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

MALKI Yacine

&

TIACHACHAT Nassim

Deep Learning pour la détection des plaques d'immatriculation

Proposé et encadré par : Mme. BOUGHERIRA Hamida

Co-encadré par : Mme. BOUGHERIRA Nadia

Année Universitaire 2018-2019

Remerciements

On remercie en premier lieu notre Allah qui nous a donné la santé pour terminer ce modeste travail.

Et en deuxième lieu on exprime nos sincères remerciement à ceux qui nous ont aidé à l'élaboration de ce modeste travail, en particulier notre promotrice Mme. BOUGHRIRA HAMIDA et Mme. BOUGHRIRA NADIA.

Nous remercions nos collègues FAIZA et RADIA. Et nos familles et nos amis (KHALFOUNI ABDE REZAK, YAHIAOUI NABIL, AMEUR NESSRINE, LAZAREF CHAHRAZED, MEHDAOUI CHESSEDINE) qui nous ont aidés à terminer notre travail

ملخص: : الهدف من هذا المشروع هو اقتراح طريقة تستخدم تقنيات التعلم العميق لحل مهمة الكشف التلقائي عن لوحات الترخيص. استخدمنا Python تحت Anaconda كبيئة تطوير ومكتبة Tensorflow. لقد أنشأنا قاعدة بيانات خاصة بنا ووسمنا الصور يدويًا وبرمجيًا. وتمت مقارنة خوارزميات التعلم العميق Faster R-CNN و SSD MobileNet تحت وحدة المعالجة المركزية والجرافيك. كان الأول أكثر دقة ولكن أبطأ واستخدام GPU أسرع بكثير لكليهما. نظهر بشكل عام أننا حصلنا على دقة تنبؤ عالية.

كلمات المفاتيح: الكشف عن لوحة الترخيص ؛ التعلم العميق ؛ Tensorflow.

Résumé : Le but de ce projet est de proposer une méthode utilisant des techniques d'apprentissage en profondeur pour résoudre la tâche de détection automatique de plaques d'immatriculation. Nous avons utilisé Python sous anaconda comme environnement de développement et la bibliothèque Tensorflow. Nous avons créé notre propre base de données et labellisé les images manuellement et par programmation. Les algorithmes d'apprentissage profond Faster R-CNN et SSD MobileNet ont été comparés sous CPU et GPU. Le premier s'est avéré plus précis mais plus lent et l'utilisation de GPU est beaucoup plus rapide pour les deux. Nous montrons qu'avons globalement obtenu une précision de prédiction élevée.

Mots clés : détection de plaque d'immatriculation ; Deep Learning ; Tensorflow.

Abstract : The purpose of this project is to propose a method using deep learning techniques to solve the task of automatic detection of license plates. We used Python under anaconda as the development environment and the Tensorflow library. We created our own database and labeled the images manually and programmatically. The Faster R-CNN and SSD MobileNet deep learning algorithms were compared under CPU and GPU. The first proved more accurate but slower and the use of GPU is much faster for both. We show that overall we have obtained a high prediction accuracy.

Keywords : License Plate Detection ; Deep learning ; Tensorflow

Listes des acronymes et abréviations

CNN : convolutional Neural Network (Réseau neuronal convolutif).

AI : Intelligence Artificielle.

ML : Machine Learning.

DL : Deep Learning.

ALPD : Automatic License Plate Detection (détection de plaque d'immatriculation automatique).

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

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

SSD : Single Shot MultiBox Detector .

R-CNN : Region-based convolutional neural network.

ILSVRC : ImageNet Large Scale Visual Recognition Competition.

VGGNet : Visual Geometry Group Network.

VOC : Classes d'Objets Visuals.

mAP : Mean Average Precision.

SVM : Support Vector Machine (Machine à vecteurs de support).

HOG : Histogram of Oriented Gradients (histogramme de gradient orienté).

RNA : Réseau de neurones artificiels.

PMC : perceptron multicouche.

Table des matières :

Liste des figures

Liste des tableaux

Introduction générale

La reconnaissance automatique des plaques d'immatriculation (RAPI) aide à identifier les plaques d'immatriculation de véhicules de manière efficace sans recourir à des ressources humaines importantes et est devenue de plus en plus importante ces dernières années. Il y a plusieurs raisons pour lesquelles leur importance a augmenté. Il y a de plus en plus de voitures sur les routes et toutes ont des plaques d'immatriculation. Le développement rapide de la technologie de traitement d'image numérique a également permis de détecter et d'identifier rapidement les plaques d'immatriculation.

Un système RAPI typique peut être divisé en trois grandes étapes :

1. Détection de plaque d'immatriculation - détecte la plaque dans l'image capturée.
2. Segmentation de caractères - extraire les caractères alphanumériques de la plaque.
3. Reconnaissance des caractères - reconnaître chaque caractère individue.

L'identification des véhicules est utile pour opérateurs différents. Elle peut être utilisée par les agences gouvernementales pour détecter les voitures impliquées dans un crime, rechercher si des frais annuels sont acquittés ou identifier les personnes qui enfreignent les règles de circulation.

Ce projet présentera une approche de détection automatique des plaques d'immatriculation en utilisant Deep learning. Les techniques d'apprentissage profond n'utilisent pas de fonctionnalités conçues à la main, mais sélectionnent automatiquement les fonctionnalités elles-mêmes. Ils ont connu du succès dans de nombreux domaines de la vision artificielle, tels que la reconnaissance de l'écriture manuscrite [1] et la reconnaissance d'objet visuel [2]. Les méthodes d'apprentissage

en profondeur les plus efficaces font appel à des réseaux de neurones convolutif (CNN). Les réseaux de neurones hiérarchiques de CNN reposent sur des connexions clairsemées et un partage du poids, ce qui leur confère une immense capacité de représentation et un potentiel d'apprentissage élevé. Les plus grands défis des CNN sont leurs coûts de calcul élevés et leur demande pour de grandes quantités d'échantillons d'apprentissage.

L'objectif de ce projet est de pouvoir détecter les plaques d'immatriculation apparaissant sur les routes algériennes en utilisant les apprentissages profonds et particulièrement les réseaux neurone a convolution (CNN et RCNN). Cela signifie que la partie la plus importante du système consiste à pouvoir détecter les plaques d'immatriculation avec une grande précision, ce système ou algorithme pourrait plus tard être implémenter dans un système embarqué et cela dans le but de rendre l'identification et la détection des voiture facile par exemple pour la police, pendant les accidents, ou lors d'un vol.

Il existe un certain nombre de variantes qui posent des problèmes lors de la détection de ces plaques, comme : l'emplacement, la taille, la rotation, l'occlusion, la qualité de l'image, l'objets parasites.

Les approches classiques pour l'identification des plaques d'immatriculation utilisent la reconnaissance humaine à partir d'images saisies au préalable, ou le traitement d'image par des algorithmes de classification divers basés ou pas sur les réseaux de neurones classiques, ou d'autres algorithmes d'intelligences artificielle.

De telles méthodes requièrent l'implication de services techniques et administratifs et un temps de communication et de traitement important.

Une détection et reconnaissance automatique des plaques d'immatriculation, embarquées sur le système d'acquisition des images, permettraient la communication instantanée du matricule à la base de données pour l'extraction des renseignements relatif au véhicule et à son propriétaire.

La disponibilité de processeurs puissants a permis l'implémentation des algorithmes de machine learning d'intelligence artificielle impossible à exécuter jusqu'à récemment.

Les CPU modernes, et rapides, la mémoire RAM en quantité suffisante permettent de réaliser des apprentissages machine en quelques jours, les GPU encore plus puissants, et réservés au départ aux jeux, sont actuellement utilisés sur les PC et permettent d'exécuter les algorithmes de machine learning en quelques heures.

Nous utilisons deux méthodes dans notre projet pour exécuter l'algorithmes de DL, et établir une comparaison.

Nous avons utilisé la bibliothèque Tensorflow et les deux modèles de détection, Faster R-CNN Inception v2 et SSD-MobileNet v1.

L'apprentissage et le test ont été faites sur la base de données de véhicules algériennes que nous avons créée en plus des bases de données télécharger.

Le mémoire est organisé en trois chapitres :

Dans le premier chapitre, nous expliqueront les approches et les architectures de deep learning, notamment les CNN qui sont les théories de base nécessaire à ce projet.

Dans le second chapitre, nous parlerons d'abords des modèles de détection d'objet basique, ensuite nous parleront des modèles développer que nous avons utilisé dans notre projet.

Dans le dernier chapitre nous expliquerons les étapes de notre étude, les outils, la base de données utiliser et enfin représentons les résultats de notre étu

Chapitre 1 Etude des architectures de DL

1.1 Introduction

Notre travail consiste à utiliser l'apprentissage profond pour la détection des plaques d'immatriculation.

Le présent chapitre définit le deep learning, qui est une branche du machine learning, lui-même une branche de l'intelligence artificielle.

Il décrit auparavant le fonctionnement théorique des réseaux de neurones, ainsi que la structure des réseaux de neurones à convolution nécessaires à l'implémentation de l'apprentissage profond.

1.2 Les réseaux de neurones artificiels

1.2.1 Le neurone formel

Un neurone formel (artificiel) est une unité de traitement qui reçoit des données en entrées, sous la forme d'un vecteur, et produit une sortie réelle. Cette sortie est une fonction des entrées et des poids des connexions.[3]

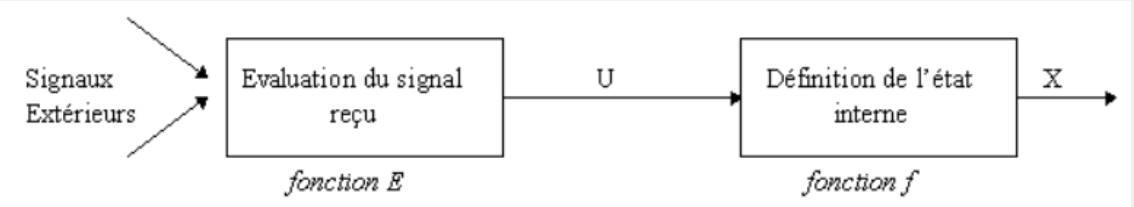

Figure 1-1: les parties de réseaux de neurone formel.

Le neurone formel, l'unité élémentaire d'un RNA, se compose de deux parties :

- Évaluation de la stimulation reçue (fonction E).
- Évaluation de son activation (fonction f) Il est caractérisé par :
  - Son état X (binaire, discret, continu).
  - Le niveau d'activation reçu en entrée U.
  - Le poids des connections en entrée W.

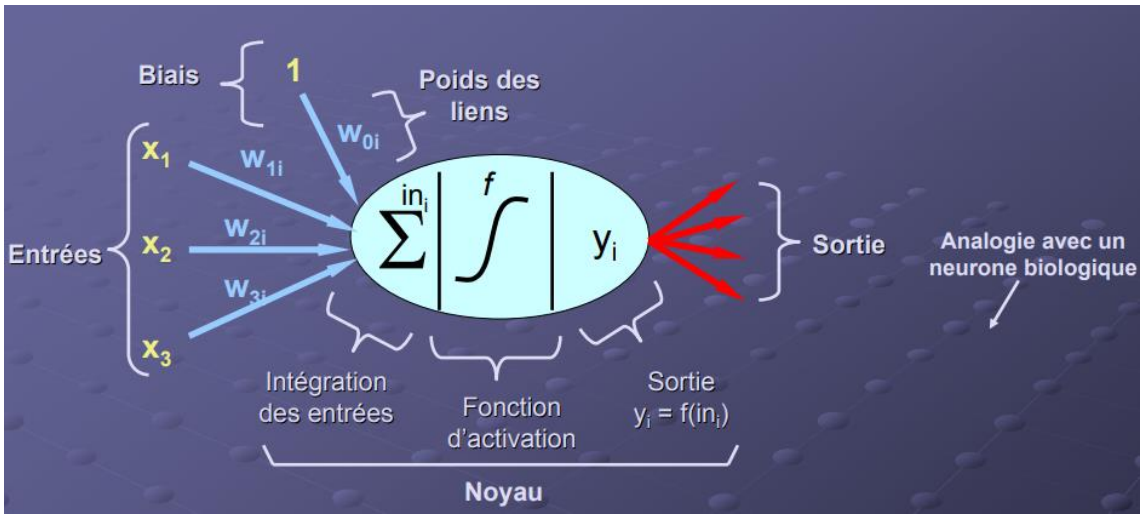

Figure 1-2 : les entrées et les sorties d'un réseau de neurone

Entrées : Directement les entrées du système ou peuvent provenir de d'autres neurones.

Biais : permet d'ajouter de la flexibilité au réseau en permettant de varier le seuil de déclenchement du neurone par l'ajustement du poids du biais lors de l'apprentissage.

