**R**épublique **A**lgérienne **D**émocratique et **P**opulaire
**M**inistère de l'**E**nseignement **S**upérieur et de la **R**echerche **S**cientifique

**U**NIVERSITE **M**OULOUD **M**AMMERI DE **T**IZI-**O**UZOU

**F**ACULTE DU **G**ENIE **E**LECTRIQUE ET D'**I**NFORMATIQUE
**DEPARTEMENT D'AUTOMATIQUE**

# Mémoire de Fin d'Etudes de MASTER ACADEMIQUE

Domaine : **Sciences et Technologies**
Filière : **Automatique**
Spécialité : **Automatique et Informatique Industrielle**

*Présenté par*
**KADIR Sarah**
**MOUDOUD Anya**

**Thème**

# Détection d'objets en utilisant les réseaux de neurones convolutifs

*Mémoire soutenu publiquement le 30/09/2024 devant le jury composé de :*

**M Youcef MESSAR**
MAA, UMMTO, Président

**Mlle Farida DORBANE**
MAA, UMMTO, Encadrant

**Mme Sadia ALKAMA**
MCA, UMMTO, Examinateur

**M Rabah MELLAH**
Professeur, UMMTO, Examinateur

# Remerciements

Je tiens à exprimer ma profonde gratitude envers toutes les personnes qui ont contribué à l'aboutissement de ce mémoire et, plus largement, à l'ensemble de mon parcours académique.

Tout d'abord, un immense merci à mes parents, dont le soutien indéfectible et l'encouragement constant ont permis d'aplanir les difficultés et d'atteindre mon objectif.

Je souhaite également adresser mes sincères remerciements à mon encadrant, en l'occurrence, Mlle DORBANE Farida, dont la guidance précieuse, les critiques constructives et la disponibilité ont été déterminants dans la réalisation de ce travail.

Je tiens aussi à exprimer ma gratitude à Mme NAIT BELAID Ouiza et M MAIDI Ahmed, dont l'aide et les contributions ont joué un rôle essentiel dans la réalisation de ce mémoire.

Enfin, je remercie chaleureusement mon binôme, KADIR Sarah, pour sa collaboration féconde. Notre travail d'équipe a été une source d'inspiration et d'apprentissage.

À mes enseignants et à toutes personnes qui, par leur présence, leur aide ou leurs conseils, ont rendu ce projet possible, je dédie ces remerciements avec toute ma reconnaissance.

**MOUDOUD Anya**

# Remerciements

Je tiens à exprimer ma profonde gratitude envers toutes les personnes qui ont contribué à l'aboutissement de ce mémoire et, plus largement, à l'ensemble de mon parcours académique.

Tout d'abord, un immense merci à mes parents, dont le soutien indéfectible et l'encouragement constant ont permis d'aplanir les difficultés et d'atteindre mon objectif.

Je souhaite également adresser mes sincères remerciements à mon encadrant, en l'occurrence, Mlle DORBANE Farida, dont la guidance précieuse, les critiques constructives et la disponibilité ont été déterminants dans la réalisation de ce travail.

Je tiens aussi à exprimer ma gratitude à Mme NAIT BELAID Ouiza et M MAIDI Ahmed, dont l'aide et les contributions ont joué un rôle essentiel dans la réalisation de ce mémoire.

Enfin, je remercie chaleureusement mon binôme, MOUDOUD Anya, pour sa collaboration féconde. Notre travail d'équipe a été une source d'inspiration et d'apprentissage.

À mes enseignants et à toutes personnes qui, par leur présence, leur aide ou leurs conseils, ont rendu ce projet possible, je dédie ces remerciements avec toute ma reconnaissance.

**KADIR Sarah**

# Dédicace

À mes parents, à ma famille, à mes amis et à toutes personnes ayant pavé le chemin de mes réussites, ce mémoire est le fruit de votre foi en moi et de votre précieuse contribution à ce chapitre de ma vie. À vous, je dédie cette réalisation avec une gratitude infinie.

**MOUDOUD Anya**

# Dédicace

À mes parents, à ma famille, à mes amis et à toutes personnes ayant pavé le chemin de mes réussites, ce mémoire est le fruit de votre foi en moi et de votre précieuse contribution à ce chapitre de ma vie. À vous, je dédie cette réalisation avec une gratitude infinie.

**KADIR Sarah**

# Table des matières

# Table des figures

# Liste des tableaux

# Introduction générale

L'intelligence artificielle (IA) a connu une croissance exponentielle au cours des dernières décennies, révolutionnant divers domaines grâce à des avancées notables dans l'apprentissage automatique (*machine learning*) et l'apprentissage profond (*deep learning*).

Parmi les technologies les plus influentes dans ces disciplines, les réseaux de neurones artificiels (RNA), qui jouent un rôle crucial en permettant aux machines d'apprendre et de traiter de grandes quantités de données, de détecter des motifs complexes et de prendre des décisions de manière autonome et précise.

Ce mémoire s'appuie sur cette base théorique pour explorer les réseaux de neurones convolutifs (CNN), une catégorie des RNA spécialement conçue pour traiter des données structurées en grille, telles que les images. Ces réseaux ont révolutionné la vision par ordinateur en permettant une localisation et une classification d'objets avec une précision remarquable. L'évolution des CNN a donné naissance à des algorithmes de détection d'objets avancés, dont l'algorithme YOLO (*You Only Look Once*).

YOLO est une méthode de détection d'objets particulièrement efficace car elle permet de localiser et de classifier plusieurs objets dans une image en une seule passe, ce qui améliore considérablement la vitesse et la précision de la détection.

Dans ce projet, nous mettrons l'accent sur l'utilisation de YOLOv4, une version améliorée de l'algorithme YOLO et analyserons en profondeur ses performances en matière de détection d'objets sur 4 bases d'images différentes.

Ce mémoire est structuré en quatre chapitres, organisés comme suit :

Dans un premier temps, nous aborderons les réseaux de neurones artificiels (RNA) en tant que fondement théorique et pratique de l'apprentissage profond. Nous discuterons dans ce chapitre de l'évolution historique des RNA et analyserons également leurs différents types, leurs mécanismes d'apprentissage, les algorithmes qui les régissent et leurs propriétés, ainsi que leurs applications variées dans des secteurs tels que la vision par ordinateur, la robotique ou encore la santé.

Ensuite, nous nous intéresserons aux réseaux de neurones convolutifs (CNN), conçus pour surmonter les limites des RNA dans l'analyse des images et des vidéos. Grâce à l'utilisation d'opérations de convolution, ces réseaux réalisent des progrès significatifs dans la classification des images. Ce chapitre détaillera les principes fondamentaux des CNN, leur architecture, leurs mécanismes d'apprentissage et leurs types tout en soulignant leur avantages et inconvénients.

Dans le troisième chapitre, nous aborderons la détection d'objets ; un domaine incontournable de la vision par ordinateur. Les progrès technologiques récents, notamment avec l'apparition des réseaux de neurones convolutifs, ont permis d'améliorer la précision et la vitesse des algorithmes de détection d'objets. Ce chapitre abordera l'évolution des méthodes, en commençant par les techniques classiques, pour finir sur les approches modernes basées sur les CNN, tels que YOLO.

Le dernier chapitre de ce mémoire est dédié à la mise en œuvre pratique du réseau YOLOV4, un modèle de détection d'objets en temps réel utilisé pour sa rapidité et sa précision. Nous décrirons l'architecture de ce réseau et détaillerons les différentes étapes de son implémentation et analyserons les résultats obtenus. Ce chapitre se conclura par une discussion sur les perspectives d'amélioration pour surmonter les défis rencontré lors de l'implémentation de YOLOV4.

