الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Master

Filière Électronique
Spécialité électronique des systèmes embarqués

présenté par

Boulahchiche Nadir

&

Ferkioui Mecheri

Système Intelligent de reconnaissance de plaque d'immatriculation via Raspberry Pi

Proposé par : Naceur Djamila & Bougherira Hamida

Année Universitaire 2020-2021

Au terme de ce mémoire, on tient à exprimer nos remerciements et notre profonde gratitude au bon DIEU le tout puissant, qui nous a donné la force pour mener à bien ce modeste travail.

On remercie vivement notre professeur madame **NACEUR DJAMILA** pour avoir supervisé ce travail avec de la compétence et de la disponibilité.

Nos vifs remerciements vont également aux membres du jury pour avoir accepter d'évaluer ce travail.

A nos familles et nos amis qui par leurs prières et leurs encouragements, on a pu surmonter toutes les difficultés.

Enfin, qu'il nous soit permis de remercier toutes les personnes qui nous ont aidé durant notre cursus universitaire.

ملخص: في هذا المشروع ، أنجزنا نظامًا تلقائيًا للكشف عن لوحة الترخيص باستخدام تقنيات التعلم العميق ومعالجة الصور كلغة برمجة وكود الاستوديو المرئي كبيئة تطوير Python ، وبعد ذلك قمنا بتطبيق هذا النظام في العاسبيري بي. استخدمنا لتدريب نماذج التعلم العميق الخاصة بنا. نستخدم نوعين من الخوارزميات لاكتشاف البلاك google بالإضافة إلى تعاون.

بعد الاكتشاف ، استخدمنا معالجة الصور مرة أخرى لتقسيم الأحرف في اللوحة ، ثم تتم قراءة كل حرف على حدة ، وفي النهاية نحفظ اللوحة في قاعدة بيانات mysql لاستخدامها في المستقبل..

كلمات المفاتيح: لوحة الترخيص. التعلم العميق .

Résumé : Dans ce projet nous avons réalisé un système de détection de plaque d'immatriculation automatique en utilisant des techniques d'apprentissage en profondeur et de traitement d'images, par la suite nous avons implémenté ce système dans un raspberry pi. Nous avons utilisé Python comme langage de programmation et visual studio code comme environnement de développement ainsi que google collab pour entrainer nos modèles d'apprentissage approfondi .Nous utilisons deux types d'algorithmes pour la détection de la plaque. Après la détection nous avons utilisé à nouveau le traitement d'image pour segmenter les caractères dans la plaque, puis chaque caractère va être lu individuellement. A la fin nous enregistrons la plaque dans une base de données mysql pour de futures utilisations.

Mots clés : Apprentissage approfondi; OpenCV ; Tensorflow ; SSD MobileNets ; Raspberry pi ; plaque d'immatriculation.

Abstract : In this project we made an automatic license plate detection system using deep learning and image processing techniques, subsequently we implemented this system in a raspberry pi. We used Python as the programming language and visual studio code as the development environment as well as google collab to train our deep learning models. We use two types of algorithms for plate detection.

After the detection we used image processing again to segment the characters in the plate, then each character will be read individually. At the end we save the plate in a mysql database for future use.

Keywords : Deep learning; OpenCV; Tensorflow; MobileNets SSD; Raspberry pi; numberplate.Raspberry pi.

Listes des acronymes et abréviations

IA : Intelligence Artificielle.

ANN : Réseau neuronal artificielle

CNN : Convolutional Neural Network (réseau neuronal convolutif).

ML : Machine Learning.

DL : Deep Learning.

ANPR : Automatic Number Plate Recognition

CPU : Central Proccessing Unit (unités centrales de traitement).

GPU : Graphics Processing Unit (unités de traitement graphique).

SSD : Single Shot MultiBox Detector.

OS : Operating Système (Système d'exploitation).

mAP : Mean Average Precision.

VOC : Classes d'Objets Visuals.

VGGNet : Visual Geometry Group Network.

FPS : Image par seconde (Frames per seconde).

CUDA : Compute Unified Device Architecture (Architecture de périphérique unifiée de calcul).

YOLO : You only look once

OCR : optical character recognition (reconnaissance optique de caractères)

API : interface de programme d'application (application program interface)

Table des matières

Liste des figures

Liste des tableaux

Utiliser cette liste si vous avez des tableaux dans votre manuscrit.

Introduction générale

La reconnaissance automatique des plaques d'immatriculation aide à leur identification sans le besoin d'une intervention humaine, ce besoin de reconnaissance est devenu de plus en plus important ces dernières décennies. Grâce à l'utilisation de la capture d'images à grande vitesse avec éclairage d'appoint, la détection de caractères dans les images fournies, la vérification des séquences de caractères comme étant celles d'une plaque d'immatriculation de véhicule, la reconnaissance de caractères pour convertir l'image en texte, nous mène ainsi à un ensemble de métadonnées qui identifie une image contenant une plaque d'immatriculation de véhicule et le texte décodé associé à cette plaque.

L'ANPR (Automatic Number Plate Recognition) est la technologie sous-jacente utilisée pour détecter une plaque d'immatriculation et son numéro de véhicule puis fournit des informations pouvant être interprétées, stockées ou mises en correspondance pour créer une application basée sur l'ANPR.

La plupart des membres du public savent que l'ANPR est utilisé par de nombreuses forces de l'ordre pour traquer les comportements criminels et est également considéré sur de nombreuses autoroutes comme une méthode de détection des excès de vitesse grâce au calcul de la vitesse moyenne. Cependant, l'ANPR est utilisé de diverses autres manières pour soutenir la sécurité et la sûreté du public ainsi que pour favoriser l'efficacité dans la façon dont nous interagissons avec les transports et les infrastructures basées sur les véhicules.

Un système ANPR typique peut être divisé en quatre étapes :

1. Détection de plaque d'immatriculation dans l'image capturée.
2. Segmentation de caractères dans la plaque.
3. Reconnaissance de chaque caractère individuellement.
4. Implémentation dans une carte électronique avec camera connecté avec un centre pour la surveillance en temps réel.

L'objectif de ce projet est la détection des plaques d'immatriculation algériennes en utilisant l'apprentissage profond et particulièrement les réseaux de neurone à convolution (CNN et SSD), et/ou en utilisant le traitement d'image .

L'étape de segmentation des caractères de la plaque est ensuite réalisée en utilisant le traitement d'image, et enfin l'étape de reconnaissance fera appel à un autre réseau de neurones à convolution (SSD). Après la réalisation de notre programme, nous procédons à son implémentation dans un système embarqué à base de carte raspberry pi 4 model b, connecté à une camera afin que le traitement soit en temps réel.

Nous avons finalisé notre travail par la réalisation d’une base de données d'un personnel universitaire, gouvernemental ou autre afin d'enregistrer les présences de chaque véhicule.

Le mémoire est alors organisé en trois chapitres :

Le premier chapitre, est articulé autour des généralités sur l’intelligence artificielle, un aperçu sur les plaques d'immatriculation et un descriptif du matériel utilisé.

Le deuxième chapitre fait l’objet d’une description du traitement d'image au deep learning en passant par les réseaux de neurones et détaillera les techniques que nous avons utilisées pour la détection et la lecture. Ainsi il mettra notre travail dans le contexte du sujet.

Enfin le dernier chapitre est consacré à notre travail d'implémentation, aux résultats obtenus, et aux problèmes et solutions trouvés.

Chapitre 1 Généralités

1.1 Introduction

Dans ce chapitre nous commençons par l'introduction de l'intelligence artificielle et de ses domaines d'application, nous introduisons également les sous-ensembles de l'IA et l'application de chaque ensemble. Un bref descriptif des plaques d'immatriculation et du matériel utilisé qui consiste en une carte électronique (raspberry pi) et d'une camera (raspberry pi Rev 1.3 5mp) est présenté, suivi d'une conclusion .

1.2 Intelligence artificielle

1.2.1 définition

L'IA désigne la possibilité pour une machine de reproduire des comportements liés aux humains, tels que le raisonnement, la planification et la créativité.

L'IA permet à des systèmes techniques de percevoir leur environnement, gérer ces perceptions, résoudre des problèmes et entreprendre des actions pour atteindre un but précis. L'ordinateur reçoit des données (déjà préparées ou collectées via ses capteurs - une caméra, par exemple) les analyse et réagit.

Les systèmes dotés d'IA sont capables d'adapter leurs comportements (plus ou moins) en analysant les effets produits par leurs actions précédentes, travaillant de manière autonome.[1]

1.2.2 Historique

a L'intelligence artificielle avant 2000

Les premières traces de l'IA remontent à 1950 dans un article d'Alan Turing intitulé "Computing Machinery and Intelligence"[1] dans lequel le mathématicien explore le problème de définir si une machine est consciente ou non. De cet article découlera ce que l'on appelle

aujourd'hui le Test de Turing qui permet d'évaluer la capacité d'une machine à tenir une conversation humaine.

L'officialisation de l'intelligence artificielle comme véritable domaine scientifique date de 1956 lors d'une conférence aux États-Unis qui s'est tenue au Dartmouth College. Par la suite, ce domaine atteindra de prestigieuses universités comme celles de Stanford, du MIT, ou encore d'Édimbourg.

Dès le milieu des années 60, la recherche autour de l'IA sur le sol américain était principalement financée par le Département de la Défense. Dans le même temps, des laboratoires font leur apparition à travers le monde. Certains experts prédisaient à l'époque que « des machines seront capables, d'ici 20 ans, de faire le travail que toute personne peut faire ». Si l'idée était visionnaire, même à l'heure actuelle, l'intelligence artificielle n'a pas encore pris cette importance dans nos vies.

Dans les années 80, le succès des systèmes experts permet de relancer les projets de recherche sur l'intelligence artificielle. Un système expert était un ordinateur capable de se comporter comme un expert (humain), mais dans un domaine bien précis. Grâce à ce succès, le marché de l'IA atteint une valeur d'un milliard de dollars, ce qui motive les différents gouvernements à de nouveau soutenir financièrement plus de projets académiques.

Le développement exponentiel des performances informatiques, notamment en suivant la loi de Moore, permet entre 1990 et 2000 d'exploiter l'IA sur des terrains jusqu'alors peu communs. On retrouve à cette époque le data mining, ou encore les diagnostics médicaux. Il faudra attendre 1997 pour une véritable sortie médiatique lorsque le fameux Deep Blue créé par IBM a battu Garry Kasparov, alors champion du monde d'échec.[1]

b 2000 - 2010 : l'IA représente un enjeu de société

Entre 2000 et 2010, notre société vit un véritable boom informatique. Non seulement la loi de Moore poursuit son chemin, mais les Hommes s'équipent. Les ordinateurs personnels deviennent de plus en plus accessibles, Internet se déploie, les smartphones voient le jour … La connectivité et la mobilité lancent l'ère de l'Homo Numericus.

Jusqu'à 2010, on s'interroge également sur l'éthique de l'intégration de l'IA dans de nombreux secteurs. Ainsi, en 2007 la Corée du Sud dévoile une charte de l'éthique des robots dans le but de poser des limites et des normes aux utilisateurs ainsi qu'aux constructeurs. En 2009, le MIT lance un projet réunissant de grands scientifiques de l'IA pour réfléchir aux grandes lignes de la recherche sur ce domaine.[1]

c À partir de 2010 : l'IA sans limites

Dès le début de notre décennie, l'IA s'illustre grâce aux prouesses de Waston d'IBM. En 2011, ce super-cerveau a battu en direct les deux plus grands champions de Jeopardy!. Un exercice loin d'être simple pour un ordinateur. Néanmoins, après Deep Blue, les années 2010 marquent un tournant dans la médiatisation des recherches.

La loi de Moore continue de guider les progrès de l'intelligence artificielle, mais le traitement de la donnée vient renforcer tout cela. Pour exécuter une tâche, un système n'a besoin que de règles. Lorsqu'il s'agit d'avoir une réflexion ou de livrer la réponse la plus juste possible, il faut que ce système apprenne. C'est ainsi que les chercheurs développent de nouveaux procédés pour le machine learning puis le deep learning. Rapidement, ces approches nourries par les données passent de nombreux records, poussant de nombreux autres projets à suivre cette voie. De plus, le développement des technologies pour l'intelligence artificielle permet de lancer des projets très divers et de ne plus penser calcul pur et dur, mais d'intégrer le traitement des images.

C'est à partir de ce moment que certaines sociétés vont prendre les devants. En effet, la problématique de l'IA n'est plus d'avoir les cerveaux pour élaborer des systèmes, mais d'avoir de la donnée à traiter. C'est pour cela que Google devient rapidement un pionnier. En 2012, la firme de Mountain View n'avait que quelques projets d'usages, contre 2 700 trois ans plus tard.[1]

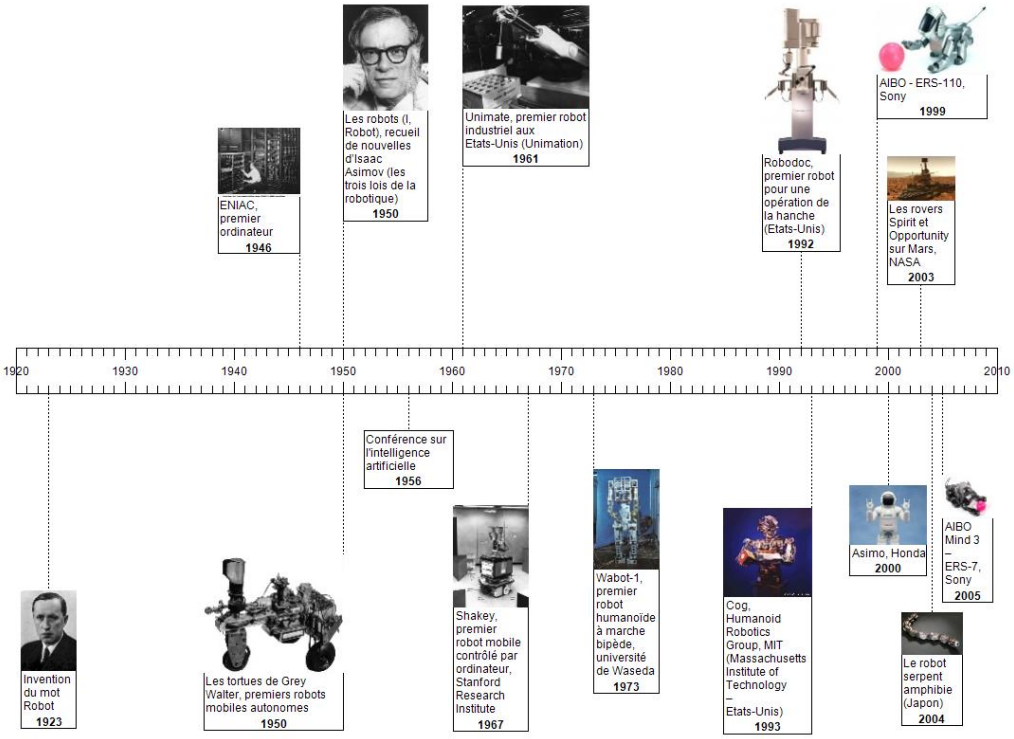


Figure 1.1. l'histoire de l'IA.

1.2.3 Domaines d'utilisation

a L'intelligence artificielle dans la santé

L'Intelligence artificielle est utilisée dans plusieurs solutions proposées dans le domaine de la santé. Elle est utilisée pour accélérer le processus de diagnostic, avec un taux de précision qui pourrait dépasser celui de l'homme. De même, il devient plus facile de traiter des pathologies graves comme le cancer. L'IA peut aussi favoriser la conception de nouveaux médicaments et réduire le temps qui sépare la découverte d'une molécule de sa mise sur le marché.

Par ailleurs, de nombreuses entreprises mettent sur le marché des applications mobiles destinées à suivre le traitement de certaines pathologies chez les patients. Dans les pays asiatiques, des robots sont déjà expérimentés pour venir en aide aux personnes en difficulté dans leurs activités quotidiennes..[2]

b le transport et l'IA

L'utilisation de l'intelligence artificielle contribue à de nombreux progrès dans le domaine du transport. Des systèmes sont développés pour détecter les signes de fatigue sur le visage d'un conducteur. Dans certains cas, le contrôle du véhicule est entièrement assuré par la machine quand le conducteur n'est plus en état de conduire. Cela est valable pour les voitures de particuliers ainsi que les semi-remorques. Il existe aussi des dispositifs qui modélisent le transport public ou encore d'autres qui vont réguler les feux de circulation pour améliorer le flux de circulation des automobiles.[2]

c Le traitement du langage, une branche de l'Intelligence artificielle

Le traitement du langage naturel est une branche de l'Intelligence artificielle, qui concerne les programmes de reconnaissance vocale, les robots de conversation (chat bot) et bien d'autres. Des applications développées grâce à l'IA sont ainsi utilisées pour réaliser rapidement des tâches plus ou moins fastidieuses pour l'homme. Elles sont capables de structurer un document volumineux, de traduire instantanément un texte ou une conversation en plusieurs langues, de résumer des textes ou encore de répondre à des questions récurrentes.[2]

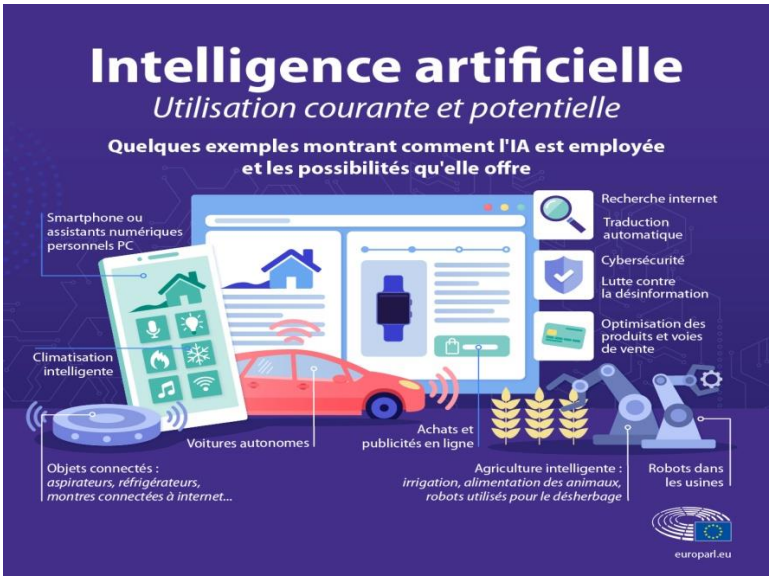


Figure 1.2. Utilisation des IA dans différents domaines.

1.2.4 sous-ensembles de l'IA

L'intelligence artificielle ce compose d'un totale de 3 sous-ensembles , chaque ensemble a ses propriétés avec leurs propres applications, nous allons avoir une petite définition pour chaque ensemble et son utilité.[3]

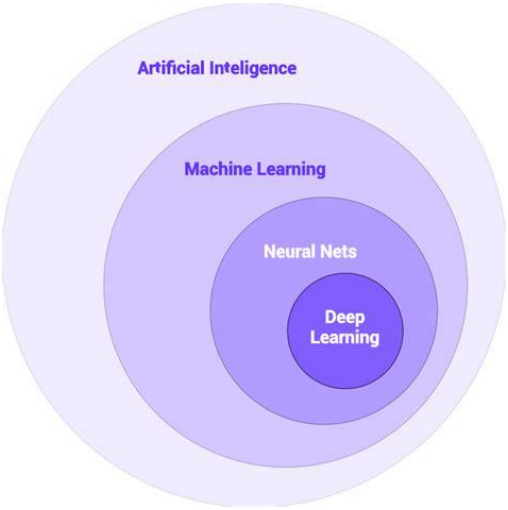


Figure 1.3. sous-ensembles des AI

a Machine learning

Le machine learning (L'apprentissage automatique) est une méthode utilisée pour réaliser l'IA en donnant à une machine un ensemble de données et en lui demandant d'apprendre à prédire un résultat réel. Dans le Machine Learning, vous pouvez avoir un Machine Learning supervisé ou non supervisé, en fonction de la disponibilité d'une variable cible ou de la sortie.

L'application du machine learning se situe dans divers segments qui sont : la reconnaissance vocale, la prédiction du trafic, la reconnaissance d'images, la santé, le courrier indésirable, l'assistant personnel (virtuel), la détection de fraude, le marketing et les ventes, etc.[4]

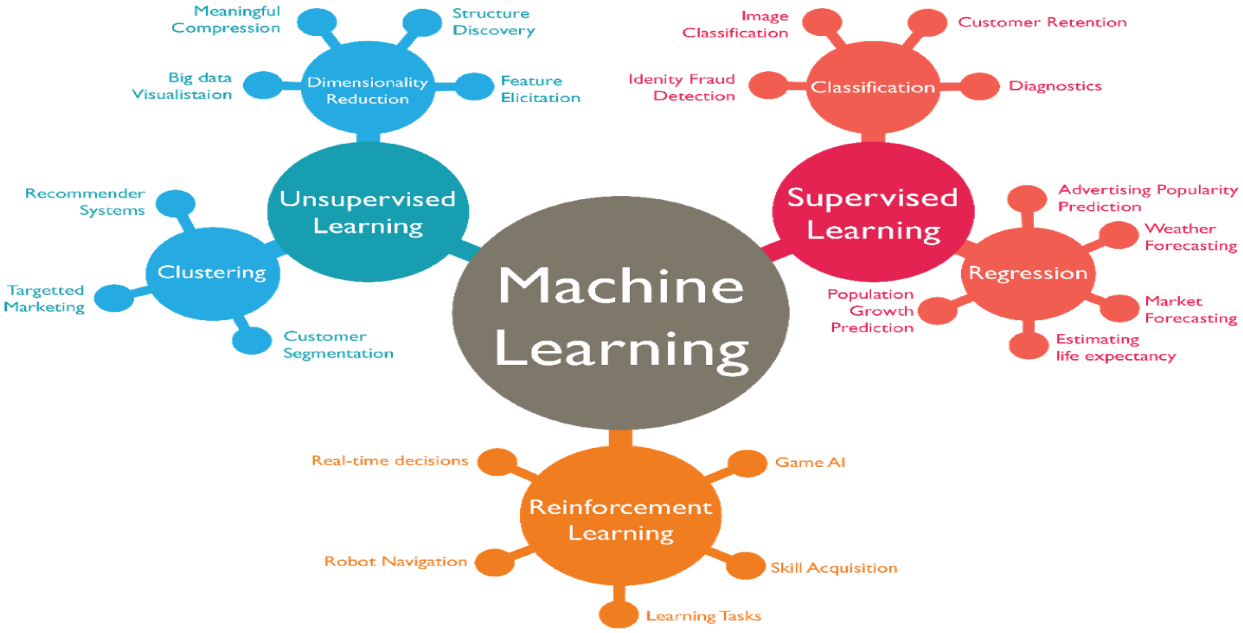


Figure 1.4. Utilisation du machine learning.

b réseau neuronal (Neural Nets)

Les réseaux de neurones sont des systèmes informatiques avec des nœuds interconnectés qui fonctionnent un peu comme les neurones du cerveau humain. À l'aide d'algorithmes, ils peuvent reconnaître des modèles et des corrélations cachés dans les données brutes, les regrouper et les classer et, au fil du temps, apprendre et s'améliorer en permanence.

