline de détection sur une gamme variée d'appareils tout en respectant au mieux les contraintes spécifiques de chaque tâche et de chaque plateforme.

## 3.7 Développement des systèmes applicatifs

### 3.7.1 Interface graphique sur PC

L'interface graphique sur PC constitue l'un des trois systèmes applicatifs développés dans le cadre de ce projet, permettant la détection et la vérification des plaques d'immatriculation en temps réel. Cette section détaille les aspects techniques de cette implémentation.

#### Technologies utilisées

L'interface PC repose sur un ensemble de technologies modernes et complémentaires :

1. **Python** : Langage de programmation principal, choisi pour sa polyvalence et sa riche écosystème de bibliothèques pour le traitement d'images et l'apprentissage automatique.

2. **Tkinter** : Framework d'interface graphique natif de Python, utilisé pour créer l'ensemble des éléments visuels de l'application :
   - Fenêtres et cadres personnalisés.
   - Boutons arrondis et stylisés.
   - Zone d'affichage vidéo.
   - Liste des détections avec défilement.
3. **PIL (Python Imaging Library) / Pillow** : Utilisée pour la manipulation d'images, notamment :
   - Conversion entre formats d'images (OpenCV/NumPy arrays ↔ Tkinter PhotoImage).
   - Redimensionnement et prétraitement des images d'arrière-plan.
   - Affichage des flux vidéo dans l'interface Tkinter.
4. **OpenCV (cv2)** : Bibliothèque de vision par ordinateur utilisée pour :
   - Capture vidéo depuis la caméra.
   - Prétraitement des images (redimensionnement, normalisation, conversion de couleurs).
   - Dessin des rectangles de détection sur les images.
   - Algorithmes de Non-Maximum Suppression (NMS) pour filtrer les détections redondantes.
5. **OpenVINO™ Toolkit** : Framework d'inférence d'Intel utilisé pour :
   - Chargement et exécution optimisée des modèles ONNX et TFLite.
   - Accélération matérielle sur CPU Intel et GPU compatible.
   - Gestion efficace des tenseurs d'entrée/sortie.
6. **NumPy** : Bibliothèque de calcul numérique utilisée pour :
   - Manipulation efficace des tableaux multidimensionnels.
   - Opérations mathématiques sur les coordonnées des boîtes englobantes.
   - Prétraitement et post-traitement des données d'inférence.
7. **Threading** : Module de gestion des threads permettant :
   - Exécution parallèle de la capture vidéo et de l'interface utilisateur.
   - Maintien de la réactivité de l'interface pendant le traitement vidéo intensif.
   - Synchronisation des accès aux ressources partagées via des verrous (locks).
8. **Firebase Admin SDK** : Interface Python pour Firebase, utilisée pour :
   - Authentification sécurisée via un compte de service.
   - Lecture/écriture dans la base de données Realtime Database.
   - Vérification du statut des vignettes associées aux plaques détectées.

**Architecture logicielle**

L'application est structurée selon une architecture modulaire figure3.37 avec séparation claire des responsabilités :

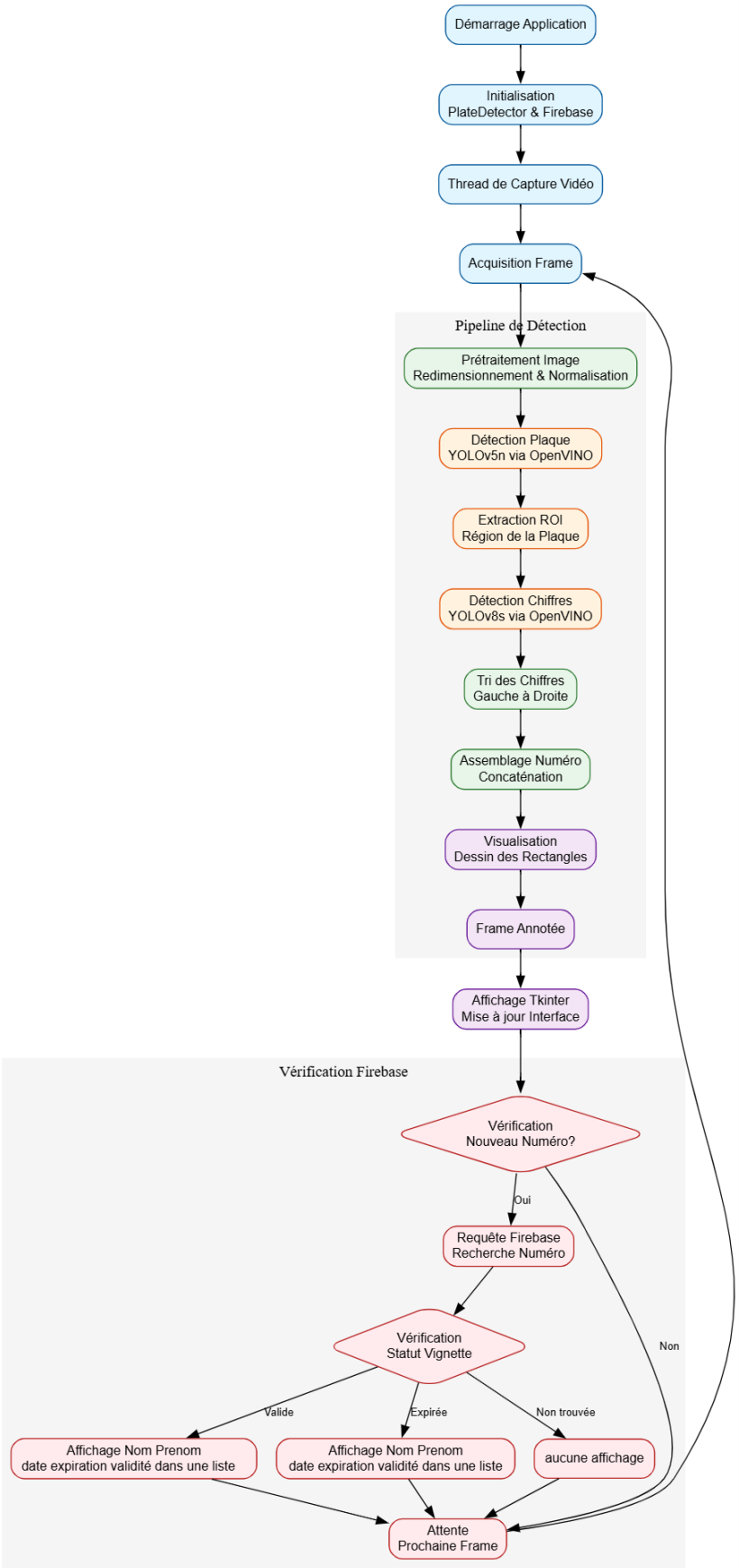


FIGURE 3.37 – Architecture modulaire

1. **Structure générale** :

Le code est organisé autour de deux classes principales :

- **'PlateDetector'** : Responsable de tout le pipeline de détection et de reconnaissance
- **'CameraApp'** : Gère l'interface utilisateur et la logique d'application.

Cette séparation permet une maintenance plus aisée et une meilleure évolutivité du code.

2. **Pipeline de traitement vidéo** :

Le pipeline de traitement vidéo constitue le cœur fonctionnel de l'application et se décompose en plusieurs étapes séquentielles :

a) **Capture vidéo** :

La capture vidéo est réalisée dans un thread dédié pour maintenir la réactivité de l'interface.

Le thread de capture exécute une boucle continue qui :

- Lit une image depuis la caméra.
- La stocke de manière thread-safe.
- Déclenche le traitement et l'affichage.

b) **Détection des plaques d'immatriculation** : La première étape du pipeline de détection utilise le modèle YOLOv5n optimisé en float16.

Les détections sont ensuite filtrées par score de confiance et par Non-Maximum Suppression pour éliminer les détections redondantes.

c) **Extraction et prétraitement des régions d'intérêt** : Pour chaque plaque détectée, une région d'intérêt (ROI) est extraite et préparée pour la détection des chiffres.

d) **Détection et reconnaissance des chiffres** : La ROI extraite est ensuite traitée par le modèle YOLOv8s pour détecter et identifier les chiffres.

Le prétraitement inclut le redimensionnement avec préservation du ratio et le padding (technique "letterbox") pour adapter l'image au format d'entrée du modèle.

e) **Assemblage du numéro de plaque** Les chiffres détectés sont triés de gauche à droite selon leur position horizontale, puis assemblés pour former le numéro complet de la plaque.

f) **Visualisation des résultats** Les résultats de la détection sont visualisés directement sur l'image.

3. **Affichage et mise à jour de l'interface** :

L'interface est mise à jour à intervalles réguliers pour afficher le flux vidéo traité.

**Intégration Firebase**

L'application intègre Firebase Realtime Database pour vérifier la validité des vignettes associées aux plaques détectées.

1. **Initialisation et authentification** :

   L'initialisation de Firebase est réalisée au démarrage de l'application.

   L'authentification utilise un fichier de clé de compte de service JSON stocké localement, garantissant un accès sécurisé à la base de données.

2. **Lecture et vérification des données** :

   Pour chaque plaque détectée, l'application interroge la base de données Firebase pour vérifier si le numéro est enregistré et si la vignette associée est valide.

3. **Affichage des résultats de vérification** :

   Les résultats de la vérification sont affichés dans la liste des détections.

   L'application gère également différents types d'erreurs Firebase de manière détaillée, avec des messages spécifiques pour les problèmes de configuration, de permissions ou de réseau.

**Gestion des performances et optimisations**

Plusieurs optimisations ont été implémentées pour garantir des performances en temps réel :

**Traitement multi-thread** : Séparation de la capture vidéo et de l'interface utilisateur pour maintenir la réactivité.

**Verrous de synchronisation** : Utilisation de ‘threading.Lock()‘ pour éviter les conflits d'accès aux frames.

**Détection de matériel** : Tentative d'utilisation du GPU avec repli automatique sur CPU si non disponible.

**Mise en cache des détections** : Utilisation d'un ensemble (‘set‘) pour éviter les doublons de plaques détectées.

**Gestion efficace de la mémoire** : Libération des ressources non utilisées et nettoyage explicite.

Ces optimisations permettent à l'application de fonctionner efficacement même sur des ordinateurs aux ressources limitées, tout en maintenant une détection fiable et une interface réactive. l'interface graphique PC développée constitue une solution complète et performante pour la détection et la vérification des plaques d'immatriculation en temps réel, combinant des technologies modernes de vision par ordinateur, d'inférence optimisée et de gestion de base de données.

### 3.7.2 Système embarqué sur Raspberry Pi 4

Le déploiement du système de détection de plaques d'immatriculation sur Raspberry Pi 4 représente une étape cruciale dans la démonstration de la polyvalence et de l'adaptabilité de notre solution. Cette implémentation embarquée conserve les fonctionnalités essentielles de la version PC tout en intégrant des adaptations spécifiques pour tirer parti des caractéristiques matérielles du Raspberry Pi 4 et surmonter ses limitations.

### Adaptations pour Raspberry Pi 4

#### Gestion de la caméra avec Picamera2 :

L'une des principales adaptations concerne l'utilisation de la bibliothèque Picamera2, spécifiquement conçue pour les caméras officielles du Raspberry Pi. Contrairement à la version PC qui utilise OpenCV pour la capture vidéo, l'implémentation RPi4 exploite l'API native Picamera2 pour une meilleure performance et une intégration optimale avec le matériel.

L'initialisation de la caméra est réalisée avec une configuration spécifique qui tire parti des capacités du module caméra Raspberry Pi.

### Optimisation des performances

#### Gestion de la résolution

Le système embarqué utilise une stratégie de résolution à deux niveaux pour optimiser les performances :

1. **Capture haute résolution** : La caméra capture les images à une résolution native élevée (3280×2464 pixels) pour maximiser la qualité des détails nécessaires à la détection précise des plaques et des chiffres.

2. **Affichage redimensionné** : L'interface utilisateur affiche les images à une résolution réduite (1280×720 pixels) pour maintenir la fluidité de l'interface graphique.

Cette approche double permet de concilier la précision de la détection (qui bénéficie d'une haute résolution) avec la fluidité de l'interface (qui nécessite un traitement plus léger).

#### Optimisation de la boucle de capture

La boucle de capture a été spécifiquement adaptée pour le Raspberry Pi 4, avec une gestion efficace des ressources.

Les optimisations notables incluent :

- **Traitement périodique** : Le système ne traite pas chaque image capturée mais seulement une image toutes les 0.5 secondes, réduisant considérablement la charge CPU tout en maintenant une détection efficace.
- **Gestion de la mémoire** : Utilisation de verrous (threading.Lock) pour éviter les conflits d'accès mémoire entre les threads de capture et d'affichage.
- **Pauses stratégiques** : Introduction de micro-pauses (time.sleep(0.01)) pour réduire l'utilisation du CPU et permettre à d'autres processus de s'exécuter.

#### Technologies utilisées

Le système embarqué sur Raspberry Pi 4 utilise un ensemble de technologies similaires à la version PC, avec adaptation de Gestion de la caméra avec Picamera2.

#### Architecture logicielle

L'architecture logicielle du système embarqué sur Raspberry Pi 4 conserve la structure modulaire de la version PC

#### Pipeline de détection

Le pipeline de détection reste fondamentalement le même que sur la version PC.

#### Fonctionnalités

Le système embarqué sur Raspberry Pi 4 offre les mêmes fonctionnalités principales que la version PC :

(a) Détection en temps réel des plaques d'immatriculation.

(b) Reconnaissance des chiffres composant le numéro de plaque.

(c) Vérification dans Firebase du statut des vignettes associées.

(d) Affichage des résultats.

(e) Interface utilisateur intuitive.

L'adaptation du système de détection de plaques d'immatriculation pour le Raspberry Pi 4 démontre la flexibilité de l'architecture logicielle développée. Malgré les contraintes matérielles inhérentes à une plateforme embarquée, le système maintient un niveau de performance satisfaisant grâce à des optimisations ciblées, notamment l'utilisation de Picamera2, la gestion intelligente de la résolution et le traitement périodique des images.

Cette implémentation embarquée constitue un maillon essentiel dans l'écosystème global de notre solution, permettant un déploiement dans des contextes où l'encombrement, la consommation énergétique et le coût sont des facteurs critiques, tout en conservant les fonctionnalités essentielles de détection et de vérification des plaques d'immatriculation.

### 3.7.3 Application mobile Android

L'application mobile Android constitue le troisième volet de notre système de détection de plaques d'immatriculation, offrant une solution portable et accessible aux utilisateurs finaux.

Cette implémentation mobile permet non seulement la détection en temps réel des plaques et des chiffres, mais également la gestion des comptes utilisateurs et le suivi des vignettes associées aux véhicules. Cette section détaille l'architecture, les technologies et les fonctionnalités de cette application.