Enfin, nous terminerons ce mémoire par une conclusion récapitulant les principales idées développées et une réflexion sur les perspectives futures.

# Chapitre 1

# Réseaux de neurones artificiels

## 1.1 Introduction

Les réseaux de neurones artificiels (RNA) constituent un sous-ensemble de l'apprentissage automatique (*machine learning*) et sont au cœur des algorithmes de l'apprentissage en profondeur (*deep learning*). Leur nom et leur structure sont inspirés du cerveau humain.

Les réseaux de neurones artificiels sont nés de la curiosité scientifique et de la nécessité de résoudre des problèmes complexes. Une fois correctement entraînés, ces réseaux peuvent apprendre par eux-mêmes et s'adapter en permanence pour produire des données de sortie de plus en plus précises.

Au cours de ce chapitre, nous explorerons leurs principes fondamentaux, en commençant par une brève présentation de leur évolution historique. Nous parlerons ensuite de l'analogie entre les neurones biologiques et les neurones artificiels, nous aborderons les réseaux de neurones artificiels en expliquant leur fonctionnement et leurs types, nous explorerons également leurs règles d'apprentissage et discuterons des différentes propriétés de ces derniers.

Enfin, nous aborderons leurs applications actuelles dans divers domaines, tels que la santé, la robotique et la vision par ordinateur.

## 1.2 Historique

Les premiers modèles de neurones artificiels ont été introduits pour la première fois en 1943 par le neurophysicien Warren McCulloch et le mathématicien Walter Pitts. Ils proposent un modèle mathématique du neurone artificiel inspiré directement du neurone biologique. Il s'agit d'un modèle à une ou plusieurs entrées et une seule sortie binaire (la sortie s'active (=1) si la somme pondérée des entrées dépasse un certain seuil).

Ensuite, en 1949, Donald Hebb, un physiologiste américain, publie son livre "*The Organization of Behavior*" (l'organisation du comportement) où il évoque une règle d'apprentissage ; " Règle de Hebb" qui est toujours utilisée dans les techniques d'apprentissage des réseaux de neurones.

En 1957, Frank Rosenblatt, un psychologue américain développe le modèle du Perceptron, en mettant en œuvre les résultats des recherches précédentes. C'est un modèle composé de deux couches de neurones : une couche de perception (d'entrée) capable de réaliser des fonctions (logiques, arithmétiques), et une couche de prise de décision (de sortie). C'est le premier modèle à apprendre par expérience [1].

En 1960, Bernard Widrow et Marcian Hoff développent le modèle ADALINE (*ADAptive LINear Element*). Sa structure est semblable à celle du modèle du Perceptron mais il possède une loi d'apprentissage différente de celle de ce dernier. Il est à la base des réseaux de neurones multicouches.

En 1969, Marvin Minsky et Seymour Papert publient l'ouvrage "Perceptron" où ils démontrent les limitations du modèle du Perceptron. Suite à la publication de ce livre, plusieurs chercheurs se réorientent vers d'autres domaines plus pertinents (à savoir l'intelligence artificielle), et l'intérêt au réseaux de neurones baisse.

Certains de ces chercheurs poursuivent les recherches dans ce domaine, comme le physicien américain John Joseph Hopfield qui publie en 1982 un nouveau modèle de réseaux de neurones qui porte son nom. Cette découverte relance l'intérêt des réseaux de neurones.

En 1986, l'apprentissage par l'algorithme de rétro-propagation, adapté au Perceptron multicouches, voit le jour grâce aux travaux de Rumelhart, Hinton et Williams.

Depuis, ce domaine a connu beaucoup de progrès et ne cesse d'évoluer, ce qui explique son large usage dans différents domaines.

## 1.3 Neurone biologique

Un neurone biologique ou une cellule nerveuse, est une cellule excitable constituant l'unité de base du système nerveux. Il joue un rôle essentiel dans la transmission électrique et chimique des signaux nerveux. Certains neurones génèrent des influx nerveux déclenchés en réponse à des stimulus alors que d'autres jouent le rôle de station de relais où les impulsions sont transmises, et parfois redirigées [2]. Le système nerveux contient entre 100 millions et 100 milliards de neurones interconnectés. Les parties du système nerveux sont réparties en : système nerveux central et en système nerveux périphérique. Certains neurones sont sensitifs (afférents) et transmettent des influx vers le système nerveux central, d'autres sont moteurs (efférents) et transmettent des influx depuis le système nerveux central, et certains sont

mixtes [2]. Le neurone biologique est constitué d'un corps cellulaire, dans lequel se trouve un noyau, des dendrites, un axone et une synapse, comme le montre la figure 1.1 :

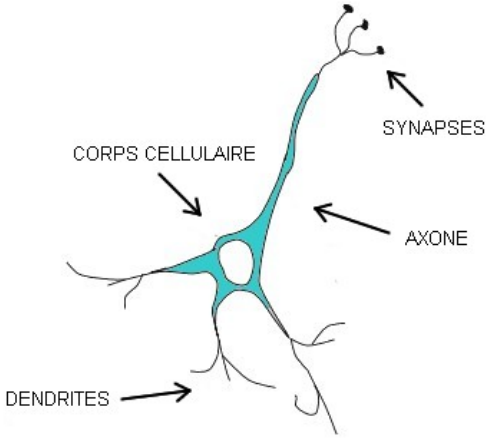



FIGURE 1.1 – Neurone biologique

* **Corps cellulaire :** Il s'agit de la partie centrale du neurone, il contient le cytoplasme et le noyau. C'est l'endroit dans la cellule où sont produits les matériaux nécessaires à la création de nouveaux prolongements dendritiques et axonaux lors de la phase embryonnaire. Il participe généralement lui aussi directement à la réception de l'information par les autres neurones [3].
* **Axone :** Chaque cellule nerveuse n'a qu'un seul axone. Il transporte l'influx nerveux né dans le corps cellulaire. Il est habituellement plus long que les dendrites, atteignant parfois 100 cm.
* **Dendrites :** Ils constituent les nombreux petits prolongements qui reçoivent des influx nerveux et les transportent vers d'autres corps cellulaires. Ils ont la même structure que les axones, mais ils sont habituellement plus courts et ramifiés. Dans les neurones moteurs, ils forment une partie des synapses et dans les neurones sensitifs, ils forment les récepteurs sensitifs qui répondent aux stimulus [2].
* **Synapse :** Il s'agit d'une zone située entre deux neurones (cellules nerveuses) et assurant la transmission des informations de l'une à l'autre [4].

Le neurone a la capacité de créer un phénomène électrique connu sous le nom d'influx nerveux. Il permet de transmettre un message, information sensitive ou ordre moteur. Les messages arrivent à un neurone donné par ses dendrites. Celles-ci amènent les messages au corps cellulaire qui les analysent et en produit de nouveaux, lesquels cheminent le long de l'axone. Un neurone est ainsi relié à d'autres neurones ou à des cellules musculaires. Le point de jonction entre l'axone d'un neurone et les dendrites d' un autre neurone s'appelle la synapse [4].