Les réseaux de neurones peuvent être appliqués dans les même domaines que ceux du machine learning et les améliorer, ce qui les rend encore plus importants.

Les réseaux de neurones peuvent également améliorer les processus de décision dans des domaines tels que : la vision artificielle pour interpréter les photos et vidéos brutes, le diagnostic médical et des maladies, les systèmes de contrôle robotique, les prévisions financières pour les cours des actions, les devises, les options, les contrats à terme, les faillites et les notations obligatoires, etc. [3]

c Deep Learning

Le Deep Learning fait partie du Machine Learning utilisé pour résoudre des problèmes complexes et créer des solutions intelligentes. Le concept de base du Deep Learning a été dérivé de la structure et de la fonction du cerveau humain. Le Deep Learning utilise des réseaux de neurones artificiels pour analyser les données et faire des prédictions. Il a trouvé son application dans presque tous les secteurs d'activité et de recherche [3].

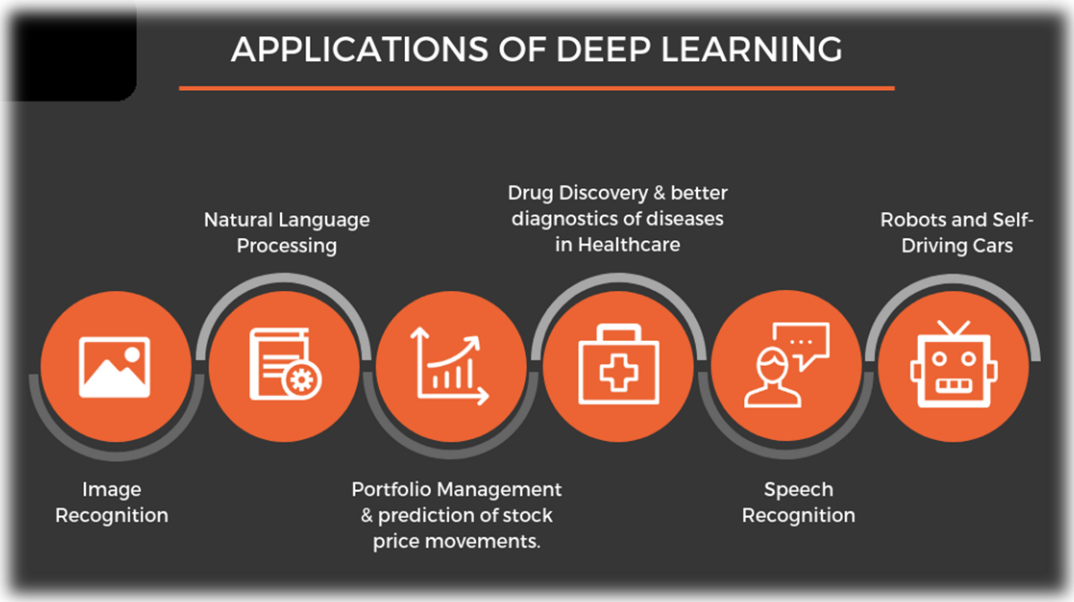


Figure 1.5. Applications du deep learning.

1.3 plaque d'immatriculation

1.3.1 définition

Une plaque d'immatriculation de véhicule figure1.6, est une plaque de métal ou de plastique fixée à un véhicule à moteur ou à une remorque à des fins d'identification officielle. Tous les pays exigent des plaques d'immatriculation pour les véhicules routiers tels que les voitures, les camions et les motos. Le fait qu'elles soient requises pour d'autres véhicules, tels que des vélos, des bateaux ou des tracteurs, peut varier selon la juridiction. L'identifiant d'enregistrement est un identifiant numérique ou alphanumérique qui identifie de manière unique le véhicule ou le propriétaire du véhicule dans le registre des véhicules de la région émettrice. Dans certains pays, l'identifiant est unique dans tout le pays, tandis que dans d'autres, il est unique dans un état ou une province. Le fait que l'identifiant soit associé à un véhicule ou à une personne varie également selon l'agence émettrice. Il existe également des plaques d'immatriculation électroniques [5].

Figure 1.6. Plaques d'immatriculation à travers le monde.

1.3.2 plaque d'immatriculation algérienne

Les plaques d'immatriculation algériennes sont fabriquées selon les mêmes normes que leurs homologues françaises, en utilisant la même police et les mêmes dimensions

Les plaques d'émission standard sont blanches avec des chiffres noirs (fixées à l'avant du véhicule) et jaunes avec des chiffres noirs (montées à l'arrière du véhicule). Étant composées uniquement de chiffres, elles sont l'une des rares plaques d'immatriculation des véhicules à pouvoir être appelées avec précision «plaques d'immatriculation».

Dans notre cas, nous allons identifier la plaque d'immatriculation algérienne, Figure 1.7, on a vu dans la figure la différence entre chaque plaque d'immatriculation, la première chose à remarquer est que la plaque d'immatriculation algérienne ne contient que des nombres dans son matricule, ce qui facilite la tache d'entrainement de notre modèle IA où nous allons avoir besoin seulement des numéros pour l'identification et non pas des caractères.[8]

Figure 1.7. Plaque d'immatriculation algérienne.

1.4 Matériel

Le matériel utilisé est comme suit :

1.4.1 La carte raspberry pi

Le Raspberry Pi est un ordinateur de petite taille à faible coût qui se branche sur un écran d'ordinateur ou un téléviseur et utilise un clavier et une souris standard. C'est un petit appareil qui permet aux personnes de tout âge d'explorer l'informatique et d'apprendre à programmer dans des langages comme Scratch et Python. Il est capable de faire tout ce que vous attendez d'un ordinateur de bureau, de la navigation sur Internet à la lecture de vidéos haute définition, en passant par la création de feuilles de calcul, par le traitement de texte et par les jeux.

De plus, le Raspberry Pi a la capacité d'interagir avec le monde extérieur et a été utilisé dans un large éventail de projets de fabricants numériques, des machines à musique à tweets avec caméras infrarouges. [6]

1.4.2 La carte raspberry pi 4 model b

Raspberry Pi 4 Model B est le dernier produit de la gamme populaire Raspberry Pi des ordinateurs. Il offre des augmentations révolutionnaires de la vitesse du processeur.

Raspberry Pi 3 modèle B+, conserve la rétrocompatibilité et similarité en consommation d'énergie. Pour l'utilisateur, Raspberry Pi 4 modèle B fournit un ordinateur de bureau dont les performances sont comparables aux systèmes PC x86 d'entrée de gamme.

Les principales caractéristiques de ce produit incluent un quad-core 64 bits hautes performances du processeur, prise en charge du double affichage à des résolutions jusqu'à 4K via une paire de Ports micro-HDMI, décodage vidéo matériel jusqu'à 4Kp60, jusqu'à 8 Go de RAM, LAN sans fil bibande 2,4/5,0 GHz, Bluetooth 5.0, Ethernet Gigabit, USB 3.0, et la capacité PoE (via un module complémentaire PoE HAT distinct). Le LAN sans fil double bande et le Bluetooth ont une certification de conformité modulaire, permettant d'être conçu dans des produits finis avec considérablement de tests de conformité, améliorant à la fois les coûts et les délais de mise sur le marché.12.[6]

Figure 1.8. la carte raspberry pi 4 model b

1.4.3 La camera raspberry pi

Les modules de caméra PI de framboise sont des produits officiels de la Fondation Raspberry Pi. Le modèle d'origine de 5 mégapixels a été publié en 2013 et un module de caméra de 8 mégapixels V2 a été publié en 2016. Pour notre projet, nous utiliserons la caméra de 5 mégapixels car elle répond déjà à toutes les exigences nécessaires à notre projet, il a une résolution de capteur de 2592 x 1944 pixels et un mode vidéo de 640 x 480p60 / 90, ce qui signifie qu'il convient aux applications en temps réel.[7]

Figure 1.9. module camera raspberry pi.

1.5 Conclusion

Ce chapitre a été consacré à l'histoire de l'intelligence artificielle et à ses utilisations ainsi qu'aux plaques d'immatriculation et au matériel utilisé dans notre Projet.

Dans le prochain chapitre, nous allons détailler le traitement d'image et les modèles de deep learning développés (CNN , SSD) .

Chapitre 2 Traitement d'image et apprentissage profond

2.1 Introduction

Dans ce chapitre, nous allons présenter et expliquer tous les algorithmes et les méthodes utilisés.

Nous allons tout d'abord commencer par définir le traitement d'image puis nous introduirons OpenCV qui est une bibliothèque graphique dont certaines fonctions ont été utilisées pour notre projet.

L'apprentissage profond est ensuite défini, à savoir le fonctionnement théorique des réseaux de neurones, la structure des réseaux de neurones à convolution (CNN) ainsi que l'algorithme Single Shot Multibox Detector (SSD) .

2.2 Traitement d'image

2.2.1 définition de l'image

On commence d'abord par comprendre les images. Une image, telle que les humains la voient, est une grille à deux dimensions avec chaque cellule de la grille remplie d'une valeur de couleur, autrement appelée valeur de pixel. Chaque cellule de la grille est formellement appelée un élément d'image (généralement abrégé en pixel). Un ordinateur voit également l'image de la même manière. Une image sur un ordinateur est une matrice bidimensionnelle de nombres, chaque cellule de la matrice stockant la ou les valeurs de pixel correspondantes dans l'image.[9]

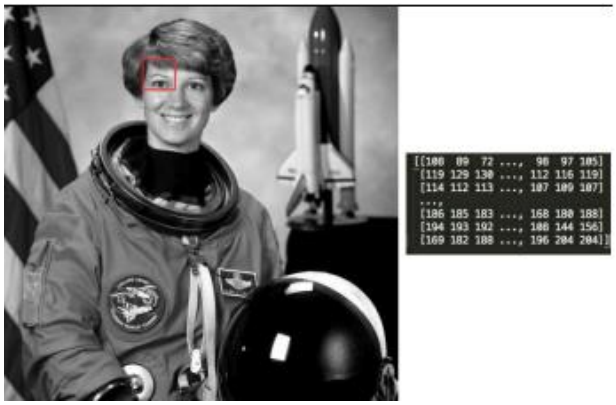


Figure 2.1.1. Stockage d'une image sur un ordinateur.

2.2.2 Définition

Le principe du traitement d'image est d'utiliser les différentes propriétés d'une image telles que la couleur, les co-relations entre les différents pixels, les placements d'objets et d'autres détails fins pour extraire des informations significatives telles que les bords, les objets et les contours, qui sont formellement appelés caractéristiques d'image. Ces fonctionnalités peuvent ensuite être utilisées dans différentes applications telles que la médecine, la sécurité, les services de médias sociaux et les voitures autonomes.[9]

2.2.3 pixels

Le pixel, ou px, est le plus petit élément d'une image numérique. Il est disposé en grille avec d'autres pixels, sur un écran d'ordinateur ou de téléphone portable par exemple. La combinaison d'un grand nombre de pixels crée une image matricielle.[25]

2.2.4 Espaces et canaux colorimétriques

Pour qu'un pixel puisse représenter une couleur spécifique, il est composé de sous-pixels aux couleurs rouge, verte et bleue (RVB). Ces sous-pixels peuvent avoir des formes différentes afin de créer une image avec la meilleure résolution et le moins d'espace possible entre les éléments.[25]

Les différents espaces colorimétriques existent parce qu'ils présentent des informations sur les couleurs de manière à faciliter certains calculs ou parce qu'ils permettent d'identifier les couleurs de manière plus intuitive. Par exemple, l'espace colorimétrique RVB définit une couleur comme les pourcentages de teintes rouges, vertes et bleues mélangées. D'autres modèles de couleurs décrivent les couleurs par leur teinte (nuance de couleur), leur saturation (quantité de gris ou de couleur pure) et leur luminance (intensité ou luminosité globale).[26]

La boîte à outils permet de convertir les données de couleur d'un espace colorimétrique à un autre grâce à des transformations mathématiques.

RVB

L'espace colorimétrique RVB représente les images sous la forme d'un tableau numérique m par n par 3 dont les éléments spécifient les valeurs d'intensité des canaux de couleur rouge, vert et bleu. La plage de valeurs numériques dépend du type de données de l'image.

- Pour les tableaux simples ou doubles, les valeurs RVB vont de [0, 1].
- Pour les tableaux uint8, les valeurs RVB vont de [0, 255].
- Pour les tableaux uint16, les valeurs RVB vont de [0, 65535].

La boîte à outils prend en charge les variations de l'espace colorimétrique RVB.[26]

TSL

L'espace colorimétrique TSL (teinte, Saturation, Lumière) correspond mieux à la façon dont les gens perçoivent la couleur que l'espace colorimétrique RVB. Par exemple, cet espace colorimétrique est souvent utilisé par les personnes qui sélectionnent des couleurs, telles que la couleur de la peinture ou de l'encre, à partir d'une roue ou d'une palette de couleurs.[26]

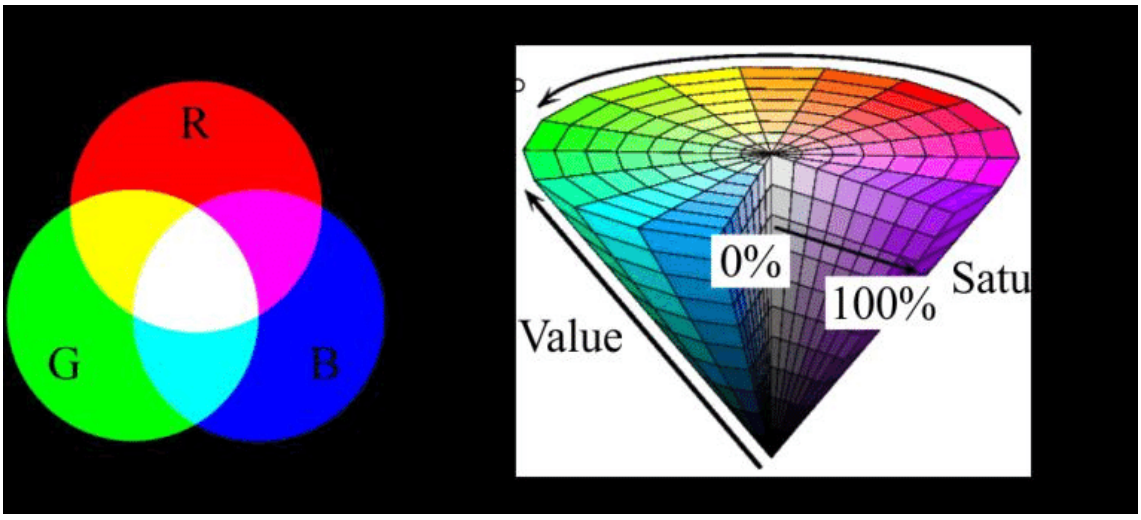


Figure 2.1.2. Exemple des espaces RVB et TSL

2.2.5 Niveaux de gris

C'est l'un des espaces de couleur les plus simples en terme de compréhension et stockage sur un ordinateur. Chaque valeur de pixel dans une image en niveaux de gris est une valeur unique comprise entre 0 et 255, 0 représentant le noir et 255 représentants le blanc. La valeur 255 n'est pas une valeur fixe, mais dépend de la profondeur de l'image. Les images en niveaux de gris sont aussi parfois appelées images en noir et blanc, mais elles ne sont pas entièrement exactes. Une image en noir et blanc signifie que les valeurs de pixels ne peuvent être que 0 ou 255 et rien entre les deux [9].

Figure 2.1.3. Exemple d'image en niveaux gris

Figure 2.1.4. *Gradient d'image montrant les valeurs de pixels allant du noir (0) au blanc (255).*

2.2.6 Notion d'histogramme

L'histogramme d'une image mesure la distribution des niveaux de gris dans l'image. Pour un niveau de gris x, l'histogramme permet de connaitre la probabilité de tomber sur un pixel de valeur x en tirant un pixel au hasard dans l'image.[27]

Concrètement, l'histogramme d'une image à valeurs entières est construit de la manière suivante: pour chaque niveau de gris x, on compte le nombre de pixels ayant la valeur x.

Par exemple, soit l'image de 5 pixels par 5 pixels de côté avec des valeurs comprises entre 0 et 4 :

0	1	2	2	3
0	1	2	2	3
0	1	2	2	4
0	1	2	2	4
0	1	2	2	4

Figure 2.1.5. matrice 5 x 5

Son histogramme est une fonction qui, à chaque valeur de niveau de gris compris entre 0 et 4, associe le nombre de pixels ayant cette valeur:

Valeur de niveau de gris	0	1	2	3	4
Nombre de pixels	5	5	10	2	3

Figure 2.1.6. *Valeur de niveau de gris / nombre de pixels.*

Où bien, sous forme de diagramme baton :

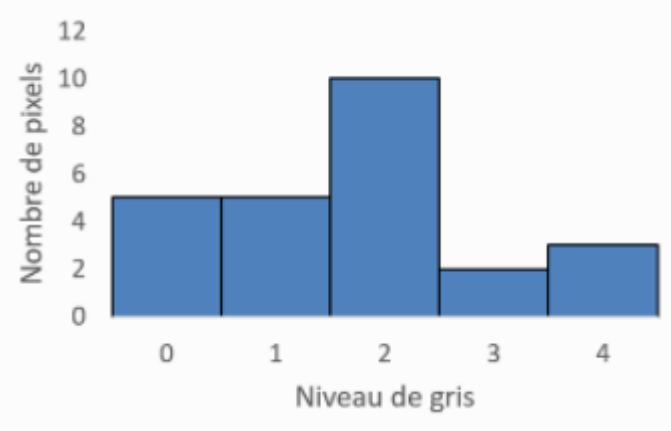


Figure 2.1.7. *Valeur de niveau de gris / nombre de pixels.*

2.2.7 Seuillage

Le seuillage d'image est la méthode la plus simple de segmentation d'image. À partir d'une image en niveau de gris, le seuillage d'image peut être utilisé pour créer une image comportant uniquement deux valeurs, noir ou blanc.

L'exemple ci-dessous montre un exemple de seuil combiné reprenant les effets des deux exemples ci-dessus.

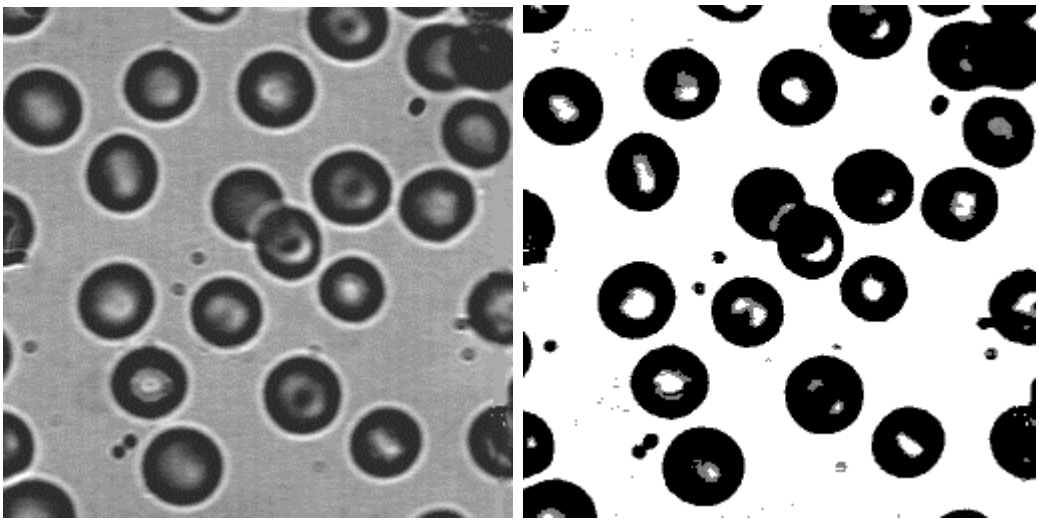


Figure 2.1.10. *Seuil combiné reprenant les effets des deux seuil haut et bas*

2.2.8 OpenCV 

a définition

OpenCV est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel.

La bibliothèque OpenCV a rencontré une disposition de nombreuses fonctionnalités très diversifiées permettant de créer des programmes en partant des données brutes pour aller jusqu'à la création d'interfaces graphiques basiques.

OpenCV a été lancé chez Intel en 1999 par Gary Bradsky, et la première version est sortie en 2000. Vadim Pisarevsky a rejoint Gary Bradsky pour gérer l'équipe russe de logiciels OpenCV d'Intel. En 2005, OpenCV a été utilisé sur Stanley, le véhicule qui a remporté le DARPA Grand Challenge 2005. Plus tard, son développement actif s'est poursuivi avec le soutien de Willow Garage avec Gary Bradsky et Vadim Pisarevsky à la tête du projet. OpenCV prend désormais en charge une multitude d'algorithmes liés à la vision par ordinateur et à l'apprentissage automatique et se développe de jour en jour.

OpenCV prend en charge une grande variété de langages de programmation tels que C++, Python, Java, etc., et est disponible sur différentes plates-formes, notamment Windows, Linux, OS X, Android et iOS. Des interfaces pour les opérations GPU à haute vitesse basées sur CUDA et OpenCL sont également en cours de développement.

