الجمهورية الجزائرية الديمقراطية الشعبية

LA REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التعليم العالي والبحث العلمي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab Blida 1

Institut d'Aéronautique et des Études Spatiales

Département Études Spatiales

Mémoire de fin d'études

En vue de l'obtention du diplôme de

Master en Aéronautique

Option : **Télécommunications Spatiales**

THEME

Implementation d'un système embarqué pour la détection des cibles.

Proposé et dirigé par :

Dr. BOUDIBA Ouissem

Dr. AZMEDROUB Boussad

Réalisé par :

DRIEF Ahmed

LALIBI Mohammed

Soutenue devant le jure composer de :

Dr.KRIM Mohamed	*Président*
Dr. BENKERCHA Rabah	*Examinateur*

Promotion : 2022 / 2023

Remerciement

Tout d'abord, nous tenons à exprimer nos sincères remerciements à Dieu Tout-Puissant pour nous avoir accordé la force et le courage nécessaires pour mener à bien ce travail.

Nous souhaitons exprimer notre gratitude particulière à Le Chef Département Dr. S Tahraoui, Dr. B. Azmedroub et Monsieur Dr. B. Ouissem pour nous avoir proposé ce sujet de recherche et pour leur encadrement tout au long de notre parcours. Leurs connaissances, leur expérience et leurs conseils précieux ont été d'une grande aide pour la réalisation de ce mémoire.

Nous aimerions également exprimer notre profonde reconnaissance envers nos parents respectifs, qui nous ont soutenus et encouragés tout au long de cette aventure académique.

Nos remerciements vont également à tous les membres du jury qui ont accepté de consacrer leur temps pour évaluer notre travail et pour leurs précieuses observations et recommandations.

Enfin, nous tenons à exprimer notre gratitude envers tous nos enseignants qui ont contribué à notre formation et à notre développement académique.

Nous sommes conscients que sans le soutien et l'implication de toutes ces personnes, la réalisation de ce travail aurait été impossible. Nous leur sommes profondément reconnaissants pour leur précieuse contribution à notre réussite.

Que tous ceux qui ont contribué à notre parcours soient remerciés du fond du cœur.

Table des matières

Remerciement

Table des matières **i**

Listes des abréviations **iv**

Table des figures **v**

Liste des tableaux **viii**

Introduction Générale **1**

1 Généralités sur la détection des objets et les systèmes embarqués **3**

1.1 Introduction 4

1.2 Détection d'objets 4

1.2.1 Principes de la détection d'objets 4

1.2.2 Applications de la détection d'objet 5

1.2.3 Défis de la détection d'objets 5

1.2.4 État de l'Art sur la détection d'objets 7

1.3 Systèmes Embarqués 8

1.3.1 Importance des systèmes embarqués dans les applications en temps réel 9

1.3.2 Déploiement des systèmes embarqués 9

1.3.3 Contraintes en temps réel dans les systèmes embarqués 10

1.3.4 Types des systèmes embarqués 11

1.3.5 Raspberry Pi 12

1.4 Conclusion 13

2 Détection d'objets avec Deep Learing **14**

2.1 Introduction 15
2.2 Intelligence artcielle 15
2.3 Machine Learning 15
2.4 Deep Learning 17
2.4.1 Concept du Deep Learning 18
2.4.2 L'avantage du Deep Learning 20
2.4.3 Applications du Deep Learning 20
2.5 Les Neurones 22
2.5.1 les Neurones artificiels 23
2.5.2 Fonctions d'Activation 25
2.5.3 Architectures des réseaux de neurone 27
2.6 Entraîner un réseau de neurones artificiels 29
2.6.1 La fonction de cout 29
2.6.2 les algorithmes d'optimisation 30
2.6.3 Underfitting and Overfitting 33
2.6.4 Gérer le overfitting par la régularisation 34
2.6.5 Réseau neuronal convolutif 36
2.6.6 Architecture du réseau neuronal convolutif 36
2.6.7 Les architectures CNN les plus courantes 39
2.7 Modèles de détection d'objets 40
2.7.1 Modèles à un seul stage (Single-Stage Models) 40
2.7.2 Modèles à deux étapes (Two-Stage Models) 45
2.8 Conclusion 46

3 Résultats et analyses **47**

3.1 Introduction 48
3.2 Outils et environement de développement 48
3.3 Dataset utilisée 49
3.3.1 Pretraitement des donnees 50
3.4 Apprentissage et détection du drone 53
3.5 Perte dans la détection d'objet 55
3.5.1 La perte de classification 56
3.5.2 La perte de localisation 56
3.5.3 La perte de regularization 56

3.5.4 La perte total 57
3.6 Modèle SSD-Mobilenet 57
3.7 Modèle Faster R-CNN-ResNet50 59
3.8 Comparaison de la méthode SSD et Faster R-CNN 61
3.9 Conclusion 62

Conclusion Générale **63**

Bibliographie **64**

Listes des abréviations

AI	Artificial Intelligence (Intelligence Artificielle)
ARM	Advanced RISC Machine (Machine RISC Avancée)
BM2711	Broadcom BCM2711 (Broadcom BCM2711)
CNN	Convolutional Neural Network (Réseau Neuronal Convolutif)
CSI	Camera Serial Interface (Interface Série de Caméra)
CV	Computer Vision (Vision par Ordinateur)
DCN	Deformable Convolutional Networks (Réseaux Convolutifs Déformables)
DenseNet	Densely Connected Convolutional Networks (Réseau Neuronal Convolutif Densément Connecté
DLA	Dictionary Learning-based Algorithms (Algorithmes basés sur l'apprentissage de dictionnaires)
EfficientNet	Efficient Neural Network (Réseau Neuronal Efficace)
FPS	Frames Per Second (Images par Seconde)
GPIO	General Purpose Input/Output (Entrée/Sortie à Usage Général)
GPS	Global Positioning System (Système de Positionnement Global)
GPU	Graphics Processing Unit (Unité de Traitement Graphique)
HDMI	High-Definition Multimedia Interface (Interface Multimédia Haute Définition)
HoG	Histogram of Oriented Gradients (Histogramme des Gradients Orientés)
IDE	Integrated Development Environment (Environnement de Développement Intégré)
Jupyter	Julia, Python and R
Go	Gigaoctet
MLP	Multilayer Perceptron (Perceptron Multicouche)
MobileNet	Efficient Convolutional Neural Networks for Mobile Vision Applications (Réseaux Neuronaux C
NLP	Natural Language Processing (Traitement du Langage Naturel)
ReLU	Rectified Linear Unit (Unité Linéaire Rectifiée)
Resnet	Residual Neural Network (Réseau Neuronal Résiduel)
RetinaNet	Focal Loss for Dense Object Detection (Perte Focale pour la Détection Dense d'Objets)
R-CNN	Region-based Convolutional Neural Network (Réseau Neuronal Convolutif Basé sur les Région
RMSprop	Root Mean Square Propagation (Propagation de la Racine de la Moyenne Carrée)
RoI	Region of Interest (Région d'Intérêt)
RPN	Region Proposal Network (Réseau de Proposition de Région)
SGD	Stochastic Gradient Descent (Descente de Gradient Stochastique)
SSD	Single Shot MultiBox Detector (Détection d'objets en une seule passe)
SVM	Support Vector Machine (Machine à Vecteurs de Support)
Tanh	Hyperbolic Tangent (Tangente Hyperbolique)
USB	Universal Serial Bus (Bus Universel en Série)
VGG16	Visual Geometry Group 16 (Groupe de Géométrie Visuelle 16)
Wi-Fi	Wireless Fidelity (Fidélité Sans Fil)
YOLO	You Only Look Once (Détection d'objets en une seule passe)

Table des figures

1.1 Étapes de base de la mise en œuvre de la détection et de la classification d'objets. 5
1.2 Exemples d'applications diverses de la détection d'objet 6
1.3 Exemples sur les défis pour la détection d'objets 7
1.4 Les Composants Raspberry Pi 4 Model B [27] 13

2.1 Intelligence Artficielle, Machine Learning et Deep Learning [28]. 16
2.2 Machine learning : un nouveau paradigme de programmation. 16
2.3 Réseau neuronal profond. 17
2.4 Différence entre l'apprentissage automatique et l'apprentissage profond. . . 18
2.5 Un réseau neuronal est paramétré par ses poids 18
2.6 Une fonction de perte mesure la qualité de la sortie du réseau [28] 19
2.7 Le score de perte est utilisé comme signal de rétroaction pour ajuster les poids[28] 19
2.8 Difference application du deep learning. 22
2.9 Neurone biologique. 22
2.10 Schéma d'un réseau de neurones monocouche [30] 23
2.11 Illustration de la fonction de décision du perceptron et de la séparabilité linéaire [29] 24
2.12 Schéma d'Unité linéaire rectifiée (ReLU) 25
2.13 Schéma de la fonction Sigmoïde 26
2.14 Schéma de la fonction tangente hyperbolique (tanh) 26
2.15 Schéma de la fonction Softmax 27
2.16 Schéma d'un réseau de neurones à une seule couche [30] 28
2.17 Schéma d'un réseau de neurones multicouche [31] 28
2.18 Illustration de la descre de gradient. 30
2.19 Exemples de sous-apprentissage (modèle linéaire), bon ajustement (modèle quadratique) et surapprentissage (polynôme de degré 15)[32] 33

2.20 Schéma de la technique de dropout. 35
2.21 Exemples des différentes techniques d'augmentation de données. 35
2.22 Exemple d'un réseau de neurones convolutif (CNN) pour la génération de cartes de caractéristiques 36
2.23 Exemples d'Architecture d'un réseau neuronal convolutif 37
2.24 Exemple de convolution 2D avec rembourrage (padding) 37
2.25 Exemple de max pooling 2D. 38
2.26 Exemple d'une couche entièrement connectée 39
2.27 Exemples des types de modèles de détection d'objets 40
2.28 Les cartes de caractéristiques de résolution inférieure (à droite) détectent des objets à plus grande échelle [33] 41
2.29 SSD : Détecteur multi-boîtes en une seule passe 42
2.30 CenterNet régresse les centres des objets ainsi que leurs attributs (largeur et hauteur pour la détection d'objets en 2D) [34] 43
2.31 Schéma d'entraînement de CenterNet [34] 44
2.32 Architecture Faster R-CNN [35] 45
2.33 Architecture VGG16. [36] 46

3.1 Logos des outils et environnements utilisés 48
3.2 Logo de Kaggle [40] 49
3.3 Exemples d'images utilisées comme dataset [41] 49
3.4 Les données utilisées dans l'entraînement et le test. 50
3.5 Exemples d'images utilisées comme dataset 51
3.6 Installation de l'outil LabelImg à l'aide de Jupyter Notebook 51
3.7 Chargement d'une image 51
3.8 Annotation d'une région à l'aide de LabelImg 52
3.9 Attribution d'une classe à l'aide de LabelImg 52
3.10 Sauvegarde de l'annotation dans LabelImg 52
3.11 Logo de Jypter Notebook [42] 53
3.12 Configuration des paramètres et des fichiers à l'aide de Jupyter Notebook . 53
3.13 Configuration des chemins et des répertoires à l'aide de Jupyter Notebook 54
3.14 Création des répertoires à l'aide de Jupyter Notebook 54
3.15 Installation de Tensorflow Object Detection à l'aide de Jupyter Notebook . 54
3.16 Création de Label Map à l'aide de Jupyter Notebook 54
3.17 Création de TF records à l'aide de Jupyter Notebook 54
3.18 Mise à jour de la configuration à l'aide de Jupyter Notebook 55

3.19 Apprentissage du modèle à l'aide de Jupyter Notebook 55
3.20 Evaluation du modèle 55
3.21 Analyses des pertes dans le modèles SSD-Mobilenet 58
3.22 Résultats obtenue du modèles SSD-Mobilenet 59
3.23 Analyses des pertes dans le modèles Faster RCNN ResNet50 60
3.24 Résultats obtenue du modèles Faster R-CNN ResNet50 61

Liste des tableaux

2.1 Comparaison entre les quatre architectures CNN 40
2.2 Expérience avec des cartes de caractéristiques à échelles multiples 42

3.1 Analyses quantitative des graphiques des modèles SSD-Mobilenet 59
3.2 Analyses quantitative des graphiques des modèles Faster R-CNN ResNet50 61

Introduction Générale

L'implémentation d'un système embarqué pour la détection des cibles a été un sujet de recherche de longue date dans le domaine de la vision par ordinateur. Au fil des années, l'utilisation croissante de la détection d'objets dans diverses applications a rendu impératif le développement de méthodes plus précises et rapides.

Traditionnellement, la détection d'objets reposait sur des approches classiques, mais l'avènement de l'apprentissage en profondeur (Deep Learning) a introduit une alternative prometteuse. Cependant, la formation de modèles profonds reste un défi complexe. Les travaux en cours offrent de nouvelles perspectives de recherche dans ce domaine, ouvrant ainsi la voie à des méthodes compétitives.[1]

La détection d'objet et tous les systèmes de détection ont deux limitations principales :

Une limitation en termes de précision, et une limitation en termes du temps.

L'obtention des hautes performances pour un système de détection d'objet dans le monde réel est un problème ouvert pour les chercheurs depuis quelques années.

Cependant, l'utilisation des systèmes embarqués pour la détection d'objets présente également des défis. En raison des ressources limitées, tels que la puissance de calcul et la mémoire, ces systèmes peuvent être confrontés à des contraintes en termes de capacité de traitement et de stockage des modèles de détection. De plus, l'intégration de capteurs sur des plateformes embarquées peut être difficile en raison de contraintes d'espace et de poids.

Dans cette étude, notre objectif est de mettre en place un système embarqué pour la détection des cibles en utilisant différents modèles (SSD et Faster R-CNN). Cette approche nous permettra de bénéficier d'une méthode rapide et efficace basée sur le Deep Learning, spécialement adaptée à l'implémentation sur des systèmes embarqués.

Le présent mémoire se concentre sur la mise en œuvre des algorithmes de détection d'objets basés sur le traitement d'images, et éventuellement sur l'intelligence artificielle telle que les méthodes de deep learning, sera réalisée au sein d'un système embarqué utilisant un Raspberry Pi équipé d'une caméra. Ce système est conçu pour des applications en temps réel.

Le premier chapitre jette les bases d'une compréhension approfondie de la détection d'objets et met en lumière l'importance cruciale des systèmes embarqués dans ce domaine. En examinant de manière détaillée la détection d'objets, ce chapitre offre une vue d'ensemble complète des techniques, des algorithmes et des méthodologies utilisés pour identifier et localiser des objets dans des images.

Le deuxième chapitre nous allons examiner le concept du Deep Learning, en commençant par le domaine de l'apprentissage automatique (machine learning), qui constitue une branche importante de l'IA. Nous discuterons de la façon dont l'apprentissage automatique permet aux ordinateurs d'accomplir efficacement des tâches sans programmation explicite. En outre, nous mettrons en évidence les principes fondamentaux de la détection d'objets par le biais du Deep Learning. Enfin, nous explorerons différentes méthodes de détection d'objets qui ont émergé ces dernières années.

Le troisième et dernier chapitre de notre mémoire nous explorerons l'application de la détection d'objets en utilisant le Deep Learning. Plus spécifiquement, nous mettrons en œuvre les algorithmes SSD et Faster R-CNN pour créer notre propre détecteur d'objets. Enfin, nous analyserons les résultats obtenus en appliquant ce détecteur à différents exemples d'images d'entrée.

Chapitre 1

Généralités sur la détection des objets et les systèmes embarqués

1.1 Introduction

De nos jours, nous assistons à une évolution de notre monde vers davantage d'intelligence, et la technologie de traitement d'image se répand à travers le globe. Des applications telles que la surveillance par caméra, le suivi d'objets ou encore la détection et la classification des objets se multiplient. Cependant, la détection d'objets demeure l'un des éléments les plus essentiels et spécifiques du traitement d'image. Les avantages de cette technique sont que nous pouvons l'utiliser à de nombreux objectifs pour faciliter et accélérer nos travaux, devenant ainsi plus facile et plus rapide que l'homme [2].

Dans ce chapitre, nous abordons les notions de base de la détection d'objets et expliquons comment les systèmes embarqués jouent un rôle fondamental dans le domaine de la vision par ordinateur.

1.2 Détection d'objets

La détection d'objets est l'un des domaines de recherche les plus actifs dans le monde de la vision par ordinateur (CV). Elle englobe de manière simultanée la classification de chaque objet présent dans une image ainsi que sa localisation précise. Remontant à l'année 2001, l'histoire de la détection d'objets est marquée par l'introduction emblématique des fameuses cascades de Haar par Paul Viola et Michael Jones[3]. Depuis lors, son utilisation s'est répandue de manière exponentielle dans diverses applications.

Cette popularité croissante a engendré un besoin pressant de développer des systèmes plus précis et plus rapides [4]. Les nouveaux algorithmes continuent de repousser les limites en termes de vitesse et de précision, surpassant constamment leurs prédécesseurs.

1.2.1 Principes de la détection d'objets

La détection d'objets englobe l'identification des zones potentiellement pourraient contenir un objet au sein de l'image, à les extraire et à les classifier en utilisant un modèle de classification d'images. Seules les régions ayant des résultats de classification optimaux sont conservées, tandis que les autres sont rejetées. Ainsi, pour obtenir une méthode de détection d'objets fiable, il est essentiel de disposer d'un algorithme de détection de régions solide et d'un algorithme de classification d'images performant[5].

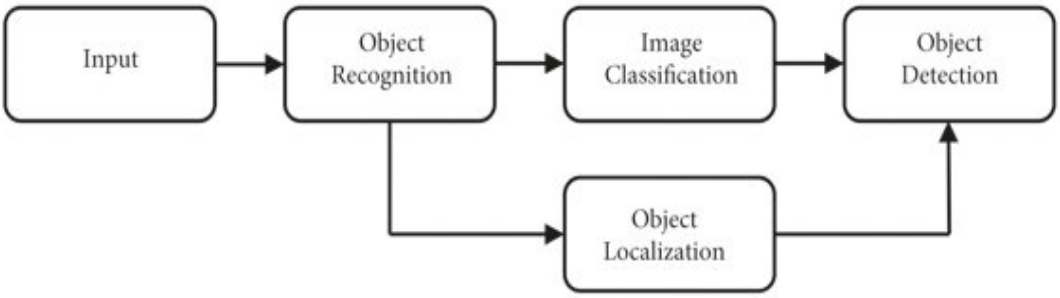


FIGURE *1.1 – Étapes de base de la mise en œuvre de la détection et de la classification d'objets.*

1.2.2 Applications de la détection d'objet

Récemment plusieurs études ont démontré l'efficacité de la détection d'objets et son rôle qui est très important dans divers domaines et large éventail, notamment :

- Vidéosurveillance
- Robotique
- Militaire
- Diagnostique d'images médicales
- Reconnaissance faciale
- Conduite autonome et Sécurité routière en prévenant les accidents potentiellement causés par la fatigue au volant.

