CDTA

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique
Département Informatique

Faculté d'Eléctronique et d'Informatique
Bibliothèque Informatique

Mémoire de Master

Filière: Informatique

Spécialité Systèmes informatiques intelligents

Thème

Détection visuelle d'objets statiques et dynamiques dans un environnement de type route et classification en exploitant l'apprentissage profond.

Organisme d'accueil :
Centre de développement des technologies avancées (CDTA)

Sujet Proposé par :
Encadreur : Mme K.AIT SADI (CDTA)
Co-encadreur : Mme N.BAHA

Soutenu le : 12/07/2019

Présenté par :
CHAIBEDDRA Insaf
MADENE Selma

Devant le jury composé de:

Mr S.LARABI	Président
Mme A.DAHMANE	Membre

Binôme N° : 088/ 2019

Remerciements

« Avant toutes choses, nous remercions ***DIEU*** *Grand et Puissant de nous avoir donné la volonté et le courage d'accomplir notre cursus universitaire et de mener ce travail jusqu'à la fin et cela en dépit des difficultés.*

Nous tenons à remercier notre directrice de mémoire ***Mme AIT SADI Karima*** *pour son aide ainsi que pour ses conseils précieux tout au long de cette expérience professionnelle et d'avoir relu et corrigé notre mémoire. Ses conseils de rédaction ont été très précieux.*

Veuillez accepter nos profonds respects.

Nous remercions également notre co-promotrice ***Mme BAHA Nadia*** *pour son suivi tout au long de ce travail, pour ces encouragements et sa bonté.*

Nos remerciements vont également aux membres du jury, vous nous faites honneur en jugeant notre humble travail. Veuillez accepter notre profonde reconnaissance.

Enfin, nous tenons à exprimer nos vifs remerciements à tous ceux qui ont contribué de près ou de loin à l'élaboration de ce travail. »

Dédicaces

*Je dédie ce travail, A mes très chers parents **Nadira** et **Youcef** qui ont toujours été là pour moi, et qui m'ont donné un magnifique modèle de labeur et de persévérance. J'espère qu'ils trouveront dans ce travail toute ma reconnaissance et tout mon amour, que Dieu vous accorde santé, longue vie et bonheur.*

*Je dédie ce travail aussi à mes très chers frères **Walid**, **Nazim** et **Racim** dont leurs présences et leurs sourires me comble de bonheur.*

*Je dédie ce travail, à mes grands-parents, à mes tantes et oncles, à ma cousine **Soumia**, ainsi qu'à toute ma famille.*

Je remercie Dieu de m'avoir entouré de personnes aussi pures, sages et sincères.

*Et pour finir. Je dédie ce travail, à mon binôme et mon amie **Selma**, je la remercie pour son soutien, sa patience et son amitié. À mes amies **Yasmine**, **Razika**, **Siham**.*

CHAIBEDDRA Insaf

*A mes très chers parents **Badia** et **Mohammed**, honorables et aimables, les symboles de la bonté par excellence, source de tendresse et exemple de dévouement, vous n'avez pas cessé de m'encourager et de prier pour moi.*

*Je dédie ce travail aussi à mes très chers frères **Nacereddine** et **Abdesslam** pour leurs encouragements permanents, et leur soutien moral.*

*Je dédie ce travail, à mes grands-parents, à mes tantes et oncles, à mes cousines **Nachida**, **Imene** et **Asma**, ainsi qu'à toute ma famille.*

*Et pour finir je dédie ce travail, A mon binôme et mon amie **Insaf**, je la remercie pour son soutien, sa patience et sa compréhension. A mes amies **Yasmine**, **Sylia** et **Fatim**. Et à tous ceux qui représentent pour moi mon amour, mon pilier, ma force, et ma foi.*

MADENE Selma

Sommaire:

Introduction générale **6**

Chapitre 1 : Concepts de base **8**

1.1 Introduction 8

1.2 Robotique 8

1.2.1 Un robot 8

1.2.2 L'autonomie 9

1.2.3 Composition d'un robot 9

1.2.4 Les types des robots 10

1.2.5 Les véhicules autonomes 10

1.2.6 La navigation 10

1.3 L'environnement visuel du robot 12

1.3.1 Les capteurs actifs 12

1.3.2 Les capteurs passifs 13

1.4 Quelques notions de traitement d'image 14

1.4.1 L'image 14

1.4.2 Le pixel 14

1.4.3 Les types de l'image 14

1.4.4 Le filtrage 15

1.4.5 La convolution 15

1.5 Détection d'objets 16

1.6 Conclusion 20

Chapitre 2 : Les réseaux de neurones **21**

2.1 Introduction 21

2.2 L'apprentissage automatique 22

2.3 Les réseaux de neurones 24

2.3.1 Neurone biologique 24

2.3.2 Les réseaux de neurones artificiels 24

2.3.2.1 Algorithme d'apprentissage d'un perceptron par correction d'erreur 25

2.3.2.2 Algorithme d'apprentissage d'un perceptron par la descente du gradient 26

2.3.3 Réseau de neurones multicouches : 27

2.3.3.1 La propagation 28
2.3.3.2 La rétropropagation 28
2.3.3.3 Fonction d'activation 28
2.4 L'apprentissage profond 30
2.4.1 C'est quoi l'apprentissage profond 30
2.4.2 Les différents modèles de réseau profond 31
2.4.2.1 Les réseaux neuronaux récurrents (Reccurent Neural Networks) 31
2.4.2.2 Deep Belief networks 31
2.4.2.3 Réseaux de neurones convolutifs (Convolutional neural networks) 31
2.4.2.4 L'architecture des CNN : 32
2.4.2.5 Les architectures des CNN les plus connus 37
2.5 Conclusion 43
Chapitre 3 : Conception de l'approche utilisée 44
3.1 Introduction 44
3.2 Fonctionnement de YOLO (You Only Look Once) 44
3.2.1 Principe de l'intersection sur l'union (IoU) 46
3.2.2 La fonction de perte (loss function) 47
3.2.3 Hyperparamètres 49
3.3 Les variantes de YOLO 49
3.3.1 YOLO version 1 (YOLO V1) 49
3.3.2 YOLO version 2 (YOLO V2) 51
3.3.3 YOLO version 3 (YOLO V3) 53
3.4 L'application de détection d'objets développée basée sur YOLO V3 56
3.4.1 Données pour l'entrainement du modèle 57
3.4.1.1.1 Ensembles de données utilisés 57
3.4.1.1.2 Préparation de données 58
3.4.2 Paramètres d'apprentissage 59
3.4.2.1 Les hyperparamètres taille du lot et subdivision 60
3.5 Conclusion 63
Chapitre 4 : Implémentation et résultats 64
4.1 Introduction : 64
4.2 Environnements de travail : 64
4.2.1 Environnement matériel: 64
4.2.2 Environnement logiciel 64

4.2.3 La plateforme d'apprentissage profond : 65
4.2.4 La plateforme d'apprentissage Google Colaboratory : 66
4.3 Interface de l'application dévelopée 66
4.4 Evaluation des tests et analyse des résultats 67
4.4.1 Evaluation des performances de la détection 68
4.4.2 Tests du modèle sur quelques images : 71
4.5 Conclusion : 76
Conclusion générale et perspectives 77

Table des Figures:

Figure 1-1 : Schéma des interactions d'un robot avec son environnement 12
Figure 1-2 : Exemple de convolution. 16
Figure 1-3 : Schéma bloc d'un système de détection d'objet 17
Figure 1-4 : Exemple d'une classification 17
Figure 1-5 : Exemple d'une localisation. 18
Figure 1-6 : Exemple d'une détection d'objet multicible 18
Figure 2-1 : Représentation d'un réseau de neurone biologique 24
Figure 2-2 : Représentation mathématique d'un perceptron. 25
Figure 2-3 : Architecture d'un réseau de neurone avec deux couches cachées. 27
Figure 2-4 : Graphe de la fonction sigmoïde 29
Figure 2-5 : Graphe d'une fonction linéaire. 29
Figure 2-6 : Graphe de la fonction Relu 30
Figure 2-7 : Architecture d'un réseau de neurone convolutif basique. 33
Figure 2-8 : Principe de l'opération de padding. 34
Figure 2-9 : Principe de la convolution. 34
Figure 2-10 : Principe du sous échantillonnage. 35
Figure 2-11 : Couches entièrement connectées. 36
Figure 2-12 : Architecture du réseau LeNet 38
Figure 2-13 : Architecture du réseau AlexNet. 39
Figure 2-14 : Architecture du réseau VGG16. 40
Figure 2-15 : Architecture du réseau Google net 42
Figure 2-16 : Architecture d'un module 'Inception'. 42
Figure 3-1 : Un exemple de détection avec YOLO 45
Figure 3-2 : Illustration IoU 47
Figure 3-3 : Illustration par des exemples de IoU. 47
Figure 3-4 : Illustration de l'architecture de YOLO V3 53
Figure 3-5 : Schéma blocs de l'application de détection d'objets développée 56
Figure 4-1 : Page d'accueil de l'application développée. 66
Figure 4-2 : Page de lancement de l'application de détection d'objets et classification. 67
Figure 4-3 : Exemple d'images : (a) image de KITTI , (b) Image de OpenImage. 68
Figure 4-4 : Variation de l'erreur moyenne en fonction du nombre d'itérations en utilisant KITTI. 69
Figure 4-5 : Variation de l'erreur moyenne en fonction du nombre d'itérations en utilisant OpenImagesv4. 70
Figure 4-6 : Résultats de test du modèle sur les images (objet voiture). 72
Figure 4-7 : Résultats de test du modèle sur les images (objet camion). 73
Figure 4-8 : Résultats de test du modèle sur les images (objet panneau de signalisation et voiture). 74
Figure 4-9 : Résultats de test du modèle sur les images (objet bicyclette). 75

Liste des tableaux:

Tableau 2-1 : Architecture VGG16 41
Tableau 2-2 : Architecture Inception 'Google Net'. 43
Tableau 3-1 : Architecture de YOLO V2 52
Tableau 3-2 : Architecture de YOLO V3 54
Tableau 3-3 : Paramètres de l'apprentissage pour l'ensemble de données OpenImagesv4 59
Tableau 3-4 : Paramètres de l'apprentissage pour l'ensemble de données KITTI. 60
Tableau 4-1 : L'erreur moyenne obtenue par l'apprentissage pour les ensembles de données KITTI, OpenImagesv4 et COCO. 71

Introduction générale

Qui aurait cru il y a un siècle que les voitures rouleraient un jour toutes seules ?

Le véhicule autonome est un véhicule remplissant la fonction de transporter d'un point à un autre des biens et/ou des passagers sans intervention humaine. Il est muni d'un système de pilotage automatique lui permettant de se déplacer dans les conditions réelles de circulation. Actuellement en développement, il sera l'aboutissement, après de nombreuses années de recherche, du véhicule intelligent, qui remplace ou seconde de plus en plus le conducteur humain.

John McCarthy, informaticien, a inventé le terme «intelligence artificielle (IA)» en 1955. L'IA est définie comme la capacité d'un programme ou d'une machine informatique à penser, à apprendre et à prendre des décisions. En général, le terme signifie une machine qui imite la cognition humaine . Avec l'IA, des programmes informatiques et des machines sont obtenus pour faire ce que font les humains. Ces programmes et ces machines sont alimentés avec une quantité énorme de données qui sont analysées et traitées pour finalement penser logiquement et effectuer des actions humaines. Le processus d'automatisation des tâches humaines répétitives n'est que la partie émergée de l'iceberg de l'IA, des équipements des voitures autonomes ont mis en œuvre l'IA dans le but de sauver des vies humaines. Pour passer son permis, l'IA devra progresser d'abord sur la perception, *"le premier enjeu, car sans la perception, rien n'est possible"*, relève Vanessa Picron. Mieux détecter et identifier les objets et êtres vivants s'impose. Y compris par mauvais temps.

La technologie de base de la conduite non surveillée comprend la perception de l'environnement, un positionnement précis et la planification de trajectoire. Les environnements routiers sont des milieux ouverts, peuplés d'une grande variété d'objets et d'acteurs souvent imprévisibles, régis par de nombreuses règles, explicites et implicites, et une signalisation complexe, ainsi, les véhicules autonomes peuvent être confrontés à de nombreuses situations différentes sur la route. Si les conducteurs doivent confier leur vie à des voitures autonomes, ils doivent être sûrs que ces voitures seront prêtes à affronter les situations les plus folles.

Les algorithmes d'apprentissage automatique, et plus particulièrement sa version apprentissage profond «Deep Learning », aide à la détection d'obstacles et d'objets. Ainsi, après avoir visionné un très grand nombre d'images, la voiture pourra déterminer la nature des objets qui l'entourent et donc adopter un comportement et des règles de conduite appropriés. Pour fonctionner correctement, la voiture autonome doit parcourir une quantité colossale de kilomètres pour affiner sa compréhension de l'environnement qui l'entoure (notamment les changements de conditions météorologiques) grâce à un grand nombre d'images qu'elle aura capturée.

Ce projet se situe dans le contexte d'une des tâches principales de tout algorithme d'apprentissage automatique dans une voiture autonome qui consiste à restituer en continu l'environnement environnant et à prédire les modifications éventuelles de cet environnement. Ces tâches sont principalement divisées en quatre sous-tâches: Détection

d'objet, c'est-à-dire la détection d'un ensemble de régions d'intérêt dans la scène routière, la reconnaissance d'objet (classification et la localisation d'objet.

Problématique et contribution : Tous les systèmes de détection antérieurs utilisent des classificateurs ou des localisateurs pour effectuer la détection. Ils appliquent le modèle à une image située à plusieurs endroits et à différentes échelles. Les régions de l'image présentant des scores élevés sont considérées comme des détections. Notre but est d'utiliser une approche totalement différente qui consiste en un algorithme d'apprentissage automatique supervisé, nommé YOLO (You Only Look Once). Ce dernier permet de classifier des objets tels que des voitures, des personnes, camions, bicyclettes et panneaux de signalisation en appliquant un seul réseau de neurones à l'image complète. Ce réseau analyse l'image dans son ensemble et la divise en segments et prédit des boîtes englobantes et des probabilités pour chaque segment. Ces boîtes englobantes sont pondérées par les probabilités prédites. L'algorithme YOLO est un excellent outil pour la détection d'objets dans les véhicules autonomes. Il garantit un traitement rapide et une réponse du véhicule aux situations réelles.

Ce document est composé de quatre chapitres. Dans le premier chapitre, nous présenterons quelques notions de base de la robotique, de la vision par ordinateur, et de la détection d'objet. Dans le second chapitre, nous aborderons l'apprentissage automatique et quelques méthodes existantes pour la détection d'objets. Dans le chapitre suivant à savoir le chapitre trois, nous détaillons l'approche YOLO et ses variantes, des approches utilisées pour la détection intelligente des objets routiers. Nous présenterons ensuite dans le quatrième chapitre les résultats obtenus de la phase d'apprentissage développée et les tests et les évaluations effectués. Enfin, nous conclurons ce mémoire par des perspectives de l'application développée en termes d'amélioration du taux de détection et augmentation du nombre de classes pour qu'elle soit par la suite intégré dans le système de navigation du robot mobile de type voiture du CD

Chapitre 1 : Concepts de base

1.1 Introduction

L'intelligence artificielle (IA) s'adapte à tous les domaines, mais si on parle du principe que l'IA tend à remplacer le cerveau humain ou au moins à le seconder, on parle alors de la robotique autonome [1].

De tout temps l'homme a cherché à reproduire son semblable comme s'il avait un double, concevoir un robot ressemblant à un homme ne peut s'imaginer sans la réalisation d'un système autonome.

La robotique actuellement vise à créer des machines capables d'aider l'être humain ou de le remplacer dans des tâches complexes, dangereuses ou répétitives. Ces machines sont dotées de capteurs qui leur donnent des informations sur les caractéristiques de l'environnement où elles fonctionnent. Elles sont équipées aussi de parties mobiles qui leur permettent d'interagir avec leur environnement. Mis à part le côté matériel, les machines qui peuvent remplacer l'homme disposent d'un degré d'intelligence (IA) représenté par l'implémentation d'une ou plusieurs techniques issues de cette science.

Dans ce chapitre nous allons introduire les notions de base de la robotique, ensuite nous présenterons l'architecture d'un robot mobile autonome en général, suivi par sa perception de l'environnement.

Nous terminons le chapitre par les notions liées à la détection d'objets et les différentes approches développées dans ce sens.

1.2 Robotique

C'est une discipline qui fait partie de la mécanique, de l'électrique, et de l'informatique en même temps. Elle s'intéresse à la conception, à la construction et au fonctionnement des robots, ainsi qu'à la création des systèmes informatiques qui les contrôlent. [2]

1.2.1 Un robot

Un robot mobile est un agent physique, réalisant des tâches dans son environnement doté d'un certain niveau de capacité. L'objectif de la robotique est de permettre au robot d'interagir avec son environnement automatiquement (sans aucune intervention humaine).

1.2.2 L'autonomie

Un système robotique est autonome si [3] :

- Il est capable d'accomplir sans intervention humaine les objectifs pour lesquels il a été conçu,
- Il est capable de choisir ses actions afin d'accomplir ses missions.

1.2.3 Composition d'un robot

Un robot est un assemblage complexe de pièces mécaniques et de pièces électroniques, le tout piloté par une intelligence artificielle. Lorsque les robots autonomes sont mobiles, ils possèdent également une source d'énergie embarquée: généralement une batterie d'accumulateurs électriques.

- **Structure mécanique**

C'est le squelette du robot. Le choix des articulations permettra de déterminer les mouvements possibles et d'orienter son type d'utilisation. On compte un degré de liberté pour chaque axe de rotation selon lequel un membre (bras, patte) peut se mouvoir.

- **Servo-moteurs (actionneurs)**

Le servo-moteur permettra au robot d'effectuer concrètement les actions commandées par son cerveau, en fonction de ses objectifs et suite au traitement des informations recueillies par les capteurs. Les actionneurs sont des organes qui, sous commande, transforment l'énergie qui leur est fournie en actions physiques utilisables comme des mouvements.

- **Capteurs**

Les capteurs s'occupent de la partie sensorielle du robot. Ils permettent d'acquérir des informations à propos de l'environnement du robot ou de ses composants internes.

- **Cerveau**

Le cerveau du robot permet d'analyser les données provenant des capteurs et d'envoyer les ordres relatifs aux servo-moteurs. La partie commande est matérialisée physiquement par un microcontrôleur, qui est un cerveau électronique spécialement conçu pour l'interagir avec des capteurs et des actionneurs

1.2.4 Les types des robots [4]

Il existe plusieurs types de robots répondant à différentes fonctions, Nous allons aborder les trois catégories de robots les plus communes.

Les humanoïdes : c'est Catégorie la plus connue, en grande partie grâce à leur promotion faite par la science-fiction, elle regroupe tous les robots anthropomorphes, ceux dont la forme rappelle la morphologie humaine. Ces robots ont généralement un torse, une tête, deux bras et deux jambes

Les robots industriels : sont construits à partir de bras manipulateurs et sont utilisés dans les lignes de production.

Les robots mobiles : Cette catégorie englobe tous les robots à base mobile, ce sont les robots qui ont la possibilité de se déplacer dans leur environnement pour accomplir leurs tâches.

1.2.5 Les véhicules autonomes

Un véhicule autonome est un véhicule automobile apte à rouler, sur route ouverte, sans intervention d'un conducteur. Le concept vise à développer et produire un véhicule pouvant réellement circuler sur la voie publique dans le trafic sans intervention humaine en toutes situations, à terme. C'est une application typique du domaine de la robotique mobile dans laquelle de nombreux acteurs sont engagés.

1.2.6 La navigation

Le but de la robotique mobile actuelle est de développer un système de navigation intelligent, où l'autonomie de navigation d'un robot mobile est la capacité des robots à prendre des décisions indépendamment de l'opérateur humain dans leurs déplacements. La navigation est définie comme le procédé permettant de répondre aux questions suivantes [5] :

«Où suis-je ? » se localiser dans son environnement.

«Où dois-je aller ?» trouver des objets liés à sa mission ou des zones d'intérêts.

«Comment dois-je y aller ? » planifier ses actions, et les exécuter.

Un robot est une machine équipée de capacités de perception, de décision et d'action qui lui permettent d'agir de manière autonome dans son environnement en fonction de la perception qu'il en a.

Cette définition s'illustre par un schéma classique des interactions d'un robot avec son environnement (Figure 1.1).

Nous expliquons le modelé perception-planification-action qui est le modèle fondateur de la plupart des systèmes robotiques :

La perception : La fonction de perception d'un robot autonome, rassemble les informations sur l'environnement externe ainsi que sur son système interne, elle les fusionne afin de construire un modèle qui représente et décrit le robot, son environnement et la relation entre les deux.

La localisation est incluse dans la perception, elle consiste à déterminer la position du véhicule par rapport à son environnement, c'est un problème très complexe, il nécessite la combinaison du plusieurs capteurs [6] [7].

La planification : planification a pour rôle de prendre les décisions nécessaires pour atteindre un but, qui consiste généralement à amener le véhicule du point de départ à sa destination tout en évitant les obstacles.

Le système des véhicules autonomes applique un algorithme qui analyse le modèle créé par la fonction de perception et détermine un ensemble d'actions en vue d'atteindre son objectif [6].

L'action : Appelé aussi les compétences d'exécution des systèmes autonomes est le processus permettant d'exécuter les actions prévues afin de produire les mouvements désirés, il s'agit du logiciel qui actionne les freins, l'accélérateur et autre fonction.

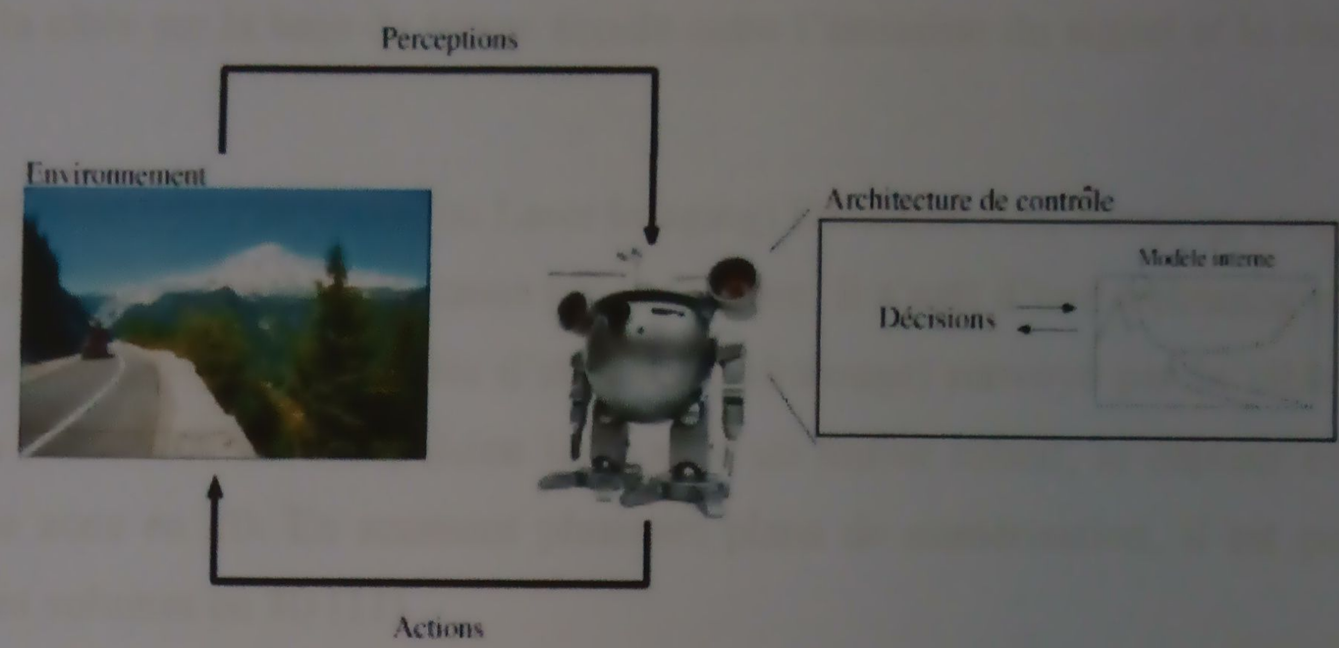


Figure 0-1 : Schéma des interactions d'un robot avec son environnement [8].

Nous nous intéressons particulièrement dans ce document, aux fonctions de perception embarquées sur un robot ou un véhicule autonome, fonctions qui doivent analyser l'environnement, pour détecter et classifier les objets qu'il contient d'une part, et d'autre part déterminer leur position.

1.3 L'environnement visuel du robot

La perception est une étape essentielle et fondamentale dans les robots autonomes, elle fournit au robot des informations cruciales, lui permettant de faire une représentation interne de l'environnement, pour pouvoir agir et réagir dans le monde qui l'entoure.