**Technologies utilisées**

(a) **Langage et SDK** : L'application est développée en Java, le langage traditionnel pour le développement Android, offrant une stabilité éprouvée et une compatibilité maximale avec l'écosystème Android. Le développement s'appuie sur l'Android SDK (Software Development Kit) qui fournit les API nécessaires pour créer des interfaces utilisateur, gérer le cycle de vie des activités et accéder aux fonctionnalités du système.

(b) **Gestion de la caméra avec CameraX** : L'application utilise CameraX, une bibliothèque Jetpack qui simplifie considérablement le développement d'applications utilisant la caméra. CameraX offre une API cohérente sur différents appareils Android, gérant automatiquement les complexités liées aux différentes implémentations de caméra des fabricants.

**TensorFlow Lite pour l'inférence embarquée**

TensorFlow Lite est utilisé comme moteur d'inférence pour exécuter les modèles YOLO optimisés sur l'appareil mobile. Cette bibliothèque est spécialement conçue pour les appareils à ressources limitées, offrant des performances optimales tout en minimisant l'empreinte mémoire.

(a) **Firebase pour l'authentification et la base de données**

L'application intègre Firebase Authentication pour la gestion des comptes utilisateurs et Firebase Realtime Database pour la récupération des informations sur les plaques d'immatriculation et les vignettes.

(b) **Autres bibliothèques et composants**

a) **SharedPreferences** pour la persistance locale des données utilisateur.

b) **SimpleDateFormat** pour le formatage des dates d'expiration des vignettes.

c) **Executors** pour la gestion des threads et l'exécution asynchrone.

d) **Canvas et Paint** pour le dessin des rectangles de détection et des étiquettes.

**Intégration de TensorFlow Lite**

L'intégration de TensorFlow Lite constitue un aspect central de l'application, permettant l'exécution efficace des modèles YOLO optimisés sur l'appareil mobile.

1. **Chargement des modèles**

L'application charge deux modèles TFLite distincts.

• **YOLOv5n** pour la détection des plaques (format float16).

• **YOLOv8s** pour la reconnaissance des chiffres (format float32).

Les modèles sont stockés dans le dossier assets de l'application et chargés en mémoire via un MappedByteBuffer, ce qui permet un accès efficace aux données du modèle sans copie supplémentaire en mémoire.

2. **Prétraitement des images**

Le prétraitement des images est une étape cruciale pour adapter les données de la caméra au format attendu par les modèles YOLO :

a) Conversion YUV vers Bitmap.

b) Prétraitement pour le modèle de détection de plaques.

c) Prétraitement pour le modèle de reconnaissance des chiffres.

Les étapes clés du prétraitement incluent :

a) Redimensionnement de l'image à la taille d'entrée du modèle (512×512 pour les plaques, 320×320 pour les chiffres).

b) Normalisation des valeurs de pixels entre 0 et 1.

c) Réorganisation des canaux selon le format attendu par le modèle (NHWC).

d) Rotation de l'image selon l'orientation de l'appareil.

3. **Exécution de l'inférence**

L'inférence est réalisée séquentiellement : d'abord la détection des plaques, puis pour chaque plaque détectée, la reconnaissance des chiffres.

4. **Post-traitement des résultats**

Le post-traitement transforme les sorties brutes des modèles en détections exploitables :

a) Post-traitement pour les plaques d'immatriculation

b) Post-traitement pour les chiffres

Les étapes clés du post-traitement incluent :

-Filtrage par seuil de confiance pour éliminer les détections peu fiables.

-Application de Non-Maximum Suppression pour éliminer les détections redondantes.

-Conversion des coordonnées normalisées en coordonnées d'image.

-Tri des chiffres de gauche à droite pour former le numéro complet.

-Transformation des coordonnées pour l'affichage correct dans l'OverlayView.

**Architecture logicielle**

L'application est structurée selon une architecture modulaire avec une séparation claire des responsabilités entre les différentes activités et classes.

1. **Activités principales**

L'application est composée de quatre activités principales, chacune avec un rôle spécifique :

• **MainActivity** : Point d'entrée de l'application, offrant un accès aux fonctionnalités Principales

• **Authentifier** : Gère l'authentification des utilisateurs via Firebase

• **Profileactivity** : Affiche les informations de l'utilisateur et de sa vignette

• **Detection** : Implémente la détection en temps réel des plaques d'immatriculation

2. **Classes de support**

Plusieurs classes de support complètent l'architecture :

• **YoloDetector** : Implémente l'analyse d'images et la détection avec TensorFlow Lite.

• **DetectionResult** : Structure de données pour les résultats de détection.

3. **Flux de l'application**

Le flux de l'application suit un parcours logique :

• **Démarrage** : L'utilisateur lance l'application et arrive sur MainActivity

• **Navigation** : L'utilisateur choisit entre l'authentification et la détection directe.

• **Authentification** : Si l'utilisateur choisit de se connecter :

  a) Saisie des identifiants ou connexion automatique via "Se souvenir de moi".

  b) Redirection vers l'écran de profil en cas de succès

• **Profil** : Affichage des informations utilisateur et vignette

  a) Vérification de la validité de la vignette

  b) Notification si la vignette est expirée

• **Détection** : Lorsque l'utilisateur accède à la détection :

  a) Demande de permission caméra si nécessaire

  b) Initialisation de la caméra et du détecteur

  c) Analyse en temps réel du flux vidéo

  d) Affichage des détections et des informations associées

**Interface utilisateur**

L'interface utilisateur de l'application est conçue pour être intuitive et fonctionnelle, avec une attention particulière portée à l'expérience utilisateur.

1. **Layouts XML**

Les layouts XML définissent la structure visuelle de chaque écran :

• **Activity_main.xml** : Écran d'accueil avec les boutons de navigation.

• **Authentifier.xml** : Formulaire de connexion avec champs email/mot de passe.

• **Profileactivity.xml** : Affichage des informations utilisateur et vignette.

• **Detection.xml** : Écran de détection avec prévisualisation caméra et overlay.

2. **OverlayView pour l'affichage des détections** La classe OverlayView est un composant personnalisé qui se superpose à la prévisualisation de la caméra pour afficher les rectangles de détection et les numéros de plaque reconnus.

Cette implémentation permet :

(a) Superposition transparente sur la prévisualisation de la caméra

(b) Dessin en temps réel des rectangles de détection

(c) Affichage des numéros reconnus au-dessus de chaque plaque

(d) Mise à jour dynamique à chaque nouvelle détection

(e) Transformation des coordonnées

Un aspect crucial de l'affichage des détections est la transformation des coordonnées entre l'image analysée et la vue d'affichage

Cette transformation garantit que les rectangles de détection s'affichent correctement sur la prévisualisation, quelle que soit la différence de résolution entre l'image analysée et l'affichage.

**Fonctionnalités**

1. **Gestion de compte utilisateur** :

L'application offre une gestion complète des comptes utilisateurs : Authentification via email/mot de passe avec Firebase Authentication Mémorisation des identifiants avec l'option "Se souvenir de moi" Récupération de mot de passe via email déconnexion sécurisée avec effacement des données locales.

2. **Consultation et notification vignette** :

L'application permet aux utilisateurs de consulter les informations relatives à leur vignette :

-Affichage des informations : numéro de plaque, statut de la vignette, date d'expiration.

-Vérification de validité : comparaison de la date d'expiration avec la date actuelle Notification en cas d'expiration de la vignette

3. **Détection en temps réel** :

La fonctionnalité phare de l'application est la détection en temps réel des plaques d'immatriculation :

- Capture vidéo en continu via CameraX.
- Analyse d'images avec YoloDetector.java.
- Détection des plaques avec YOLOv5n.
- Reconnaissance des chiffres avec YOLOv8s.
- Affichage des résultats via OverlayView.
- Vérification en temps réel des plaques détectées dans Firebase.

L'application mobile Android constitue une composante essentielle du système global de détection de plaques d'immatriculation, offrant une solution portable et accessible aux utilisateurs finaux. Grâce à l'intégration de technologies modernes comme CameraX, TensorFlow Lite et Firebase, l'application permet une détection en temps réel des plaques et une vérification instantanée des vignettes associées. L'architecture modulaire et la séparation claire des responsabilités entre les différentes classes facilitent la maintenance et l'évolution future de l'application. L'interface utilisateur intuitive et les fonctionnalités complètes de gestion de compte et de notification garantissent une expérience utilisateur optimale.

Cette implémentation mobile démontre la polyvalence de notre solution de détection de plaques d'immatriculation, capable de fonctionner efficacement sur différentes plateformes, du PC au Raspberry Pi en passant par les appareils Android.

## 3.8 Conclusion

Ce chapitre a permis de présenter l'ensemble de la méthodologie mise en œuvre pour le développement du système ANPR, depuis le choix des modèles et la préparation des données jusqu'à l'optimisation et le déploiement multiplateforme. L'utilisation combinée de YOLOv5n pour la détection rapide des plaques et de YOLOv8s pour la reconnaissance fine des caractères a permis d'obtenir un équilibre satisfaisant entre précision, vitesse d'exécution et compatibilité avec les dispositifs embarqués. Les tests réalisés ont confirmé la robustesse et la fiabilité de notre approche, même dans des conditions réelles complexes. La modularité du système, couplée aux différentes interfaces déployées (PC, Raspberry Pi, application mobile Android), confirme la pertinence de la méthodologie adoptée et ouvre la voie à des extensions futures dans le domaine de la surveillance intelligente et des systèmes embarqués.

CHAPITRE

# 4

# EXPÉRIMENTATIONS ET RÉSULTATS

## 4.1 Introduction

Dans ce chapitre, nous présentons les résultats expérimentaux obtenus lors de la mise en œuvre de notre système de reconnaissance automatique de plaques d'immatriculation (ANPR) basé sur une architecture à double modèle YOLO. L'objectif est d'évaluer les performances de la solution proposée dans différents environnements matériels et conditions réelles. Trois plateformes ont été ciblées : un ordinateur portable sous Ubuntu, un Raspberry Pi 4, et un smartphone Android.

Nous détaillons tout d'abord les interfaces graphiques développées pour chaque plateforme, ainsi que les performances obtenues en termes de temps d'inférence, de précision et de fluidité. Nous examinons ensuite la robustesse du système face à différentes conditions météorologiques (lumière directe, nuit, salissures, bruit, etc.), et nous analysons les avantages conceptuels et techniques de notre approche par rapport aux méthodes traditionnelles basées sur l'OCR. Enfin, nous discutons les limites actuelles de notre solution et les perspectives d'amélioration, tant sur le plan logiciel que matériel.

## 4.2 Présentation des interfaces et résultats de détection

### 4.2.1 Interface PC et résultats

#### Présentation de ressources PC

L'ordinateur que on'à utiliser est un HP ProBook 640 G4 qu'aux les performances suivant :

- **CPU** : Intel Core i5-8250U, quad-core 64 bits, MT MCP, 1.60 GHz ×8.
- **GPU** : Intel Corporation UHD Graphics 620.
- **RAM** : 8 Go.
- **Système** : Ubuntu 22.04.5 LTS.
- **Caméra** : HP HD Camera, 640×480, jusqu'à 30 fps

### Présentation de l'interface graphique PC

Dans notre interface, nous avons une zone dédiée à l'affichage de la caméra, un bouton permettant de démarrer ou d'arrêter la capture vidéo, ainsi qu'une liste affichant les informations relatives à la plaque d'immatriculation et les résultats de l'inférence sur une image (frame), comme illustré dans la figure 4.1 :

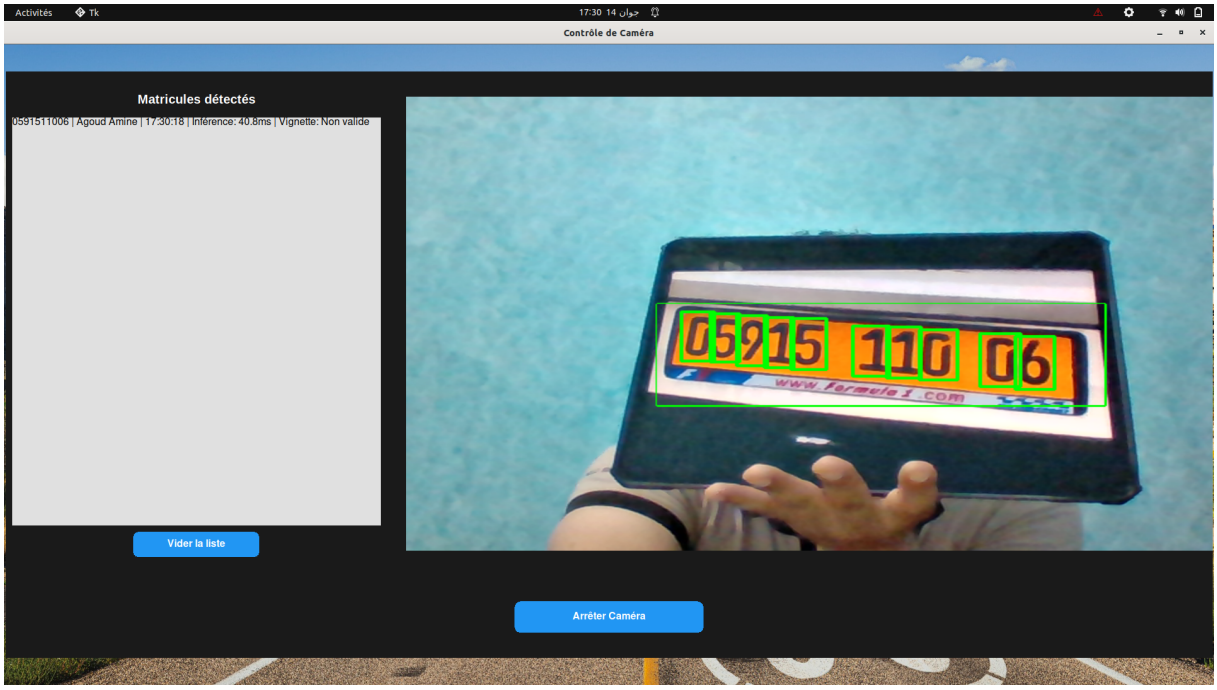


FIGURE 4.1 – Interface logicielle du PC

### Démonstration et Visualisation de la détection de plaques en temps réel

Lors du test de la détection, nous avons d'abord enregistré des plaques d'immatriculation dans notre base de données Firebase Realtime, puis nous avons testé la détection sur ces mêmes plaques, comme illustré dans les figures suivantes :