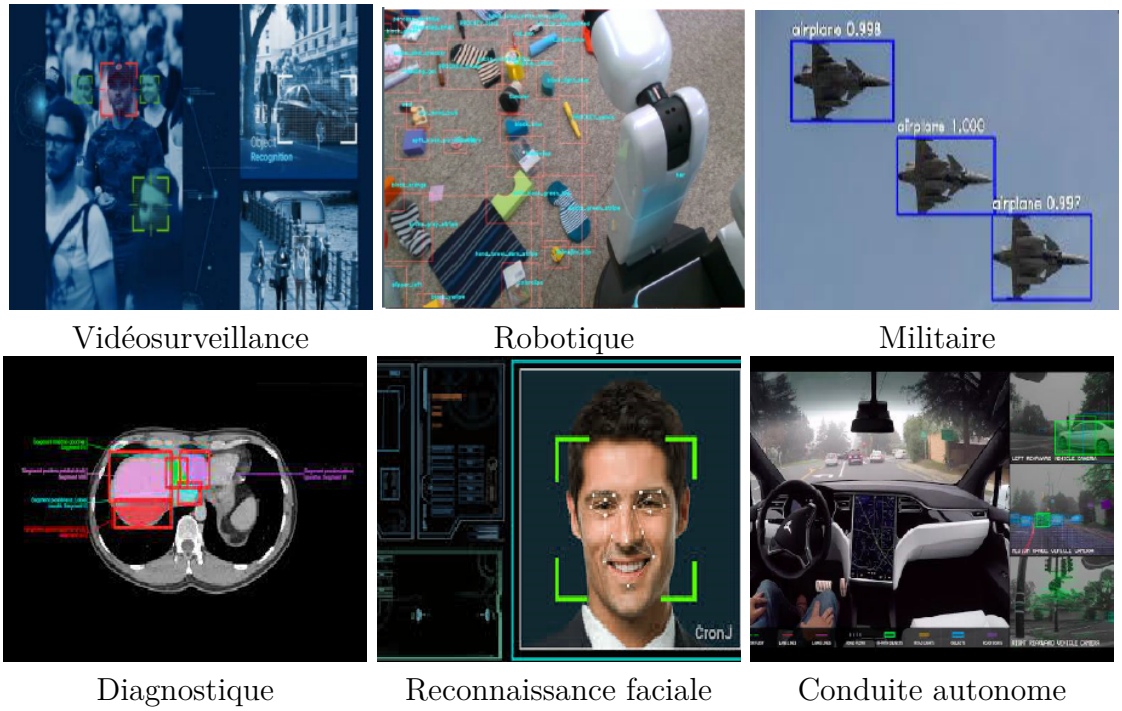
Ces applications démontrent l'importance cruciale de la détection d'objets dans différents domaines et soulignent son potentiel à améliorer notre quotidien en termes de sécurité, d'efficacité et de prise de décision.

1.2.3 Défis de la détection d'objets

La détection d'objets est une tâche complexe qui trouve son utilité dans de multiples contextes liés au traitement d'images. Pour qu'un détecteur soit idéal, il doit avoir deux caractéristiques clés :

- Haute précision de localisation et de reconnaissance : le détecteur doit être capable de localiser et de reconnaître les objets dans les images avec précision.
- Haute efficacité en temps et en mémoire : la tâche de détection doit s'exécuter à une fréquence d'images suffisante (fps) avec une utilisation de mémoire et de stockage acceptable.

Toutefois, la réalisation d'un tel détecteur n'est pas sans défis.



Vidéosurveillance | Robotique | Militaire

Diagnostique | Reconnaissance faciale | Conduite autonome

FIGURE 1.2 – *Exemples d'applications diverses de la détection d'objet*

L'un des principaux obstacles est la présence de variations intra-classe. Chaque catégorie d'objet peut comporter de nombreuses instances d'objets qui diffèrent en fonctionnalités telles que la couleur, la texture, la taille, la forme et la pose. Ces variations peuvent être causées par une multitude de facteurs, tels que :

- Eclairage
- Multi-échelle
- Angle
- Conditions météorologiques
- Distribution non uniforme
- Changements d'échelle
- Occultations
- Déformations
- Mouvement d'objets ou de caméra
- Bruit et basse résolution
- Taille d'objet
- Arrière-plans et ombrage

Le deuxième défi consiste à assurer la détection en temps réel des objets. Cela nécessite des performances élevées, qui peuvent parfois se faire au détriment de la précision pour gagner en rapidité. De plus, nous devons développer un détecteur efficace qui fonctionne de manière optimale sur des appareils ayant des ressources de calcul et de stockage limitées

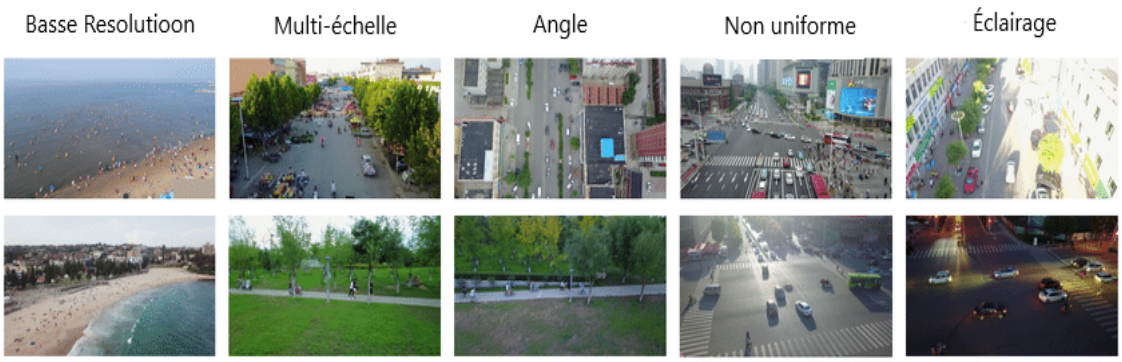


FIGURE *1.3 – Exemples sur les défis pour la détection d'objets*

[6].

1.2.4 État de l'Art sur la détection d'objets

Les méthodes de détection d'objets relèvent généralement d'approches basées sur l'apprentissage automatique (Machine Learning) ou d'approches basées sur l'apprentissage en profondeur (Deep Learning). Pour les approches d'apprentissage automatique, il devient nécessaire de définir d'abord les entités en utilisant l'une des méthodes ci-dessous, puis en utilisant une technique telle que la machine à vecteur de support (SVM) pour effectuer la classification.

D'un autre côté, les techniques d'apprentissage en profondeur sont capables de détecter des objets de bout en bout sans définir spécifiquement de caractéristiques, et sont généralement basées sur des Réseaux de Neurones Convolutifs (CNN). Les réseaux de neurones convolutifs ont contribué à une augmentation significative de la précision de la détection d'objets et ont largement dépassé d'autres modèles classiques tels que le cadre Viola & Jones [3]et les histogrammes de gradient orienté (HoG) [7].

- **Approches d'apprentissage automatique (Machine Learning) :**
  - Viola & Jones - Cadres de détection d'objets basés sur les caractéristiques de Haar[3]
  - SVM (Machine à Vecteurs de Support) - Classification des régions d'intérêt[8]
  - SIFT (Scale-Invariant Feature Transform) - Transformation d'entité invariante à l'échelle[9]
  - HOG (Histogramme de Gradient Orienté) - Caractéristiques basées sur les gradients orientés[7]
- **Approches d'apprentissage en profondeur (Deep Learning) :**
  - R-CNN (Region-based Convolutional Neural Network) - Détection d'objets basée sur des régions[10]

- Fast R-CNN (Fast Region-based Convolutional Network) - Réseau de Neurones Convolutifs Rapide avec Régions d'Intérêt[11]
- Faster R-CNN (Faster Region-based Convolutional Neural Network) - Réseau de Neurones Convolutifs avec Régions d'Intérêt plus Rapide[12]
- YOLO (You Only Look Once) - Détection d'objets en une seule passe[13]
- SSD (Single Shot MultiBox Detector) - Détection de boîtes multiples en un seul tir[14]
- ResNet (Residual Neural Network) - Réseau neuronal résiduel[15]
- DenseNet (Densely Connected Convolutional Networks) - Réseau de neurones dense pour une détection d'objets améliorée[16]
- MobileNet (Efficient Convolutional Neural Networks for Mobile Vision Applications) - Réseaux Neuronaux Convolutifs Efficients pour les Applications de Vision Mobile[17]
- EfficientNet (Rethinking Model Scaling for Convolutional Neural Networks) - Modèle d'apprentissage en profondeur efficace pour la détection d'objets[18]
- RetinaNet (Focal Loss for Dense Object Detection) - Perte Focale pour la Détection Dense d'Objets[19]
- DCN (Deformable Convolutional Networks) - Réseaux Convolutifs Déformables[20]
- CenterNet (Objects as Points) - Objets en tant que points[21]

1.3 Systèmes Embarqués

Un système embarqué est une combinaison de matériel informatique, de logiciels, et éventuellement de composants supplémentaires, qu'ils soient mécaniques ou électroniques, conçus pour accomplir une fonction dédiée. Un bon exemple est le four à micro-ondes. Presque tous les foyers en possèdent un, et des dizaines de millions d'entre eux sont utilisés chaque jour, mais très peu de personnes réalisent qu'un processeur informatique et des logiciels sont impliqués dans la préparation de leur déjeuner ou dîner.

La conception d'un système embarqué destiné à accomplir une fonction dédiée diffère directement de celle de l'ordinateur personnel. L'ordinateur personnel est lui aussi composé de matériel informatique, de logiciels et de composants mécaniques (comme les disques durs, par exemple). Cependant, un ordinateur personnel n'est pas conçu pour accomplir une fonction spécifique.[22]

1.3.1 Importance des systèmes embarqués dans les applications en temps réel

Les systèmes embarqués jouent un rôle essentiel dans les applications en temps réel en raison de leur capacité à fournir des fonctionnalités avancées et une réponse rapide aux événements. Dans le domaine de la détection d'objets, il est crucial de pouvoir traiter en temps réel les flux de données provenant de capteurs, tels que des caméras, et d'effectuer des analyses instantanées pour détecter et reconnaître les objets d'intérêt.

Les systèmes embarqués offrent une plateforme compacte et économe en énergie qui peut être utilisée pour exécuter des tâches de détection d'objets en temps réel. En intégrant des algorithmes de détection d'objets basés sur le deep learning dans ces systèmes embarqués. L'utilisation de systèmes embarqués pour la détection d'objets permet une exécution rapide et précise des algorithmes de détection, offrant ainsi des avantages tels qu'une meilleure réactivité, une faible latence et une consommation d'énergie réduite. Cette combinaison de performances en temps réel et de capacités embarquées ouvre la voie à de nombreuses applications innovantes dans le domaine de la détection d'objets.[23]

1.3.2 Déploiement des systèmes embarqués

Les systèmes embarqués trouvent des applications dans divers domaines :
- Automobile : systèmes de freinage antiblocage, navigation GPS, systèmes d'infodivertissement.
- Télécommunications : téléphones mobiles, modems, équipements de réseau.
- Électronique grand public : téléviseurs, lecteurs multimédias, montres intelligentes.
- Soins de santé : dispositifs médicaux, surveillance des signes vitaux.
- Industrie : systèmes de contrôle, automates programmables, robots industriels.
- Avionique : navigation, contrôle de vol, communication.
- Énergie : gestion de l'énergie, compteurs intelligents, panneaux solaires.
- Sécurité : surveillance vidéo, systèmes d'alarme, contrôle d'accès.

Les systèmes embarqués jouent un rôle essentiel dans notre société moderne, offrant des solutions polyvalentes et puissantes pour diverses applications.

L'intégration de nos algorithmes dans un système embarqué revêt une importance cruciale, car elle permet d'exploiter efficacement les ressources matérielles limitées tout en garantissant des performances en temps réel.

Dans ce contexte, nous devons choisir une plateforme embarquée équipée d'une caméra afin de créer un système compact et autonome.

Ce système embarqué sera capable d'effectuer la détection des cibles en temps réel, tout en capturant et en traitant la vidéo de manière fluide. Cette approche offre une flexibilité et une portabilité accrues, ouvrant la voie à des applications potentiellement variées, telles que la surveillance vidéo, la robotique ou encore la reconnaissance d'objets dans des environnements dynamiques.

1.3.3 Contraintes en temps réel dans les systèmes embarqués

Lorsqu'il s'agit de systèmes embarqués pour la détection d'objets, il existe plusieurs contraintes à prendre en compte. Voici quelques-unes des contraintes courantes associées à la détection d'objets dans les systèmes embarqués :

- Contraintes de puissance de calcul : La détection d'objets peut nécessiter des algorithmes de traitement d'image ou de vision par ordinateur complexes qui exigent une puissance de calcul significative. Dans les systèmes embarqués, les ressources de calcul peuvent être limitées, il est donc important de concevoir des algorithmes optimisés et efficaces pour répondre à ces contraintes.
- Contraintes de mémoire : Les systèmes embarqués ont généralement une quantité limitée de mémoire disponible. Les modèles de détection d'objets, tels que les réseaux de neurones convolutifs (CNN), peuvent être volumineux et nécessiter une grande quantité de mémoire pour stocker les paramètres du modèle. Il est crucial de sélectionner ou de concevoir des modèles de détection d'objets qui sont adaptés aux contraintes de mémoire des systèmes embarqués.
- Contraintes de latence : Dans certains cas, la détection d'objets doit être réalisée en temps réel avec des délais de réponse très courts. Par exemple, dans les systèmes de détection d'objets embarqués dans des véhicules autonomes, la latence doit être minimisée pour permettre des décisions rapides et réactives. Les algorithmes de détection d'objets doivent être optimisés pour réduire la latence et respecter les contraintes temporelles.
- Contraintes de consommation d'énergie : Les systèmes embarqués sont souvent alimentés par des sources d'énergie limitées, telles que des batteries. La détection d'objets peut être une tâche intensive en termes de consommation d'énergie, notamment en raison du traitement d'images ou de flux vidéo en continu. Il est important de concevoir des algorithmes économes en énergie et d'optimiser la consommation d'énergie pour prolonger l'autonomie du système.
- Contraintes de robustesse et de précision : Les systèmes de détection d'objets embarqués doivent être robustes et précis, même dans des environnements difficiles ou avec des variations des conditions d'éclairage. Les algorithmes de détection d'objets doivent être

capables de détecter les objets avec précision tout en étant résilients aux variations et aux perturbations environnementales.
– Contraintes de taille et de poids : Les systèmes embarqués sont généralement conçus pour être compacts et légers, ce qui impose des contraintes de taille et de poids. Les solutions de détection d'objets doivent être adaptées à ces contraintes pour permettre une intégration facile dans des dispositifs embarqués compacts.

En considérant ces contraintes, il est possible de concevoir des systèmes embarqués efficaces et performants pour la détection d'objets dans une variété d'applications telles que la robotique, la surveillance, la sécurité, les véhicules autonomes, etc[24].

1.3.4 Types des systèmes embarqués

Les architectures de logiciels embarqués jouent un rôle crucial dans la conception et le développement de systèmes efficaces et performants. Voici une liste raccourcie de différents types d'architectures, avec des exemples représentatifs :

– Monolithique : Un système avec un seul bloc de code logiciel exécuté sur le système embarqué. Exemple : Un système embarqué monolithique contrôlant les opérations d'une machine industrielle.
– Modulaire : Une architecture où le logiciel est divisé en modules distincts, permettant la réutilisation du code et des mises à jour ciblées. Exemple : Un système embarqué modulaire pour la domotique, avec des modules distincts pour le contrôle de l'éclairage, la surveillance de la température et la sécurité.
– Temps réel : Une architecture conçue pour répondre à des contraintes temporelles strictes. Exemple : Un système embarqué temps réel pour le contrôle autonome des véhicules, assurant une réponse précise et immédiate aux entrées des capteurs.
– Orienté objet : Une architecture basée sur des objets encapsulant des données et des fonctionnalités. Exemple : Un système embarqué orienté objet pour la détection d'objets, utilisant le Raspberry Pi pour traiter les images et identifier les objets.

En choisissant judicieusement l'architecture de logiciel embarqué adaptée à chaque projet, il est possible de créer des systèmes robustes, modulaires et réactifs, répondant aux besoins spécifiques de chaque application. La diversité des architectures permet d'exploiter les avantages du Raspberry Pi et d'autres plateformes pour des projets variés, de la domotique à l'automatisation industrielle [25].

1.3.5 Raspberry Pi

Un système embarqué Raspberry Pi est un système informatique complet qui intègre une carte Raspberry Pi comme cœur du système. Le Raspberry Pi lui-même est une carte de développement compacte et abordable, qui peut être utilisée comme un ordinateur à part entière ou comme une unité de traitement centrale dans un système embarqué [26].

Le Raspberry Pi a été initialement créé pour encourager l'apprentissage de l'informatique et de la programmation dans les écoles et les pays en développement. Cependant, il est rapidement devenu populaire auprès des amateurs, des étudiants, des développeurs et des passionnés de l'électronique.

le Raspberry Pi a conquis notre cœur grâce à son accessibilité, sa communauté et sa flexibilité.

Raspberry Pi 4

Dans le cadre de notre projet de fin d'études, on a opté pour l'utilisation du Raspberry Pi 4 comme plateforme principale. Avec ses performances améliorées par rapport aux modèles précédents.

Le Raspberry Pi 4, l'un des modèles les plus récents de la série Raspberry Pi, comprend les composants suivants :

- Processeur : Le Raspberry Pi 4 est équipé d'un processeur Broadcom BCM2711, qui est un processeur ARM Cortex-A72 64 bits quad-core cadencé à 1,5 GHz. Cela offre une puissance de calcul significativement plus élevée par rapport aux modèles précédents.
- Mémoire : Le Raspberry Pi 4 est disponible avec différentes options de mémoire vive (RAM) : 2 Go, 4 Go ou 8 Go. La quantité de RAM dépend du modèle choisi et influence les performances globales du système.
- Stockage : Le Raspberry Pi 4 utilise une carte microSD pour le stockage du système d'exploitation et des données. Cependant, il dispose également de deux ports USB 3.0 qui peuvent être utilisés pour connecter des périphériques de stockage externes tels que des disques durs ou des clés USB.
- Connectivité : Le Raspberry Pi 4 est doté de nombreuses options de connectivité. Il dispose de deux ports USB 2.0 et deux ports USB 3.0, d'un port Ethernet Gigabit pour la connexion réseau filaire, ainsi que d'une connectivité sans fil intégrée comprenant le Wi-Fi 802.11ac et le Bluetooth 5.0.
- Ports d'E/S : Le Raspberry Pi 4 est équipé de plusieurs ports d'entrée/sortie, notamment deux ports micro HDMI pour la sortie vidéo jusqu'à 4K, un port jack audio 3,5 mm, deux ports GPIO 40 broches pour la connexion de périphériques électroniques et

un port caméra CSI (Camera Serial Interface) pour connecter une caméra Raspberry Pi.

– Alimentation : Le Raspberry Pi 4 est alimenté via un adaptateur secteur USB-C avec une tension de 5 V.
– Système d'exploitation : Le Raspberry Pi 4 est compatible avec divers systèmes d'exploitation basés sur Linux, notamment Raspberry Pi OS (anciennement Raspbian), Ubuntu, etc. Ces systèmes d'exploitation sont spécifiquement optimisés pour les architectures ARM et les fonctionnalités du Raspberry Pi 4.

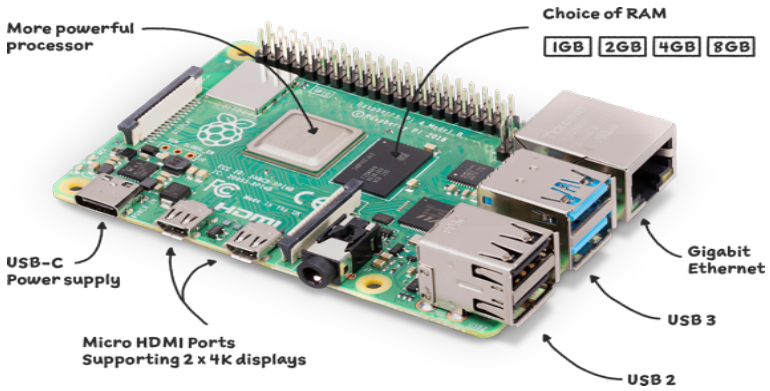


FIGURE *1.4 – Les Composants Raspberry Pi 4 Model B [27]*

Le Raspberry Pi 4 est un choix populaire pour une variété d'applications, y compris les projets embarqués, les médias, l'automatisation domestique et bien d'autres [26].

1.4 Conclusion

Dans l'ensemble, ce chapitre a démontré le potentiel de l'apprentissage en profondeur et des systèmes embarqués pour révolutionner le domaine de la vision par ordinateur. En exploitant la puissance de ces technologies, nous pouvons créer des solutions innovantes qui permettent la détection d'objets en temps réel dans une large gamme d'applications. À mesure que nous avançons, il reste encore beaucoup à explorer et à améliorer dans ce domaine.

Cependant, les avancées réalisées dans ce chapitre fournissent une base solide pour de nouvelles recherches et développements, et nous sommes impatients de voir ce que l'avenir réserve pour la détection d'objets embarquée avec l'apprentissage en profondeur.