Noyau : Intègre toutes les entrées et le biais et calcule la sortie du neurone selon une fonction d'activation qui est souvent non-linéaire pour donner une plus grande flexibilité d'apprentissage. Sortie : Directement une des sorties du système ou peut être distribuée vers d'autres neurones.[4]

1.2.2 Les réseaux de neurones

a Les topologies

- **Perceptron multicouche**

Le perceptron multicouche (PMC) est un réseau composé de couches successives une couche est un ensemble de neurones n'ayant pas de connexion entre eux une couche d'entrée lit les signaux entrants, un neurone par entrée xj, une couche en sortie fournit la réponse du système dans un perceptron, un neurone d'une couche cachée est connecté en entrée à chacun des neurones de la couche précédente et en sortie à chaque neurone de la couche suivante :

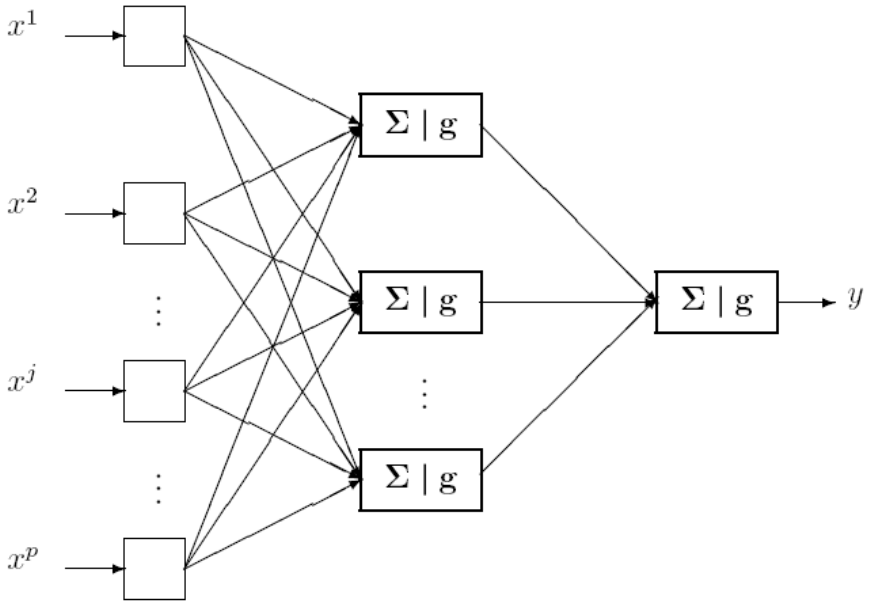

Figure 1-3 : schéma synoptique de PMC.[3]

- Les entrées du réseau sont les variables explicatives x1, . . . , xp
- Les poids sont des paramètres α à estimer lors de la procédure D'apprentissage
- La sortie est la variable y .

- **CNN**

Un CNN est constitué d'une succession de couches qui effectuent plusieurs opérations sur les données d'entrée. Tout d'abord, les couches convolutives C convoluent les images présentées à leurs entrées avec un nombre prédéfini de noyaux, k. Ces noyaux ont une certaine taille, s, et sont généralement suivis par des unités d'activation qui redimensionnent les résultats de la convolution de manière non linéaire. Les couches de regroupement réduisent la dimensionnalité des réponses produites par les couches de convolution par le biais d'un sous-échantillonnage

b Les fonctions d'activation

La fonction d'activation est une partie importante d'un réseau de neurones car elle définit la sortie d'un nœud à partir d'un ensemble d'entrées. Cela peut être vu comme un commutateur pouvant activer ou désactiver un neurone en fonction de l'entrée donnée. Des fonctions linéaires simples telles que la fonction d'identité f (x) = x peuvent être utilisées, mais pour rendre le réseau neuronal capable d'approcher également des fonctions non convexes, il est nécessaire d'utiliser une fonction d'activation non linéaire. Il y a principalement trois fonctions d'activation non linéaires.

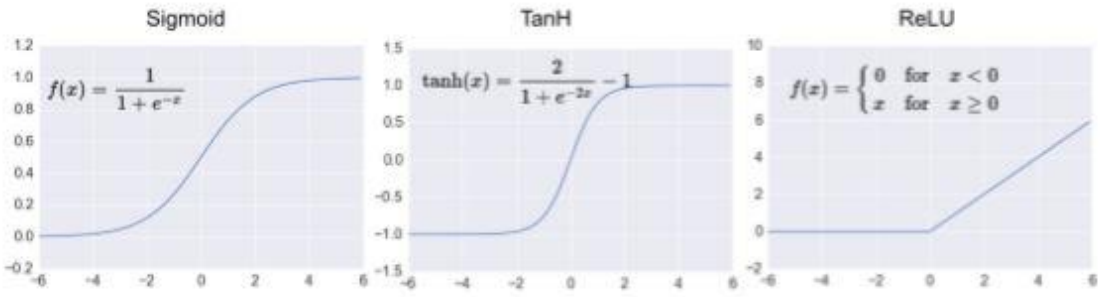

Figure 1-4 : Les graphes des fonctions d'activation

- **Sigmoïde** La fonction sigmoïde est donnée par

$$\boldsymbol{f(x) = \frac{1}{1 + e^{-1}}}$$ Équation 1

Et renvoie une valeur comprise entre 0 et 1. Il s'agit d'une bonne imitation d'un neurone qui s'active ou se désactive car il renvoie le plus souvent des valeurs proches de 0 ou 1. Le problème principal est que le gradient sur les queues est en train de disparaître. Ainsi, l'algorithme de rétropropagation modifie à peine les paramètres et l'apprentissage est ralenti.

- **Tangente hyperbolique (TanH)** La fonction TanH est donnée par

$$\boldsymbol{f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1}$$ Equation 2

Et renvoie une valeur comprise entre -1 et 1. Il a beaucoup des mêmes propriétés que la fonction sigmoïde, mais aussi le même problème principal.

- **Unité linéaire rectifiée (ReLU)** Le ReLU est donné par

$$\boldsymbol{f(x) = max(0, x)}$$ Equation 3

Et renvoie 0 ou une valeur réelle positive. La fonction ReLU a beaucoup été utilisée ces dernières années car elle présente des avantages

importants par rapport à la fonction sigmoïde et tanh. Il ne souffre pas de gradients en voie de disparition. Ainsi, il converge plus rapidement et accélère la formation. Cela implique également des opérations moins coûteuses car il ne calcule pas la fonction exponentielle coûteuse. Cependant, en raison de la suppression de toutes les valeurs négatives, il ne convient pas à toutes les architectures ni à tout la base de données.

1.2.3 L'apprentissage

a La perte

Pour un modèle, l'apprentissage signifie déterminer les bonnes valeurs pour toutes les pondérations et le biais à partir d'exemples étiquetés. Dans l'apprentissage supervisé, un algorithme de Machine Learning crée un modèle en examinant de nombreux exemples, puis en tentant de trouver un modèle qui minimise la perte. Ce processus est appelé minimisation du risque empirique.

La fonction la plus utilisée est l'erreur quadratique moyenne.

$$\boldsymbol{MSE} = \frac{\mathbf{1}}{\boldsymbol{N}}\sum_{\boldsymbol{i=0}}^{\boldsymbol{N}}|\boldsymbol{y}-\boldsymbol{x}|^2 \qquad \text{Equation 4}$$

L'erreur quadratique moyenne (MSE) correspond à la perte quadratique moyenne pour chaque exemple. Pour calculer l'erreur MSE, il faut additionner toutes les pertes quadratiques de chaque exemple, puis diviser cette somme par le nombre d'exemples.[5]

b Back propagation

En statistiques, la rétropropagation du gradient est une méthode pour calculer le gradient de l'erreur pour chaque neurone d'un réseau de neurones, de la dernière couche vers la première. De façon abusive, on appelle souvent technique de rétropropagation du gradient l'algorithme classique de correction des erreurs basé sur le calcul du gradient grâce à la rétropropagation. Cette technique consiste à corriger les erreurs selon l'importance des éléments qui ont justement participé à la réalisation de ces erreurs. Dans le cas des réseaux de neurones, les poids synaptiques qui contribuent à engendrer une erreur importante se verront modifiés de manière plus significative que les poids qui ont engendré une erreur marginale.[6]

1.2.4 CNNs

a Convolution

Réseau de neurones consolatifs largement utilisé pour la reconnaissance d'image, cette technique importante pour l'amélioration des couches profondes pour le traitement de l'information (images).

La convolution de deux signaux nous donne un autre signal contenant une meilleure représentation caractéristique. Convolution à temps continue.

L'objectif de cette couche est de détecter les caractéristiques telles que les bords, les taches de couleur et d'autres éléments visuels, la convolution d'image avec le filtre crée des images appelées cartes de caractéristiques de sortie, plus de filtres dans les couches de convolution plus des caractéristiques ont détecté. L'opération de convolution commence dans le coin supérieur gauche de la Sous-matrice.[7]

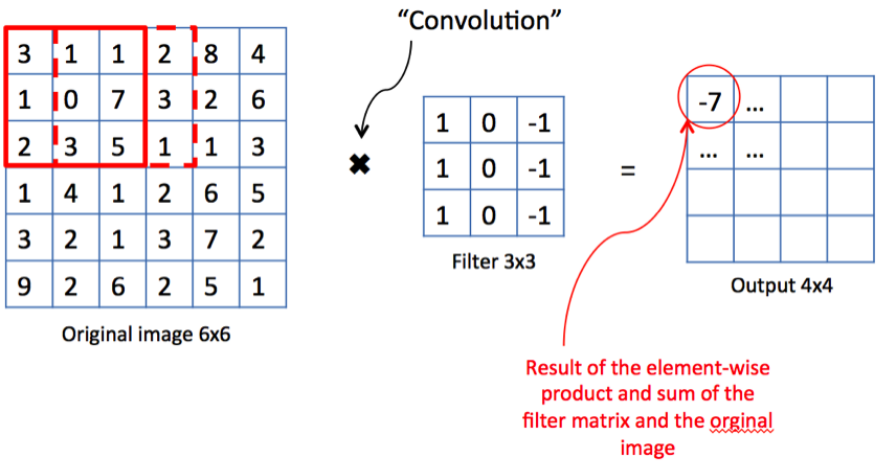

Figure 1-5 : l'opération convolution d'une image avec un filtre 3x3.

b Pooling

Un autre concept important des CNN est le pooling (mise en commun), ce qui est une forme de sous-échantillonnage de l'image. L'image d'entrée est découpée en une série de rectangles de n pixels de côté ne se chevauchant pas (pooling). Chaque rectangle peut être vu comme une tuile. Le signal en sortie de tuile est défini en fonction des valeurs prises par les différents pixels de la tuile.

Le pooling réduit la taille spatiale d'une image intermédiaire, réduisant ainsi la quantité de paramètres et de calcul dans le réseau. Il est donc fréquent d'insérer périodiquement une couche de pooling entre deux couches convolutives successives d'une architecture CNN pour contrôler l'overfitting (sur-apprentissage). L'opération de pooling créait aussi une forme d'invariance par translation.

La couche de pooling fonctionne indépendamment sur chaque tranche de profondeur de l'entrée et la redimensionne uniquement au niveau de la surface. La forme la plus courante est une couche de mise en commun avec des tuiles de taille 2x2 (largeur/hauteur) et comme valeur de sortie la valeur maximale en entrée. On parle dans ce cas de Max-Pool 2x2.

Il est possible d'utiliser d'autres fonctions de pooling que le maximum. On peut utiliser un average pooling (la sortie est la moyenne des valeurs du patch d'entrée), du L2-norm pooling. Dans les faits, même si initialement l'average pooling était souvent utilisé il s'est avéré que le max-pooling était plus efficace car celui-ci augmente plus significativement l'importance des activations fortes. En d'autres circonstances, on pourra utiliser un pooling stochastique.

Le pooling permet de gros gains en puissance de calcul. Cependant, en raison de la réduction agressive de la taille de la représentation (et donc de la perte d'information associée), la tendance actuelle est d'utiliser de petits filtres (type 2x2). Il est aussi possible d'éviter la couche de pooling mais cela implique un risque sur-apprentissage plus important.

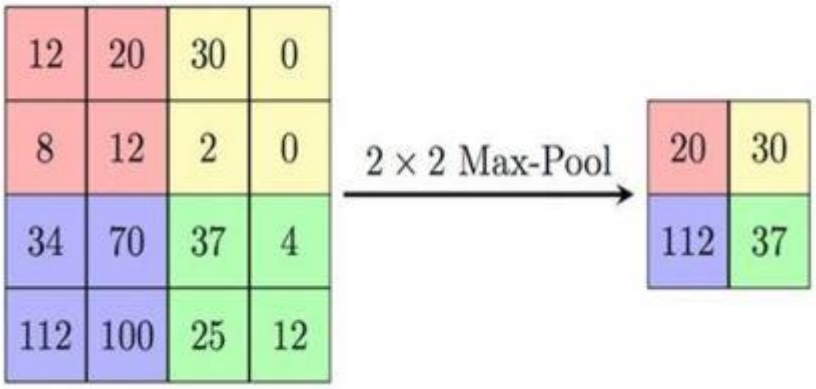

Figure 1-6 : Exemple de Max-Pooling 2x2

c La couche entièrement connecté (Fully-connected)

La couche fully-connected constitue toujours la dernière couche d'un réseau de neurones, convolutif ou non – elle n'est donc pas caractéristique d'un CNN.

Ce type de couche reçoit un vecteur en entrée et produit un nouveau vecteur en sortie. Pour cela, elle applique une combinaison linéaire puis éventuellement une fonction d'activation aux valeurs reçues en entrée.

La dernière couche fully-connected permet de classifier l'image en entrée du réseau : elle renvoie un vecteur de taille N, où N est le nombre de classes dans notre problème de classification d'images. Chaque élément du vecteur indique la probabilité pour l'image en entrée d'appartenir à une classe.

Par exemple, si le problème consiste à distinguer les chats des chiens, le vecteur final sera de taille 2 : le premier élément (respectivement, le deuxième) donne la probabilité d'appartenir à la classe "chat" (respectivement "chien"). Ainsi, le vecteur [0.9 ; 0.1] signifie que l'image à 90% de chances de représenter un chat.

Chaque valeur du tableau en entrée "vote" en faveur d'une classe. Les votes n'ont pas tous la même importance : la couche leur accorde des poids qui dépendent de l'élément du tableau et de la classe.