L'élément de base du système de perception est les capteurs, ils ont pour fonction d'acquérir des données provenant de l'environnement comme des systèmes mono ou multi-caméras, les télémètres laser, les radars, les sonars, entre autres modalités sensorielles. Certains sont actifs, d'autres sont passifs.

1.3.1 Les capteurs actifs

Un capteur est dit "actif" s'il produit une énergie envoyée sur la scène ; les obstacles sont détectés par l'énergie réfléchie par leur surface. Un capteur actif possède donc à la fois un émetteur et un récepteur, parmi les capteurs actifs nous citons [9] :

Télémètre ultrason : émet à intervalles réguliers de courtes impulsions sonores à haute fréquence. Ces impulsions se propagent dans l'air à la vitesse du son. Lorsqu'elles rencontrent un objet, elles sont réfléchies er reviennent sous forme d'écho. Celui-ci calcule la distance le

séparant de la cible sur la base du temps écoulé entre l'émission du signal et la réception de l'écho [10].

Lidar : est l'acronyme de Light (ou Laser Imaging) Detection And Ranging, ce qui signifie la détection et l'estimation de la distance par la lumière. Il s'agit d'une technologie temps de vol basée sur l'analyse des propriétés d'un laser (infrarouge) renvoyé par la cible vers son émetteur. En faisant tourner l'impulsion laser avec un miroir rotatif, le capteur est capable d'évaluer une zone en 2D. En scannant plusieurs plans de numérisation, il est possible de déterminer des volumes en 3D [11].

1.3.2 Les capteurs passifs

Un capteur passif ne fait que recevoir l'énergie naturellement réfléchie par les objets présents dans la scène. Plusieurs capteurs passifs sont utilisés pour la robotique nous citons [9] :

La stéréo-vision : Les systèmes de stéréovision permettent de calculer par triangulation, la profondeur des objets dans l'espace 3D, grâce à leurs projections dans les plans des deux caméras. La différence des positions dans les deux images, de la projection d'un même point de l'espace 3D, s'appelle la disparité ; la recherche des points correspondants entre ces deux images, permet donc de produire une image de disparité. En supposant que les caméras ont des axes optiques parallèles, alors la disparité est inversement proportionnelle à la profondeur. Ainsi la disparité est nulle pour un point à l'infini, par exemple un point sur la ligne d'horizon [9].

Capteur infrarouge : Les télémètres infrarouges possèdent l'avantage d'avoir un cône de détection beaucoup plus restreint. Ils utilisent une lumière infrarouge au lieu d'une onde sonore pour la détection et peuvent être basés sur différentes techniques qui permettent de recueillir plus ou moins d'information. Il est possible de mesurer simplement le retour ou le non-retour d'une impulsion codée, ce qui permet de détecter la présence ou l'absence d'un obstacle dans une certaine portion de l'espace. Il est également possible de réaliser une triangulation sur le faisceau de retour de l'onde lumineuse, ce qui permet d'avoir une mesure de la distance de l'obstacle.

Les capteurs transforment la scène réelle en une représentation mathématique qui est l'image.

1.4 Quelques notions de traitement d'image

1.4.1 L'image

Est une représentation d'une personne, ou d'un objet par la peinture, le dessin, la photographie, etc. Elle est définie comme étant une fonction f (x, y) à deux dimensions, où x et y sont les coordonnées spatiales, et l'amplitude à tous points (x, y) correspondant à l'intensité ou au niveau de gris [12] [13].

Elle peut être analogique ou numérique, lorsque les points (x, y) et l'amplitude sont discrétisés, on parle de l'image numérique, c'est le type d'image que l'ordinateur peut manipuler, da surface est divises en élément de taille fixes appelé pixel.

1.4.2 Le pixel

Contraction de l'expression anglaise «picture elements » : élément d'image, le pixel le plus petit point de l'image, c'est une quantité calculable qui peut recevoir une structure et une quantification. Le pixel est le plus petit élément que peuvent manipuler les matériels et logiciels d'affichage ou d'impression [13]. Chaque pixel est caractérisé par ses coordonnées (x,y), les informations qu'ils contiennent permettent de distinguer entre plusieurs types d'images.

1.4.3 Les types de l'image

Nous distinguons les types suivants :

- **Image binaire :** Chaque pixel est soit blanc soit noir. Puisqu'il y a uniquement deux valeurs pour chaque pixel, un seul bit est utilisé pour le coder.

- **Image en niveaux de gris :** Chaque pixel est un niveau de gris, allant de 0 (noir) à 255 (blanc). Cet intervalle de valeurs signifie que chaque pixel est codé sur huit bits (un octet).

- **Image couleur (ou RGB) :** Chaque pixel possède une couleur décrite par la quantité de rouge (R), vert (G) et bleu (B). Chacune de ces trois composantes est codée sur l'intervalle [0, 255].

Plusieurs opérations matricielles usuelles telles que l'addition, la concaténation s'appliquent sur les images, en prenant en considération les intervalles des canaux, c'est à dire sans déborder ces intervalles qui sont généralement entre 0 et 255. Parmi les opérations les plus utilisées on trouve *le filtrage.*

1.4.4 Le filtrage

Le filtrage consiste à appliquer une transformation (appelée filtre) à tout ou une partie d'une image numérique son principe est de modifier la valeur des pixels d'une image, généralement dans le but d'améliorer son aspect à titre d'exemple l'élimination des valeurs aberrantes. En pratique il s'agit de créer une nouvelle image en se servant des valeurs des pixels de l'image d'origine. Ces filtres sont classés selon l'opérateur appliqué en deux catégories :

Les filtres linéaires : Ils utilisent une combinaison linéaire des valeurs du voisinage pour déterminer le niveau de gris d'un pixel.

Les filtres non-linéaires : ces filtres ne peuvent pas être exprimés par une combinaison linéaire, elles sont plus complexes à mettre en œuvre que les filtres linéaires. Cependant les résultats obtenus avec les filtres non-linéaires sont très souvent de meilleure qualité que ceux obtenus par les filtres linéaires.

Parmi les filtres linéaires, nous citons la convolution.

1.4.5 La convolution

C'est l'opérateur de base du traitement linéaire des images. Elle permet de calculer l'approximation des dérivées horizontales et verticales. Elle est principalement utilisée dans l'extraction des caractéristiques (features). La convolution (Figure1.2) calcule le produit scalaire entre une matrice de convolution et le voisinage d'un pixel, selon l'équation (1.1) [14]

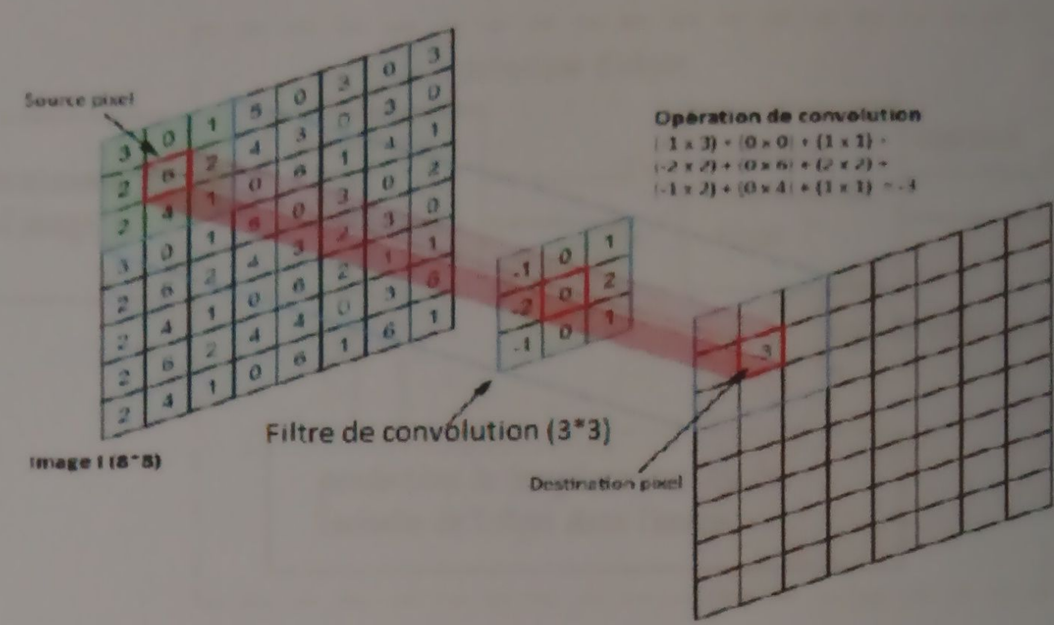


Figure 0-2 : Exemple de convolution.

$$(I * h)[x, y] = \sum_{i=x_1}^{x_2} \sum_{j=y_1}^{y_2} h[i, j].I[x + i, y + j] \tag{1.1}$$

L'objectif d'effectuer ces opérateurs sur l'image est d'obtenir une image améliorée afin d'effectuer certains traitements utiles comme : l'extraction des informations utiles, la compréhension de l'image et la description de ses caractéristique et des objets qu'elle contienne. La détection d'objets est l'une des applications qui utilisent ces traitements.

1.5 Détection d'objets

La détection d'objets est une tâche principale dans la vision par ordinateur, car elle est appliquée dans nombreuses applications du monde réel tel que la robotique et les véhicules autonomes. Cette tâche est difficile à réaliser pour les machines par rapport aux êtres humains qui effectuent une détection d'objet très facilement et instantanément.

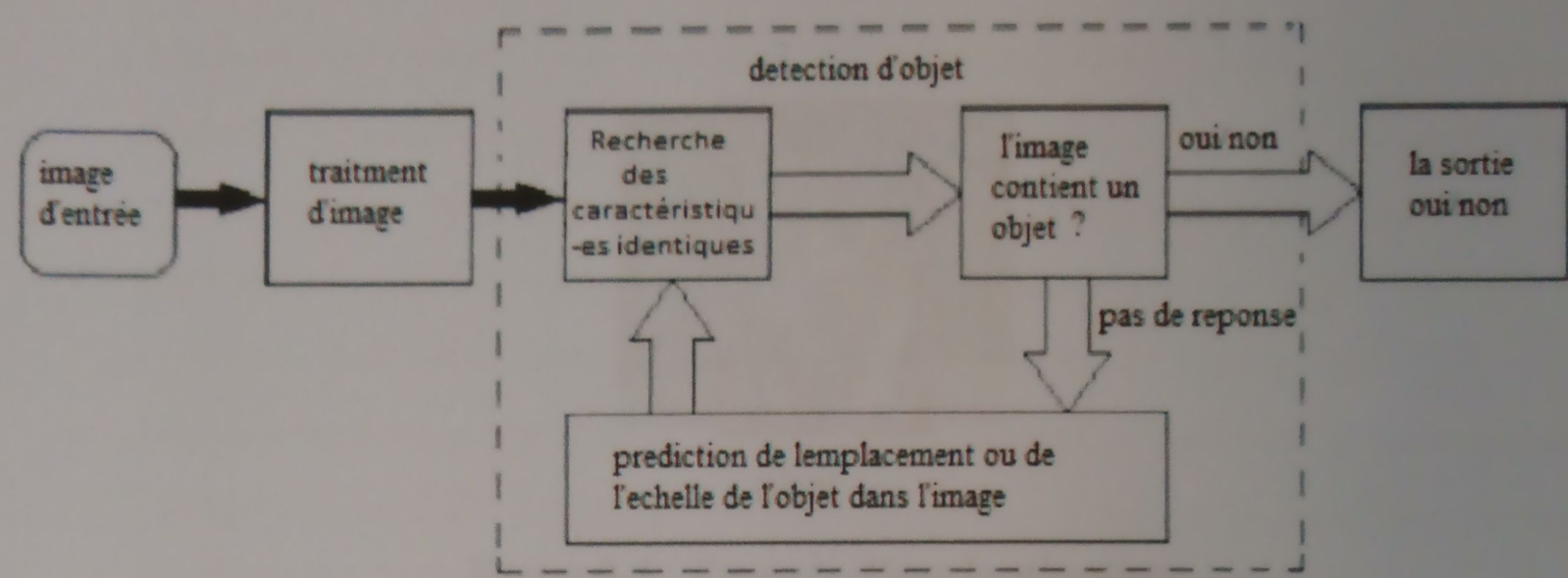


Figure 0-3 : Schéma bloc d'un système de détection d'objet.

Le but fondamental de ce système est de détecter des objets qui sont présents dans l'image ou la scène et de classer chaque objet dans la classe correspondante. La détection d'objets dans une image englobe trois processus :

La Classification : elle fait référence à l'action de classifier donc de déterminer des critères de classement afin de découvrir ce qu'est cet objet. En d'autres termes, Elle consiste à attribuer une classe ou une catégorie à chaque objet (ou individu) à classer, en se basant sur en se basant sur des données statistique [15] (Figure1.4).

chat

Figure 0-4 : Exemple d'une classification.

La localisation : Il s'agit de trouver la position de l'objet dans l'image (figure 1.5). Beaucoup de définitions différentes de la localisation d'objets peuvent être trouvés dans la littérature. Elles diffèrent par la forme par laquelle l'emplacement de l'objet dans une image est représenté, par exemple par son point central, ses contours, un rectangle qui l'encadre, ou une segmentation pixel par pixel.

Figure 0-5 : Exemple d'une localisation.

La Détection d'objet : classifiez et détectez tous les objets de l'image. Attribuez une classe à chaque objet et tracez un cadre de sélection autour de celui-ci (figure1.6)

Figure 0-6 : Exemple d'une détection d'objet multicible.

Dans ce document, nous passerons en revue les différentes techniques classiques utilisées pour détecter les objets dans les images, nous citons :

- **Minimisation de distance [16]:** Le principe des méthodes reposant sur la minimisation de distance consiste à rechercher la classe la plus proche pour chaque pixel, ou groupe de pixels si l'on travaille dans une fenêtre d'analyse centrée sur le pixel courant. La notion de proximité est liée à la distance considérée.
- **Les approches stochastiques:** (Champs de Markov, inférence bayésienne, règle de Bayes) : dans ce type d'approches de nombreux travaux reposent sur la théorie des champs de Markov et l'estimation bayésienne.
- **L'histogramme de gradient oriente :** Les descripteurs HoG sont introduits par Dalal et Triggs. L'idée essentielle de HoG est que l'apparence locale et la forme d'objet dans

une image sont décrites par la distribution d'intensité des gradients ou de direction des contours. La mise en œuvre de ces descripteurs peut être obtenue en divisant l'image en petites régions connectées, appelées cellules, et pour chaque cellule un histogramme des directions de gradient ou des orientations de contour est calculé pour les pixels dans la cellule. La combinaison de ces histogrammes représente le descripteur [17].

- **La transformation d'entité invariante par l'échelle :** est une méthode développée par David Lowe en 2004 permettant de transformer une image en un ensemble de vecteurs de caractéristiques qui sont invariants par les transformations géométriques usuelles (homothétie, rotation) et de manière moins fiables aux transformations affines et l'illumination [18].
- **La méthode de Viola-Jones :** La méthode de Viola et Jones consiste à balayer une image à l'aide d'une fenêtre de détection de taille initiale 24x 24 pixels (dans l'algorithme original) et de déterminer si un visage y est présent. Lorsque l'image est parcourue entièrement, la taille de la fenêtre est augmentée et le balayage recommence, jusqu'à ce que la taille de la fenêtre aboutisse à la taille de l'image à traiter. L'augmentation de la taille de la fenêtre se fait par un facteur multiplicatif de 1.25. Le balayage, quant à lui, consiste simplement à décaler la fenêtre d'un pixel. Ce décalage peut être changé afin d'accélérer le processus, mais un décalage d'un pixel assure une précision maximale [19] [20].

Les approches citées ci-dessus sont des méthodes classiques, leur problème réside dans la complexité de construction d'un bon extracteur de caractéristiques qui doit être réétudié pour chaque nouvelle application. C'est là qu'intervient l'apprentissage profond ou deep learning en anglais. C'est une classe de méthodes dont les principes sont connus depuis la fin des années 1980, mais dont l'utilisation ne s'est généralisée que depuis 2012, environ.

Le système entraînable est constitué d'une série de modules, chacun représentant une étape de traitement. Chaque module est entraînable en comportant des paramètres ajustables similaires aux poids des classifieurs linéaires. Le système est entraîné de bout en bout: à chaque exemple, tous les paramètres de tous les modules sont ajustés de manière à rapprocher la sortie produite par le système de la sortie désirée. Le qualificatif profond vient de l'arrangement de ces modules en couches successives [21].

1.6 Conclusion

Dans ce chapitre, une introduction générale à la robotique particulièrement le fonctionnement et les composants constituant un robot autonome a été présentée. Nous avons abordé les notions de base des traitements d'images usuels pour un robot mobile afin de détecter les objets perçus dans son environnement et les classifier. Par la suite nous avons cité quelques méthodes traditionnelles de la détection d'objet. Dans le prochain chapitre, nous aborderons particulièrement les réseaux de neurones et les réseaux de neurones profonds.

Chapitre 2 : Les réseaux de neurones

2.1 Introduction

L'intelligence artificielle souvent abrégée avec le sigle IA, elle est une branche de l'informatique fondamentale, qui cherche à résoudre des problèmes logiques ou algorithmiques complexes qui ne peuvent pas être résolus avec des algorithmes classiques ou qui sont très difficiles à programmer.

Plus précisément IA est une technique informatique qui permet à la machine de reproduire des fonctions que seul l'être humain peut effectuer du fait de leur complexité. En d'autres termes elle constitue des méthodes imitant ou remplaçant l'être humain. Mais c'est à l'homme d'entrainer la machine ce qu'elle doit apprendre pour l'aider à reproduire sa propre représentation du monde et c'est ce qu'on appelle l'apprentissage automatique, en anglais « Le machine Learning » [22].

Arthur Samuel propose la définition suivante [23]:« L'apprentissage automatique est un champ d'étude qui donne aux ordinateurs la capacité d'apprendre sans être explicitement programmés ».

L'apprentissage automatique est donc l'ensemble des méthodes permettant à une machine d'apprendre à partir de données d'échantillonnage, des exemples et des expériences passés.
L'un des grands domaines de l'apprentissage automatique et les réseaux de neurones, comme son nom l'indique ce sont des réseaux de neurones mais artificiels, inspirés du système nerveux humain, ils sont très utilisés pour effectuer l'apprentissage. Ils ont la possibilité d'apprendre à partir des exemples et des échantillons, en plus de cela, ils sont capables de généraliser c'est à dire après avoir fait la tâche de l'apprentissage, ces derniers peuvent traiter de nouvelles données (données qui n'ont pas intervenu pendant la phase apprentissage). Aujourd'hui les réseaux de neurones sont capables de résoudre des problèmes dans divers domaines tel que la médecine, l'industrie, le commerce.

Dans ce chapitre nous allons aborder en détails le principe des réseaux de neurones plus particulièrement l'apprentissage profond. Par la suite, nous évoquerons quelques modèles de l'apprentissage profond cités dans la littérature.

2.2 L'apprentissage automatique

L'apprentissage automatique fait référence au développement, à l'analyse et à l'implémentation de méthodes qui permettent à une machine d'évoluer grâce à un processus d'apprentissage, et ainsi de remplir des tâches qu'il est difficile ou impossible de remplir par des moyens algorithmiques classiques. Son objectif est d'extraire et d'exploiter automatiquement l'information présente dans un jeu de données.

Tom MITCHELL définit le concept de l'apprentissage dans son livre de la manière suivante [24] : un programme apprend d'une expérience *E* à effectuer une tâche *T*, si sa mesure de performance *P*, mesurée pour la tâche *T*, s'améliore avec l'expérience *E*. Cette définition suggère l'idée qu'un algorithme d'apprentissage est capable d'inférer un comportement, des règles à partir d'exemples qui lui sont fournis et de les généraliser à des situations nouvelles [25].

La tâche *T* peut être un processus regroupant une ou plusieurs fonctions. Parmi ces fonctions, on peut citer :

- **La régression :** La régression est l'une des méthodes statistiques les plus simples, elle cherche à établir une relation linéaire entre une variable dépendante ou expliquée et une ou plusieurs variables explicatives ou variables indépendantes [26].
- **La classification :** La classification est l'une des techniques de l'apprentissage supervisé les plus utilisées. Son principe consiste à identifier à quelle catégorie (classe) appartient une nouvelle observation. À titre d'exemple, on peut considérer la classification des fleurs, où il est question de trouver la classe à laquelle appartient une fleur sachant le nombre de pétales, la couleur, etc ...
- **Le regroupement :** aussi connu sous nom (Segmentation) est un regroupement en classes homogènes représentant un nuage des points d'un espace quelconque en un ensemble de groupes appelé Cluster [27]. Un «Cluster» est donc une collection d'objets qui sont «similaires» entre eux et qui sont «dissemblables» par rapport aux objets appartenant à d'autres groupes.
- **La détection d'anomalies :** Le but de la détection d'anomalie est de repérer des *données* qui ne sont pas conformes à ce à quoi l'on peut s'attendre par rapport aux autres données. Il s'agit, par exemple, de données qui ne suivent pas le même schéma ou qui sont atypiques pour la distribution de probabilité observée. Parmi les exemples d'applications courantes, citons les transactions bancaires (où une anomalie sera vue

comme une fraude potentielle), la surveillance des données physiologiques d'un malade (l'anomalie est un problème de santé possible), ou encore la détection de défauts dans des chaines de production [28].

- **Dé-bruitage (Denoising) :** Cette tâche consiste à nettoyer ou reconstituer une donnée en entrée qui est corrompue, et retourner la donnée originale.

L'expérience E peut être définie par un scénario que le programme est sensé suivre durant son apprentissage, ce scénario est appelé un algorithme d'apprentissage classé en trois grandes catégories :

- **L'apprentissage supervisé :** Dans l'apprentissage supervisé, l'ordinateur est fourni avec des exemples d'entrées qui sont étiquetés avec les sorties souhaitées. Le but de cette méthode est que l'algorithme puisse «apprendre» en comparant sa sortie réelle avec les sorties «enseignées» pour trouver des erreurs et modifier le modèle en conséquence. L'apprentissage supervisé utilise donc des modèles pour prédire les valeurs d'étiquettes sur des données non étiquetées supplémentaires. Par exemple, avec un apprentissage supervisé, un algorithme peut être alimenté avec des images de requins étiquetés Poisson des images d'océans étiquetés comme océan. En étant formé sur ces données, l'algorithme d'apprentissage supervisé devrait être capable d'identifier plus tard des images de requin non marquées comme Poisson des images océaniques non étiquetées Ocean.
- **L'apprentissage non supervisé :** L'apprentissage non supervisé ou clustering ne demande aucun étiquetage préalable des données. Le but est que le modèle réussit à regrouper les observations disponibles en catégories par lui-même.
- **L'apprentissage par renforcement :** Consiste, pour un agent autonome (robot, etc…), à apprendre les actions à réaliser, à partir d'expériences, de façon à optimiser une récompense quantitative au cours du temps. L'agent est plongé au sein d'un environnement, et prend ses décisions en fonction de son état courant [29].

La performance P désigne des mesures quantitatives qui servent à évaluer l'algorithme de l'apprentissage effectué par la tâche T, à titre d'exemple nous citons la précision et le rappel.

Quelques exemples d'application de l'apprentissage

L'apprentissage automatique est très utilisé de nos jours, pour résoudre des problèmes dans différents domaines, parmi ces problèmes on cite :

- **La reconnaissance de forme :** plusieurs applications dans ce domaine par exemple la reconnaissance du visage, la reconnaissance de caractère, reconnaissance d'objets.
- **Aide à la médecine :** des applications qui prédisent la maladie du patient à partir des symptômes et des résultats des analyses.
- **Les jeux :** il existe plusieurs programmes qui maîtrisent les jeux.
- **Traitement du langage naturel :** traduction automatique, correction d'orthographe…
- **La détection de fraudes :** prédire le cas de transactions frauduleuses.

2.3 Les réseaux de neurones

2.3.1 Neurone biologique

Le neurone biologique est une cellule nerveuse qui se compose de : dendrites, Noyau, Axone et Synapse (Figure 2.1). Cette cellule renvoie des signaux électriques à d'autres cellules à travers des connexions appelées synapse. Chaque neurone reçoit et accumule des charges au niveau de ses dendrites provenant des autres neurones qui émet à son tour un signal électrique au long de son axone lorsque les charges atteignent un certain seuil [30].

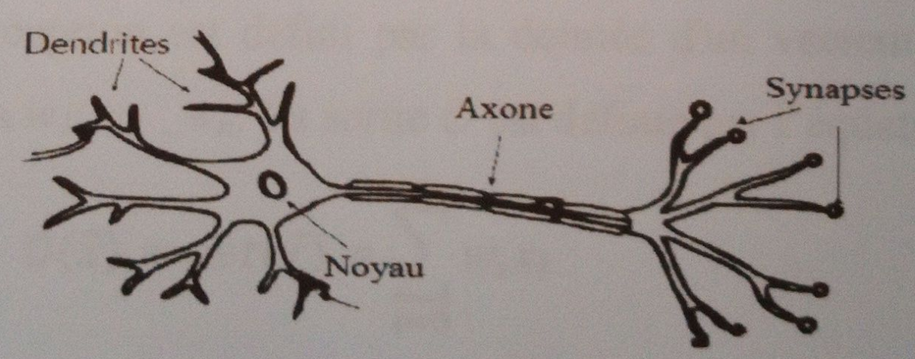


Figure 2-1 : Représentation d'un réseau de neurone biologique.