OpenCV-Python est l'API Python pour OpenCV, combinant les meilleures qualités de l'API OpenCV C++ et du langage Python.

b VideoCapture

C'est la Classe pour la capture vidéo à partir de fichiers vidéo, de séquences d'images ou de caméras. La classe fournit une API C++ pour capturer des vidéos à partir de caméras ou pour lire des fichiers vidéo et des séquences d'images.
Cette classe sera notre classe principale pour la capture d'image en temps réel.

```python
# importer la bibilothéque opencv
import cv2

# definir un capteur ou webcam (dans ce cas la webcam)
vid = cv2.VideoCapture(0)

while(True):

    # Capture de chaque image dans la video
    ret, image = vid.read()

    # afficher l'image dans chaque lecture
    cv2.imshow('image', image)

    # le bouton 'q' est défini comme
    # bouton d'arrêt, vous pouvez utiliser n'importe quel
    # bouton souhaité de votre choix
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Après la boucle, relâchez l'objet capturé
vid.release()
# detruire toutes les fenétre
cv2.destroyAllWindows()
```

Figure 2.1.11. Implémentation de la classe en python.

c cvtColor

Cette classe convertit une image d'un espace colorimétrique à un autre.
La fonction convertit une image d'entrée d'un espace colorimétrique à un autre. Dans le cas d'une transformation vers-depuis l'espace colorimétrique RVB, l'ordre des canaux doit être spécifié explicitement (RVB ou BVR). Notez que le format de couleur par défaut dans OpenCV est souvent appelé RVB mais il s'agit en fait de BVR (les octets sont inversés). Ainsi, le premier octet d'une image couleur standard (24 bits) sera un composant bleu 8 bits, le deuxième octet sera vert et le troisième octet sera rouge. Les quatrième, cinquième et sixième octets seraient alors le deuxième pixel (bleu, puis vert, puis rouge), et ainsi de suite.
Les plages conventionnelles pour les valeurs des canaux R, V et B sont :

0 à 255 pour les images CV_8U
0 à 65535 pour les images CV_16U
0 à 1 pour les images CV_32F

En cas de transformations linéaires, la plage n'a pas d'importance. Mais dans le cas d'une transformation non linéaire, une image RVB d'entrée doit être normalisée à la plage de valeurs appropriée pour obtenir les résultats corrects.[10]

Dans notre cas nous procédons à une conversion vers l'échelle de gris (grayscale), à une transformation dans l'espace RVB telles que l'ajout/suppression du canal alpha, l'inversion de l'ordre des canaux, la conversion vers/depuis la couleur RVB 16 bits (R5:G6:B5 ou R5:G5:B5), ainsi que la conversion vers/depuis les niveaux de gris en utilisant les expressions suivantes:

RGB[A] to Gray:Y←0.299·R+0.587·G+0.114·B (1)

et

Gray to RGB[A]:R←Y,G←Y,B←Y,A←max(ChannelRange) (2)

```
# importer la bibilothéque opencv
import cv2

# lire l'image
image = cv2.imread('image.jpg')

#convertire a l'echelle gris
gray = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)

#afficher l'image
cv2.imshow('gray',gray)
cv2.waitKey(0)
```

Figure 2.1.12. code de conversion échelle gris en python

Figure 2.1.13. conversion échelle gris

d Flou

Fonctionnement du flou d'une image par la méthode du flou gaussien :
Le lissage, également appelé flou, est une opération de traitement d'image simple et fréquemment utilisée pour réduire le bruit dans une image. Il existe de nombreuses raisons pour le lissage. Dans ce qui suit nous allons nous concentrer sur le lissage afin de réduire le bruit [10]

Pour effectuer une opération de lissage nous allons appliquer un filtre à notre image. Le type de filtres le plus courant est linéaire, dans lequel la valeur d'un pixel de sortie (c'est-à-dire g(i,j)) est déterminée comme une somme pondérée des valeurs de pixel d'entrée (c'est-à-dire f(i+k,j+l)) :

$g(i,j)=\sum k,l f(i+k,j+l)h(k,l)$ (3)

où h(k,l) est appelé le noyau, qui n'est rien de plus que les coefficients du filtre.

Il permet de visualiser un filtre comme une fenêtre de coefficients glissant sur l'image.
Il existe de nombreux types de filtres, nous citerons ici les plus utilisés :
Le Filtre de boîte normalisé [10]

Chaque pixel de sortie est la moyenne de ses voisins du noyau (tous contribuent avec des poids égaux). Le noyau est ci-dessous :

$$K = \frac{1}{K_{width} \cdot K_{height}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ . & . & . & \dots & 1 \\ . & . & . & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

Le Filtre médian

Le filtre médian parcourt chaque élément du signal (dans ce cas l'image) et remplace chaque pixel par la médiane de ses pixels voisins (situés dans un voisinage carré autour du pixel évalué).

```
1   # importer la bibilothéque opencv
2   import cv2
3
4   # lire l'image
5   image = cv2.imread('image.jpg')
6
7   #flou d'image
8   blur = cv2.medianBlur(image , 5)
9
10  #afficher l'image
11  cv2.imshow('blur',blur)
12  cv2.waitKey(0)
```

Figure 2.1.14.code du *flou d'une image*

Figure 2.1.15. *flou d'une image*

e Detection de contours

Le détecteur de contours Canny est un opérateur de détection de contours qui utilise un algorithme à plusieurs étapes pour détecter une large gamme de contours dans les images. Il a été développé par John F. Canny

Il s'agit d'un algorithme en plusieurs étapes :

1. Réduction du bruit

Étant donné que la détection des contours est sensible au bruit dans l'image, la première étape consiste à supprimer le bruit dans l'image avec un filtre gaussien 5x5.[10]

2. Trouver le gradient d'intensité de l'image

L'image lissée est ensuite filtrée avec un noyau de Sobel dans les directions horizontale et verticale pour obtenir la première dérivée dans les directions horizontale (Gx) et verticale (Gy). À partir de ces deux images, nous pouvons trouver le gradient et la direction des bords pour chaque pixel comme suit :

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$
$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \quad (4)$$

La direction du gradient est toujours perpendiculaire aux bords. Il est arrondi à l'un des quatre angles représentant les directions verticale, horizontale et deux diagonales.

3. Suppression non maximale

Après avoir obtenu l'amplitude et la direction du gradient, une analyse complète de l'image est effectuée pour supprimer tous les pixels indésirables qui pourraient ne pas constituer le bord. Pour cela, à chaque pixel, le pixel est vérifié s'il s'agit d'un maximum local dans son voisinage dans le sens du gradient:

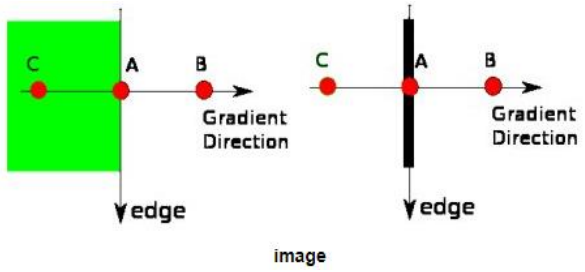
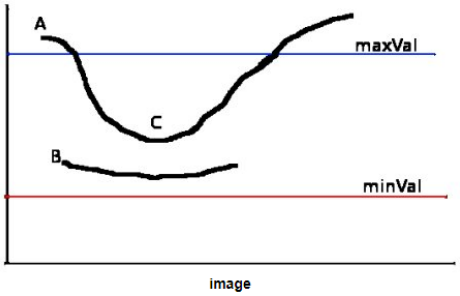


Figure 2.1.16. *Suppression non maximale*

4. Seuil d'hystérésis

Cette étape décide lesquels sont tous des bords et lesquels ne le sont pas. Pour cela, nous avons besoin de deux valeurs de seuil, minVal et maxVal. Tous les bords avec un gradient d'intensité supérieur à maxVal sont sûrs d'être des bords et ceux en dessous de minVal sont

sûrs d'être des non-bords, donc rejetés. Ceux qui se situent entre ces deux seuils sont classés arêtes ou non arêtes en fonction de leur connectivité. S'ils sont connectés à des pixels "à bords sûrs", ils sont considérés comme faisant partie des bords. [10], sinon, ils sont également rejetés. Voir l'image ci-dessous :



Le bord A est au-dessus du maxVal, donc considéré comme "sur-edge". Bien que l'arête C soit inférieure à maxVal, elle est connectée à l'arête A, de sorte qu'elle est également considérée comme une arête valide et nous obtenons cette courbe complète. Mais l'arête B, bien qu'elle soit au-dessus de minVal et se trouve dans la même région que celle de l'arête C, elle n'est connectée à aucune « arête sûre », de sorte qu'elle est rejetée. Il est donc très important de sélectionner minVal et maxVal en conséquence pour obtenir le résultat correct.
Cette étape supprime également les bruits de petits pixels en supposant que les bords sont de longues lignes.
Donc, ce que nous obtenons finalement, ce sont des bords forts dans l'image.[10]

Détection Canny Edge dans OpenCV

OpenCV met tout ce qui précède dans une seule fonction, cv.Canny(). Nous allons voir comment l'utiliser. Notre première entrée de données est notre image d'entrée. La deuxième et la troisième entrée sont respectivement nos minVal et maxVal. La quatrième entrée est la taille_ouverture. C'est la taille du noyau de Sobel utilisé pour trouver les dégradés d'images. Par défaut, c'est 3. Le dernier argument est le gradient qui spécifie l'équation pour trouver la magnitude du gradient. S'il est vrai, il utilise l'équation mentionnée ci-dessus qui est plus précise, sinon il utilise cette fonction : Edge_Gradient(G)=|Gx|+|Gy|. Par défaut, c'est Faux.[10]

```
1   # importer la bibilothéque opencv
2   import cv2
3
4   # lire l'image
5   image = cv2.imread('image.jpg')
6
7   # on converti a l'echelle gris
8   gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
9
10  #on utiliser canny edge detection
11  canny = cv2.Canny(gray,100,200)
12
13  #afficher l'image
14  cv2.imshow('canny',canny)
15  cv2.waitKey(0)
```

Figure 2.1.17. *détecter canny avec code et résultat*

f thresholding

Pour chaque pixel, la même valeur de seuil est appliquée. Si la valeur du pixel est inférieure au seuil, elle est fixée à 0, sinon elle est fixée à une valeur maximale. La fonction cv.threshold permet d'appliquer le seuillage. Le premier argument est l'image source, qui doit être une image en niveaux de gris. Le deuxième argument est la valeur seuil qui est utilisée pour classer les valeurs de pixels. Le troisième argument est la valeur maximale qui est attribuée aux valeurs de pixels dépassant le seuil. OpenCV fournit différents types de seuillage qui sont donnés par le quatrième paramètre de la fonction. Le seuillage de base tel que décrit ci-dessus est effectué en utilisant le type cv.THRESH_BINARY. Tous les types de seuillage simples sont :

- cv.THRESH_BINARY
- cv.THRESH_BINARY_INV
- cv.THRESH_TRUNC
- cv.THRESH_TOZERO
- cv.THRESH_TOZERO_INV

La méthode renvoie deux sorties. La première est le seuil qui a été utilisé et la seconde sortie est l'image souillée.[10]
Ce code compare les différents types de seuillage simple :

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('sudoku.png',0)
img = cv.medianBlur(img,5)

ret,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
th2 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_MEAN_C,\
            cv.THRESH_BINARY,11,2)
th3 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,\
            cv.THRESH_BINARY,11,2)

titles = ['Original Image', 'Global Thresholding (v = 127)',
            'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]

for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

Pour tracer plusieurs images, nous avons utilisé la fonction plt.subplot(). [ ] :

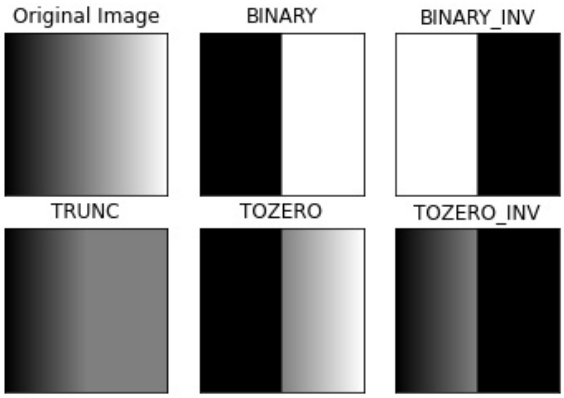


Figure 2.1.18. *code et exemple de tout les seullies dans le code*

- ***Binarisation d'Otsu***

Dans le seuillage global, nous avons utilisé une valeur choisie arbitrairement comme seuil. En revanche, la méthode d'Otsu évite d'avoir à choisir une valeur et la détermine automatiquement.

Considérons une image avec seulement deux valeurs d'image distinctes (image bimodale), où l'histogramme ne serait constitué que de deux pics. Un bon seuil serait au milieu de ces deux valeurs. De même, la méthode d'Otsu détermine une valeur seuil globale optimale à partir de l'histogramme de l'image.[10]

Pour ce faire, la fonction cv.threshold() est utilisée, où cv.THRESH_OTSU est passé comme indicateur supplémentaire. La valeur seuil peut être choisie arbitrairement. L'algorithme trouve alors la valeur de seuil optimale qui est renvoyée comme première sortie.[10]

L'exemple ci-dessous donne l'image d'entrée qui est une image bruitée. Dans le premier cas, un seuillage global avec une valeur de 127 est appliqué. Dans le second cas, le seuillage d'Otsu est appliqué directement. Dans le troisième cas, l'image est d'abord filtrée avec un noyau gaussien 5x5 pour supprimer le bruit, puis un seuillage Otsu est appliqué. On remarque que le filtrage du bruit améliore le résultat.

```python
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('noisy2.png',0)

# global thresholding
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)

# Otsu's thresholding
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

# Otsu's thresholding after Gaussian filtering
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

# plot all the images and their histograms
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
          'Original Noisy Image','Histogram',"Otsu's Thresholding",
          'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]

for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```

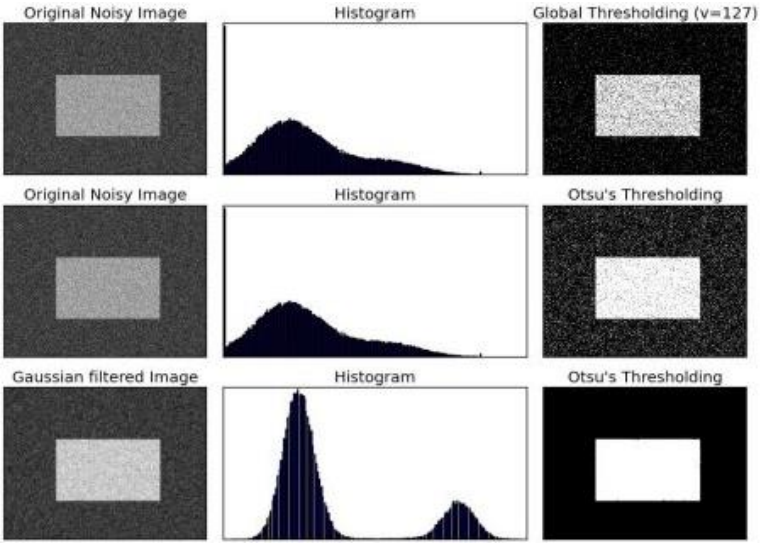


Figure 2.1.19. *Code et exemple de tous les seuils otsu*

g contours

Les contours peuvent être expliqués simplement comme une courbe joignant tous les points continus (le long de la frontière), ayant la même couleur ou intensité. Les contours sont un outil utile pour l'analyse de forme et la détection et la reconnaissance d'objets.

- Pour une meilleure précision, on utilise des images binaires. Donc, avant de trouver des contours, on applique une détection de seuil ou de contour astucieuse.

- Depuis OpenCV 3.2, findContours() ne modifie plus l'image source mais renvoie une image modifiée comme premier des trois paramètres de retour.
- Dans OpenCV, trouver des contours revient à trouver un objet blanc sur fond noir. donc l'objet à trouver doit être blanc et l'arrière-plan doit être noir.

Voyons comment trouver les contours d'une image binaire :

```
import numpy as np
import cv2 as cv

im = cv.imread('test.jpg')
imgray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(imgray, 127, 255, 0)
im2, contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
```

On peut voir trois arguments dans la fonction cv.findContours(), le premier est l'image source, le second est le mode de récupération de contour, et le troisième est la méthode d'approximation de contour. Il en ressort une image modifiée, les contours et la hiérarchie. Contours est une liste Python de tous les contours de l'image. Chaque contour individuel est un tableau Numpy de coordonnées (x,y) des points limites de l'objet.

Pour dessiner les contours, la fonction cv.drawContours est utilisée. Cette fonction peut également être utilisée pour dessiner n'importe quelle forme à condition que vous ayez ses points de délimitation. Son premier argument est l'image source, le deuxième argument sont les contours qui doivent être passés en tant que liste Python, le troisième argument est l'index des contours (utile pour dessiner un contour individuel) et les arguments restants sont la couleur, l'épaisseur etc.

- Pour dessiner tous les contours d'une image :

cv.drawContours(img, contours, -1, (0,255,0), 3)

- Pour dessiner un contour individuel, disant le 4ème contour :

cv.drawContours(img, contours, 3, (0,255,0), 3)

- Mais la plupart du temps, la méthode ci-dessous sera utile:

cnt = contours[4]
cv.drawContours(img, [cnt], 0, (0,255,0), 3)

- **Méthode d'approximation des contours**

C'est le troisième argument de la fonction cv.findContours. Nous avons dit au préalable que les contours sont les limites d'une forme avec la même intensité. Il stocke les coordonnées (x,y) de la limite d'une forme. Mais stocke-t-il toutes les coordonnées ? Cela est spécifié par cette méthode d'approximation de contour. cv.CHAIN_APPROX_NONE, stockent tous les points limites Mais cv.CHAIN_APPROX_SIMPLE supprime tous les points redondants et compresse le contour, économisant ainsi de la mémoire.
La Figure 2.1.9 d'un rectangle illustre cette technique. On trace simplement un cercle sur toutes les coordonnées du tableau de contours (dessiné en bleu). La première image montre

les points obtenus avec cv.CHAIN_APPROX_NONE (734 points) et la deuxième image montre celui avec cv.CHAIN_APPROX_SIMPLE (seulement 4 points).[10]

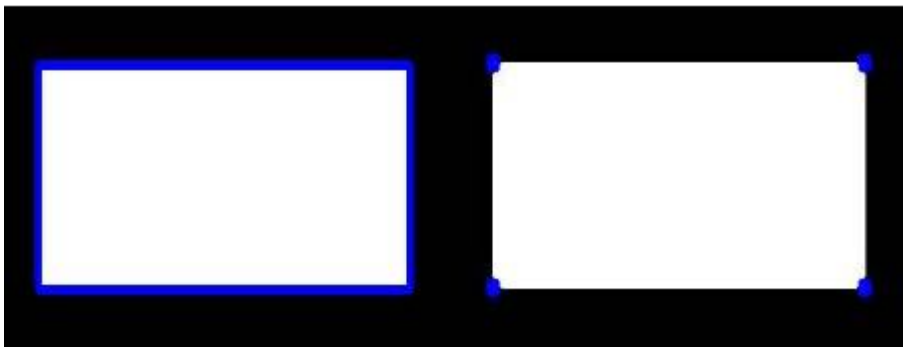


Figure 2.1.20. *exemple de contour de la fonction*

h Approximation des contours

Il rapproche une forme de contour d'une autre forme avec moins de sommets en fonction de la précision que nous spécifions. C'est une implémentation de l'algorithme de Douglas-Peucker. [10]

on suppose qu'on essaye de trouver un carré dans une image, mais en raison de certains problèmes dans l'image, on n'obtient pas un carré parfait, mais une "mauvaise forme", comme indiqué dans la première image de la figure 2.1.20. On peut maintenant utiliser cette fonction pour approximer la forme. Le deuxième argument est appelé epsilon, qui est la distance maximale du contour au contour approximé. C'est un paramètre de précision. Une sélection judicieuse d'epsilon est nécessaire pour obtenir la sortie correcte.