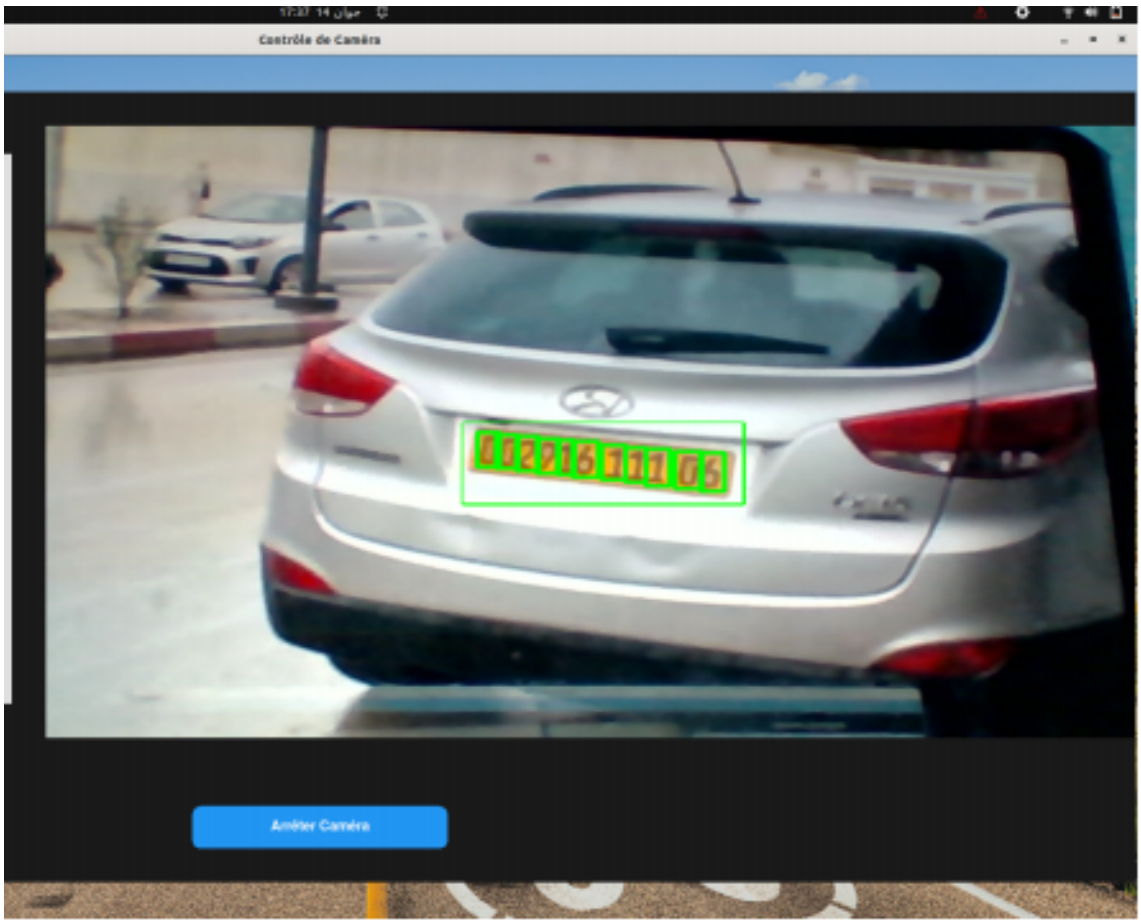


FIGURE 4.2 – Capture 1 – Résultat de la détection sur PC

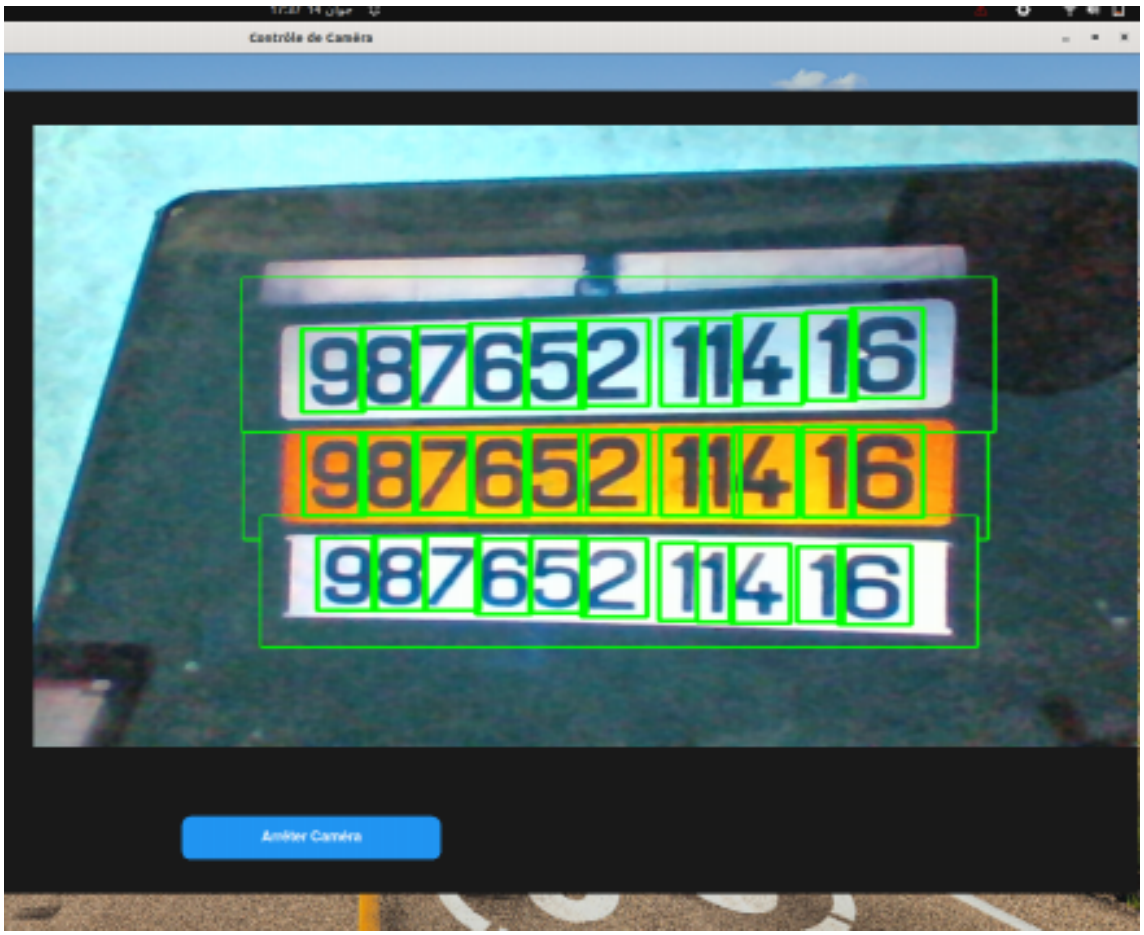


FIGURE 4.3 – Capture 2 – Résultat de la détection sur PC

Sur notre ordinateur, la résolution de la caméra est de 640×480 pixels, ce qui empêche la détection des plaques d'immatriculation éloignées en raison du manque de détails visuels.

### Affichage des informations de vignette depuis Firebase

Pour les plaques d'immatriculation présentes dans notre base de données, le modèle a réussi à les détecter, à vérifier le statut de la vignette correspondante, ainsi qu'à afficher le nom du propriétaire associé, comme illustré dans la figure 4.25 :

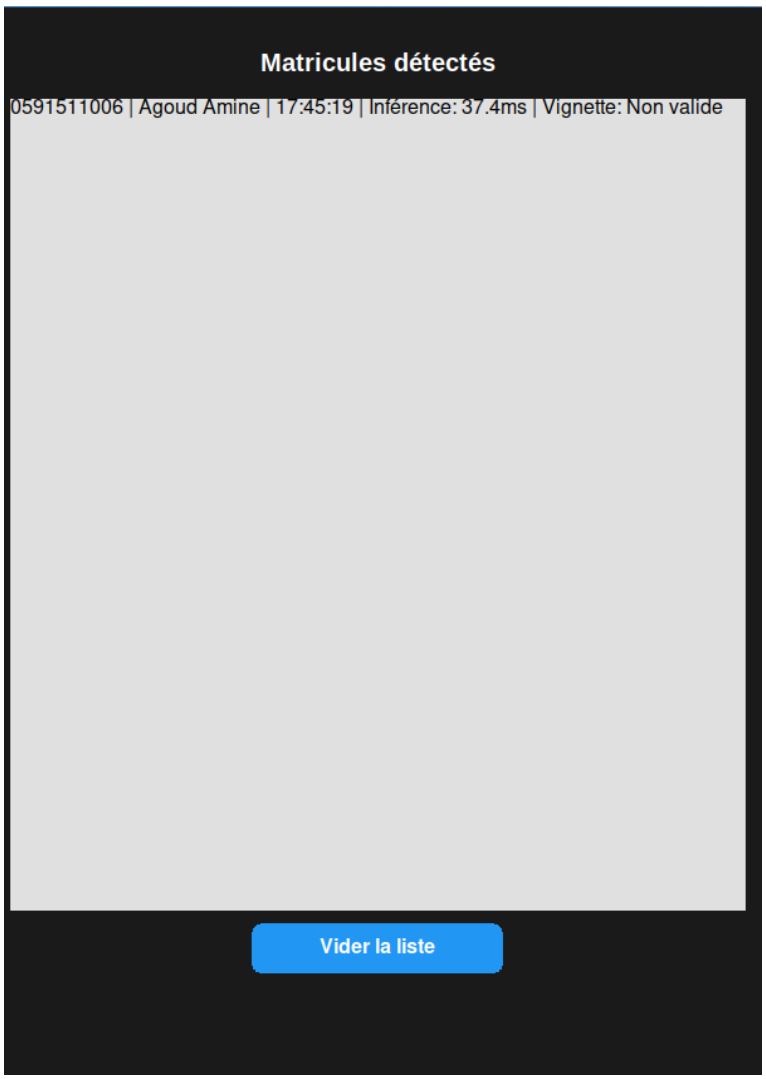


FIGURE 4.4 – Capture de l'affichage des informations de la plaque

**Temps d'inférence pour le pipeline (YOLOv5n) pour détection de plaques et (YOLOv8s) pour la reconnaissance de chiffres**

Le temps d'inférence du pipeline sur notre ordinateur est de 37 ms, ce qui signifie que l'interface peut traiter environ 27 à 28 images par seconde.

### 4.2.2 Interface Raspberry Pi et résultats

#### Présentation de l'interface Raspberry Pi

L'interface graphique que nous avons développée sur le Raspberry Pi est identique à celle utilisée sur PC. Elle offre la même organisation des éléments, les mêmes fonctionnalités et une expérience utilisateur similaire. La seule différence réside dans les bibliothèques utilisées pour la gestion de la caméra, qui diffèrent en raison de l'environnement matériel et système du Raspberry Pi. Cette adaptation a été nécessaire pour assurer la compatibilité avec le module caméra du Raspberry Pi tout en conservant une interface cohérente entre les deux plateformes.

**Démonstration et Visualisation de la détection de plaques en temps réel**

Dans cette expérimentation, nous avons placé notre Raspberry Pi en bord de route. Le système a réussi à détecter tous les véhicules passant à proximité, comme illustré dans les figures suivantes. Les plaques d'immatriculation détectées ne figurent pas dans notre base de données, ce qui explique pourquoi aucune information ne s'affiche dans la liste, il s'agit de véhicules réels circulant sur la voie publique.