2.3.2 Les réseaux de neurones artificiels

Le système des réseaux de neurones artificiels est un système dont la conception est schématiquement inspirée du fonctionnement des neurones biologiques, et qui par la suite s'est rapprochée des méthodes statistiques [30]. Les neurones biologiques sont représentés par de simples neurones artificiels et l'interconnexion entre ces neurones (les synapses) est représentée

par des poids (valeurs) pour chaque interconnexion. Le corps cellulaire est modélisé par la fonction de transfert, appelée aussi fonction d'activation et l'axone par l'élément de sortie. Mathématiquement parlant, un réseau de neurones est une fonction qui pour un certain vecteur d'entrée elle calcule une valeur de sortie. La version la plus simplifiée d'un réseau de neurones est un perceptron nommé aussi perceptron linéaire.

Un perceptron contient plusieurs entrées et chaque entrée possède un poids. Les entrées sont toutes connectées à une seule sortie. La figure 2.2 illustre une représentation mathématique d'un perceptron.

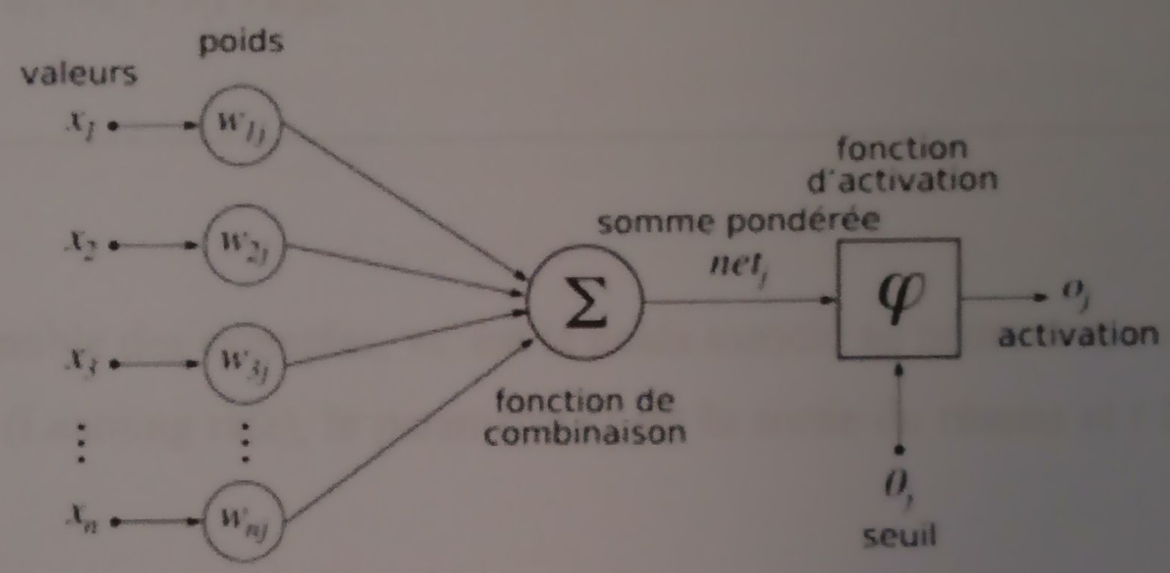


Figure 2-2 : Représentation mathématique d'un perceptron.

Un perceptron prend en entrée un vecteur x de n valeurs $x_1, x_2, \ldots, x_n$ et calcule une sortie O *(équation 2.1)*. Un perceptron est défini par la donnée d'un vecteur w de n constantes les coefficients synaptiques $w_1, w_2, \ldots w_n$. La sortie O est définie par l'équation (2.1) :

$$O(\vec{x}) = net(\vec{x}) = \sum_{i=0}^{n} w_i x_i \qquad (2.1)$$

2.3.2.1 Algorithme d'apprentissage d'un perceptron par correction d'erreur

L'algorithme d'apprentissage peut être décrit succinctement de la manière suivante :
Les poids du perceptron sont initialisés aléatoirement. A chaque nouvel exemple, les poids sont ajustés selon que le perceptron l'a correctement classé ou non. L'algorithme s'arrête lorsque tous les exemples ont été présentés sans modification des poids.

Les étapes de l'algorithme *Train-Perceptron* sont comme suit :

Algorithme *Train-Perceptron* (D º $\{<x, t(x)$ º $c(x)>\}$) :

Début

Initialiser tous les poids w_i à des valeurs aléatoires

TANTQUE pas tous les exemples sont prédit correctement **FAIRE**

POUR chaque exemple d'apprentissage x Î D

Calculer la sortie actuel $O(x)$

POUR ($i = 1$ à n) FAIRE

$w_i := w_i + r(t - o)x_i$

FIN

---

où:

D désigne l'ensemble des exemples, w_i est le poids associé au neurone i, r représente le taux d'apprentissage (Learning rate), le paramètre O est la sortie du réseau et t la cible (la valeur désirée).

Le processus d'apprentissage du perceptron est une procédure de correction d'erreur puisque les poids ne sont pas modifiés lorsque la sortie attendue est égale à la sortie calculée par le perceptron courant.

2.3.2.2 Algorithme d'apprentissage d'un perceptron par la descente du gradient

Le principe de cet algorithme consiste à calculer l'erreur $E[x]$ et de la minimiser à la sortie O du perceptron. Cette dernière est définie en utilisant la fonction erreur quadratique par l'équation 2.2:

$$E[x] = \frac{1}{2}\sum_{x \in D}\left(t(x) - O(x)\right)^2 \qquad (2.2)$$

L'erreur mesurée représente l'écart entre les sorties attendues et calculées sur l'échantillon complet. Si l'erreur $E[x]$ converge vers la valeur 0, ceci induit que le perceptron a classifié correctement l'échantillon.

Le problème est donc de déterminer un vecteur qui minimise ($E[x]$). Une méthode qui permet de rechercher le minimum d'une fonction est d'utiliser la méthode du gradient. L'algorithme de

descente du gradient calcule le gradient de la courbe de perte au point de départ. Un gradient est un vecteur de dérivées partielles indiquant la direction à suivre pour atteindre le minimum recherché.

La descente du gradient stochastique : Cet algorithme calcule la moyenne des gradients de la fonction d'erreur pour chaque paramètre du réseau à partir de tous les exemples d'apprentissage puis minimise la fonction en chaque itération en mettant à jour les paramètres en fonction des moyennes calculées [31].

La descente du gradient ordinaire : L'algorithme prend en considération les exemples d'apprentissage un par un. Il calcule à chaque itération les valeurs des gradients des paramètres du réseau en fonction d'un seul exemple pour ensuite mettre à jours ces paramètres. Le processus s'arrête lorsqu'une valeur négligeable de l'erreur globale est atteinte [31].
Le perceptron peut résoudre seulement les problèmes des données qui sont linéairement séparables, dans d'autres cas un réseau multicouche est utilisé.

2.3.3 Réseau de neurones multicouches :

Un réseau de neurones multicouches est un réseau de neurones avec deux ou plusieurs couches cachées, souvent appelé « Feedforward ».

La figure 2.3 montre un réseau de neurones multicouches avec deux couches cachées.

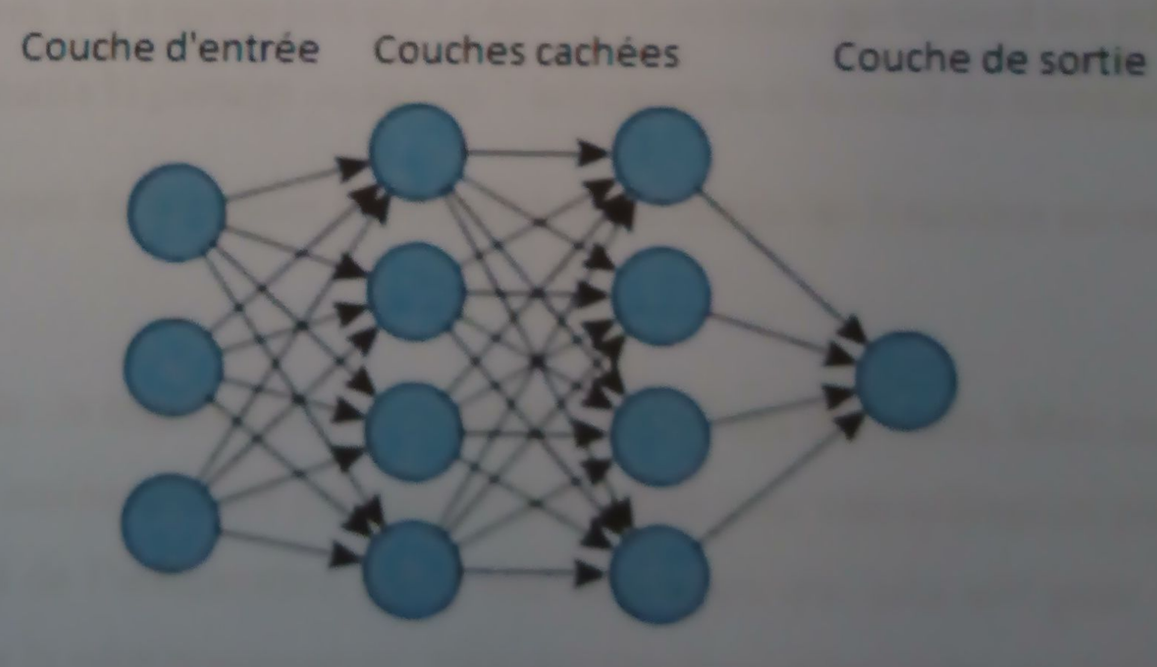


Figure 2-3 : Architecture d'un réseau de neurone avec deux couches cachées.

2.3.3.1 La propagation

Un neurone reçoit des neurones en amont avec un certain nombre de valeurs via ses connexions synaptiques, et il produit une valeur en utilisant une fonction de combinaison. Les réseaux de neurones calculent une combinaison linéaire des entrées, c'est-à-dire que la fonction de combinaison renvoie (propage) le produit scalaire entre le vecteur des entrées et le vecteur des poids. Ceci est fait systématiquement entre les neurones de la couche courante et ceux de la suivante. Ce processus est appelé la propagation.

2.3.3.2 La rétropropagation

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. En pratique, les coefficients du réseau sont modifiés de façon à corriger les erreurs de classification rencontrées, selon une méthode de descente de gradient. Ces gradients sont rétropropagés dans le réseau depuis la couche de sortie.

2.3.3.3 Fonction d'activation

La fonction d'activation est une fonction mathématique inspirée du "potentiel d'action", un phénomène électrique entre deux neurones biologiques. L'activation a pour but de transformer le signal de manière à obtenir une valeur de sortie à partir de transformations complexes entre les entrées. Son rôle est d'appliquer une transformation mathématique aux informations qui arrivent aux neurones. En d'autres termes il s'agit des fonctions qui traitent les informations des neurones pour permettre le passage ou non de l'information si le seuil de stimulation est atteint.

Il existe plusieurs types de fonctions de transfert, nous citons les fonctions suivantes :

- **Fonction sigmoïde** : la fonction la plus populaire depuis des décennies. Mais aujourd'hui, elle devient beaucoup moins efficace par rapport à d'autres pour une utilisation pour les couches cachées. Elle perd de l'information due à une saturation que cela soit pour la phase de la *propagation* ou de *la rétropropagation*. Une des fonctions sigmoïdes est la fonction donnée par l'équation 2.3 dont les valeurs sont comprises dans l'intervalle [0, 1]. La figure 2.4 représente le graphe de la fonction sigmoïde.

$$F(x) = \frac{1}{1+e^{x}} \tag{2.3}$$

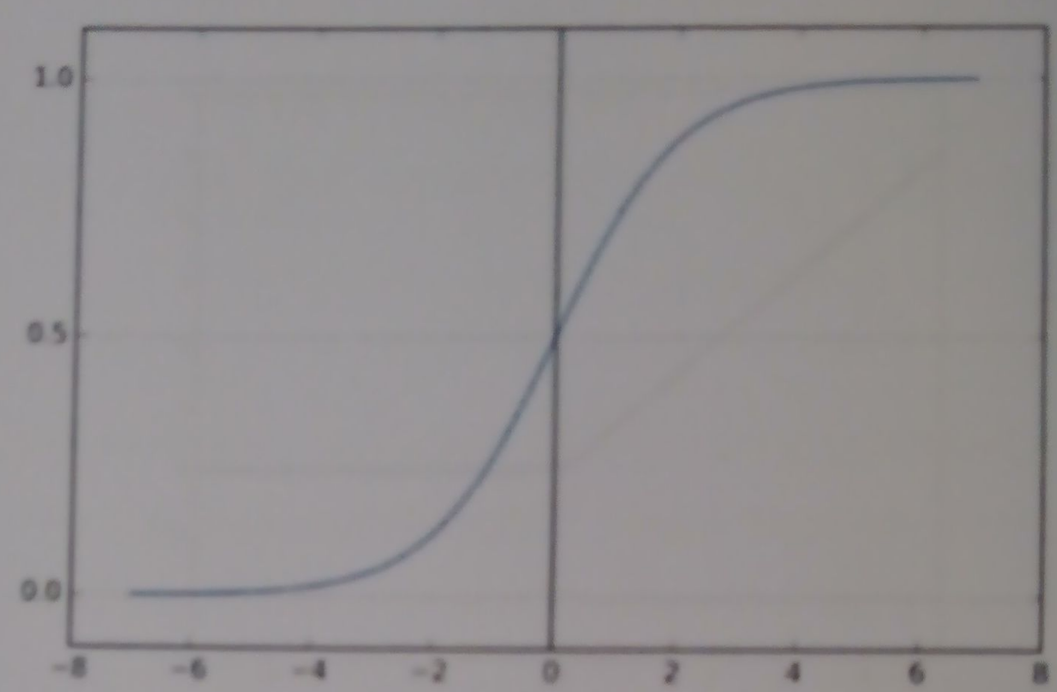


Figure 2-4 : Graphe de la fonction sigmoïde.

- **Fonction linéaire :** Elle est utilisée dans la couche de sortie pour un problème de régression. Cette fonction conduit à des valeurs continues selon une droite d'équation *(y = a x)*. La figure 2.5 illustre un graphe d'une fonction linéaire.

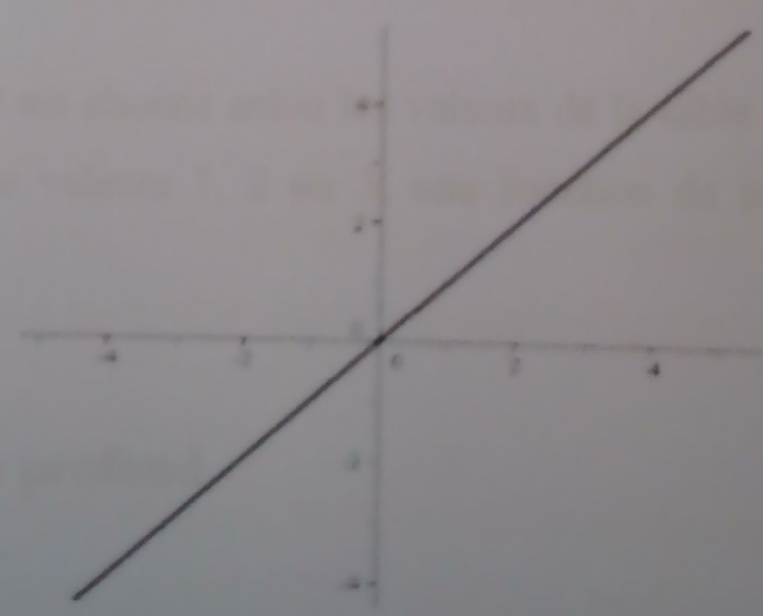


Figure 2-5 : Graphe d'une fonction linéaire.

- **Fonction Relu** : Ce sont les fonctions les plus populaires de nos jours. Elles permettent un entrainement plus rapide comparé aux fonctions sigmoïde. Elles sont aussi plus légères la raison pour laquelle elles sont très utilisées pour les CNN (réseaux neuronaux convolutifs), RBM (Restricted Boltzmann Machines), et les réseaux de multi perceptron.

Les valeurs de la sortie sont comprises dans l'intervalle [0, +∞ [. La figure 2.6 montre le graphe de la fonction RELU.

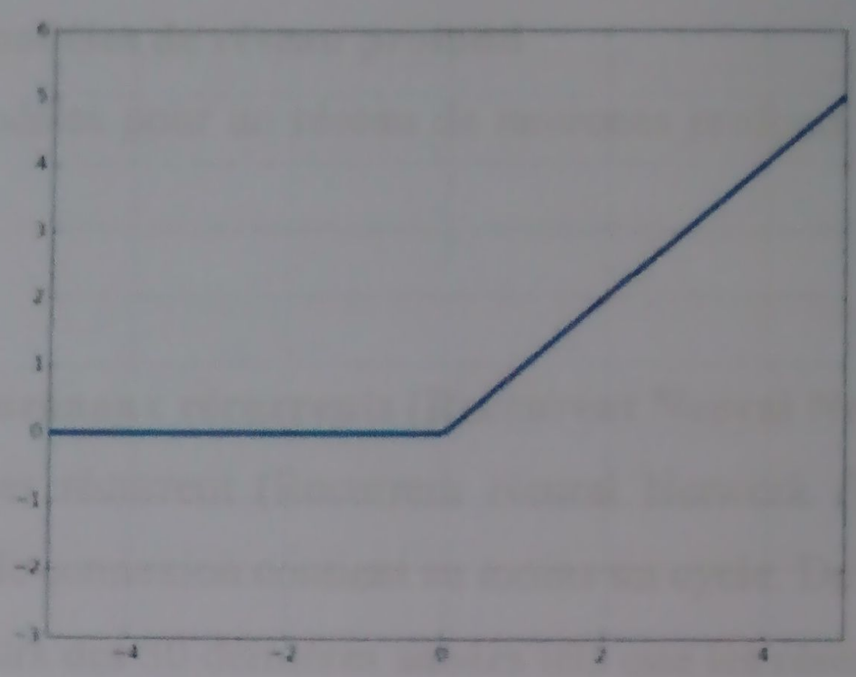


Figure 2-6 : Graphe de la fonction Relu

- **La fonction softmax** : le rôle de cette fonction est de calculer la distribution des probabilités de l'événement sur *n* différents événements. En règle générale, cette fonction calcule les probabilités de chaque classe cible sur toutes les classes cibles possibles. Elle est utilisée généralement dans les couches de sorties [32].

La fonction de transfert est choisie selon les valeurs de la cible (target), par exemple si les valeurs de cible sont des valeurs 1, 2 ou 3, une fonction de transfert linéaire serait plus appropriée.

2.4 L'apprentissage profond

2.4.1 C'est quoi l'apprentissage profond

Le qualificatif profond vient de l'arrangement des modules du réseau en couches successives. Ce qui fait l'avantage des architectures profondes, c'est leur capacité d'apprendre à représenter le monde de manière hiérarchique. Comme toutes les couches sont entraînables, nul besoin de construire un extracteur de caractéristiques à la main. L'entraînement s'en chargera. De plus, les premières couches extrairont des caractéristiques simples (présence de contours) que les couches suivantes combineront pour former des concepts de plus en plus

complexes et abstraits : assemblages de contours en motifs, de motifs en parties d'objets, de parties d'objets en objets, etc.»

2.4.2 Les différents modèles de réseau profond

Il existe plusieurs modèles pour un réseau de neurones profond nous en-citons quelques-uns :

2.4.2.1 Les réseaux neuronaux récurrents (Reccurent Neural Networks)

Un réseau de neurones récurrent (Recurrent Neural Network (RNN)) est un réseau de neurones dont le graphe de connexion contient au moins un cycle. De nombreux types de RNN ont été développés au cours des 30 dernières années tels que les réseaux d'Elman, les réseaux de Jordan et les Echo StateNetworks. [33]

Au cours de ces dernières années, un type de RNN qui est le réseau de neurones à base de cellules Long Short-Term Memory (LSTM) est devenu une norme dû aux performances élevées obtenus des applications aussi nombreuses que variées.

2.4.2.2 Deep Belief networks

Un réseau profond de croyance peut être défini comme une pile de machines RBM dans lesquelles chaque couche RBM communique avec les couches précédente et suivante. Les nœuds d'une couche unique ne communiquent pas entre eux latéralement. Cette pile de RBM peut se terminer par une couche Softmax pour créer un classificateur, ou peut simplement aider à regrouper des données non étiquetées dans un scénario d'apprentissage non supervisé. [34]

2.4.2.3 Réseaux de neurones convolutifs (Convolutional neural networks)

Les réseaux neuronaux convolutifs (CNN) sont une catégorie de réseaux neuronaux. Ils représentent l'architecture profonde la plus répandue. Développés à la fin des années 80 par le pionnier de l'apprentissage profond, Yann Le Cun, ces réseaux sont une forme particulière de réseau neuronal multicouche, dont l'architecture des connexions est inspirée de celle du cortex visuel des mammifères. [35]

Se sont à ce jour les modèles les plus performants pour classer les images : Identification des objets, des visages, des panneaux de signalisation, alimentation de la vision des robots et des voitures autonomes

Leur conception suit la découverte de mécanismes visuels dans les organismes vivants. Ces réseaux de neurones artificiels (aussi baptisés réseau de neurones à convolution, ou CNN) sont capables de catégoriser les informations des plus simples aux plus complexes. Ils consistent en un empilage multicouche de neurones, des fonctions mathématiques à plusieurs paramètres ajustables, qui prétraitent de petites quantités d'informations. Les réseaux convolutifs sont caractérisés par leurs premières couches convolutionelles (généralement une à trois). Une couche convolutive est basée comme son nom l'indique sur le principe mathématique de convolution, et cherche à repérer la présence d'un motif (dans un signal ou dans une image par exemple).

Pour une image, la première couche convolutionelle peut détecter les contours des objets (par exemple un cercle), la seconde couche convolutionelle peut combiner les contours en objets (par exemple une roue), et les couches suivantes (non nécessairement convolutionelles) peuvent utiliser ces informations pour distinguer une voiture d'une moto. Une phase d'apprentissage sur des objets connus permet de trouver les meilleurs paramètres en montrant par exemple à la machine des milliers d'images d'un chien, d'une voiture ou d'un sport. L'un des enjeux est de trouver des méthodes pour ajuster ces paramètres le plus rapidement et le plus efficacement possible. Les réseaux neuronaux convolutifs ont de nombreuses applications dans la reconnaissance d'images, de vidéos ou le traitement du langage naturel.

2.4.2.4 L'architecture des CNN :

Il existe plusieurs architectures des CNNs qui varient des plus basiques aux plus complexes. Une architecture spécifique est plus performante dans certains domaines que d'autres. Par exemple, les réseaux de neurones à convolution sont très utilisés dans le domaine de la vision par ordinateur, tandis que les réseaux de neurones récursifs sont, par exemple, utilisés pour les problèmes d'optimisation combinatoire.

Les principales étapes dans l'architecture d'un réseau de neurone convolutif basique (figure 2.7) sont :

- Une couche de convolution (Convolutional Layer),
- Une couche ReLu (Rectified Linear Unit),
- Une couche de sous-échantillonnage (Pooling/ Subsampling),
- Une couche entièrement connectée (Fully Connected Layer),
- Une fonction de perte
- Une fonction d'activation
- La normalisation

La figure (2.7) représente le schéma d'un réseau de neurone convolutif basique.