Pour calculer les probabilités, la couche fully-connected multiplie donc chaque élément en entrée par un poids, fait la somme, puis applique une fonction d'activation (logistique si N=2, softmax si N>2) :

Ce traitement revient à multiplier le vecteur en entrée par la matrice contenant les poids. Le fait que chaque valeur en entrée soit connectée avec toutes les valeurs en sortie explique le terme fully-connected. [8]

1.3 Les architectures de classification par convolution

Différentes architectures de convolution ont été développées à partir des années 1990. Cette section examinera certaines des architectures les plus remarquables. Comme nous le verrons, de nombreuses architectures révolutionnaires ont été construites et soumises au concours de reconnaissance visuelle à grande échelle ImageNet (ILSVRC). Il s’agit d’un concours annuel dont l’objectif est de rassembler au

maximum les classes, les classes, les classes, les images et au-delà de 1000 classes. Cette compétition a été, ces dernières années, l'élément moteur du développement futur des CNN. Toutes les architectures présentées résolvent les tâches générales de classification des objets dans une image et peuvent être de bons candidats pour l'apprentissage par transfert afin de répondre aux besoins de ce projet.

1.3.1 CNNs traditionnels

LeNet5 [9], développé par Yann LeCun, était un pionnier de CNN qui a accéléré la recherche sur l'apprentissage en profondeur. Il était utilisé pour reconnaître les codes postaux et les chiffres. L'architecture a inspiré les CNN plus tard, mais elle diffère fondamentalement des CNN modernes car les GPU n'étaient pas disponibles pour la formation à l'époque et les processeurs étaient beaucoup plus lents. Ainsi, seul un bref résumé de l'architecture est présenté. Il utilisait alternativement des couches de convolution et de regroupement avec des couches entièrement connectées comme classificateurs finaux. En tant que fonction d'activation non linéaire, la tangente hyperbolique ou le sigmoïde a été utilisée et la mise en commun moyenne a été utilisée comme fonction de mise en commun. Il y avait des connexions éparses entre les couches pour réduire le nombre de dimensions. L'erreur quadratique moyenne a été utilisée comme fonction de perte.

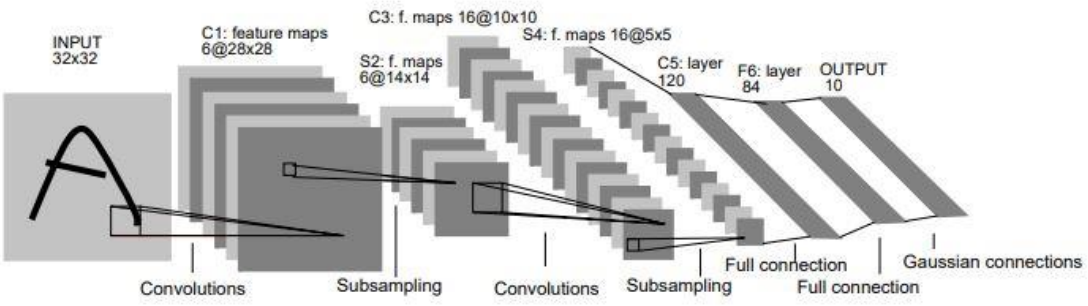

Figure 1-7 : Architecture LeNet5 utilisée pour la reconnaissance des chiffres.[9]

1.3.2 DeepCNNs

AlexNet [10] a participé à la compétition ILSVRC 2012 et a remporté une victoire considérable (84% de précision top5 par rapport à 74% pour le deuxième prix). Il est basé sur LeNet, mais est beaucoup plus profond et large avec un total de 60 millions de paramètres. En raison du nombre de paramètres, il a été implémenté pour utiliser plusieurs GPU afin de répondre aux besoins en mémoire.

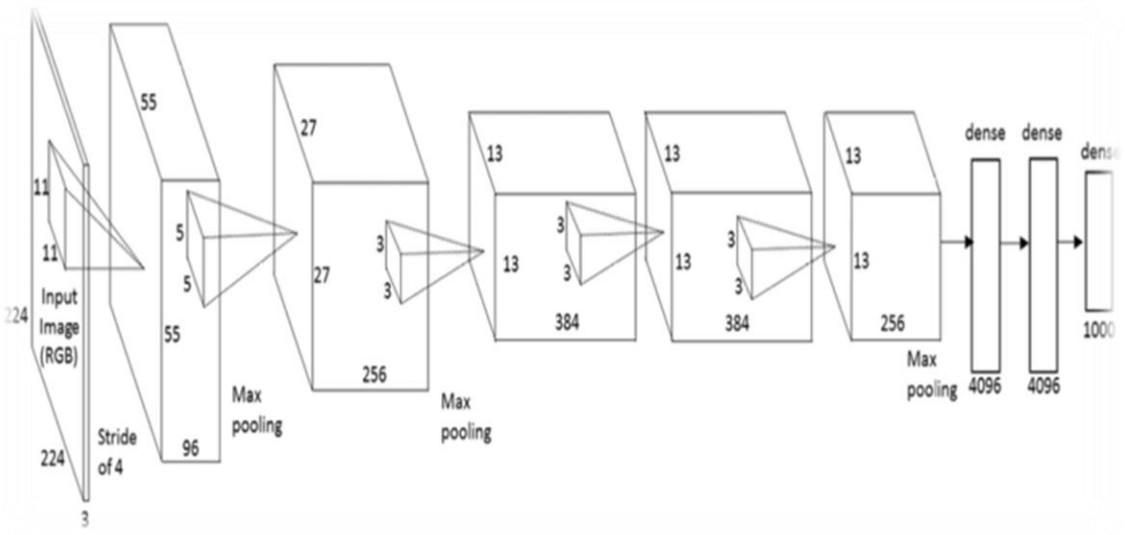

Figure 1-8 : Mise en œuvre multi-GPU AlexNets. [10]

1.3.3 Very deep CNNs

GoogLeNet [11], également appelé Inception, a remporté le ILSVRC 2014. Inspiré de VggNet [12], il utilisait des filtres 3 par 3 plus petits dans chaque couche convolutif au lieu des filtres 5 par 5 ou 7 par 7. Cette idée ouvre la possibilité d'augmenter le nombre de couches et de construire des réseaux plus profonds. Sa propre contribution était de réduire considérablement le nombre de paramètres utilisés. Ceci a été réalisé en utilisant des filtres bon marché 1 sur 1 avant des blocs parallèles coûteux, appelés goulot d'étranglement. Pour réduire encore le nombre de paramètres, le regroupement moyen a été utilisé comme classificateurs au lieu de couches entièrement connectées. Le développement de Inception s'est poursuivi et la troisième et dernière version est décrite dans l'InceptionV4 [13].

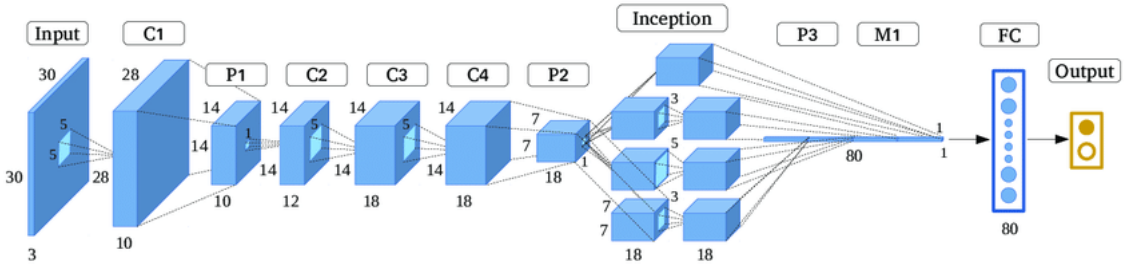

Figure 1-9 : Architecture GoogLeNet [11]

1.3.4 Residual CNNs

ResNet [14] a remporté le ILSVRC 2015 et représente une nouvelle façon révolutionnaire de construire des CNN, appelés CNN résiduels. Il se compose de 152 couches, une augmentation extrême par rapport aux architectures précédentes. Une connexion de saut est une connexion utilisée par le signal d'entrée pour contourner un certain nombre de couches. Avec cette technique, les CNN comportant plus de 1000 couches peuvent être entraînés. Des recherches sur les raisons pour lesquelles cela fonctionne si bien sont toujours

en cours. Des recherches empiriques ont montré que ResNet fonctionnait avec des blocs d'une profondeur modérée d'environ 20 à 30 couches agissant en parallèle, au lieu d'un flux en série de l'ensemble du réseau [15]. Ainsi, le comportement peut être comparé à des ensembles de réseaux relativement peu profonds. De plus, lorsque la sortie est renvoyée de manière récursive, comme dans les RNN, elle peut être considérée comme un modèle biologiquement plausible du flux ventral dans le cortex visuel [16].

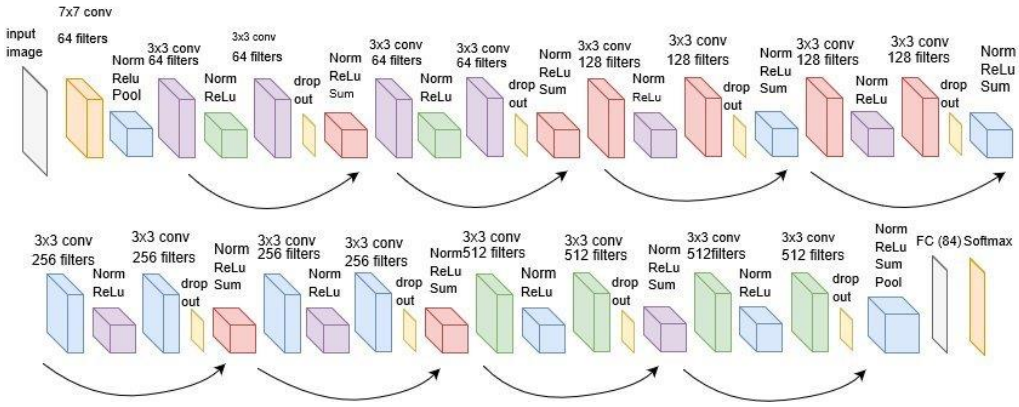

Figure 1-10 : Architecture ResNet [14]

1.4 Conclusion

Dans ce chapitre nous avons parlé des réseaux neurones et leurs topologie, nous avons aussi entamer le terme de deep Learning et enfin nous avons particulièrement expliquer les CNN et leurs différentes couches car ils sont la base de notre projet .

Dans le prochain chapitre, nous expliquerons les algorithme de détection, notamment les RCNN , fast RCNN….etc. que nous avons utilisé dans ce projet.

Chapitre 2 LES MODELES DES DETECTIONS

2.1 Introduction

Nous Présentons dans ce chapitre les algorithmes Region-based CNNs, Fast R-CNN, Faster R-CNN et Single Shot Multibox Detector, parce qu'ils sont utilisés pour la détection d'objets par la proposition de région confinées dans les boites englobantes.

Nous pouvons ainsi choisir l'architecture la plus appropriée pour notre application.

2.2 Architectures de détection par convolution

Les architectures de convolution utilisées pour la détection d'objet diffèrent des CNN de classification qui tentent également de déterminer l'emplacement de l'objet. C'est une tâche plus ardue, mais elle est plus efficace car elle peut être utilisée à la fois pour détecter, segmenter et classer des objets dans une image.

Le défi des classes d'objets visuels (VOC) [17] est un concours annuel de détection d'objets. Leurs jeux de données publiés comprenant des images d'objets appartenant à différentes classes avec des annotations de localisation sont souvent utilisés pour comparer des réseaux de détection. La précision moyenne (mAP) est utilisée comme mesure d'évaluation standard.

Jusqu'à ces dernières années, les algorithmes de détection d'objets avaient une faible précision compte tenu de la localisation et de la classification correctes des objets. Les machines à vecteurs de support (SVM) utilisant les fonctions HOG [18] et les modèles de pièces déformables (DPM) utilisant des modèles de racine et de pièce étaient à la pointe de la technologie, mais n'atteignaient pas une grande

précision. L'un des meilleurs DPM basés sur HOG n'a obtenu qu'un score mAP de 33,7 sur l'ensemble de données VOC 2007 [19].

La découverte de CNN basés sur les régions (R-CNN) et les améliorations suivantes des R-CNN ont révolutionné le domaine de la détection d'objets en améliorant considérablement la précision. Les architectures one-shot plus récentes ont encore amélioré le manque de vitesse de traitement dans les réseaux R-CNN en supprimant le pipeline séquentiel de propositions de régions et de classification d'objets et en traitant le problème global dans son ensemble en conservant la même précision de prédiction. Les sous-sections suivantes présentent les architectures les plus pertinentes et examinent leur exactitude de prévision et leur vitesse de traitement.

2.2.1 Region-based CNNs

Girshick et al. Construit un CNN basé sur la région (R-CNN) en 2014 [20] pour détecter des objets dans une image. Il a amélioré la détection par rapport aux meilleurs résultats précédents sur le jeu de données VOC 2007 de moins de 50% et de parvenir à un score de réduction de 66. La vitesse est plutôt lente et le temps de traitement par image est de 20 secondes. Globalement, le système comprend trois modules.