Chapitre 2

Détection d'objets avec Deep Learing

2.1 Introduction

Aujourd'hui, le Deep Learning, connu sous le nom d'apprentissage approfondi en français, est utilisé par de nombreuses applications et technologies que nous utilisons au quotidien. Il alimente des fonctionnalités variées, telles que la capacité de Google Maps à repérer les numéros de rue sur une image, la reconnaissance faciale de Facebook sur les photos d'amis, la compréhension vocale de Siri et l'adaptation de ses réponses aux demandes des utilisateurs, ainsi que la traduction en temps réel de conversations orales sur des applications comme Skype et Google Traduction, entre autres. On parle beaucoup du Deep Learning aujourd'hui, et ce n'est pas sans raison.

Mais que se cache-t-il derrière ces deux mots ? Dans ce chapitre, nous allons explorer le concept du Deep Learning, en commençant par l'apprentissage automatique (machine learning) et et l'intelligence artificielle (IA) en général. Nous aborderons comment l'apprentissage automatique permet aux ordinateurs d'effectuer efficacement des tâches sans programmation explicite. De plus, nous mettrons en évidence les notions fondamentales liées à la détection d'objets par le Deep Learning. Enfin, nous explorerons différentes méthodes de détection d'objets proposées au cours des dernières années cité dans le chapitre précédent.

2.2 Intelligence artcielle

L'intelligence artificielle (IA) est une discipline informatique qui permet aux machines de simuler l'intelligence humaine. Elle a révolutionné de nombreux domaines en permettant aux machines d'apprendre, de raisonner et de résoudre des problèmes de manière autonome. Grâce à des algorithmes avancés, l'IA est capable d'analyser de grandes quantités de données et d'extraire des informations pertinentes. Cette technologie a des applications variées, allant des assistants virtuels aux voitures autonomes, en passant par les systèmes de recommandation. Cependant, elle soulève également des questions éthiques et sociétales. Dans cette étude, nous explorerons les bases de l'intelligence artificielle, ses applications et les défis qui l'accompagnent [28].

2.3 Machine Learning

Le machine learning, ou apprentissage automatique, est une branche de l'intelligence artificielle qui permet aux machines d'apprendre à partir des données et d'améliorer leurs

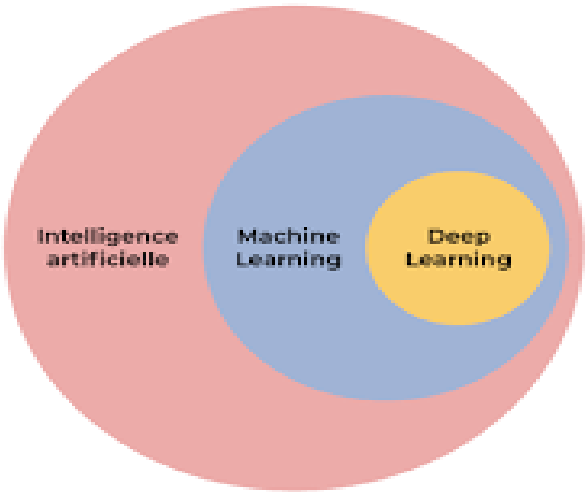


FIGURE 2.1 – *Intelligence Artficielle, Machine Learning et Deep Learning [28].*

performances sans être explicitement programmées. Il s'agit d'une approche basée sur l'exploitation de modèles statistiques et d'algorithmes afin de permettre aux machines de reconnaître des schémas, de détecter des tendances et de prendre des décisions autonomes. Le processus de machine learning implique généralement plusieurs étapes : la collecte et la préparation des données, la sélection et l'entraînement d'un modèle, et enfin, l'évaluation et l'optimisation de ce modèle. En utilisant les données d'entraînement, les machines apprennent à généraliser et à faire des prédictions sur de nouvelles données, en établissant des règles et des relations à partir des exemples fournis comme la montre la figure 2.2 Cela permet d'automatiser des tâches complexes et de réaliser des prédictions précises.

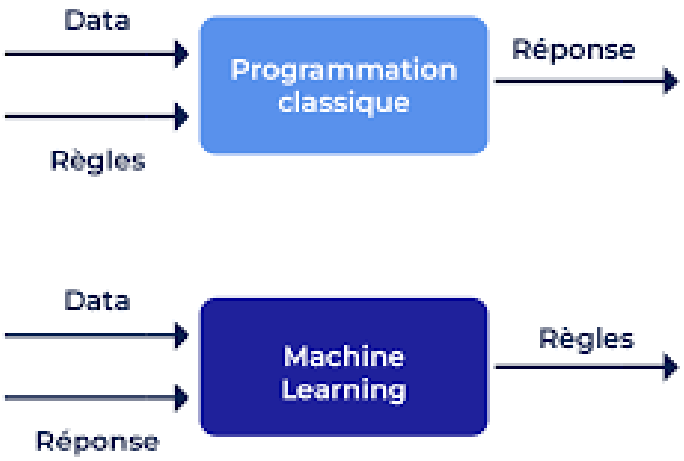


FIGURE 2.2 – *Machine learning : un nouveau paradigme de programmation.*

L'apprentissage dans le domaine de l'intelligence artificielle se réfère à un processus par lequel un système ou une machine acquiert des connaissances et des compétences à partir

de l'expérience ou des données.

Les types d'apprentissage comprennent l'apprentissage supervisé, où les modèles sont entraînés sur des données étiquetées, l'apprentissage non supervisé, qui trouve des structures dans les données non étiquetées, l'apprentissage par renforcement, où un agent apprend à travers des interactions avec un environnement, et l'apprentissage semi-supervisé, qui utilise à la fois des données étiquetées et non étiquetées pour l'apprentissage. L'apprentissage par transfert permet de transférer des connaissances d'une tâche à une autre.

Ces approches permettent aux machines d'apprendre et de s'adapter à des problèmes complexes dans divers domaines de l'intelligence artificielle.

2.4 Deep Learning

Le deep learning, également connu sous le nom d'apprentissage profond, est une branche de l'intelligence artificielle qui se concentre sur l'utilisation de réseaux de neurones artificiels profonds pour apprendre et effectuer des tâches complexes à partir de données brutes. Le terme "profond" fait référence à la structure en couches multiples des réseaux neuronaux utilisés dans cette approche.

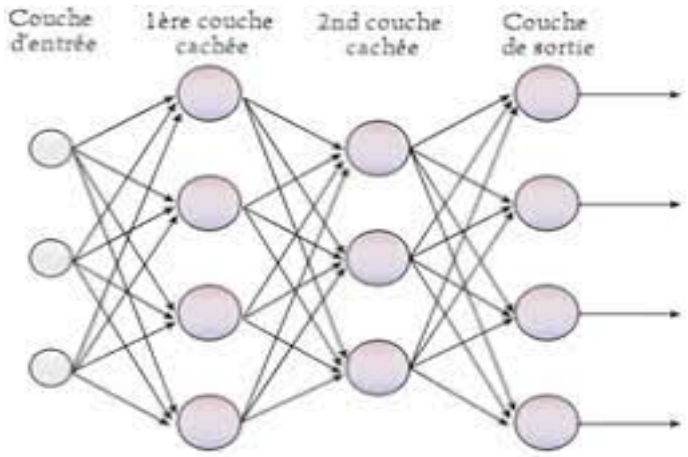


FIGURE *2.3 – Réseau neuronal profond.*

La différence principale entre le machine learning et le deep learning est que le machine learning utilise diverses techniques pour apprendre à partir des données, tandis que le deep learning est une approche plus avancée qui utilise des réseaux de neurones profonds pour extraire automatiquement des caractéristiques hiérarchiques des données brutes, offrant ainsi une capacité plus puissante pour résoudre des problèmes d'intelligence artificielle.

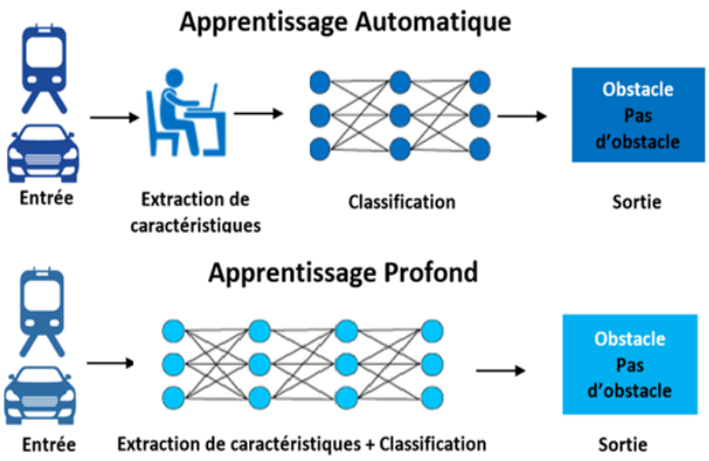


FIGURE 2.4 – *Différence entre l'apprentissage automatique et l'apprentissage profond.*

2.4.1 Concept du Deep Learning

La spécification de ce qu'une couche fait à ses données d'entrée est stockée dans les poids de la couche. En termes techniques, on pourrait dire que la transformation mise en œuvre par une couche est paramétrée par ses poids (voir figure 2.5). Dans ce contexte, l'apprentissage signifie trouver un ensemble de valeurs pour les poids de toutes les couches d'un réseau, de sorte que le réseau puisse mapper correctement les entrées d'exemple à leurs cibles associées. Mais voici le problème : un réseau neuronal profond peut contenir des dizaines de millions de paramètres. Trouver la valeur correcte pour chacun d'entre eux peut sembler une tâche intimidante.

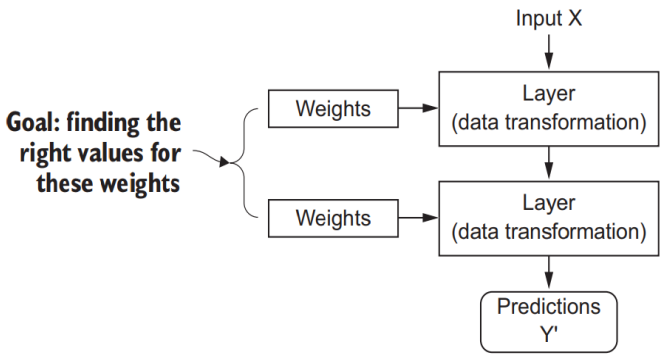


FIGURE 2.5 – *Un réseau neuronal est paramétré par ses poids [28]*

Pour contrôler quelque chose, il est d'abord nécessaire de pouvoir l'observer. Pour contrôler la sortie d'un réseau neuronal, il faut être capable de mesurer à quel point cette sortie diffère de ce que l'on attendait. C'est le rôle de la fonction de perte du réseau, également appelée fonction objectif. La fonction de perte prend les prédictions du réseau et la vraie

cible et calcule un score de distance, qui mesure à quel point le réseau a réussi sur cet exemple spécifique (voir figure 2.6) Le truc fondamental dans le deep learning est d'utiliser

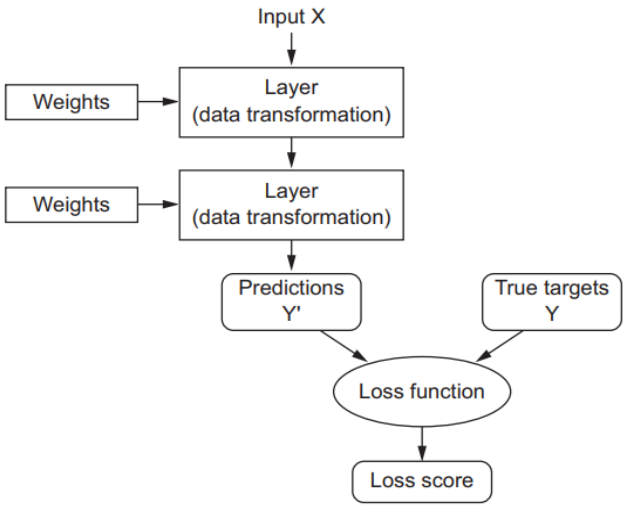


FIGURE 2.6 – *Une fonction de perte mesure la qualité de la sortie du réseau [28]*

ce score comme un signal de rétroaction pour ajuster légèrement la valeur des poids, dans une direction qui réduira le score de perte pour l'exemple actuel (voir figure 2.7). Cette adaptation est le rôle de l'optimiseur, qui met en œuvre ce qu'on appelle l'algorithme de rétropropagation (Backpropagation). Initialement, les poids du réseau sont assignés à des

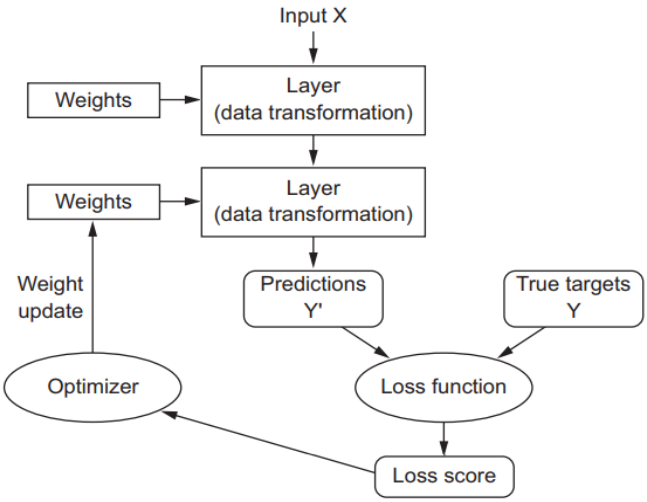


FIGURE 2.7 – *Le score de perte est utilisé comme signal de rétroaction pour ajuster les poids[28]*

valeurs aléatoires, de sorte que le réseau ne fait que mettre en œuvre une série de transformations aléatoires. Naturellement, sa sortie est loin de ce qu'elle devrait idéalement être, et le score de perte est en conséquence très élevé. Mais à chaque exemple que le réseau traite, les poids sont ajustés légèrement dans la bonne direction, et le score de perte diminue. C'est la boucle d'apprentissage.

2.4.2 L'avantage du Deep Learning

Dans le domaine de l'intelligence artificielle, l'apprentissage profond présente de nombreux avantages par rapport aux approches classiques de l'apprentissage machine. Parmi ces avantages, nous pouvons souligner :

- Automatisation : Les algorithmes d'apprentissage profond peuvent générer de nouvelles fonctionnalités à partir de données d'entraînement limitées, sans intervention humaine, ce qui permet des déploiements d'applications plus rapides et une précision supérieure.
- Bonne adaptation aux données non structurées : L'apprentissage profond excelle dans l'analyse de données non structurées telles que le texte, les images et la voix, permettant aux entreprises de tirer parti de précieuses données non structurées pour l'optimisation.
- Meilleures capacités d'auto-apprentissage : Les modèles d'apprentissage profond apprennent des caractéristiques complexes et effectuent des tâches intensives de manière plus efficace, car ils peuvent apprendre à partir des erreurs, vérifier les prédictions et apporter les ajustements nécessaires de manière autonome.
- Prise en charge d'algorithmes parallèles et distribués : Les algorithmes parallèles et distribués accélèrent l'entraînement des modèles d'apprentissage profond, offrant une évolutivité et une formation plus rapides à l'aide de plusieurs machines ou GPU.
- Rentabilité : Bien que l'entraînement des modèles d'apprentissage profond puisse être coûteux, ils aident les entreprises à réaliser des économies à long terme en réduisant les marges d'erreur et en surpassant les modèles d'apprentissage automatique classiques.
- Analyses avancées : L'apprentissage profond améliore la science des données en fournissant une analyse plus précise et fiable, alimentant des logiciels de prédiction dans divers domaines tels que le marketing, les ventes, la finance, etc.
- Évolutivité : L'apprentissage profond est hautement évolutif, capable de traiter de grands ensembles de données et d'effectuer des calculs de manière efficace, permettant un déploiement plus rapide, une modularité et une portabilité des modèles d'entraînement.

En bref, l'apprentissage profond offre des avantages considérables en automatisation, analyse de données non structurées, auto-apprentissage et rentabilité, faisant de lui une approche essentielle pour l'intelligence artificielle et l'innovation.

2.4.3 Applications du Deep Learning

Dans le cadre de l'analyse de données volumineuses, l'apprentissage profond offre de nombreuses applications pratiques, notamment :

- Reconnaissance d'images et de la parole : Les algorithmes d'apprentissage profond peuvent analyser et comprendre les images et la parole, permettant des applications telles que la reconnaissance faciale, la détection d'objets et les assistants vocaux.
- Traitement du langage naturel (NLP) : Les modèles d'apprentissage profond peuvent traiter et comprendre le langage humain, permettant des tâches telles que l'analyse des sentiments, la traduction de langues et les chatbots.
- Détection et prévention de la fraude : L'apprentissage profond peut aider à identifier les schémas et les anomalies dans de grands ensembles de données, permettant la détection d'activités frauduleuses dans des domaines tels que les transactions par carte de crédit, les demandes d'assurance et la cybersécurité.
- Systèmes de recommandation : Les algorithmes d'apprentissage profond peuvent analyser le comportement et les préférences des utilisateurs pour fournir des recommandations personnalisées pour des produits, des films, de la musique, les résaux sociaux .
  Aujourd'hui, les médias sociaux exploitent largement ces systèmes de recommandation pour fournir des suggestions personnalisées en fonction des comportements et des préférences des utilisateurs.
- Maintenance prédictive dans la fabrication : En analysant les données des capteurs des machines, l'apprentissage profond peut prédire quand une maintenance est nécessaire, aidant ainsi à prévenir les pannes coûteuses et à optimiser les calendriers de maintenance.
- Santé et diagnostic médical : L'apprentissage profond peut aider à l'analyse d'images médicales, au diagnostic de maladies et à la découverte de médicaments, améliorant ainsi la précision et l'efficacité des soins de santé.
- Analyse financière et trading : Les modèles d'apprentissage profond peuvent analyser les données financières, prédire les tendances du marché et aider à élaborer des stratégies de trading algorithmique.
- Véhicules autonomes et robotique : L'apprentissage profond joue un rôle crucial dans la conduite autonome des voitures et des robots autonomes en traitant les données des capteurs, en reconnaissant les objets et en prenant des décisions en temps réel.

Ces applications sont rendues possibles grâce à la capacité de l'apprentissage profond à extraire des représentations abstraites des données, permettant ainsi de découvrir des modèles et des relations complexes qui seraient difficiles à détecter avec des méthodes traditionnelles d'analyse de données [28].

FIGURE 2.8 – *Difference application du deep learning.*

2.5 Les Neurones

Les réseaux de neurones peuvent être définis comme des modèles de calcul qui apprennent, généralisent et organisent des données. Ils déduisent des propriétés émergentes qui permettent de résoudre des problèmes complexes. Leur mécanisme s'inspire des cellules nerveuses humaines, appelées neurones.

Un neurone biologique est constitué d'un corps cellulaire entouré de dendrites et d'un axone qui transmet l'influx nerveux. Les neurones sont connectés les uns aux autres par le biais de synapses.

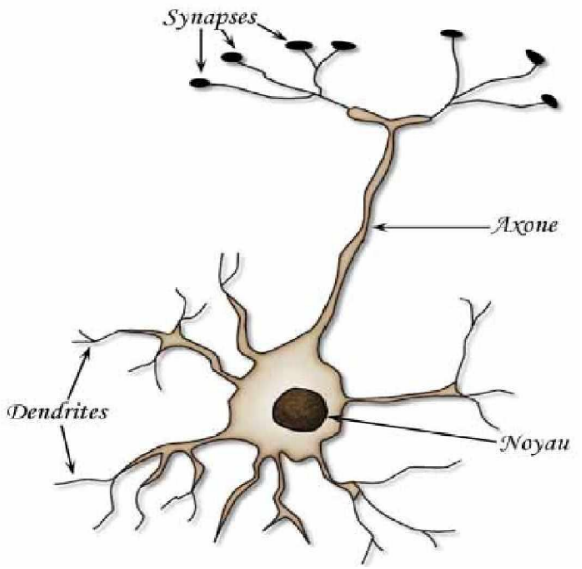


FIGURE 2.9 – *Neurone biologique.*

2.5.1 les Neurones artificiels

Plus formellement, nous pouvons situer l'idée derrière les neurones artificiels dans le contexte d'une tâche de classification binaire où nous désignons nos deux classes comme 1 (classe positive) et -1 (classe négative) pour simplifier. Nous pouvons ensuite définir une fonction d'activation $\varphi(z)$ qui prend une combinaison non-linéaire de certaines valeurs d'entrée, x, et d'un poids correspondant [29].