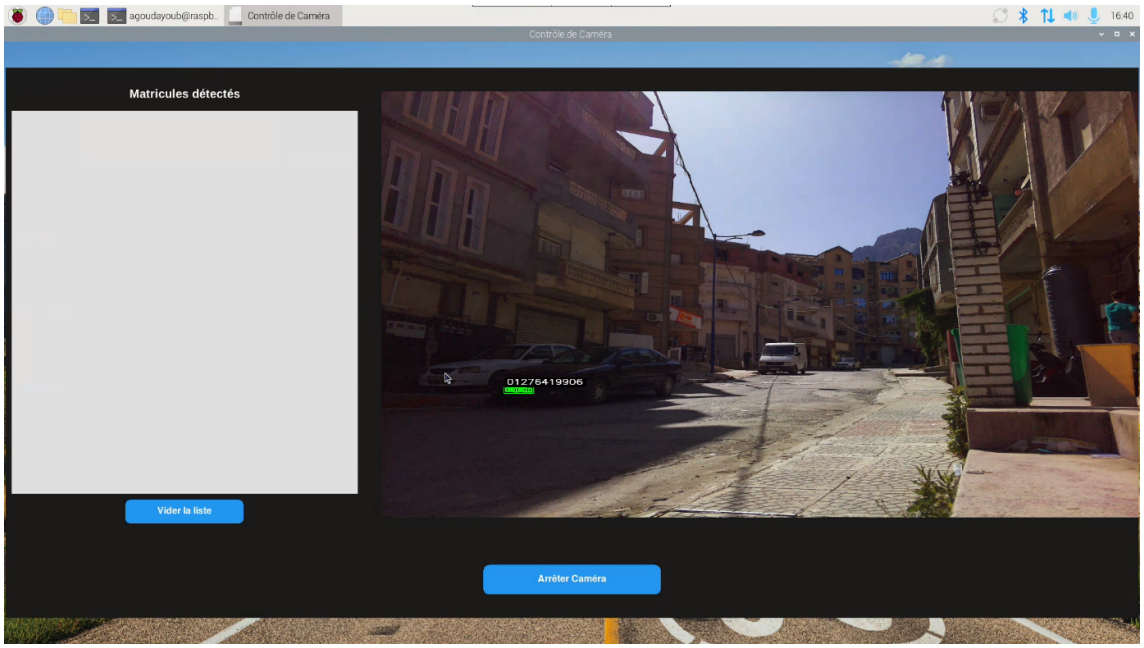


FIGURE 4.5 – Capture 1 – Résultat de la détection sur Raspberry Pi

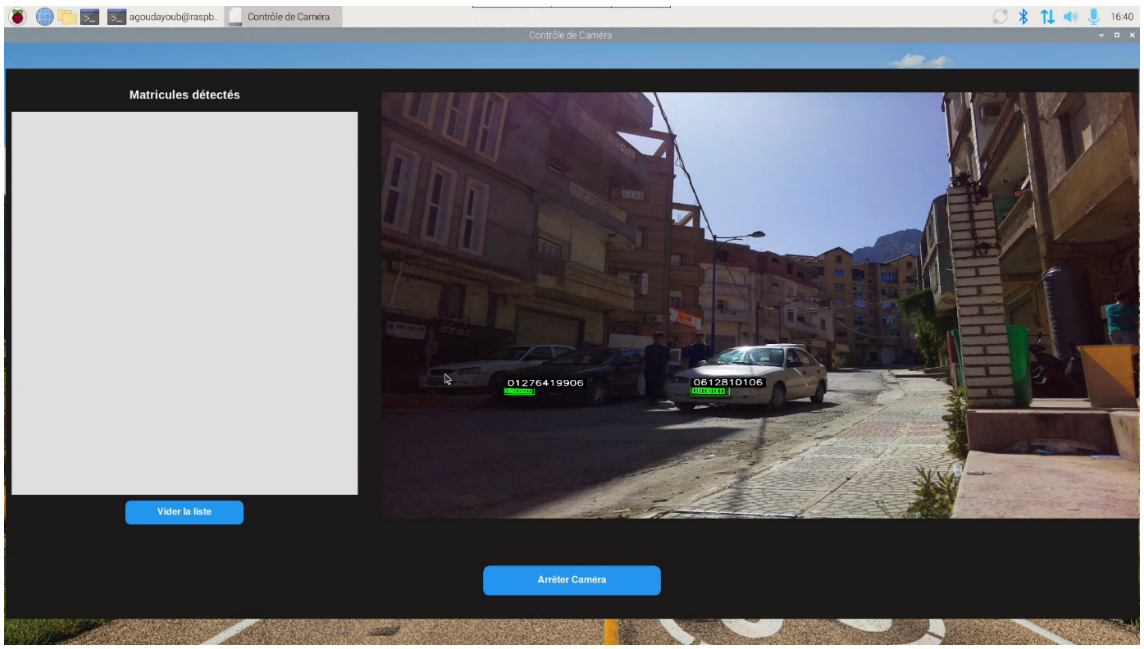


FIGURE 4.6 – Capture 2 – Résultat de la détection sur Raspberry Pi

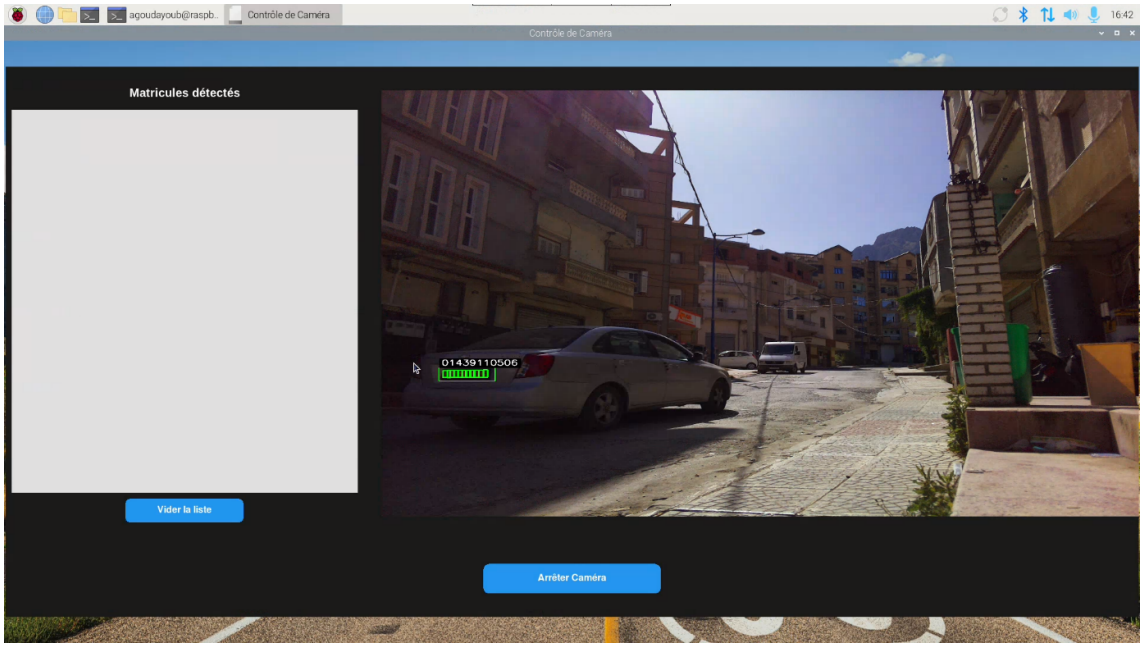


FIGURE 4.7 – Capture 3 – Résultat de la détection sur Raspberry Pi

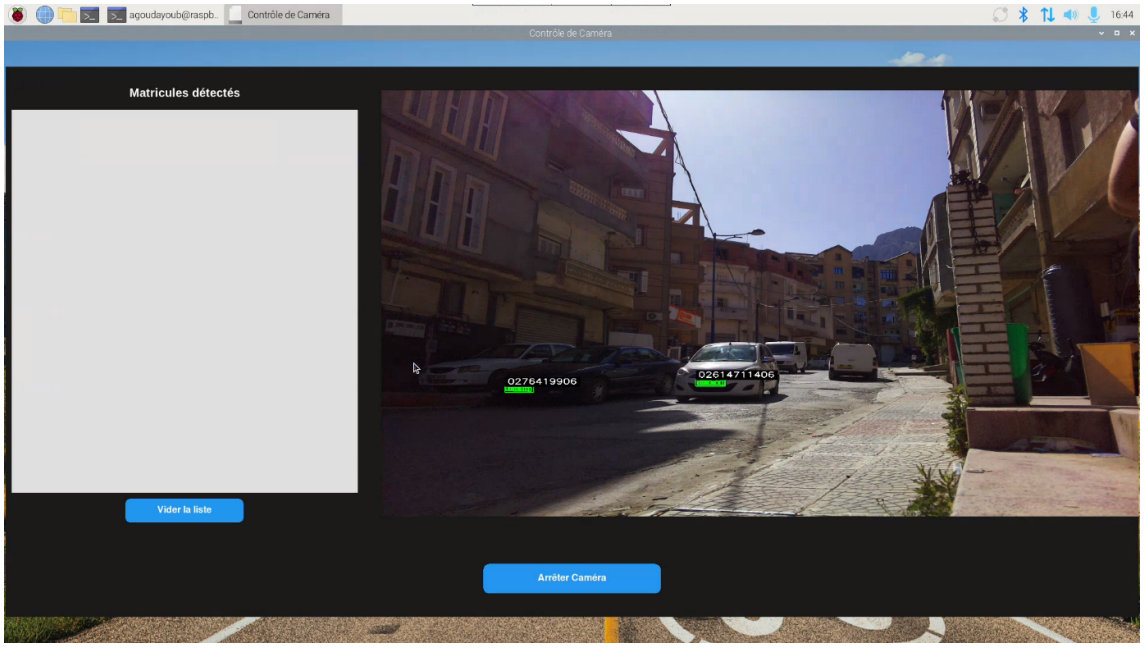


FIGURE 4.8 – Capture 4 – Résultat de la détection sur Raspberry Pi

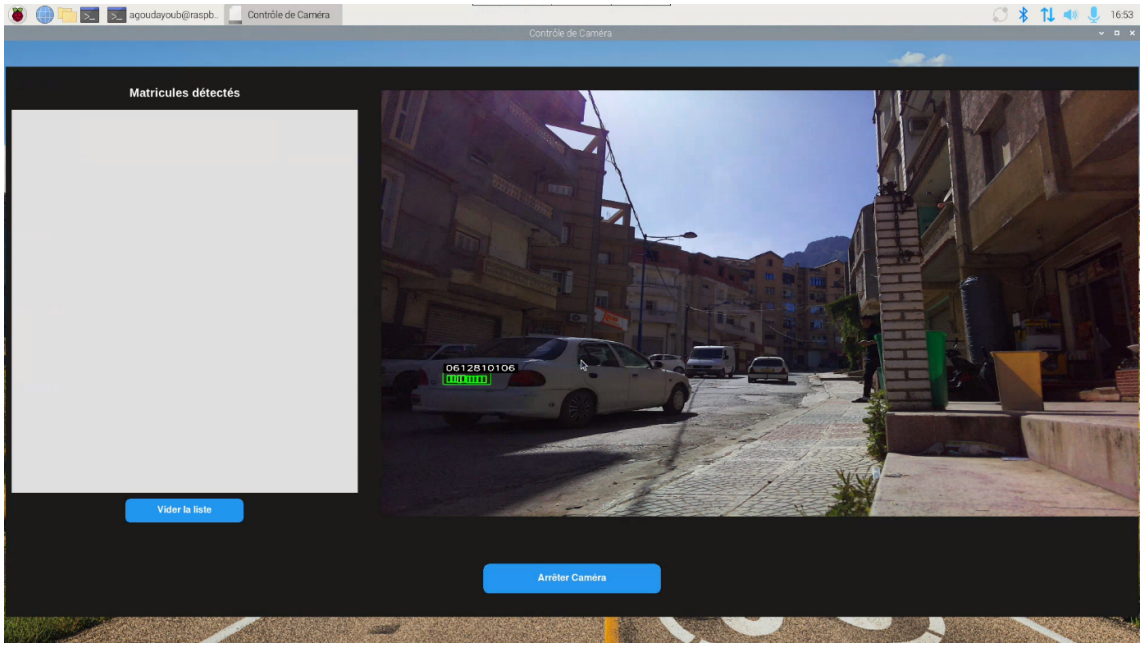


FIGURE 4.9 – Capture 5 – Résultat de la détection sur Raspberry Pi

Sur notre Raspberry Pi, la caméra offre une résolution de 3280×2464 pixels, ce qui facilite la détection des plaques d'immatriculation éloignées grâce à une meilleure précision des détails visuels.

**Temps d'inférence pour le pipiline (YOLOv5n) pour détection de plaques et (YOLOv8s) pour la reconnaissance de chiffres**

Le temps d'inférence pour le pipeline sur notre Raspberry Pi est de 700 ms ce qui veut dire que l'interface peut analyser 1 à 2 frime par second.

### 4.2.3 Application Android et résultats

**Présentation de ressources téléphone**

Nous avons utilisé un Samsung Galaxy Note 9 qui a les performances suivantes :

- **CPU** : Qualcomm Snapdragon 845 (10 nm), Octa-core (4×2.8 GHz + 4×1.7 GHz)
- **GPU** : Adreno 630 (Snapdragon)
- **RAM** : 6 Go
- **Système** : Android 12
- **Caméra** : 12 MP, 4K à 60 fps, ralenti jusqu'à 960 fps en HD

**Présentation des différentes activités de l'application**

1. **Écran d'accueil**

L'interface d'accueil est très simple, car notre priorité actuelle est centrée sur le bon fonctionnement du système. Elle contient deux boutons, l'un permettant d'accéder à l'interface d'authentification, et l'autre redirigeant vers l'interface de détection, comme illustré dans la figure 4.10 :


FIGURE 4.10 – Écran d'accueil de l'application sur téléphone

2. **Écran d'authentification** L'interface d'authentification, illustrée dans la figure ci-dessous, permet à l'utilisateur de se connecter à son compte, préalablement créé par nous ou par les administrateurs via le service Firebase authentication utilisant l'email et le mot de passe. L'utilisateur peut choisir de rester connecté afin de sauvegarder sa session, ou cliquer sur 'Mot de passe oublié' pour recevoir un lien de réinitialisation envoyé par la base d'authentification. Une fois connecté, l'utilisateur est redirigé vers l'interface de son profil.

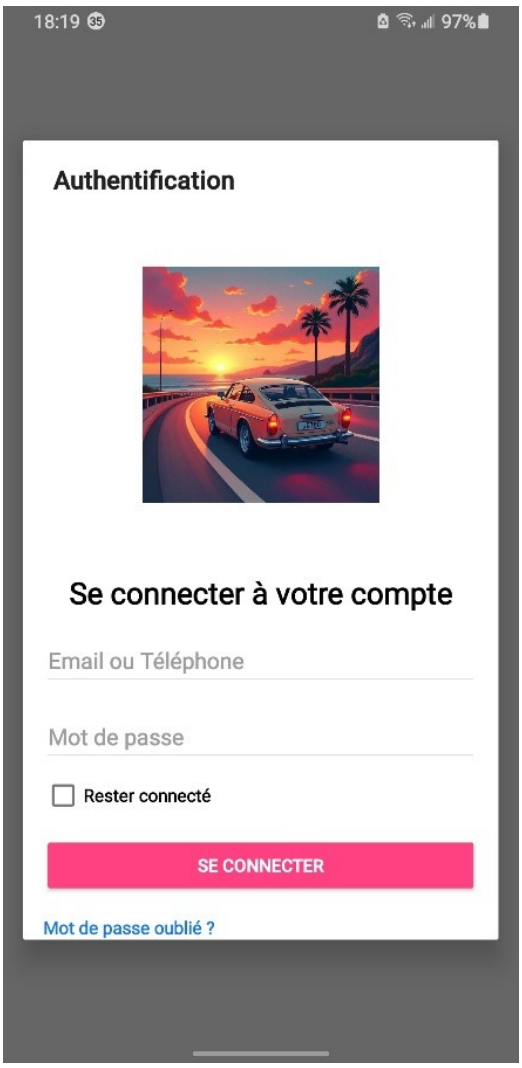


FIGURE 4.11 – Interface d'authentification de l'application sur téléphone

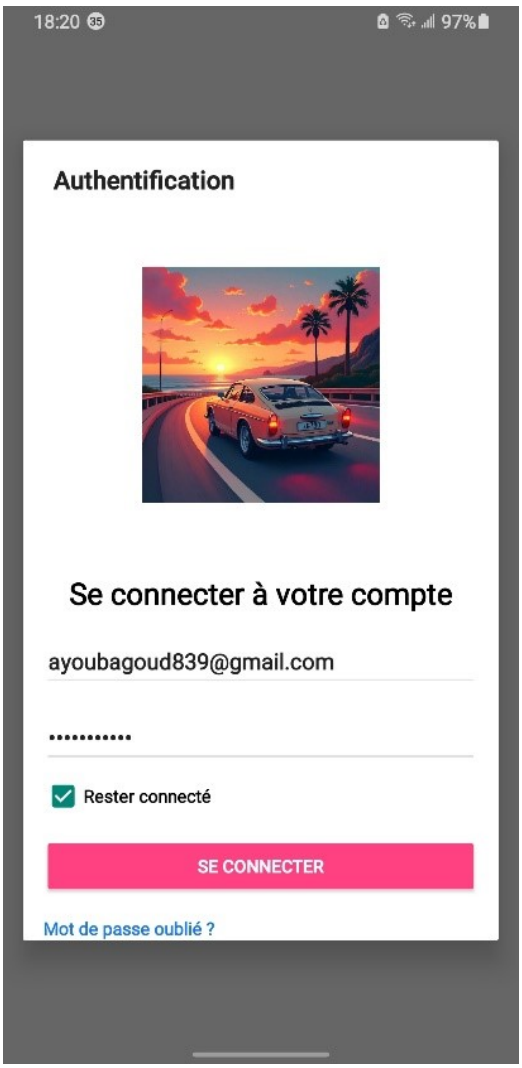


FIGURE 4.12 – Connexion réussie et sauvegarde de la session dans l'interface d'authentification de l'application sur téléphone

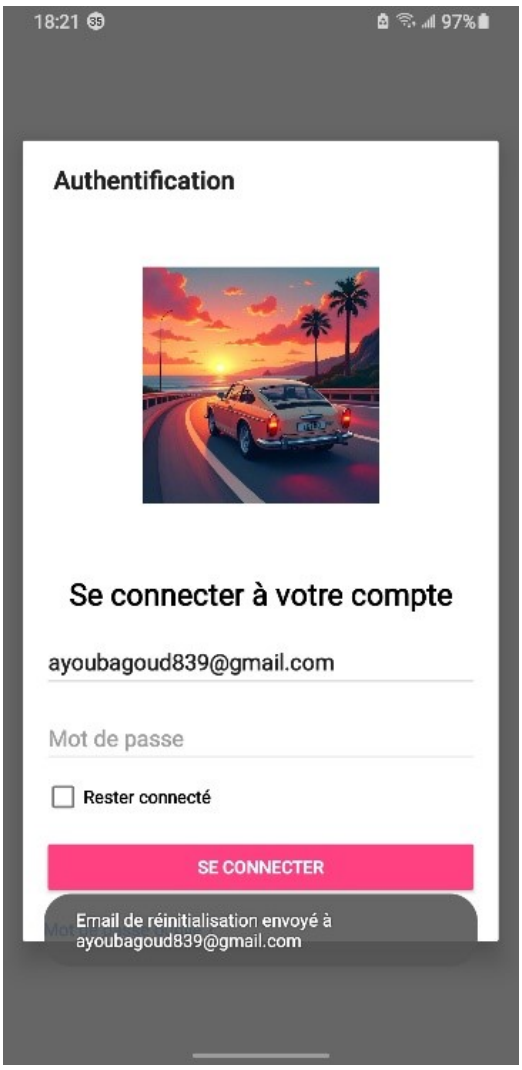


FIGURE 4.13 – Envoi d'un lien de réinitialisation du mot de passe par e-mail

3. **Profil utilisateur et informations de vignette** L'interface du profil affiche la plaque d'immatriculation associée à l'utilisateur ainsi que la date d'expiration de la vignette. Lorsque cette date est dépassée, une notification est automatiquement envoyée à l'utilisateur afin de le prévenir.