## 1.4 Neurone artificiel

Un neurone artificiel est une unité de traitement de l'information utilisée dans les réseaux de neurones artificiels. Tout comme les neurones biologiques, ces neurones reçoivent des signaux d'entrée (données), effectuent des calculs pour les traiter, puis produisent des signaux de sortie (données) comme montré sur la figure 1.2. Les poids et les biais associés aux entrées ainsi que les fonctions d'activation utilisées par chaque neurone constituent les éléments d'un neurone artificiel (paramètres internes).

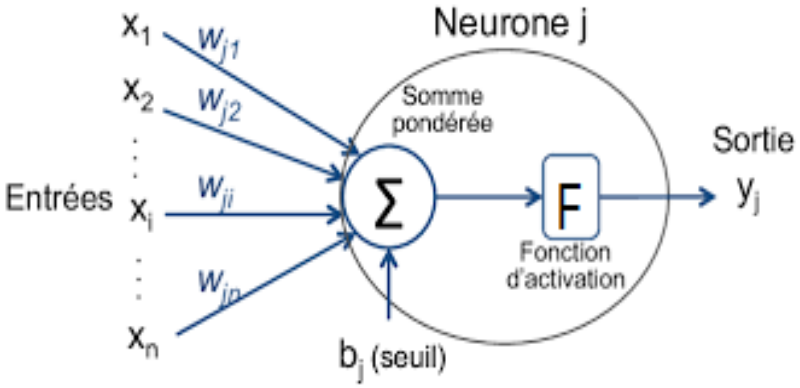



FIGURE 1.2 – Neurone artificiel

La sortie du neurone artificiel est donnée par la relation suivante :

$$Y_j = F(\sum_{i=1}^{n} x_i.W_{ji} + b_j) \tag{1.1}$$

Avec :

* $Y_j$ : sortie du neurone artificiel.
* $F$ : fonction d'activation.
* $x_i$ : entrée i du neurone artificiel.
* $W_{ji}$ : poids de connexion entre le neurone j et l'entrée i.
* $b_j$ : biais.

### 1.4.1 Poids et biais

Les connexions entre les neurones ne sont pas toutes égales en termes d'influence. Chaque connexion est associée à un poids spécifique (w), qui détermine l'importance de l'information transmise d'un neurone à un autre. Les poids sont continuellement ajustés par le réseau de neurones pour améliorer la précision des résultats.

Parallèlement, les biais sont des paramètres supplémentaires (valeurs numériques) ajoutés à chaque couche dans un réseau de neurones pour ajuster les valeurs d'entrée après l'application des poids. Ils permettent d'influencer le comportement des fonctions d'activation des neurones.

Les biais permettent au modèle de décider à quel niveau la fonction d'activation devrait être activée ou désactivée (changer le point de basculement (*threshold*) de la fonction), indépendamment des valeurs d'entrée spécifiques, ajoutant ainsi une flexibilité dans les prédictions du modèle et une capacité d'adaptation aux données d'entrée.

### 1.4.2 Fonction d'activation

La fonction d'activation est une formule mathématique utilisée par chaque neurone pour normaliser les données d'entrée qu'il reçoit avant de les transmettre ou non. Cette fonction est activée lorsque la valeur calculée dépasse un seuil prédéfini. Si la valeur ne dépasse pas ce seuil, la fonction d'activation reste inactive, et aucune donnée n'est transmise, ce qui est similaire au fonctionnement des neurones biologiques.

Les fonctions d'activation permettent d'introduire de la non linéarité au modèle, sans laquelle ce dernier serait incapable de représenter des relations complexes et non linéaires entre ses entrées et sorties.

Différentes fonctions d'activation peuvent être utilisées dans un réseau de neurones, telles que la fonction sigmoïde, l'unité linéaire rectifiée (ReLU), la fonction tangente hyperbolique (tanh(x)) et la fonction softmax qui sont résumées dans le tableau 1.1 :

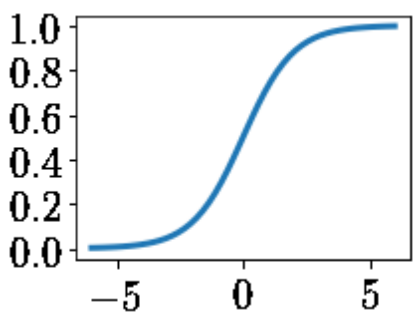
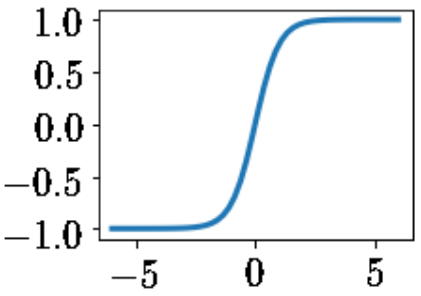
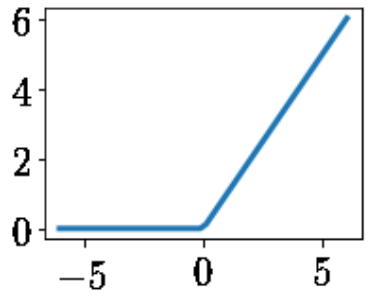
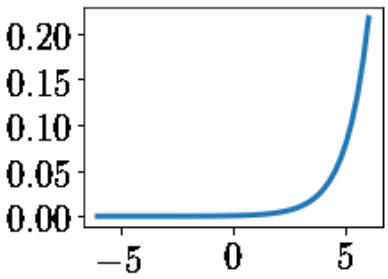
| Fonction | Formule | Description | Allure |
|---|---|---|---|
| Sigmoïde | $f(x) = \frac{1}{1+e^{-x}}$ | C'est la plus populaire, elle est dérivable, d'autant plus que sa dérivée est simple à calculer ce qui permet la réduction du temps d'apprentissage. Elle est utilisée pour normaliser les valeurs entre 0 et 1. |   |
| Tanh (tangente hyperbolique) | $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ | Elle est similaire à la fonction sigmoïde, mais elle a une plage de valeurs plus large allant de -1 à 1.Elle est utilisée pour introduire de la non-linéarité dans les réseaux de neurones et permet d'apprendre des modèles et des relations complexes. |   |
| ReLu (Rectified Linear Unit) | $f(x) = \begin{cases} x & si, x \geq 0 \\ 0 & sinon \end{cases}$ | Elle effectue du filtrage sur les données ; redresse les valeurs négatives à zéro, tout en conservant les valeurs positives. Elle est couramment utilisée dans les couches cachées. |   |
| Softmax | $softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$ | Elle transforme un vecteur de valeurs en un vecteur de probabilité, dont la somme totale des probabilités sera égale à 1. Elle est utilisée dans la couche de sortie du réseau de classification multi-classes, où chaque instance doit appartenir à une seule classe parmi plusieurs. |   |

TABLE 1.1 – Fonctions d'activations les plus utilisées

## 1.5 Analogie entre le neurone biologique et le neurone artificiel

Il existe pour chaque élément composant le neurone biologique, un élément correspondant dans le neurone artificiel, comme le montre le tableau 1.2 :

| Neurone biologique | Neurone artificiel |
|---|---|
| Dendrites | Entrées du signal |
| Synapses | Poids de connexions |
| Noyau | Fonction d'activation |
| Axones | Sorties du signal |

TABLE 1.2 – Analogie entre le neurone biologique et le neurone artificiel

## 1.6 Réseau de neurones artificiels

Un réseau de neurones artificiels est un modèle mathématique inspiré du fonctionnement du cerveau humain. Il est composé de plusieurs neurones interconnectés, organisés en couches successives : une couche d'entrée qui reçoit les données, une ou plusieurs couches cachées où s'effectuent les calculs intermédiaires, et une couche de sortie qui produit le résultat final. Cette architecture est illustrée par la figure 1.3.