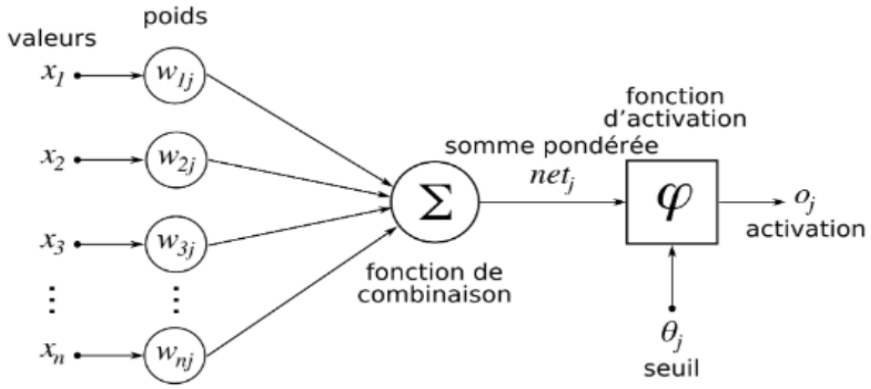


FIGURE *2.10 – Schéma d'un réseau de neurones monocouche [30]*

vecteur, w, où z est ce qu'on appelle l'entrée nette $z = w_1x_1 + w_2x_2 + \ldots + w_mx_m$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \tag{2.1}$$

Maintenant, si l'entrée nette d'un exemple particulier, $\boldsymbol{x}^{(i)}$, est supérieure à un seuil défini, θ, nous prédisons la classe 1, et la classe -1 sinon. Dans l'algorithme du perceptron, la fonction de décision, $\varphi(\cdot)$, est une variante d'une fonction d'étape unitaire :

$$\varphi(z) = \begin{cases} 1 & \text{if } z \geq \theta, \\ -1 & \text{otherwise.} \end{cases} \tag{2.2}$$

De plus, nous avons défini la fonction d'activation $\varphi(\cdot)$ de la manière suivante :

$$\varphi(z) = z = a \tag{2.3}$$

Ici, l'entrée nette, z , est une combinaison linéaire des poids qui relient la couche d'entrée à la couche de sortie :

$$z = \sum_j x_j w_j = \boldsymbol{w}^T \boldsymbol{x} \tag{2.4}$$

Alors que nous avons utilisé l'activation $\varphi(z)$ pour calculer la mise à jour du gradient, nous avons implémenté une fonction seuil pour convertir la sortie à valeur continue en étiquettes de classes binaires pour la prédiction :

$$\hat{y} = \begin{cases} 1 & \text{if } g(z) \geq 0, \\ -1 & \text{otherwise.} \end{cases} \tag{2.5}$$

La figure 2.18 illustre comment l'entrée nette $z = \boldsymbol{w}^T \mathbf{x}$ est réduite à une sortie binaire (–1 ou 1) par la fonction de décision du perceptron.

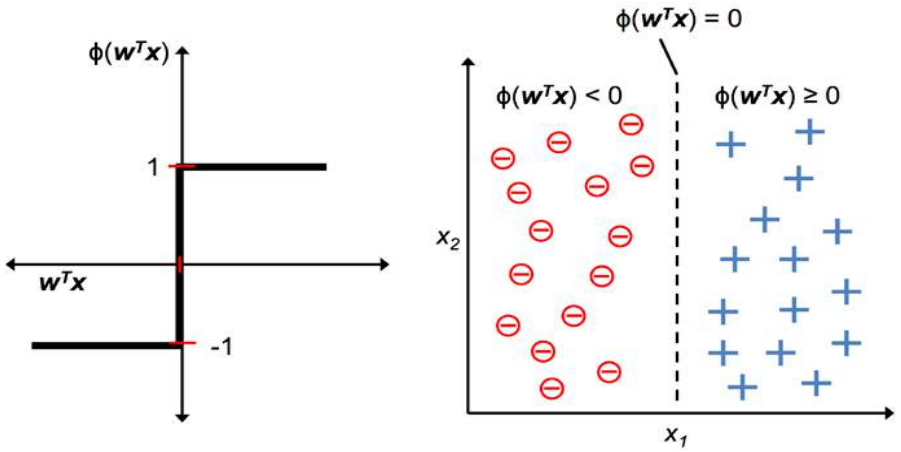


FIGURE *2.11 – Illustration de la fonction de décision du perceptron et de la séparabilité linéaire [29]*

La règle d'apprentissage du perceptron consiste à initialiser les poids à 0 ou à de petites valeurs aléatoires. Pour chaque exemple d'entraînement représenté par $\boldsymbol{x}^{(i)}$, les étapes suivantes sont réalisées :

– Calculer la valeur de sortie prédite, $\hat{y}$.
– Mettre à jour les poids du perceptron.

$$w_{j+1} = w_j - \eta \nabla J(w_j) \tag{2.6}$$

En résumé, le perceptron reçoit les entrées d'un exemple, $\boldsymbol{x}$, et les combine avec les poids, $\boldsymbol{w}$, pour calculer l'entrée nette. L'entrée nette est ensuite transmise à la fonction de seuil, qui génère une sortie binaire de -1 ou +1 - l'étiquette de classe prédite de l'exemple. Pendant la phase d'apprentissage, cette sortie est utilisée pour calculer l'erreur de prédiction et mettre à jour les poids.

2.5.2 Fonctions d'Activation

La particularité de la fonction d'activation est qu'elle est non linéaire. Cette non-linéarité permet de changer la représentation des données, d'avoir une nouvelle approche sur ces données. Ce changement de représentation ne serait pas possible avec une transformation linéaire. Chaque neurone d'une couche va appliquer la fonction d'activation de la couche sur les données. Quelques exemples de certaines des fonctions d'activation populaires utilisées dans les réseaux de neurones pour les tâches de classification sont données comme suit :

Unité linéaire rectifiée (ReLU)

La couche d'unité linéaire rectifiée (en anglais "rectified linear unit layer") est l'une des fonctions d'activation les plus couramment utilisées dans l'apprentissage en profondeur, et elle est calculée comme suit :

$$f(x) = max(0, x) \tag{2.7}$$

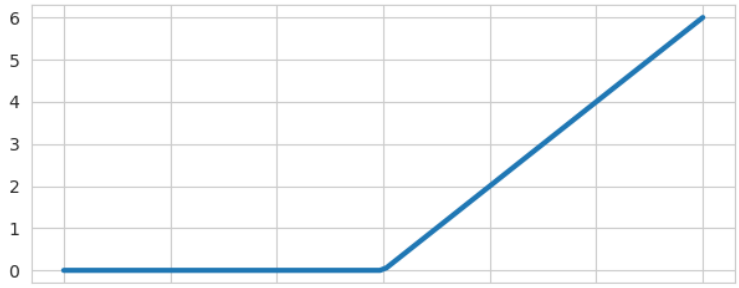


FIGURE *2.12 – Schéma d'Unité linéaire rectifiée (ReLU)*

Les fonctions d'activation ReLU sont très populaires pour la création d'un réseau non linéaire. Cette fonction permet d'effectuer un filtre sur nos données. Elle laisse passer les valeurs positives $x > 0$ dans les couches suivantes du réseau de neurones. Elle est utilisée presque partout mais surtout pas dans la couche finale, elle est utilisée dans les couches intermédiaires.

La Fonction Sigmoïde

La fonction d'activation sigmoïde est une fonction mathématique utilisée dans les réseaux de neurones. Elle transforme les valeurs d'entrée en une sortie continue entre 0 et 1. Cette fonction est couramment utilisée pour représenter des probabilités ou normaliser les valeurs dans un réseau de neurones. La fonction sigmoıde est définie comme suit :

$$\sigma(z) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

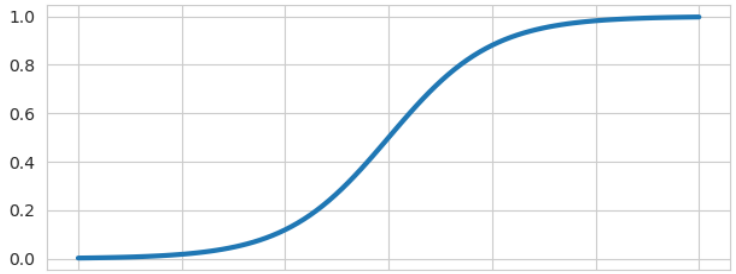


FIGURE *2.13 – Schéma de la fonction Sigmoïde*

Un avantage de la fonction sigmoïde est qu'elle produit une sortie continue dans la plage de 0 à 1, ce qui la rend particulièrement adaptée pour représenter des probabilités. Elle permet également une différenciation facile, ce qui est crucial pour les algorithmes d'apprentissage automatique basés sur le gradient. De plus, la fonction sigmoïde est symétrique par rapport à son point médian, ce qui peut être bénéfique dans certains scénarios d'apprentissage.

La Fonction Tanh

La fonction d'activation tangente hyperbolique (tanh) est une fonction mathématique utilisée dans les réseaux de neurones. Elle transforme les valeurs d'entrée en une sortie continue entre -1 et 1. La fonction tanh est similaire à la fonction sigmoïde, mais elle est centrée autour de zéro, ce qui lui permet de mieux capturer les valeurs négatives. Elle est couramment utilisée pour modéliser des données qui peuvent être symétriques par rapport à zéro. La fonction tanh est définie comme suit :

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.9}$$

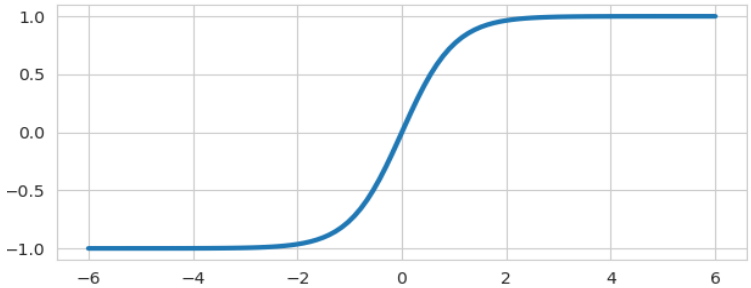


FIGURE *2.14 – Schéma de la fonction tangente hyperbolique (tanh)*

la Fonction Softmax

La fonction Softmax permet-elle de transformer un vecteur réel en vecteur de probabilité. On l'utilise souvent dans la couche finale d'un mod'ele de classification, notamment pour les problèmes en multiclasse.

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.10}$$

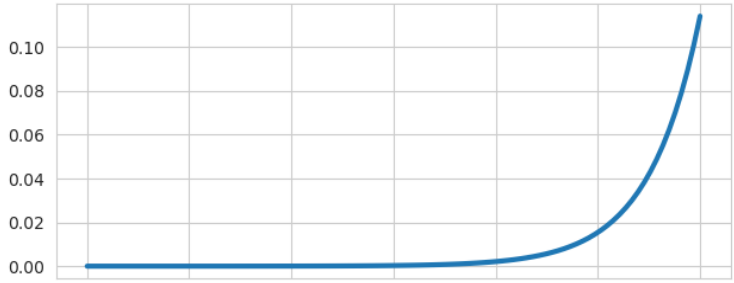


FIGURE *2.15 – Schéma de la fonction Softmax*

2.5.3 Architectures des réseaux de neurone

L'architecture d'un réseau neuronal désigne la structure et l'organisation des différents composants qui le composent. Cela inclut le nombre de couches, le nombre de neurones dans chaque couche, ainsi que les connexions entre les neurones. L'architecture d'un réseau neuronal est déterminante pour sa capacité à apprendre et à résoudre des problèmes spécifiques. En choisissant une architecture adaptée, il est possible d'optimiser les performances du réseau et d'obtenir des résultats précis et efficaces pour une tâche donnée.

Réseau de neurone a une seule couche

Un réseau de neurones à une seule couche, également appelé réseau de neurones monocouche, est une architecture simple où tous les neurones sont organisés dans une seule couche. Cette couche est souvent appelée couche de sortie, car elle génère les sorties du réseau. Chaque neurone de cette couche est connecté à toutes les entrées du réseau et effectue un calcul pondéré suivi d'une fonction d'activation pour générer sa sortie. Les réseaux de neurones à une seule couche sont couramment utilisés pour des tâches simples de classification ou de régression.

Les S neurones d'une même couche sont tous branchés aux R entrées, on dit alors que la couche est totalement connectée. On obtient la matrice $\boldsymbol{x}$ comme suit :

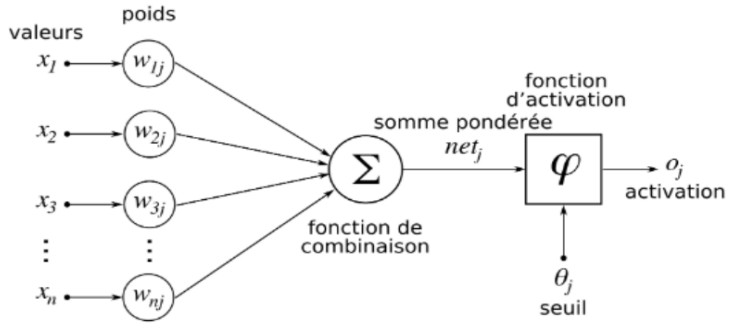


FIGURE *2.16 – Schéma d'un réseau de neurones à une seule couche [30]*

$$w = \begin{bmatrix} w_{11} & \cdots & w_{1R} \\ \vdots & \ddots & \vdots \\ w_{S1} & \cdots & w_{SR} \end{bmatrix} \tag{2.11}$$

Architecture de réseau neuronal multicouche

Un réseau de neurones multicouches, également appelé réseau de neurones profond, est une architecture composée de plusieurs couches de neurones interconnectées. Chaque couche, à l'exception de la couche d'entrée, est constituée de nombreux neurones qui transmettent les informations aux neurones de la couche suivante. Les neurones d'une couche reçoivent des entrées pondérées, effectuent un calcul et appliquent une fonction d'activation pour générer leur sortie. Les couches intermédiaires, appelées couches cachées, permettent au réseau de capturer des caractéristiques et des représentations de plus en plus abstraites à mesure que l'information se propage. La dernière couche, appelée couche de sortie, produit les résultats finaux du réseau. La figure 2.17 illustre le concept d'un MLP (Multi-Layer Perceptron) composé de trois couches :

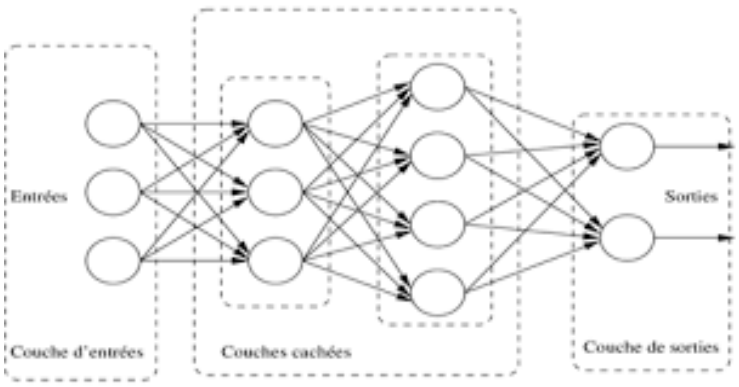


FIGURE *2.17 – Schéma d'un réseau de neurones multicouche [31]*

Les réseaux de neurones multicouches sont capables de modéliser des relations complexes et sont largement utilisés dans des domaines tels que la vision par ordinateur, le traitement du langage naturel et d'autres tâches d'apprentissage automatique. Comme le montre la figure précédente, nous désignons l'unité d'activation ith dans la couche l comme $a_i^{(l)}$.

$$\mathbf{a}^{(in)} = \begin{bmatrix} a_0^{(in)} \\ a_1^{(in)} \\ \vdots \\ a_m^{(in)} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^{(in)} \\ \vdots \\ x_m^{(in)} \end{bmatrix} \tag{2.12}$$

2.6 Entraîner un réseau de neurones artificiels

Maintenant que nous avons examiné l'architecture d'un réseau neuronal, il est temps d'approfondir certains concepts tels que la fonction de coût, l'algorithme de rétropropagation et les algorithmes d'optimisation que nous mettons en œuvre pour l'apprentissage des paramètres.

2.6.1 La fonction de cout

La fonction de coût est une mesure utilisée pour évaluer la performance d'un modèle d'apprentissage automatique. Elle quantifie l'écart entre les prédictions du modèle et les valeurs réelles des données. L'objectif est de minimiser cette fonction de coût afin d'obtenir les meilleures prédictions possibles. Différentes fonctions de coût peuvent être utilisées en fonction du problème et du type d'apprentissage, telles que l'erreur quadratique moyenne (MSE) pour la régression ou l'entropie croisée pour la classification :

$$J(\mathbf{w}) = -\sum_{i=1}^{n} y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)}) \tag{2.13}$$

Ici, $a^{(i)}$ est l'activation fonction du ième échantillon dans l'ensemble de données :

$$a^{(i)} = \varphi(z^{(i)}) \tag{2.14}$$

N'oublions pas que notre objectif est de minimiser la fonction de coût J(W) ; ainsi, nous devons calculer la dérivée partielle des paramètres W par rapport à chaque poids pour chaque couche du réseau (rétropropagation) :

$$\frac{\partial}{\partial w_{ji}^{(l)}} J(\mathbf{w}) \tag{2.15}$$

Mettre à jour les poids :

$$w_{j+1} = w_j - \eta \nabla J(w_j) \tag{2.16}$$

Dans la prochaine section, nous parlerons de l'algorithme d'optimisation, qui nous permet de trouver les valeurs optimales des paramètres du modèle afin de minimiser la fonction de coût.

2.6.2 les algorithmes d'optimisation

Les algorithmes d'optimisation en apprentissage automatique sont des méthodes utilisées pour ajuster les paramètres d'un modèle de manière itérative afin de minimiser la fonction de coût associée à la tâche d'apprentissage. Leur objectif principal est de réduire la fonction de coût en trouvant les valeurs optimales des paramètres qui permettent d'obtenir les meilleures performances du modèle. Ces algorithmes utilisent différentes stratégies pour ajuster les paramètres du modèle de manière itérative. Voici quelques-uns des algorithmes d'optimisation couramment utilisés :

Descente de gradient (Gradient Descent)

L'algorithme de descente de gradient utilise le gradient de la fonction de coût par rapport aux paramètres pour déterminer la direction dans laquelle les paramètres doivent être ajustés. Il ajuste les paramètres itérativement en suivant la direction opposée au gradient, de sorte que la fonction de coût diminue à chaque itération. Pour calculer le gradient de

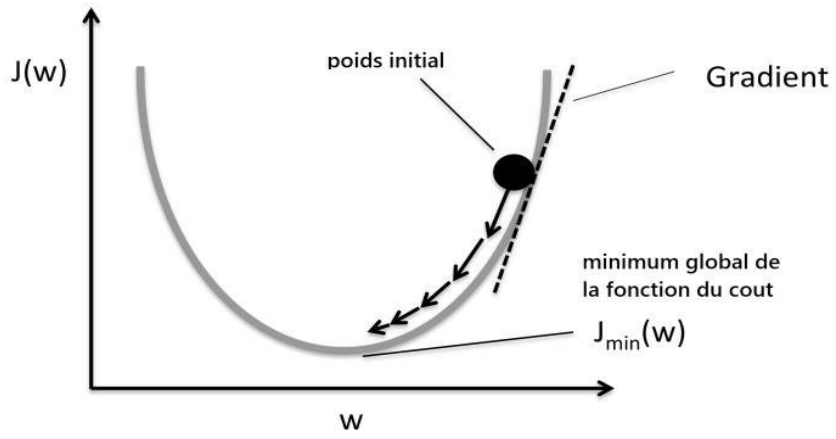


FIGURE *2.18 – Illustration de la descre de gradient.*

la fonction de coût, nous devons calculer la dérivée partielle de la fonction de coût par rapport à chaque poids, w_j

$$\frac{\partial J}{\partial w_j} = -\sum(y^{(i)} - \varphi(z^{(i)}))x_j^{(i)} \tag{2.17}$$

Puisque nous mettons à jour tous les poids simultanément :

$$w_{j+1} = w_j - \eta \nabla J(w_j) \tag{2.18}$$

La descente de gradient peut être réalisée avec différentes variantes, telles que la descente de gradient stochastique (SGD) qui utilise un échantillon aléatoire à chaque itération, la descente de gradient par lot (batch gradient descent) qui utilise tous les exemples d'entraînement à chaque itération, ou la descente de gradient mini-batch qui utilise un sous-ensemble d'exemples d'entraînement.