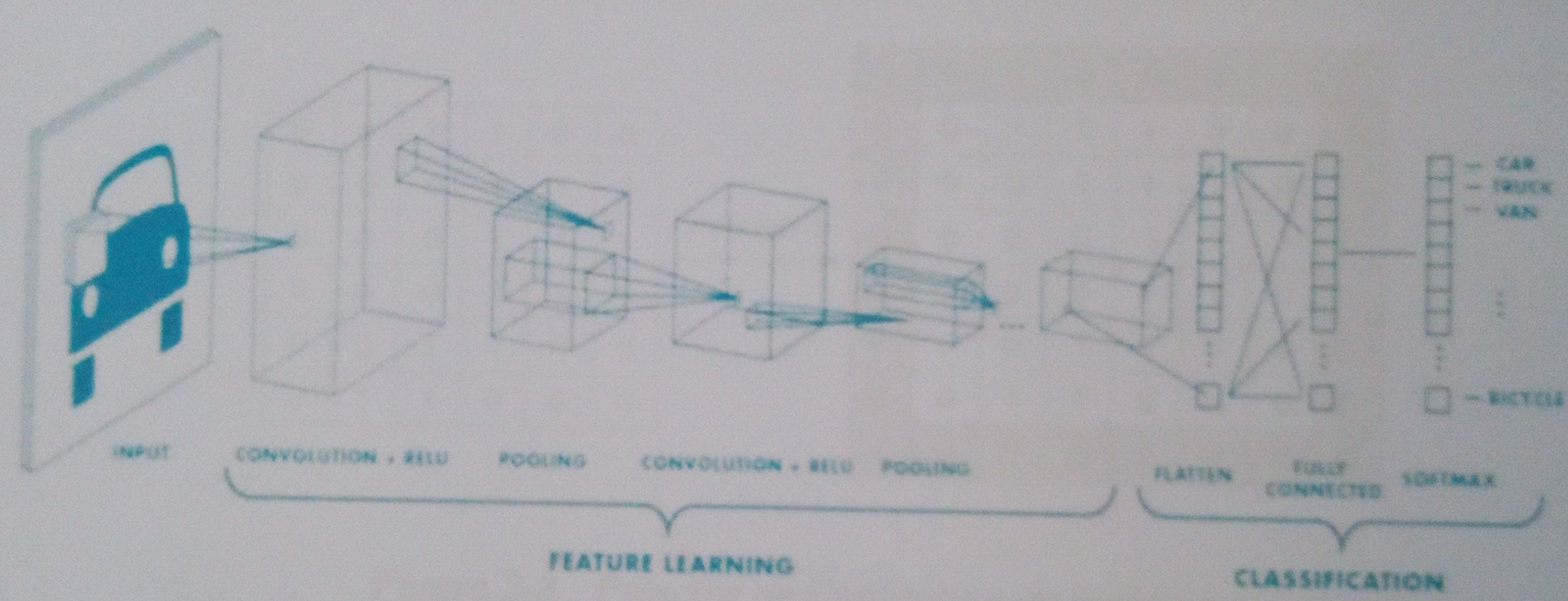


Figure 2-7 : Architecture d'un réseau de neurone convolutif basique.[35]

❖ **La couche de convolution :**

C'est une composante essentielle dans ces réseaux de neurones (d'où est venu leur nom). Son but est de repérer la présence d'un ensemble de caractéristiques (*features*) dans les images reçues en entrée. Pour cela, on réalise un filtrage par convolution : le principe est de faire "glisser" une fenêtre représentant la caractéristique de l'image, et de calculer le produit de convolution entre le filtre et chaque portion de l'image balayée et produire une carte à motif ou bien carte de caractéristiques (features map) à sortie.

Avant l'étape de la convolution, il faut déterminer quelques paramètres, ces paramètres font partie du calcul de la taille des images à la sortie de chaque couche (les cartes à motifs). Ces paramètres sont :

- **Le pas (Stride)** : Le nombre de pixels avec lequel le filtre de convolution se déplace sur l'image et ou le nombre de pixel avec lequel la matrice de pooling se glisse.
- **Le padding** : le padding est le rajout des pixels aux bords de l'image pour pouvoir appliquer la convolution sur les bordures de l'image. Le nombre de lignes et de colonnes rajoutés dépend de la taille du filtre de la convolution.

Si par exemple on a un filtre de taille *SxS* on rajoute $\frac{S-1}{2}$ lignes en dessus de la première ligne et $\frac{S-1}{2}$ au-dessus de la dernière ligne, même chose pour les colonnes. Le padding le plus utilisé est le zéro-pading, comme le montre la figure 2.8.

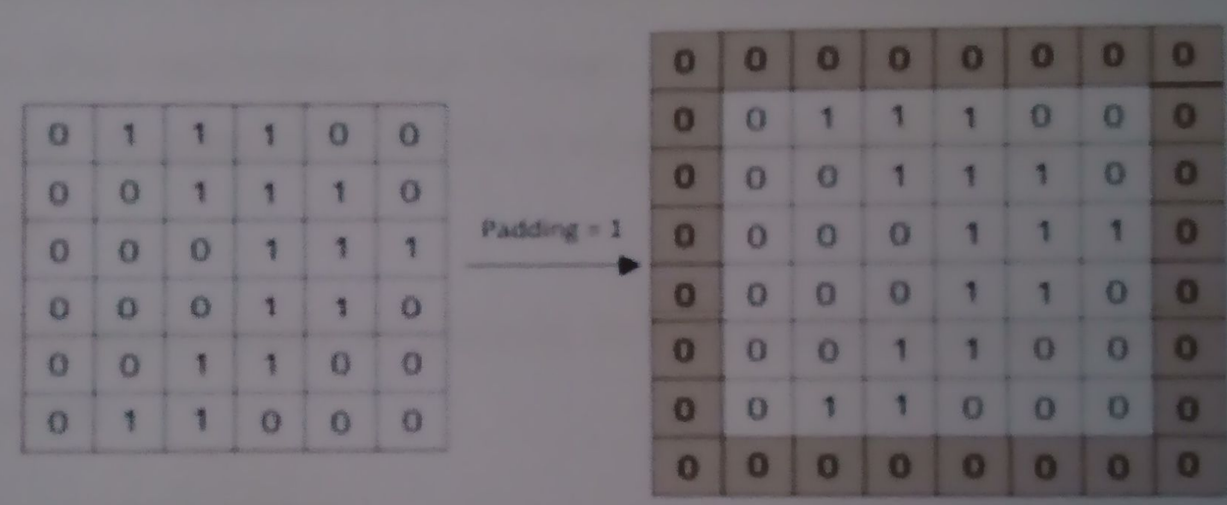


Figure 2-8 : Principe de l'opération de padding.

Le principe de la convolution est illustré sur la figure (2.9).

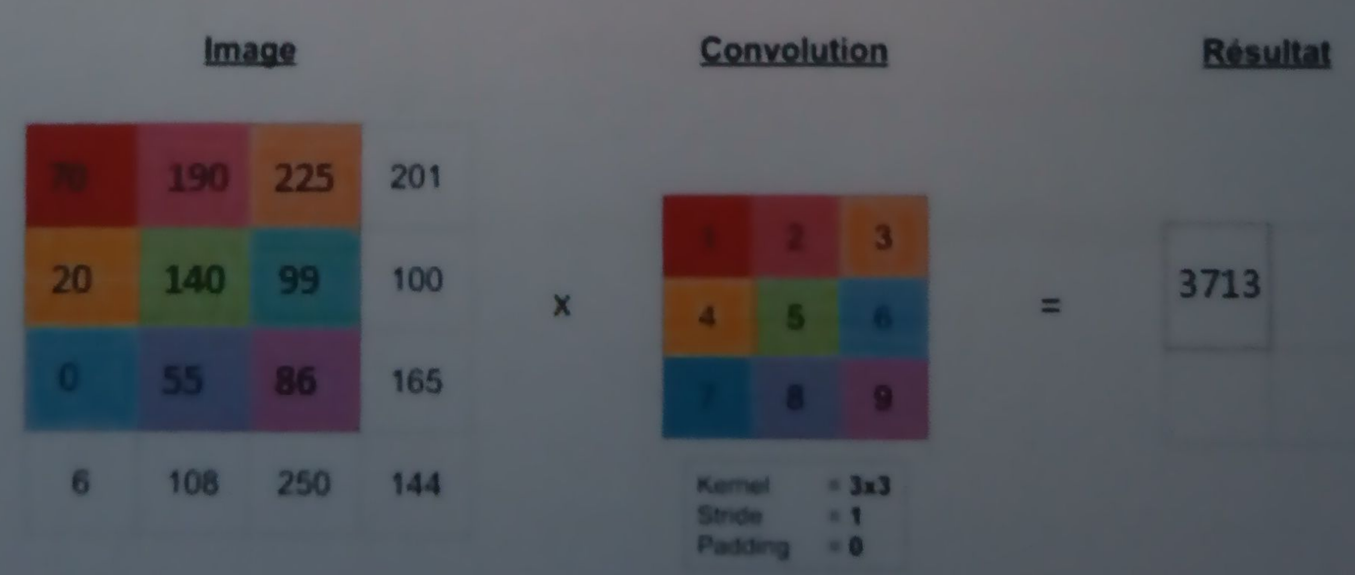


Figure 2-9 : Principe de la convolution.

Selon les valeurs de l'image en entrée illustrée sur la figure (2.9), le calcul de la convolution est effectué comme suit :

Résultat = 70x1+190x2+225x3+20x4+140x5+99x6+0x7+55x8+86x9=3713

A la sortie de chaque couche de convolution la taille de l'image est réduite. La nouvelle dimension MxM est calculée par l'équation (2.4) suivante :

$$M = \frac{W-K+2P}{S} + 1 \qquad (2.4)$$

où :

w est la taille de l'image en entrée, K représente la taille du filtre, P est la valeur du padding et S désigne le pas de convolution

- **Sous échantillonnage (Le pooling)** :

C'est une méthode mathématique permettant de réduire la taille de l'image sans perdre les informations les plus importantes dans l'image. Son principe consiste à diviser l'image en petites matrices par exemple *2x2* et choisir la valeur maximale de chaque matrice, pour obtenir une image réduite.

La figure (2.10) suivante montre un exemple de l'application d'un sous échantillonnage 2x2 avec un pas égal à 2.

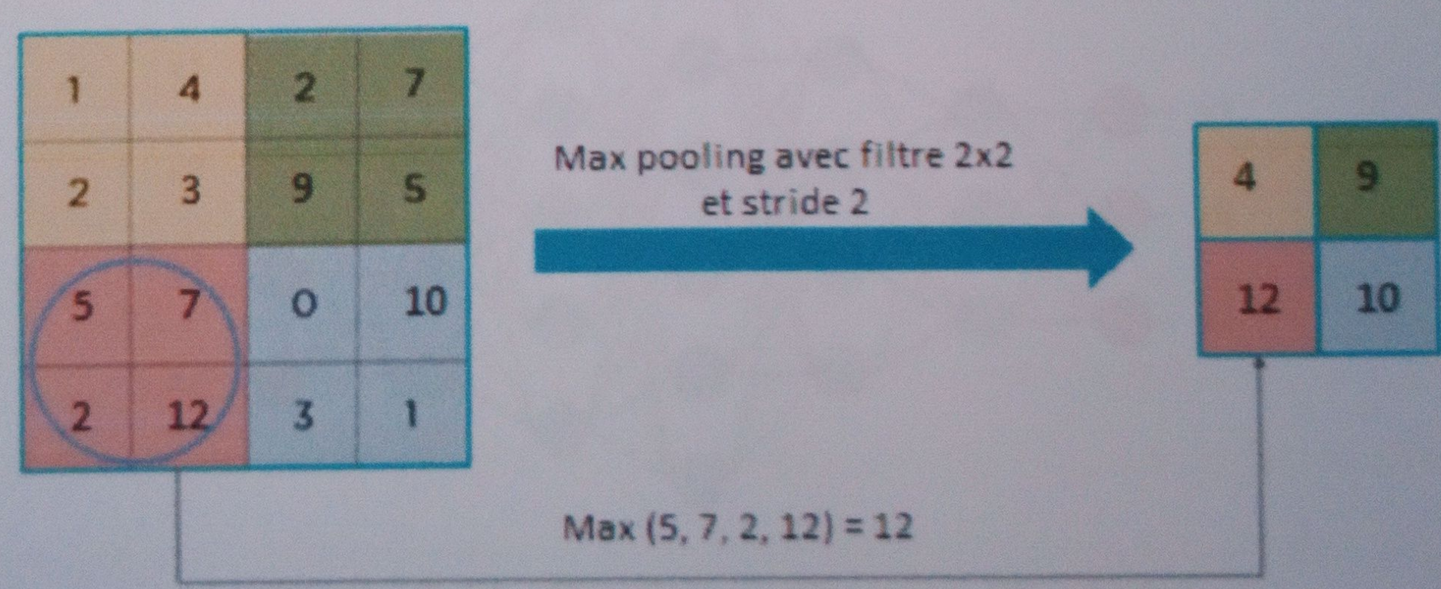


Figure 2-10 : Principe du sous échantillonnage.

❖ **La couche Relu (*Rectified Linear Units*)** :

Chaque couche du réseau convolutif possède une fonction d'activation, la fonction la plus utilisée dans les CNN est la fonction Relu pour les couches de convolution, et softmax pour la couche entièrement connectée.

❖ **La couche entièrement connectée :**

Elle se trouve en sortie du réseau (figure 2.11). Contrairement aux autres couches, cette dernière n'effectue l'extraction de caractéristiques mais elle est consacrée pour le traitement de classification. Cette couche prend essentiellement un volume d'entrée (quelle que soit la sortie de la couche précédente convolution ou *ReLU* ou la couche de pooling et génère un vecteur à *N* dimensions. Le paramètre *N* est le nombre de classes que l'utilisateur doit le définir. Par exemple, pour un programme de classification de chiffres, *N* est égale à 10, car il y a 10 chiffres. Chaque nombre dans le vecteur généré représente la probabilité de la classe correspondante. Par exemple, si le vecteur résultant d'un programme de classification de chiffres est [0.1, 0.1, 0.75, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.05], cela signifie que il y a une probabilité de 10% que l'image soit un 1, une probabilité de 10% que l'image soit un 2, une probabilité de 75% que l'image soit un 3 et une probabilité de 5% que l'image soit un 9.

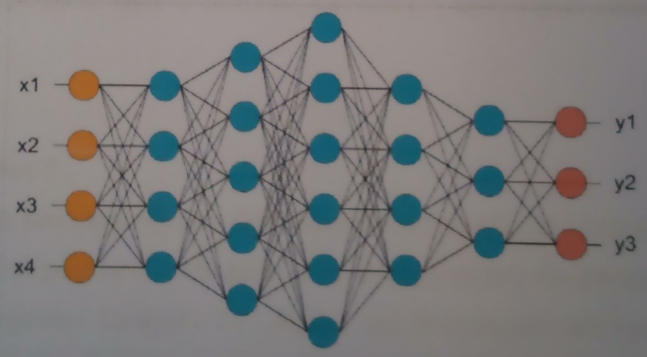


Figure 2-11 : Couches entièrement connectées.

La couche de perte spécifie comment l'entrainement du réseau pénalise l'écart entre le résultat prévu et réel. Elle est normalement la dernière couche dans le réseau. Diverses fonctions de perte adaptées à différentes tâches peuvent y être utilisées.

❖ **La Normalisation :**

Il est connu depuis longtemps que l'apprentissage du réseau converge plus rapidement si ses entrées sont centrées réduites (moyenne = 0, variance = 1). Comme chaque couche observe des entrées produites par des couches qui la précèdent, il serait avantageux d'obtenir des entrées centrées et réduites pour chaque couche.

Batch normalisation est une composante qui se trouve entre les couches du réseau de neurones et qui prend continuellement la sortie d'une couche particulière et la normalise avant de l'envoyer à la couche suivante comme entrée. Le principe est de diviser les intensités des pixels par 255.

❖ **Le Dropout :**

Le Dropout est une technique où des neurones sélectionnés au hasard sont ignorés pendant l'apprentissage. Cela signifie que leur contribution à l'activation des neurones qui leur succède est temporairement supprimée lors de la phase de propagation et toutes les mises à jour de poids ne sont pas appliquées aux neurones lors de la phase de rétro-propagation.

2.4.2.5 Les architectures des CNN les plus connus

Plusieurs architectures dans le domaine des réseaux convolutifs existent, les plus les plus utilisées sont :

- **LeNet-5**

Est l'un des premiers réseaux de neurones convolutifs, développé par LeCun et al. en 1998. Il a été utilisé pour la classification et la reconnaissance des caractères manuscrits dans les banques et les postes. La figure 2.12 illustre son architecture, elle contient des couches de convolution, des sous-échantillonnages et des couches entièrement connectées à la sortie.

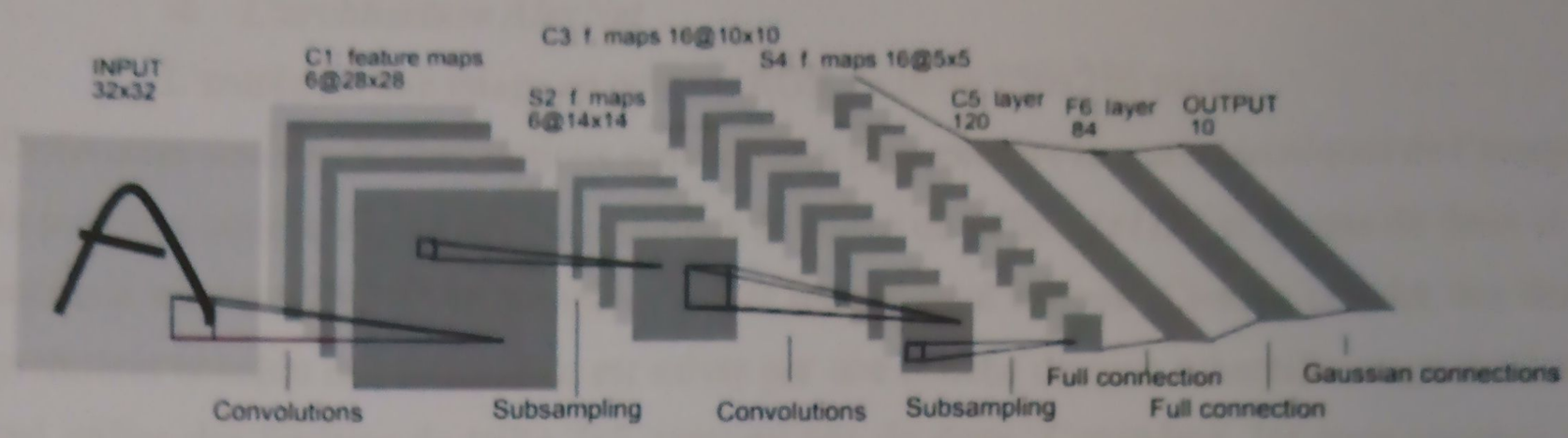


Figure 2-12 : Architecture du réseau LeNet [36].

Description des Couche du réseau *LeNet*

- **La première couche :** Cette première couche est appliquée sur une image en niveau de gris de taille 32x32, avec six filtres de convolution de taille 5x5 et un pas (stride) égal à 1. La dimension de l'image passe de 32x32x1 à 28x28x6.
- **La deuxième couche :** Le LeNet-5 applique une couche de sous-échantillonnage avec une taille de filtre de 2x2 et un pas de deux. Les dimensions de l'image résultante seront réduites à 14x14x6.
- **La troisième couche :** C'est une deuxième couche de convolution avec 16 cartes de caractéristiques (filtres) de taille 5x5 et un pas de 1, les dimensions de l'image passent à 10x10x16.
- **La quatrième couche :** La quatrième couche est à nouveau une couche de sous échantillonnage avec une taille de filtre de 2 x 2 et un pas de 2. Cette couche est identique à la deuxième couche, la différence est qu'elle comporte 16 cartes de caractéristiques de sorte que la sortie soit réduite à 5x5x16.
- **La cinquième couche :** La cinquième couche (*C5*) est une couche convolutionnelle entièrement connectée avec 120 cartes de caractéristiques de taille 1x1. Chacune des 120 unités est connectée à tous les 400 nœuds (5x5x16) de la quatrième couche.
- **La sixième couche :** La sixième couche est une couche entièrement connectée avec 84 unités.
- **La couche de sortie:** Enfin, il existe une couche de sortie softmax entièrement connectée avec 10 valeurs possibles correspondant aux chiffres de 0 à 9.

- ***L'architecture AlexNet***

L'entrée est une image en couleur RGB de taille 256x256 pixels.

Différentes couches de convolutions sont utilisées pour extraire les caractéristiques de l'image : la première couche de convolution contient 96 filtres de taille *11x11x3* et un pas de deux et la seconde contient 256 filtres de taille 5x5 et un pas égal à deux. Chaque couche des deux premières couches de convolution est suivie par une couche de sous échantillonnage similaire qui utilise des fenêtres de taille 3x3 avec un pas de deux. Cependant, les trois couches de convolution suivantes (3ième, 4ième et 5ième couches) sont connectées directement suivies par une couche se sous-échantillonnage. La sortie contient deux couches entièrement connectées et une couche softmax avec 1000 éléments pour la classification. La figure 2.13 montre l'architecture du réseau AlexNet.

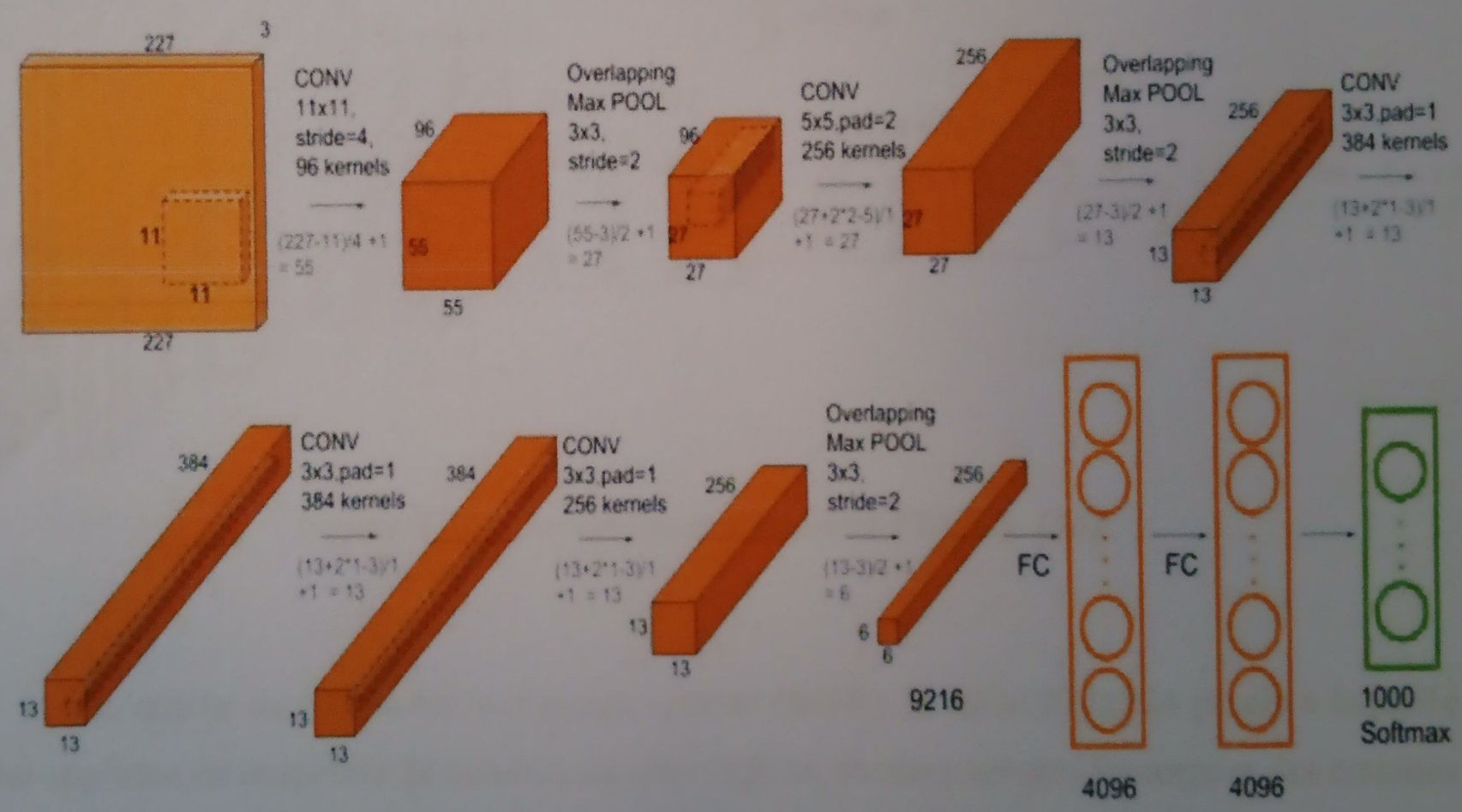


Figure 2-13 : Architecture du réseau AlexNet.[34]

- ***L'architecture VGG16***

VGG16 est un modèle de réseau de neurones convolutif proposé par *K*. Simonyan et A. Zisserman de l'Université d'Oxford dans l'article intitulé «Réseaux convolutionnels très profonds pour la reconnaissance d'images à grande échelle». Le modèle atteint une précision de 92,7% dans le top 5 des tests dans ImageNet, qui est un jeu de données de plus de 14 millions d'images appartenant à 1000 classes. C'était l'un des fameux modèles soumis à la conférence ILSVRC-2014. Ses résultats sont beaucoup plus élevés par rapport à ceux obtenus par l'architecture AlexNet. Son architecture est montrée dans la figure 2.14 suivante.