```
epsilon = 0.1*cv.arcLength(cnt,True)
approx = cv.approxPolyDP(cnt,epsilon,True)
```

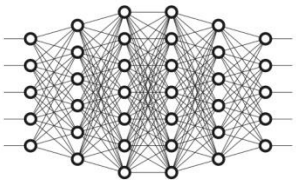
Ci-dessous, dans la deuxième image, la ligne verte montre la courbe approximative pour epsilon = 10 % de la longueur de l'arc. La troisième image montre la même chose pour epsilon = 1% de la longueur de l'arc. Le troisième argument spécifie si la courbe est fermée ou non.

Figure 2.1.21. *Exemple de détection a approximation*

2.3 Deep Learning

2.3.1 définition

L'apprentissage en profondeur est une approche de l'apprentissage automatique caractérisée par de profonds empilements de calculs. Cette profondeur de calcul est ce qui a permis aux modèles d'apprentissage en profondeur de résoudre les types de modèles complexes et hiérarchiques trouvés dans les ensembles de données du monde réel les plus difficiles.



Grâce à leur puissance et leur évolutivité, les réseaux de neurones sont devenus le modèle déterminant de l'apprentissage en profondeur. Les réseaux de neurones sont composés de neurones, où chaque neurone n'effectue individuellement qu'un simple calcul. La puissance d'un réseau de neurones vient plutôt de la complexité des connexions que ces neurones peuvent former. [11]

2.3.2 Les réseaux de neurone artificiels

Le réseau de neurones artificiels (ANN) est un algorithme d'apprentissage en profondeur qui a émergé et a évolué à partir de l'idée de réseaux de neurones biologiques du cerveau humain. Une tentative de simuler le fonctionnement du cerveau humain a abouti à l'émergence de l'algorithme ANN. Ce dernier fonctionne de manière très similaire aux réseaux de neurones biologiques mais ne ressemble pas exactement à son fonctionnement.

L'algorithme ANN accepte uniquement des données numériques et structurées en entrée. Pour accepter les formats de données non structurés et non numériques tels que l'image, le texte et la parole, des réseaux de neurones convolutifs (CNN) et des réseaux de neurones récursifs (RNN) sont utilisés respectivement. Dans ce qui suit nous nous concentrons uniquement sur les réseaux de neurones artificiels.[11]

- Un réseau de neurones à une seule couche s'appelle un perceptron, figure 2.2.1. Un perceptron multicouche est appelé réseaux de neurones artificiels.
- Un réseau de neurones peut posséder un nombre quelconque de couches. Chaque couche peut avoir un ou plusieurs neurones ou unités. Chacun des neurones est interconnecté avec chacun des autres neurones. Chaque couche pourrait également avoir des fonctions d'activation différentes.
- L'ANN se compose de deux phases de propagation vers l'avant et de rétropropagation. La propagation vers l'avant consiste à multiplier les poids, à ajouter un biais, à appliquer une fonction d'activation aux entrées et à la propager vers l'avant.
- L'étape de rétropropagation est l'étape la plus importante qui consiste généralement à trouver des paramètres optimaux pour le modèle en se propageant dans le sens

inverse des couches du réseau de neurones. La rétropropagation nécessite une fonction d'optimisation pour trouver les poids optimaux pour le modèle.

- L'ANN peut être appliqué aux tâches de régression et de classification en modifiant les fonctions d'activation des couches de sortie. (Fonction d'activation sigmoïde pour la classification binaire, fonction d'activation Softmax pour la classification multi-classe et fonction d'activation linéaire pour la régression)

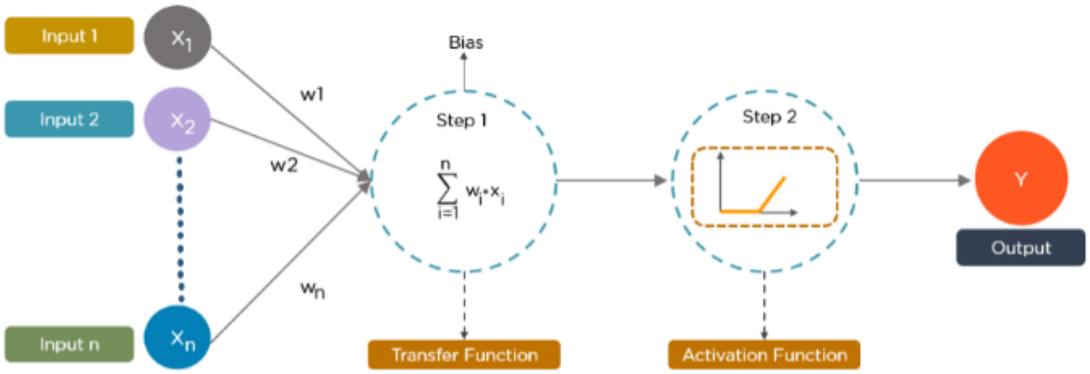


Figure 2.2.1. *un exemple du fonctionnement du perceptron*

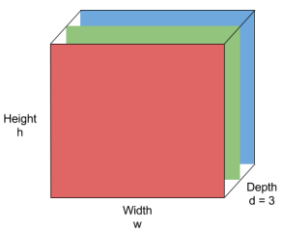
2.3.3 Les réseaux de neurones convolutifs (CNN)

a définition

Un réseau de neurones convolutifs (CNN ou ConvNet) est une architecture de réseau pour l'apprentissage en profondeur qui apprend directement à partir des données, éliminant ainsi le besoin d'extraction manuelle de fonctionnalités.
Les CNN sont particulièrement utiles pour trouver des motifs dans les images afin de reconnaître des objets, des visages et des scènes. Ils peuvent également être très efficaces pour classer les données non-image telles que les données audio, les séries temporelles et les signaux.
Les applications qui font appel à la reconnaissance d'objets et à la vision par ordinateur, telles que les véhicules autonomes et les applications de reconnaissance faciale, reposent fortement sur les CNN.

b fonctionnement

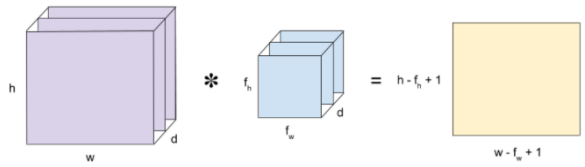
Pour les images colorées, la matrice d'image a une profondeur de 3, une pour chacun des trois canaux de couleur : (R)ed, V(vert), B(lue).



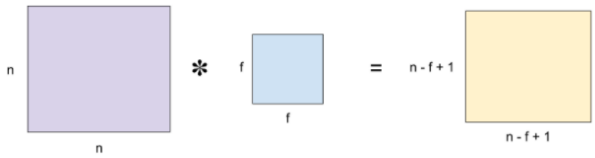
La convolution est une opération mathématique qui prend deux entrées :

1. Une matrice d'image (volume) de dimension (h x l x d)
2. Un filtre (fh x fw x d)

et génère un volume de dimension (h — fh + 1) x (w — fw + 1) x 1.



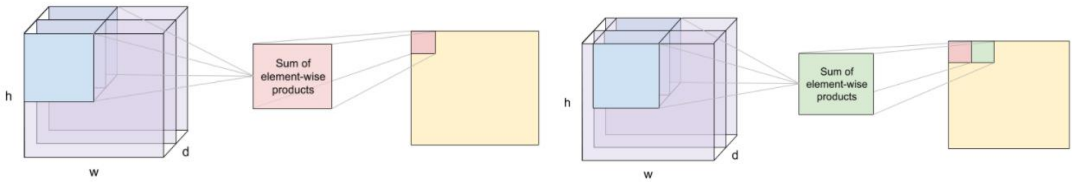
Dans un cas simplifié où d = 1, h = w = n et fh = fw = f (ce qui signifie que l'image d'entrée et le filtre sont des images en niveaux de gris au carré), la dimension du volume de sortie se simplifie en (n - f + 1) x (n — f + 1).



l La sortie de la convolution est définie comme suit :
pour le pixel du haut à gauche de la sortie :

1. Le filtre est placé dans le coin supérieur gauche de l'image d'entrée
2. Le produit est calculé élément par élément pour chaque valeur de pixel correspondante
3. Les produits sont additionnés pour obtenir la valeur de pixel en haut à gauche de la sortie

Ensuite, le filtre est déplacé d'une position vers la droite et les étapes 2 et 3 sont répétées pour obtenir la prochaine valeur du pixel de la sortie ! ces étapes vont se répéter ainsi jusqu'au balayage total de l'image d'entrée d'origine. On note que la profondeur du volume d'entrée et du filtre doivent se correspondre.[12]



Une couche de convolution est composée de filtres nf de même taille et profondeur. Pour chaque filtre, nous le convoluons avec le volume d'entrée pour obtenir nf sorties. Ensuite, les sorties sont transmises à une fonction d'activation, ReLU, par exemple. Enfin, ces sorties nf sont empilées dans un volume (h - fh + 1) x (w - fw + 1) x nf.

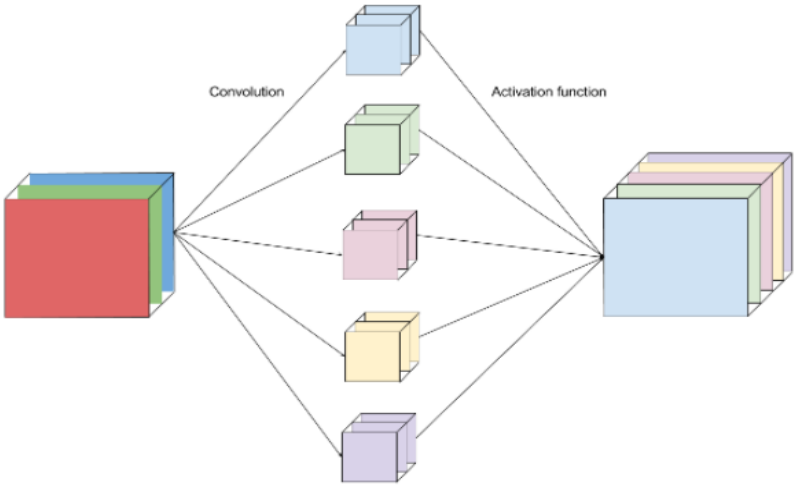


Figure 2.2.2. *un exemple du fonctionnement de la convolution*

Intuitivement, nous pouvons considérer chaque filtre comme un « capteur » pour détecter une caractéristique de l'image d'entrée. Par exemple, un filtre de détection de contour est « activé » et génère une valeur de pixel élevée.

Effectuer des convolutions sur les images au lieu de connecter chaque pixel aux unités du réseau de neurones présente deux avantages principaux :

1. Réduction du nombre de paramètres que nous devons apprendre. Au lieu d'apprendre les poids reliant chaque pixel d'entrée, nous avons seulement besoin d'apprendre les poids du filtre (qui est généralement beaucoup plus petit que l'image d'entrée).

2. Préservation de la localité. Nous n'avons pas à aplatir la matrice de l'image en un vecteur, ainsi les positions relatives des pixels de l'image sont préservées. Si on cite une image de chat par exemple, les informations qui composent lechat comprennent les positions relatives de ses yeux, de son nez et de ses oreilles.

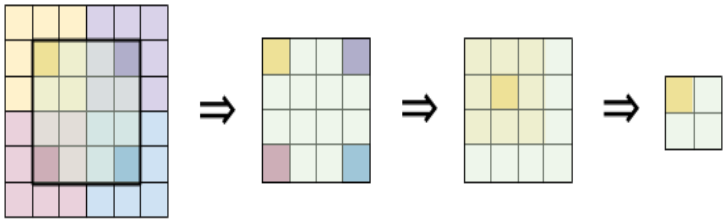
Intuitivement, nous pouvons imaginer chaque convolution d'un filtre sur une image comme une collecte et un traitement des informations sur une partie de l'image et les résumer en une valeur unique et significative.[12]

- **Padding**

Que faire si le filtre ne « s'adapte » pas parfaitement à l'image d'entrée ? Eh bien, nous avons deux options :

1. Complétez l'image avec des zéros (zéro-padding) de manière à ce qu'elle tienne, ou
2. Déposez la partie de l'image où le filtre ne tenait pas. C'est ce qu'on appelle le valid-padding, en ne gardant que la partie valide de l'image.

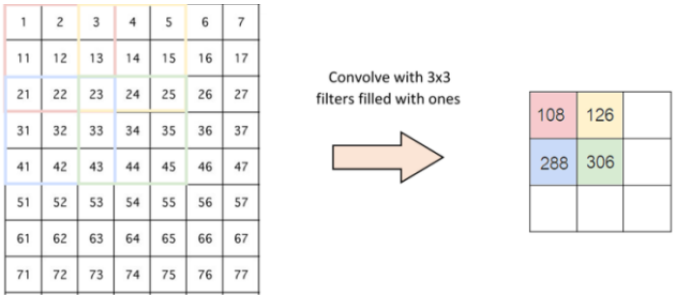
Un autre problème avec la convolution est que la dimension de l'image est réduite. Si nous passons l'entrée à travers de nombreuses couches de convolution sans remplissage, la taille de l'image diminue et devient finalement trop petite pour être utile.



La solution consiste à appliquer un remplissage de zéro à l'image de sorte que la sortie ait la même largeur et la même hauteur que l'entrée. C'est ce qu'on appelle formellement le même remplissage(same padding).[12]

- **Strides**

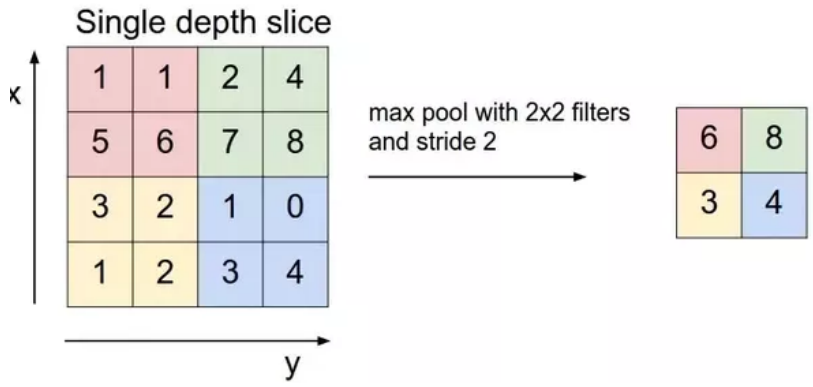
Le paramètre suivant que nous pouvons choisir pendant la convolution est connu sous le nom de foulée(Strides). La foulée signifie le nombre de pixels que notre filtre déplacera à chaque fois. La plupart des exemples que vous avez vus auparavant ont une foulée d'un. L'image ci-dessous montre comment la convolution fonctionnerait avec une foulée de 2.



Avec une foulée de 1, notre sortie ci-dessus donnerait une image 5x5. Cependant, avec une foulée de 2, nous obtiendrions plutôt une image 3x3. Pour certaines images où les valeurs de pixels proches sont assez similaires, nous n'avons pas besoin d'échantillonner chaque pixel car nous pourrions obtenir les mêmes informations avec une image plus petite.[12]

- **padding layer**

Après la couche convolutive, il est courant de transmettre ces valeurs à la couche suivante, appelée couche de mise en commun. La mise en commun applique un type spécial de filtre à votre sortie où les deux formes les plus courantes sont la mise en commun maximale et moyenne. Avec le max pooling, c'est comme si on appliquait un filtre max sur l'image. Il en est de même pour la mutualisation moyenne (average pooling).



La raison pour laquelle nous utilisons la couche de pooling est encore une fois de réduire la taille des données. Avec une mise en commun maximale ou moyenne, nous sommes en mesure de conserver les informations de l'image avec une image plus petite.. [12]

- **Couche entièrement connectée**

Après la couche de mise en commun, nous avons aplati nos données dans un vecteur et les avons introduites dans une couche entièrement connectée comme dans un réseau de neurones primitif. Cette couche a été définie pour ajouter de la non-linéarité à nos données. Pour un exemple de visage humain, si la couche convolutive pourrait être capable d'identifier des caractéristiques telles que les visages, le nez, les oreilles, etc. Cependant, la connaissance de la position ou de l'endroit où ces caractéristiques devraient se trouver n'est pas acquise . Avec les couches entièrement connectées, une combinaison de ces fonctionnalités pour créer un modèle plus complexe pourrait donner au réseau plus de puissance de prédiction quant à l'emplacement de ces fonctionnalités afin de le classer comme humain. Enfin, nous avons une fonction d'activation telle que la fonction softmax ou la fonction sigmoïde qui génère le résultat. [12]

- **Récapitulatif des réseaux CNN**:

1. On introduit l' image d'entrée dans la couche convolutive.
2. Couche convolutive. On choisit les paramètres de convolution, notamment la foulée(strides), le rembourrage(padding) et la taille du filtre. On effectue une convolution sur l'image , une activation ReLU sur l'ensemble de la matrice, puis un pooling sur la sortie pour réduire la taille. On ajoute autant de couches convolutives jusqu'à satisfaction.
3. la sortie est aplatie et alimentation de la couche entièrement connectée.
4. Sortir la classe à l'aide d'une fonction d'activation telle que softmax ou fonction sigmoïde

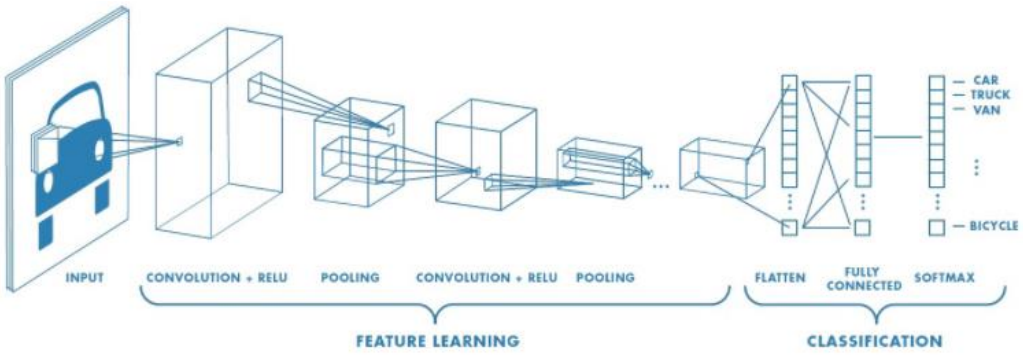


figure 2.2.3. *l'architecture d'un réseau CNN*

2.3.4 SSD (Single Shot Multibox detector)

La détection SSD Single Shot MultiBox Detector (de C. Szegedy et al.) a vu le jour en fin novembre 2016 et a atteint de nouveaux records en termes de performances et de précision pour les tâches de détection d'objets, avec un score supérieur à 74% mAP (mean Average Precision) à 59 images par seconde sur des jeux de données standard tels que PascalVOC et COCO. Pour mieux comprendre le SSD, commençons par expliquer d'où vient le nom de cette architecture : [13]

- Single Shot : cela signifie que les tâches de localisation et de classification des objets sont effectuées en un seul passage avant du réseau
- MultiBox : c'est le nom d'une technique de régression de la boîte englobante développée par Szegedy et al.
- Détecteur : le réseau est un détecteur d'objets qui classe également les objets détectés

a Architecture

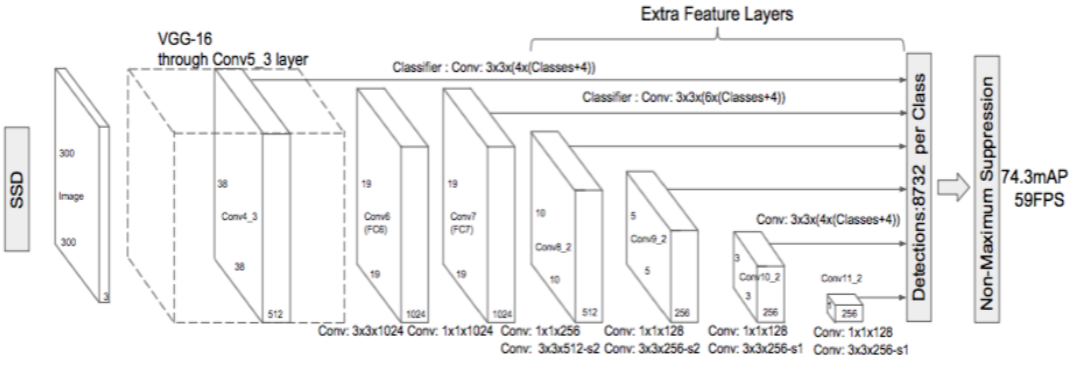


figure 2.2.4. *L'architecture SSD*

La ***figure 2.2.3.*** montre l'architecture du SSD qui s'appuie principalement sur l' architecture VGG-16, mais rejette les couches entièrement connectées. La raison pour laquelle VGG-16 a été utilisé comme réseau de base est due à ses performances élevées dans les tâches de classification d'images de haute qualité et à sa popularité pour les problèmes où

l'apprentissage par transfert aide à améliorer les résultats. Au lieu des couches VGG d'origine entièrement connectées, un ensemble de couches convolutives auxiliaires (à partir de conv6) a été ajouté, permettant ainsi d'extraire des caractéristiques à plusieurs échelles et de réduire progressivement la taille de l'entrée de chaque couche suivante.[13]

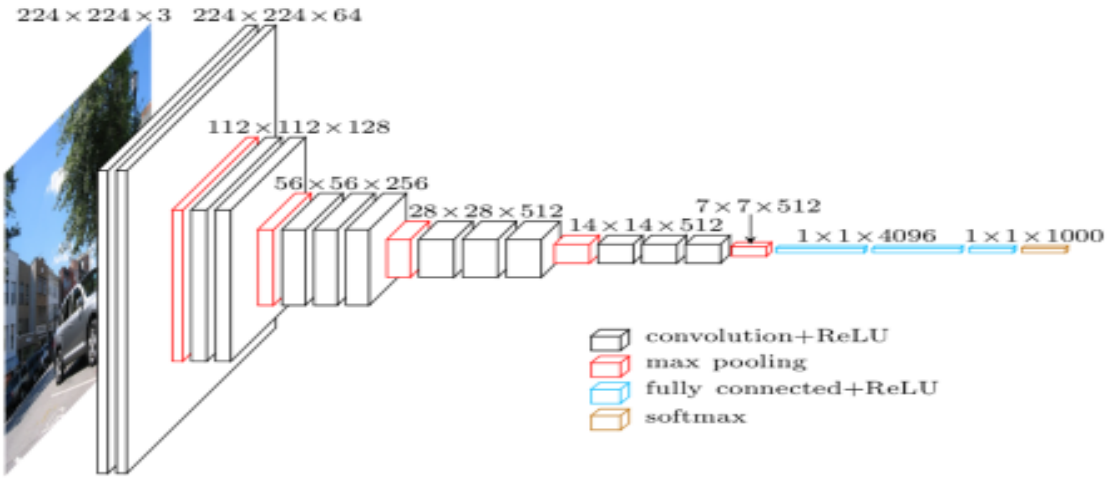


figure 2.2.5. *L'architecture VGG*

- **le Multibox**

La technique de régression de la boîte englobante de SSD est inspirée des travaux de C.Szegedy sur le MultiBox, une méthode pour des propositions de coordonnées de boîte englobante rapides et indépendantes des classes. Dans le travail effectué sur MultiBox, un réseau convolutif de style Inception est utilisé. Les convolutions 1x1 ci-dessous aident à réduire la dimensionnalité puisque le nombre de dimensions diminuera (mais la "largeur" et la "hauteur" resteront les mêmes).

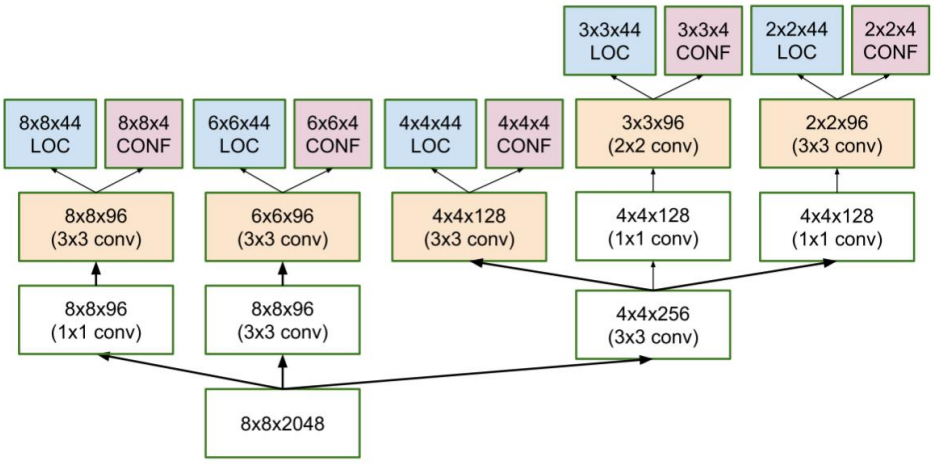


figure 2.2.6. *Architecture de prédiction convolutive multi-échelle de la localisation et des confidences du multibox*

La fonction de perte de MultiBox combinait également deux composants critiques qui se sont retrouvés dans le SSD :

- Perte de confiance(confidence loss) : cela mesure la confiance du réseau quant à l'objectivité de la boîte englobante calculée. L'entropie croisée catégorique est utilisée pour calculer cette perte.
- Perte de localisation (location loss): cela mesure à quelle distance les cadres de délimitation prévus du réseau sont éloignés de ceux de la vérité terrain de l'ensemble d'apprentissage. L2-Norm est utilisé ici.

L'expression de la perte, qui mesure la distance de la prédiction atteinte , est donc :

perte_multibox = perte_confiance + alpha * perte_localisation

Le terme alpha nous aide à équilibrer la contribution de la perte de localisation. En apprentissage profond, l'objectif est de trouver les valeurs des paramètres qui réduisent le plus efficacement la fonction de perte, rapprochant ainsi nos prédictions de la vérité terrain. [13]

- **Priors MultiBox et IoU**

Dans MultiBox, les chercheurs ont créé ce que nous appelons des priors (ou des ancres dans la terminologie Faster-R-CNN), qui sont des cadres de délimitation de taille fixe pré-calculés qui correspondent étroitement à la distribution des cadres de vérité terrain d'origine. En fait, ces priors sont sélectionnés de telle manière que leur ratio Intersection sur Union (alias IoU, et parfois appelé indice Jaccard) est supérieur à 0,5. Comme on peut le déduire de l'image ci-dessous, une IoU de 0,5 n'est toujours pas assez bonne, mais elle fournit cependant un point de départ solide pour l'algorithme de régression de la boîte englobante - c'est une bien meilleure stratégie que de commencer les prédictions avec des coordonnées aléatoires ! Par conséquent, MultiBox commence avec les à priori en tant que prédictions et tente de régresser plus près des boîtes englobantes de la vérité terrain.

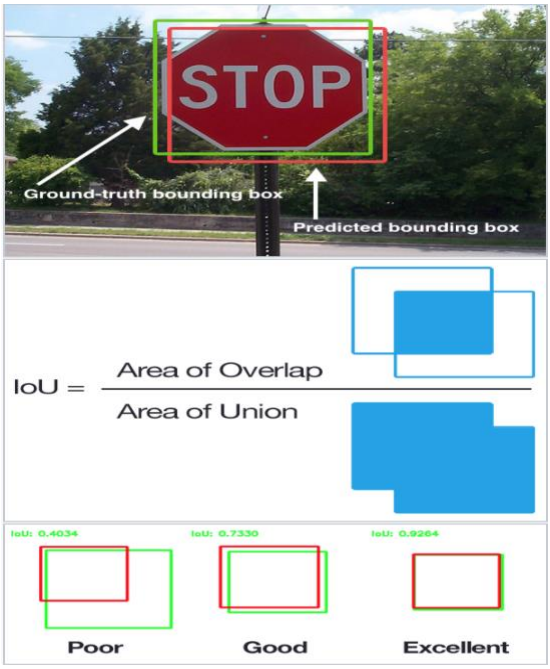


figure 2.2.7. *Diagramme expliquant l'IoU*

L'architecture résultante contient 11 priors par cellule de carte de caractéristiques (8x8, 6x6, 4x4, 3x3, 2x2) et un seul sur la carte de caractéristiques 1x1, ce qui donne un total de 1420 priors par image, permettant ainsi une couverture robuste des images d'entrée à plusieurs échelles, pour détecter des objets de différentes tailles. À la fin, MultiBox ne conserve que les K meilleures prédictions qui ont minimisé à la fois les pertes de localisation (LOC) et de confiance (CONF). [13]

- **Améliorations SSD**

Un certain nombre d'ajustements ont été ajoutés pour rendre ce réseau encore plus capable de localiser et de classer les objets.

- **Priorités fixes** : contrairement à MultiBox, chaque cellule de la carte des caractéristiques est associée à un ensemble de cadres de délimitation par défaut de différentes dimensions et proportions. Ces priors sont choisis manuellement (mais soigneusement), alors que dans MultiBox, ils ont été choisis parce que leur IoU par rapport à la vérité terrain était supérieure à 0,5. Cela devrait en théorie permettre à SSD de généraliser pour tout type d'entrée, sans nécessiter une phase de pré-apprentissage pour la génération préalable. Par exemple, en supposant que nous ayons configuré 2 points diagonalement opposés (x1, y1) et (x2, y2) pour chaque b cadres de délimitation par défaut par cellule de carte de caractéristiques, et c classes à classer, sur une carte de caractéristiques donnée de taille f = m * n, SSD calculerait les valeurs f * b * (4 + c) pour cette carte de caractéristiques.

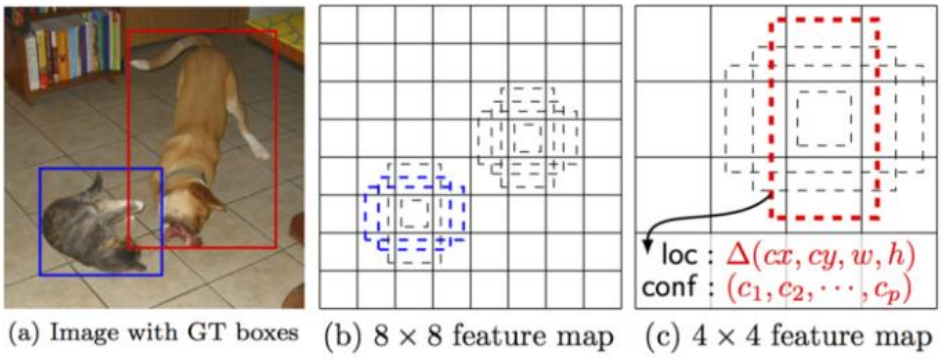


figure 2.2.8. *Boîtes SSD par défaut aux cartes de caractéristiques 8x8 et 4x4*

- Perte de localisation : le SSD utilise la norme L1 lisse pour calculer la perte de localisation. Bien qu'il ne soit pas aussi précis que L2-Norm, il est toujours très efficace et donne plus de marge de manœuvre au SSD car il n'essaie pas d'être « pixel parfait » dans sa prédiction de boîte englobante (c'est-à-dire qu'une différence de quelques pixels serait à peine perceptible pour plusieurs parmi nous).
- Classification : MultiBox n'effectue pas de classification d'objets, contrairement à SSD. Par conséquent, pour chaque boîte englobante prédite, un ensemble de prédictions de classe c est calculé, pour chaque classe possible dans l'ensemble de données.

2.4 Transfer learning

L'apprentissage par transfert (TL) est un problème de recherche en apprentissage automatique (ML) qui se concentre sur le stockage des connaissances acquises lors de la résolution d'un problème et de son application à un problème différent mais connexe. Par exemple, les connaissances acquises en apprenant à reconnaître les voitures pourraient s'appliquer en essayant de reconnaître les camions. Ce domaine de recherche a un certain lien avec la longue histoire de la littérature psychologique sur le transfert des apprentissages, bien que les liens pratiques entre les deux domaines soient limités. D'un point de vue pratique, la réutilisation ou le transfert d'informations provenant de tâches apprises précédemment pour l'apprentissage de nouvelles tâches a le potentiel d'améliorer considérablement l'efficacité de l'échantillon d'un agent d'apprentissage par renforcement. [28]

2.5 Conclusion

Ce chapitre a été consacré à l'introduction du traitement d'image avec la bibliothèque opencv, au deep learning ainsi qu'aux algorithmes utilisés dans notre projet

Le prochain chapitre sera dédié à notre travail à savoir les algorithmes utilisés, l'entrainement du modèle, et le dataset qui seront détaillés. L'implémentation dans le matériel et la base d'enregistrement feront également l'objet d'étude.

Chapitre 3 Expérimentations et résultats

3.1 Introduction

Maintenant que nous avons compris toutes les théories, c'est l'heure de passer à la pratique.

Dans ce chapitre nous allons voir l'environnement et le langage de programmation ainsi que toutes les bibliothèques utilisées et l'installation et l'initialisation du système d'exploitation du raspberry pi.

Nous allons ainsi montrer comment nous avons pu entrainer nos algorithmes avec une base de données dans Google collab (puisque qu'on ne dispose pas d'un matériel puissant), ensuite on montre comment l'algorithme fonctionne étape par étape, puis nous allons afficher tous les résultats .

Par la suite nous allons faire une étude comparative de la vitesse, de la précision et de l'erreur pour les deux algorithmes

Enfin, on a situé tous les problèmes que nous avons rencontré avec toutes les solutions que nous avons pu trouver.

3.2 Environnement de travail

Toutes les librairies et logiciel vont être installés sous raspberry pi.

3.2.1 Raspbian OS

Le Raspberry Pi OS (anciennement appelé Raspbian) est le système d'exploitation officiel pris en charge pour le Raspberry Pi. Ce système d'exploitation va être installer sur une carte microSD [1].

Pour l'installation nous aurons besoin d'une carte mémoire SD minimum 8 GB (nous utilisons celle de 32GB).

On commence tout d'abord par installer l'image du système d'exploitation par le site officiel, puis nous allons installer un logiciel qui va permettre de graver notre OS dans la carte mémoire. Ce logiciel est connu sous **BelenaEtcher** qui est un logiciel qui grave les systèmes d'exploitation dans les disques mémoires, il est simple à utiliser et gratuit ce qui facilite notre travail.

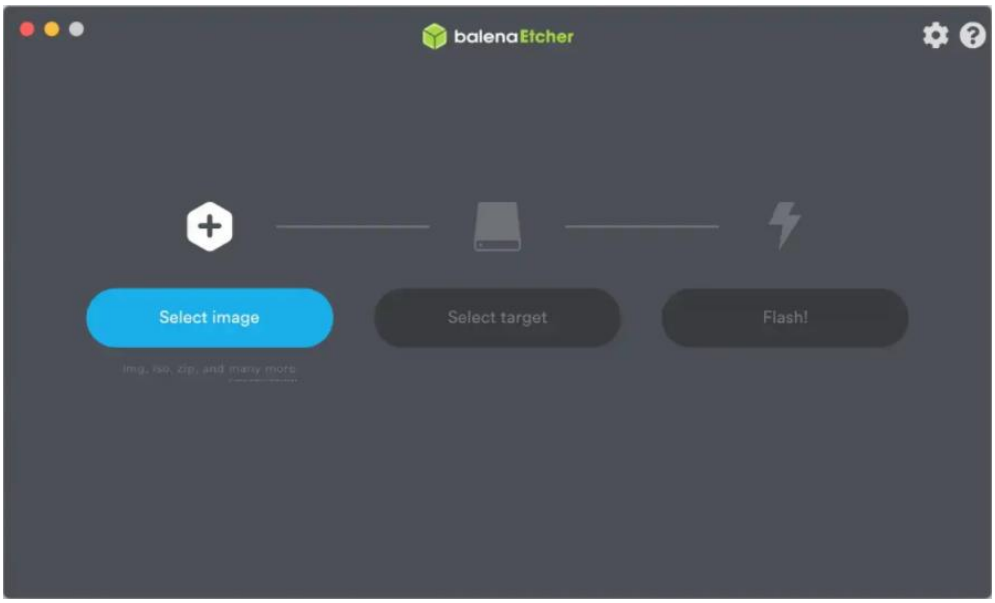


Figure 3.1. *Sélection de l'image du système d'exploitation*

Tout d'abord nous allons commencer par sélectionner l'image, puis nous allons sélectionner l'objet dont nous allons graver l'image (dans notre cas c'est une carte SD).

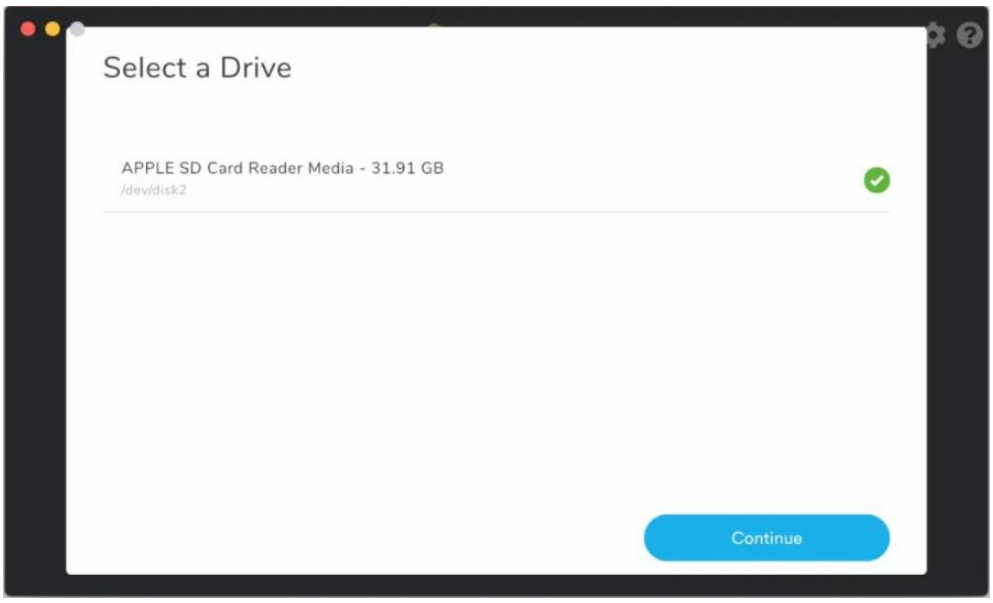


Figure 3.2. *Sélection de la carte mémoire où nous allons graver notre OS*

A la fin nous allons graver notre image dans la carte mémoire SD par un clique sur "flash", ce processus peut prendre quelques minutes.

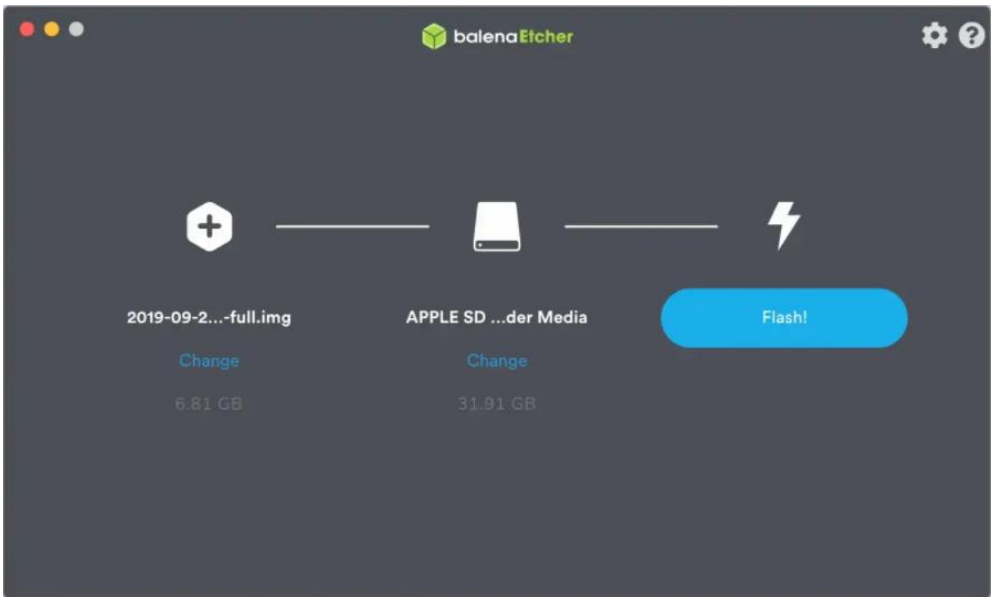


Figure 3.3. *"flash" : graver notre OS dans l'objet sélectionné*

Nous insérons notre carte SD et nous allons nous connecter à notre raspberry via le câble **Ethernet** (afin d'avoir l'adresse ip ou l'adresse DNS) et **VNC viewer (**entrer le nom DNS du dispositif **raspberry.local**). Cette connexion est faite par un réseau local (LAN).

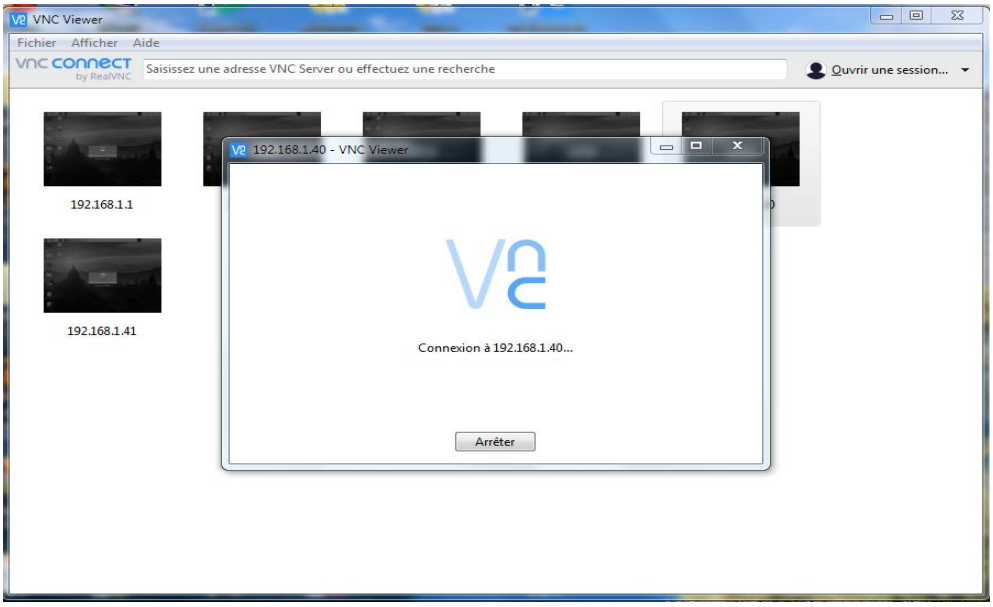


Figure 3.4. *Sélection de l'adresse IP ou DNS pour une connexion à distance*

L'image de la ***figure 3.5.nous montre le*** résultat de la connexion à notre raspberry.

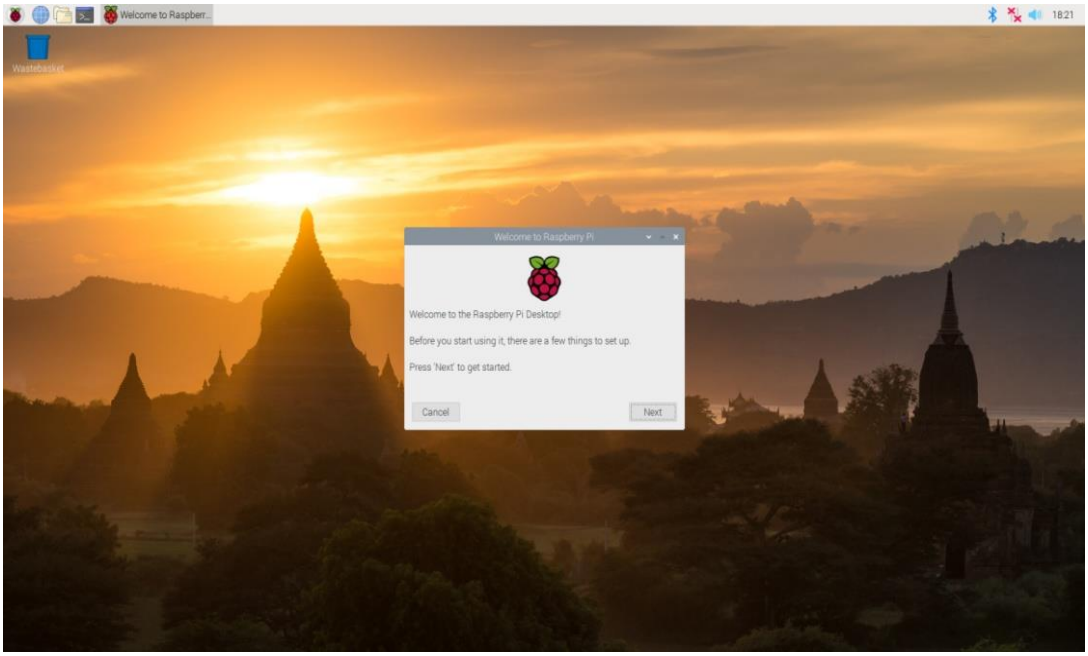


Figure 3.5. *Bureau du raspberry pi*

3.2.2 Langage de programmation (python)

Python est un langage de programmation qui peut s'utiliser dans de nombreux contextes et s'adapter à tout type d'utilisation grâce à des bibliothèques spécialisées. Il est cependant particulièrement utilisé comme langage de script pour automatiser des tâches simples mais fastidieuses [15].

Python est un langage interprété ce qui le rend relativement lent par rapport à d'autres langages de programmation (C/C++ , java) , c'est l'un des langages les plus utilisés pour le développement de l'intelligence artificielle et des applications de traitement d'image.

Python 3 et préinstallé dans le système d'exploitation raspbianOS.

3.2.3 Environnement de développement (Visual studio code)

Visual Studio Code est un éditeur de code source créé par Microsoft pour Windows, Linux et macOS. Les fonctionnalités incluent la prise en charge du débogage, de la mise en évidence de la syntaxe, de la complétion intelligente du code, des extraits de code, de la refactorisation du code et de Git intégré. Les utilisateurs peuvent modifier le thème, les raccourcis clavier, les préférences et installer des extensions qui ajoutent des fonctionnalités supplémentaires [16].

Visual Studio Code est officiellement distribué via le référentiel APT Raspberry Pi OS (auparavant appelé Raspbian), dans les variantes 32 bits et 64 bits. Nous allons l'installer en exécutant :