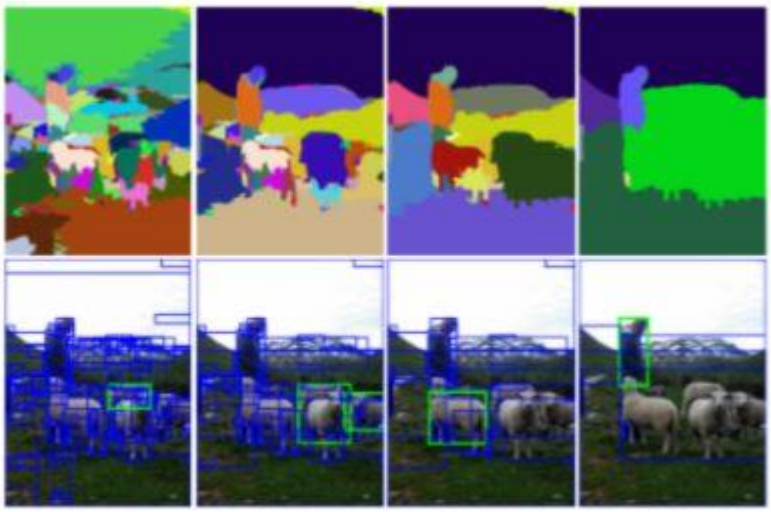

Figure 2-1 : Des cadres de sélection dans les images inférieures sont proposés sur la base de groupes de pixels adjacents identifiés comme objets par la méthode de recherche sélective dans les images supérieures. [21]

Le premier module génère des propositions de région indépendantes de la catégorie, également appelées boîtes englobantes, avec l'approche proposée dans [22]. Les limites d'occlusion sont étiquetées à l'aide d'approches traditionnelles basées sur les contours et les régions, ainsi que de méthodes d'extraction de caractéristiques 3D de surface et de profondeur [23]. Une interprétation 3D de l'image est reconstruite et avec apprentissage supervisé, elle est en mesure de proposer des régions indépendantes de la catégorie. Une recherche sélective [24] est effectuée pour éviter une recherche exhaustive tout en ayant une recherche diversifiée. À un niveau élevé, cette recherche sélective examine l'image avec différentes tailles de fenêtre. Chaque taille tente de regrouper les pixels adjacents en fonction de leur couleur, de leur intensité et de leur texture pour identifier les objets.

Le deuxième module est un grand réseau convolutif qui extrait un vecteur de caractéristiques fixé à 4096 dimensions. Il se compose de cinq couches convolutives suivies de deux couches entièrement connectées utilisant la même architecture de réseau que celle décrite par GoogleNet [11]. Comme le

réseau n'autorise qu'une entrée de taille fixe, toutes les images sont déformées et converties en un vecteur 227 par 227.

Le dernier module introduit l'extracteur de caractéristiques fixes dans un SVM linéaire classifiant chaque région en catégories prédéfinies.

La dernière étape du R-CNN après la classification des régions consiste à déterminer s'il est possible d'améliorer et de resserrer les cadres de sélection trouvés. Ceci est effectué par un modèle de régression linéaire simple dans lequel les sous-régions correspondant aux objets sont prises en entrée et les nouvelles coordonnées améliorées du cadre de sélection de la zone de contour sont données en sortie.

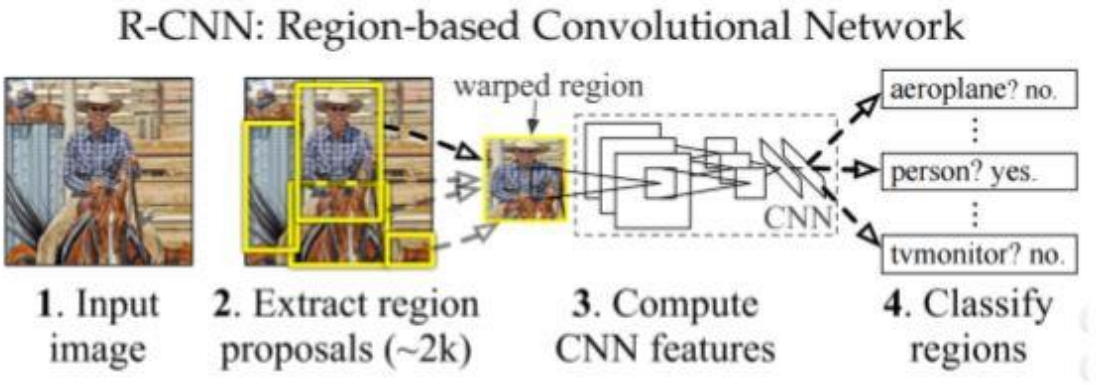

Figure 2-2 : Aperçu de R-CNN. 1. Une image d'entrée est donnée. 2. Les propositions de région sont extraites. 3. Le vecteur de fonction est calculé. 4. Chaque région est classifiée. [25]

2.2.2 Fast R-CNN

Fast R-CNN [26] (2015) est une version améliorée et simplifiée du modèle R-CNN existant de Girshick et al. Les modifications ont permis d'améliorer la vitesse et la précision. Fast R-CNN a obtenu un score mAP de 70,0 sur le VOC 2007 avec un temps de traitement de 2 secondes par image, 10 fois plus rapide que R-CNN.

Une faiblesse importante du R-CNN est qu'elle nécessite un passage en aval via le CNN pour chaque proposition de région. Ces propositions se chevauchent

invariablement, obligeant le CNN à effectuer le même calcul plusieurs fois. La solution mise en œuvre dans Fast R-CNN consiste à partager les calculs avec RoIPool (pool de régions d'intérêt), technique étroitement liée aux réseaux de pooling de pyramides spatiales [27]. RoIPool calcule uniquement une seule carte de caractéristiques convolutifs partagée sur toute l'image d'entrée et un vecteur de caractéristiques de longueur fixe est extrait de la carte de caractéristiques pour chaque proposition d'objet. Cela réduit le nombre de passages en aval d'environ 2000 à un seul passage en avant, massivement améliorer la vitesse du réseau.

L'autre faiblesse de R-CNN est le nombre de modules. R-CNN utilise trois modules différents en plus du modèle de régression linéaire utilisé pour resserrer les cadres de sélection. Cela rend le pipeline séquentiel très difficile à former. Fast R-CNN améliore cet aspect en combinant tous les modules en un seul réseau. La formation est convertie en un processus unique, combinant l'apprentissage pour classer les objets et leur emplacement. Cela améliore la vitesse d'entraînement. Les vecteurs caractéristiques sont introduits dans une séquence de couches fully-connected se ramifiant dans deux couches en sortie. Une fois que la probabilité d'obtention d'une classe d'objet est atteinte, la seconde renvoie quatre valeurs représentant les angles de l'emplacement estimé. Une couche de régression linéaire est introduite. Cela resserre automatiquement les limites du réseau. Enfin, le classificateur SVM est remplacé par un classificateur softmax. Un autre avantage du réseau à module unique est que les couches de réseau peuvent être mises à jour pendant la formation, ce qui n'était pas possible dans R-CNN, ce qui améliore la précision.

2.2.3 Faster R-CNN

Faster R-CNN [28] (2016) est le résultat d'améliorations apportées au Fast R-CNN en changeant complètement le proposant de région. La vitesse et la précision ont été améliorées. Faster R-CNN a obtenu un score mAP de 73,2 avec un temps de traitement de 140 ms par image. C'est 10 fois plus rapide que

Fast R-CNN et plus de 100 fois plus rapide que R-CNN. Un score mAP encore plus élevé de 76,4 est obtenu en modifiant l'architecture sous-jacente de VGGnet à ResNet.

Faster R-CNN contribue à Fast R-CNN en remplaçant la recherche sélective par un réseau de proposition de région (RPN) accélérant massivement l'étape de proposition de région. RPN est un réseau convolutif entièrement connecté prenant une image en entrée et en sortie des propositions de région rectangulaire. Cela fonctionne en glissant une fenêtre sur la carte de fonctionnalité. À chaque emplacement, k cases d'ancrage avec 4 coordonnées sont proposées ainsi que deux scores de confiance pour chaque case d'ancrage, un score pour la précision des limites de la région et un pour l'objectivité de l'objet trouvé dans la case d'ancrage. Une boîte à ancre est une boîte englobante choisie de manière sélective. Intuitivement, lorsqu'un réseau est formé à la détection d'humains, nous voulons des boîtes englobantes rectangulaires ressemblant à la forme d'un humain. Des zones de délimitation hautes et fines apparaissent souvent, tandis que des zones de délimitation larges et basses apparaissent rarement. Le RPN essaie d'exploiter ce fait en ne suggérant que des boîtes d'ancrage avec des ratios d'aspect communs améliorant la précision de la localisation et réduisant le nombre de propositions de régions. Cela améliore la vitesse.

Les propositions de région sont ensuite transmises au réseau Fast R-CNN en vue de la détection et de la classification des objets dans les régions. Le réseau RPN et le réseau Fast R-CNN sont formés de manière indépendante en utilisant la rétropropagation et la descente de gradient stochastique. Pour que le réseau Fast R-CNN utilise pleinement les propositions de région données par le RPN, les deux réseaux doivent partager des fonctionnalités. Pour gérer cela, deux étapes sont introduites après que les deux réseaux ont terminé la formation de manière indépendante. Premièrement, le réseau Fast R-CNN est utilisé pour initialiser le réseau RPN, mais les couches convolutives partagées sont conservées fixes tandis que les couches uniques de RPN sont ajustées.

Deuxièmement, les couches uniques de Fast R-CNN sont finement ajustées. Cette résolution partagée peut toujours être réparée.

En partageant des fonctionnalités convolutifs, l'étape de proposition de la région est maintenant presque gratuite, supprimant ainsi un goulot d'étranglement important dans le réseau et améliorant considérablement la vitesse.

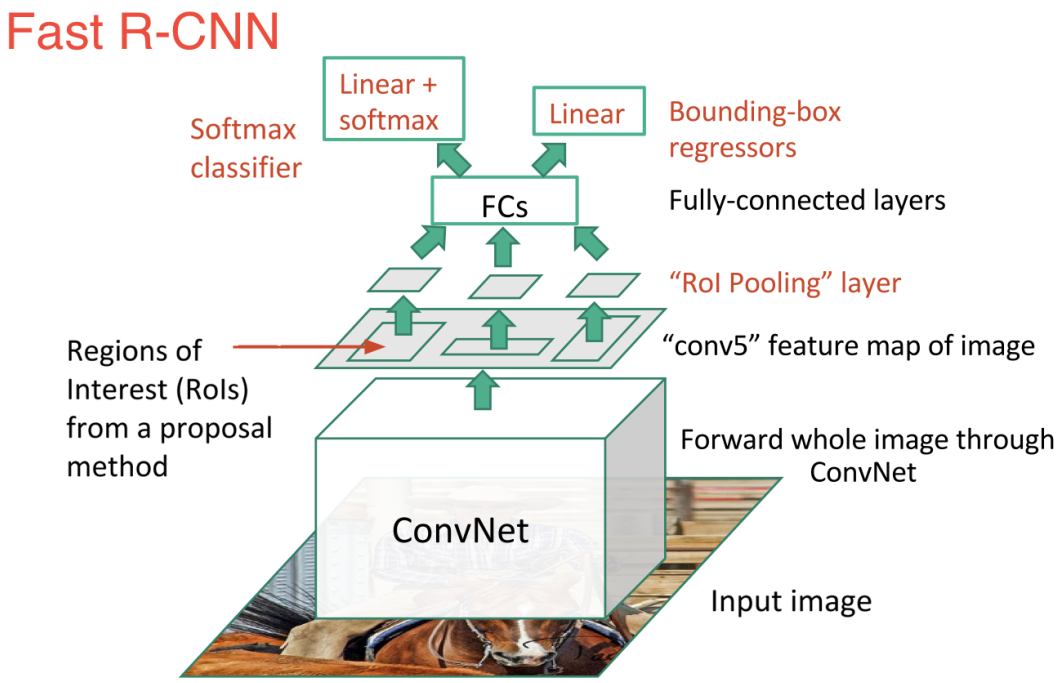

Figure 2-3 : Architecture de Faster R-CNN

2.2.4 Single Shot MultiBox Detector

L'approche SSD est basée sur un réseau de convolution à feed-forward qui produit une collection de boîtes englobantes et de scores de taille fixe pour la présence d'instances de classe d'objets dans ces boîtes, suivie d'une étape de suppression non maximale pour produire les détections finales. Les premières couches de réseau reposent sur une architecture standard utilisée pour la classification des images de haute qualité (tronquée avant toute couche de classification), qui s'appelle le réseau de base. [29]

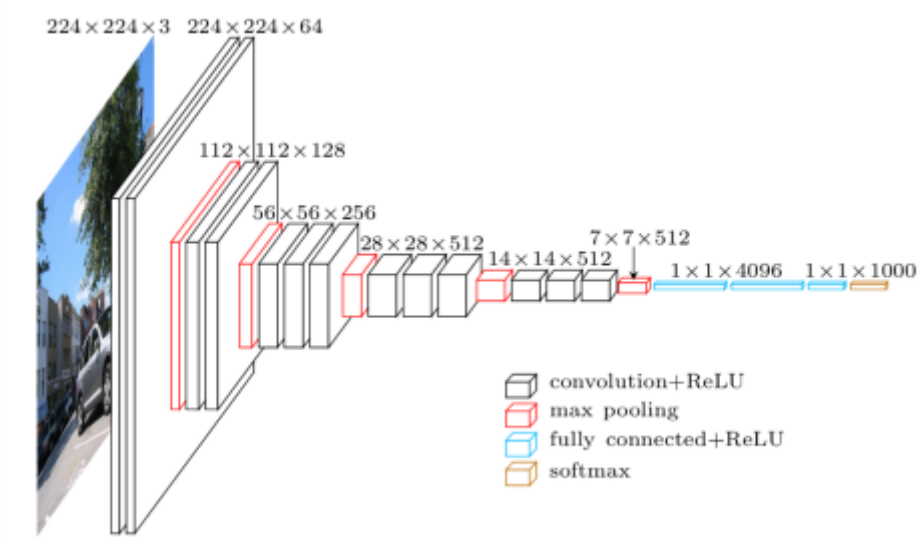

Figure 2-4 : Architecture de Single Shot MultiBox Detector

2.3 Conclusion

Dans ce second chapitre, nous avons expliqué les différents modèles de détection plus particulièrement les modèles utiliser dans notre projet.