Momentum

Le momentum est un algorithme d'optimisation conçu pour accélérer la convergence de la descente de gradient en utilisant un terme d'inertie. L'idée principale derrière le momentum est d'accumuler une moyenne pondérée des gradients précédents et de l'utiliser pour guider les mises à jour des poids du modèle. Pour un poids w_j à l'itération t :

$$v_t = \mu v_{t-1} - \mathrm{lr} \cdot \eta \nabla J(w_j) \tag{2.19}$$

ou :

$$w_j = w_j + v_t \tag{2.20}$$

- η est le taux d'apprentissage (learning rate), qui contrôle la taille des mises à jour des poids.
- $\nabla J(w_j)$ est le gradient de la fonction de coût par rapport au poids w_j.
- v_t est la variable de momentum à l'itération t.
- μ est le coefficient de momentum..

RMSprop (Root Mean Square Propagation)

L'algorithme RMSprop utilise les estimations des carrés des gradients précédents pour mettre à jour les taux d'apprentissage. Il favorise une convergence plus rapide et une meilleure adaptation des taux d'apprentissage aux différentes caractéristiques des données.

$$g_t = \nabla J(w_j) \quad \text{(gradient du poids } w_j\text{)} \tag{2.21}$$

$$v_t = \gamma v_{t-1} + (1-\gamma) g_t^2 \quad \text{(mise à jour du deuxième moment)} \tag{2.22}$$

$$w_j = w_j - \eta \frac{g_t}{\sqrt{v_t + \epsilon}} \quad \text{(mise à jour du poids)} \tag{2.23}$$

Adam (Adaptive Moment Estimation)

Adam est un algorithme d'optimisation couramment utilisé dans l'apprentissage automatique. Il combine les avantages du momentum et de RMSprop en adaptant les taux d'apprentissage pour chaque paramètre du modèle. Pour un poids w_j à l'itération t :

$$g_t = \nabla J(w_j) \quad \text{(gradient du poids } w_j\text{)} \tag{2.24}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad \text{(mise à jour du premier moment)} \tag{2.25}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad \text{(mise à jour du deuxième moment)} \tag{2.26}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \text{(correction du premier moment biaisé)} \tag{2.27}$$

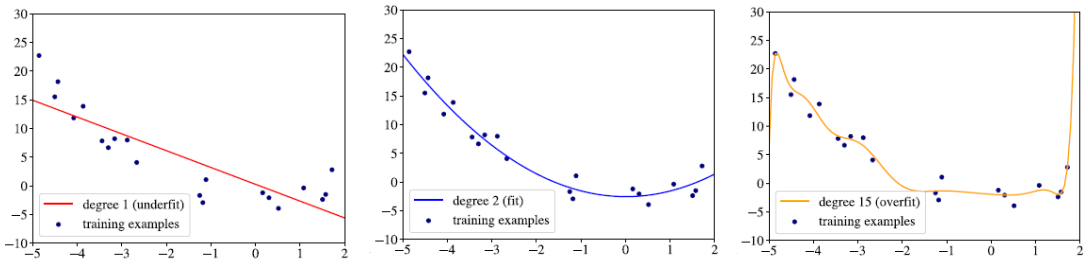
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \quad \text{(correction du deuxième moment biaisé)} \tag{2.28}$$

$$w_j = w_j - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \quad \text{(mise à jour du poids)} \tag{2.29}$$

L'algorithme Adam ajuste les taux d'apprentissage de manière adaptative en utilisant les estimations du premier moment et du deuxième moment, corrige les biais introduits par l'initialisation des moments et effectue les mises à jour des poids en utilisant ces estimations adaptées. L'utilisation de l'algorithme Adam permet généralement d'améliorer la vitesse de convergence, la robustesse du modèle et la capacité à s'adapter à différentes situations d'optimisation.

2.6.3 Underfitting and Overfitting

Si le modèle commet de nombreuses erreurs sur les données d'entraînement, nous disons que le modèle a un biais élevé ou qu'il présente un sous-apprentissage. Ainsi, le sous-apprentissage correspond à l'incapacité du modèle à prédire correctement les étiquettes des données sur lesquelles il a été entraîné. La figure 2.19 présente les résultats de l'apprentissage sous forme de trois graphiques distincts.



Underfitting (Sur-Apprentissage) | Good fit (Bon ajustement) | Overfitting (Sous-Apprentissage)

FIGURE *2.19 – Exemples de sous-apprentissage (modèle linéaire), bon ajustement (modèle quadratique) et surapprentissage (polynôme de degré 15)[32]*

Il peut y avoir plusieurs raisons au **Underfiting** (sous-apprentissage), dont les plus importantes sont les suivantes :

- La première raison est facile à illustrer dans le cas de la régression unidimensionnelle : l'ensemble de données peut ressembler à une ligne courbée, mais notre modèle est une ligne droite.
- La deuxième raison peut être illustrée comme ceci : supposons que vous souhaitez prédire si un patient a un cancer, et les caractéristiques dont vous disposez sont la taille, la pression artérielle et la fréquence cardiaque. Ces trois caractéristiques ne sont clairement pas de bons prédicteurs du cancer, donc notre modèle ne sera pas en mesure d'apprendre une relation significative entre ces caractéristiques et l'étiquette.

La solution au problème de sous-apprentissage est d'essayer un modèle plus complexe ou d'élaborer des caractéristiques ayant un pouvoir prédictif plus élevé.

Overfitting se produit lorsque le modèle devient trop complexe et s'adapte excessivement aux données d'entraînement spécifiques, en mémorisant les bruits et les variations aléatoires. Cela se traduit par une excellente performance sur les données d'entraînement, mais une performance médiocre sur de nouvelles données. Plusieurs solutions sont possibles pour résoudre le problème de overfitting (surapprentissage) :

- Essayez un modèle plus simple
- Réduisez la dimensionnalité des exemples dans le jeu de données.

– Ajoutez davantage de données d'entraînement, si possible.
– Appliquer une régularisation sur le modèle.

La **régularisation** et le **dropout** est l'approche la plus couramment utilisée pour prévenir le surapprentissage.

2.6.4 Gérer le overfitting par la régularisation

La régularisation est un terme générique qui englobe des méthodes qui obligent l'algorithme d'apprentissage à construire un modèle moins complexe. En pratique, cela entraîne souvent un biais légèrement plus élevé mais réduit considérablement la variance. Ce problème est connu dans la littérature sous le nom **biais-variance tradeoff**. Parmi les différentes méthodes de régularisation, nous pouvons citer L1 et L2 regularization, dropout, data augmentation.

L1/L2 regularization

Les deux types de régularisation les plus couramment utilisés sont appelés régularisation L1 et régularisation L2. L'idée est assez simple. Pour créer un modèle régularisé, nous modifions la fonction objective en ajoutant un terme de pénalisation dont la valeur est plus élevée lorsque le modèle est plus complexe. Dans **la régularisation** L1, les coefficients de régression sont obtenus en minimisant la fonction de perte L1, exprimée comme suit :

$$J(w)_{\text{L1}} = \sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2 + \alpha\|w\|_1 \tag{2.30}$$

ou :

$$\|w\|_1 = \sum_{j=1}^{m}|w_j| \tag{2.31}$$

Dans la régularisation L2, les coefficients de régression sont obtenus en minimisant la fonction de perte L1, exprimée comme suit :

$$J(w)_{\text{L2}} = \sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2 + \alpha\|w\|_2^2 \tag{2.32}$$

ou :

$$\|w\|_2^2 = \sum_{j=1}^{m} w_j^2 \tag{2.33}$$

Dropout

Dropout est une technique de régularisation spécifique aux réseaux de neurones. Pendant l'entraînement, des neurones aléatoires sont abandonnés ou désactivés avec une certaine probabilité. Cela oblige le réseau à apprendre de manière plus robuste en évitant de trop dépendre de certains neurones spécifiques, ce qui réduit le risque de surajustement.

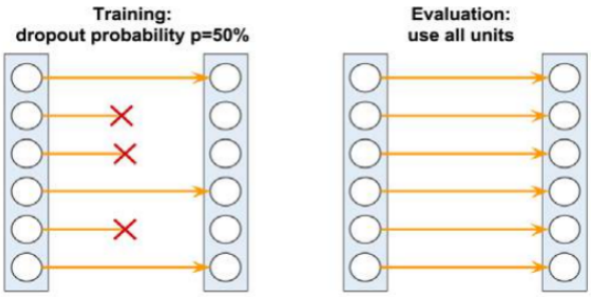


FIGURE *2.20 – Schéma de la technique de dropout.*

Data augmentation

La data augmentation est une technique qui consiste à augmenter la taille du jeu de données d'entraînement en générant de nouveaux exemples synthétiques. Elle joue un rôle important dans la régularisation du modèle en introduisant de la variabilité et en aidant le modèle à mieux généraliser. En appliquant des transformations aléatoires ou contrôlées sur les exemples existants. Voici quelques-unes des techniques couramment utilisées :

- Translations et rotations
- Miroir (symétrie)
- Ajustement du contraste la luminosité
- Ajout de bruit
- Recadrage et redimensionnement

FIGURE *2.21 – Exemples des différentes techniques d'augmentation de données.*

La data augmentation permet d'améliorer la capacité du modèle à s'adapter à des données nouvelles et inconnues, tout en réduisant la sensibilité aux variations mineures des données d'entrée.

2.6.5 Réseau neuronal convolutif

Les réseaux de neurones convolutionnels (CNN) sont un type de modèle d'apprentissage automatique utilisé principalement pour l'analyse d'images et de données structurées. Ils sont largement utilisés dans des domaines tels que la vision par ordinateur, la reconnaissance d'objets, la classification d'images et la détection d'anomalies.

Comme vous pouvez le voir dans la figure 2.22, un réseau de neurones convolutif (CNN) calcule des cartes de caractéristiques à partir d'une image d'entrée, où chaque élément provient d'un patch local de pixels dans l'image d'entrée.

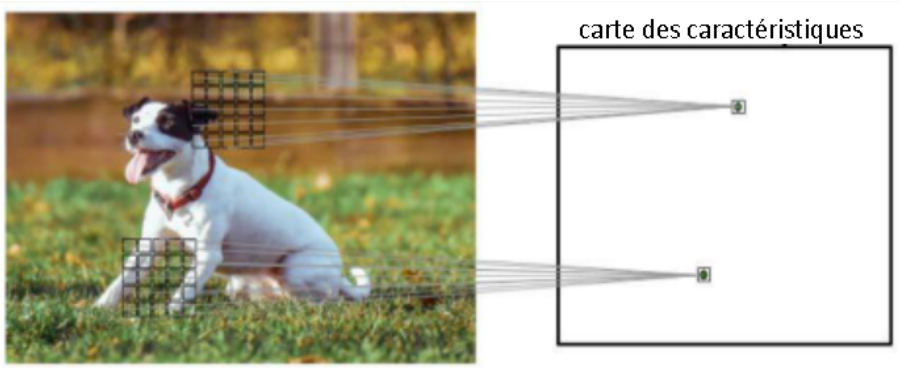


FIGURE *2.22 – Exemple d'un réseau de neurones convolutif (CNN) pour la génération de cartes de caractéristiques*

- **La connectivité parcimonieuse (sparse connectivity) :** se réfère au fait que chaque neurone dans une couche convolutive est connecté uniquement à un sous-ensemble restreint de neurones de la couche précédente, appelé champ récepteur local. Cela réduit la complexité du modèle tout en capturant les motifs locaux importants de l'image.
- **Partage des paramètres (parameter-sharing) :** Les mêmes poids sont utilisés pour différents patches de l'image d'entrée.

2.6.6 Architecture du réseau neuronal convolutif

Un CNN (réseau de neurones convolutif) comprend généralement trois couches : une couche convolutive, une couche de pooling et une couche entièrement connectée. La figure 2.23 illustre une exemple d'architecture d'un CNN :

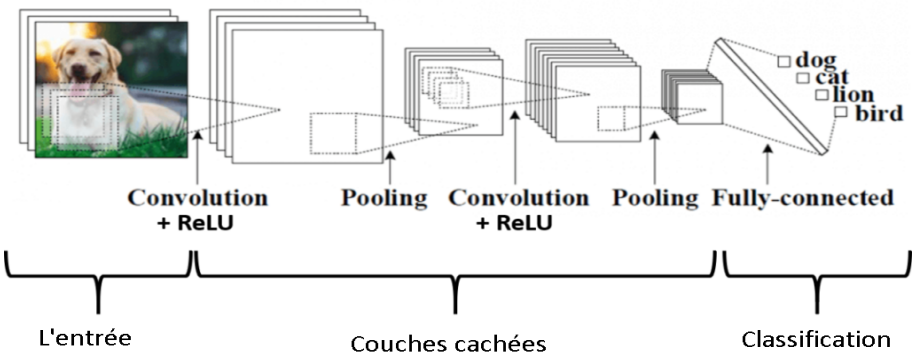


FIGURE 2.23 – *Exemples d'Architecture d'un réseau neuronal convolutif*

Couche convolutif

La couche convolutive extrait les caractéristiques d'une image ou d'une entrée spatiale en appliquant des filtres de convolution. Ces filtres permettent de détecter des motifs visuels locaux tels que des bords, des textures et des formes, ce qui permet au réseau de neurones convolutif (CNN) de reconnaître des objets et des structures plus complexes.

La figure 2.24 illustre la convolution 2D d'une matrice d'entrée de taille 8×8 à l'aide d'un noyau de taille 3×3. La matrice d'entrée est rembourrée de zéros avec $p = 1$. En conséquence, la sortie de la convolution 2D aura une taille de 8×8 :

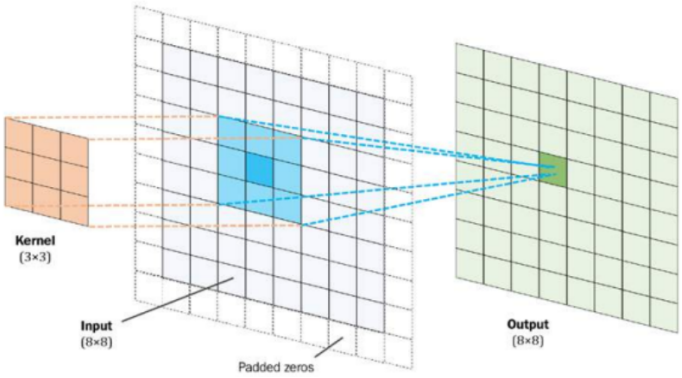


FIGURE 2.24 – *Exemple de convolution 2D avec rembourrage (padding)*

Si nous avons une entrée de taille W x W x D avec Dout nombres de noyaux d'une taille spatiale de F, avec un pas de S et un padding de P, alors la taille du volume de sortie peut être déterminée par la formule 2.34 :

$$W_{\text{out}} = \frac{W - F + 2P}{S} + 1 \tag{2.34}$$

En résumé, la couche convolutive dans un CNN effectue une opération de convolution sur l'image ou l'entrée spatiale pour extraire des caractéristiques significatives, ce qui est

crucial pour la capacité du réseau à comprendre et à traiter les données visuelles.

Couche de pooling

Dans une couche de pooling, une fenêtre de pooling (généralement de taille 2×2) se déplace sur la carte des caractéristiques de la couche précédente avec un pas spécifié. À chaque position, le pooling effectue une opération d'agrégation, généralement une fonction de maximum (*max pooling*) ou de moyenne (*average pooling*), en prenant la valeur maximale ou moyenne des éléments dans la fenêtre. La figure suivante illustre le *max pooling* 2D d'une matrice d'entrée de taille 8×8. Si nous avons une carte d'activation de taille

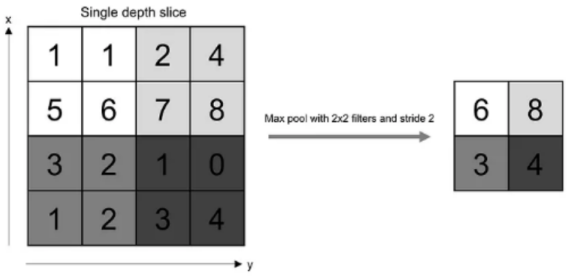


FIGURE *2.25 – Exemple de max pooling 2D.*

W x W x D, un noyau de pooling de taille spatiale F et un pas S, alors la taille du volume de sortie peut être déterminée par la formule 2.35 :

$$W_{\text{out}} = \frac{W - F}{S} + 1 \tag{2.35}$$

En résumé, la couche de pooling dans un CNN réduit la taille spatiale des caractéristiques en effectuant une opération d'agrégation, aidant ainsi à réduire la complexité du modèle, à améliorer l'efficacité computationnelle et à introduire une certaine invariance aux translations locales.

Couche entièrement connectée :

La couche entièrement connectée, également appelée couche dense, est une composante essentielle d'un réseau de neurones. Elle est utilisée pour effectuer la classification ou la prédiction finale en prenant en compte les caractéristiques extraites par les couches précédentes.La figure 2.26 représente une couche entièrement connectée (fully connected layer) avec une fonction d'activation softmax :

En résumé, la couche entièrement connectée dans un réseau de neurones est responsable de la combinaison des caractéristiques extraites et de la prise de décision finale pour la classification ou la prédiction des données.

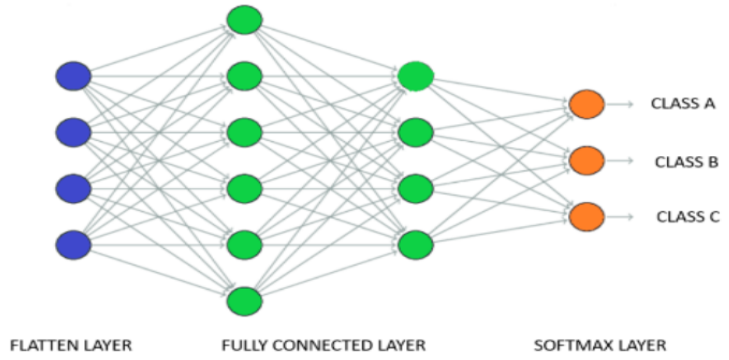


FIGURE *2.26 – Exemple d'une couche entièrement connectée*

2.6.7 Les architectures CNN les plus courantes

Il existe plusieurs architectures de réseaux de neurones convolutionnels. Dans notre travail, nous nous intéressons à quatre architectures CNN les plus performantes (SqueezeNet, VGG16, ResNet50, InceptionV3).

SqueezeNet est le nom d'un réseau neuronal profond lancé en 2016. Il a été développé par des chercheurs de DeepScale, de l'Université de Californie à Berkeley et de l'Université de Stanford. L'objectif des auteurs lors de la conception de SqueezeNet était de créer un réseau de neurones plus petit avec moins de paramètres, qui pourrait facilement s'intégrer dans la mémoire de l'ordinateur et être transmis plus facilement sur un réseau informatique. SqueezeNet est un réseau neuronal convolutif de 18 couches de profondeur, avec une taille d'entrée d'image de 227 par 227.

VGG16 est un modèle de réseau neuronal convolutif proposé par K. Simonyan et A. Zisserman de l'Université d'Oxford dans l'article intitulé "Very Deep Convolutional Networks for Large-Scale Image Recognition". Ce modèle atteint une précision de 92,7% dans le top 5 des tests d'ImageNet et est l'un des modèles les plus célèbres soumis à l'ILSVRC [14]. VGG-16 est un réseau de 16 couches de profondeur, avec une taille d'entrée d'image de 224 par 224.