```
pi@raspberrypi:~ $ sudo apt update
```

```
pi@raspberrypi:~ $ sudo apt install code
```

Nous aurons l'addition suivante :

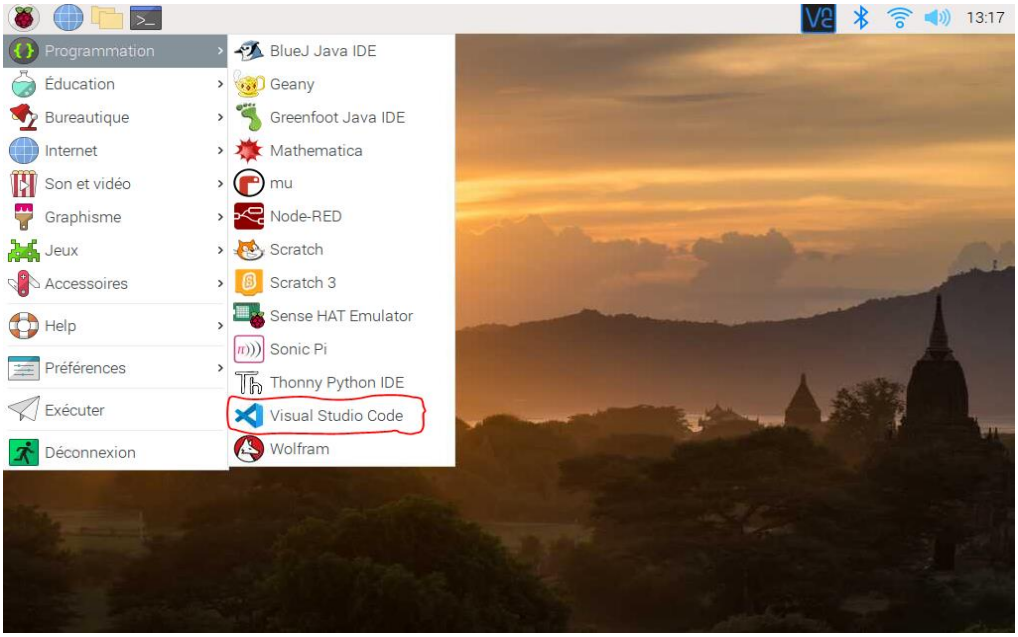


Figure 3.6. *Addition de visual studio code dans la barre de tache*

3.2.4 Google collab

Google Colab ou Colaboratory est un service cloud, offert par Google (gratuit), basé sur Jupyter Notebook et destiné à la formation et à la recherche dans l'apprentissage automatique. Cette plateforme permet d'entraîner des modèles de Machine Learning directement dans le cloud.

3.2.5 OpenCV

Comme nous l'avons déjà défini, opencv est une bibliothèque graphique qui est utilisé pour la vision par ordinateur et le traitement d'image .

Afin d'installer chaque bibliothèque de python, il suffit d'entrer la commande "sudo pip3 -m install " et à la fin mentionner le nom du Framework ou de la librairie.