Dans le prochain chapitre, nous représentant les résultats de notre détection, et les étapes d'installations et de téléchargement des différents outils nécessaire.

Chapitre 3 Expérimentations et résultats

3.1 Introduction

Pour implémenter un modèle de Deep Learning pour la détection des plaques d'immatriculation nous utilisons le langage de programmation Python et la bibliothèque Tensorflow, d'abord on doit installer Python sur notre PC et l'environnement Anaconda qui permet de faciliter la programmation, ensuite on installe toutes les bibliothèques nécessaires. Nous créons par la suite la base d'apprentissage constituée images de plaques que nous avons prises nous-mêmes ajoutes à celle que nous avons téléchargées. Nous procédons finalement à l'exécution do notre apprentissage profond et ce chapitre présente les différent logiciels et outils utilisé, les données, l'architecture du réseau CNN et les résultats obtenus.

3.2 Le langage de programmation utilisé (Python)

Bien qu'il ne soit pas le langage de programmation le plus rapide ni le plus sûr, Python est le plus utilisé dans le domaine de l'intelligence artificielle et surtout en deep Learning.

Nous avons choisi d'installer python dans l'environnement ANACONDA à cause des nombreuses fonctionnalités disponibles.

Python contient un nombre important de bibliothèques performantes et utiles pour la vision artificielle, les réseaux de neurones et pour la gestion des données, notamment Tensorflow et Theano pour le deep learning, Opencv et Pillow pour la vision artificielle.

L'organigramme (Figure 3-1) illustre les différentes bibliothèques et les outils de notre travail.

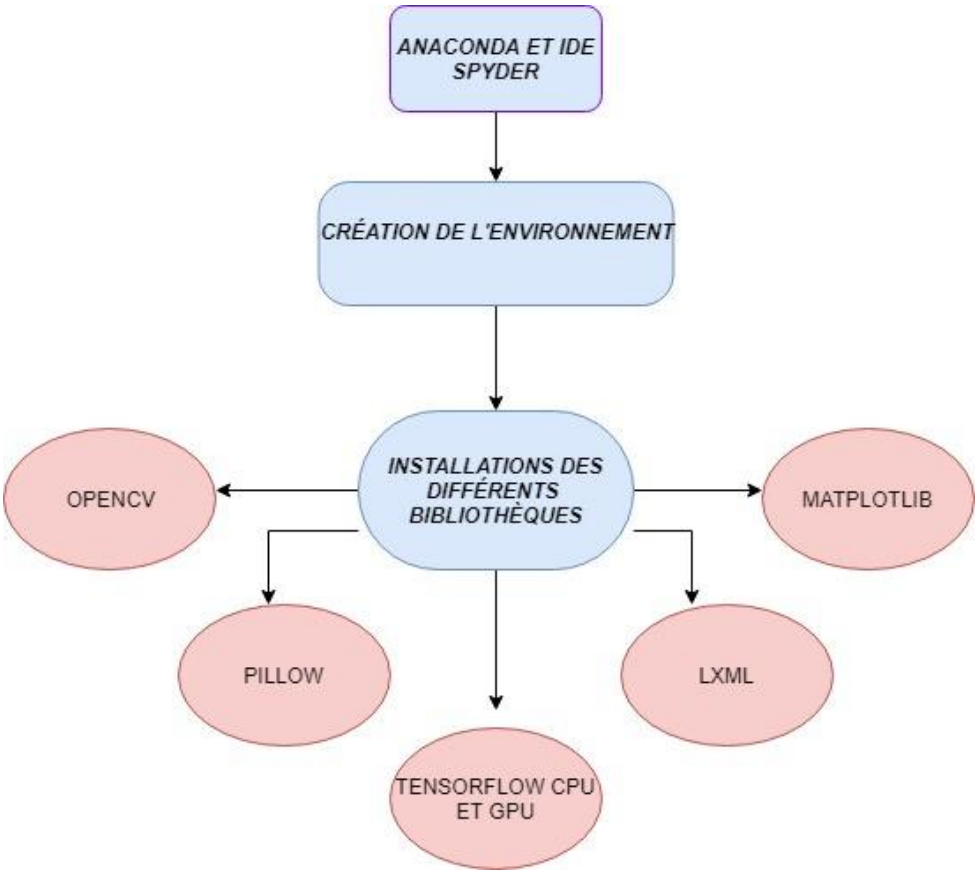

Figure 3-1 : Les bibliothèques et les outils utilisé

3.3 L'environnement utilisé (Anaconda)

Sur Windows, plusieurs options se présentent pour utiliser python, parmi eux en distingue l'environnement qui possèdes de nombreuses fonctionnalités dont l'analyse des data.

Anaconda est une distribution libre et open source des langages de programmation Python et R appliqué au développement d'applications dédiées à la science des données et à l'apprentissage automatique (traitement de données à grande échelle, analyse prédictive, calcul scientifique), qui vise à simplifier la gestion des paquets et de déploiement4. Les versions de paquetages sont gérées par le système de gestion de paquets conda5. La distribution Anaconda est utilisée par plus de 6 millions d'utilisateurs et comprend plus de 250 paquets populaires en science des données adaptés pour Windows, Linux et MacOs.

On a choisi Spyder 3 IDE (nommé Pydee dans ses premières versions) est un environnement de développement pour Python. Libre (Licence Massachusetts Institute of Technology (MIT)) et multiplateforme (Windows, Mac OS, GNU/Linux), il intègre de nombreuses bibliothèques d'usage scientifique : Matplotlib, NumPy, SciPy et IPython.

En comparaison avec d'autres IDE pour le développement scientifique, Spyder a un ensemble unique de fonctionnalités - multiplateforme, open-source, écrit en Python et disponible sous une licence non-copyleft. Spyder est extensible avec des plugins, comprend le support d'outils interactifs pour l'inspection des données et incorpore des instruments d'assurance de la qualité et d'introspection spécifiques au code Python, tels que Pyflakes, Pylint et Rope.

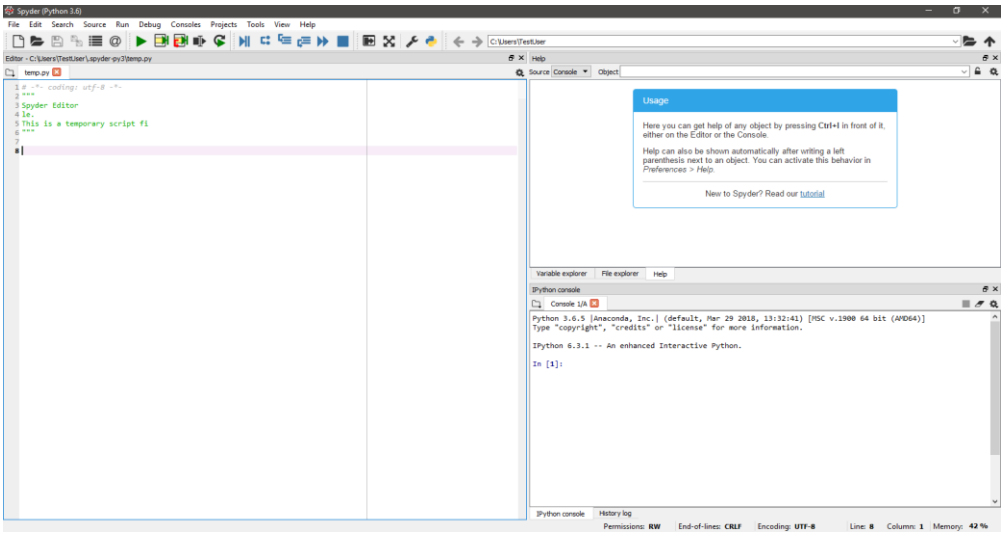

Figure 3-2 : IDE spyder interface sur windows

3.4 Création de l'environnement

Pour créer un environnement, nous avons utilisé l'invite de commande anaconda et utilisé la commande suivante :

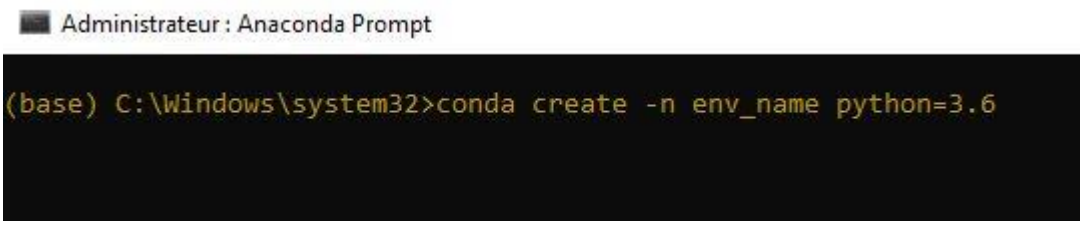

Figure 3-3 : Création d'un environnement dans anaconda

Après la création de l'environnement, nous l'activons par cette commande :

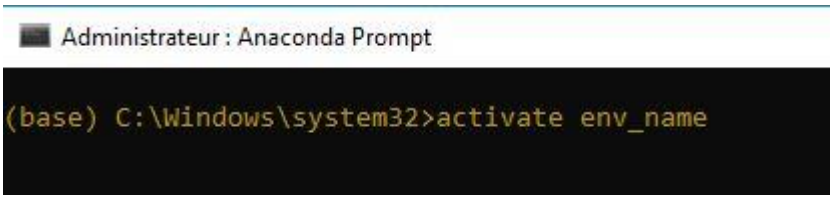

Figure 3-4 : Activation d'un environnement

Il y a deux commandes pour installer des bibliothèques dans Anaconda, conda install et pip install.

3.4.1 Installation des bibliothèques

a OpenCV

OpenCV (pour Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. La société de robotique Willow Garage et la société ItSeez se sont succédé au support de cette bibliothèque. Depuis 2016 et le rachat de ItSeez par Intel, le support est de nouveau assuré par Intel.

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

Fonctionnalités :

- Traitement d'images
- Traitement vidéo
- Algorithmes d'apprentissages
- Calculs Matriciels

b PILLOW :

Python Imaging Library (ou PIL) est une bibliothèque de traitement d'images, Elle permet d'ouvrir, de manipuler, et de sauver différents formats de fichiers graphiques.

La bibliothèque supporte plusieurs formats de fichier, parmi lesquels PNG, JPEG, GIF, TIFF, et BMP. Il est aussi possible d'ajouter son propre décodeur de fichiers pour étendre le nombre de formats disponibles.

c Pandas

Pandas est une bibliothèque écrite pour le langage de programmation Python permettant la manipulation et l'analyse des données. Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques et de séries temporelles. Pandas est un logiciel libre sous licence BSD.

Les principales structures de données sont les séries (pour stocker des données selon une dimension - grandeur en fonction d'un index), les DataFrames (pour stocker des données selon 2 dimensions - lignes et colonnes), les Panels (pour représenter des données selon 3 dimensions, les Panels4D ou les DataFrames avec des index hiérarchiques aussi nommés MultiIndex (pour représenter des données selon plus de 3 dimensions - hypercube)

d LXML

lxml est une bibliothèque Python qui permet de manipuler facilement les fichiers XML et HTML et peut également être utilisée pour le nettoyage

Web. Il existe de nombreux analyseurs XML standard, mais pour obtenir de meilleurs résultats, les développeurs préfèrent parfois écrire leurs propres analyseurs XML et HTML. C'est à ce moment que la bibliothèque lxml entre en jeu. Les principaux avantages de cette bibliothèque sont sa simplicité d'utilisation, son extrême rapidité lors de l'analyse de documents volumineux, son excellente documentation et sa facilité de conversion des données en types de données Python, ce qui simplifie la manipulation des fichiers.

e MATPLOTLIB

Matplotlib est une bibliothèque destinée à tracer et visualiser des données sous formes de graphiques. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy. Compatible avec la version 3 de Python.

f Tensorflow

Tensorflow est un logiciel bibliothèque open-source utilisé dans les applications des réseaux de neurones, il a été développé par l'équipe Google Brain de Google et le code a été ouvert en 2015.

Pour utiliser Tensorflow il est recommandé d'avoir un GPU NVIDIA® pour réduire le temps d'exécution énormément comparé au CPU (de quelques jours à quelque heures).

Tensorflow est disponible pour CPU et pour GPU.

3.4.2 Installation de Tensorflow CPU

Il y a deux méthodes pour installer Tensorflow pour le CPU, 1ère méthode en utilisant ''pip '' comme est montré dans la figure 3-5

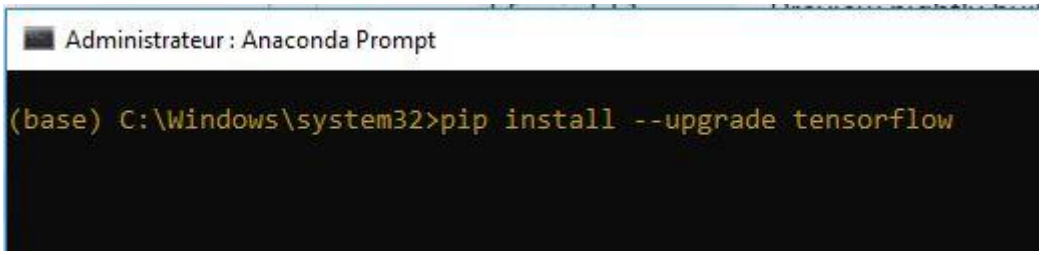

Figure 3-5 : Installation de Tensorflow CPU avec pip

Ou bien par l'utilisation de ''conda'' voir la figure 3-6

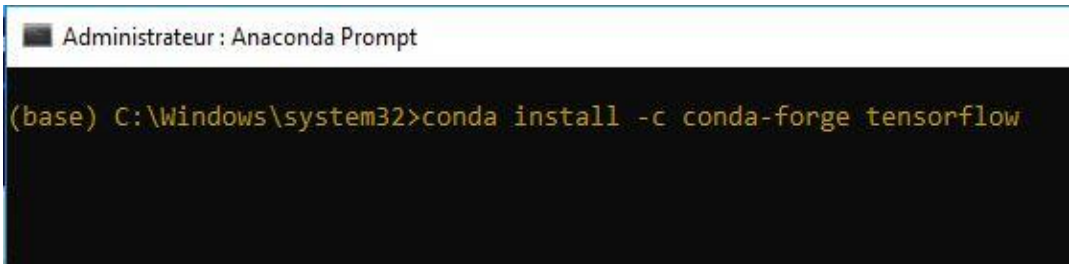

Figure 3-6 : Installation de Tensorflow CPU avec conda

3.4.3 Installation de Tensorflow GPU

Le diagramme suivant montrer les différentes étapes pour installer Tensorflow GPU.