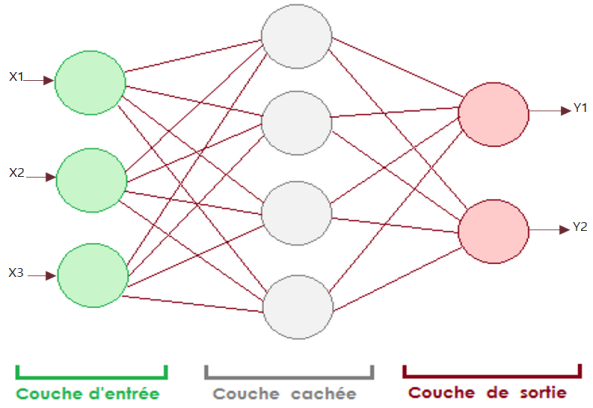



FIGURE 1.3 – Réseau de neurones artificiel

Les neurones de la couche d'entrée reçoivent les données initiales, qui sont multipliées par les poids attribués aux connexions reliant ces neurones à ceux des couches suivantes. Les valeurs obtenues sont ensuite additionnées pour former une somme pondérée, à laquelle un biais est ajouté. La nouvelle valeur obtenue passe par une fonction d'activation, qui détermine

si le neurone doit s'activer et transmettre l'information à la couche suivante. Ce processus se répète à chaque couche, jusqu'à ce que le réseau atteigne la couche de sortie, où la décision finale est prise.

## 1.7 Types de réseaux de neurones

Un réseau de neurones est composé de couches. En fonction de la présence ou de l'absence des couches cachées, un réseau de neurones peut être monocouche ou multicouches.

* **Monocouche :** c'est le type le plus simple des réseaux de neurones. Les neurones de la couche d'entrée sont reliés directement à ceux de la couche de sortie. Cela signifie qu'il n'y a pas de couches cachées entre elles, comme le montre la figure1.4 :

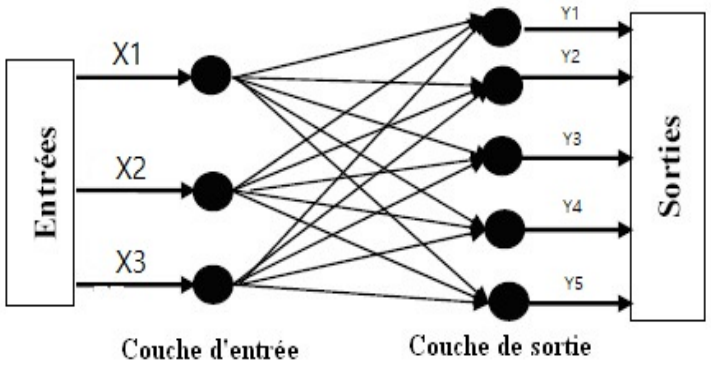



FIGURE 1.4 – Réseau de neurones monocouche

* **Multicouches :** Un réseau de neurones multicouches, comme le montre la figure 1.5, possède, en plus d'une couche d'entrée et d'une couche de sortie, au moins une ou plusieurs couches cachées. L'ajout des couches cachées augmente la capacité d'apprentissage d'un réseau de neurones.

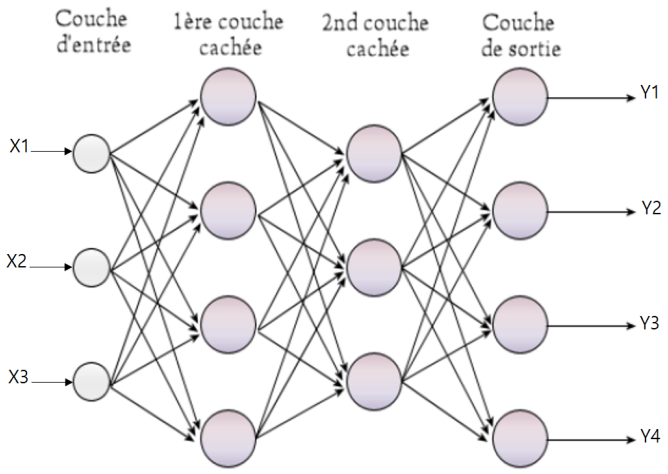



FIGURE 1.5 – Réseau de neurones multicouches

En fonction du sens de la propagation de l'information, les réseaux de neurones peuvent être composé de 2 familles : des réseaux de neurones non récurrents (non bouclés) et des réseaux de neurones récurrents (bouclés).

### 1.7.1 Réseaux de neurones non récurrents

Les réseaux de neurones non récurrents (*feed-forward*) sont un type de réseau de neurones artificiels où les données circulent dans un seul sens, de l'entrée vers la sortie, sans boucle de rétroaction comme le montre la figure 1.6 ; ils ne peuvent pas stocker d'informations sur les entrées précédentes. Dans ce réseau, chaque couche de neurones est connectée à la couche suivante pour former un flux de données unidirectionnel, les entrées et les sorties sont toutes deux des vecteurs de données indépendantes du temps. Les opérations dans chaque neurone sont effectuées instantanément d'où l'appellation "Réseaux de neurones statiques". Les réseaux de neurones à propagation avant sont utilisés pour résoudre des problèmes tels que la reconnaissance de formes, la classification d'images et la prédiction de valeurs numériques. Parmi ces réseaux, on trouve : Le Perceptron monocouche et le Perceptron multicouches (MLP).

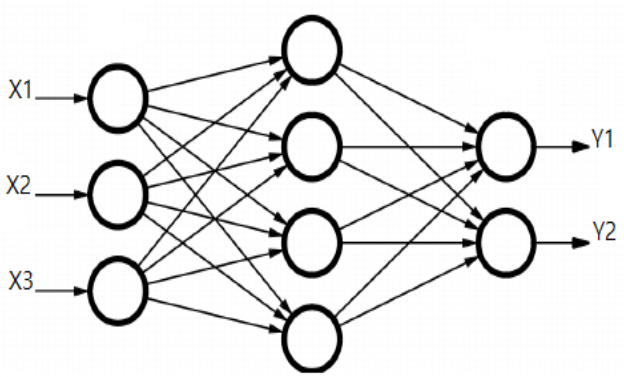



FIGURE 1.6 – Réseau de neurones non récurrent

### 1.7.2 Réseaux de neurones récurrents

Les réseaux de neurones récurrents (*back-forward*) comme illustré dans la figure 1.7, sont des réseaux de neurones dans lesquels l'information peut se propager dans les deux sens, se rapprochant ainsi du vrai fonctionnement du système nerveux, qui n'est pas à sens unique. Ils utilisent des boucles récurrentes qui leur permettent de conserver et de traiter des informations sur les états précédents et de les utiliser pour influencer les calculs futurs. Les réseaux de neurones récurrents sont adaptés pour traiter des données séquentielles en prenant en compte leur ordre et leur contexte temporel d'où l'appellation "Réseaux de neurones dynamiques". Ce type de réseaux offre des avantages pour le traitement des données séquentielles et est utilisé dans de nombreux domaines tels que la traduction automatique, la reconnaissance vocale, la génération de texte, et bien d'autres [5].