```
pi@raspberrypi:~ $ pip3 install opencv-python
```

3.2.6 Numpy

NumPy est le package fondamental pour le calcul scientifique en Python. Il s'agit d'une bibliothèque Python qui fournit un objet tableau multidimensionnel, divers objets dérivés (tels que des tableaux et des matrices masqués) et un assortiment de routines pour des opérations rapides sur des tableaux, notamment mathématiques, logiques, manipulation de forme, tri, sélection, E/S , transformées de Fourier discrètes, algèbre linéaire de base, opérations statistiques de base, simulation aléatoire et bien plus encore.[17]

La librairie numpy est déjà existante dans le raspbian os sous la version 1.16.2 , nous pouvons mettre une mise à jour à l'aide de cette commande.

```
pi@raspberrypi:~ $ pip3 install -U numpy
```

3.2.7 Matplotlib

Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et à visualiser des données sous formes de graphiques. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy.[18]

Tout comme numpy, matplotlib est pré-installé dans le os sous la version 3.0.2

3.2.8 Scikit-learn

Scikit-learn (Sklearn) est la bibliothèque la plus utile et la plus robuste pour l'apprentissage automatique en Python. elle fournit une sélection d'outils efficaces pour l'apprentissage automatique et la modélisation statistique, notamment la classification, la régression, le clustering et la réduction de la dimensionnalité via une interface de cohérence en Python.[19] Cette bibliothèque, qui est en grande partie écrite en Python, est basée sur NumPy, SciPy et Matplotlib, l'installation est simple il suffit de lancer cette commande :

```
pi@raspberrypi:~ $ pip3 install numpy scipy scikit-learn
```

3.2.9 Tensorflow

TensorFlow est une plate-forme Open Source de bout en bout dédiée au machine learning. Elle propose un écosystème complet et flexible d'outils, de bibliothèques et de ressources communautaires permettant aux chercheurs d'avancer dans le domaine du machine learning, et aux développeurs de créer et de déployer facilement des applications qui exploitent cette technologie. L'installation de tensorflow 2 peut prendre plus de temps puisque l'installation est plus compliquée que d'autres librairies, la figure ci dessous montre toutes les commandes qu'il faut exécuter [20].

```
# prendre un nouveau départ
$ sudo apt-get update
$ sudo apt-get upgrade
# supprimer les anciennes versions, si elles ne sont pas placées dans un environnement virtuel
$ sudo pip uninstall tensorflow
$ sudo pip3 uninstall tensorflow
# installer les dépendances (si ce n'est déjà fait à bord)
$ sudo apt-get install gfortran
$ sudo apt-get install libhdf5-dev libc-ares-dev libeigen3-dev
$ sudo apt-get install libatlas-base-dev libopenblas-dev libblas-dev
$ sudo apt-get install openmpi-bin libopenmpi-dev
$ sudo apt-get install liblapack-dev cython
$ sudo pip3 install keras_applications==1.0.8 --no-deps
$ sudo pip3 install keras_preprocessing==1.1.0 --no-deps
$ sudo pip3 install -U --user six wheel mock
$ sudo -H pip3 install pybind11
$ sudo -H pip3 install h5py==2.10.0
# mettre à niveau les outils de configuration 40.8.0 -> 52.0.0
$ sudo -H pip3 install --upgrade setuptools
# installer gdown pour télécharger à partir de Google Drive
$ pip install gdown
# télécharger la roue
$ gdown https://drive.google.com/uc?id=11mujzVaFqa7R1_lB7q0kVPW22Ol51MPg
# installer TensorFlow
$ sudo -H pip3 install tensorflow-2.2.0-cp37-cp37m-linux_armv7l.whl wrapt --upgrade --ignore-installed
# et terminer l'installation en redémarrant
$ sudo reboot
```

Figure 3.7. *Toutes les commandes à exécuter pour installer tensorflow 2*

Après l'installation, on exécute la commande de la figure 3.8 pour vérifier l'installation correcte de TensorFlow.

```
pi@raspberrypi:~ $ python3
Python 3.7.3 (default, Dec 20 2019, 18:57:59)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> tf.__version__
'2.2.0'
>>>
```

Figure 3.8. *Commande de test pour vérifier si tensorflow 2 et installé*

3.2.10 Imutils

Une série de fonctions pratiques pour rendre les fonctions de traitement d'image de base telles que la traduction, la rotation, le redimensionnement, la squelettisation, l'affichage d'images Matplotlib, le tri des contours, la détection des contours et bien plus encore avec OpenCV et Python 3. [21] pour l'installation : `pi@raspberrypi:~ $ pip3 install imutils`

3.2.11 Mysql

MySQL est un système de gestion de bases de données relationnelles. Il est distribué sous une double licence GPL et propriétaire. Il fait partie des logiciels de gestion de base de données les

plus utilisés au monde, autant par le grand public que par des professionnels, [22] en concurrence avec Oracle, PostgreSQL et Microsoft SQL Server.

3.2.12 MysqlConnector

MySQL fournit des pilotes basés sur des normes pour JDBC, ODBC et .Net permettant aux développeurs de créer des applications de base de données dans la langue de leur choix [23]. De plus, une bibliothèque C native permet aux développeurs d'intégrer MySQL directement dans leurs applications. Pour l'installation il suffit d'utiliser cette commande:

```
pip3 install mysql-connector
```

3.3 Entrainement des modèles

Nous allons voir comment nous avons pu entrainer nos modèles de deep learning.

3.3.1 Modèle de plaques

a Préparer nos données

Dans la toute première partie, nous allons entrainer un modèle SSD pour détecter les plaques d'immatriculation, nous avons besoin d'abord d'une grande quantité d'images étiquetées de plaque d'immatriculation de véhicules, sous le nom 'licence', nous avons pu faire cela à l'aide d'un logiciel appelé **labelimg.**

Labelimg est un logiciel qui permet d'étiqueter n'importe quel image sous forme **XML (pascal VOC)** ou bien sous forme **YOLO (darknet)**, puisque on utilise un algorithme SSD nous allons utiliser le format **XML**.

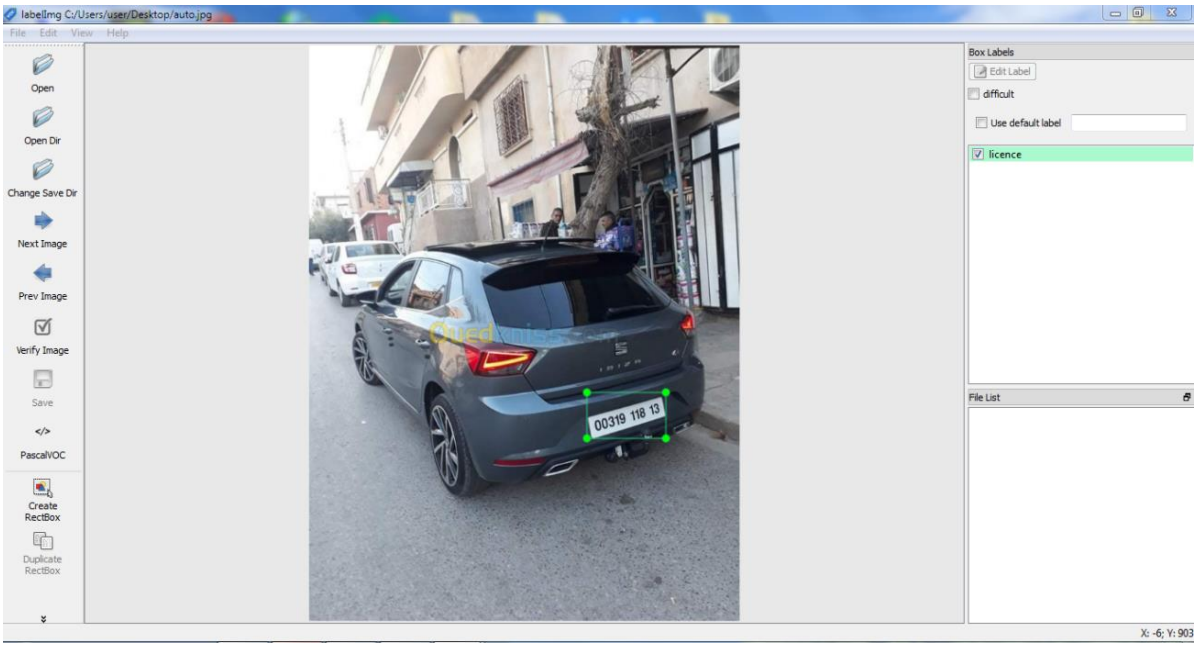


Figure 3.9. *Logiciel Labelimg*

auto.xml

C: > Users > user > Desktop > auto.xml

```xml
<annotation>
    <folder>Desktop</folder>
    <filename>auto.jpg</filename>
    <path>C:/Users/user/Desktop/auto.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>676</width>
        <height>902</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>licence</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>436</xmin>
            <ymin>543</ymin>
            <xmax>560</xmax>
            <ymax>615</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 3.10. *Exemple d'un fichier format xml généré par Labelimg*

Pour notre base de données, nous avons pris une base déjà étiquetée et nous avons additionné nos propres images (à peu prés 100 images), d'où un total de 500 image, 80% d'images attribuées pour l'entrainement et 20% d'images pour le test.

b Données pour l'entrainement

Maintenant que nos données sont prêtes, nous allons les préparer pour l'entrainement, on va commencer par étiqueter nos données sous le nom 'licence'.

```
labels = [{'name':'licence', 'id':1}]

with open(files['LABELMAP'], 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```

Figure 3.11. *Code de création de la carte d'étiquettes*

Avec les images étiquetées, les enregistrements TFRecord qui servent de données d'entrée au modèle d'entrainement (Le format TFRecord sert à stocker une séquence d'enregistrements binaires.).

```
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'],
'train')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'train.record')}
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'],
'test')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'test.record')}
```

Figure 3.12. Code *générépar les fichier .record*

c Architecture

Nos données sont prêtes, nous devons adapter nos données au modèle conformément à la figure 3.14 :

```
[ ] import tensorflow as tf
    from object_detection.utils import config_util
    from object_detection.protos import pipeline_pb2
    from google.protobuf import text_format
    config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
    config
    pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
    with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
        proto_str = f.read()
        text_format.Merge(proto_str, pipeline_config)
    pipeline_config.model.ssd.num_classes = len(labels)
    pipeline_config.train_config.batch_size = 4
    pipeline_config.train_config.fine_tune_checkpoint = os.path.join(paths['PRETRAINED_MODEL_PATH'],
                                                                     PRETRAINED_MODEL_NAME, 'checkpoint', 'ckpt-0')
    pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
    pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
    pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'],
                                                                                'train.record')]
    pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
    pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'],
                                                                                 'test.record')]

    config_text = text_format.MessageToString(pipeline_config)
    with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:
        f.write(config_text)
```

Figure 3.14. *Ajustement des données dans notre modèle*

L'architecture et les paramètre SSD MobileNets *du modèle, figure 3.15 est ensuite chargée.*

```
Model: "mobilenetv2_1.00_160"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 160, 160, 3) 0
__________________________________________________________________________________________________
Conv1 (Conv2D)                  (None, 80, 80, 32)   864         input_1[0][0]
__________________________________________________________________________________________________
bn_Conv1 (BatchNormalization)   (None, 80, 80, 32)   128         Conv1[0][0]
__________________________________________________________________________________________________
Conv1_relu (ReLU)               (None, 80, 80, 32)   0           bn_Conv1[0][0]
__________________________________________________________________________________________________
expanded_conv_depthwise (Depthw (None, 80, 80, 32)   288         Conv1_relu[0][0]
__________________________________________________________________________________________________
```

Figure 3.15. *Architecture et paramètre SSD_MobileNets pour l'entrainement*

d Entrainement

Le modèle est entrainé avec à peu prés 10000 étapes d'entrainement pour que notre modèle converge et ait le minimum d'erreur.

Tout notre entrainement a été fait sous Google collab.

```
INFO:tensorflow:Step 6700 per-step time 0.197s
I0826 12:55:18.784991 140707585763200 model_lib_v2.py:700] Step 6700 per-step time 0.197s
INFO:tensorflow:{'Loss/classification_loss': 0.13654579,
 'Loss/localization_loss': 0.1027747,
 'Loss/regularization_loss': 0.13737032,
 'Loss/total_loss': 0.3766908,
 'learning_rate': 0.077358514}
I0826 12:55:18.785339 140707585763200 model_lib_v2.py:701] {'Loss/classification_loss': 0.13654579,
 'Loss/localization_loss': 0.1027747,
 'Loss/regularization_loss': 0.13737032,
 'Loss/total_loss': 0.3766908,
 'learning_rate': 0.077358514}
```

Figure 3.16. *Entrainement du modèle*

A la fin nous aurons une erreur totale de 14% et une précision de 96%.

```
INFO:tensorflow:Step 10000 per-step time 0.196s
I0826 13:06:08.640887 140707585763200 model_lib_v2.py:700] Step 10000 per-step time 0.196s
INFO:tensorflow:{'Loss/classification_loss': 0.14158344,
 'Loss/localization_loss': 0.087769486,
 'Loss/regularization_loss': 0.12317155,
 'Loss/total_loss': 0.35252446,
 'learning_rate': 0.07352352}
I0826 13:06:08.641280 140707585763200 model_lib_v2.py:701] {'Loss/classification_loss': 0.14158344,
 'Loss/localization_loss': 0.087769486,
 'Loss/regularization_loss': 0.12317155,
 'Loss/total_loss': 0.35252446,
 'learning_rate': 0.07352352}
```

Figure 3.17. *Fin de l'entrainement du modèle*

e Résultats

Après le test avec quelques images, le modèle est précis tous comme il a été mentionné.

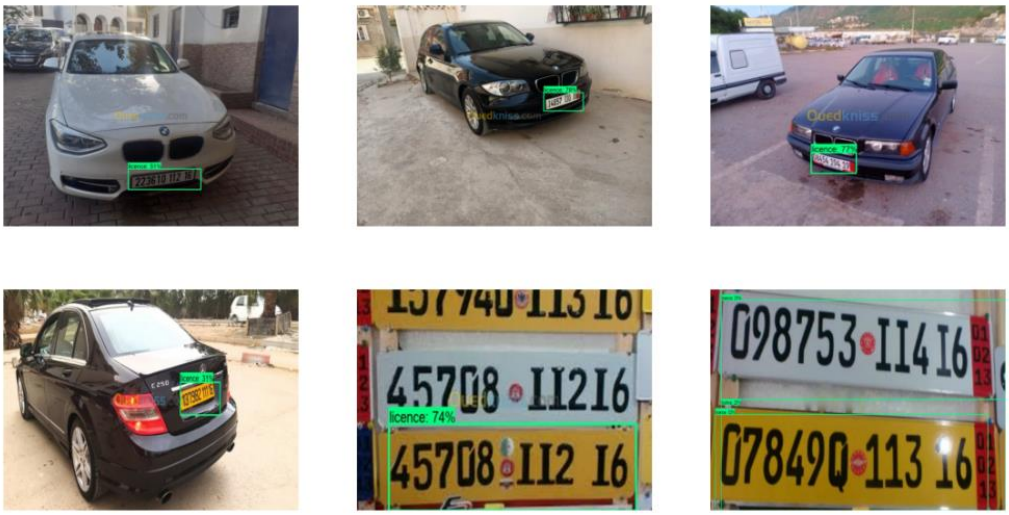


Figure 3.18. *Résultat test du model*

3.3.2 Modèle des caractères

Pour cette partie, nous devons charger les données de caractères ou de chiffres numériques puisque la plaque d'immatriculation algérienne ne contient que des nombres.

a Préparation des données

Il y'a des centaines de librairies qui sont déjà prêtes, dans notre cas nous avons pris une librairie qui contient tous les caractères avec 81950 images.

Bien sur nous aurons besoin que de numéros donc après avoir supprimé tous les autres caractères, il nous reste que 10570 images pour le traitement. La figure ci dessus est un programme qui visualise nos types de données.

```
[ ]  cols=4
     rows=3
     fig = plt.figure(figsize=(10,8))
     plt.rcParams.update({"font.size":14})
     grid = gridspec.GridSpec(ncols=cols,nrows=rows,figure=fig)

     # create a random list of images will be displayed
     np.random.seed(45)
     rand = np.random.randint(0,len(dataset_paths),size=(cols*rows))

     # Plot example images
     for i in range(cols*rows):
         fig.add_subplot(grid[i])
         image = tf.keras.preprocessing.image.load_img(dataset_paths[rand[i]])
         label = dataset_paths[rand[i]].split(os.path.sep)[-2]
         plt.title('"{:s}"'.format(label))
         plt.axis(False)
         plt.imshow(image)
```

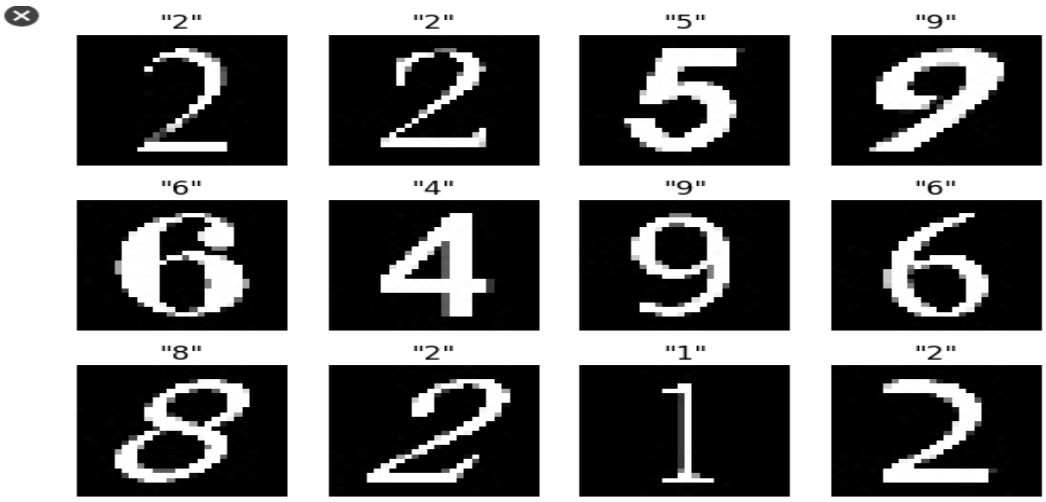


Figure 3.19. *visualisation des données*

Après nous allons séparer nos données en deux tranches une pour l'entrainement et l'autre pour le test, tout comme la partie précédente.

Il faut aussi redimensionner toutes les images à une résolution (80x80) pour que notre modèle **MobileNets** soit capable de lire l'image et la traiter.

```
# Arange input data and corresponding labels
X=[]
labels=[]

for image_path in dataset_paths:
  label = image_path.split(os.path.sep)[-2]
  image=tf.keras.preprocessing.image.load_img(image_path,target_size=(80,80))
  image=tf.keras.preprocessing.image.img_to_array(image)

  X.append(image)
  labels.append(label)

X = np.array(X,dtype="float16")
labels = np.array(labels)

print("[INFO] Find {:d} images with {:d} classes".format(len(X),len(set(labels))))
```

Figure 3.20. *Organisation des données et leur préparation pour le traitement.*

b Architecture et entrainement

Nous allons utiliser la même architecture SSD MobileNets v1.

La ***figure 3.21.*** montre l'architecture de l'algorithme mobileNetsv1avec plus de paramètres

```
================================
Total params: 2,423,242
Trainable params: 2,389,130
Non-trainable params: 34,112
________________________________
```

Figure 3.21. *Paramètres de l'architecture*

Après avoir chargé notre modèle, nous allons fournir nos données et commencer l'entrainement.