FIGURE 4.14 – Interface profile de l'application sur téléphone

4. **Interface de détection en temps réel** Voici l'interface de détection, conçue pour tester notre approche de détection directement sur un téléphone, en utilisant Java et TensorFlow :

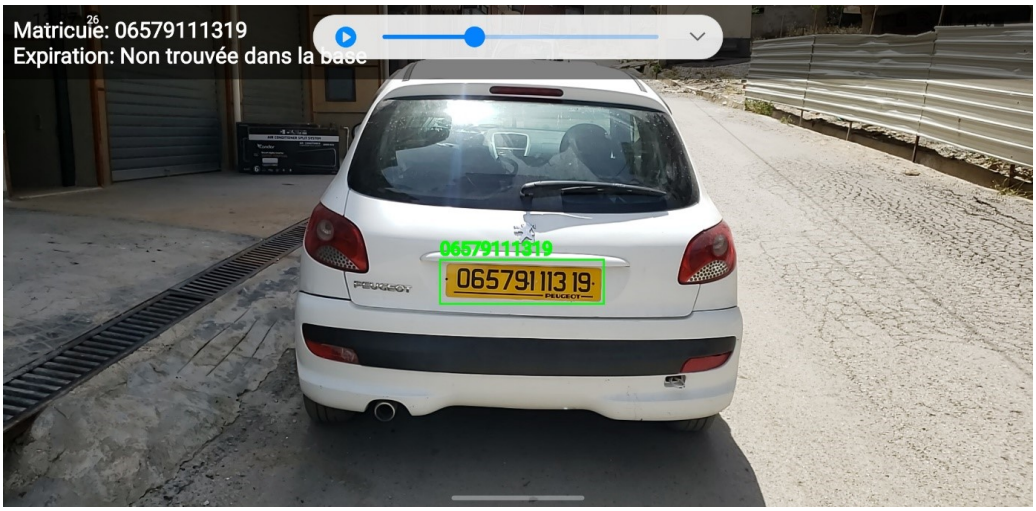


FIGURE 4.15 – Interface de détection de l'application sur téléphone

**Démonstration et Visualisation de la détection de plaques en temps réel sur smartphone**

Comme illustré dans les figures, la détection et la reconnaissance de la plaque d'immatriculation ont été réussies avec succès.


FIGURE 4.16 – Détection de la plaque dans l'interface de détection de l'application sur téléphone


FIGURE 4.17 – Détection de la plaque dans l'interface de détection de l'application sur téléphone


FIGURE 4.18 – Capture 9 – Détection de la plaque dans l'interface de détection de l'application sur téléphone

Concernant le statut de la vignette, si la plaque d'immatriculation est présente dans notre base de données, la date d'expiration de la vignette sera affichée, comme montré dans la figure suivante :


FIGURE 4.19 – Détection de la plaque dans l'interface de détection de l'application sur téléphone

**Temps d'inférence pour le pipiline (YOLOv5n) pour détection de plaques et (YOLOv8s) pour la reconnaissance de chiffres**

Le temps d'inférence pour le pipeline sur notre smartphone et de 400 ms ce qui veut dire que l'interface peut analyser 2 a 3 frime par second.

## 4.3 Performances dans différentes conditions météorologiques

Notre approche présente des performances acceptables dans différentes conditions météorologiques :

1. **Face au soleil**

Le système fonctionne correctement même lorsqu'il est orienté face au soleil, comme le montre la figure. La caméra parvient à capturer des images nettes, ce qui facilite la détection et la reconnaissance des plaques d'immatriculation. Toutefois, des reflets lumineux sur les carrosseries, la caméra ou directement sur les plaques peuvent parfois perturber légèrement la précision de la reconnaissance.

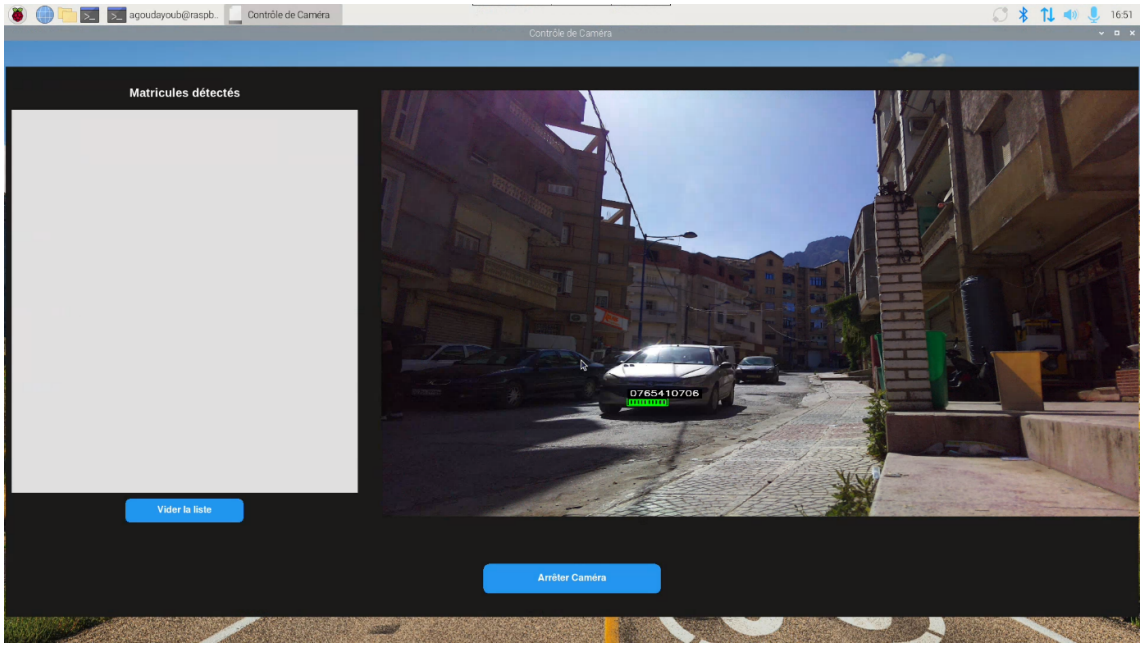


FIGURE 4.20 – Capture de detection en Contre soleil

2. **Plein soleil**

Dans cette conditions, notre système maintient une détection raisonnable.

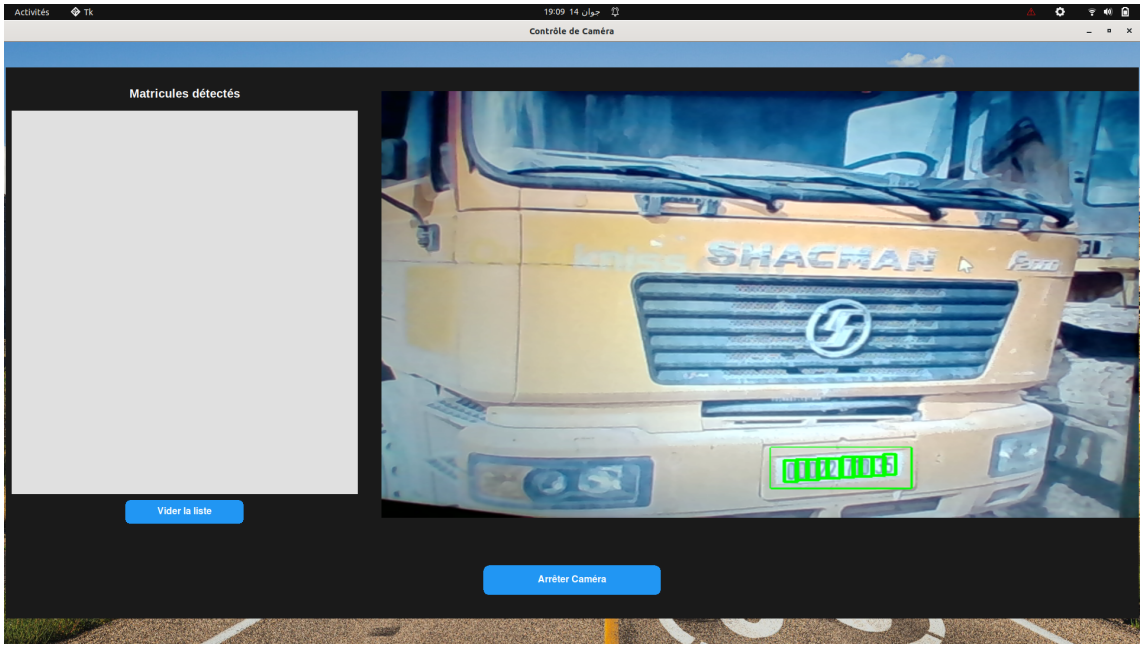


FIGURE 4.21 – Capture de detection en plein soleil

3. **Nuit avec éclairage**

   Comme le montre la figure 4.22, la détection a été effectuée avec succès de nuit grâce à un bon éclairage

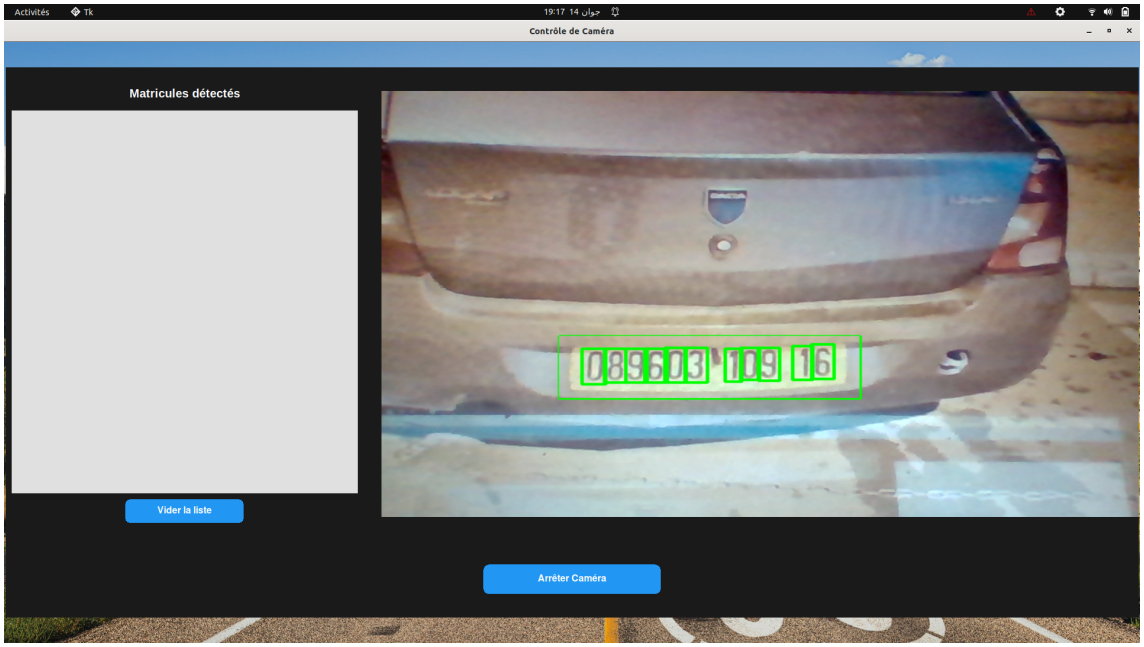


FIGURE 4.22 – Capture de détection dans la nuit

4. **Plaque d'immatriculation sale**

   Lorsque la plaque est partiellement couverte de poussière, de boue ou d'autres salissures, la détection devient plus difficile. Notre approche parvient tout de même à identifier certaines plaques si les chiffres restent partiellement visibles (figure 4.23).

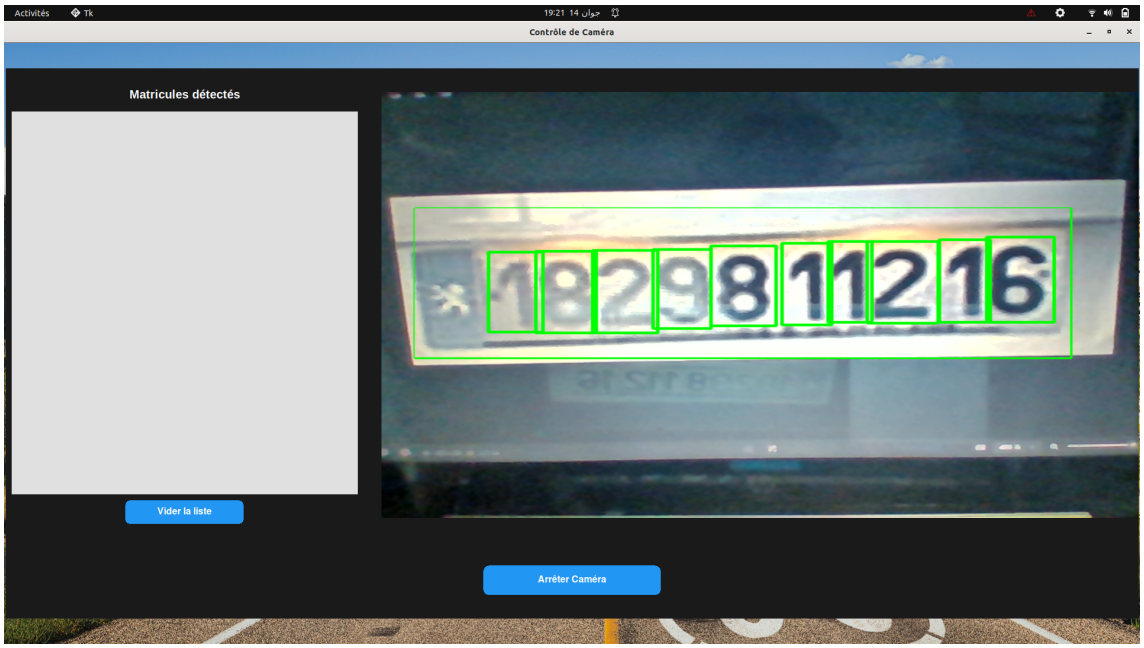


FIGURE 4.23 – Capture de detection de plaque d'immatriculation sale

5. **Bruit et de flou**

   Lorsque l'image contient du bruit, dû par exemple à une faible luminosité, à un capteur de mauvaise qualité ou à une compression excessive, les performances de détection et de reconnaissance peuvent être affectées. Malgré cela, notre système montre une certaine robustesse face au bruit modéré (figure 4.24).