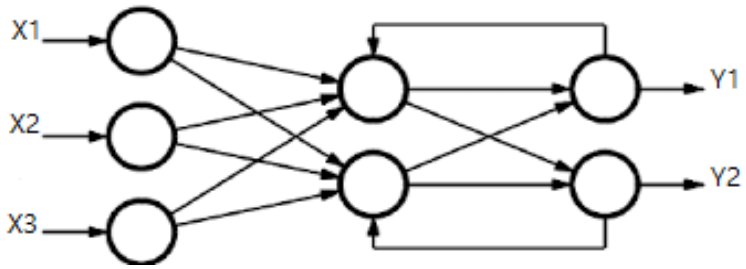



FIGURE 1.7 – Réseau de neurones récurrent

Parmi ces réseaux, on trouve le réseau Hopfield. Il s'agit d'un réseau où chaque neurone est connecté à tous les autres neurones du réseau avec des connexions bidirectionnelles. Il est constitué d'une seule couche qui représente à la fois l'entrée du réseau et sa sortie. La règle d'apprentissage proposée par Hopfield est basée sur la régle de Hebb. La figure 1.8 représente un réseau Hopfield avec 6 neurones complètement interconnectés [6] :

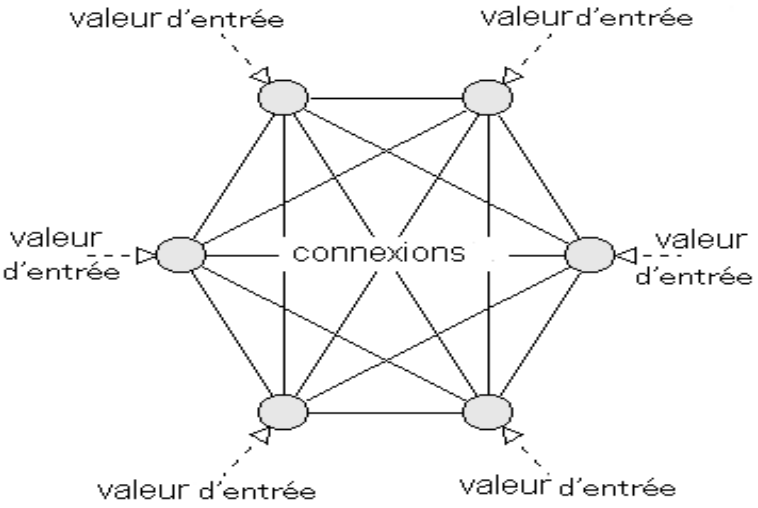



FIGURE 1.8 – Réseau Hopfield

Les réseaux de neurones non récurrents (non bouclés) réalisent des fonctions non linéaires alors que les réseaux récurrents (bouclés), réalisent des équations aux différences (ou équations récurrentes) non linéaires.

## 1.8 Apprentissage des réseaux de neurones

L'apprentissage est la caractéristique la plus intéressante des réseaux de neurones. Il s'agit de la phase d'adaptation des paramètres internes du réseau (poids et biais) jusqu'à l'obtention du comportement désiré en sortie.

Ce processus nécessite deux éléments : des exemples et un algorithme d'apprentissage.

* **Exemples :**Également appelés échantillons d'apprentissage ; ce sont les données utilisées pour entraîner le réseau de neurones. Une fois les poids initialisés en leur attribuant des valeurs aléatoires, ces exemples sont présentés au réseau et leurs sorties correspondantes sont calculées. Par la suite, une valeur d'erreur est calculée et en fonction de cette valeur, les poids seront ajustés afin de la minimiser.
* **Algorithmes d'apprentissage :** Ce sont les méthodes utilisées pour effectuer l'opération d'apprentissage des réseaux de neurones.

### 1.8.1 Types d'apprentissage

En fonction des échantillons d'apprentissage, on distingue 3 types d'apprentissage : supervisé, non supervisé et semi supervisé.

#### 1.8.1.1 Apprentissage supervisé

Dans ce cas, les échantillons d'apprentissage sont présentés au réseau sous forme de couples du type : (valeur d'entrée-valeur de sortie correspondante). L'apprentissage supervisé consiste à calculer les sorties du réseau de neurones à partir des valeurs d'entrées introduites et à comparer ces sorties obtenues à celles désirées, puis à ajuster les poids des connexions du réseau jusqu'à ce que l'écart entres les valeurs des deux sorties soit réduit.

#### 1.8.1.2 Apprentissage non supervisé

Dans ce cas, les exemples ne contiennent que des valeurs d'entrées, sans leurs valeurs de sorties associées. Le réseau classe les données en groupes (classes) en se basant sur la similarité de leurs caractéristiques. Ainsi, le réseau apprend tout seul (auto-adaptation) à générer des sorties proches de toutes valeurs d'entrées de mêmes natures.

#### 1.8.1.3 Apprentissage semi supervisé

Dans le cadre de l'apprentissage semi supervisé, on présente au réseau une majorité de données non étiquetées et une minorité qui est étiquetée. C'est donc une approche hybride de l'apprentissage supervisé et non supervisé. Cette technique traite les données différemment en fonction de leur étiquetage ou non. Les données étiquetées sont traitées de la même manière que dans le cas de l'apprentissage supervisé (calculer la sortie à partir des données introduites, calculer l'erreur, et ajuster les poids en fonction de la valeur de cette dernière), tandis que pour les données non étiquetées, un autre traitement est nécessaire ; une fois que le modèle est entraîné sur les données étiquetées, il est utilisé pour prédire les étiquettes des données non étiquetées. Ces prédictions approximatives sont ensuite utilisées pour étendre l'ensemble de données en ajoutant les nouvelles étiquettes prédites aux données non étiquetées.

L'apprentissage semi-supervisé est basé sur l'hypothèse que les données non étiquetées contiennent des informations utiles pour améliorer les performances du modèle. En exploitant ces informations, le modèle peut apprendre des caractéristiques plus robustes et généralisables.

### 1.8.2 Règles d'apprentissage

L'apprentissage est soumis à des règles bien définies qui garantissent la réussite de ce processus. Parmi ces règles on cite : la règle de Hebb, la règle de Widrow-Hoff, et la règle de rétropropagation.

#### 1.8.2.1 Règle de Hebb

C'est une règle élaborée par D.Hebb en 1949. Cette théorie est souvent résumée par la formule : « des neurones qui s'excitent ensemble se lient entre eux » (« *cells that fire together, wire together* ») [7]. Cela signifie que lorsque deux neurones sont excités simultanément, il se crée ou renforce un lien les unissant (voir figure 1.9). Elle est souvent utilisée pour expliquer l'apprentissage associatif (la formation des associations entre des stimulus ou des événements qui se produisent simultanément ou successivement).

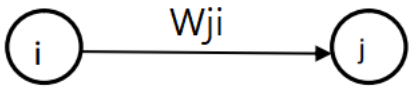



FIGURE 1.9 – Connexion entre deux neurones i et j

Cet algorithme consiste à modifier les poids dans le cas où une erreur existe, afin d'adapter la sortie obtenue à celle souhaitée suivant l'équation suivante :

$$W_{ji}(t+1) \quad = \quad W_{ji}(t) + \eta a_i a_j \tag{1.2}$$

Avec :

* $W_{ji}(t)$ : poids de connexion entre le neurone j et le neurone i à l'instant t (ancienne valeur du poids).
* $W_{ji}(t+1)$ : poids de connexion entre le neurone j et le neurone i à l'instant t+1 (nouvelle valeur du poids).
* $\eta$ : pas d'apprentissage (contrôle la vitesse à laquelle les poids sont mis à jour).$\eta \in ]0,1]$
* $a_i, a_j$ : respectivement les valeurs d'activation des neurones i et j.