```
Epoch 1/30
486/486 [==============================] - 147s 303ms/step - loss:
1.5441 - accuracy: 0.5989 - val_loss: 1.5866 - val_accuracy: 0.6090

Epoch 00001: saving model to License_character_recognition.h5
Epoch 2/30
486/486 [==============================] - 124s 256ms/step - loss:
0.4201 - accuracy: 0.8930 - val_loss: 0.4698 - val_accuracy: 0.8788

Epoch 00002: saving model to License_character_recognition.h5
Epoch 3/30
486/486 [==============================] - 125s 258ms/step - loss:
0.2769 - accuracy: 0.9281 - val_loss: 0.2617 - val_accuracy: 0.9352

Epoch 00003: saving model to License_character_recognition.h5
Epoch 4/30
486/486 [==============================] - 125s 257ms/step - loss:
0.2103 - accuracy: 0.9454 - val_loss: 0.1744 - val_accuracy: 0.9563

Epoch 00004: saving model to License_character_recognition.h5
Epoch 5/30
486/486 [==============================] - 126s 259ms/step - loss:
0.1642 - accuracy: 0.9554 - val_loss: 0.1491 - val_accuracy: 0.9592
```

Figure 3.22. *Entrainement du model ssd_mobileNets*

A la fin nous avons eu une précision de 95% et une perte de 14%, ce qui veut dire que notre modèle est plus précis mais ne l'ai pas à 100%.

Nos modèles vont être sauvegardés sous format **.json** et **.h5,** ce qui facilitera leur appel pour de futures utilisations.

3.4 Système développé

Cet organigramme présente le fonctionnement du système :

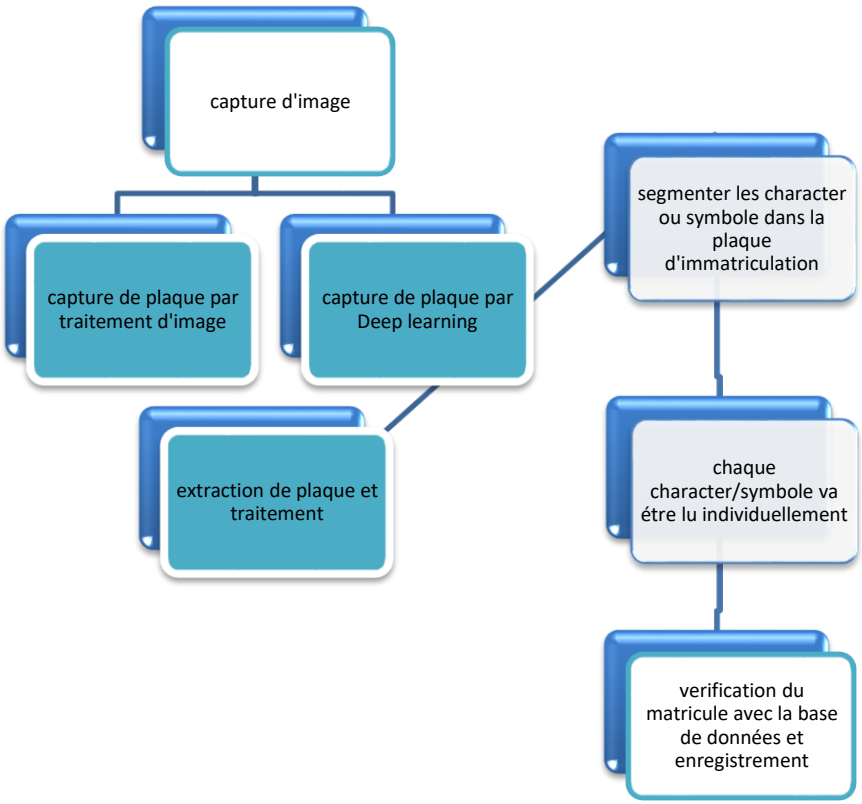


Figure 3.23. *Organigramme montrant le fonctionnement du programme*

3.4.1 Capture d'image

La capture se fait avec la raspberry cam en temps réel , nous allons tout d'abord commencer par capturer l'image en temps réel , nous capturons nos images en résolution de 640x480 pixel, il est important de noter que la résolution joue une grande partie pour montrer les détails dans une image.

Nous allons aussi inverser l'image car la camera raspberry va être placé d'une façon inverse, donc nous allons la flipper de 180° pour capter l'image clair et net.

3.4.2 Extraction de plaque par traitement d'image

Afin de pouvoir extraire la plaque d'immatriculation avec le traitement d'image on passe par plusieurs étapes:

- Convertir l'image en filtre noir et blanc
- Appliquer le flou médian à l'image (medianblur)
- Appliquer le détecteur de canny
- Trouver tous les contours avec findContours

- Capturer tous les points des contours et appliquer la méthode d'approximation pour voir ci c'est un rectangle (représentation de la plaque)

L'organigramme suivant montre tout le fonctionnement.

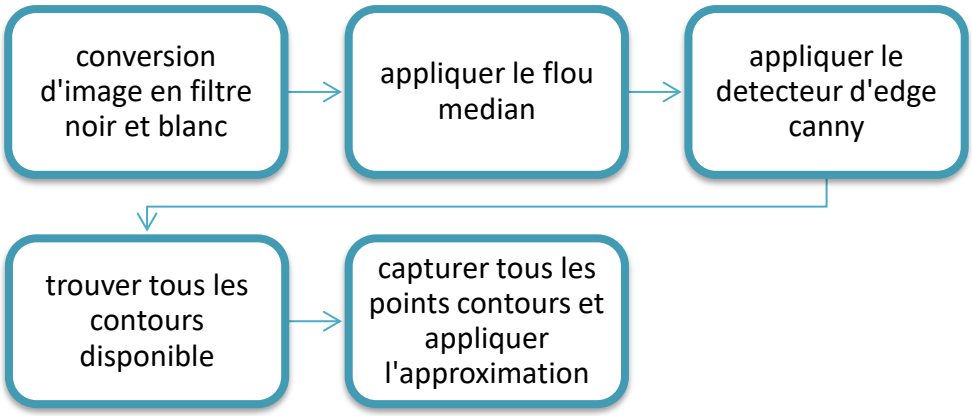


Figure 3.24. *Organigramme montrant l'extraction de plaque via le traitement d'image*

- ***Conversion vers le filtre noir et blanc***

Comme nous l'avons expliqué dans le chapitre 2, La conversion à l'échelle de gris est essentielle pour commencer le traitement d'une image en format 8 bits.

- ***Appliquer le flou median***

Il existe plusieurs sortes de type de flou d'image, dans notre cas nous avons utilisé le flou médian car c'est le plus rapide et le plus optimisé pour notre image.

- ***Appliquer le détecteur de bord canny***

Afin de pouvoir détecter la plaque il est nécessaire que l'ordinateur sache où sont les bordures. c'est pour ca qu'on utilise le détecteur canny, et le mettre en une valeur entre 30 valeur minimale de seuillage et 200 valeur maximale.

- ***Trouver tous les contours disponibles***

Après avoir appliqué le détecteur de bord il est temps qu'on renvoie l'ensemble des contours qui correspondent à chacune des taches blanches sur l'image

- ***Capturer tous les points contours et appliquer l'approximation***

Par la suite nous allons saisir les coordonnées appropriées en utilisant la fonction python **sorted, celle-ci** va renvoyer une nouvelle liste triée à partir des éléments itérables.

Figure 3.25. *Etapes de traitement d'image.*

3.4.3 Extraction de plaque par deep learning

Dans cette deuxième méthode nous allons utiliser le modèle que nous avons entrainé auparavant , seulement nous allons le convertir en model **TensorflowLite** ce qui veut dire un fichier format **.tflite** qui est plus optimisé pour un traitement en temps réel.[24]

a TensorflowLite

TensorFlow Lite est un framework d'apprentissage en profondeur open source pour l'inférence sur l'appareil.

En utilisant ces 3 commandes, on a pu convertir notre model **SSD .chkpt** vers **.tflite** , il est important de noter que notre modèle a été entrainé sous des images de résolution 320x320.

```
FROZEN_TFLITE_PATH = os.path.join(paths['TFLITE_PATH'], 'saved_model')
TFLITE_MODEL = os.path.join(paths['TFLITE_PATH'], 'saved_model', 'detect.tflite')
```

```
command = "tflite_convert \
--saved_model_dir={} \
--output_file={} \
--input_shapes=1,300,300,3 \
--input_arrays=normalized_input_image_tensor \
--output_arrays='TFLite_Detection_PostProcess','TFLite_Detection_PostProcess:1','TFLite_Detection_PostProcess:2',
'TFLite_Detection_PostProcess:3' \
--inference_type=FLOAT \
--allow_custom_ops".format(FROZEN_TFLITE_PATH, TFLITE_MODEL, )
```

Figure 3.26. *Conversion du modèle en tflite*

- Convertir l'image du format BGR a RGB.
- Redimensionner vers une résolution 320x320 .
- Appliquer le modèle sur l'image
- Extraire les coordonnées de la plaque

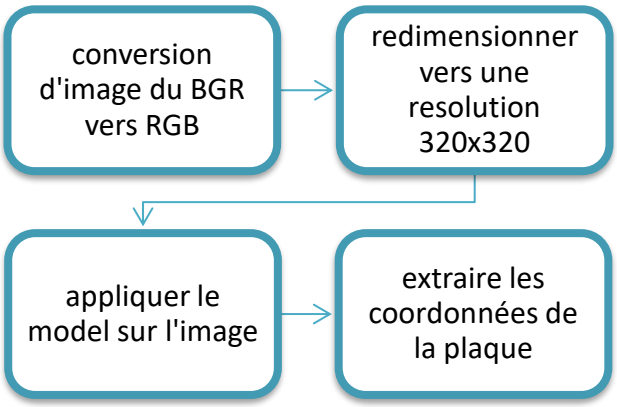


Figure 3.27. *Organigramme montrant l'extraction de plaque via deep learning*

L'image suivante montre toutes les étapes du traitement.

Figure 3.28. *Exemple des étapes de détection via deep learning*

3.4.4 Capture de plaque et traitement

Après avoir détecté notre plaque d'immatriculation dans chaque méthode, il suffit d'extraire l'image de la plaque afin de passer vers l'étape suivante .

il suffit qu'on ait les 2 coordonnées du rectangle (ou de la plaque) détecté dans l'image afin de pourvoir l'extraire. La figure ci dessous montre le code pour les 2 méthodes. (xmin , ymin , xmax , ymax sont les coordonnées de la plaque détectée dans l'image).

```
plaque = image[ymin:ymax,xmin:xmax]
```

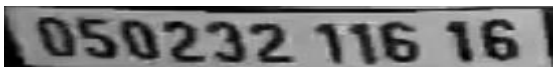


Figure 3.29. *Extraction de la plaque d'immatriculation*

3.4.5 Segmentation de caractères dans la plaque d'immatriculation

Pour la segmentation de caractères on passe par plusieurs étapes :

- Redimensionnement d'images vers une résolution (380,140)

- Conversion vers l'échelle de gris
- Appliquer le seuillage binaire inverse
- Trouver tous les contours disponibles dans l'image
- Les arranger de gauche a la droite
- Vérifier s'il s'agit vraiment d'un caractère
- Ajouter à la liste de préparation pour le traitement

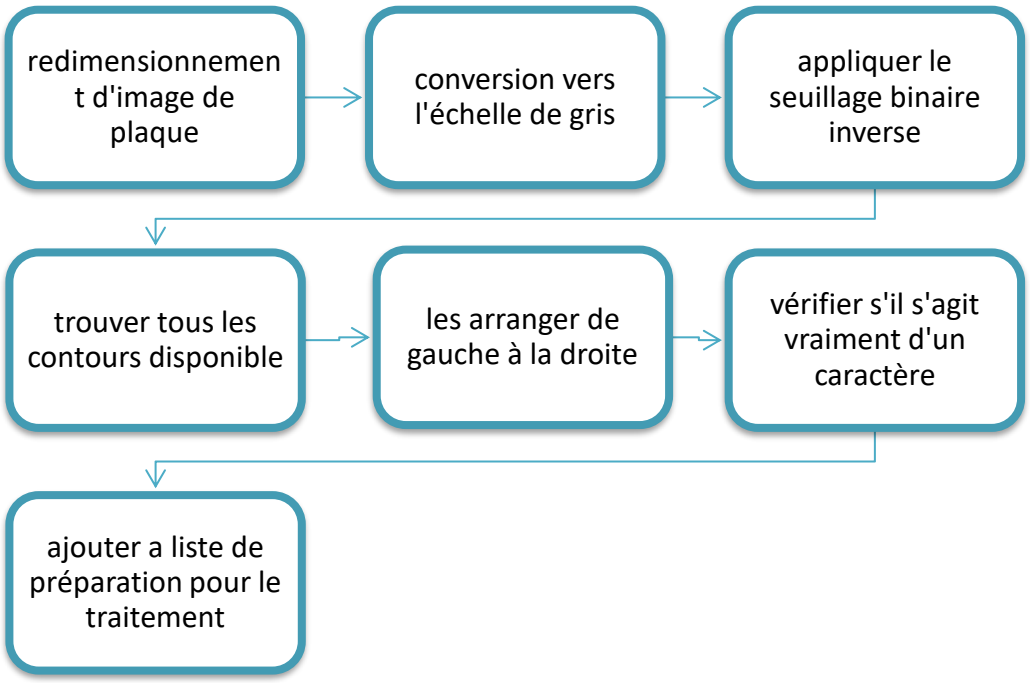


Figure 3.30. *Organigramme qui montre les étapes de segmentation des caractères*

- ***Redimensionnement d'image***

Après avoir extrait l'image de la plaque, il faut la convertir vers une résolution de largeur : 140, hauteur : 50, ceci va faciliter le traitement de l'image.

- ***Conversion à l'échelle de gris***

La couleur joue un rôle négligeable pour la plaque d'immatriculation, nous pouvons donc la supprimer pour optimiser la puissance de calcul.

- ***Appliquer le seuillage binaire inverse***

Nous avons défini un seuil pour que toute valeur de pixel inférieure à celle-ci soit convertie en 255 et vice versa. Nous utilisons un seuillage appelé seuillage binaire inverse. Dans l'exemple, nous avons utilisé une valeur seuil de 180, cette valeur peut être modifiée pour être plus compatible avec notre image.

- ***Trouver tous les contours disponibles***

Ensuite, nous avons utilisé findContours pour identifier les coordonnées du caractère de la licence. Cette fonction est basée sur une théorie simple : les contours sont simplement des courbes joignant tous les points continus (le long de la frontière) partageant la même couleur et la même intensité.

- ***Les arranger en ordre***

Nous avons créé une fonction appelée sort_contours qui trie essentiellement les contours fondés de gauche à droite. Ceci est essentiel car nous voulons non seulement reconnaître le caractère, mais aussi les ranger dans le bon ordre. Chaque rectangle détecté va récupérer ces dimensions , la fonction va calculer quelle dimension est plus proche de la gauche que l'autre , et quel rectangle est placé avant l'autre

- ***Vérifier s'il s'agit d'un caractère***

Puisque nous savons que notre caractère de plaque a généralement une hauteur supérieure à la largeur, nous pouvons filtrer les contours non pertinents en sélectionnant uniquement les contours avec un rapport compris entre 1 et 5,5(Le rapport est calculé comme la hauteur divisée par la largeur). Nous savons également que chaque caractère de la plaque doit avoir sa hauteur supérieure à la moitié de la largeur de la plaque, nous pouvons donc ajouter un autre filtre de contour.

- ***Ajout à la liste de préparation pour le traitement***

Nous dessinons des limites avec tous les contours qui passent ces filtres, appliquons un seuillage binaire sur chaque contour déterminé et les ajoutons à la liste crop_characters (qui représente la liste de toutes les images de nos caractères).

La figure ci dessous montre toutes les étapes de traitement.

Figure 3.30. *Exemple de traitement*

3.4.6 Lecture individuel des symboles

Maintenant que nous avons extrait notre plaque et segmenté tous les caractères, on passe à la lecture du matricule (la plaque d'immatriculation algérienne contient soit 11 ou 10 chiffres).

Nous aurions d'abord besoin de charger notre architecture de modèle, de charger les poids après la phase d'entraînement et les classes d'étiquettes d'origine.

```
# Load model architecture, weight and labels
json_file = open('MobileNets_character_recognition.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)
model.load_weights("License_character_recognition_weight.h5")
print("[INFO] Model loaded successfully...")

labels = LabelEncoder()
labels.classes_ = np.load('license_character_classes.npy')
print("[INFO] Labels loaded successfully...")
```

```
[INFO] Model loaded successfully...
[INFO] Labels loaded successfully...
```

Figure 3.31. *Modèle chargé avec succès*

nous avons configuré la couche d'entrée de notre modèle pour recevoir des images d'entrée avec la forme de (80,80,3). Ainsi, nous avons eu besoin de convertir notre image à la taille appropriée, nous avons implémenté les classes d'étiquettes chargées pour inverser l'étiquette d'encodage obtenue du modèle aux caractères numériques. Une boucle est générée sur chaque image de caractère dans crop_characters, qui stocke toutes les prédictions du modèle dans final_result, et trace chaque image avec ses prédictions correspondantes, comme illustré à la figure 3.32.

```
labels = LabelEncoder()
labels.classes_ = np.load('license_character_classes.npy')
print("[INFO] Labels loaded successfully...")

# pre-processing input images and pedict with model
def predict_from_model(image,model,labels):
    image = cv2.resize(image,(80,80))
    image = np.stack((image,)*3, axis=-1)
    prediction = labels.inverse_transform([np.argmax(model.predict(image[np.newaxis,:]))])
    return prediction

fig = plt.figure(figsize=(15,3))
cols = len(crop_characters)
grid = gridspec.GridSpec(ncols=11,nrows=1,figure=fig)

final_string = ''
for i,character in enumerate(crop_characters):
    fig.add_subplot(grid[i])
    title = np.array2string(predict_from_model(character,model,labels))
    plt.title('{}'.format(title.strip("'[]"),fontsize=20))
    final_string+=title.strip("'[]")
    plt.axis(False)
    plt.imshow(character,cmap='gray')


print("Achieved result: ", final_string)
plt.savefig('final_result.png', dpi=300)
```