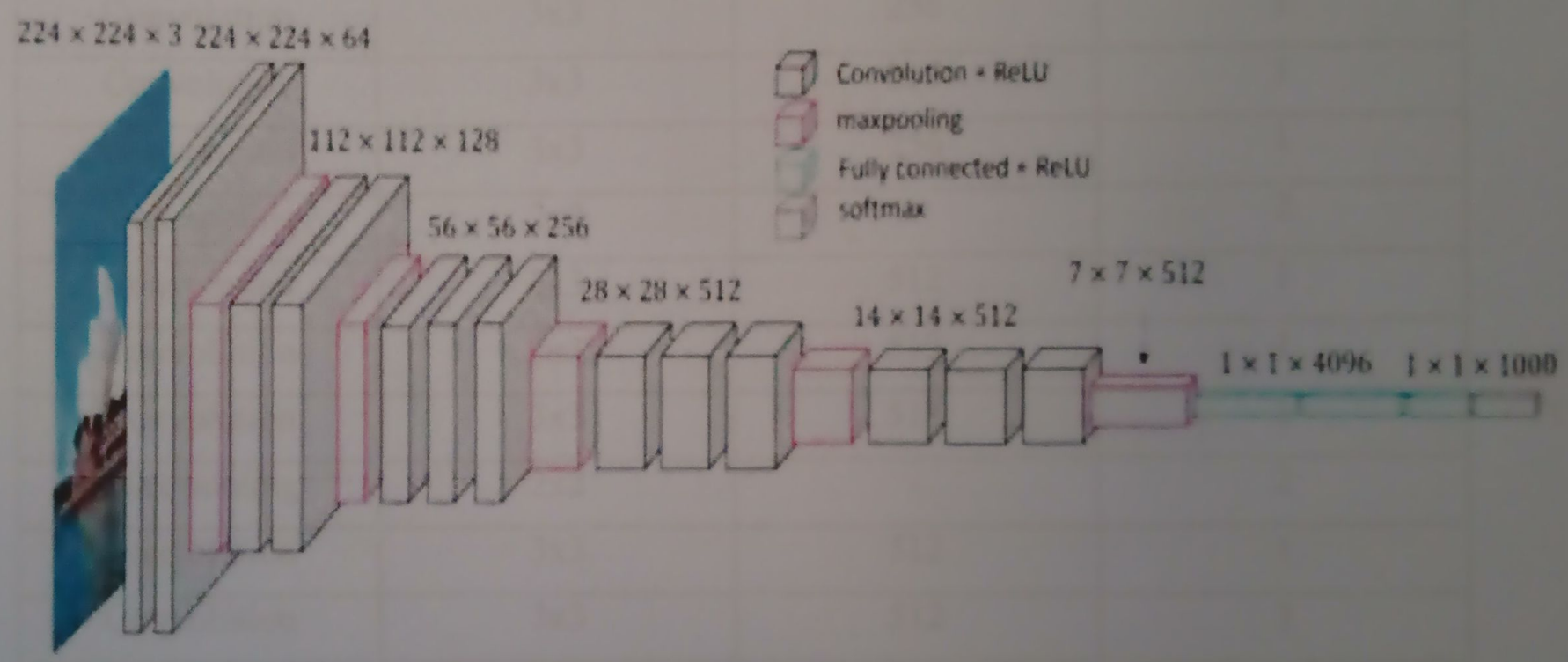


Figure 2-14 : Architecture du réseau VGG16. [36]

L'entrée du réseau est une image couleur (RGB) de taille 224x224 pixels à laquelle on lui applique un ensemble de couches de convolution, de sous-échantillonnage et des couches entièrement connectées. Le tableau 2.1 détaille le type des couches, la taille des filtres et le pas utilisés dans l'architecture VGG16.

Tableau 2-1 : Architecture VGG16 [34].

Type de couche	Taille du filtre	Nombre de filtre	pas
Convolution	3x3	64	1
Convolution	3x3	64	1
Max pooling	2x2	-	2
Convolution	3x3	128	1
Convolution	3x3	128	1
Max pooling	2x2	-	2
Convolution	3x3	256	1
Convolution	3x3	256	1
Convolution	3x3	256	1
Max pooling	2x2	-	2
Convolution	3x3	512	1
Convolution	3x3	512	1
Convolution	3x3	512	1
Max pooling	2x2	-	2
Convolution	3x3	512	1
Convolution	3x3	512	1
Convolution	3x3	512	1
Max pooling	2x2	-	2
Full-connection	4096	-	-
Full-connection	4096	-	-

- ***GoogleNet***

GoogleNet appelé aussi Inception Module, il contient 22 couches, ces couches ne sont pas toutes séquentielles comme dans les architectures précédentes. Il comporte des couches qui se traitent en parallèle. Son architecture est montrée sur la figure 2.15.

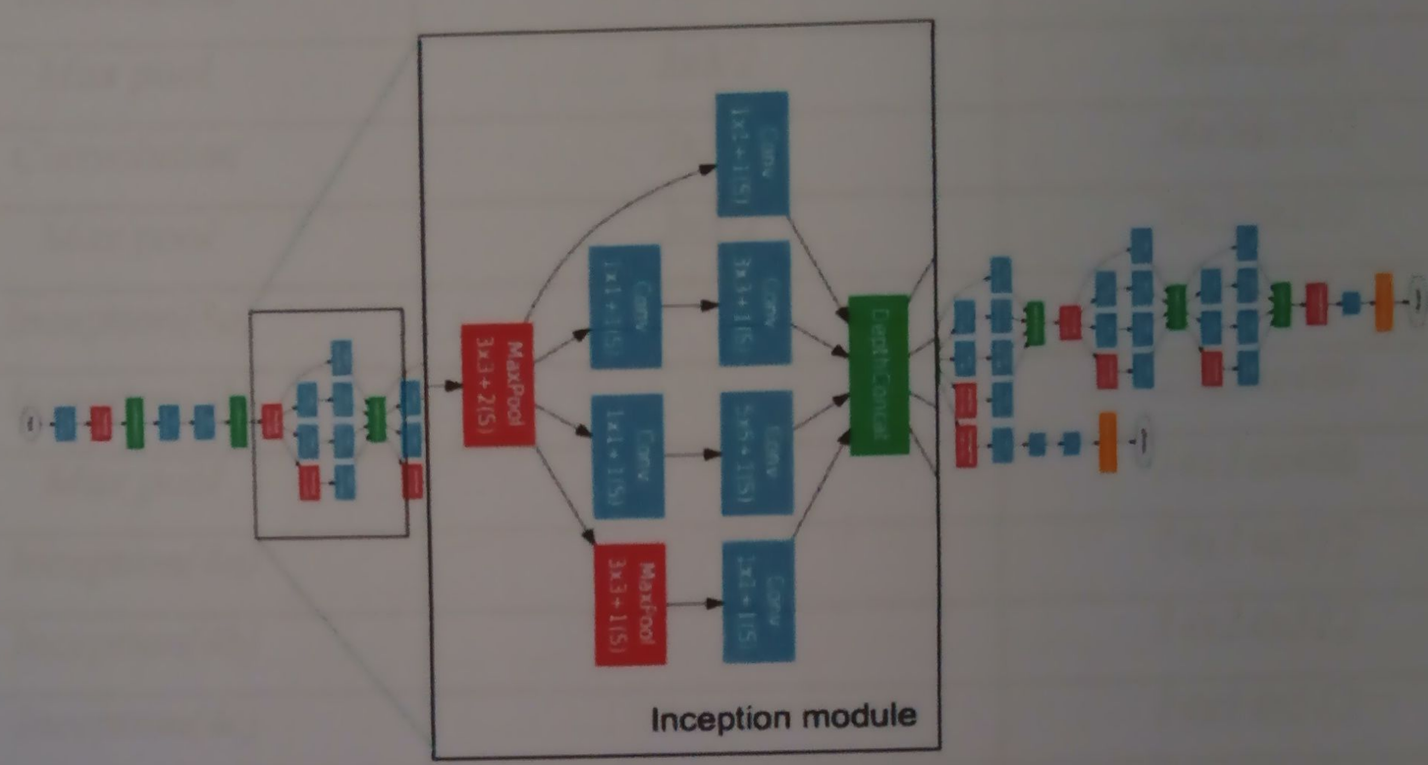


Figure 2-15 : Architecture du réseau Google net [36]

La partie encadré s'appelle « inception module », la figure 2.16 l'énonce avec plus précision.

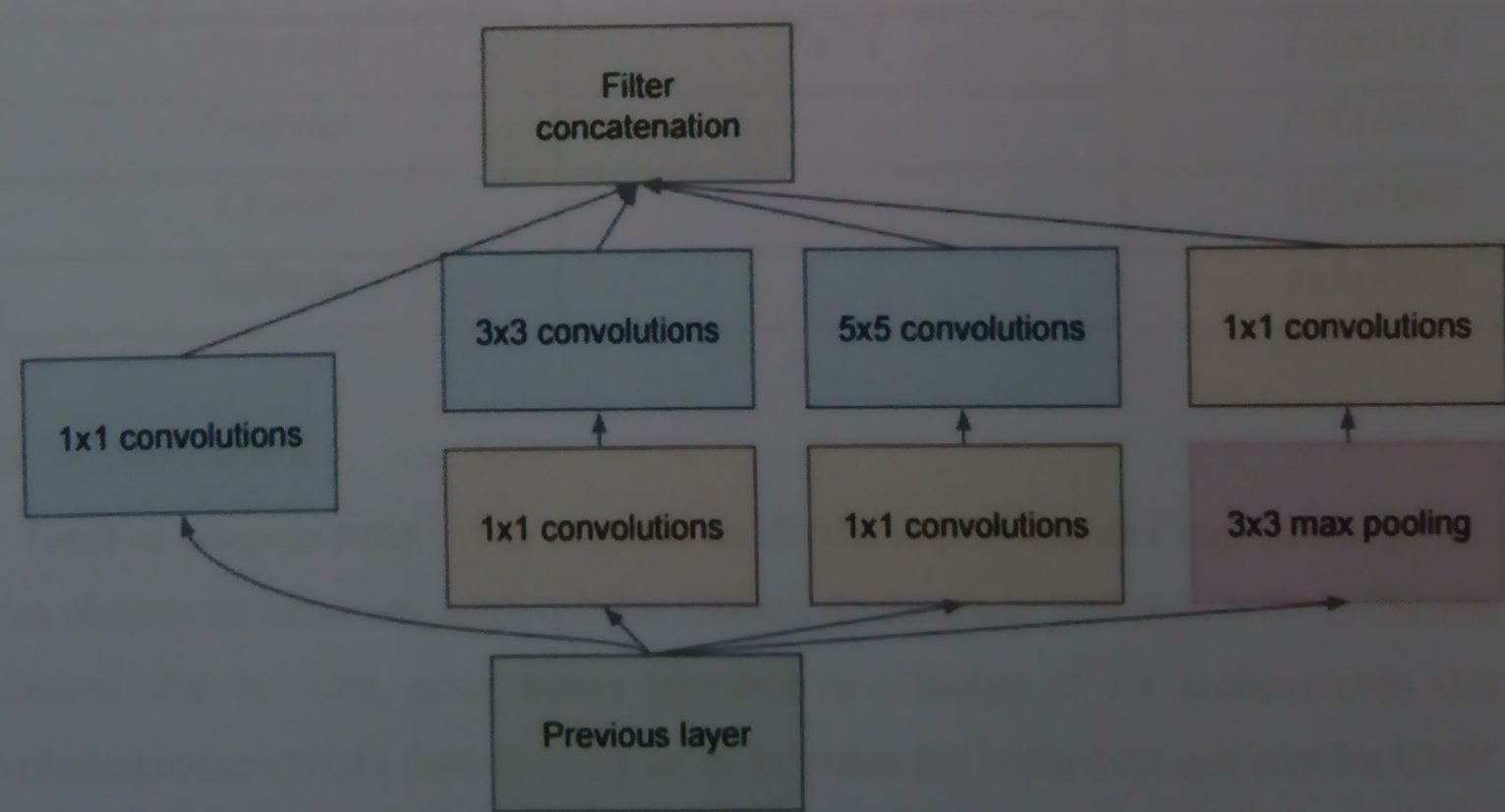


Figure 2-16 : Architecture d'un module 'Inception'. [36]

Le tableau 2.2 suivant représente l'architecture Inception 'Google Net'.

Tableau 2-2 : Architecture Inception 'Google Net'.

Type	*taille/ pas*	*taille de l'output*
convolution	*7x7/2*	*112x112x64*
Max pool	*3x3/2*	*56x56x64*
Convolution	*3x3/1*	*56x56x192*
Max pool	*3x3/2*	*28x28x192*
Inception(3a)		*28x28x256*
Inception(3b)		*28x28x480*
Max pool		*14x14x480*
Inception(4a)		*14x14x512*
Inception(4b)		*14x14x512*
Inception(4c)		*14x14x512*
Inception(4d)		*14x14x528*
Inception(4e)		*14x14x832*
Max pool	*3x3/2*	*7x7x832*
Inception(5a)		*7x7x832*
Inception(5b)		*7x7x1024*
Avg pool	*7x7/1*	*1x1x1024*
Dropout		*1x1x1024*
Linear		*1x1x1000*
Softmax		*1x1x1000*

2.5 Conclusion

Dans ce chapitre nous avons présenté en détails le principe de l'apprentissage automatique et les réseaux de neurones, ainsi que les réseaux de neurones profonds et les différents modèles existants. Par la suite, nous avons introduit le principe et les architectures des réseaux révolutionnaires utilisés dans le domaine de la vision par ordinateur qui sont les CNN. Dans le chapitre suivant, nous aborderons l'architecture et la conception du réseau YOLO que nous avons exploité pour développer notre application de détection et de classification d'objets en temps réel.

Chapitre 3 : Conception de l'approche utilisée

3.1 Introduction

Ce chapitre vise à expliquer plus en détail le modèle YOLO et les traitements utilisés pour détecter et localiser les objets dans la scène.

3.2 Fonctionnement de YOLO (You Only Look Once)

Le modèle YOLO (You Only Look Once a été créé en 2015 par Joseph Redmon et Ali Farhadi et a subi de nombreuses évolutions afin d'arriver à sa version 3 en 2018 [37]. Les premières couches du modèle s'inspirent du Googlenet. Joseph Redmon a été embauché par Google pour améliorer son modèle. YOLO a surpassé toutes les autres techniques à cette époque. La tâche de détection d'objet consiste à déterminer l'emplacement sur l'image où certains objets sont présents, ainsi qu'à les classifier. L'architecture de YOLO se base sur les CNN décrits dans le chapitre précédent. Le modèle applique un seul réseau de neurones sur l'image entière

Le fonctionnement de YOLO de base qui est YOLO version 1 est comme suit : L'image en entrée est divisée en une grille de cellules (SxS). Pour chaque objet présent sur l'image, une cellule de la grille est dite « responsable » de sa prédiction. C'est la cellule dans laquelle le centre de l'objet se situe. Chaque cellule de la grille prédit les boîtes englobantes (B) ainsi que les probabilités de classe (C).

La couche qui effectue les calculs finaux est la couche entièrement connectée "Fully Connected". Elle effectue une somme pondérée de tous les éléments de la matrice d'entrée pour obtenir un seul nombre en résultat. Les pondérations sont des paramètres à entraîner. Chaque cellule de la grille consiste en $(5 + C)$ entièrement connectée avec c le nombre de classes que le modèle est entraîné à détecter.

Supposons qu'il existe trois classes pour la classification : personnes, voitures et bicyclettes les composants du vecteur sont [37] [38] [39]:

Pc : la probabilité qu'il y ait un objet dans la grille

x, y : les cordonnées x et y représentent le centre de la boîte par rapport à l'emplacement de la cellule de la grille. Ces coordonnées sont normalisées entre 0 et 1.

h, w : la hauteur et la largeur du cadre de sélection de l'objet (les boites). Ces coordonnées sont également normalisées entre 0 et 1.

Les paramètres c_1, c_2, c_3 : représentent les classes.

Le vecteur de sortie sera comme suit :

Pc	bx	by	bh	bw	c_1	c_2	c_3

En sortie de la dernière couche un algorithme de décision est appliqué : Si la cellule ne contient aucun objet, son vecteur est nul. La figure 3.1 montre un exemple.

Figure 3-1 : Un exemple de détection avec YOLO [38].

YOLO peut effectuer des détections en double pour le même objet. Pour résoudre ce problème, YOLO applique une suppression non maximale (*NMS*) afin de filtrer les boxes dupliqués avec une faible confiance. La confiance étant la multiplication de la suppression non maximale augmentée par un pourcentage de 2% à 3% dans sa précision.

L'algorithme de la suppression non max (NMS)[39]

Début

Définir Seuil de confiance (par exemple 0.5)

Supprimer toutes les boites avec un score < Seuil de confiance

POUR le Nombre de classes **FAIRE**

TANTQUE des prédictions ne sont pas triés **FAIRE**

Garder la boite avec la plus grande confiance

Supprimer toute les boîtes dont l'IoU (algorithme détaille en bas) est supérieure au seuil défini avec la boîte sélectionnée de l'étape ci-dessus

Répétez l'étape 2 jusqu'à ce que toutes les boites soient utilisées comme prédiction de sortie ou supprimées

FIN

3.2.1 Principe de l'intersection sur l'union (IoU) [40]

L'intersection sur l'union (*IoU*) est une fonction qui quantifie à quel point le cadre de sélection prédit est correctement positionné par rapport au cadre de l'objet. Elle permet de définir l'intersection d'une proposition tout en sanctionnant les débordements.

Dans la figure 3.2, la boîte rouge est la boîte englobante réelle et la boîte bleue est la boîte prédite. Le processus *IoU* calcule la zone de l'intersection par rapport à l'union de ces deux boites, avec l'équation 3.1 suivante :

$$IoU = \text{L'air de l'intersection / l'air de l'union} \qquad (3.1)$$

Si la valeur *IoU* est supérieure à un seuil (choisis selon le problème), la prédiction est considérée suffisamment bonne. Quelques exemples de valeurs sont disponibles dans la figure 3.3.

Figure 3-2 : Illustration IoU [41].

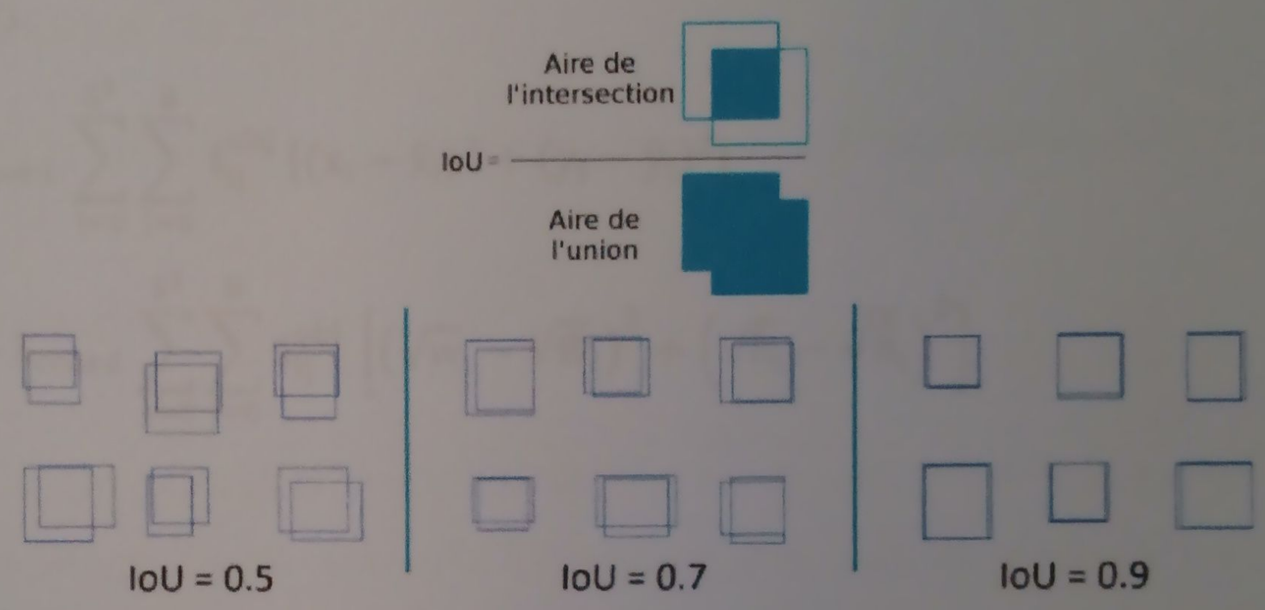


Figure 3-3 : Illustration par des exemples de IoU. [41]

3.2.2 La fonction de perte (loss function) [41]

YOLO utilise une erreur au carré entre les prédictions et la cible pour calculer la perte. La fonction de perte est la somme de la perte de classification, de localisation (erreurs entre la boîte de limite prévue et la boite réelle) et la perte de confiance.

La perte de classification

Si un objet est détecté, la perte de classification (équation 3.2) à chaque cellule est l'erreur quadratique des probabilités conditionnelles de la classe:

$$\text{erreur}_{cla} = \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classe} (p_i(c) - \hat{p}_i(c))^2 \qquad (3.2)$$

où :

$\mathbb{1}_i^{obj}$ est égal 1 si l'objet appartient à la cellule, 0 sinon.

$\hat{p}_i(c)$ désigne la probabilité de classe conditionnelle pour la classe c dans la cellule i

La perte de localisation

La perte de localisation (équation 3.3) mesure les erreurs dans les emplacements des boites englobantes par rapport aux limites de la boite contenant l'objet réel (la case dont le centre de l'objet appartient).

$$\text{erreur}_{pre} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$
$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] \qquad (3.3)$$

où :

x et y : représentent le centre de la boîte englobante prédite pour l'objet.

$\hat{x}_i$ et $\hat{y}_i$: représentent le centre de la vrai boîte englobante de l'objet.

W et h : représentent les dimensions de la boîte englobante prédite de l'objet

$\hat{h}$ $\hat{w}$: représentent les dimensions de la vrai boîte englobante de l'objet

λ_{coord} : facteur pour augmenter le poids pour la perte dans les coordonnées de la boîte englobante

La perte de confiance

Si un objet est détecté dans la boîte, la formule de la perte de confiance est donnée par l'équation 3.4 :

$$\text{erreur}_{con} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \qquad (3.4)$$

Dans le cas où l'objet n'est pas détecté, la formule deviendra alors (équation 3.5) :

$$\text{erreur}_{\text{con}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2 \tag{3.5}$$

où :

$\hat{C}$ est le score de confiance de la case *j* dans la cellule *i*

$\mathbb{1}_i^{\text{obj}} = 1$ si la $j^{ième}$ boîte englobante dans la cellule *i* est responsable de détecter l'objet, 0 sinon

$\mathbb{1}_{ij}^{\text{noobj}}$: est le complément de $\mathbb{1}_i^{\text{obj}}$

λ_{coord} : représente le facteur pour alourdir la perte lors de la détection de l'arrière-plan

3.2.3 Hyperparamètres

Les hyperparamètres d'un modèle sont les paramètres qui doivent se régler à la main avant le début de l'apprentissage. Par exemple pour les deux couches Yolo du modèle il y a plusieurs hyperparamètres à fixer à savoir : (1) Les "anchors", qui sont les tailles prédéfinies des boites en fonction des annotations du jeu d'entraînement, (2) le nombre de classes (pour savoir le nombre de sorties à donner à la couche) et (3) la valeur du seuil pour le *NMS*.

3.3 Les variantes de YOLO

3.3.1 YOLO version 1 (YOLO V1)

YOLO V1 contient 24 couches de convolution, quatre couches de sous-échantillonnage suivis par deux couches entièrement connectées, la figure 3.4 suivante montre la conception du réseau.