#### 1.8.2.2 Règle de Widrow-Hoff

Établie par B.Widrow et M.Hoff en 1960, elle est applicable aux réseaux de neurones à apprentissage supervisé. Elle est basée sur le modéle du Perceptron de Rosenblatt, qui est très proche de la règle de Hebb, la grande différence étant qu'il tient compte de l'erreur observée en sortie. Son équation est donc donnée comme suit :

$$W_{ji}(t+1) \quad = \quad W_{ji}(t) + \eta(Y_t - Y)x_i \tag{1.3}$$

Avec :

* $W_{ji}(t)$ : poids de connexion entre le neurone j et le neurone i à l' instant t (ancienne valeur du poids).
* $W_{ji}(t+1)$ : poids de connexion entre le neurone j et le neurone i à l' instant t+1 (nouvelle valeur du poids).
* $\eta$ : pas d'apprentissage (contrôle la vitesse à laquelle les poids sont mis à jour).$\eta \in ]0,1]$
* $Y_t$ : sortie désirée.
* $Y$ : sortie calculée.
* $x_i$ : valeur de sortie de la $i^{ème}$ cellule d'entrée.

#### 1.8.2.3 Règle de rétropropagation

Apparue pour la première fois en 1986, cette règle s'applique pour les réseaux de neurones multicouches à apprentissage supervisé. Tout comme les autres règles, elle vise à minimiser l'erreur entre la sortie calculée et celle désirée. Elle effectue cette tâche en ajustant les poids de manière itérative en allant de la dernière couche (couche de sortie) jusqu'à la première (couche d'entrée), d'où son appellation (propagation vers l'arrière). Elle est basée sur la technique de la descente de gradient.

* **Descente de gradient :**C'est un algorithme d'optimisation, utilisé pour trouver les valeurs optimales d'une fonction en minimisant son erreur. L'idée principale de la descente de gradient est de trouver la direction dans laquelle la fonction de perte diminue le plus rapidement. Pour cela, l'algorithme calcule le gradient de la fonction de perte par rapport aux paramètres du modèle (dans notre cas, les paramètres sont les poids, les activations et les biais). Le gradient représente la direction et l'amplitude du taux de changement de la fonction de perte par rapport à chaque paramètre puis utilise cette information pour mettre à jour itérativement les valeurs des paramètres du modèle, en se déplaçant dans la direction opposée au gradient.

Soit le réseaux de neurones multicouches, représenté sur la figure 1.10 :

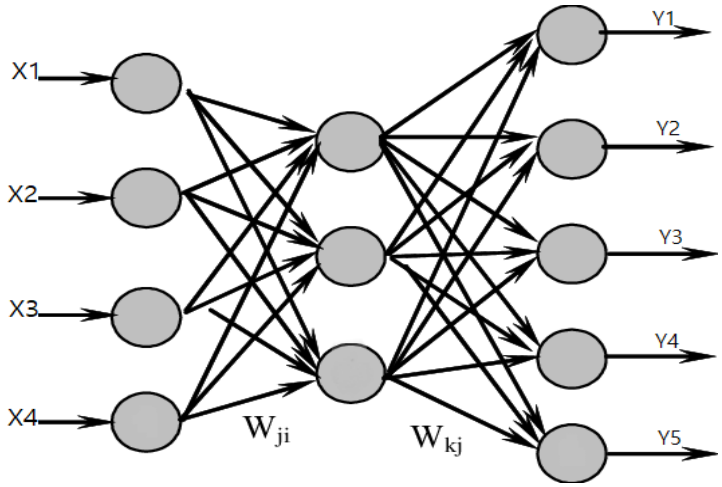



FIGURE 1.10 – Réseau de neurones multicouches

Avec :

* $X_i = [X_1, ..., X_4]$ : Représente le vecteur d'entrée.
* $Y_k = [Y_1, ..., Y_5]$ : Représente le vecteur de sortie.
* $W_{ji}$ : Représente le poids de connexion entre le $j^{ème}$neurone et la $i^{ème}$entrée.
* $W_{kj}$ :Représente le poids de connexion entre le $k^{ème}$ neurone de sortie et le $j^{ème}$neurone.

L'algorithme de rétropropagation du gradient appliqué sur le réseau de la figure suit les étapes suivantes :

1. Initialiser l'ensemble des vecteurs de poids $W_{ji}(0)$ et $W_{kj}(0)$ et le taux d'apprentissage $\eta$ à des valeurs aléatoires.
2. Commencer la propagation en avant : Appliquer le vecteur $X_i$ en entrée du réseau. On aura comme entrée de la couche cachée :

$$S_j = \sum_{i=1}^{4} X_i.W_{ji} + b_j \tag{1.4}$$

Avec :

* $S_j$ :somme pondérée des entrées au $j^{ème}$neurone de la couche cachée.
* $b_j$ :biais du $j^{ème}$ neurone de la couche cachée.

La sortie de la couche cachée est donnée par l'application d'une fonction d'activation F à la somme pondérée $S_j$ comme suit :

$$a_j = F(S_j) \tag{1.5}$$

3. Calculer la sortie $Y_k$ :

De la même manière que 1.4 l'entrée de la couche de sortie est donnée comme suit :

$$S_k = \sum_{j=1}^{3} a_j.W_{kj} + b_k \tag{1.6}$$

Avec :

* $S_k$ :somme pondérée des entrées au $k^{ème}$neurone de la couche de sortie.
* $b_k$ :biais du $k^{ème}$ neurone de la couche de sortie.

La sortie de la couche de sortie (sortie globale) est donnée par l'application d'une fonction d'activation G à la somme pondérée $S_k$ comme suit :

$$Y_k = G(S_k) \tag{1.7}$$

La propagation en avant se termine par cette étape.

4. Calculer l'erreur (la fonction de perte) :

Il s'agit de calculer la différence entre la sortie désirée $Y_k^d$ et la sortie obtenue à la fin de la propagation en avant $Y_k$. Elle est donnée par :

$$e_k = Y_k^d - Y_k \tag{1.8}$$

5. Calcul des nouvelles valeurs des poids :

Soit la somme des erreurs quadratiques observées sur l'ensemble des neurones de la couche de sortie :

$$E_k = \tfrac{1}{2} \sum_{k=1}^{5} (e_k)^2 = \frac{1}{2} \sum_{k=1}^{5} (Y_k^d - Y_k)^2 \tag{1.9}$$

Pour ramener la sortie obtenue vers celle désirée, on ajuste les poids du réseau en commençant par ceux de la couche de sortie puis on passe aux poids des neurones de la couche cachée.