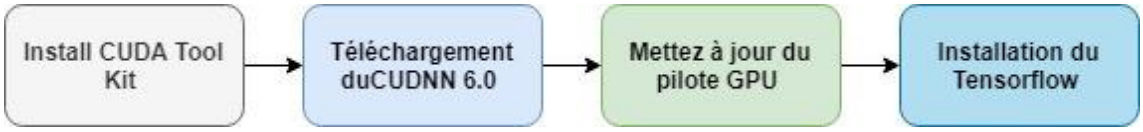

Figure 3-7 : Le diagramme des étapes de l'installation de Tensorflow GPU

La dernière étape est l'installation de Tensorflow avec la commande pip, la figure 3-8 présente la commande utilisée pour l'installation.

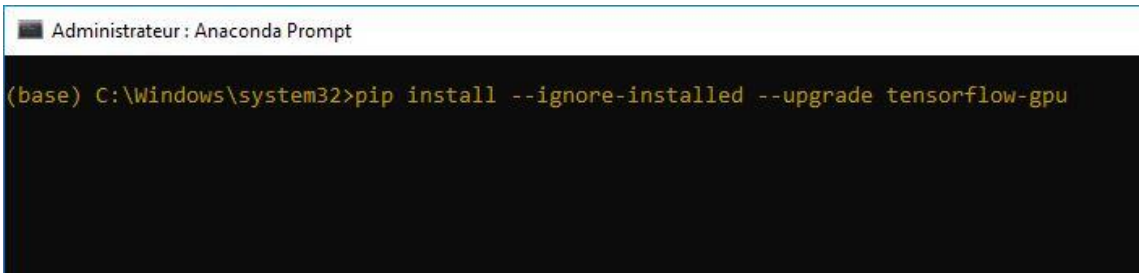

Figure 3-8: Installation de la bibliothèque Tensorflow GPU avec pip

Remarque : nous n'avons pas utiliser les versions des bibliothèques les plus récentes, on a choisi les plus compatible.

On a utilisé Cuda toolkit 8.0 et Cudnn 6.0 avec Tensorflow 1.13 parce que la version de Tensorflow la plus récente n'est pas supporter par les versions récentes de Cuda toolkit et Cudnn.

```
>>> import cv2
>>> print(cv2.__version__)
4.1.0
>>> import tensorflow as tf
>>> print(tf.__version__)
1.13.1
>>> import pandas as pd
>>> print(pd.__version__)
0.24.2
>>> import matplotlib as plt
>>> print(plt.__version__)
3.1.0
```

Figure 3-9 : Les version des bibliothèques utilisé

3.5 Préparation de la base d'apprentissage

Tensorflow a besoin de centaines d'images d'un objet pour former un bon classifieur de détection. Pour former un classificateur robuste, les images d'apprentissage doivent comporter des objets aléatoires dans l'image ainsi que les objets souhaités, ainsi que divers arrière-plans et conditions d'éclairage.

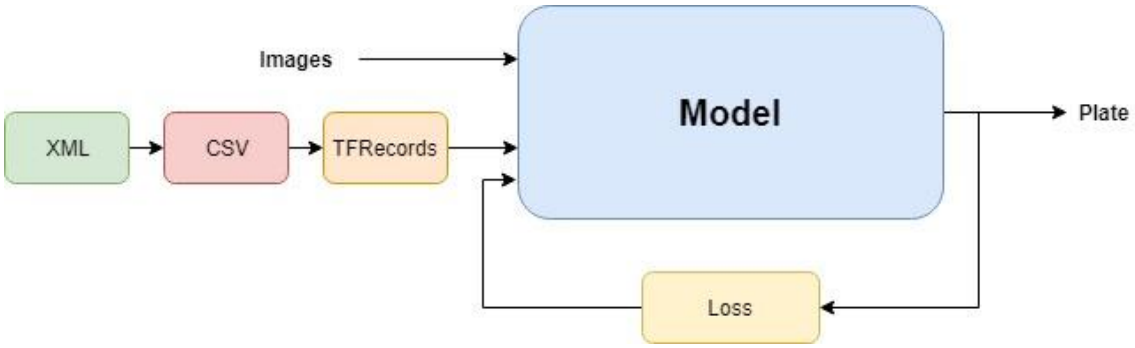

Figure 3-10 : schéma synoptique de I/O

Il doit y avoir des images où l'objet souhaité est partiellement masqué, chevauchant quelque chose d'autre ou seulement à mi-chemin dans l'image. On voit dans la figure 3-11 les étapes de la préparation des données.

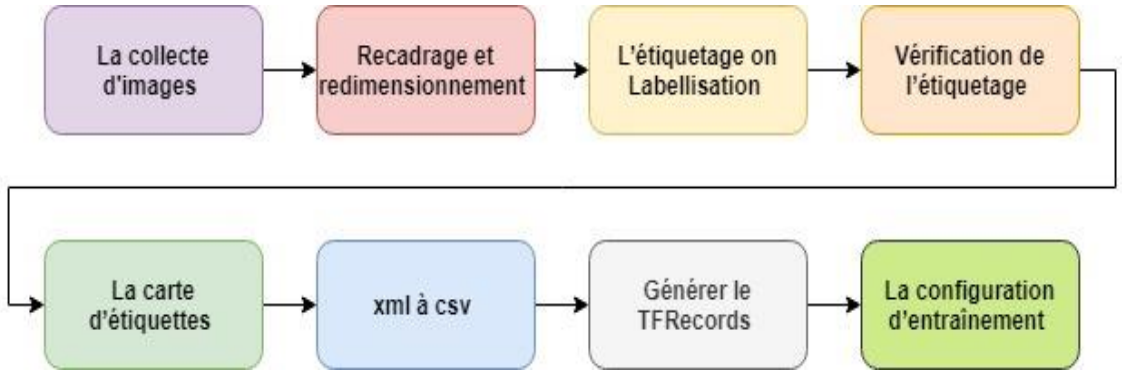

Figure 3-11 : Les étapes de la préparation des données

3.5.1 La collecte d'images

On a recueilli des centaines des photos des véhicules pour la base de données. Certains d'entre eux ont été capturés et les autres ont été téléchargés à partir d'Internet, les images ne doivent pas être trop grandes pour optimiser le temps d'apprentissage. La taille doit être inférieure à 200 Ko et la résolution ne doit pas dépasser 720x1280. Plus les images sont grandes, plus l'entraînement du classificateur sera long. La base de données est divisée en deux, 80% pour l'entraînement et 20% pour le test.

Figure3-12 : Exemples d'images capturées

3.5.2 Recadrage et redimensionnement

Après la collecte des images, nous devons les redimensionner à petite dimension, nous avons utilisé ce script présenté (figure 3-13) pour le redimensionnement.

Nous faisions le recadrage manuellement.

```
from PIL import Image
import os, sys

path = "D:/license_plates/"
dirs = os.listdir( path )

def resize():

    for item in dirs:

        if os.path.isfile(path+item):

            im = Image.open(path+item)
            f, e = os.path.splitext(path+item)
            imResize = im.resize((956,538), Image.ANTIALIAS)
            imResize.save(f + 'resized.jpg', 'JPEG', quality=90)
resize()
```

Figure 3-13 : Le programme du redimensionnement

3.5.3 L'étiquetage on Labellisation

Après avoir redimensionner les images, nous devons les étiqueter et définir Xmin Xmax Ymin Ymax, Nous avons utilisé le logiciel LabelImg, nous dessinons un cadre autour de chaque plaque d'immatriculation dans chaque image et sauvegarder sous fichier xml, ces fichiers .xml après la conversion en csv seront utilisés pour générer des TFRecords ,

qui sont l'une des entrées du ''TensorFlow trainer''. Une fois que vous avez étiqueté et enregistré chaque image, il y aura un fichier .xml pour chaque image.

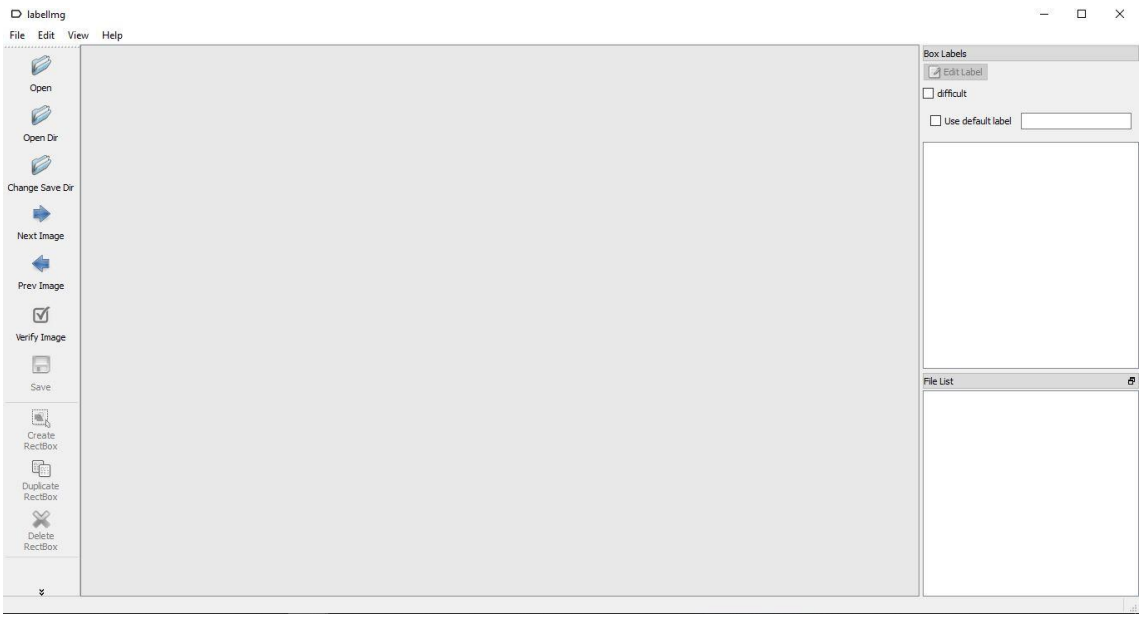

Figure 3-14 : L'interface du logiciel Labelimg

```
<annotation>
    <folder>license_plates_v2_resized</folder>
    <filename>()resized.jpg</filename>
    <path>D:\license_plates_v2_resized\()resized.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>956</width>
        <height>538</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>license_plate</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>394</xmin>
            <ymin>332</ymin>
            <xmax>584</xmax>
            <ymax>366</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 3-15 : Exemple de fichier XML obtenu

a Vérification de l'étiquetage

Après avoir étiqueté toutes les images, c'est le temps de vérifier la taille des plaques d'immatriculation à l'aide des fichiers xml que nous avions créé.

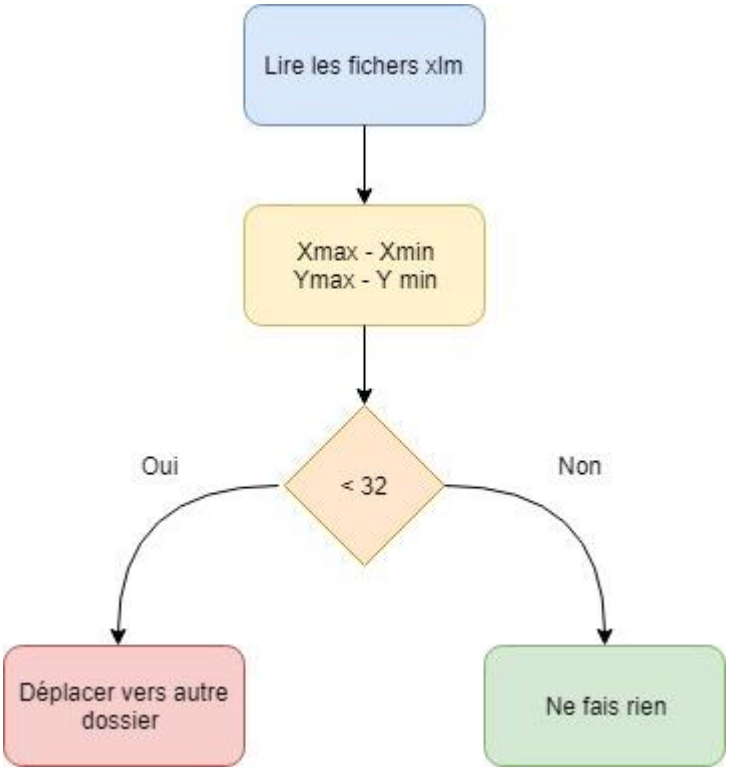

Figure 3-16 : Diagramme de la vérification d'étiquetage

b La carte d'étiquettes

Après nous avons créé une carte d'étiquettes sous format. pbtxt.

c Convertir xml en csv

La conversion des fichiers xml doit être à l'aide de la bibliothèque panda et de la bibliothèque lxml.

La figure 3-17 présente le programme utiliser pour la conversion.

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET


def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df


def main():
    for folder in ['train','test']:
        image_path = os.path.join(os.getcwd(), ('images/' + folder))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv(('images/' + folder + '_labels.csv'), index=None)
        print('Successfully converted xml to csv.')


main()
```

Figure 3-17 : Le programme utiliser pour la conversion xml en csv

Après la conversion nous obtenons le dataframe présente dans le tableau suivant.

	filename	width	height	class	xmin	ymin	xmax	ymax
0	()resized.jpg	956	538	license_plate	394	332	584	366
1	(1)resized.jpg	956	538	license_plate	151	185	835	333
2	(10)resized.jpg	956	538	license_plate	306	223	663	320
3	(11)resized.jpg	956	538	license_plate	476	423	722	475
4	(13)resized.jpg	956	538	license_plate	448	298	724	358
5	(14)resized.jpg	956	538	license_plate	348	252	496	288
6	(15)resized.jpg	956	538	license_plate	571	248	783	303
7	(16)resized.jpg	956	538	license_plate	42	154	914	328
8	(19)resized.jpg	956	538	license_plate	516	293	657	343
9	(2)resized.jpg	956	538	license_plate	136	386	395	447

Tableau 3-1 : le dataframe obtenu par la conversion

d Générer le TFRecords

Avec les images étiquetées, il est temps de générer les enregistrements TFR qui servent de données d'entrée au modèle d'entraînement TensorFlow.