Le réseau résiduel (ResNet50) a été développé par Kaiming He et al. Il a remporté l'ILSVRC 2015 en apprenant des représentations de fonctionnalités riches pour un large éventail d'images. ResNet-50 est un réseau neuronal convolutif de 50 couches de profondeur, avec une taille d'entrée d'image de 224 par 224.

Inception V3 est un modèle de reconnaissance d'images capable d'obtenir une précision significative. Il est le résultat de nombreuses idées développées par plusieurs chercheurs au fil des années. Il est basé sur l'article original intitulé "Rethinking the Inception Architecture for Computer Vision" de C. Szegedy et al . InceptionV3 est un réseau neuronal

convolutif de 48 couches de profondeur, avec une taille d'entrée d'image de 299 par 299. Le tableau 2.1 montre la comparaison entre les quatre architectures CNN.

Architecture	Nombre de couche	Taille d'entrée d'image
SqueezeNet	18	227*227*3
VGG16	16	224*224*3
InceptionV3	50	224*224*3
ResNet	48	299*299*3

TABLE *2.1 – Comparaison entre les quatre architectures CNN*

2.7 Modèles de détection d'objets

Les modèles de détection d'objets sont des outils puissants dans le domaine de la vision par ordinateur qui permettent de localiser et d'identifier différents objets dans des images ou des vidéos.

L'objectif principal des modèles de détection d'objets est de comprendre et d'analyser les informations visuelles contenues dans une scène afin d'identifier les objets qui y sont présents. Ils sont capables de détecter et de localiser différents types d'objets.

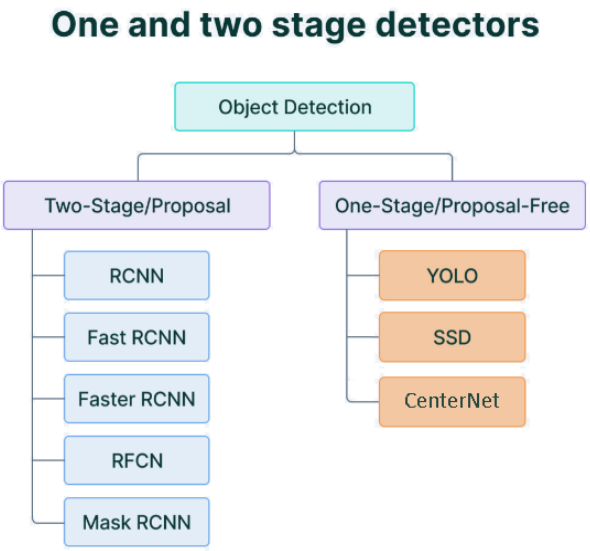


FIGURE *2.27 – Exemples des types de modèles de détection d'objets*

2.7.1 Modèles à un seul stage (Single-Stage Models)

Les modèles à un seul stage sont conçus pour détecter et classifier les objets en une seule passe, sans nécessiter d'étapes supplémentaires de proposition de régions d'intérêt.

Ils effectuent la détection directement sur la grille de localisation de l'image en utilisant des techniques telles que les ancres (anchors) ou les boîtes englobantes (bounding boxes) prédéfinies. Ces modèles effectuent généralement des prédictions pour chaque position de la grille et chaque ancre, en estimant la probabilité d'objet présent, les coordonnées de la boîte englobante et les classes associées. Les modèles à un seul stage, tels que YOLO (You Only Look Once) et SSD (Single Shot MultiBox Detector) et CenterNet, sont réputés pour leur vitesse de traitement rapide et leur capacité à détecter efficacement des objets de différentes tailles et formes.

SSD

Le SSD est un modèle de détection d'objets à un seul stage largement utilisé en vision par ordinateur. Il permet d'identifier et de localiser des objets spécifiques dans une image en une seule passe, sans nécessiter d'étapes supplémentaires de proposition de régions d'intérêt [14]

Tout d'abord, nous décrivons comment le SSD détecte les objets à partir d'une seule couche. En réalité, il utilise plusieurs couches (cartes de caractéristiques multi-échelles) pour détecter les objets de manière indépendante. À mesure que le CNN réduit progressivement la dimension spatiale, la résolution des cartes de caractéristiques diminue également. Le SSD utilise des couches de résolution plus basse pour détecter des objets à plus grande échelle. Par exemple, la figure 2.28 présente les cartes de caractéristiques de 4×4 sont utilisées pour les objets à plus grande échelle. Le SSD ajoute 6 couches de

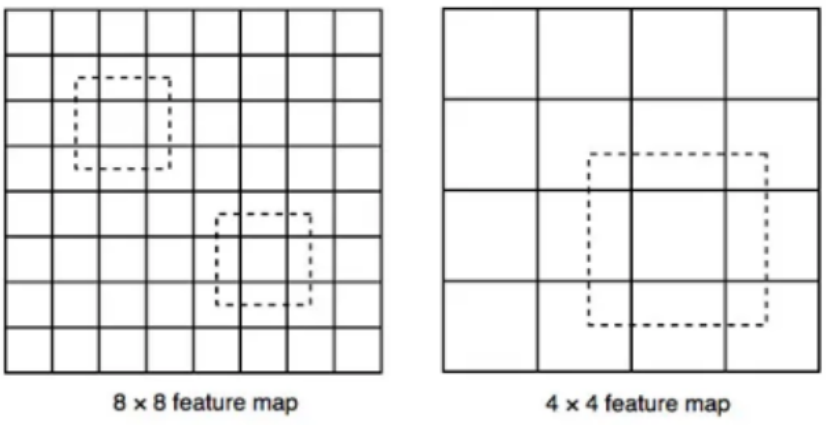


FIGURE *2.28 – Les cartes de caractéristiques de résolution inférieure (à droite) détectent des objets à plus grande échelle [33]*

convolution auxiliaires supplémentaires après le VGG16. Cinq d'entre elles seront ajoutées pour la détection d'objets. voir figure 2.29, Dans trois de ces couches, nous effectuons 6

prédictions au lieu de 4. Au total, le SSD effectue 8732 prédictions en utilisant 6 couches.

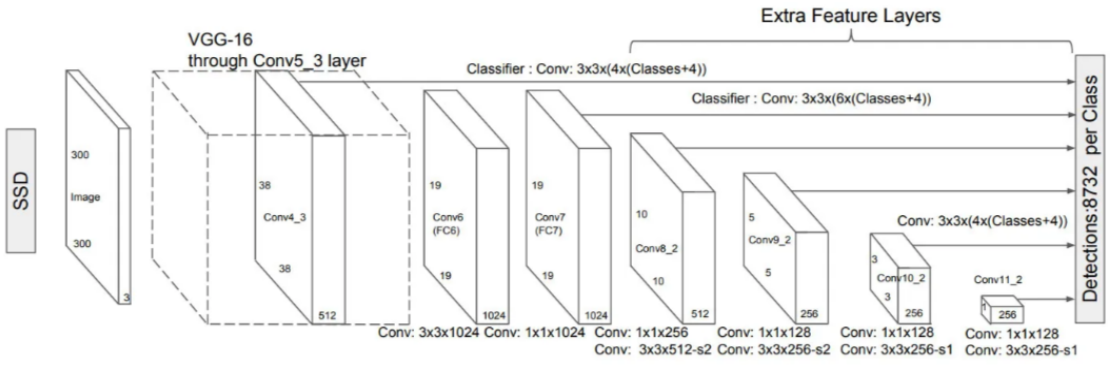


FIGURE *2.29 – SSD : Détecteur multi-boîtes en une seule passe*

Les cartes de caractéristiques multi-échelles (Multi-scale feature maps) améliorent considérablement la précision. la table 3.1 présente la précision obtenue en fonction du nombre de couches de cartes de caractéristiques utilisées pour la détection d'objets

Prediction source layers from						mAP		# Boxes
38 x 38	19 x 19	10 x 10	5 x 5	3 x 3	1 x 1	Yes	No	
x	x	x	x	x	x	74.3	63.4	8732
x	x	x				70.7	69.2	9864
	x					62.4	64.0	8664

TABLE *2.2 – Expérience avec des cartes de caractéristiques à échelles multiples*

En conclusion, le SSD est un modèle efficace et rapide pour la détection d'objets. Il offre une détection en une seule passe, une vitesse élevée et une précision satisfaisante pour de nombreux scénarios d'application. Le SSD a révolutionné la détection d'objets en combinant efficacement vitesse et précision.

CenterNet

CenterNet est un modèle de détection d'objets qui se distingue par son approche novatrice basée sur la prédiction des centres d'objets. Au lieu de se concentrer sur la détection de boîtes englobantes, CenterNet vise à localiser précisément les centres des objets dans une image [21]

FIGURE *2.30 – CenterNet régresse les centres des objets ainsi que leurs attributs (largeur et hauteur pour la détection d'objets en 2D) [34]*

Dans le modèle de détection d'objets CenterNet, plusieurs éléments clés sont utilisés pour localiser et décrire les objets détectés.

Tout d'abord, il utilise une carte de chaleur (heatmap) pour prédire les centres des objets avec précision.

Chaque pixel de la carte de chaleur indique la probabilité qu'il soit le centre d'un objet.

Ensuite, CenterNet prédit la taille et la forme des objets en utilisant des informations de taille (object size).

Cela permet d'estimer la taille exacte des objets détectés. Enfin, le modèle utilise des décalages locaux (local offset) pour ajuster les coordonnées des boîtes englobantes par rapport aux centres détectés, ce qui permet une localisation plus précise des objets dans l'image.voir la figure 2.30, Ensemble, ces éléments, la carte de chaleur, la taille des objets et les décalages locaux, contribuent à la précision et à l'exactitude de la détection d'objets dans le modèle CenterNet. CenterNet traite les entrées en passant l'image par un réseau de neurones convolutifs pour extraire des caractéristiques, puis utilise ces caractéristiques pour prédire la carte de chaleur (heatmap head), la taille des objets (Dimension head) et les décalages locaux (Offset Head). L'architecture de traitement du modèle CenterNet est présentée dans la figure 2.31. Ces informations sont ensuite utilisées pour extraire les résultats de détection d'objets voir la figure 2.31.

Le backbone (feature extractor) du modèle CenterNet est constitué d'un réseau de neurones convolutifs pré-entraîné, tel que ResNet ou Hourglass ou DLA.Il est responsable de l'extraction des caractéristiques de l'image, ce qui contribue à la capacité de CenterNet à détecter et localiser précisément les objets.

En conclusion, le modèle CenterNet est une approche avancée et précise pour la détection d'objets, utilisant des cartes de chaleur, des informations sur la taille et les décalages locaux. Il offre une combinaison optimale de précision et d'efficacité, et peut être appliqué à diverses applications de vision par ordinateur.

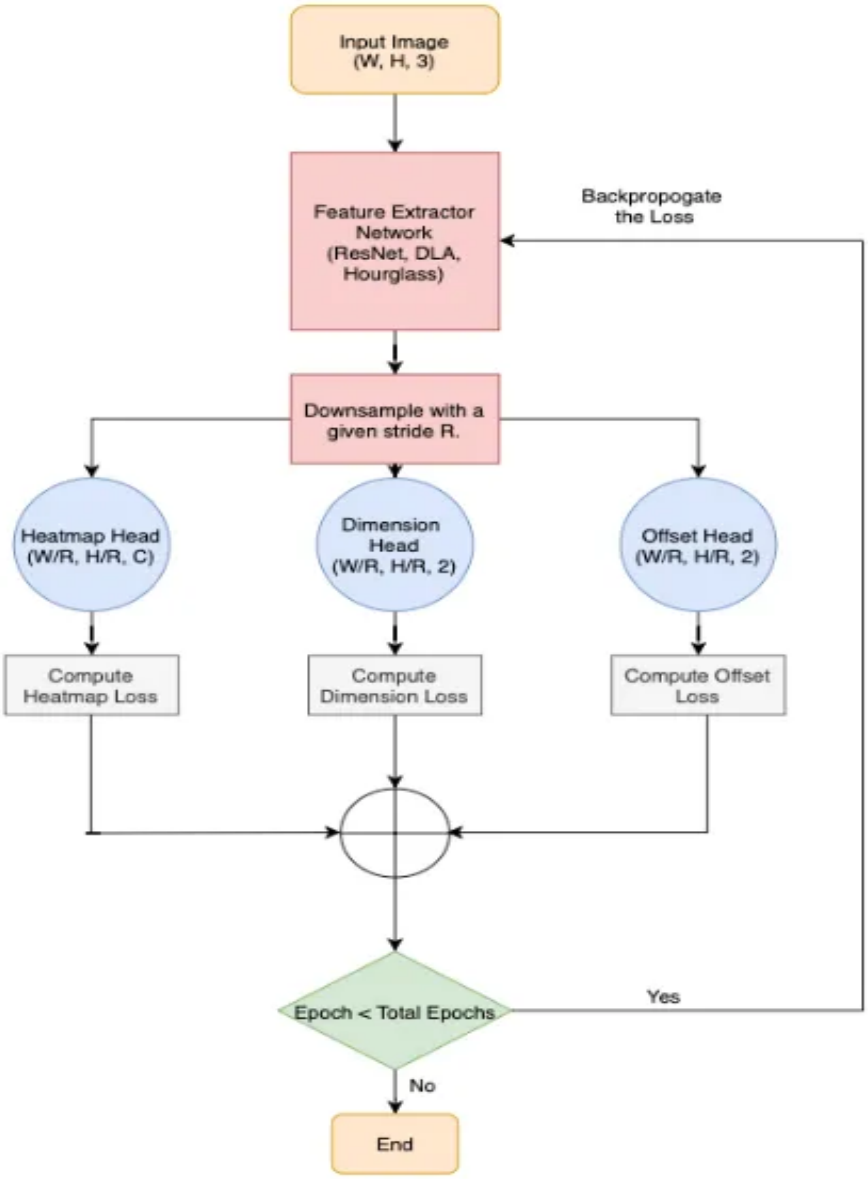


FIGURE 2.31 – *Schéma d'entraînement de CenterNet [34]*

2.7.2 Modèles à deux étapes (Two-Stage Models)

Les modèles à deux étapes sont basés sur une approche en cascade avec deux étapes principales. La première étape consiste à générer un ensemble de propositions de régions d'intérêt potentielles à partir de l'image en utilisant des techniques comme les régions d'intérêt (RoI) ou les régions de recherche (Region Proposal). Ces régions sont ensuite alignées et adaptées pour correspondre plus précisément aux objets présents dans l'image.

Dans la deuxième étape, les régions proposées sont classifiées et étiquetées pour déterminer la présence ou l'absence d'objets spécifiques, ainsi que leurs classes correspondantes. Les modèles à deux étapes, tels que R-CNN (Region-based Convolutional Neural Network), Fast R-CNN et Faster R-CNN, sont appréciés pour leur précision de détection élevée, en particulier pour les petits objets ou les scènes complexes.

Faster R-CNN

Faster R-CNN est un modèle de détection d'objets basé sur des réseaux de neurones convolutifs (CNN) qui propose une approche en deux étapes pour détecter et localiser les objets dans une image. Il utilise une région de proposition (Region Proposal Network - RPN) pour générer des régions d'intérêt potentielles, puis applique un réseau de neurones convolutifs pour classer et raffiner ces régions d'intérêt.voir figure 2.32. Faster R-CNN a introduit le concept d'apprentissage de la région de proposition, ce qui lui permet d'être plus rapide et plus précis par rapport aux approches précédentes. Il est largement utilisé dans les tâches de détection d'objets, offrant une performance élevée et une flexibilité dans la détection d'objets de différentes tailles et catégories.

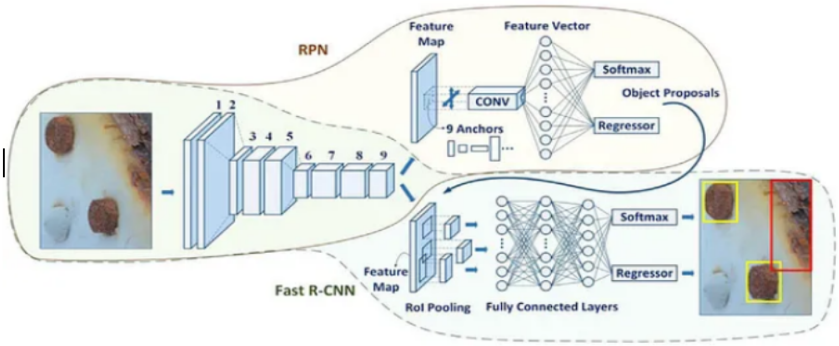


FIGURE *2.32 – Architecture Faster R-CNN [35]*

Le choix du backbone peut varier, mais des architectures populaires telles que VGG16, ResNet, et Inception sont couramment utilisées. figure 2.33 présente architecture de VGG16. Le backbone du modèle Faster R-CNN est responsable de l'extraction des caractéristiques

à partir de l'image en appliquant des opérations de convolution, de regroupement (pooling) et d'autres opérations non linéaires. Il s'agit d'une partie essentielle du modèle qui permet de capturer des informations de bas niveau et de haut niveau à partir de l'image, permettant ainsi une détection précise des objets.

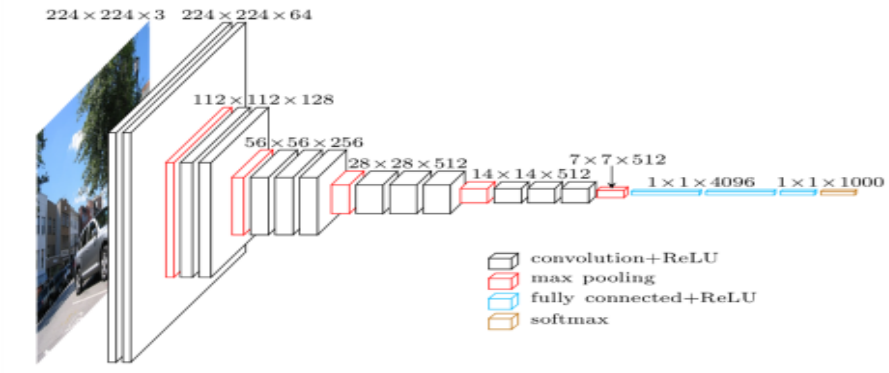


FIGURE *2.33 – Architecture VGG16. [36]*

En résumé, le modèle Faster R-CNN traite les données en utilisant un RPN pour générer des régions d'intérêt, puis utilise un réseau de neurones convolutifs pour classer et localiser précisément les objets dans ces régions. Cette approche en deux étapes permet au modèle de détecter efficacement les objets dans des images complexes tout en conservant une précision élevée.

2.8 Conclusion

En conclusion, ce chapitre a permis de mettre en évidence l'importance croissante du Deep Learning dans de nombreux domaines, notamment la vision par ordinateur et la détection d'objets. Les réseaux neuronaux convolutifs se sont avérés être une approche puissante pour extraire des caractéristiques significatives à partir d'images, permettant ainsi des tâches de classification et de localisation précises. Les méthodes de détection d'objets, telles que SSD, RCNN et centerNet, ont été présentées comme des avancées significatives dans le domaine de la détection précise et rapide des objets dans des images ou des vidéos. Chaque méthode a ses propres particularités et avantages, adaptés à des scénarios d'utilisation spécifiques. En résumé, ce chapitre a posé les bases pour la compréhension des concepts clés du Deep Learning, des réseaux neuronaux convolutifs et des méthodes de détection d'objets. Il ouvre la voie vers une analyse plus approfondie et des expérimentations futures dans votre mémoire.

Chapitre 3

Résultats et analyses

3.1 Introduction

Dans ce chapitre, nous verrons comment appliquer la détection d'objets à l'aide Deep Learning. Plus précisément, nous allons utiliser les algorithmes SSD-Mobilenet et Faster R-CNN-ResNet50 pour construire notre détecteur d'objets. A la fin, nous examinerons les résultats de l'application du détecteur à des exemples d'images d'entrées.