6. Adaptation des poids des neurones de la couche de sortie $W_{kj}$ [8] :

Pour modifier les poids , on doit minimiser le gradient de l'erreur $\frac{\partial E_k}{\partial W_{kj}}$ tel que la variation

des poids est :

$$\triangle W_{kj} = -\eta \frac{\partial E_k}{\partial W_{kj}} \tag{1.10}$$

Avec $\eta$ : pas d'apprentissage $\in ]0,1]$.
On a :

$$\frac{\partial E_k}{\partial W_{kj}} = \frac{\partial E_k}{\partial Y_k} . \frac{\partial Y_k}{\partial S_k} . \frac{\partial S_k}{\partial W_{kj}} \tag{1.11}$$

On dérive chaque terme séparément :

* De 1.9 on a :

$$\frac{\partial E_k}{\partial Y_k} = -(Y_k^d - Y_k) \tag{1.12}$$

* De 1.7 on a :

$$\frac{\partial Y_k}{\partial S_k} = G'(S_k) \tag{1.13}$$

* De 1.6 on a :

$$\frac{\partial S_k}{\partial W_{kj}} = a_j \tag{1.14}$$

On remplace 1.12, 1.13 et 1.14 dans 1.11 :

$$\frac{\partial E_k}{\partial W_{kj}} = -(Y_k^d - Y_k).G'(S_k).a_j \tag{1.15}$$

Nous avons : $\frac{\partial E_k}{\partial S_k} = \frac{\partial E_k}{\partial Y_k} . \frac{\partial Y_k}{\partial S_k} = -(Y_k^d - Y_k)G'(S_k)$
On pose : $\partial k = (Y_k^d - Y_k)G'(S_k)$, on aura donc :

$$\frac{\partial E_k}{\partial W_{kj}} = -\partial k.a_j \tag{1.16}$$

Alors :

$$\triangle W_{kj} = -\eta \frac{\partial E_k}{\partial W_{kj}} = \eta.\partial k.a_j \tag{1.17}$$

La nouvelle valeur des poids de la couche de sortie $W_{kj}$ est donnée par la relation suivante :

$$W_{kj}(t) = W_{kj}(t-1) + \triangle W_{kj}(t) = W_{kj}(t-1) + \eta.\partial k.a_j \tag{1.18}$$

Adaptation des poids des neurones de la couche cachée $W_{ji}$ :

$$\frac{\partial E_k}{\partial W_{ji}} = \frac{\partial E_k}{\partial a_j}.\frac{\partial a_j}{\partial S_j}.\frac{\partial S_j}{\partial W_{ji}} \tag{1.19}$$

Les nouvelles valeurs des poids $W_{ji}$ données par la formule 1.20, se calculent de la même manière que pour les poids $W_{kj}$.

$$W_{ji}(t) = W_{ji}(t-1) + \triangle W_{ji}(t) \tag{1.20}$$

Avec : $\triangle W_{ji} = -\eta \frac{\partial E_k}{\partial W_{ji}} = -\eta.\partial j.X_i$ et :$-\partial j = (-\sum_{i=1}^{n0} \partial k.W_{ji}).F'(S_j)$

Ces étapes sont effectuées itérativement jusqu'à ce qu'un critère prédéfini soit atteint. Ce critère peut être :

–**Nombre d'itérations :** Fixer un nombre précis d'itérations à effectuer avant de terminer l'apprentissage.

–**Valeur d'erreur :** Fixer une valeur d'erreur minimale acceptable, en dessous de laquelle l'apprentissage est considéré comme terminé.

–**Convergence :**Vérifier si l'erreur de prédiction a convergé vers un plateau (stade où la valeur de l'erreur stagne et ne varie pas significativement malgré l'ajustement des paramètres du modèle).

–**Temps d'exécution :** Définir une limite de temps pour l'apprentissage, après laquelle l'algorithme s'arrête, quel que soit le critère d'arrêt atteint.

La règle de rétropropagation du gradient est très utilisée pour l'apprentissage des réseaux de neurones. Cependant, elle présente certaines limitations. On cite :

* Les problèmes de convergence ; lorsque le réseau de neurones est profond (comporte plusieurs couches), il est difficile d'atteindre la convergence vu le nombre élevé de paramètres à optimiser (les poids, les biais...). C'est également le cas lorsque l'erreur possède plusieurs minimums locaux (points où l'erreur est minimale), ce qui crée plusieurs directions possibles dans lesquelles le gradient peut effectuer la mise à jour des poids.

* Les performances de cette méthode dépendent de certains paramètres qui ne sont pas appris par le réseau et doivent être prédéfinis comme le pas d'apprentissage, la fonction d'activation, le nombre de neurones par couche, ...
* La descente de gradient peut être piégée dans des minimums locaux qui ne représentent pas forcément le minimum global recherché. Le réseau sera donc bloqué dans une solution sous-optimale sans pouvoir atteindre la meilleure performance possible.

## 1.9 Propriétés des réseaux de neurones

Les réseaux de neurones sont caractérisés par les propriétés suivantes :

* **Parallélisme :** En utilisant plusieurs unités de calcul en même temps, les réseaux de neurones peuvent effectuer des calculs en parallèle, ce qui signifie que plusieurs calculs peuvent être effectués simultanément accélérant ainsi le traitement des données et améliorant les performances globales du réseau. Le parallélisme est donc une caractéristique clé des réseaux de neurones.
* **Résistance aux pannes :** Les réseaux de neurones peuvent être conçus pour être résistants aux pannes, ce qui signifie qu'ils peuvent continuer à fonctionner correctement même en présence de défaillances ou d'erreurs dans certains de leurs composants. Pour cela, des techniques telles que la redondance des nœuds ou des connexions sont utilisées. En garantissant la fiabilité et la tolérance aux pannes, les réseaux de neurones peuvent fonctionner de manière stable et robuste, ce qui les rend adaptables à une large gamme d'applications.
* **Capacité d'adaptation :** La capacité d'adaptation est rendue possible grâce à un processus itératif appelé apprentissage, où le réseau de neurones est exposé à des exemples de données et ajuste ses paramètres internes (les poids et les biais) en fonction des erreurs commises. Cette capacité d'adaptation permet aux réseaux de neurones de s'améliorer avec l'expérience et de s'adapter à des environnements changeants faisant d'eux des outils puissants dans le domaine de l'intelligence artificielle.
* **Généralisation :** Les réseaux de neurones peuvent généraliser et extraire des modèles et des relations à partir des exemples d'apprentissage pour effectuer des prédictions sur de nouvelles données qui n'ont pas été vues auparavant.
* **Structure de connexion :** Il s'agit d'une caractéristique clé des réseaux de neurones. Elle fait référence à la manière dont les neurones et les couches d'un réseau de neurones sont connectés les uns aux autres. Ces structures sont très variées vu le nombre important des connexions.
* **Apprentissage en ligne :** Les réseaux de neurones peuvent être entraînés à partir

de données qui arrivent en continu, c'est-à-dire en ligne. Cette capacité est utile pour le traitement en temps réel des données qui arrivent rapidement et qui doivent être traitées en conséquence.

## 1.10 Domaines d'application des réseaux de neurones

Grâce à leurs propriétés, les réseaux de neurones sont utilisés dans plusieurs domaines, on cite :

* **Vision par ordinateur :** La vision par ordinateur consiste à analyser des images et des vidéos par des machines. Les réseaux de neurones y sont utilisés pour diverses tâches telles que : La détection d'objets, la surveillance et la reconnaissance faciale.
* **Sciences de la santé :** Les réseaux de neurones ont de multiples applications en santé, à savoir : l'analyse d'images médicales, la découverte de médicaments, et la prédiction de maladies.
* **Finance :** Dans le domaine financier, les réseaux de neurones permettent : La détection de fraudes, la recommandation de produits financiers et autres.
* **Robotique :** Les réseaux de neurones sont utilisés en robotique, notamment pour des tâches telles que la manipulation d'objets et la navigation autonome.