3.5.4 La configuration d'entraînement

La configuration d'entraînement doit être dans un fichier sous format .config.

```
train_input_reader: {
  tf_record_input_reader {
    input_path: "../train.record"
  }
  label_map_path: "../label_map.pbtxt"
}

eval_config: {
  num_examples: 8000
  use_moving_averages: true
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "../testrecord"
  }
  label_map_path: "../label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

Figure 3-18 : Exemple des paramètres de l'entraînement

3.6 Les modèles utilisés pour l'entraînement

On a utilisé deux différents modèles pour réaliser le ALPD, 1er modèle était Faster R-CNN Infection v2 et le 2e était SSD MobileNet v1.

Nous avons utilisé une station de travaille avec -GPU- une carte graphique nvidia geforce titan black de 6GB et RAM de 32GB et processeur Intel I7 4ème génération octa cœur et 3,6 GHz vitesse d'horloge, et un PC avec CPU I5 6ème génération quad cœurs et 2,4 GHz vitesse d'horloge et 4 GB de RAM.

3.6.1 Faster R-CNN Inception v2

a **L'architecture simplifiée**

Figure 3-19 : Architecture simplifié de Faster R-CNN Inception v2

b L'architecture détaillée

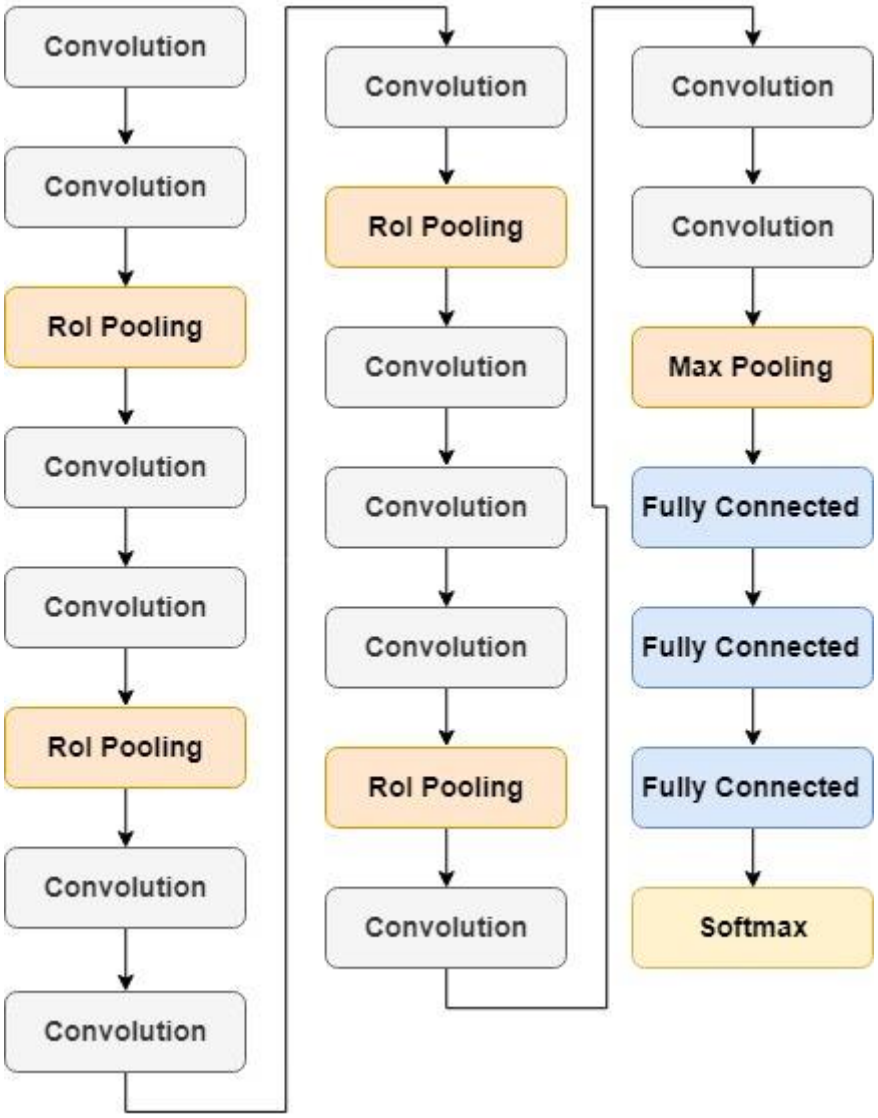

Figure 3-20 : Architecture détaillé de Faster R-CNN Inception v2

c Le temps d'entraînement

Comme on voit dans le tableau suivant la différence de temps d'entraînement entre GPU et CPU

GPU	**6 heures**
CPU	**3 jours**

Tableau 3-2 : Comparaison de temps d'entraînement de Faster R-CNN Inception v2

3.6.2 SSD-Mobilenet v1

a L'architecture simplifié

Figure 3-21 : Architecture simplifié de SSD-Mobilenet v1

b L'architecture détaillée

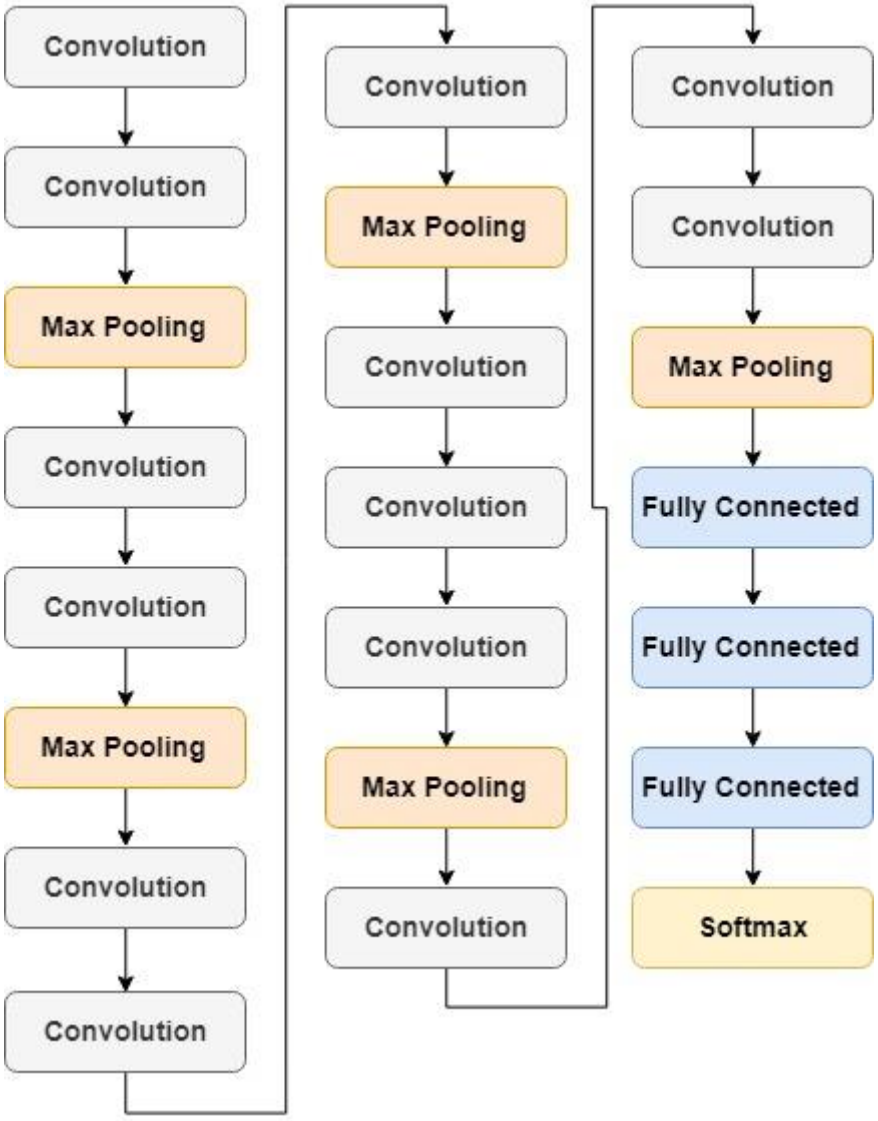

Figure 3-22 : ***Architecture détaillé de SSD-Mobilenet v1***

c Le temps d'entraînement

Comme on voit dans le tableau suivant le temps d'entraînement avec GPU, nous n'avons pas essayé avec CPU parce que le temps estimé est 13 jours.

GPU	**26 heures**
CPU	**Le temps estimé est 13 jours. Please don't try xD**

Tableau 3-3 : Comparaison de temps d'entraînement de SSD-Mobilenet v1

3.7 Evaluation des modèles

Il y a plusieurs critères pour évaluer un modèle de détection, la figure 3-23 présente la vitesse de traitement par image, mean average precision, frame per second.

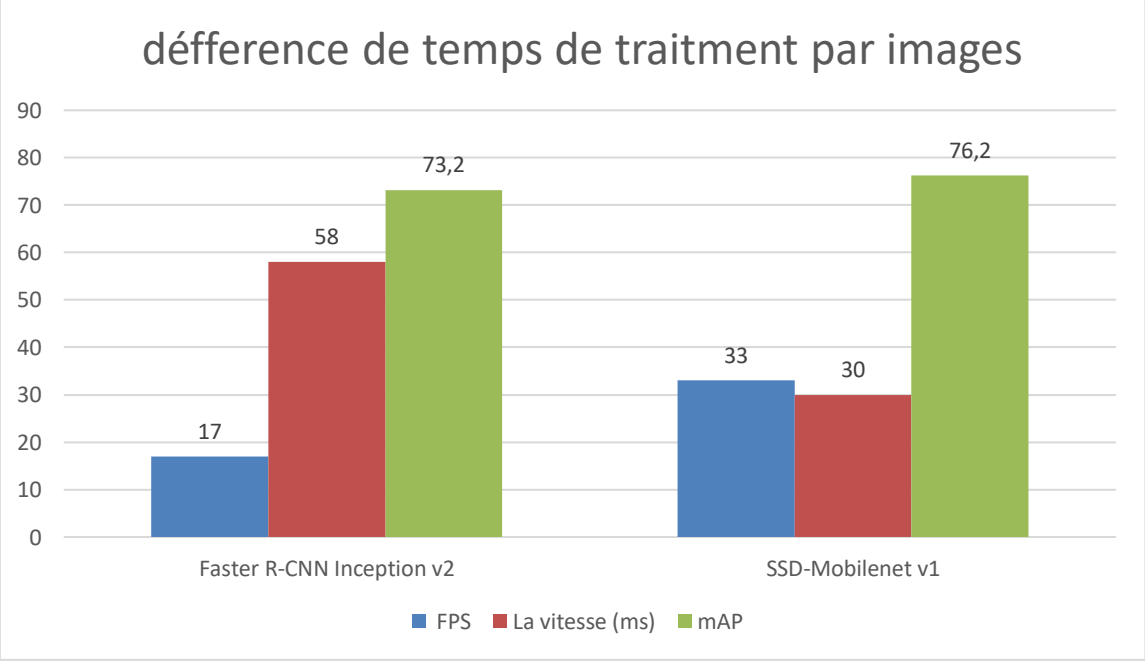

Figure 3-23 : Comparaison de la vitesse, mAP (mean average precision), FPS (frame per second) des deux modèles

3.8 L'entraînement

Après avoir configuré tous les paramètres, il est temps d'entraîner le modèle.

La figure 3-24 présente l'avancement d'entraînement, la perte et le temps de chaque étape.

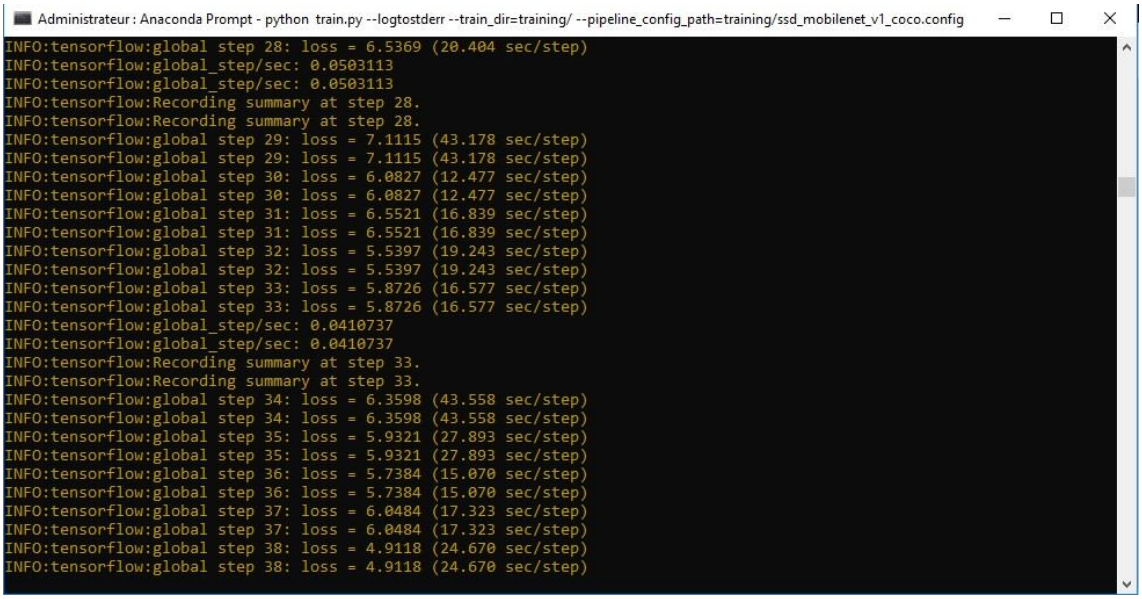

Figure 3-24 : L'avancement d'entraînement

3.9 Les résultats

Avec ces deux figures 3-25 et 3-26 on voit que la perte du modèle Faster R-CNN Inception v2 est trop faible, nous interrompons l'entraînement après une perte de 0,05 et cela s'est produit après 60000 pas.

Par contre avec le modèle SSD-MobileNet v1 nous obtenons après 60000 pas une perte de 1,6 à peu près et ceci est un peu trop.

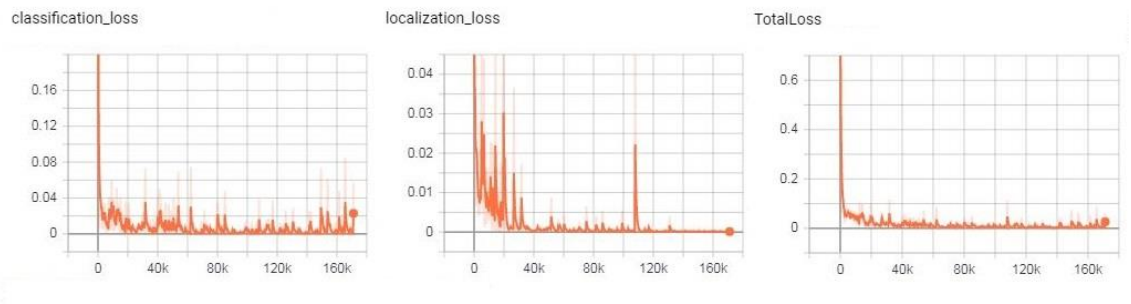

Figure 3-25 : La perte d'entraînement de modèle Faste R-CNN Inception v2

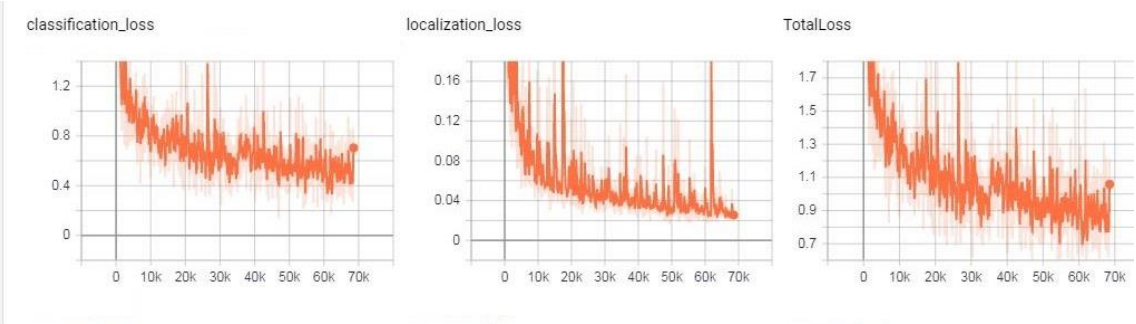

Figure 3-26 : La perte d'entraînement de modèle SSD-Mobilenet v1

On a obtenu haute précision par le modèle Faster-R-CNN-Inception-v2, et grand temps d'exécution (58 ms par image), on ne peut pas utiliser ce modèle en temps réel dans les systèmes embarqués.

Le modèle SSD-MobileNet-v1 donne une faible précision et petit temps d'exécution (30 ms par image), donc on peut l'utilise dans les systèmes embarqués.

On peut le faire par exemple avec Raspberry Pi.

3.10 Le test

Pour le test nous pouvons utiliser différents outils d'acquisition d'images, par exemple la webcam, les photos, les vidéos.