3.2 Outils et environement de développement

Notre projet de fin d'études portait sur la détection d'objets, et pour cela on a utilisé Python, TensorFlow et Keras comme principaux outils de développement voir figure 3.1 (a) (b) (c). Python, un langage de programmation polyvalent, et TensorFlow, une bibliothèque d'apprentissage automatique open-source, nous a permis de créer et d'entraîner efficacement des modèles de détection d'objets. On a également utilisé Keras, une bibliothèque haut niveau intégrée à TensorFlow, qui a simplifié la création et l'entraînement de réseaux de neurones convolutifs. Pour la réalisation de notre projet, On a utilisé Raspberry Pi 4 comme plateforme de déploiement, ce qui a permis une exécution en temps réel et une portabilité accrue voir figure 3.1 (d). L'ensemble de ces outils et environnements de développement nous a permis de développer un modèle de détection d'objets précis et efficace, tout en offrant une interface conviviale pour le développement et l'entraînement des modèles.

(a) Python [37]

(b) TensorFlow [38]

(c) Keras [39]

(d) Raspberry pi [27]

FIGURE *3.1 – Logos des outils et environnements utilisés*

3.3 Dataset utilisée

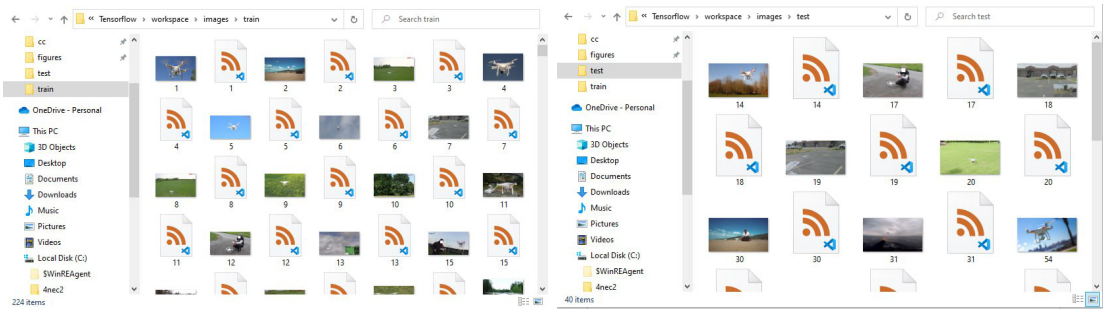
Dans le cadre de notre recherche sur la détection de drones, on a utilisé une dataset provenant de Kaggle, une plateforme en ligne populaire pour les ensembles de données et les compétitions en apprentissage automatique, voir la figure 3.2. La source de données provenant de Kaggle a fourni une base solide pour notre projet.

FIGURE *3.2 – Logo de Kaggle [40]*

L'avantage d'utiliser une dataset de Kaggle est qu'elle est souvent prétraitée et bien organisée, ce qui m'a permis de gagner du temps et de me concentrer davantage sur le développement de notre modèle. De plus, la possibilité de télécharger des ensembles de données provenant de Kaggle a élargi mes options et m'a offert une diversité d'exemples pour former notre modèle de détection de drones.

FIGURE *3.3 – Exemples d'images utilisées comme dataset [41]*

On a divisé notre ensemble de données pour la détection de drones en deux parties : un ensemble d'entraînement avec 128 images (80%) et un ensemble de test avec 26 images (20%) voir figure 3.4. L'ensemble d'entraînement a été utilisé pour former le modèle, tandis que l'ensemble de test a servi à évaluer ses performances. Cette division permet



Dossier d'entraînement. Dossier de test.

FIGURE 3.4 – *Les données utilisées dans l'entraînement et le test.*

d'évaluer la capacité du modèle à généraliser et à détecter les drones dans des situations réelles.

L'utilisation d'un ensemble d'entraînement plus grand (dans ce cas, 80% des images) permet au modèle d'apprendre à partir d'un plus grand nombre d'exemples et d'ajuster ses paramètres en conséquence. Cela peut améliorer les performances du modèle et sa capacité à capturer les relations entre les caractéristiques des données.

D'autre part, l'ensemble de test plus petit (dans ce cas, 20% des images) est utilisé pour évaluer les performances du modèle de manière indépendante, en mesurant son exactitude sur des données qu'il n'a pas vues auparavant. Cela permet de fournir une estimation réaliste de la performance du modèle sur de nouvelles données.

3.3.1 Pretraitement des donnees

L'image labeling est un processus d'attribution d'étiquettes ou d'annotations aux images pour fournir des informations spécifiques aux modèles d'apprentissage automatique. Cela permet aux modèles de détecter, de segmenter ou de classifier des objets ou des régions d'intérêt dans les images.Bien qu'il ait été nécessaire de consacrer du temps et des efforts à l'étiquetage manuel.

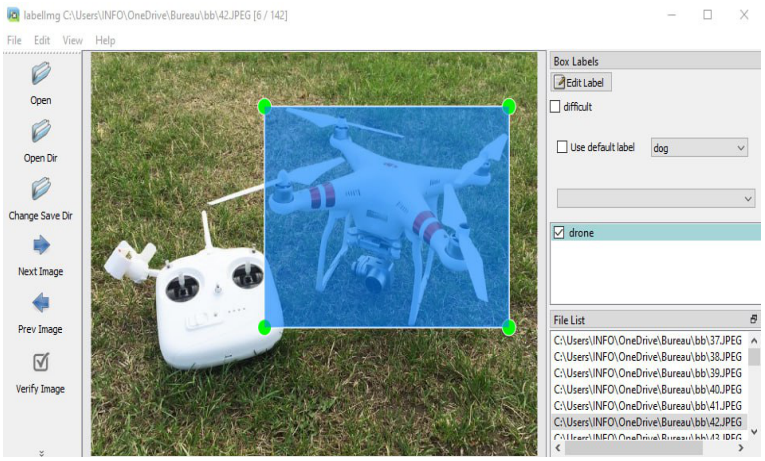


FIGURE *3.5 – Exemples d'images utilisées comme dataset*

Pour l'étiquetage manuel des images de drones, on a utilisé l'outil LabelImg voir la figure précédente. LabelImg est un logiciel open-source largement utilisé pour l'annotation d'images dans le domaine de la vision par ordinateur. Cet outil convivial offre une interface graphique intuitive permettant de marquer facilement les régions d'intérêt dans les images. Voici les étapes pour étiqueter les images de drones en utilisant l'outil LabelImg :

1. **Installation** : Commencez par installer l'outil LabelImg sur votre ordinateur.

```
Entrée [8]: if not os.path.exists(LABELIMG_PATH):
                !mkdir {LABELIMG_PATH}
                !git clone https://github.com/tzutalin/labelImg {LABELIMG_PATH}
            Cloning into 'Tensorflow\labelimg'...

Entrée [8]: if os.name == 'posix':
                !make qt5py3
            if os.name =='nt':
                !cd {LABELIMG_PATH} && pyrcc5 -o libs/resources.py resources.qrc

Entrée [*]: !cd {LABELIMG_PATH} && python labelImg.py
```

FIGURE *3.6 – Installation de l'outil LabelImg à l'aide de Jupyter Notebook*

2. **Chargement d'une image** : Dans LabelImg, cliquez sur "Open" ou "Ouvrir" pour charger une image à étiqueter depuis le dossier contenant vos images de drones.

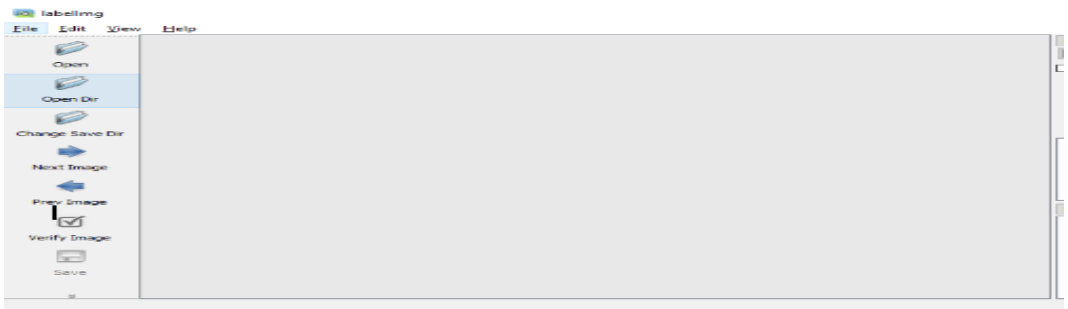


FIGURE *3.7 – Chargement d'une image*

3. **Annotation d'une région** : Utilisez la souris pour dessiner un rectangle de délimitation

autour du drone dans l'image. Ajustez la taille et la position du rectangle pour qu'il englobe précisément le drone.

FIGURE 3.8 – *Annotation d'une région à l'aide de LabelImg*

4. **Attribution d'une classe** : Une fois que vous avez annoté la région du drone, sélectionnez la classe correspondante dans la liste déroulante des classes prédéfinies (par exemple, "Drone" ou "Drones") dans l'interface LabelImg.

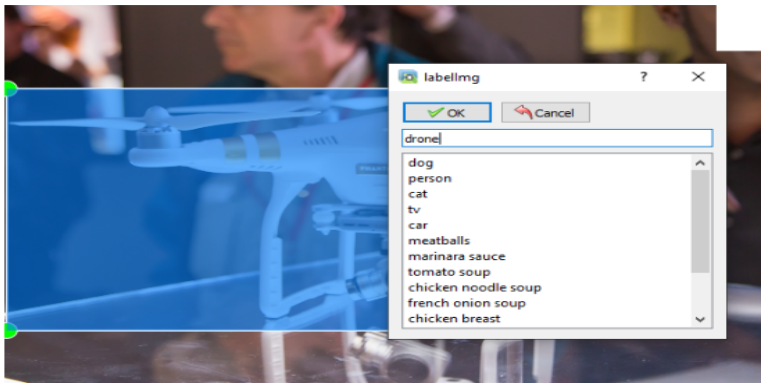


FIGURE 3.9 – *Attribution d'une classe à l'aide de LabelImg*

5. **Sauvegarde de l'annotation** : Cliquez sur le bouton "Save" ou "Enregistrer" pour sauvegarder l'annotation de l'image. L'outil LabelImg créera automatiquement un fichier XML ou un fichier de texte contenant les informations d'annotation correspondantes.

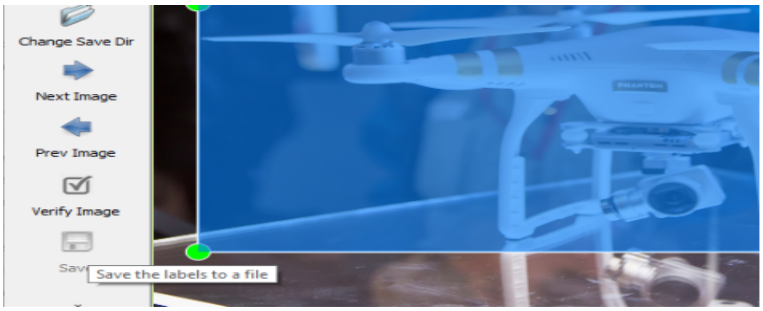


FIGURE 3.10 – *Sauvegarde de l'annotation dans LabelImg*

6. **Répétition du processus** : Répétez les étapes précédentes pour chaque image de votre dataset, en annotant toutes les régions de drones visibles.

3.4 Apprentissage et détection du drone

Durant notre projet, nous avons utilisé Jupyter Notebook comme IDE. et Jupyter Notebook est une application web open-source qui vous permet de créer et partager des documents contenant du code en direct, des équations, des visualisations et du texte explicatif. Il offre un environnement computationnel interactif où vous pouvez combiner l'exécution de code, la visualisation de données et le texte explicatif. Le nom "Jupyter" est un mélange des langages de programmation Julia, Python et R, qui étaient les langages originaux pris en charge par le projet.

FIGURE *3.11 – Logo de Jypter Notebook [42]*

Notre programme de détection d'objets repose sur une série d'étapes bien définies qui permettent d'identifier et de localiser des objets spécifiques dans des images ou des vidéos. En utilisant des techniques d'apprentissage automatique et des modèles de détection d'objets pré-entraînés, notre programme est capable de reconnaître et de classifier différentes classes d'objets, y compris les drones. voici les étapes clés de notre programme :

1. Configuration des paramètres et des fichiers essentiels du modèle de détection d'objets.

FIGURE *3.12 – Configuration des paramètres et des fichiers à l'aide de Jupyter Notebook*

2. Configuration des chemins et des répertoires nécessaires pour le modèle de détection d'objets.

```
Entrée [3]: paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
}
```

FIGURE *3.13 – Configuration des chemins et des répertoires à l'aide de Jupyter Notebook*

3. Création des répertoires nécessaires pour le modèle de détection d'objets.

```
Entrée [5]: for path in paths.values():
    if not os.path.exists(path):
        if os.name == 'posix':
            !mkdir -p {path}
        if os.name == 'nt':
            !mkdir {path}
```

FIGURE *3.14 – Création des répertoires à l'aide de Jupyter Notebook*

4. Installer Tensorflow Object Detection

```
Entrée [ ]: # Install Tensorflow Object Detection
if os.name=='posix':
    !apt-get install protobuf-compiler
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && cp object_detection/packages/tf2/

if os.name=='nt':
    url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-3.15.6-win64.zip"
    wget.download(url)
    !move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
    !cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip
    os.environ['PATH'] += os.pathsep + os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && copy object_detection\\packages\\t
    !cd Tensorflow/models/research/slim && pip install -e .
```

FIGURE *3.15 – Installation de Tensorflow Object Detection à l'aide de Jupyter Notebook*

5. Créer Label Map

```
Entrée [47]: labels = [{'name':'drone', 'id':1}]

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

FIGURE *3.16 – Création de Label Map à l'aide de Jupyter Notebook*

6. Créer TF records

```
Entrée [48]: # OPTIONAL IF RUNNING ON COLAB
ARCHIVE_FILES = os.path.join(paths['IMAGE_PATH'], 'archive.tar.gz')
if os.path.exists(ARCHIVE_FILES):
  !tar -zxvf {ARCHIVE_FILES}

Entrée [49]: if not os.path.exists(files['TF_RECORD_SCRIPT']):
    !git clone https://github.com/nicknochnack/GenerateTFRecord {paths['SCRIPTS_PATH']}

Entrée [50]: !python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {os.path.join(path
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {os.path.join(paths