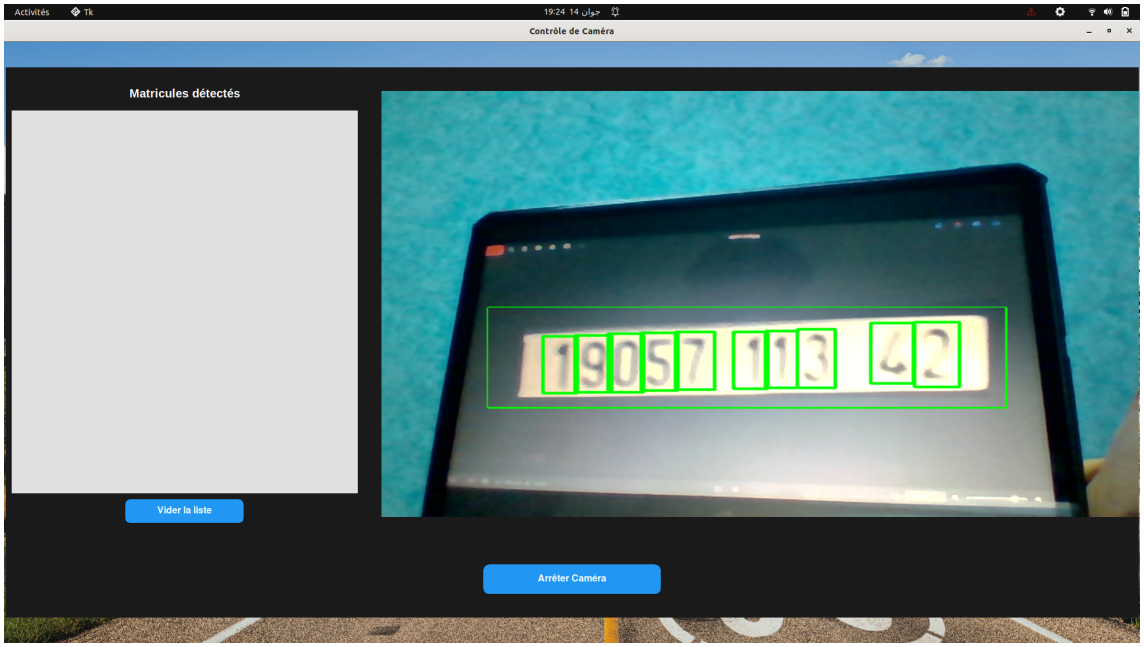


FIGURE 4.24 – Capture de la détection en présence de bruit et de flou

## 4.4 Positionnement de l'approche proposée

### 4.4.1 Avantages du pipeline à deux YOLO

Notre approche basée sur un pipeline à deux modèles YOLO présente des avantages significatifs par rapport aux méthodes traditionnelles de détection et reconnaissance de plaques d'immatriculation. Cette architecture innovante repose sur une philosophie d'optimisation "tâche par tâche", où chaque modèle est spécialisé pour une fonction précise.

#### Optimisation tâche par tâche

Le premier avantage majeur de notre pipeline réside dans sa conception modulaire qui permet une optimisation fine de chaque étape du processus :

1. **Spécialisation des modèles**

   Chaque modèle YOLO est entraîné et optimisé pour une tâche spécifique :

1. YOLOv5n pour la détection des plaques d'immatriculation
2. YOLOv8s pour la reconnaissance des chiffres

2. **Équilibre performance/précision adapté à chaque tâche**

- Pour la détection des plaques, nous avons privilégié la vitesse avec YOLOv5n en format float16, permettant une détection rapide même sur des appareils à ressources limitées.
- Pour la reconnaissance des chiffres, nous avons favorisé la précision avec YOLOv8s en format float32, essentielle pour distinguer correctement les 10 classes de chiffres.

Cette approche "tâche par tâche" permet d'atteindre un équilibre optimal entre vitesse d'exécution et précision, comme le démontrent nos résultats expérimentaux. Par exemple, sur Raspberry Pi 4, notre système maintient une détection fluide à environ 2 FPS tout en conservant une précision élevée dans la reconnaissance des numéros de plaque.

**Réduction de la complexité par rapport aux approches séquentielles classiques**

Les approches traditionnelles de détection et de reconnaissance de plaques d'immatriculation suivent généralement un pipeline complexe composé de trois étapes ou plus, comme nous l'avons vu dans les travaux connexes présentés au chapitre 2 :

a) Détection du véhicule
b) Localisation de la plaque d'immatriculation
c) Segmentation des caractères
d) Reconnaissance des caractères par OCR

Notre pipeline à deux YOLO simplifie considérablement ce processus :

a) Détection directe de la plaque d'immatriculation (YOLOv5n)
b) Reconnaissance directe des chiffres (YOLOv8s)

Cette simplification présente plusieurs avantages :

- **Réduction des erreurs cumulatives** : Chaque étape d'un pipeline traditionnel introduit des erreurs qui se propagent et s'amplifient. Notre approche réduit ces points de défaillance potentiels.
- **Diminution de la latence globale** : L'élimination des étapes intermédiaires réduit le temps de traitement total, permettant une détection en temps réel même sur des appareils à ressources limitées.
- **Simplification de l'architecture logicielle** : Moins de composants signifie une maintenance plus simple, moins de dépendances et une meilleure portabilité.

- **Économie de ressources computationnelles** : L'absence d'étape de détection de véhicule et de segmentation des caractères réduit significativement la charge de calcul.

  Notre approche élimine notamment l'étape coûteuse de détection préalable du véhicule, partant du principe qu'une plaque d'immatriculation peut être détectée directement dans l'image, indépendamment de la reconnaissance du véhicule luimême. Cette simplification permet d'économiser des ressources précieuses tout en maintenant d'excellentes performances.

### 4.4.2 Comparaison conceptuelle avec les méthodes basées sur l'OCR

Les méthodes traditionnelles de reconnaissance de plaques d'immatriculation, ainsi que celles présentées dans les travaux connexes du chapitre 2, reposent généralement sur des techniques d'OCR (Optical Character Recognition) appliquées après une étape de segmentation des caractères. En revanche, notre approche basée sur YOLO présente plusieurs avantages conceptuels par rapport à ces méthodes classiques.

#### Robustesse aux polices variées

Les systèmes OCR traditionnels sont souvent sensibles aux variations de polices et de styles de caractères :

- **Limitations des OCR classiques** : Les moteurs de reconnaissance optique de caractères (OCR) traditionnels, tels que Tesseract ou EasyOCR, sont généralement entraînés sur des textes imprimés avec des polices standardisées. Par conséquent, ils rencontrent souvent des difficultés dans des contextes complexes tels que :
  - les polices inhabituelles ou personnalisées (ex. : plaques d'immatriculation régionales),
  - les déformations optiques dues à l'angle de prise de vue, aux ombres ou au flou,
  - les caractères stylisés ou partiellement effacés.

  Dans le cadre de notre projet, ces limitations sont apparues clairement lors de l'utilisation d'OCR classiques sur des images de plaques réelles. Les erreurs fréquentes de lecture, notamment dues aux résolutions variables et aux contrastes faibles, ont motivé le recours à une alternative plus robuste[60].
- **Approche par apprentissage profond** : Afin de dépasser les limites des OCR conventionnels, nous avons adopté une approche par apprentissage profond, en entraînant un modèle YOLOv8s pour la détection directe des chiffres présents sur les plaques d'immatriculation.

  Notre modèle a été entraîné sur un ensemble de 34 466 images réelles, couvrant une large diversité de styles d'écriture, d'angles de vue, de résolutions, et de conditions de luminosité. Contraire-

ment aux OCR, cette méthode ne repose pas sur la reconnaissance d'une police spécifique, mais apprend à localiser et classer visuellement chaque chiffre dans son contexte naturel.

Cette approche rend notre système plus résilient aux variations graphiques, y compris celles causées par l'usure des plaques, les effets de perspective ou les styles propres aux plaques d'immatriculation algériennes[61].

- **Reconnaissance contextuelle** : Contrairement aux OCR qui traitent chaque caractère isolément après segmentation, notre approche considère le contexte spatial des chiffres au sein de la plaque, améliorant la robustesse de la reconnaissance.

  Cette robustesse est particulièrement importante dans le contexte des plaques d'immatriculation algériennes, qui peuvent présenter des variations de police et de style selon leur année d'émission et leur catégorie.

#### Meilleure gestion des conditions difficiles

Les conditions réelles de capture d'image présentent de nombreux défis que notre approche gère mieux que les méthodes OCR traditionnelles :

- **Résistance aux problèmes d'éclairage** : Les modèles YOLO sont entraînés sur des images présentant diverses conditions d'éclairage, tandis que les OCR traditionnels nécessitent souvent une normalisation préalable de l'image[62].
- **Tolérance aux occlusions partielles** : Notre approche peut reconnaître des chiffres partiellement occultés ou endommagés, là où la segmentation préalable à l'OCR échouerait[62].
- **Adaptation aux angles de vue non optimaux** : Les modèles YOLO peuvent détecter et reconnaître des chiffres même lorsque la plaque est capturée sous un angle non idéal, contrairement aux OCR qui exigent généralement une image frontale et bien alignée.
- **Robustesse au bruit et aux artefacts** : Les méthodes de segmentation préalables à l'OCR sont particulièrement sensibles au bruit et aux artefacts d'image,problèmes que notre approche par détection directe contourne efficacement.

  Des tests comparatifs ont montré que notre système maintient un taux de reconnaissance acceptable même dans des conditions où les approches OCR traditionnelles échouent complètement, notamment en présence de reflets, d'ombres ou de salissures sur les plaques.

#### Élimination des erreurs de segmentation

L'un des points faibles majeurs des approches OCR traditionnelles réside dans l'étape de segmentation des caractères[62] :

- **Problématique de la segmentation** : La segmentation incorrecte des caractères est une source majeure d'erreurs dans les systèmes OCR traditionnels, particulièrement lorsque les caractères

sont proches ou connectés.

- **Approche holistique** : Notre modèle YOLOv8s détecte directement chaque chiffre sans nécessiter de segmentation préalable de l'image entière de la plaque, éliminant ainsi cette source d'erreur.
- **Détection simultanée de position et classe** : YOLO identifie simultanément la position et la classe (valeur) de chaque chiffre, ce qui permet une meilleure gestion des cas où les caractères se chevauchent ou sont mal alignés.

Cette élimination des erreurs de segmentation contribue significativement à la robustesse globale de notre système dans des conditions réelles d'utilisation.

### 4.4.3 Intérêt de la solution multi-plateforme développée

Notre système de détection et reconnaissance de plaques d'immatriculation a été conçu dès le départ comme une solution multi-plateforme, capable de fonctionner sur PC, Raspberry Pi 4 et appareils Android. Cette approche présente de nombreux avantages stratégiques et pratiques.

#### Flexibilité de déploiement

La nature multi-plateforme de notre solution offre une flexibilité sans précédent pour le déploiement :

- **Adaptation aux différents contextes d'utilisation** : Notre système peut être déployé dans divers scénarios, des installations fixes (PC) aux solutions embarquées (Raspberry Pi) et mobiles (Android).
- **Scalabilité** : La même base de code et les mêmes modèles peuvent être utilisés à différentes échelles, depuis des installations individuelles jusqu'à des réseaux de surveillance étendus.
- **Continuité de service** : La disponibilité sur plusieurs plateformes assure une continuité de service, même en cas de défaillance d'un dispositif particulier.

Cette flexibilité permet d'adapter la solution aux besoins spécifiques de chaque contexte d'utilisation, qu'il s'agisse de contrôle d'accès à un parking, de surveillance routière ou de vérification mobile des vignettes.

#### Optimisations spécifiques par plateforme

Notre approche multi-plateforme ne se contente pas d'une simple portabilité du code, mais intègre des optimisations spécifiques pour chaque environnement :

- **PC** : Exploitation des capacités de calcul supérieures pour un traitement haute résolution et multi-flux ,Utilisation d'OpenVINO pour l'accélération CPU/GPU, Interface graphique riche avec visualisation avancée, Capacité de traitement de flux vidéo haute vitesse.

- **Raspberry Pi 4** : Adaptations pour l'embarqué avec ressources limitées Utilisation optimisée de Picamera2 pour l'accès direct au matériel Stratégie de double résolution (capture haute résolution / traitement redimensionné), Traitement périodique pour économiser les ressources CPU.
- **Android** : L'application a été conçue pour fonctionner efficacement sur des appareils mobiles, en tenant compte des contraintes de performance et d'autonomie.

Elle utilise CameraX pour une gestion optimisée de la caméra, garantissant une meilleure stabilité, une compatibilité étendue avec les appareils Android et une consommation réduite de ressources. Les deux modèles de détection sont intégrés au format TensorFlow Lite, assurant un traitement rapide directement sur l'appareil sans connexion réseau. L'interface utilisateur est adaptée aux écrans tactiles, avec une navigation fluide et intuitive. De plus, une gestion intelligente des ressources (CPU, mémoire, caméra) a été mise en place pour limiter la consommation énergétique et préserver l'autonomie de la batterie.

Ces optimisations permettent d'obtenir les meilleures performances possibles sur chaque plateforme tout en maintenant une cohérence fonctionnelle.

**Écosystème complet et interconnecté**

La nature multi-plateforme de notre solution permet de créer un écosystème complet et interconnecté :

- **Base de données centralisée** : Toutes les plateformes accèdent à la même base de données Firebase, assurant une cohérence des informations.
- **Complémentarité des usages** :

  PC : Surveillance et administration centralisée

  Raspberry Pi : Points de contrôle fixes à faible coût

  Android : Vérifications mobiles et notifications utilisateur
- **Partage des modèles et des améliorations** : Les améliorations apportées aux modèles bénéficient immédiatement à toutes les plateformes.

Cette approche écosystémique offre une solution complète qui répond à l'ensemble des besoins liés à la gestion des plaques d'immatriculation et des vignettes, depuis la détection jusqu'à la notification des utilisateurs.