```
import cv2
import numpy as np
import tensorflow as tf
```

Figure 3-27 : importation des bibliothèques utilisé

Ce programme permet d'importer les différentes bibliothèques nécessaires.

```
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')

PATH_TO_LABELS = os.path.join(CWD_PATH,'training','labelmap.pbtxt')
```

Figure 3-28 : Emplacement de model

Ce fichier contient les poids et biais de l'apprentissage et la carte d'étiquetage que nous allons utiliser dans la détection

Pour importer une image nous utilisons la fonction suivante :

```
image = cv2.imread('test.jpg')
```

Figure 3-29 : Lire une image

Pour importer une vidéo nous utilisons la fonction suivante :

```
video = cv2.VideoCapture('test.mov')
```

Figure 3-30 : Lire une vidéo

Pour lire une vidéo à partir d'une webcam

```
video = cv2.VideoCapture(0)
ret = video.set(3,1280)
ret = video.set(4,720)
```

Figure 3-31 : Lire une webcam

Pour afficher le rectangle contenant le matricule, la classe et le score (le % de la précision)

```
vis_util.visualize_boxes_and_labels_on_image_array(
    image,
    np.squeeze(boxes),
    np.squeeze(classes).astype(np.int32),
    np.squeeze(scores),
    category_index,
    use_normalized_coordinates=True,
    line_thickness=8,
    min_score_thresh=0.60)
```

Figure 3-32 : Les fonctions utilisé pour afficher la classe, le score et le rectangle

Le programme suivant affiche les résultats ou nous souhaitons que ce soit dans une vidéo, une image et une webcam :

```
while(True):
    ret, frame = video.read()

    cv2.imshow('testing cam', frame)

    if cv2.waitKey(1) == ord('q'):
        break


video.release()
cv2.destroyAllWindows()
```

Figure 3-33 : Affichage d'une vidéo

```
cv2.imshow('testing image', image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 3-34 : Affichage d'une image

```
while(video.isOpened()):

    ret, frame = video.read()

    cv2.imshow('testing video', frame)

    if cv2.waitKey(1) == ord('q'):
        break

video.release()
cv2.destroyAllWindows()
```

Figure 3-35 : Affichage d'une webcam

Cette image présente la détection des plaques d'immatriculation avec Faster R-CNN Inception v2.

Figure 3-37 : Une image avec des plaques d'immatriculation détecté avec Faster R-CNN Inception v2.

Cette image présente la détection des plaques d'immatriculation avec SSD MobileNet v1.

Figure 3-36 : Une image avec des plaques d'immatriculation détecté avec SSD MobileNet

3.11 Conclusion

Nous avons réussi à implémenter les deux modèles de détection, Faster R-CNN inception v2 et SSD-MobileNet v1 en utilisant Tensorflow sur python.

Nous avons pu les comparer sous trois aspects :

- Le temps d'apprentissage sur GPU et CPU.
- Le temps de traitement d'une image après l'apprentissage.
- "Mean average precision" la moyenne de precision moyenne.

Qui nous permet de conclue que les deux modèles

Conclusion générale

Pour conclure, avec l'avancement de l'intelligence artificielle, plus particulièrement le Deep Learning, la détection et l'identification des personnes, matricules et maladies et bien d'autres, dans une multitude de domaines, sont devenues possibles et efficaces.

Nous avons essayé, dans cette étude, de présenter un système de détection des plaques d'immatriculation algérienne. Nous avons présenté un certain nombre de notions et de définitions concernant le domaine de traitement d'image que nous avons utilisé pour réaliser notre travail. Nous avons utilisé des fonctions de DL existantes (dans des bibliothèques que nous avons choisies, après plusieurs tentatives, pour assurer la compatibilité des versions).

Nous avons créé des bases d'apprentissage, que nous avons étiquetées, puis nous avons implémenté les algorithmes de DL Faster R-CNN, et SSD, et obtenu des résultats satisfaisants, mais plus précis pour le premier (plus grand pourcentage de réussite de détection) et plus rapides pour le second.

L'exécution de nos programmes sur CPU, et GPU, nous ont prouvés qu'il était préférable d'utiliser les GPU, bien plus puissants que les CPU, et mieux adaptés à la grande quantité de calculs requis par le DL et les énormes bases de données qu'il requiert.

Mais une fois l'étape d'apprentissage réalisée, le réseau de neurones obtenu peut être utilisé sur un CPU, implémenté sur Raspberry PI par exemple, ou implémenté sur circuit FPGA pour gagner en puissance.

L'intérêt de notre projet est d'implémenter cet algorithme dans un système embarqué avec les caméras de surveillance pour faciliter l'identification et la détections des voitures, le système pourrait être très efficace pour la police dans l'identification pendant un accident ou un vol par exemple. Comme perspectives, pour l'identification des plaques, notre travail (la détection) peut être considéré comme un préapprentissage pour un algorithme de DL pour la reconnaissance des caractères.

Annexes

Bibliographie

1. Z. Saidane and C. Garcia., “Automatic scene text recognition using a convolutional neural network.,” In Workshop on Camera-Based Document Analysis and Recognition, 2007.
2. D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, “High performance neural networks for visual object classification.,” Technical Report IDSIA-01-11, 2011.
3. http://www-lisic.univ-littoral.fr/~verel/TEACHING/08-09/sac-M1/cRdNV9.pdf
4. http://conf.laas.fr/ignotus/archives/Doncescu_reseaux_neuronnes.pdf
5. https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss?hl=fr
6. https://fr.wikipedia.org/wiki/R%C3%A9tropropagation_du_gradient
7. yacine ouassar : réseaux d'ondelettes et réseaux de neurones pour la modélisation statique et dynamique de processus, université pierre et marie curie paris , le 23 avril 2004
8. https://openclassrooms.com
9. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. IEEE, Nov. 1998.
10. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in Neural Information Processing Systems 25 (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.
11. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in Computer Vision and Pattern Recognition (CVPR), 2015.
12. K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” CoRR, vol. abs/1409.1556, 2014.
13. C. Szegedy, S. Ioffe, and V. Vanhoucke, “Inception-v4, inception-resnet and the impact of residual connections on learning,” CoRR, vol. abs/1602.07261, 2016.
14. K.He,X.Zhang,S.Ren,andJ.Sun,“Deepresiduallearningforimagerecognition,” CoRR, vol. abs/1512.03385, 2015.
15. A. Veit, M. J. Wilber, and S. J. Belongie, “Residual networks are exponential ensembles of relatively shallow networks,” CoRR, vol. abs/1605.06431, 2016.
16. Q. Liao and T. A. Poggio, “Bridging the gaps between residual learning, recurrent neural networks and visual cortex,” CoRR, vol. abs/1604.03640, 2016.

17. M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes challenge: A retrospective,” International Journal of Computer Vision, vol. 111, pp. 98–136, Jan. 2015.
18. T. B. Dalal, N., “Histograms of oriented gradients for human detection,” pp. 886– 893, 2005.
19. P.F.Felzenszwalb, R.B.Girshick, D.McAllester, andD.Ramanan, “Objectdetection with discriminatively trained part-based models,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 32, pp. 1627–1645, Sept. 2010.
20. R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in Computer Vision and Pattern Recognition, 2014.
21. J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, “Selectivesearchforobjectrecognition,”International Journal of Computer Vision, vol. 104, no. 2, pp. 154–171, 2013.
22. I. Endres and D. Hoiem, “Category independent object proposals,” in Proceedings of the 11th European Conference on Computer Vision: Part V, ECCV’10, (Berlin, Heidelberg), pp. 575–588, Springer-Verlag, 2010.
23. D. Hoiem, A. A. Efros, and M. Hebert, “Recovering occlusion boundaries from an image,” Int. J. Comput. Vision, vol. 91, pp. 328–346, Feb. 2011.
24. K.He,X.Zhang,S.Ren,andJ.Sun,“Deepresiduallearningforimagerecognition,” CoRR, vol. abs/1512.03385, 2015.
25. J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, “Selectivesearchforobjectrecognition,”International Journal of Computer Vision, vol. 104, no. 2, pp. 154–171, 2013.
26. R. Girshick, “Fast R-CNN,” in International Conference on Computer Vision (ICCV), 2015
27. R. Yang, H. Yin, and X. Chen, “License plate detection based on sparse autoencoder,” Proc. IEEE, 2015.
28. L. Xian, H. Bie, J. Sun, and Y. Hou, “License plate recognition algorithm for passenger cars in chinese residential areas.,” Sensors (Basel), p. 8355–8370, 2012.
29. S. Du, M. Ibrahim, M. Shehata, and W. Badawy, “Automatic license plate recognition (alpr): A state-of-the-art review,” IEEE Trans. Cir. and Sys. for Video Technol., vol. 23, pp. 311–325, Feb. 2013.