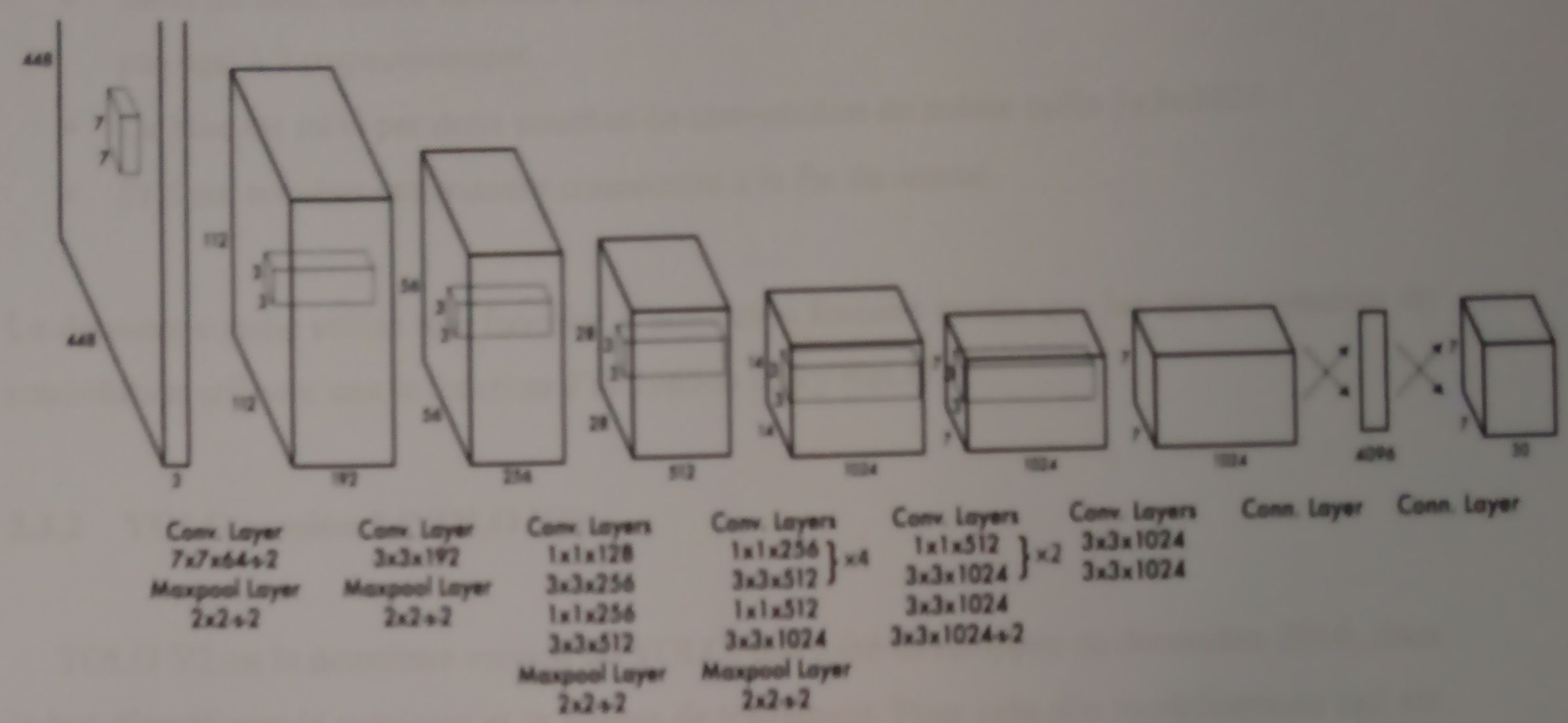


Figure 3-4 : Schéma de l'architecture de YOLO V1 [42].

L'entrée est une image de taille 448x448x3 et la sortie est un vecteur de taille 7x7x30.

Couches du modèle : convolution, max pooling, de sous-séchantillonage

- La première couche est une couche de convolution de taille 7x7x64, elle applique la convolution 64 fois avec un pas (stride) égal à 2 suivie d'une couche de max polling (sous-échantillonnage) da taille 2x2 et un pas égal à 2
- La deuxième couche de convolution est de taille 3x3x192, elle est suivie par un max pooling de taille 2x2 avec un pas égal à 2
- Les quatre couches suivantes sont des couches de convolution de taille 1x1x128, 3x3x256, 1x1x256, 3x3x512 respectivement suivies par un sous-échantillonnage de taille 2x2 avec un pas égal à 2
- Ensuite, On trouve deux couches de convolution qui se suivent de taille 1x1x256 et 3x3x512 respectivement. Cette paire de couches se répète quatre fois.
- Ce bloc de couche de convolution est suivi pas un max pooling de taille 2x2 avec un pas de 2
- Deux couches de convolution de taille 1x1x512 et 3x3x1024 qui se répètent deux fois

- Suivi de deux autres couches de convolution de taille 3x3x1024 et 3x3x1024 avec un pas égal à 2 respectivement.
- Ce bloc est suivi par deux couches de convolution de même taille 3x3x1024.
- Et deux couches entièrement connectées à la fin du réseau.

La dernière couche utilise une fonction d'activation linéaire tandis que les autres couches de convolution utilisent une la fonction d'activation leaky ReLU.

3.3.2 YOLO version 2 (YOLO V2)

YOLO V2 est la deuxième version du YOLO, elle a été développée en décembre 2016, dans le but d'améliorer la précision et la vitesse de traitement. Pour cela des modifications ont été apportées à la version V1 [51] pour aboutir à un **modèle** plus léger pour rendre la détection encore plus rapide, YOLOv2 emploie le framework plus léger DarkNet-19 qui comporte 19 couches de convolution et 5 couches de max-pooling (tableau 3.1). Les colonnes filtres et Taille/Pas sont les paramètres utilisés pour chaque couche.

Tableau 3-1 : Architecture de YOLO V2 [42]

Type	Filtres	Taille/Pas	Sortie
Convolutional	32	3*3	224*224
Maxpool		2*2 /2	112*112
Convolutional	64	3*3	112*112
Maxpool		2*2/2	56*56
Convolutional	128	3*3	56*56
Convolutional	64	1*1	56*56
Convolutional	128	3*3	56*56
Maxpool		2*2 /2	28*28
Convolutional	256	3*3	28*28
Convolutional	128	1*1	28*28
Convolutional	256	3*3	28*28
Maxpool		2*2 /2	14*14
Convolutional	512	3*3	14*14
Convolutional	256	1*1	14*14
Convolutional	512	3*3	14*14
Convolutional	256	1*1	14*14
Convolutional	512	3*3	14*14
Maxpool		2*2/2	7*7
Convolutional	1024	3*3	7*7
Convolutional	512	1*1	7*7
Convolutional	1024	3*3	7*7
Convolutional	512	1*1	7*7
Convolutional	1024	3*3	7*7
Convolutional	1000	1*1	7*7
Avgpool		Global	1000
Softmax			

- **Batch normalisation** : C'est un processus nouveau dans cette version de YOLO, c'est un processus qui s'effectue pour chaque couche, il prend continuellement la sortie et la normalise avant de l'envoyer à la couche suivante.

- **Les boites d'encrages [42]:** L'idée de la boîte d'ancrage est d'ajouter une "dimension" supplémentaire aux étiquettes de sortie en définissant un certain nombre de boîtes d'ancrage. Cela permettra d'affecter un objet à chaque boîte d'ancrage. Chaque cellule de la grille possède deux boîtes d'ancrage donc peut prédire jusqu'à 2 objets.

3.3.3 YOLO version 3 (YOLO V3)

L'architecture de YOLO V3 est une version améliorée de YOLO V2 qui intègre des blocs résiduels, des sauts de couches et un processus de sur- échantillonnage. YOLO V3 emploie le framework de DarkNet-53 qui comporte 53 couches de convolution qui conduit à une architecture composée de 106 couches. La figure 3.5 illustre l'architecture de YOLO V3.:

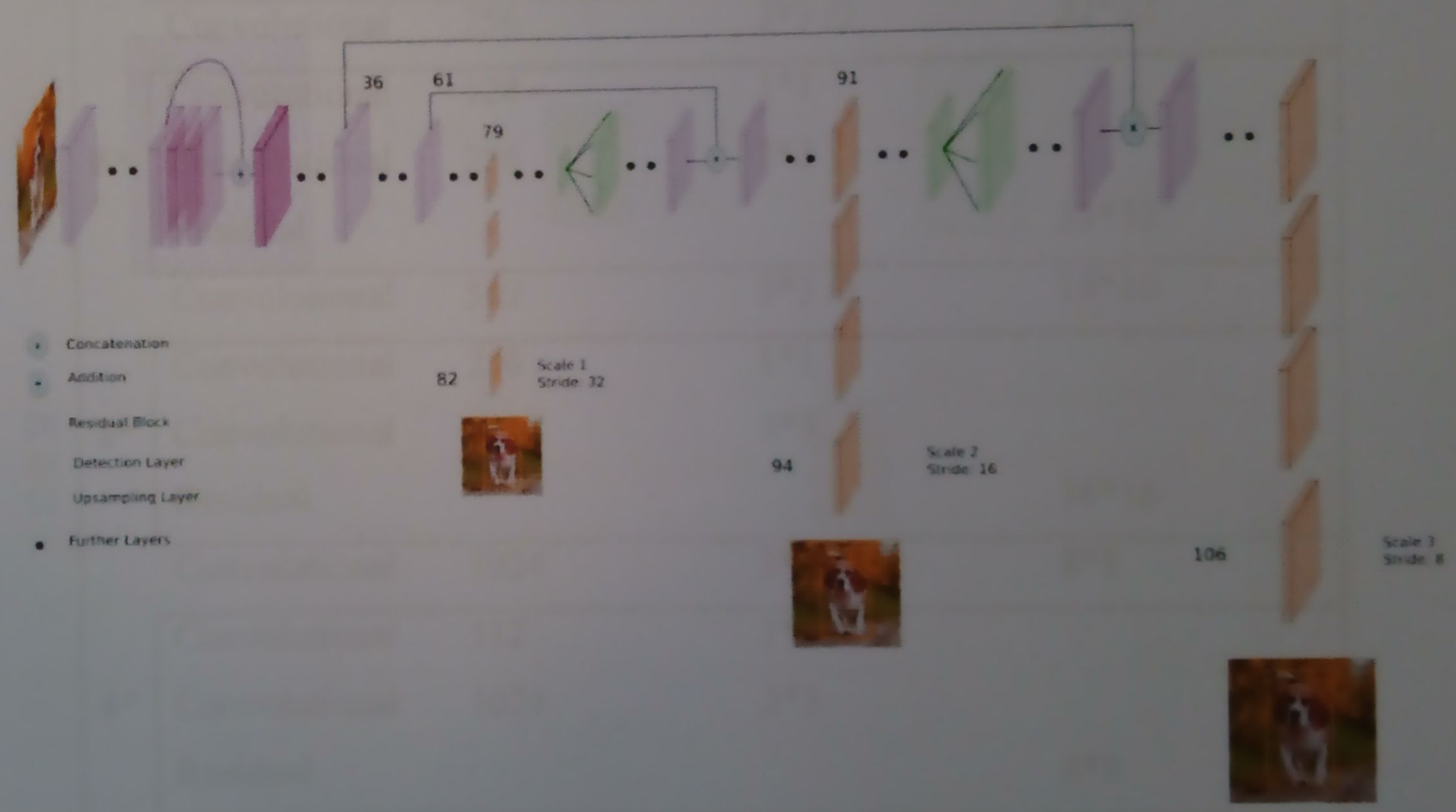


Figure 3-4 : Illustration de l'architecture de YOLO V3 [43].

Le format d'entrée et de sortie de chaque couche est donné dans le tableau expliquant l'architecture du modèle (tableau 3.2). Les colonnes filtres et Taille/Pas sont les paramètres utilisés pour chaque couche.

Tableau 3-2 : Architecture de YOLO V3 [43]

	Type	Filtres	Taille/Pas	Sortie
	Convolutional	32	3*3	256*256
	Convolutional	64	3*3/2	128*128
1*	Convolutional	32	1*1	
	Convolutional	64	3*3	
	Residual			128*128
	Convolutional	128	3*3 /2	64*64
2*	Convolutional	64	1*1	
	Convolutional	128	3*3	
	Residual			64*64
	Convolutional	256	3*3 /2	32*32
8*	Convolutional	128	1*1	
	Convolutional	256	3*3	
	Residual			32*32
	Convolutional	512	3*3 /2	16*16
8*	Convolutional	256	1*1	
	Convolutional	512	3*3	
	Residual			16*16
	Convolutional	1024	3*3 /2	8*8
4*	Convolutional	512	1*1	
	Convolutional	1024	3*3	
	Residual			8*8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Les améliorations apportées par YOLO V3 sont [43]:

- **Détection multi échelles** : YOLO V3 permet de détecter des objets à trois échelles différentes (figure 3.5).
- **Intégration des couches convolutives de taille 1x1** : elles permettent de détecter les objets dans trois endroits différents du réseau. En effet, YOLO V3 effectue la prédiction

à trois échelles, qui sont précisément obtenues en sous-échantillonnant les dimensions de l'image d'entrée de 32, 16 et 8 respectivement. La première détection est faite par la 82ième couche, la deuxième par la 94ième couche et la troisième par la dernière couche (106ième couche). Les détections à différentes couches permettent de résoudre le problème de la détection des petits objets.

- La forme du noyau de détection est de 1 x 1 x (B x ($5 + C$)). Où B représente le nombre de cadres de sélection qu'une cellule de la carte de caractéristiques peut prédire, le nombre 5 correspond aux quatre attributs du cadre de sélection et à la confiance d'un objet, et le paramètres C désigne le nombre de classes.

- **Intégration des couches sur-échantillonnées** : ces dernières sont concaténées avec les couches précédentes pour la détection des petits objets.
- **Plus de cadres de sélection par image** : Pour une image en entrée de même taille, YOLO V3 prédit plus de cadres de sélection que YOLO V2. Par exemple, avec une résolution de 416 x 416, YOLO V2 prédit 845 (13 x 13 x 5) cases. À chaque cellule de la grille, 5 boîtes sont détectées à l'aide de 5 boites d'ancrage. YOLO V3 prédit des boîtes à 3 échelles différentes. Pour la même image de 416 x 416, le nombre de cases prédites est 10 647. Cela signifie que YOLO V3 prédit 10 fois le nombre de boîtes prédites par rapport à YOLO V2. À chaque échelle, chaque grille peut prédire 3 boîtes en utilisant 3 ancres. Comme il y a trois échelles, le nombre total de boîtes d'ancrage utilisé est de 9, 3 pour chaque échelle.
- Le temps : comme nous avons cité au paragraphe précédent, YOLO V3 prédit 10 fois le nombre de boîtes prédites par YOLO V2, et c'est la raison pour laquelle il est plus lent que YOLO V2.
- YOLO V3 effectue une classification multilabel pour les objets détectés dans les images. Auparavant, dans les versions précédentes de YOLO V3, les auteurs avaient l'habitude de trier les scores de la classe et de prendre la classe avec le score maximum comme étant la classe de l'objet contenu dans le cadre de sélection. Cela a été modifié dans YOLO V3. Les classes de softmaxing reposent sur l'hypothèse que les classes s'excluent mutuellement, ou en termes simples, si un objet appartient à une classe, il ne peut pas appartenir à l'autre. Cependant, lorsque nous avons des classes telles que personne et homme dans un jeu de données, l'hypothèse ci-dessus échoue. C'est la raison pour laquelle les auteurs de YOLO se sont abstenus de faire du soft-max sur les classes.

Au lieu de cela, chaque score de classe est prédit à l'aide d'une régression logistique et un seuil est utilisé pour prévoir plusieurs libellés pour un objet. Les classes dont les scores sont supérieurs à ce seuil sont attribuées à la boîte.

- YOLO V3 ajoute des connexions entre deux couches de prédiction. Le modèle commence par échantillonner les cartes de caractéristiques, puis les fusionne par concaténation avec les caractéristiques précédentes. La combinaison d'informations plus fines facilite la détection des petits objets.

3.4 L'application de détection d'objets développée basée sur YOLO V3

Ce paragraphe vise à expliquer les différentes étapes de développement de l'application de détection d'objets. Cinq classes (objets) sont définies dans notre application à savoir : personnes, voitures, camions, bicyclettes et panneaux de signalisation. Le schéma bloc de l'application est illustré sur la figure 3.6.

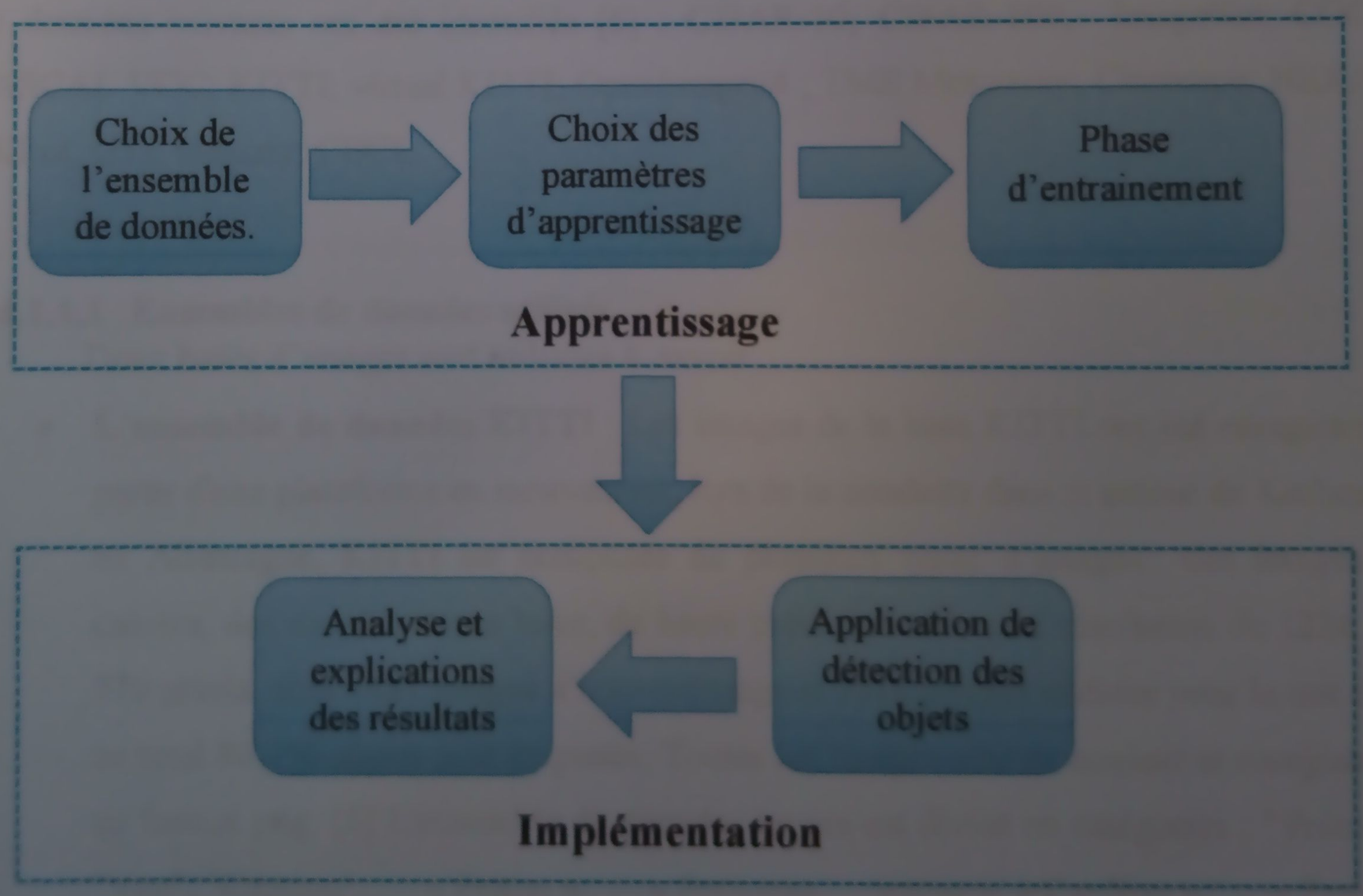


Figure 3-5 : Schéma blocs de l'application de détection d'objets développée

3.4.1 Données pour l'entrainement du modèle

Tous les modèles d'apprentissage profond nécessitent une quantité énorme de données pour s'entraîner et d'apprendre des objets afin de prédire leurs classes correspondantes. Chaque image doit être correctement annotée pour que les modèles puissent s'entraîner. à détecter les objets présents dans l'environnement. Il est certes relativement facile d'effectuer un enregistrement vidéo utile de plusieurs heures, mais annoter chaque image de cette vidéo est un processus extrêmement long. En effet, selon le nombre des images dans la base de données et les différents objets contenus dans chaque image, l'étiquetage des objets nécessite au maximum quinze minutes. Ce traitement implique un temps important pour constituer une base de données étiquetée. Il existe de nombreuses bases de données dans le domaine public mises à la disposition des développeurs. Chacune d'elles contient différents objets annotés différemment selon l'objectif visé.

Pour répondre à notre problématique, il était nécessaire de trouver une bases de données contenant des classes d'objets de la route. À la suite de nombreuses recherches les ensembles de données suivants ont été identifiés [6] : CIFAR-10; CIFAR-100; ImageNet; COCO; PASCAL VOC; KITTI; virtual KITTI; OpenImagev4 ; TME Motorway; Cityscape; NEXAR; PKLot; GTI; Udacity; CBCL.

3.4.1.1.1 Ensembles de données utilisés

Deux bases d'images sont utilisées à savoir :

- **L'ensemble de données KITTI** : Les images de la base KITTI ont été enregistrée à partir d'une plateforme en mouvement, lors de la conduite dans et autour de Karlsruhe, en Allemagne. KITTI est composée de plusieurs types d'images : des images de caméra, des numérisations laser, de haute précision avec une résolution de 1224 par 370 pixels, dont 7481 images d'apprentissage et 7518 images dédiées pour le test dont au total 80 256 objets sont étiquetés. Toutes les images sont en couleur et enregistrées au format png. [8] L'ensemble de données brutes est divisé en catégories : "Voiture", "Van", "Camion", « Piéton », « Personne assise », « Cycliste », « Tram », « Divers », « classe ignorée ». La taille totale des données fournies est de 12 Go.
- **L'ensemble de données OpenImagesv4 :** OpenImagesv4 est un ensemble de données d'environ 9 millions d'images avec des objets étiquetés, comprenant environ 600 classes d'objets différents. Dans l'ensemble, le jeu de données est supérieur à 500 Go, Les

classes d'objet choisis sont : « Personne », « Voiture », « Camion », « Bicyclette », « Panneaux de signalisation » [7].

3.4.1.1.2 Préparation de données

Contrairement aux données de OpenImagesv4, les images de KITTI ne sont pas immédiatement compatibles avec le format du framework Darknet (la plateforme d'apprentissage). Par conséquent, un traitement est effectué pour extraire les informations pertinentes à des fins de reconversion.

Processus de reformatage de l'ensemble dee images de KITTI

Darknet nécessite un fichier texte (avec l'extension .txt) correspondant à chaque image. Dans le fichier texte, il doit y avoir une ligne pour chaque objet à détecter dans l'image au format suivant:

<id-classe-d'objet> < centre-x > < centre-y > <largeur> <hauteur>

- Le premier champ **id-classe d'objet** est un entier représentant la classe de l'objet. Il varie de 0 à (nombre de classes - 1).
- Les deuxième et troisième champs **centre-x** et **centre-y** sont les coordonnées x et y du centre du cadre de sélection respectivement. Ces paramètres sont normalisés en divisant par la largeur et la hauteur de l'image respectivement.
- Les quatrième et cinquième paramètres **largeur** et **hauteur** sont la largeur et la hauteur du cadre de sélection à nouveau normalisés.

Les quatre paramètres cités ci-dessus ont des valeurs en virgule flottante comprises entre 0 et 1.

Répartition des données d'apprentissage et de validation

Les images de la base sont scindées d'une **manière aléatoire** en deux sous-ensembles :

- **Ensemble d'entrainement** : sous-ensemble destiné à l'apprentissage du modèle ;
- **Ensemble de test** : sous-ensemble destiné à la validation du modèle.

Cette procédure est effectuée après l'entrainement afin de tester le modèle sur des images qui n'ont pas été employées au cours de l'entrainement afin d'éviter le risque de sur-apprentissage.

Selon la quantité de données dont nous disposons, nous avons sélectionné 80% des données pour l'entraînement et 20% des données pour la validation. Aucune image ne doit faire partie à la fois de l'apprentissage et de l'ensemble de test.

3.4.2 Paramètres d'apprentissage

L'algorithme de détection comprend une dizaine (seize dans notre cas) de paramètres. Nous présentons dans le tableau 3.3 et la figure 3.4 les paramètres sélectionnés pour chaque ensemble de données utilisé pour l'apprentissage :

Tableau 3-3 : Paramètres de l'apprentissage pour l'ensemble de données OpenImagesv4

Paramètre	Valeur
batch	64
subdivisions	16
width	416
height	416
channels	3
momentum	0.9
decay	0.0005
angle	0
saturation	1.5
exposure	1.5
hue=.1	.1
learning_rate	0.001
burn_in	400
max_batches	5200
steps	3800
scales	.1

Tableau 3-4 : Paramètres de l'apprentissage pour l'ensemble de données KITTI.

Paramètre	Valeur
batch	64
subdivisions	16
width	1152
height	352
channels	3
momentum	0.9
decay	0.0005
angle	0
saturation	1.5
exposure	1.5
hue=.1	.1
learning_rate	0.001
burn_in	1000
max_batches	500200
steps	40000,50000
scales	1

3.4.2.1 Les hyperparamètres taille du lot et subdivision

La valeur du lot ou paramètre « batch » indique la taille du lot utilisée lors de l'entrainement. Notre ensemble d'entrainement contient quelques centaines d'images. Le processus d'apprentissage implique la mise à jour itérative des poids du réseau de neurones en fonction du nombre d'erreurs accomplit sur le jeu de données d'apprentissage. Il est peu pratique (et inutile) d'utiliser toutes les images du jeu d'apprentissage en même temps pour mettre à jour les poids. Ainsi, un petit sous ensemble d'images est utilisé dans une itération, et ce sous ensemble s'appelle « batch ». Lorsque la taille du lot est définie sur 64, cela signifie que 64 images sont utilisées dans une itération pour mettre à jour les paramètres du réseau de neurones. La taille est limitée par les ressources matérielles disponibles, l'impact de cette valeur affecte beaucoup plus la puissance et le temps de calcul que la performance.