#### Avantages économiques et pratiques

La solution multi-plateforme présente également des avantages économiques et pratiques significatifs :

- **Réduction des coûts de développement et de maintenance** : Une base de code commune réduit les efforts de développement et de maintenance.
- **Déploiement progressif et hybride** : Possibilité de commencer avec une solution à faible coût (Raspberry Pi) et d'évoluer vers des installations plus sophistiquées (PC) selon les besoins.
- **Accessibilité** : L'application Android rend le système accessible à tous les utilisateurs de smartphones, sans nécessiter d'équipement spécialisé.
- **Évolutivité technologique** : L'architecture modulaire facilite l'intégration de nouvelles technologies et l'amélioration continue des performances.

Ces avantages font de notre solution multi-plateforme une approche particulièrement pertinente et pérenne pour la détection et la vérification des plaques d'immatriculation.

## 4.5 Limites actuelles et perspectives d'amélioration

### 4.5.1 Limites identifiées

#### Limitations de performance sur Raspberry Pi

L'une des principales limites observées lors de l'exécution du système sur le Raspberry Pi est la faible vitesse d'inférence, qui se situe entre 1 à 2 images par seconde. Cette cadence réduit fortement les chances de détecter correctement un véhicule en mouvement. En effet, si une voiture passe rapidement ou même à vitesse modérée il se peut qu'elle n'apparaisse que sur une ou deux images capturées.

Si, dans ces images, la plaque d'immatriculation est floue, sale, bruitée ou partiellement visible, le système risque de ne pas détecter tous les chiffres, ou de mal les reconnaître. Étant donné qu'il n'y a pas d'autres frames disponibles pour cette même voiture, aucune nouvelle tentative de reconnaissance n'est possible. Cela entraîne donc des cas de non-détection ou de reconnaissance incorrecte, surtout pour les véhicules avec vitesse élever.

#### Cas difficiles pour la détection

Notre système actuel est conçu pour traiter uniquement les plaques d'immatriculation à une seule ligne. En effet, les chiffres détectés sont triés selon leur position horizontale (de gauche à droite), sans prendre en compte leur position verticale. Ainsi, dans le cas de plaques à deux lignes où les

caractères sont répartis en deux rangées, le système ne parvient pas à reconstituer correctement le numéro complet.

Cette limite empêche la reconnaissance fiable de nombreux formats de plaques, et nécessite une amélioration future.

#### Précision sur certains types de plaques spécifiques

Dans notre approche, nous nous sommes exclusivement concentrés sur les plaques d'immatriculation algériennes standards, appartenant aux véhicules de particuliers. Le système n'a pas été entraîné ni optimisé pour reconnaître les plaques spécifiques telles que celles des forces de l'ordre (police, gendarmerie), des véhicules diplomatiques, ou encore des véhicules étrangers, qui présentent souvent des formats différents (couleurs, dispositions, polices, etc.).

De plus, inclure ces types de plaques représenterait une complexité supplémentaire pour le modèle, en raison de leur diversité et de leur absence de suivi dans notre base de données. Leur détection serait donc inutile dans le contexte de notre application, car même en cas de reconnaissance correcte, aucune information associée ne pourrait être affichée. Cela entraînerait une perte de temps de traitement et risquerait de réduire l'efficacité globale du système.

### 4.5.2 Améliorations matérielles

#### Intégration d'accélérateurs pour Raspberry Pi

L'un des principaux goulots d'étranglement de notre système est le temps d'inférence sur Raspberry Pi, limité de 1 à 2 images par seconde. Pour améliorer significativement les performances sans changer de plateforme, nous proposons l'intégration d'accélérateurs matériels conçus pour les environnements embarqués.

1. **Google Coral USB Accelerator (TPU)**

   a) **Quantification des modèles en INT8** :

   Le Coral USB Accelerator (figure4.25) est équipé du Edge TPU, un coprocesseur capable d'exécuter des modèles TensorFlow Lite quantifiés en INT8 à très haute vitesse, avec une faible consommation énergétique[52].


FIGURE 4.25 – Coral USB Accelerator[53]

Pour utiliser le TPU[52],le modèle doit être converti en TFLite puis quantifié en INT8. Cela peut être fait via TensorFlow Model Optimization Toolkit.

b) **Gains de performance attendus** :

Jusqu'à 20 FPS sur des modèles légers (comme MobileNet ou YOLOv5n). Latence inférieure à 10 ms par image.

Les performances annoncées (jusqu'à 20 FPS ou ¡10 ms d'inférence) concernent des modèles optimisés et exécutés seuls sur le Coral TPU. Dans notre pipeline, l'inférence inclut plusieurs étapes (détection de plaques, recadrage, reconnaissance de chiffres, traitement visuel et accès Firebase), ce qui explique une latence totale d'environ 700 ms par image sur Raspberry Pi. En intégrant un accélérateur comme le Coral USB pour la détection de plaques, nous estimons une réduction de cette latence globale à 100–200 ms, avec un gain de 3 à 4 fois plus rapide tout en conservant la précision.

2. **Hailo-8 AI Accelerator**

Le Hailo-8 USB3 AI Module est un accélérateur très puissant, capable de performances comparables à un GPU, mais optimisé pour embarquer.

FIGURE 4.26 – Coral USB Accelerator[53]

a) Hailo propose un **Hailo Model Compiler** qui accepte des modèles ONNX ou TFLite et effectue une quantification INT8 automatiquement.

b) Le modèle doit être **calibré avec un dataset** de validation pour optimiser la précision. Un "calibration dataset" est un petit échantillon représentatif de ton vrai dataset (par exemple, 500 ou 1000 images), utilisé pendant la quantification pour :

- Observer la plage réelle des valeurs dans les activations du réseau.
- Choisir les meilleurs seuils de quantification (min/max) pour chaque couche.
- Réduire les pertes de précision après quantification.

c) Gains de performance attendus[53]

Le Hailo-8 AI Accelerator permet, dans des conditions optimales, d'exécuter des modèles de détection tels que YOLOv5s ou MobileNet SSD à plus de 30 FPS, avec une latence inférieure à 5 ms par image, tout en consommant moins de 2,5 W. Dans notre cas, si nous utilisons Hailo uniquement pour la détection de plaques d'immatriculation (modèle YOLOv5 quantifié en INT8), nous estimons que la latence de cette étape peut être réduite à moins de 15 ms, contre 300 à 500 ms actuellement sur CPU. Cette intégration pourrait permettre de traiter 6 à 10 images par seconde dans notre pipeline complet, même avec le traitement supplémentaire de reconnaissance des chiffres et de communication avec Firebase. Cela représente un gain de performance de 2 à 3 fois, tout en maintenant une consommation énergétique très basse.

### 4.5.3 Migration vers NVIDIA Jetson

#### Présentation des cartes Jetson (Nano, Xavier NX, AGX Orin)

Les cartes Jetson de NVIDIA sont des plateformes embarquées puissantes spécialement conçues pour les applications d'intelligence artificielle en périphérie (edge AI). Elles embarquent un GPU NVIDIA compatible CUDA, ce qui les rend très efficaces pour les inférences de modèles lourds sans recourir à la quantification INT8[54].

| GPU | RAM | Modèle | Performances (AI) |
|---|---|---|---|
| Jetson Nano | 128-core Maxwell | 4 Go | 0.5 TOPS |
| Jetson Xavier NX | 384-core Volta + 48 Tensor | 8 Go | 21 TOPS |
| Jetson AGX Orin | 2048-core Ampere + 64 Tensor | 32–64 Go | 200–275 TOPS |

TABLE 4.1 – Caractéristiques techniques des cartes NVIDIA Jetson

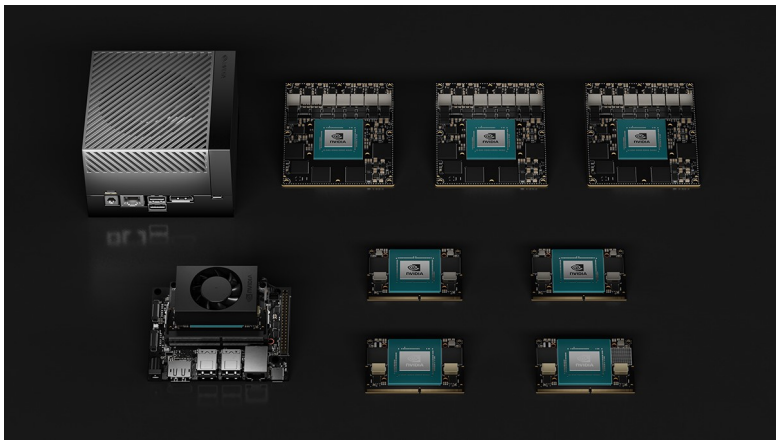

FIGURE 4.27 – Coral USB Accelerator [53]

#### Avantages de l'architecture CUDA pour l'inférence

L'architecture CUDA (Compute Unified Device Architecture) permet l'exécution hautement parallèle de tâches sur le GPU. Cela offre plusieurs avantages[55][56] :

- Accélération de l'inférence : les modèles peuvent être exécutés directement en FP32/FP16 avec des performances élevées.
- Prise en charge native de PyTorch et TensorFlow GPU : pas besoin de convertir en TFLite/ONNX.
- Interopérabilité avec OpenCV GPU, TensorRT, DeepStream, etc. un modèle YOLOv5 en .pt peut être exécuté sans conversion, simplement avec torch.cuda.

**Estimation des performances avec modèles non quantifiés (.pt)**

Sur les plateformes Jetson, il est possible d'exécuter directement les modèles PyTorch .pt ou ONNX .onnx, sans quantification[57][58] :

| Modèle | Type | Format | FPS approximatif sur Jetson NX |
|---|---|---|---|
| YOLOv5n | Détection | .pt | 20–25 FPS |
| YOLOv5s | Détection | .pt | 12–15 FPS |
| MobileNet SSD | Détection | .pt | 25–30 FPS |

TABLE 4.2 – Performances d'inférence des modèles non quantifiés sur Jetson Xavier NX

Ces chiffres dépendent :

- De la résolution (ex. : 640×640 vs 512×512),
- du backend (PyTorch direct ou optimisé avec TensorRT),
- du batch size et de la mémoire GPU disponible.

• **Gains de performance attendus**

L'utilisation d'une carte NVIDIA Jetson permettrait de réduire drastiquement la latence d'inférence dans notre pipeline. Alors que le Raspberry Pi 4 atteint difficilement 1 à 2 FPS, la Jetson Xavier NX permettrait d'atteindre 10 à 15 images par seconde, en exécutant à la fois la détection de plaques et la reconnaissance de chiffres avec des modèles non quantifiés (.pt ou .onnx). Ce gain est rendu possible grâce à l'architecture CUDA couplée à TensorRT, capable d'optimiser les modèles en FP16 pour une exécution rapide, tout en maintenant une précision quasi identique au format FP32.

**Améliorations des modèles**

1. **Augmentation de dataset de matricule**

L'amélioration de la robustesse du système passe en premier lieu par l'enrichissement du jeu de données d'entraînement, tant en quantité qu'en diversité. Cela permet aux modèles d'apprendre à généraliser sur des cas variés.

a) **Objectifs d'augmentation** :

- Couvrir un large éventail d'angles de vue et d'orientations.
- Gérer la diversité des formats de plaques (1 ligne, 2 lignes).
- Réduire les cas d'échec dus aux conditions visuelles difficiles.

b) **Techniques d'augmentation recommandées** :

- Rotation (pour inclinaisons extrêmes).

- Perspective / warp affine.
- Ajout de bruit, flou ou ombrage.
- Variation de luminosité / contraste.
- Découpage et réinsertion de plaques à deux lignes dans des scènes variées.

2. **Inclusion de plaques très inclinées**

Les plaques très inclinées, vues de biais ou à partir de caméras placées en hauteur, sont souvent mal reconnues. Pour améliorer la détection dans ces cas :

- Il est nécessaire d'inclure des exemples annotés de plaques très inclinées dans l'entraînement.
- Il est recommandé d'utiliser des transformations géométriques simulées (rotation + déformation perspective) sur les images d'entraînement.

Cela permet au modèle d'apprendre à détecter les caractères même dansdes conditions de projection non frontale, et de mieux détecter les plaques dans des situations réalistes en bord de route.

3. **Ajout d'une classe spécifique pour les plaques à deux lignes**

Une amélioration clé consiste à ajouter une seconde classe dédiée aux plaques à deux lignes lors de l'entraînement du détecteur principal (YOLO).

**Avantages** :

- Permet d'adapter le post-traitement des chiffres selon la classe détectée.
- Évite la confusion entre caractères de lignes différentes.
- Améliore la reconstruction des numéros de plaque multi-lignes.

4. **Traitement différencié des plaques selon leur classe** :

Après détection d'une plaque, deux cas de traitement sont appliqués en fonction de la classe détectée (0 = une ligne, 1 = deux lignes) :

a) **Plaque à une seule ligne (classe 0)**

Tous les chiffres sont triés selon leur coordonnée X (horizontalement)

b) **Plaque à deux lignes (classe 1)**

Les chiffres sont d'abord regroupés selon leur coordonnée Y (ligne 1 ou ligne 2), puis triés par X à l'intérieur de chaque ligne

Afin de gérer les plaques à deux lignes, une seconde classe a été introduite dans le modèle de détection. Cette distinction permet d'adapter le traitement en fonction de la structure de la plaque. Les plaques à une seule ligne sont traitées par un tri horizontal (axe X), tandis que les plaques à deux lignes nécessitent un tri combiné vertical puis horizontal (axe Y puis X). Cette adaptation améliore significativement la reconstruction du numéro, en particulier dans les cas où les caractères seraient mal ordonnés ou mélangés par un traitement uniforme.

## 4.6 Conclusion

Les expérimentations menées sur les trois plateformes ont démontré la faisabilité et l'efficacité de notre système multi-plateforme de détection et reconnaissance de plaques d'immatriculation. L'approche basée sur un pipeline à deux modèles YOLO a permis d'obtenir un bon compromis entre rapidité d'exécution et précision, même sur des dispositifs à ressources limitées comme le Raspberry Pi.

Nous avons également mis en évidence les avantages de notre méthode par apprentissage profond face aux techniques classiques de reconnaissance optique de caractères (OCR), notamment en termes de robustesse dans des conditions complexes. Toutefois, certaines limites subsistent, en particulier en ce qui concerne la gestion des plaques à deux lignes, la vitesse d'inférence sur des appareils embarqués, et la reconnaissance de plaques spéciales.

Les pistes d'amélioration proposées – telles que l'intégration d'accélérateurs matériels, l'utilisation de cartes Jetson, ou encore l'enrichissement du jeu de données – constituent des perspectives concrètes pour renforcer la performance, la généralisation et la portabilité de notre système dans des contextes variés de déploiement.