Successfully created the TFRecord file: Tensorflow\workspace\annotations\train.record
Successfully created the TFRecord file: Tensorflow\workspace\annotations\test.record
```

FIGURE *3.17 – Création de TF records à l'aide de Jupyter Notebook*

7. Mettre à jour la configuration pour le transfert d'apprentissage.

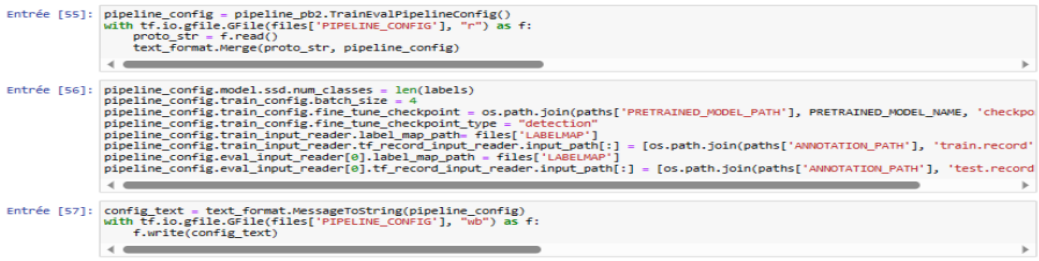


FIGURE *3.18 – Mise à jour de la configuration à l'aide de Jupyter Notebook*

8. Apprentissage du modèle

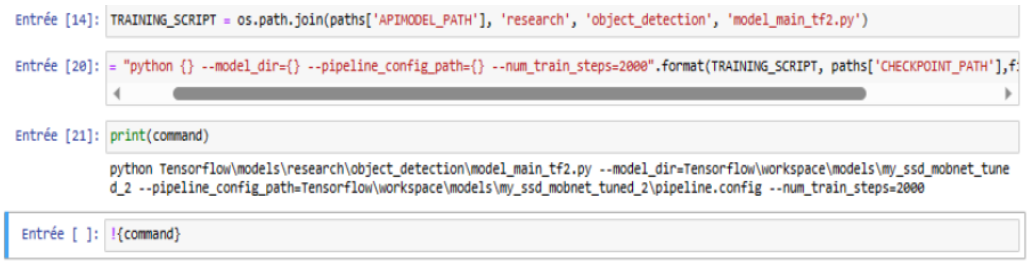


FIGURE *3.19 – Apprentissage du modèle à l'aide de Jupyter Notebook*

9. Evaluer le modèle

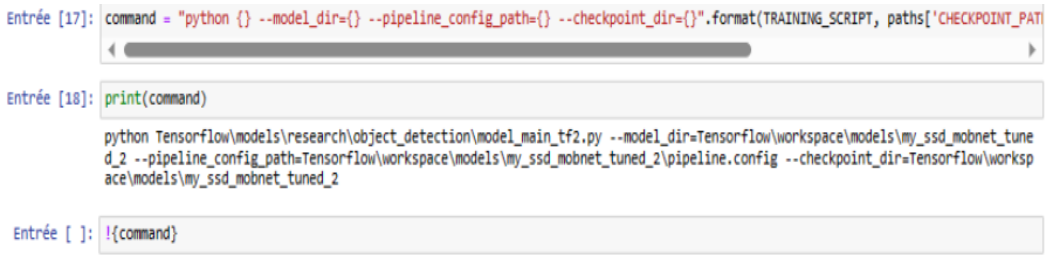


FIGURE *3.20 – Evaluation du modèle*

En conclusion, notre programme de détection d'objets offre une solution efficace pour identifier et localiser des objets spécifiques dans des images ou des vidéos. En utilisant des techniques d'apprentissage automatique et des modèles pré-entraînés, nous avons créé un système capable de détecter des drones avec précision

3.5 Perte dans la détection d'objet

Dans la détection d'objets, la perte joue un rôle essentiel pour évaluer la précision du modèle dans la localisation et la classification des objets. La perte est une mesure qui

quantifie l'écart entre les prédictions du modèle et les vérités terrain (ground truths) des objets. Elle guide l'optimisation du modèle en ajustant les poids pour minimiser cette erreur. En utilisant différentes composantes de perte telles que la perte de classification, la perte de localisation et la perte de régularisation.

3.5.1 La perte de classification

La perte de classification est une mesure utilisée pour évaluer la précision des prédictions de classification d'un modèle par rapport aux étiquettes réelles des classes. Elle quantifie l'écart entre les prédictions du modèle et les valeurs cibles. L'équation de la perte de classification :

$$L_{\text{classification}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(p_{ij}) \tag{3.1}$$

Cette équation calcule la perte de classification en utilisant la fonction de perte de l'entropie croisée. Elle mesure l'écart entre les prédictions du modèle et les valeurs cibles pour chaque classe et chaque échantillon. La somme des pertes pour tous les échantillons et toutes les classes est ensuite normalisée par le nombre total d'échantillons N.

3.5.2 La perte de localisation

La perte de localisation est généralement calculée en utilisant une distance ou une métrique appropriée pour mesurer l'écart entre les prédictions des boîtes englobantes (bounding boxes) et les boîtes englobantes réelles des objets. La perte de localisation, également connue sous le nom de perte de régression, mesure l'écart entre les prédictions de localisation et les véritables annotations de localisation dans un modèle de détection d'objets. La formule générale de la perte de localisation dépend du problème spécifique et de l'algorithme utilisé.

$$L_{\text{localisation}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{J=1}^{M} \text{Loss}(y_{i,j}, \hat{y}_{i,j}) \tag{3.2}$$

3.5.3 La perte de regularization

La perte de régularisation, également appelée perte de poids (weight regularization), est une composante ajoutée à la fonction de perte. L'équation générale pour calculer la perte de régularisation L2 est formulée comme suit :

$$L_{\text{regularization}} = \frac{\lambda}{2} \sum_{i} ||\mathbf{w}_i||^2 \tag{3.3}$$

Elle vise à prévenir le surajustement (overfitting) en limitant les valeurs extrêmes des poids et en favorisant des solutions plus simples.

3.5.4 La perte total

La perte totale (total loss) est la somme des différentes composantes de perte dans un modèle d'apprentissage automatique. Elle représente l'erreur globale du modèle lors de son entraînement et est utilisée pour guider l'optimisation du modèle.

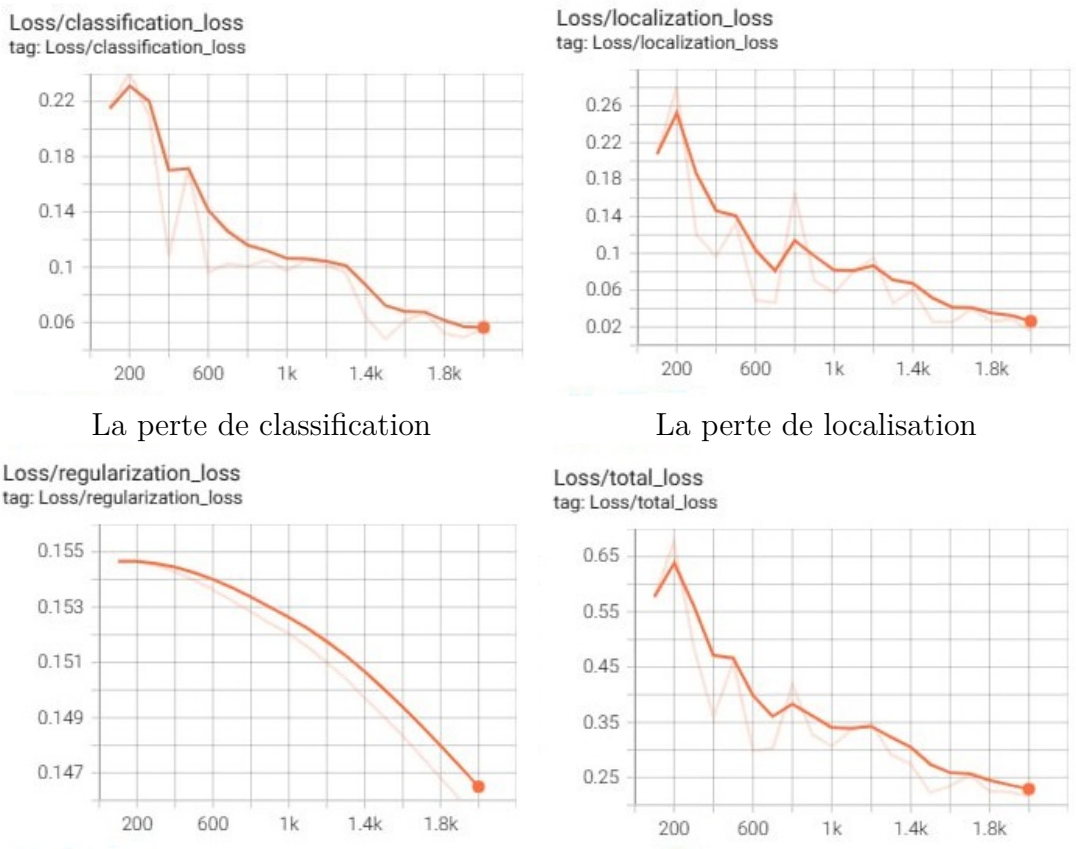
$$L_{\text{total}} = L_{\text{localisation}} + L_{\text{classification}} + L_{\text{regularization}} \tag{3.4}$$

La perte totale est utilisée pour ajuster les poids du modèle pendant l'entraînement, en minimisant l'erreur globale et en améliorant les performances du modèle sur les données d'entraînement. L'objectif est d'obtenir une perte totale minimale, ce qui indique que le modèle s'ajuste efficacement aux données et est capable de généraliser correctement sur de nouvelles données.

3.6 Modèle SSD-Mobilenet

Dans le cadre de notre projet de détection de drones, nous avons décidé d'explorer et d'évaluer l'utilisation du modèle SSD MobileNet pour cette tâche. L'objectif est de tester les performances de ce modèle dans la détection précise et rapide des drones dans des images ou des vidéos.

Après l'entraînement du modèle, nous avons réalisé des tests et des évaluations approfondies pour mesurer ses performances. Nous avons utilisé des images et des vidéos de différents scénarios, comprenant diverses conditions d'éclairage, des angles de vue variés et des situations réalistes de présence de drones voici les résultats que nous avons obtenu :



La perte de classification | La perte de localisation

La perte de régularisation | La perte totale

FIGURE *3.21 – Analyses des pertes dans le modèles SSD-Mobilenet*

La perte de classification mesure l'écart entre les prédictions du modèle et les étiquettes réelles des classes. Un résultat de 0.05 après 2000 steps indique de bonnes performances, avec une précision de classification relativement élevée.

La perte de localisation mesure la précision de localisation du modèle. Un résultat de 0.025 après 2000 steps indique de bonnes performances, avec une précision de localisation relativement élevée.

La perte de régularisation mesure l'ajustement et la généralisation du modèle. Un résultat de 0.1465 après 2000 steps indique une régularisation adéquate, favorisant un modèle plus robuste et moins susceptible de surajustement.

La perte totale est la somme des différentes composantes de perte, telles que la perte de classification, la perte de localisation et la perte de régularisation.0.225 après 2000 steps représente l'erreur globale accumulée par le modèle.

Pas	Perte de classification	Perte de localisation	Perte de régularisation	Perte totale
600	0.14	0.1	0.154	0.40
1k	0.11	0.08	0.152	0.35
1.4k	0.9	0.07	0.150	0.30
2k	0.05	0.03	0.147	0.23

TABLE *3.1 – Analyses quantitative des graphiques des modèles SSD-Mobilenet*

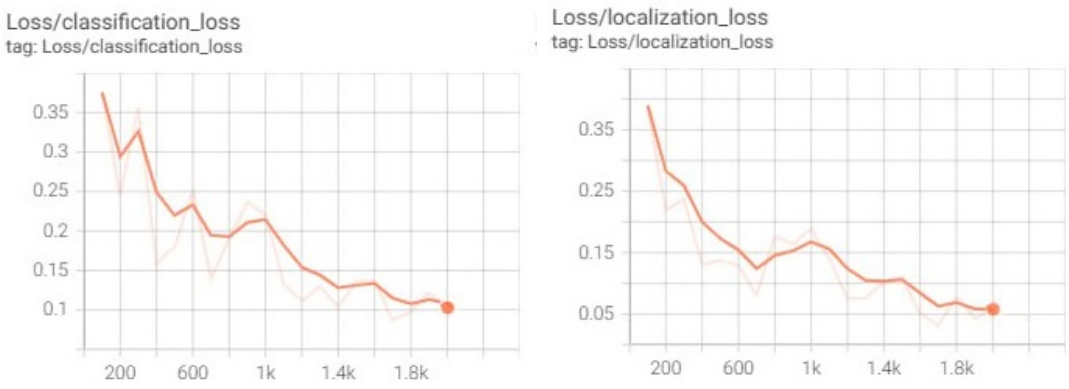
FIGURE *3.22 – Résultats obtenue du modèles SSD-Mobilenet*

Sur la première photo de détection de drone, le modèle a obtenu une précision de 98% pour détecter la présence de drones. Quant à la deuxième photo, le modèle a obtenu une précision de 97% pour la détection de drones. Une précision de 98% et 97% est un résultat solide, témoignant de la capacité du modèle à repérer efficacement les drones dans des scénarios variés.

3.7 Modèle Faster R-CNN-ResNet50

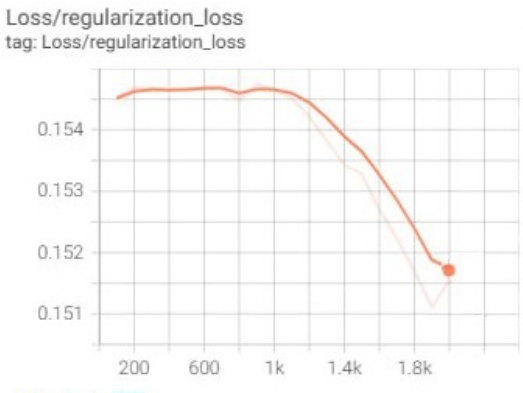
Dans le cadre de notre projet de détection de drones, nous avons exploré et évalué l'utilisation du modèle Faster R-CNN avec l'architecture ResNet50. Notre objectif était de tester les performances de ce modèle en termes de détection précise et rapide des drones dans des images et des vidéos.

Après avoir entraîné le modèle, nous avons procédé à des tests et à des évaluations approfondies pour évaluer ses performances. Nous avons utilisé différentes images et vidéos représentant une variété de scénarios, comprenant des conditions d'éclairage variées, des angles de vue différents et des situations réalistes impliquant la présence de drones.Les résultats obtenus ont été encourageants, démontrant la capacité du modèle Faster R-CNN avec l'architecture ResNet50 à détecter avec précision les drones dans différents contextes, voici les résultats que nous avons obtenu :

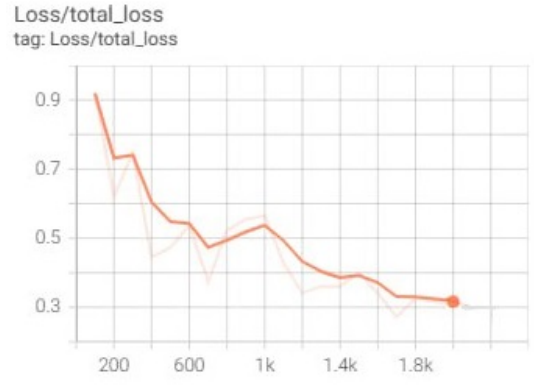


La perte de classification

La perte de localisation



La perte de régularisation



La perte totale

FIGURE *3.23 – Analyses des pertes dans le modèles Faster RCNN ResNet50*

La perte de classification mesure l'écart entre les prédictions du modèle et les étiquettes réelles des classes. Un résultat de 0.11 après 2000 steps indique de bonnes performances, avec une précision de classification relativement élevée.

La perte de localisation mesure la précision de localisation du modèle. Un résultat de 0.055 après 2000 steps indique de bonnes performances, avec une précision de localisation relativement élevée.

La perte de régularisation mesure l'ajustement et la généralisation du modèle. Un résultat de 0.1526 après 2000 steps indique une régularisation adéquate, favorisant un modèle plus robuste et moins susceptible de surajustement.

La perte totale est la somme des différentes composantes de perte, telles que la perte de classification, la perte de localisation et la perte de régularisation.0.31 après 2000 steps représente l'erreur globale accumulée par le modèle.

Pas	Perte de classification	Perte de localisation	Perte de régularisation	Perte totale
600	0.24	0.15	0.155	0.53
1k	0.21	0.16	0.155	0.53
1.4k	0.13	0.1	0.154	0.39
2k	0.1	0.06	0.151	0.3

TABLE *3.2 – Analyses quantitative des graphiques des modèles Faster R-CNN ResNet50*

FIGURE *3.24 – Résultats obtenue du modèles Faster R-CNN ResNet50*

Sur la première photo de détection de drone, le modèle a obtenu une précision de 92% pour détecter la présence de drones. Quant à la deuxième photo, le modèle a obtenu une précision de 90% pour la détection de drones. Une précision de 92% et 90% est un résultat solide, témoignant de la capacité du modèle à repérer efficacement les drones dans des scénarios variés.

3.8 Comparaison de la méthode SSD et Faster R-CNN

Dans cette section, nous avons mené une étude comparative entre deux méthodes de détection d'objets par Deep Learning : SSD et Faster R-CNN. L'objectif était de démontrer l'efficacité de ces méthodes et d'évaluer leurs performances. Nous avons constaté que la méthode SSD est plus rapide et plus fiable que Faster R-CNN pour les applications en temps réel. En effet, la méthode SSD permet de détecter et d'entraîner des objets en même temps, ce qui la rend plus efficace pour les applications en temps réel. En revanche, Faster R-CNN nécessite un temps d'exécution plus élevé, ce qui la rend moins adaptée pour les applications en temps réel. En conclusion, la méthode SSD est plus rapide, plus fiable et plus efficace que Faster R-CNN pour la détection d'objets par Deep Learning.

3.9 Conclusion

Dans ce troisième chapitre de notre mémoire, nous avons abordé les outils et l'environnement de développement utilisés dans notre projet, tels que Python, Raspberry Pi, Keras et TensorFlow. Nous avons ensuite présenté la source de notre jeu de données sur les drones, provenant de Kaggle, et expliqué le processus d'étiquetage manuel des images.

Ensuite, nous avons comparé les performances des modèles RCNN (Régions avec Convolutional Neural Network) et SSD (Single Shot MultiBox Detector) dans la détection des drones. Après une évaluation approfondie, nous avons constaté que le modèle SSD offre de meilleurs résultats de détection des drones par rapport au modèle RCNN. Cette conclusion met en évidence l'efficacité et la précision du modèle SSD dans cette tâche spécifique.

Ces résultats sont d'une grande importance, car ils montrent que le modèle SSD est plus adapté à la détection des drones dans les images ou les vidéos. Cela suggère que le SSD pourrait être un choix optimal pour des applications pratiques nécessitant une détection précise et rapide des drones.

En conclusion, ce chapitre nous a fourni une compréhension approfondie des outils et de l'environnement de développement utilisés, de la préparation des données, de l'étiquetage manuel et des performances comparatives des modèles RCNN et SSD dans la détection des drones. Ces informations sont essentielles pour la suite de notre projet de détection des drones et contribuent à l'avancement des connaissances dans ce domaine.

Conclusion Générale

Dans l'ensemble, les trois chapitres de ce mémoire ont permis d'explorer et de présenter différentes facettes de la détection d'objets et de l'apprentissage en profondeur.

Dans le premier chapitre, nous avons mis en évidence le potentiel de l'apprentissage en profondeur et des systèmes embarqués pour révolutionner le domaine de la vision par ordinateur. Nous avons souligné l'importance de ces technologies pour la détection d'objets en temps réel dans diverses applications, et nous avons noté qu'il reste encore beaucoup à explorer et à améliorer dans ce domaine.

Le deuxième chapitre nous a permis de plonger plus en profondeur dans le fonctionnement des réseaux neuronaux convolutifs et des méthodes de détection d'objets, telles que SSD, RCNN et centerNet. Nous avons analysé les outils et l'environnement de développement utilisés, la préparation du jeu de données du drone et l'étiquetage manuel des images. En comparant les performances des modèles RCNN et SSD, nous avons conclu que le modèle SSD offre de meilleurs résultats de détection des drones.

Enfin, dans le troisième chapitre, nous avons examiné les résultats obtenus en utilisant le modèle SSD pour la détection des drones. Nous avons constaté une précision élevée dans nos tests, confirmant l'efficacité du modèle dans cette tâche spécifique. Ces résultats démontrent l'importance du choix du modèle et l'impact des différentes étapes du processus, de la préparation des données à l'évaluation des performances.

En conclusion, ce mémoire a contribué à notre compréhension des concepts clés, des outils et des méthodes de détection d'objets dans le contexte de la détection des drones. Il a posé les bases pour de futures recherches et développements, offrant des perspectives intéressantes pour l'amélioration des performances des modèles et leur déploiement dans des applications pratiques.

Pour terminer, nous espérons que ce travail servira de référence et d'inspiration pour d'autres chercheurs et praticiens qui s'intéressent à la détection d'objets et à l'apprentissage en profondeur, ou plus spécifiquement à la détection des drones.

Bibliographie

[1] C. Tang, Y. Ling, X. Yang, W. Jin, and C. Zheng, "Multi-view object detection based on deep learning," *Appl. Sci.*, vol. 8, no. 9, 2018.

[2] A. c. Saaod M .Rasheed1, "Deep learning," *International Journal of Innovative Research in Science Engineering and Technology*, May 2019.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *IEEE Conf Comput Vis Pattern Recognit*, pp. I–511, Feb 2001.

[4] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, pp. 1627–1645, Sept. 2010.

[5] A. IA, "Segmentation et détection d'objets en temps réel avec tensorflow." https://www.actuia.com/contribution/jeancharlesrisch/segmentation-et-detection-dobjets-en-temps-reel-avec-tensorflow/, October 26 2019. Consulté le 13 juin 2023.

[6] "ICA2IT 2019 : International Conference on Artificial Intelligence and Information Technology." http://wikicfp.com/cfp/servlet/event.showcfp?eventid=77191&copyownerid=119134, March 2019. Consulté le 13 juin 2023.

[7] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pp. 886–893 vol. 1, June 2005.

[8] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, pp. 273–297, September 1995.

[9] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, p. 91, November 2004.

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation." arXiv, October 22 2014. Consulté le 14 juin 2023.

[11] R. Girshick, "Fast r-cnn." arXiv, September 27 2015. Consulté le 14 juin 2023.

[12] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN : Towards real-time object detection with region proposal networks." arXiv, January 6 2016. Consulté le 14 juin 2023.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once : Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, June 2016.

[14] W. Liu and et al., "SSD : Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 21–37, 2016.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

[16] G. Huang, Z. Liu, L. van der Maaten, and K. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.

[17] A. G. Howard and et al., "Mobilenets : Efficient convolutional neural networks for mobile vision applications." arXiv, April 16 2017. Consulté le 14 juin 2023.

[18] M. Tan and Q. V. Le, "Efficientnet : Rethinking model scaling for convolutional neural networks." arXiv, September 11 2020. Consulté le 14 juin 2023.

[19] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2999–3007, October 2017.

[20] A. IA, "Deformable convolutional networks," pp. 764–773, October 2017. Consulté le 13 juin 2023.

[21] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Objects as points," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[22] M. Barr and A. Massa, *Programming Embedded Systems : With C and GNU Development Tools*. O'Reilly Media, Inc., 2006.

[23] A. B. Tucker, ed., *Computer Science Handbook*. Boca Raton, FL : Chapman & Hall/CRC, 2nd ed., 2004.

[24] J. W. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[25] B. Daniel, "What are embedded systems ?." https://www.trentonsystems.com/blog/what-are-embedded-systems. Consulté le 2 juillet 2023.

[26] "Raspberry pi documentation - raspberry pi hardware." https://www.raspberrypi.com/documentation/computers/raspberry-pi.html. Consulté le 3 juillet 2023.

[27] Raspberry Pi Foundation, "Raspberry pi 4 model b." https://www.raspberrypi.com/products/raspberry-pi-4-model-b/?variant=raspberry-pi-4-model-b-8gb. Accessed on [Insert Date].

[28] F. Chollet, *Deep Learning with Python*. Manning Publications, 2017.

[29] S. Raschka and V. Mirjalili, *Python Machine Learning : Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing, 3rd ed., 2019.

[30] E. Davalo and P. Naïm, *Des réseaux de neurones*. Edition Eyrolle, 2 ed., 1990.

[31] M. Sauget, *Parallélisation de problèmes d'apprentissage par réseaux neuronaux artificiels. Application en radiothérapie externe*. PhD thesis, Université de Franche-Comté, 2007.

[32] A. Burkov, *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.

[33] Jonathan Hui, "Ssd : Single shot multibox detector for real-time processing." https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8c Accessed on 2nd July 2023 at 17 :36.

[34] VisionWizard, "Centernet : Objects as points - a comprehensive guide." https://medium.com/visionwizard/centernet-objects-as-points-a-comprehensive-guide-2ed9993c48bc. Accessed on 2nd July 2023 at 17 :31.

[35] Towards Data Science, "Faster r-cnn object detection." https://towardsdatascience.com/faster-rcnn-object-detection-f865e5ed7fc4. Accessed on 2nd July 2023 at 17 :26.

[36] Towards Data Science, "Step-by-step vgg16 implementation in keras for beginners." https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c. Accessed on 2nd July 2023 at 17 :26.

[37] "Python logo." https://www.python.org/community/logos/. Accessed on [Insert Date].

[38] Google, "Tensorflow." https://www.tensorflow.org/?hl=fr. Accessed on [Insert Date].

[39] Keras Team, "Keras." https://keras.io/. Accessed on [Insert Date].

[40] "Kaggle." https://www.kaggle.com/. Accessed on [Insert Date].

[41] Dasmehdixtr, "Drone dataset (uav)." https://www.kaggle.com/datasets/dasmehdixtr/drone-dataset-uav. Accessed on [Insert Date].

[42] "Jupyter." https://jupyter.org/. Accessed on [Insert Date].