Il est possible le processeur graphique (GPU) ne dispose pas suffisamment de mémoire pour utiliser une taille de lot de 64. Pour ce faire, la variable subdivisions permet de traiter une fraction de la taille du lot à la fois sur le GPU.
Le lot complet ou l'itération ne sera complète qu'après le traitement de toutes les 64 images

Les paramètres hauteur, largeur et canaux

- Ces paramètres de configuration spécifient la taille de l'image en entrée et le nombre de canaux de couleur (channels en anglais).
- Les images d'entraînement sont d'abord redimensionnées à la **largeur (width)** x la **hauteur (heigh)** avant l'entraînement. Nous avons utilisé les valeurs par défaut de 416x416 pour les images de l'ensemble de données OpenImgev4 et 1152x352 pour les images de l'ensemble de données KITTI.
- **Canaux** : ce paramètre est égal à 3 ce qui indique que nous traiterons des images en RVB.

Les hyperparamètres élan et décomposition

Dans ce qui précède, nous avons indiqué que les pondérations d'un réseau de neurones sont mises à jour en fonction d'un petit lot d'images et non de l'ensemble de données entier. Pour cette raison, les poids sont peu fluctués. Le paramètre élan (momentum) est utilisé pour pénaliser les variations importantes de poids entre les itérations.

Un réseau de neurones typique compte des millions de poids et peut donc facilement sur ajuster les données d'entraînement. Une sur-alimentation signifie simplement que les données d'entraînement seront très efficaces et que les données de test seront médiocres. C'est presque comme si le réseau de neurones avait mémorisé la réponse à toutes les images dans l'ensemble d'entrainement, mais n'avait pas vraiment appris le concept sous-jacent. L'un des moyens d'atténuer ce problème consiste à pénaliser une valeur élevée pour les poids. Le paramètre décomposition (decay en anglais) contrôle ce terme de pénalité.

Les hyperparamètres Taux d'apprentissage, pas, échelles, réchauffement

Le processus d'entrainement, commence avec zéro information, le taux d'apprentissage (learning rate) doit donc être élevé. Mais comme le réseau de neurones reçoit beaucoup de données, les pondérations doivent changer de manière moins agressive. En d'autres termes, le taux d'apprentissage doit être réduit au fil du temps. Dans la figure 3.9, le taux d'apprentissage commence à 0,001 et reste constant pendant 3 800 itérations, puis multiplié par des échelles (scales) pour obtenir le nouveau taux d'apprentissage.

Dans le paragraphe précédent, nous avons mentionné que le taux d'apprentissage doit être élevé au début de l'apprentissage et faible par la suite. Bien que cette affirmation soit en grande partie vraie, il a été empiriquement établi que la vitesse d'entrainement a tendance à augmenter si nous avons un taux d'apprentissage plus bas pendant une courte période au début. Ceci est contrôlé par le paramètre réchauffement (burn_in). Cette période est appelée période de réchauffement.

L'hyperparamètre nombre d'itérations

Lorsque l'algorithme parcourt tous les exemples d'apprentissage, on dit qu'il a exécuté une itération. Pour le détecteur d'objets multi-classes, le nombre « max_batches » est conseillé qu'il soit égal au moins à 2000 fois le nombre de classes.

Les paramètres d'augmentation de données

Le processus d'augmentation de données permet d'augmenter le nombre d'images dans la base de données. Il consiste à traiter les images de la base et à les ajouter aux images existantes. Comme traitements effectués, on trouve la transformation de plan de couleur de l'ensemble des images du plan RVB aux plans HSL (hue, saturation, lightness) and HSV (hue, saturation), la rotation, le redimensionnement, etc…

3.5 Conclusion

Ce chapitre se compose de deux parties, dans la première nous avons présenté le modèle YOLO et ses différentes versions et architectures. Plus particulièrement, nous avons détaillé la version 3 du modèle qui est utilisée dans notre application. Dans la deuxième partie nous avons détaillé les paramètres de configuration et les différents paramètres nécessaires pour la phase d'apprentissage du modèle YOLO V3 afin de l'intégrer dans la plateforme de détection d'objet réalisée. Dans le chapitre suivant nous allons présenter notre application, et les résultats expérimentaux du modèle d'apprentissage développé.

Chapitre 4 : Implémentation et résultats

4.1 Introduction :

Nous abordons dans ce chapitre les détails techniques de l'implémentation réalisée en termes de langage de programmation, les plateformes d'apprentissage « DARKNET » et « GOOGLE COLABORATORY ». Nous présentons l'interface de l'application développée et l'analyse des résultats obtenus de l'apprentissage automatique effectué. En fin, nous examinerons plus en détail certains exemples et résultats de détection d'objets.

4.2 Environnements de travail :

4.2.1 Environnement matériel:

La configuration de la machine utilisée est comme suit :

- Processeur: Intel (R) Core (TM) i3-5020M CPU @ 2.40 GHz 2.20GHz
- System d'exploitation : Ubuntu 16.04 64-bit
- RAM: 4.00 GO.
- Disque Dur: 500 GO.

4.2.2 Environnement logiciel

Les langages de programmation employés sont Python version 3.6.8 et OpenCV avec leurs différentes bibliothèques. L'interface graphique est réalisée en utilisant la bibliothèque graphique Tkinter « open source » pour le langage Python, permettant la création d'interfaces graphiques.

Le langage Python :

Le langage de programmation Python a été créé en 1989 par Guido van Rossum, aux Pays-Bas. Le nom Python vient d'un hommage à la série télévisée Monty Python's Flying Circus

dont G. van Rossum est fan. La première version publique de ce langage a été publiée en 1991. Ce langage de programmation présente de nombreuses caractéristiques [41]:

- Il est multiplateforme. Iil fonctionne sur de nombreux systèmes d'exploitation : Windows, Mac OS X, Linux, Android, iOS,
- Il est gratuit.
- C'est un langage de haut niveau. Il demande relativement peu de connaissance sur le fonctionnement d'un ordinateur pour être utilisé.
- C'est un langage interprété. Un script Python n'a pas besoin d'être compilé pour être exécuté, contrairement à des langages comme le C ou le C++.
- Il est orienté objet. Il est possible de concevoir en Python des entités qui miment celles du monde réel (une cellule, une protéine, un atome, etc.) avec un certain nombre de règles de fonctionnement et d'interactions.

4.2.3 La plateforme d'apprentissage profond :

Il existe de nombreux outils pour utiliser des modèles d'apprentissage profond ayant les capacités d'effectuer les opérations matricielles nécessaires aux calculs des réseaux neuronaux, parmi ces outils nous pouvons citer [42] : CAFFE, CHAINER, DARKNET, MATLAB, MXNET, NEON, TENSORFLOW. Chacun d'eux possède ses avantages et inconvénients. Dans notre application nous avons opté pour Darknet qui est un outil de réseau neuronal gratuit écrit en langage C sous licence du domaine public. Darknet est portable, rapide et supporte les calculs de l'unité centrale de traitement CPU (central processing unit) ou le processeur graphique GPU (Graphics Processing Unit). Cet outil peut fonctionner sur plus d'un processeur pour l'exécution en temps réel et propose la création de réseaux à l'aide de simples fichiers de configuration.

- **Construction de Darknet**

Darknet peut être construit avec plusieurs paramètres facultatifs qui sont désactivés par défaut tels que GPU (accelerateur matériel assurant les fonctions de calcul parallèle), CUDNN (bibliothèque de primitives accélérées par GPU pour les architectures d'apprentissage profond), OPENCV (bibliothèque graphique libre, spécialisée dans le traitement d'images en temps réel), etc... Après avoir cloné le dossier localement on lance la commande make pour le compiler. Mais d'abord, pour utiliser la fonctionnalité GPU, nous devons éditer

le **Makefile** dans les deux premières lignes, dans lequel nous lui indiquons de le compiler pour l'utilisation du GPU avec les pilotes CUDA (CUDNN) [43].

4.2.4 La plateforme d'apprentissage Google Colaboratory :

Google Colaboratory, est un service cloud gratuit de Google pour les développeurs d'intelligence artificielle permettant de développer des applications d'apprentissage profond sur le GPU Tesla K80 d'environ 12 Go gratuitement pendant 12 heures. Google colaboratory ne nécessite aucune configuration et s'exécute entièrement dans le cloud [44].

4.3 Interface de l'application dévelopée

L'interface comporte deux pages, la première est la page d'accueil (figure 4.1) contenant deux boutons, un bouton pour accéder à la page principale et un autre pour quitter l'application. La deuxième page (figure 4.2) contient des champs de saisies afin de permettre à un utilisateur de lancer l'application à partir de la Webcam pour effectuer la détection et la classification en temps réel ou bien à partir d'une image ou une séquence vidéo acquissent et stockées en temps différé .

Figure 4-1 : Page d'accueil de l'application développée.

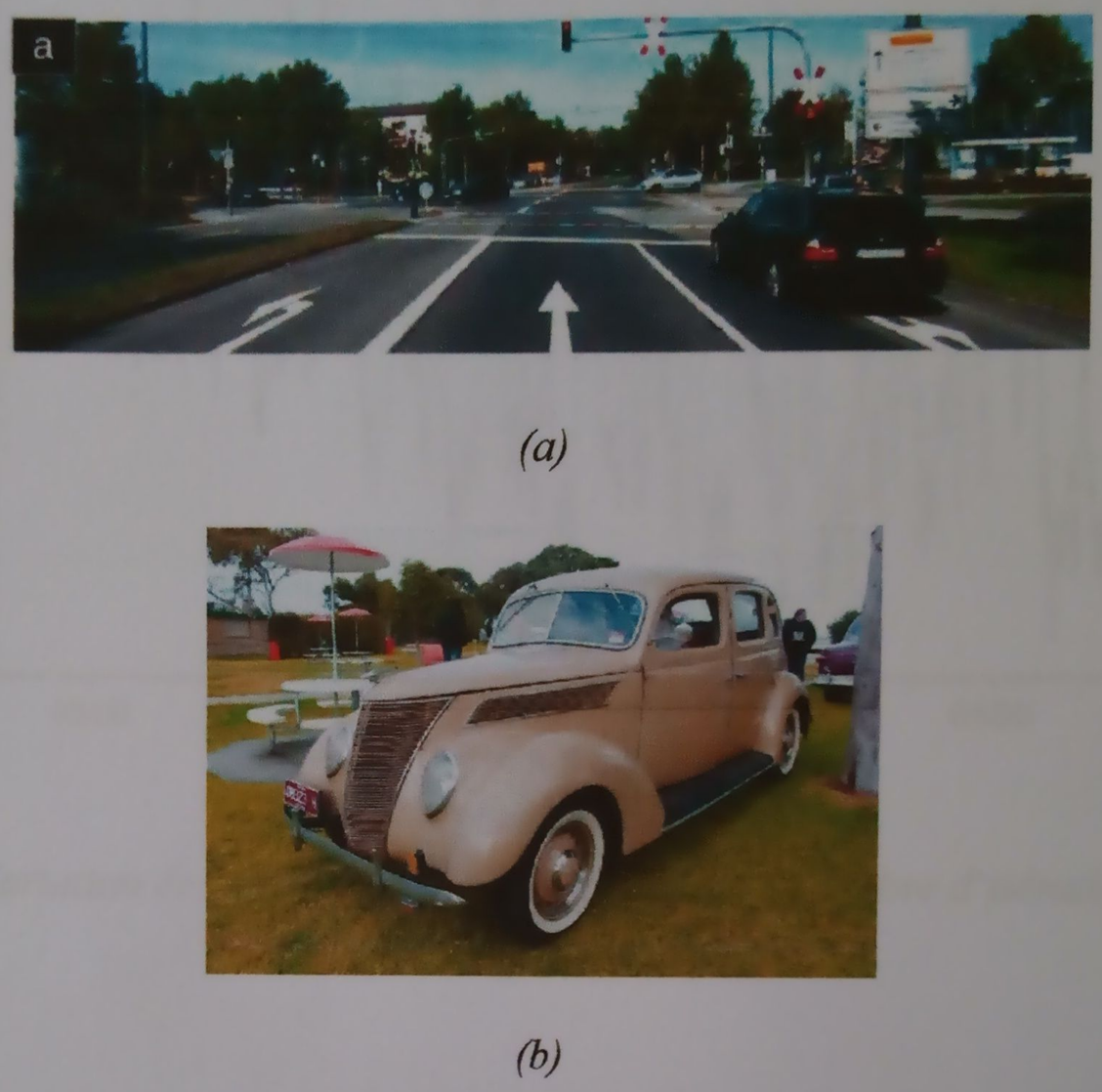
Figure 4-2 : Page de lancement de l'application de détection d'objets et classification.

4.4 Evaluation des tests et analyse des résultats

Dans ce paragraphe, nous évaluons les résultats obtenus de la détection des objets statiques et dynamiques et leur localisation dans une image ou vidéo acquissent en temps différé ou bien dans un flux vidéo acquit en temps réel par une caméra tel que la webcam. L'application est développée en exploitant l'algorithme YOLO V3, approche de détection, classification et localisation d'objets basés sur les CNN. Dans notre application cinq classes sont définies pour la détection et la classification à savoir : voiture, camion, personne, bicyclette et panneau.

Le système ainsi implémenté regroupe deux phases à savoir : la phase offline qui est la phase d'apprentissage et la phase online qui correspond à la phase des tests. Pour la phase d'apprentissage, deux bases d'images ont été employées : la base d'images benchmark KITTI

de 7000 images et la base OpenImage de 1200 images. Les images de ces deux bases comportent différents objets de différentes tailles. Ci-dessous sur la figure 4.3 nous illustrons deux exemples d'images de chacune des deux bases.



(a)

(b)

Figure 4-3 : Exemple d'images : (a) image de KITTI , (b) Image de OpenImage.

Au cours de la phase d'apprentissage, un fichier appelé fichier d'événements (train.log) contenant le nombre d'itérations et l'erreur courante est généré. Ce fichier est stocké et est utilisé pour dessiner les graphes représentant l'évolution changement des valeurs de l'erreur en fonction du nombre d'itérations.

L'utilisation de l'algorithme YOLO V3 nécessite l'installation de l'outil Darknet pour adopter le modèle prédéfini. Darknet encadre les objets détectés dans la scène, illustre les scores de confidence calculés par YOLO V3 et le temps écoulé pour la détection des objets trouvés.

4.4.1 Evaluation des performances de la détection

- **La base d'images KITTI** : Nous avons commencé l'apprentissage par l'ensemble des images KITTI. Le graphe représentant la variation des valeurs de l'erreur moyenne en fonction du nombre d'itérations est présenté dans la figure 4.4 ci-dessous :

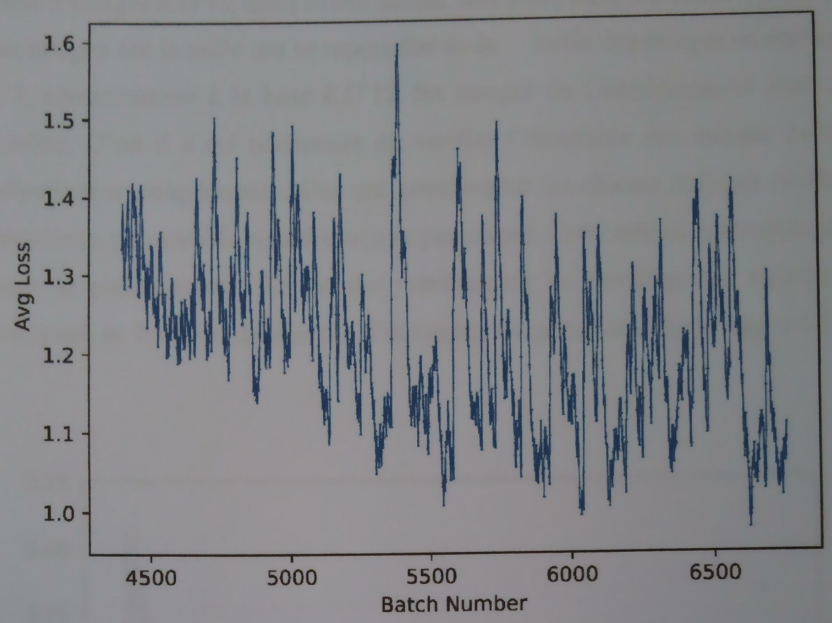


Figure 4-4 : Variation de l'erreur moyenne en fonction du nombre d'itérations en utilisant KITTI.

Nous avons remarqué que YOLO V3 n'aboutit pas à de bonne performance en termes de précision sur l'ensemble de données KITTI contrairement à d'autres bases telles que COCO. Après avoir examiné les propriétés de base de KITTI, nous avons constaté que la taille des images est grande (1224x370) par rapport à celle des images de YOLO V3 (608x608). D'où les images de KITTI nécessitent un redimensionnement qui conduit à la réduction de la taille des boites englobantes (bbox) de l'objet. Cette opération entraîne un taux de détection faible pour les objets de petite taille.

Nous avons remarqué aussi que plus la base de données des images est grande, plus l'erreur moyenne est grande au début de l'apprentissage. Par conséquent, pour atteindre un bon taux de détection, l'opération d'apprentissage réclame un nombre d'itérations assez grand de l'ordre de 18000 au minimum (dans notre cas) donc plusieurs heures voir plusieurs mois de calcul, chose qu'on n'a pas pu faire faute de matériel et de temps.

- **La base d'image Openimagev4 :** Suite à un faible taux de détection obtenu avec la base d'images KITTI, nous avons utilisé une autre base d'images Openimagesv4 dont les images ont la taille qui se rapproche de la taille des images en entrées de YOLO V3, contrairement à la base KITTI, les images de OpenImagesv4 sont de mauvaise qualité. D'où il a été nécessaire de vérifier l'ensemble des images de la base pour sélectionner uniquement celles qui contiennent les classes définies (voiture, camion, bicyclette, panneau de signalisation et personne). Cette sélection a requis plus de temps pour le prétraitement. Le graphe représentant la variation des valeurs de l'erreur moyenne en fonction du nombre d'itérations est présenté dans la figure 4.5.

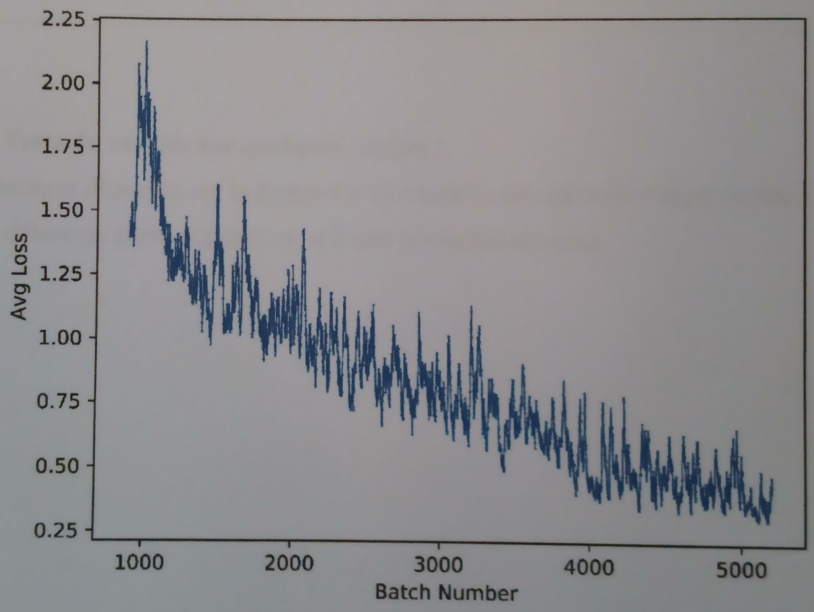


Figure 4-5 : Variation de l'erreur moyenne en fonction du nombre d'itérations en utilisant OpenImagesv4.

Le graphe illustré sur la figure 4.5 montre que l'erreur moyenne diminue et converge vers 0.4 à partir de 4000 itérations. Ceci peut être expliqué par le faite que le réseau n'est pas parfaitement paramétré. Dans notre cas nous avons utilisé 1268 images pour cinq classes, qui est un nombre d'images insuffisant pour faire un bon apprentissage, sachant qu'il faut 500 à 2000 images par classe et 10000 (2000x5) itérations.

Le tableau 4.1 suivant regroupe les résultats obtenus en termes de l'erreur moyenne et temps d'exécution pour les deux bases d'images.

Tableau 4-1 : L'erreur moyenne obtenue par l'apprentissage pour les ensembles de données KITTI, OpenImagesv4 et COCO.

	Erreur	Temps d'exécution (ms)
YOLO V3-openimagesv4	0,43	20
YOLLO V3-KITTI	1,08	44
Darknet YOLO V3-608	0,30	16

4.4.2 Tests du modèle sur quelques images :

Les images illustrées sur la figure 4.6 montrent la détection de l'objet (voiture) ainsi que le taux de détection correspondant en utilisant le modèle entrainé.

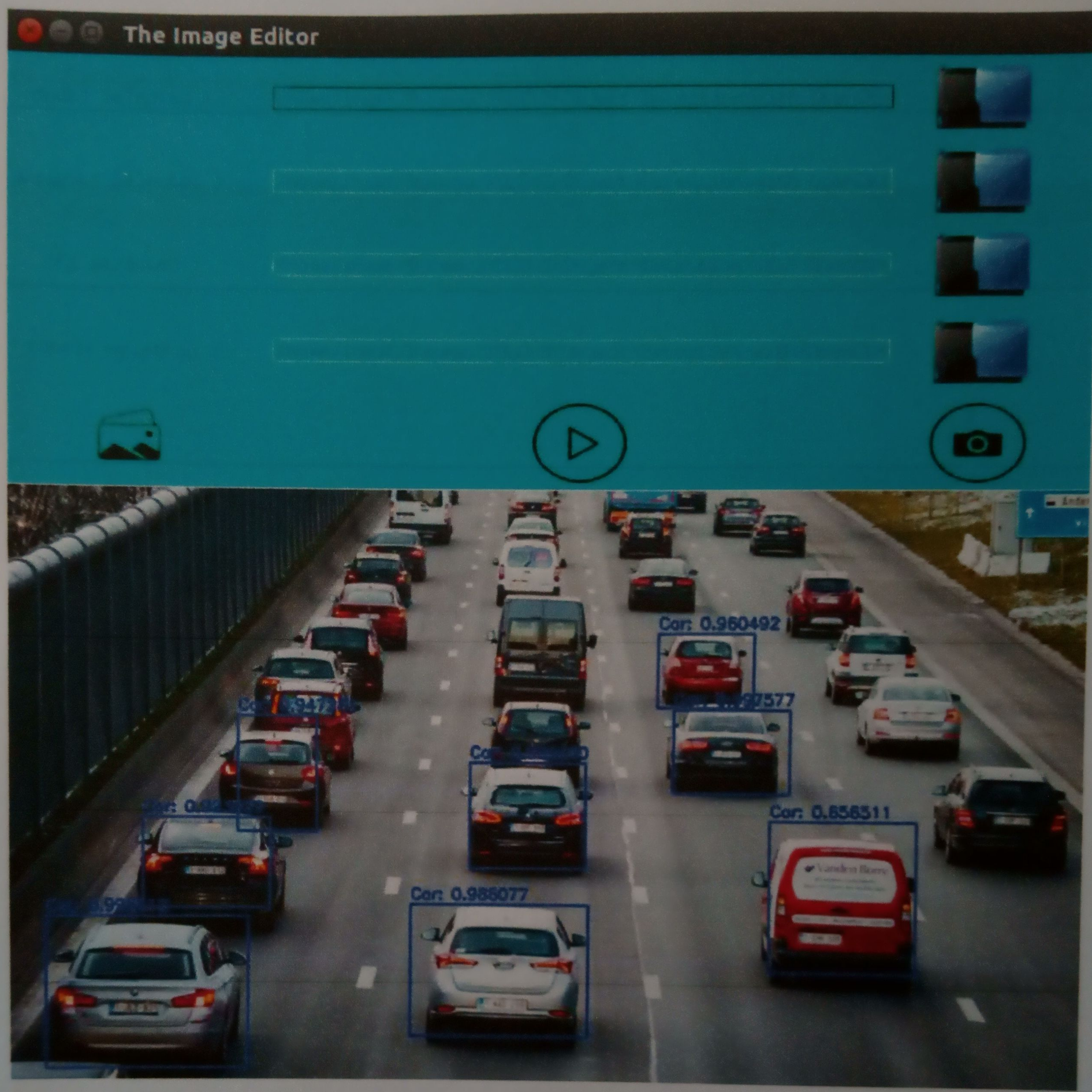


Figure 4-6 : Résultats de test du modèle sur les images (objet voiture).

L'image illustrée sur la figure 4.7 montre la détection de l'objet (camion) ainsi que le taux de détection correspondant en utilisant le modèle entrainé.