```
Achieved result:  05023211616
```

Figure 3.32. *Programme de détection du matricule avec résultat*

3.4.7 Vérification avec la base de données

Nous avons pu réaliser tout notre algorithme, mais la prédiction peut entrainer des erreurs entre une image et une autre, pour y remédier à ce problème, nous avons créé une base de données mysql afin que le traitement soit toujours correct.

On a tout d'abord créé un tableau de vérification pour valider si le matricule existe ou non.

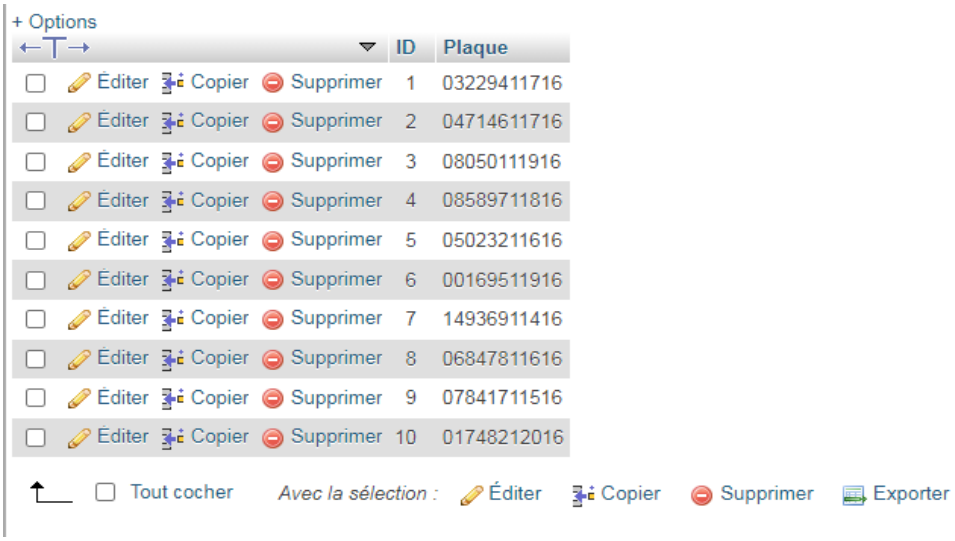


Figure 3.33. *Table de vérification des données*

Après cette étape de vérification, nous avons sauvegardé la présence du matricule avec la date et l'heure d'enregistrement. A partir de là, on peut également donner d'autres commandes à savoir actionner une barrière pour ouverture, etc.

Figure 3.34. *Table de sauvegarde de données*

3.5 Résultats et comparaison

Après une étude laborieuse et une préparation délicate, il est temps de voir les résultats et pouvoir les comparer. Nous allons juste rappeler deux choses:

- Le premier algorithme utilise le **traitement d'image** pour la détection de la plaque.
- Le deuxième algorithme utilise le **deep learning** pour la détection de la plaque.

3.5.1 Résultats

- **Résultats du premier algorithme :**

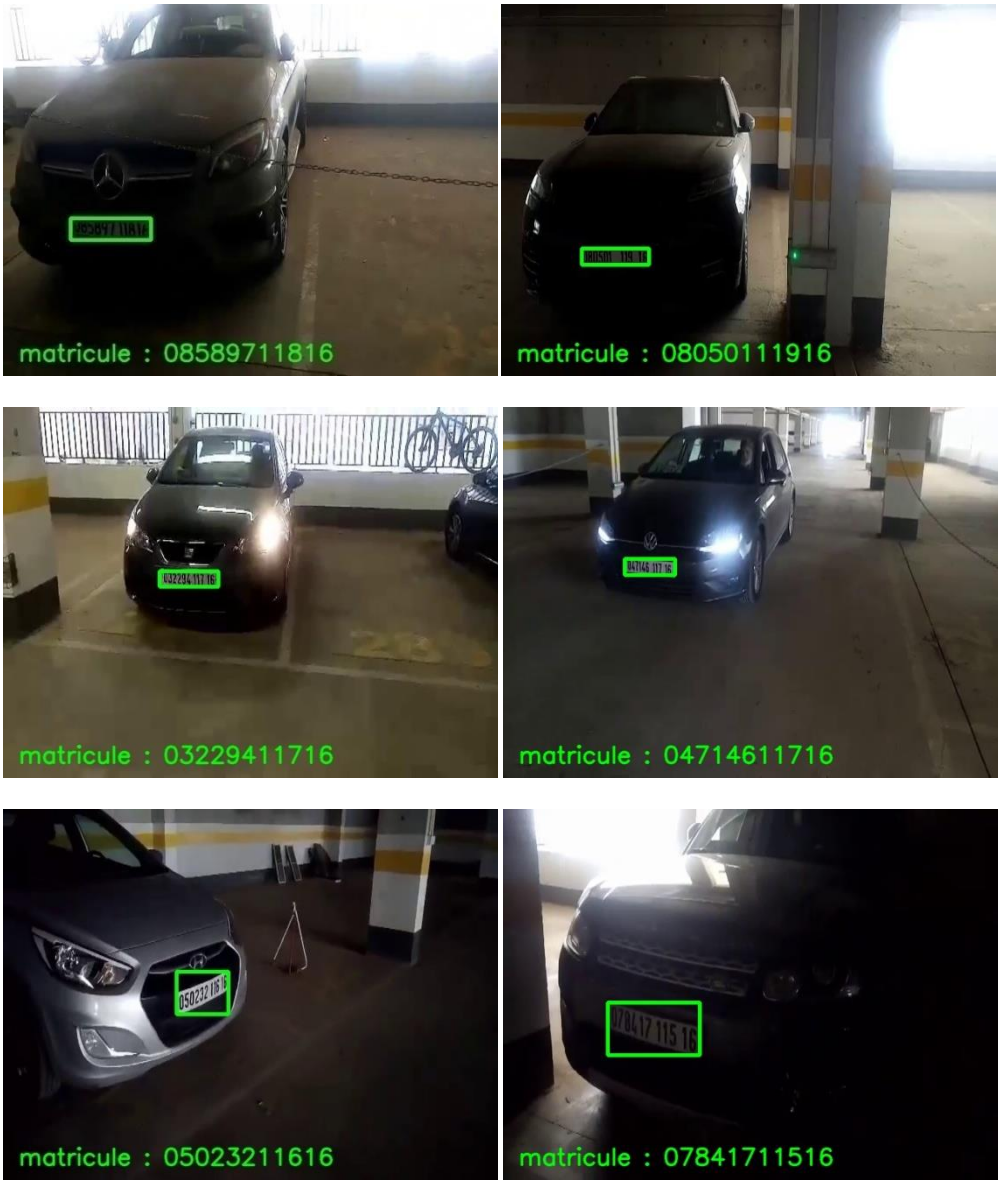


Figure 3.35. *Résultats du premier algorithme*

L'algorithme non seulement peut être précis mais aussi le plus optimisé pour un traitement en temps réel, cela n'empêche pas que dans certains cas il y'a une tolérance à l'erreur.

- **Résultats du deuxième algorithme :**

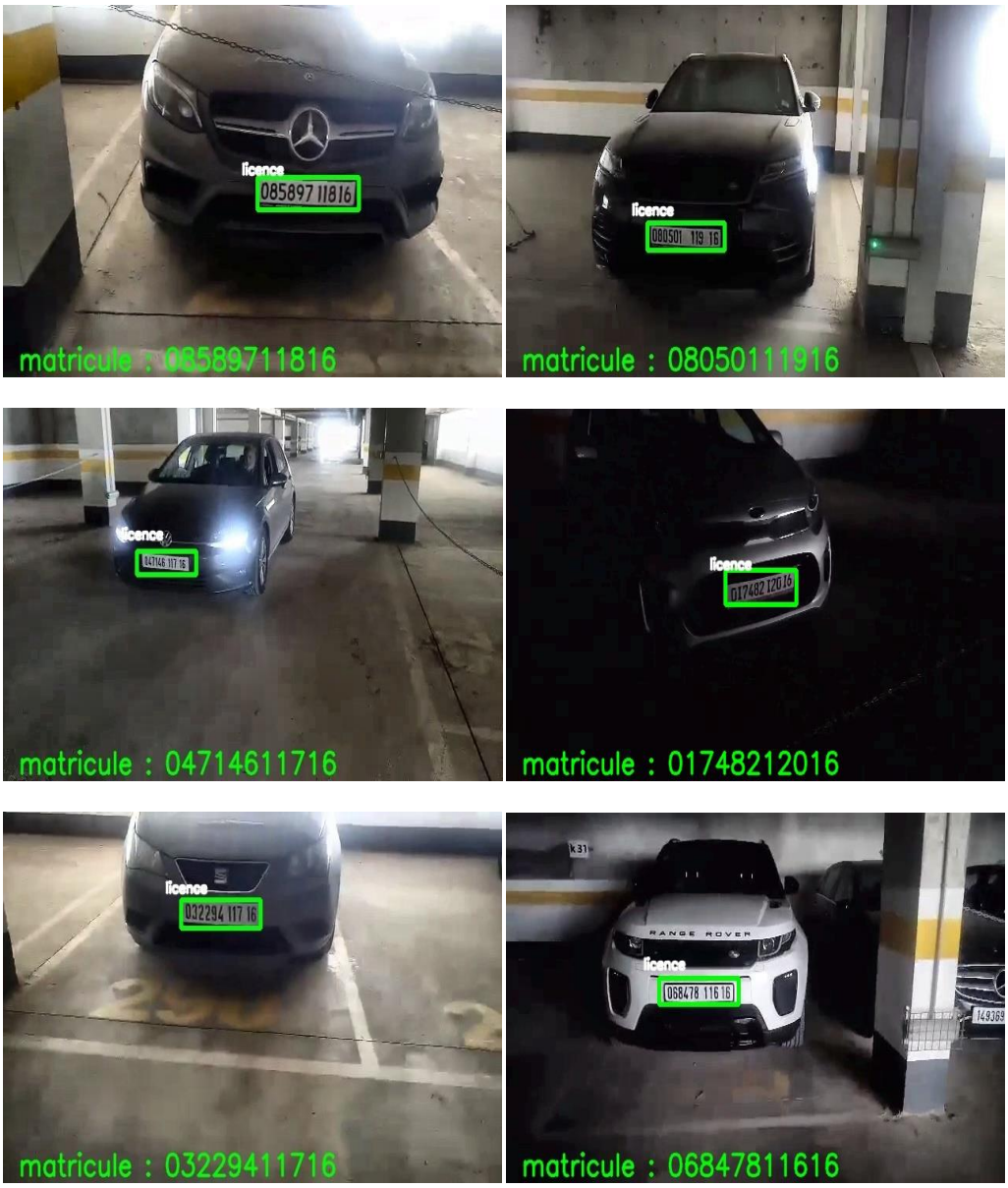


Figure 3.36. *Résultats du deuxième algorithme*

L'algorithme est très précis mais n'est pas optimisé comme le premier, il reste applicable pour le temps réel.

3.5.2 Comparaison

Nous allons maintenant comparer les deux algorithmes par la :

- Performance (image par seconde)
- Précision (taux de détection)
- Taux d'erreur.

Toutes ces comparaisons vont être évaluées sur des graphes.

Performance

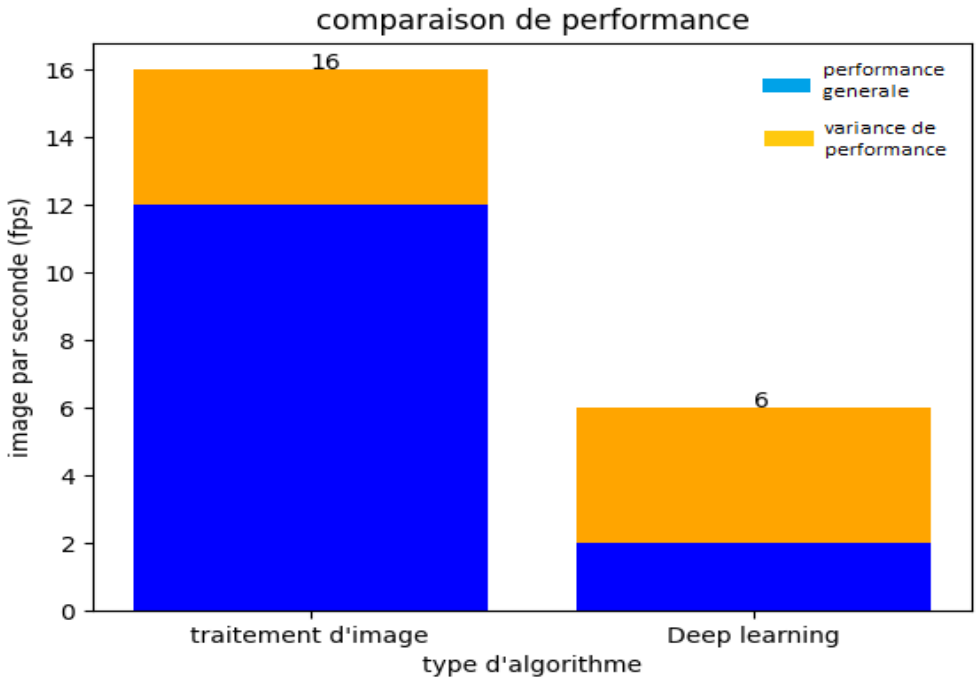


Figure 3.37. *Comparaison des performances entre les deux algorithmes*

Comme on le voit les performances pour le premier algorithme (traitement d'image) sont beaucoup plus élevées que celle du deuxième (deep Learning).

Précision

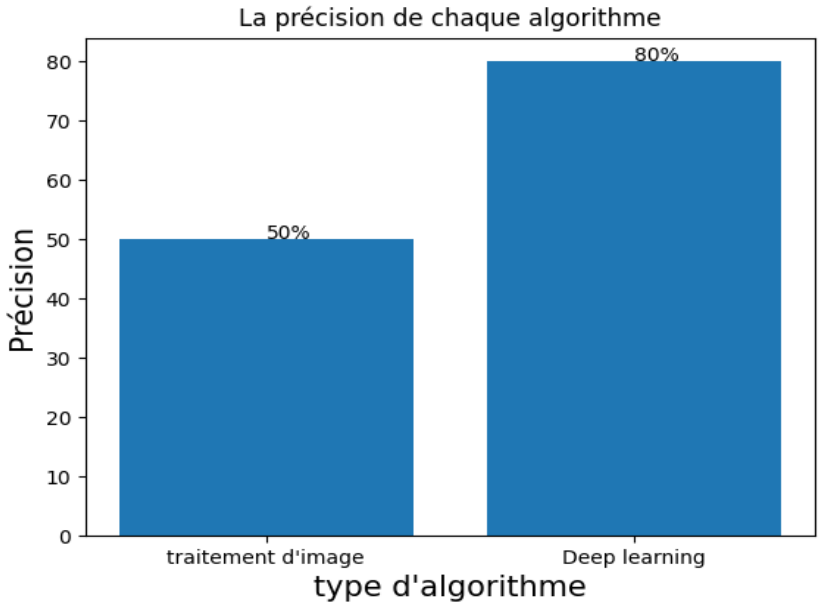


Figure 3.38. *Comparaison de précision des algorithmes*

Comme on le remarque, le modèle AI est plus précis que celui du traitement d'image, ceci s'explique par le fait que le deep learning prend plus de puissance de calcul que le traitement d'images.

Taux d'erreur

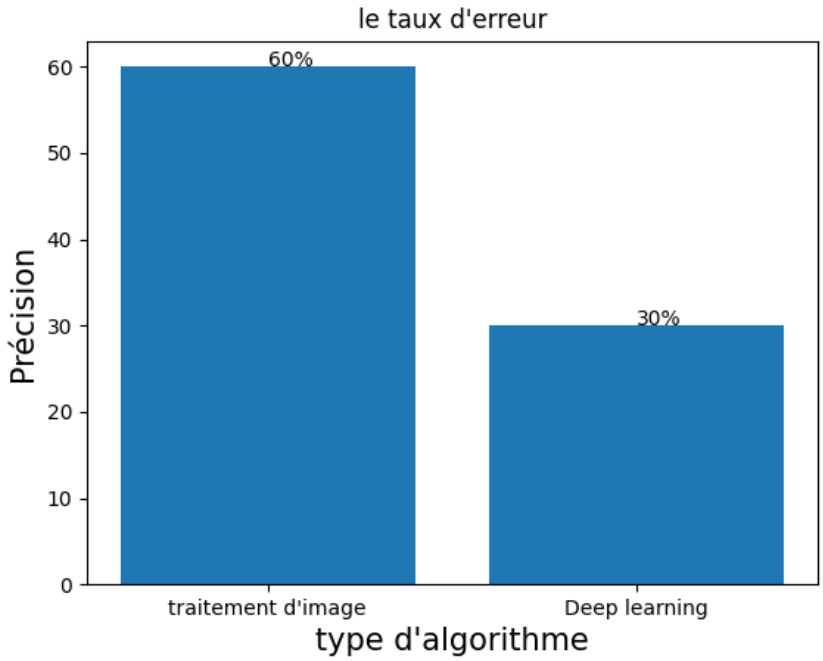


Figure 3.39. *Comparaison du taux d'erreur des algorithmes*

Le taux d'erreur dépend surtout de la situation, il peut être élevé comme il peut être bas

La figure 3.40 suivante montre toutes les comparaisons en un seul terrain.

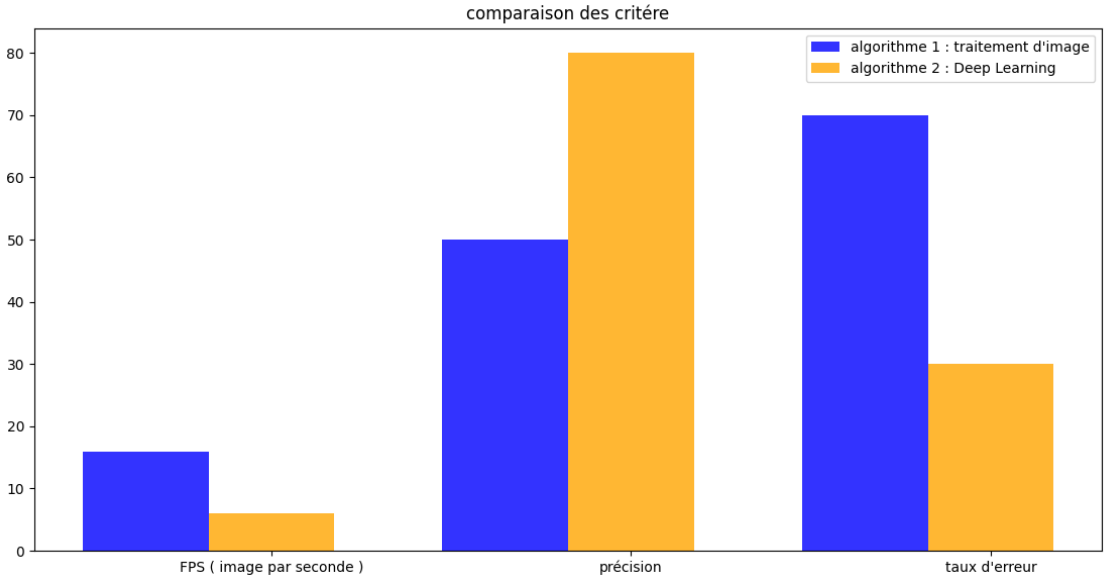


Figure 3.40. *Comparaison générale*

3.6 Problèmes et solutions

Il est important de noter qu'il ya eu plusieurs bug pendant notre réalisation.

3.6.1 Problème de l'algorithme Deep Learning

a Problème

L'un des premiers problèmes qu'on a rencontré est le type d'algorithmes, lequel choisir, nous avions le choix de plusieurs algorithmes **YOLO, fast R-CNN** , on avait fait le choix de l'algorithmes **YOLO** puisqu'il était la nouveauté et l'un des meilleurs des algorithmes de détection d'objets, le problème était sa mise en place et les performance sur la carte raspberry pi ,

b Solution

Notre choix s'est alors porté sur l'algorithme **MobileNetsSSD** , qui est un algorithme développé spécifiquement pour fonctionner dans les appareils mobiles (IOT , téléphone , microcontrôleur) , malgré que le traitement soit lent mais il reste plus optimal que la majorité des algorithmes disponibles.

3.6.2 Problèmes de performance

a Problème

Le problème le plus visible est celui de la performance, surtout dans la partie du deep learning, on remarque que les performances diminuent d'une façon très importante, avec une moyenne de 6 images par seconde, ce qui rend le traitement en temps réel très long.

b Solution

La première solution est de changer le type d'approche, c'est l'une des raisons principales d'avoir utiliser un algorithme de traitement d'image , comme nous l'avons vu dans la section précédente , les performances sont beaucoup plus élevées que celle du deep Learning.

La deuxième solution c'est d'exécuter notre code avec une carte graphique ou bien mieux encore, travailler dans un environnement **CUDA ,** ce qui n'est pas possible puisque on exécute tout dans la carte raspberry pi , mais il est possible de connecter une intel Neural compute stick 2 qui est une clé USB conçue par Intel et qui permet de développer ou d'exécuter des algorithmes d'IA.

3.6.3 Problèmes de lecture du matricule

a Problème

Pour la reconnaissance des chiffres du matricule, on a voulu utiliser l'un des algorithmes de reconnaissance optique de caractères , **OCR Tesseract** de **Google** qui était l'un des meilleurs , mais après plusieurs tests on a remarqué que les résultats étaient faibles , on a essayé d'autres alternatives comme **keras OCR** , **easyOCR** , **kerasOCR** , tous était non efficace, **easyOCR** était le plus précis parmi eux. On a noté que dans certains cas il y avait de la confusion entre nombre et chiffre, sans oublier la lenteur du traitement

b Solution

Pour cela, nous avons opté pour la réalisation de la méthode que nous avons mentionné avant, qui est la segmentation et le traitement de chaque caractère individuellement, cela a montré que c'est la méthode la plus efficace et la plus précise. , On a également noté que les performances sont meilleures.

3.7 Conclusion

Dans ce chapitre nous avons vu comment préparer notre environnement de travail.

Nous avons aussi pu entrainer notre modèle de détection en expliquant toutes les étapes d'entrainement

Ensuite on a expliqué le fonctionnement de chacun de nos algorithmes, on a visualisé tous nos résultats avec une étude comparative. .

Et enfin on a mis en évidence tous les problèmes qu'on a pu rencontrer avec leurs solutions respectives .

Conclusion générale

Avec l'avancement des technologies d'intelligence artificielle et du traitement d'image, il est devenu de plus en plus facile à détecter et à identifier toutes sortes d'objets, allant des matricules aux maladies, et à tout type de véhicule ou de personnel.

Dans cette étude, nous avons présenté et implémenté un système de détection et de reconnaissance de plaque d'immatriculation algérienne dans une carte raspberry pi à base de CPU, on a utilisé deux différentes méthodes de détection, nous sommes passé de la théorie du traitement d'image et des algorithmes d'apprentissage approfondie à l'implémentation sur la carte.

Nous avons ainsi expliqué comment chacun de nos algorithmes extrait la plaque et lit le matricule avec sa propre méthode de détection, puis nous avons testé nos algorithmes sur notre carte raspberry pi et affiché les résultats avec une petite étude comparative entre les deux algorithmes qui nous a prouvé que l'apprentissage approfondie était plus précis alors que le traitement d'image était plus rapide et plus performant.

A la fin, on a discuté des problèmes rencontrés ainsi que des solutions possibles.

Ce systèmes peut être utilisé avec une interface graphique connectée avec une camera de capture en temps réel en tant que système de surveillance pour des applications de parking ou de radar ou bien dans d'autres domaines d'applications.

Bibliographie

[1] https://siecledigital.fr/2018/08/20/histoire-intelligence-artificielle/

[2]https://www.b2b-infos.com/3253/les-domaines-d-application-de-l-intelligence-artificielle/

[3] https://www.sas.com/en_us/insights/analytics/neural-networks.html

[4] https://3nions.com/what-is-machine-learning-and-where-it-is-used/

[5] https://fr.wikipedia.org/wiki/Plaque_d%27immatriculation

[6] https://datasheets.raspberrypi.org/rpi4/raspberry-pi-4-product-brief.pdf

[7] https://www.raspberrypi.org/documentation/accessories/camera.html

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

[9] Computer Vision avec Python 3 par saurabh kapur

[10] https://www.docs.opencv.org/master/

[11] Définition de l'apprentissage approfondie et L'ANN : https://www.kaggle.com/ryanholbrook/a-single-neuron

[12] https://datawow.io/blogs/interns-explain-cnn-8a669d053f8b

[13] MobileNets : des réseaux de neurones convolutifs efficaces pour la vision mobile ,Applications ,Andrew G. Howard Menglong Zhu Bo Chen Dmitry Kalenichenko ,Weijun Wang Tobias Wey et Marco Andreetto Hartwig Adam

[14] raspbian os petite définition : https://www.raspberrypi.org/software/operating-systems/

[15] petite définition de python : https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1445304-python-definition-et-utilisation-de-ce-langage-informatique/

[16] définition de visual studio code : https://en.wikipedia.org/wiki/Visual_Studio_Code

[17] définition de numpy : https://numpy.org/doc/stable/user/whatisnumpy.html

[18] définition de matplotlib : https://matplotlib.org/

[19] Définition de sklearn : https://pypi.org/project/scikit-learn/

[20] définition de tensorflow : https://www.tensorflow.org/

[21] définition de imutils : https://pypi.org/project/imutils/

[22] définition mysql : https://fr.wikipedia.org/wiki/MySQL

[23] définition mysql.Connector : https://www.mysql.com/products/connector/

[24] définition de tensorflowLite : https://www.tensorflow.org/lite/api_docs

[25] https://www.hebergementwebs.com/tutoriels/qu-est-ce-qu-un-pixel-definition-du-terme

[26] https://in.mathworks.com/help/images/understanding-color-spaces-and-color-space-conversion.html

[27] https://perso.esiee.fr/~perretb/I5FM/TAI/histogramme/index.html

[28] https://en.wikipedia.org/wiki/Transfer_learning#cite_note-2