# BIBLIOGRAPHIE


[1] https://www.netapp.com/artificial-intelligence/what-is-artificial-intelligence/

[2] https://www.ibm.com/fr-fr/topics/computer-vision#Qu

[3] https://www.youtube.com/watch?v=dzWnBdk2EJM

[4] https://fr.wikipedia.org/wiki/Image

[5] https://di.univblida.dz/jspui/bitstream/123456789/2023/1/Untitled.pdf

[6] https://www.intelligence-artificielle-school.com/alternance-et-entreprises/secteur-d-activite/quel-est-le-role-de-lia-dans-le-secteur-automobile/#:~:text=L'Intelligence

[7] https://www.cl72.org/090imagePLib/books/Gonzales,Woods-Digital.Image.Processing.4th.Edition.pdf

[8] Jourlin,M.(2011).Traitementdimages:Fondementsetapplications.Dunod

[9] https://members.loria.fr/MOBerger/Enseignement/ENSG/intro.pdf

[10] Goodfellow,I.,Bengio,Y.,&Courville,A.(2016).DeepLearning.MITPress.

[11] LeCun,Y.,Bengio,Y.,&Hinton,G.(2015).Deeplearning.Nature,521(7553),436444

[12] Upton,E.,&Halfacree,G.(2020).RaspberryPiUserGuide(4edition).Wiley

[13] https://www.algerie360.com/

[14] https://jolimatin.com/vignette-automobile-2025-en-algerie-une-transition-vers-le-numerique

[15] Loin24-08du22JoumadaElOula1446correspondantau24novembre2024portantloidefinancespour2025

[16] https://www.lactualgerie.com/automobile/lancement-dune-nouvelle-plateforme-numerique-dedie

[17] https://legal-doctrine.com/edition/tout-sur-votre-vignette-automobile-2025-daa4ad3476b744

[18] https://qassimatouka.mf.gov.dz/

[19] https://www.lactualgerie.com/automobile/qassimatouka-procurez-vous-votre-vignette-automobi

[20] https://di.univ-blida.dz/jspui/bitstream/123456789/12090/1/m%C3%A9moireprojetfind'etude.pdf#:~:text=L%27ANPR%20,de%20v%C3%A9hicule%20puis%20fournit%20des

[21] https://www.mdpi.com/1424-8220/21/9/3028#:~:text=Number%20Plate%20Recognition%20involves%20acquisition,general%20processes%20involved%20in%20ANPR

[22] https://dspace.univ-guelma.dz/jspui/bitstream/123456789/4045/1/DownloadFile%287%29.pdf#:~:text=Recognition%20Systems%20,9

[23] https://fr.wikipedia.org/wiki/Lecture_automatis%C3%A9e_de_plaques_d%27immatriculation#:~:text=En%202005%2C%20des%20syst%C3%A8mes%20peuvent,cameras%2C%20ou%20encore%20des%20%C3%A9quipements

[24] https://www.rspsciencehub.com/article_23837_90b9e72c5c0f29336c186b69117da97e.pdf#:~:text=toll/20collec/tion/C/20intelligent/transportation/systems,technology/20develops/20quickly/20with/20the

[25] https://pubmed.ncbi.nlm.nih.gov/33925845/#:~:text=involving/20computer/20vision/20/28CV/29,Things/20is

[26] ResNetandLSTMBasedAccurateApproachforLicensePlateDetectionandRecognitionNaamanOmar: https://doi.org/10.18280/ts.390514

[27] EnhancedYOLOv8-BasedSystemforAutomaticNumberPlateRecognitionTamimMahmudAl-Hasan1, VictorBonnefille2andFaycalBensaali1,:https://doi.org/10.3390/technologies12090164

[28] V.Gnanaprakash,N.Kanthimathi,andN.Saranya,"Automaticnumberplaterecognitionusingdeeplearnin "IOPConferenceSeries:MaterialsScienceandEngineering,vol.1084,no.1,p.012027,Mar. 2021.[Online].Available:https://iopscience.iop.org/article/10.1088/1757-899X/1084/1/012027

[29] LicensePlateRecognitionSystemBasedonImprovedYOLOv5andGRUHENGLIANGSHI1ANDDONGNANZHAO2

[30] AutomaticNumberPlateRecognitionSwanandJoshi,PramodJejure,ChatrasalJadhav, VishalJankar:https://ijsrst.com/index.php/home/article/view/IJSRST2411476

[31] PatrolVision:AutomatedLicensePlateRecognitioninthewild1stAnmolSinghalNewYorkUniversityas15 nyu.edu

[32] https://fr.wikipedia.org/wiki/Plaque_d%27immatriculation_alg%C3%A9rienne

[33] https://www.datacamp.com/fr/blog/yolo-object-detection-explained

[34] MouadBensouilah(mouad.bensouilah@gmail.com)LPADdatabase.

[35] Kaggle(https://www.kaggle.com/datasets/aladdinss/plate-digits-classification-dataset)

[36] https://huggingface.co/keremberke/yolov5n-license-plate/blob/main/best.pt

[37] V7Labs.(2022,March7).MeanAveragePrecision(mAP)Explained:EverythingYouNeedtoKnow.
Consultsurhttps://www.v7labs.com/blog/mean-averageprecision

[38] https://idiotdeveloper.com/what-is-intersection-over-union-iou/

[39] LearnOpenCV.(2022,August9).MeanAveragePrecision(mAP)inObjectDetection.
Consultsurhttps://learnopencv.com/mean-average-precision-map-objectdetection-model-evaluati

[40] Hui,J.(2018,March6).mAP(meanAveragePrecision)forObjectDetection.Medium.
Consultsurhttps://jonathan-hui.medium.com/map-mean-averageprecision-for-object-detection-45

[41] Encord.(2023,November5).MeanAveragePrecisioninObjectDetection.Consultsurhttps:
//encord.com/blog/mean-average-precision-object-detection/

[42] RoboflowBlog.(2024,May30).WhatisMeanAveragePrecision(mAP)inObjectDetection?
Consultsurhttps://blog.roboflow.com/mean-average-precision/

[43] Lin,T.Y.,Maire,M.,Belongie,S.,Hays,J.,Perona,P.,Ramanan,D.,...&Zitnick,C.L.
(2014).MicrosoftCOCO:CommonObjectsinContext.Europeanconferenceoncomputervision.
(NormessouventutilisespourmAP@0.5:0.95)

[44] https://github.com/thawro/yolov8-digits-detection

[45] TensorFlowLiteDocumentation..https://ai.google.dev/edge/litert?hl=fr

[46] TensorFlowBlog.(2019,August5).TensorFlowModelOptimizationToolkitPost-trainingfloat16quanti
https://blog.tensorflow.org/2019/08/tensorflow-model-optimization-toolkit_5.html

[47] AnalyticsVidhya.(2024,January24).OptimizingNeuralNetworks:
UnveilingthePowerofQuantizationTechniques.https://www.analyticsvidhya.com/blog/
2024/01/optimizing-neural-networks-unveiling-the-power-of-quantization-techniques/

[48] ONNXDocumentation.https://onnx.ai/

[49] Viso.ai.(2023,December18).ONNXExplained:ANewParadigminAIInteroperability.https:
//viso.ai/computer-vision/onnx-explained-a-new-paradigm-in-ai-interoperability/

[50] EncordBlog.(2024,August15).UnderstandingONNX:EnhancingAIModelInteroperability.
https://encord.com/blog/onnx-open-neural-network-exchange-format/

[51] OpenVINOToolkitDocumentation.https://docs.openvino.ai/2025/index.html

[52] https://coral.ai/docs/

[53] https://tutorials-raspberrypi.de/tensorflow-lite-mit-raspberry-pi-google-coral-tpu/

[54] https://www.waveshare.com/hailo-8-027841.htm

[55] https://developer.nvidia.com/embedded/jetson-modules

[56] https://www.tensorflow.org/install/gpu?hl=fr

[57] https://pytorch.org/get-started/locally/

[58] https://developer.nvidia.com/embedded/jetson-benchmarks

[59] https://github.com/ultralytics/yolov5/pull/10467

[60] Smith,R.(2007).AnoverviewoftheTesseractOCRengine.InNinthInternationalConferenceonDocumentA
2,pp.629-633).IEEE.

[61] https://github.com/ultralytics/ultralytics

[62] https://computeroptics.ru/KO/PDF/KO45-1/450111.pdf

[63] Redmon,J.,Divvala,S.,Girshick,R.,&Farhadi,A.(2016).YouOnlyLookOnce:Unified,
Real-TimeObjectDetection.InProceedingsoftheIEEEConferenceonComputerVisionandPatternRecognit
779-788).

[64] Redmon,J.,&Farhadi,A.(2017).YOLO9000:Better,Faster,Stronger.
InProceedingsoftheIEEEConferenceonComputerVisionandPatternRecognition(pp.
7263-7271).

[65] Redmon,J.,&Farhadi,A.(2018).YOLOv3:AnIncrementalImprovement.arXivpreprintarXiv:
1804.02767.

[66] Bochkovskiy,A.,Wang,C.Y.,&Liao,H.Y.M.(2020).YOLOv4:OptimalSpeedandAccuracyofObjectDetectic
arXivpreprintarXiv:2004.10934.

[67] Jocher,G.,etal.(2020).YOLOv5.https://github.com/ultralytics/yolov5.

[68] Li,C.,etal.(2022).YOLOv6:ASingle-StageObjectDetectionFrameworkforIndustrialApplications.
arXivpreprintarXiv:2209.02976.

[69] Wang,C.Y.,Bochkovskiy,A.,&Liao,H.Y.M.(2022).YOLOv7:TrainableBag-of-FreebiesSetsNewState-of
arXivpreprintarXiv:2207.02696.

[70] Ge,Z.,Liu,S.,Wang,F.,Li,Z.,&Sun,J.(2021).YOLOX:ExceedingYOLOSeriesin2021.
arXivpreprintarXiv:2107.08430.

[71] Jocher,G.,etal.(2023).YOLOv8.https://github.com/ultralytics/ultralytics.

[72] Lin,T.Y.,Dollr,P.,Girshick,R.,He,K.,Hariharan,B.,&Belongie,S.(2017)
.FeaturePyramidNetworksforObjectDetection.InProceedingsoftheIEEEConferenceonComputerVisiona
2117-2125).

[73] Liu,S.,Qi,L.,Qin,H.,Shi,J.,&Jia,J.(2018).PathAggregationNetworkforInstanceSegmentation.
InProceedingsoftheIEEEConferenceonComputerVisionandPatternRecognition(pp.
8759-8768).

[74] Wang,C.Y.,Liao,H.Y.M.,Wu,Y.H.,Chen,P.Y.,Hsieh,J.W.,&Yeh,I.H.(2020).CSPNet: ANewBackbonethatcanEnhanceLearningCapabilityofCNN.InProceedingsoftheIEEE/ CVFConferenceonComputerVisionandPatternRecognitionWorkshops(pp.390-391).

[75] Misra,D.(2019).Mish:ASelfRegularizedNon-MonotonicNeuralActivationFunction. arXivpreprintarXiv:1908.08681.

[76] Zhang,H.,Wu,C.,Zhang,Z.,Zhu,Y.,Zhang,Z.,Lin,H.,...&Smola,A.(2020).ResNeSt: Split-AttentionNetworks.arXivpreprintarXiv:2004.08955.

[77] Tian,Z.,Shen,C.,Chen,H.,&He,T.(2019).FCOS:FullyConvolutionalOne-StageObjectDetection. InProceedingsoftheIEEE/CVFInternationalConferenceonComputerVision(pp.9627-9636).

[78] https://arxiv.org/pdf/1506.02640.pdf

[79] https://arxiv.org/pdf/1506.02640.pdf

[80] https://www.mdpi.com/2079-9292/10/3/279/htm

[81] arxiv.org/pdf/2004.10934

[82] ultralytics/yolov5WikiGitHub

[83] arxiv.org/pdf/2207.02696

[84] Figure1fromOptimizingTrafficLightControlusingYOLOv8forReal-TimeVehicleDetectionandTrafficD SemanticScholar

[85] gr1_lrg.jpg(27653056)

[86] anchorboxesyolov8fpn-SearchImages

[87] https://fr.wikipedia.org/wiki/Plaque_d%27immatriculation_alg%C3%A9rienne

[88] https://pytorch.org/

[89] https://www.ultralytics.com/

[90] https://fr.wikipedia.org/wiki/Image

[91] https://theses-algerie.com/2934377917733961/memoire-de-master/ universite-yahia-fares---medea/reconnaissance-automatique-des-plaques-d-immatriculation-pa C3%A9seau-cnn.com

[92] https://www.univ-bejaia.dz/xmlui/handle/123456789/25077

[93] https://repository.univ-msila.dz/items/c79149c8-8ece-4dd8-8910-d3ad6e15ebf1?

[94] https://repository.univ-msila.dz/items/c79149c8-8ece-4dd8-8910-d3ad6e15ebf1?

# Résumé

Ce mémoire présente la conception et la réalisation d'un système intelligent de reconnaissance automatique de plaques d'immatriculation (ANPR) basé sur l'intelligence artificielle et déployé sur différentes plateformes, dont le Raspberry PI. L'objectif est d'automatiser le processus de détection et de lecture des plaques d'immatriculation des véhicules algériens, ainsi que la vérification de la validité des vignettes.

Le système proposé repose sur l'utilisation de deux modèles de deep learning : YOLOv5n pour la détection des plaques, et YOLOv8s pour la reconnaissance des chiffres. Ces modèles ont été entraînés sur des jeux de données construits et annotés manuellement pour les besoins du projet. L'application est conçue pour fonctionner sur trois supports : une interface PC, un système embarqué avec Raspberry PI 4, et une application mobile Android. L'ensemble des interfaces est connecté à une base de données Firebase pour la consultation en temps réel des informations liées à la vignette. Les résultats expérimentaux montrent une bonne précision de détection et de reconnaissance dans différentes conditions (jour, nuit, flou, contre-jour). Le système offre une solution flexible, efficace et économique pour les contrôles routiers automatisés, tout en respectant les exigences locales en matière de numérisation. Ce projet ouvre des perspectives intéressantes, notamment en matière d'optimisation du traitement sur cartes embarquées plus puissantes comme les NVIDIA Jetson, et de gestion avancée des cas difficiles. Il illustre concrètement l'impact des technologies d'intelligence artificielle dans le domaine du transport et de la sécurité routière.

# Abstract

This thesis presents the design and development of an intelligent Automatic Number Plate Recognition (ANPR) system based on artificial intelligence and deployed across multiple platforms, including the Raspberry PI. The objective is to automate the detection and reading of Algerian vehicle license plates in response to the 2025 digital vehicle vignette reform. The proposed system uses two deep learning models : YOLOv5n for license plate detection and YOLOv8s for digit recognition. These models were trained on custom datasets prepared specifically for this project. The application operates on three platforms : a PC interface, an embedded system on Raspberry PI 4, and an Android mobile application. All interfaces are connected to a Firebase database for real-time access to vehicle vignette information. Experimental results demonstrate high accuracy in plate detection and digit recognition under various conditions (daylight, night, blur, backlight). The system provides a flexible, efficient, and cost-effective solution for automated road checks, while complying with local digitalization requirements. This project opens promising perspectives, including optimization for more

powerful embedded hardware like NVIDIA Jetson boards and better handling of challenging cases. It offers a concrete example of how artificial intelligence technologies can impact transport and road safety systems.