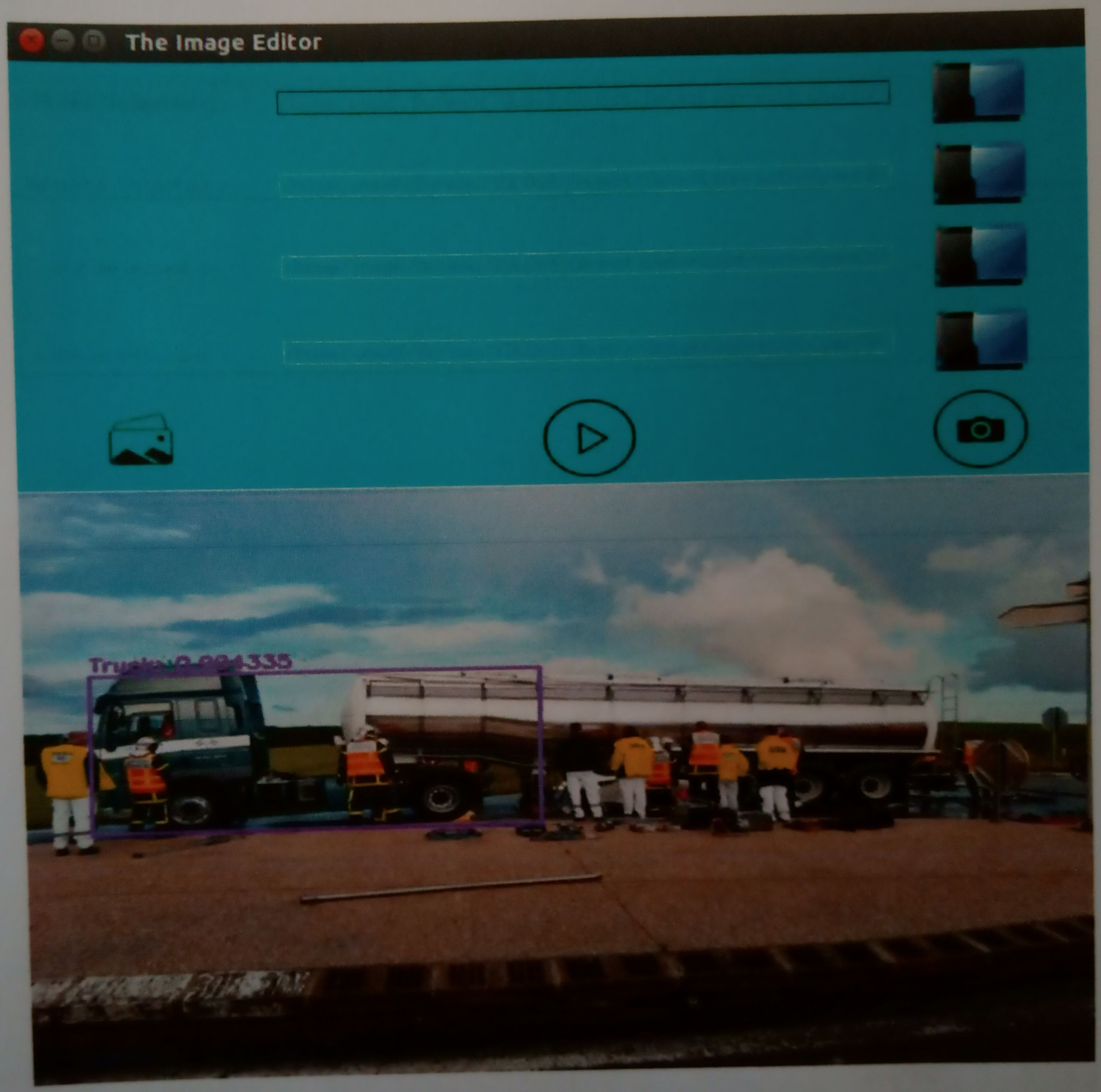


Figure 4-7 : Résultats de test du modèle sur les images (objet camion).

L'image illustrée sur la figure 4.8 montre la détection des objets des clases panneaux de signalisation et voiture ainsi que le taux de détection correspondant en utilisant le modèle entrainé.

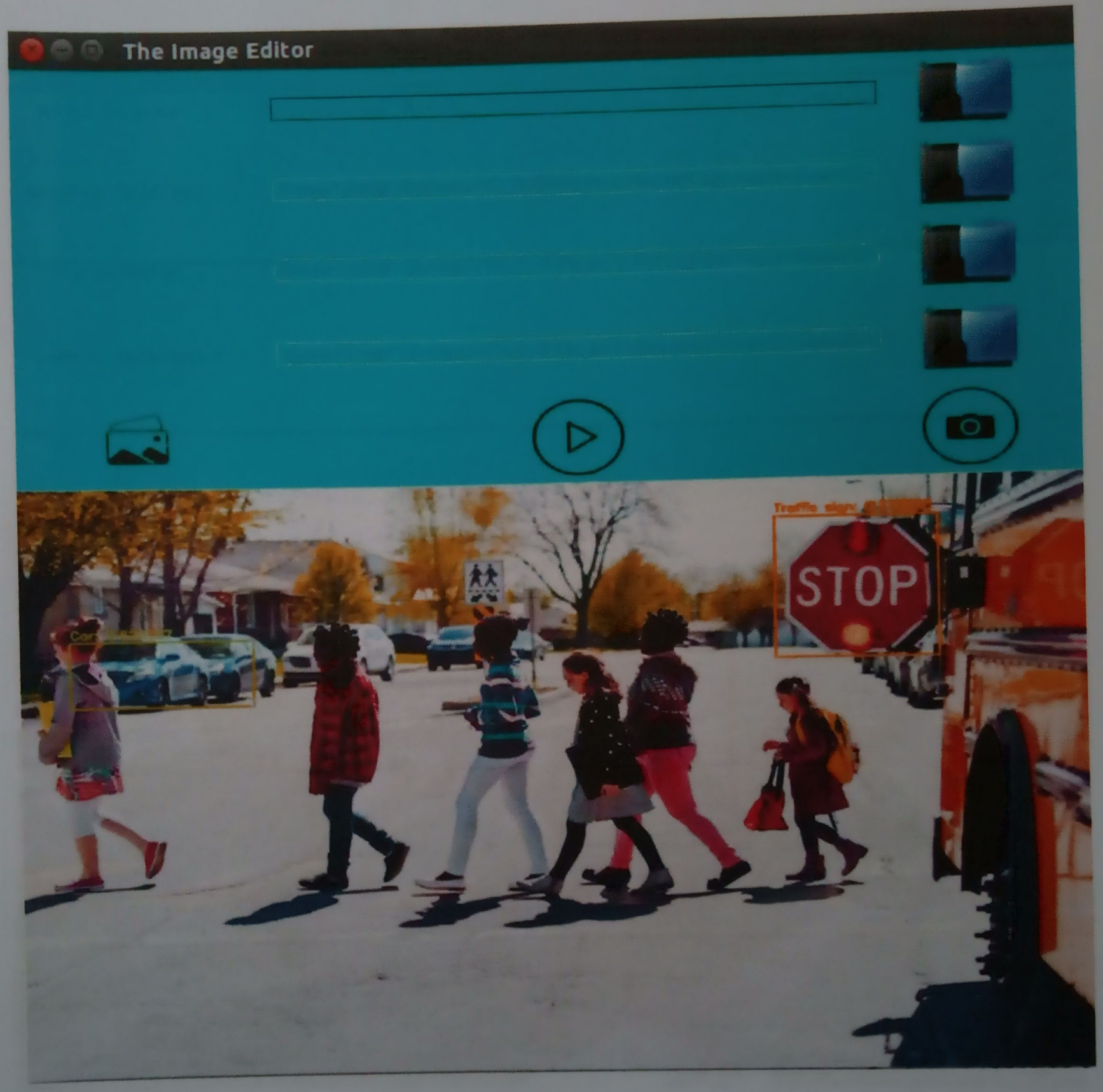


Figure 4-8 : Résultats de test du modèle sur les images (objet panneau de signalisation et voiture).

L'image illustrée sur la figure 4.9 montre la détection de l'objet de la classe bicyclette ainsi que son taux de détection.

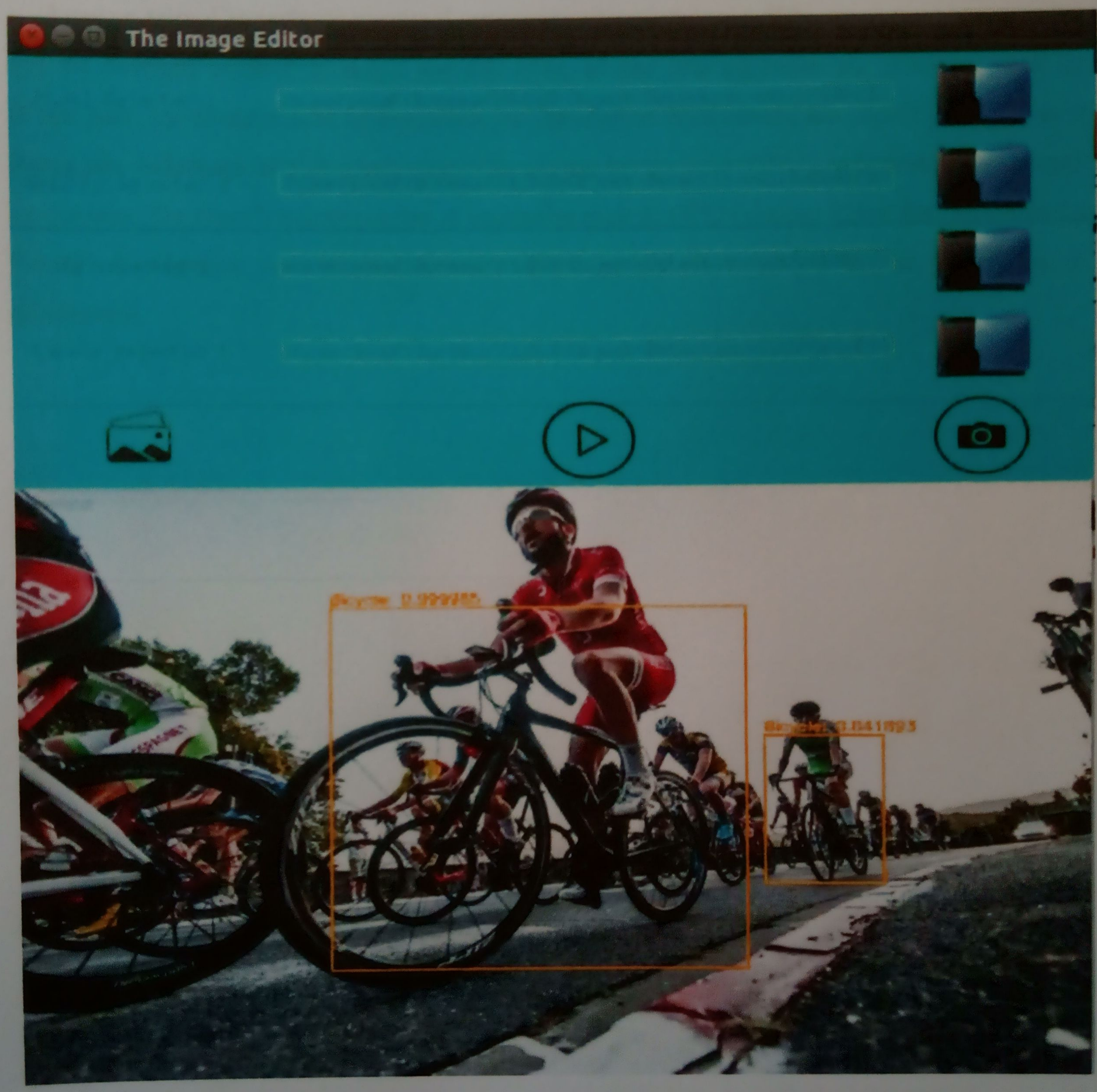


Figure 4-9 : Résultats de test du modèle sur les images (objet bicyclette).

4.5 Conclusion :

Dans ce chapitre, nous avons présenté les résultats d'apprentissage obtenus à partir des deux bases d'images utilisés. Nous avons également présenté les graphes de performances en termes d'erreur moyenne et les résultats obtenus après avoir fait la détection avec des données de test.

Avec un taux d'exactitude maximal approximativement égal à 80%, nous pouvons conclure que les résultats obtenus n'étaient pas très bons, et que cela est causé par plusieurs facteurs. D'une part, les données sont hétérogènes, car les images contiennent des objets de nature et de forme très différente. D'autre part, nous n'avons pas les moyens (GPU), ni le nombre nécessaire de données. De plus l'environnement d'exécution avec le GPU (chaque 12h il faut remonter les fichiers d'exécution sur l'environnement) offert par google colaboratory, ne faisais que se déconnecté

Conclusion générale et perspectives

Au cours de ce mémoire de fin d'études, nous avons réalisé un système de détection visuelle d'objets statique et dynamique dans un environnement de type route et classification exploitant l'apprentissage profond.

Malgré le nombre remarquable de systèmes de détection réalisés jusqu'à aujourd'hui, les résultats expérimentaux restent encore loin de la performance humaine. Cependant, par rapport aux approches classiques, les nouvelles techniques basées sur les réseaux de neurones profonds présentent des résultats qui pourraient changer le monde de la conduite autonome.

L'application implémentée est basée sur le modèle YOLO V3 avec le framework Darknet. Les raisons de ce choix est double. D'une part, le modelé est imposé par les encadreurs et d'autre part, les articles dans la littérature ont bien démontré l'avantage de ce modèle par rapport à d'autres modèles. YOLO V3 reprend les avantages de réseaux classiques tout en améliorant grandement sa vitesse.

L'application de détection d'objets en temps réel développée répond aux objectifs du projet et ses performances en termes de taux de détection sont satisfaisantes mais reste à améliorer

Ce mémoire de master 2 nous a permet de :

- Approfondir nos connaissances théoriques et pratiques déjà acquises, maîtriser les nouvelles techniques et compléter notre formation initiale pour atteindre ainsi un niveau de perfection supérieur et de pouvoir apprendre d'autres nouveautés dans les différents domaines de la science en générale et de l'informatique en particulier.
- Construire des savoirs et des savoir-faire dans le domaine de traitement d'images et de vidéo particulièrement la détection, la classification et la localisation des objets ;
- Découvrir une nouvelle technologie de l'intelligence artificielle qui est l'apprentissage profond. Cette dernière utilise des outils statistiques pour découvrir des corrélations et établir des modèles dans des données.

- Comprendre les différentes étapes de l'apprentissage profond et afin de créer des algorithmes de deep learning en Python .

Un prolongement de ce travail pourrait s'axer sur plusieurs points à savoir :

- Améliorer l'architecture afin de la rendre plus légère
- Augmenter la précision en entrainant le modèle sur une grande base de données avec un nombre élevé d'itérations.
- ·Utiliser un GPU de haute performance et d'architecture parallèle pour réduire la durée d'entraînement de réseaux de Deep Learning, de plusieurs semaines à quelques heures ou moins.
- Et enfin intégrer les modules développés dans le système de robot de navigation du Robucar du CDTA.

Bibliographie

[1] « Qu'est-ce qu'apporte l'IA à la robotique humanoïde ? » : https://www.inprincipio.xyz/robotique-humanoide.

[2] H. ABBAD. « Estimation des distances des obstacles à partir des images à l'aide de l'apprentissage automatique auto-supervisé », Mémoire Master 2 en intelligence artificielle, Université des Sciences et de la Technologie Houari Boumediene (2017).

[3] Y.CHABANE. « Elaboration d'un contrôleur adaptatif pour un robot mobile basé sur les réseaux immunitaires artificiels Ingénieur d'état en informatique », Ingénieur d'état en informatique Université Saad Dahlab (2006) : https://www.memoireonline.com/01/09/1842/m_Elaboration-controleur-adaptatif-robot-mobile-base-sur-les-reseaux-immunitaires-artificiels3.html.

[4] Les différents types de robot premiere partie (2012) :

https://www.roboticbeast.com/les-differents-types-de-robot-1/.

[5] B. Bayle. « Robotique mobile », école nationale supérieure de physique de Strasbourg, université de louis pasteur (2011).

[6] A. BIGLIA. « Analyse prospective sur l'implémentation de la voiture autonome : impact sur l'industrie automobile et le citoyen ». Louvain School of Management, Université catholique de Louvain (2015).

[7] S. PENDLETON et H. ANDERSEN. « Perception, planning, control, and coordination for autonomous vehicles », National University of Singapore (2017).

[8] L. CHERROUN. « Navigation Autonome d'un Robot Mobile par des Techniques Neuro-Floues », Doctorat en Sciences, Université Mohamed Khider Biskra (2014).

[9] D. Luz Almanza-Ojeda. « Détection et suivi d'objets mobiles perçus depuis un capteur visuel embarqué », Robotique, Université Paul Sabatier Toulouse III (2011).

[10] A. LARBI. « Detection visuelle des bords de la route pour un robot mobile autonome », Mémoire licence, Université des Sciences et de la Technologie Houari Boumediene (2016).

[11] Sensor Intelligence. LiDAR, RaDAR ou capteurs à ultrasons, quelle technologie utiliser ? (2019) : https://www.sick.com/fr/fr/lidar-radar-ou-capteurs-a-ultrasons quelle-technologie-utiliser/w/comparatif-ultrasons-lidar-radar.

[12] A. ZEROUGUI. « Traitement d'images monochromes Détection de contours, Filtrage (Spatial et fréquentiel) et segmentation par les réseaux de neurones », diplôme de Master en Informatique Industrielle, Université Larbi Ben M'hidi, Oum El Bouaghi, (2017).

[13] D. ZEROUAL. « Implémentation d'un environnement parallèle pour la compression de l'image à l'aide de fractale Magister en Informatique », Université de Batna (2006).

[14] S. AOUAT. « Cours de vision par ordinateur », Master 2 Systèmes Informatiques Intelligents, Université des Sciences et de la Technologie Houari Boumediene (2017).

[15] S. KARAGIANNAKOS. « Localization and Object Detection with Deep Learning » (2019) : https://sergioskar.github.io/Localization_and_Object_Detection/.

[16] M.SADGAL et A. EL FAZZIKI. « Reconnaissance d'objet en imagerie aérienne », RIST Vol. 14, N°02, (2004).

[17] Y. SAID, M. ATRI. « Détection d'Individus dans les Images Couleurs à base d'Histogramme Intégral de Gradient Orienté et SVM ». Universite de Monastir (2012).

[18] I. BOUTIOUTA. « Reconnaissance de l'iris par les descripteurs SIFT », Mémoire Master, Universite Badji Mokhtar Annaba (2018).

[19] Y. Boukerrouche. Mise au point d'une application de reconnaissance faciale. Mémoire master : Université Abou Bekr Belkaid (2018).

[20] S. ZHAO. « Apprentissage et Recherche par le Contenu Visuel de Catégories Sémantiques d'Objets Vidéo », Mémoire Master en Informatique, Université Paris Descartes (2007).

[21] Z. MOKRI. « Classification des images avec les réseaux de neurones convolutionnels », Mémoire Master, Université Abou Bakr Belkaid (2017).

[22] S. LIMOUSIN. « L'intelligence artificielle » (2018) : https://www.supinfo.com/articles/single/7084-intelligence-artificielle.

[23] S. Keith. « Actinvision L'apprentissage automatique, ou comment les ordinateurs apprennent à partir des données » (2016) : https://www.decideo.fr/L-apprentissage-automatique-ou-comment-les-ordinateurs-apprennent-a-partir-des-donnees_a8338.html.

[24] T. Mitchell. «Machine Learning. », McGraw-Hill, (1997).

[25] L. NEHEMY. « Estimation de modèles autorégressifs vectoriels à noyaux à valeur opérateur», Docteur de l'Université d'Évry Val d'Essonne (2015).

[26] A. Freedman. «Statistical models: Theory and practice» Cambridge University Press, (2012).

[27] M.KOUDRI. Mémoire de Master, Université Abou Bakr Belkaid (2012).

[28] Détection d'anomalie (2015) : https://dataanalyticspost.com/Lexique/detection-danomalie/.

[29] Wikipédia, « Apprentissage par renforcement » (2019) : https://fr.wikipedia.org/wiki/Apprentissage_par_renforcement/.

[30] I.CHRAIBI. KAADOUD et T. Vieville. « Reprenons les bases : Neurone artificiel, Neurone biologique » (2018) : http://www.scilogs.fr/intelligence-mecanique.

[31] D. MOUALEK. « Deep Learning pour la classification des Images », Master en Informatique, Université Abou Bakr Belkaid (2017).

[32] S. Polamuri. « Difference Between Softmax Function and Sigmoid Function » (2017): https://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/.

[33] M. Gregory Gelly. « Réseaux de neurones récurrents pour le traitement automatique de la parole », Thèse de doctorat de l'Université Paris-Saclay préparée à l'Université Paris-Sud (2017).

[34] R.Khandelwal. « Deep learning –deep belief network » (2018) : https://medium.com/datadriveninvestor/deep-learning-deep-belief-network-dbn-ab715b5b8afc.

[35] J.Brownlee. « What are Convolutional Layers for Deep Learning Neural Networks? » (2019) : https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

[36] S. Das « CNN Architectures : LeNet, AlexNet, VGG, GoogLeNet, ResNet and more… » (2017) : https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5.

[37] M. MENEGAZ. « Understanding YOLO » (2018) : https://hackernoon.com/understanding-yolo-f5a74bbc7967.

[38] G. NISHAD. « You Only Look Once(YOLO) : Implementing YOLO in less than 30 lines of Python Code » (2019) : https://towardsdatascience.com/you-only-look-once-yolo-implementing-yolo-in-less-than-30-lines-of-python-code-97fb9835bfd2.

[39] TSANG. « Review : YOLOv1 You Only Look Once (Object Détection) » (2018) : https://towardsdatascience.com/yolov1-you-only-look-once-object-detection-e1f3ffec8a89.

[40] J. HUI « Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3 » (2018): https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

[41] CHENGWEI. « Gentle guide on how YOLO Object Localization works with Keras part 2» (2017) : https://www.dlology.com/blog/gentle-guide-on-how-yolo-object-localization-works-with-keras-part-2/.

[42] R. Joseph chet. « YOLO : Real Time object detection » (2016) : https://pjreddie.com/darknet/yolo/.

[43] A. KATHURIA. « What's new in YOLO v3? Towards Datascience » (2018) : https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b.

[44] L. WENG. « Object Detection Part 4: Fast Detection Models » (2018) : https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html.

[45] P.Fuchs et P.Poulain. « Cours de Python », Université Paris Diderot-Paris 7: https://python.sdv.univ-paris-diderot.fr/.

[46] S. Bahrampour, N. Ramakrishnan et M. Shah, « Comparative Study of Deep Learning Software Frameworks » (2016): https://www.semanticscholar.org/paper/Comparative-Study-of-Deep-Learning-Software-Bahrampour-Ramakrishnan/540d3197d6b100e9ddec0706b90cb37e13742774, Consulté le: 17/07/2019

[47] J. chet Redmon. « YOLO » : Real Time object detection (2016): https://pjreddie.com/darknet/yolo/

[48] Fuat. « Google Colab Free GPU Tutorial » (2018) : https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d Consulté le 04/05/2019.

Faculté d'Eléctronique
Bibliothèque
Informatique
et d'Informatique

Résumé :

Ce sujet de master s'inscrit dans la problématique des véhicules ou robots se déplaçant de façon autonome, et s'intéresse plus particulièrement à la perception de l'environnement. Pour comprendre ce dernier, le robot doit d'abord apprendre à reconnaître les objets et les obstacles qui l'entourent quel que soit le type d'objets à détecter (piétons, véhicules, panneaux…). Afin de résoudre ce problème, les techniques d'apprentissage automatique, particulièrement l'apprentissage profond, sont exploitées et donnent actuellement les meilleurs résultats. C'est dans ce contexte que s'insère ce projet de master 2, Il s'agit de développer un algorithme basé sur l'apprentissage profond « deep learning » permettant de détecter, de classer et de localiser en temps-réel tous les objets présents dans l'environnement perceptuel du robot. Le module développé est par la suite exploité dans le robot mobile de type voiture du Centre de développement des technologies avancées (CDTA).

Mot clés: *détection, localisation, robot autonome, apprentissage profond.*

Abstract :

This subject of master is part of the issue of vehicles or robots moving autonomously, and are particularly interested in the perception of the environment. To understand the latter, the robot must first learn to recognize the objects and obstacles we need to be the type of objects to be detected (pedestrians, vehicles, panels...). In order to solve this problem, machine learning techniques, especially deep learning, are being exploited and are currently producing the best results. It is in the context of this master 2 project, it is to develop an algorithm based on deep learning to detect, classify and locate real time all objects present in the perceptual environment of the robot. The module developed is used in the following in the car-type mobile robot of Center for Advanced Technology Development (CDTA).

Key words: detection, localization, autonomous robot, deep learning.

SII.088/19