## 1.11 Conclusion

Ce chapitre consacré à l'étude des réseaux de neurones artificiels nous permet de conclure opportunément que ce sont sont des modèles puissants qui ont révolutionné le domaine de l'intelligence artificielle. Leur architecture inspirée du cerveau humain leur confère une grande capacité d'apprentissage et d'adaptation aux données. Dans le chapitre suivant, nous nous pencherons plus en détail sur les réseaux de neurones convolutifs (CNN), qui sont essentiels dans plusieurs domaines liés au traitement des données visuelles et à l'analyse d'images.

# Chapitre 2

# Réseaux de neurones convolutifs

## 2.1 Introduction

Les réseaux de neurones convolutifs (CNN) sont une catégorie spécialisée de réseaux de neurones artificiels conçue pour traiter et analyser des données structurées en grille, telles que les images. Inspirés par le fonctionnement du cortex visuel chez les animaux, les CNN utilisent des opérations de convolution pour extraire des caractéristiques hiérarchiques et locales à partir des données d'entrée. Initialement développés pour surmonter les limitations des approches traditionnelles en traitement d'images (RNA), les réseaux de neurones convolutifs sont aujourd'hui au cœur des applications modernes telles que la classification d'images et la détection d'objets. Ce chapitre explorera en profondeur les principes fondamentaux des CNN, leur architecture, leur apprentissage et leurs types, en mettant en lumière leur rôle crucial dans l'évolution des technologies de vision par ordinateur.

## 2.2 Réseaux de neurones convolutifs

Les réseaux de neurones convolutifs (*convolutional neural networks* ou CNN) sont des réseaux de neurones artificiels multicouches qui reçoivent en entrée des images. Ces données sont présentées au réseau sous forme d'une matrice de pixels multidimensionnelle structurée comme suit : (largeur, hauteur, nombre de composantes). Le nombre de composantes dépend du type de l'image : les images au niveau du gris ne contiennent qu'une seule composante, les images en couleur contiennent trois composantes qui correspondent aux trois couleurs de base (rouge, vert et bleu).

Les réseaux de neurones convolutifs s'inspirent du cortex visuel animal. Ce dernier est la région du cerveau responsable du traitement et de l'interprétation des informations visuelles qui proviennent des yeux. Il se trouve à l'arrière du cerveau, dans le lobe occipital, il est

composé de plusieurs aires, chacune traitant des informations spécifiques, comme montré sur la figure 2.1 :

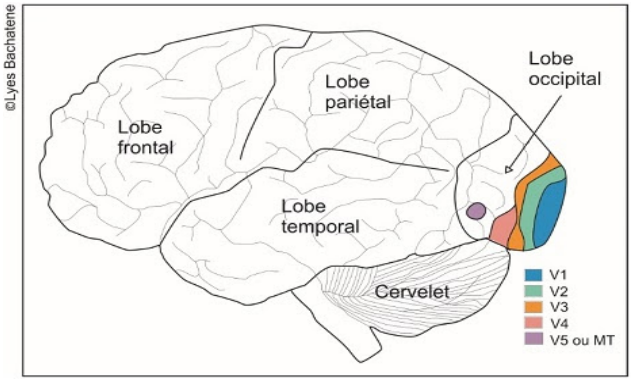



FIGURE 2.1 – Cortex visuel

L'information visuelle est captée par les cellules photoréceptrices de la rétine, située à l'arrière de l'œil puis la convertissent en un signal électrique qui sera transmis le long du nerf optique jusqu'au cortex visuel. L'aire V1 (également connu sous le nom de cortex visuel primaire) est la première zone du cortex visuel à recevoir cette information. C'est dans cette zone que les caractéristiques de base telles que les bords, les orientations ... sont extraites. L'information est ensuite transmise aux aires visuelles supérieures V2, V3, V4 et V5 qui sont spécialisées dans le traitement de caractéristiques visuelles plus complexes, comme la reconnaissance d'objets, la perception de formes, la couleur, le mouvement ...

Le traitement de l'information visuelle est donc effectué de manière hiérarchique, commençant par les caractéristiques simples et progressant vers les caractéristiques plus complexes.

## 2.3 Passage aux réseaux de neurones convolutifs

La classification des images par des réseaux de neurones classiques doit d'abord passer par une étape d'extraction de caractéristiques avant de pouvoir les présenter au réseau, afin qu'il effectue la tâche de classification, comme montré sur la figure 2.2.a. Cette étape d'extraction est réalisée à l'aide d'extracteurs, en sélectionnant des caractéristiques spécifiques (couleur, texture, forme...) à partir de l'image. Il existe plusieurs types d'extracteurs à savoir HOG (*Histogram of Oriented Gradients*) [9], SIFT (*Scale-Invariant Feature Transform*) [10], SURF (*Speeded-Up Robust Features*) [11]. Ceci rend cette phase d'extraction délicate, car le choix de l'extracteur approprié est souvent un processus fastidieux et compliqué.

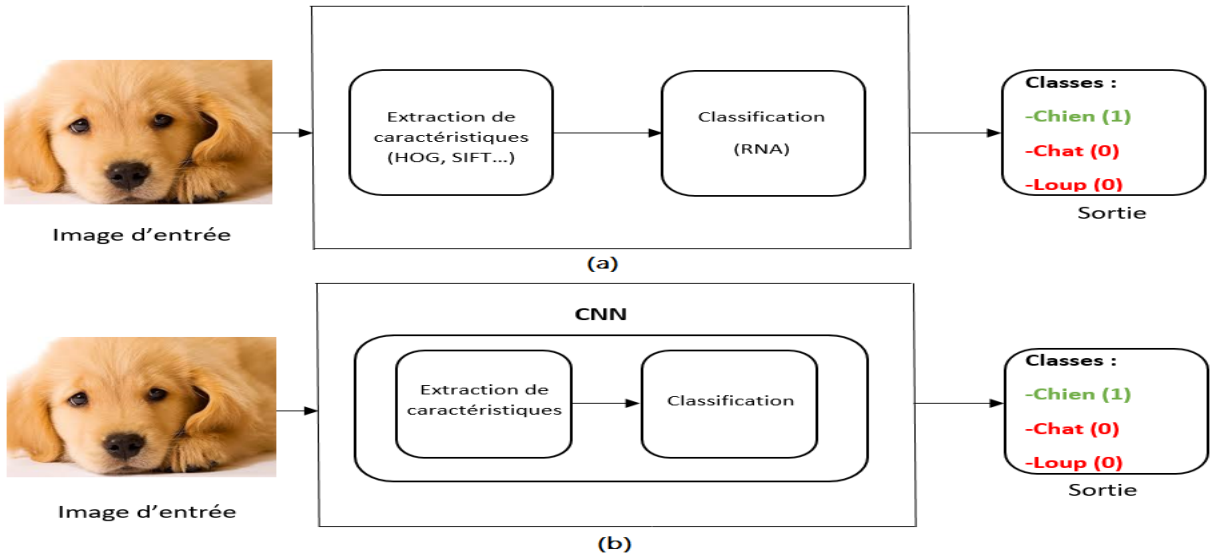



FIGURE 2.2 – Processus de classification en utilisant les réseaux de neurones classiques (a) et les réseaux de neurones convolutifs (b)

C'est là où réside la force des réseaux de neurones convolutifs : Ils effectuent la tâche d'extraction des caractéristiques automatiquement grâce à leur disposition d'une partie extraction en amont de la partie classification (voir figure 2.2.b). Cette partie d'extraction permet de prendre en compte la structure spatiale et la localité des informations de l'image.

## 2.4 Architecture d'un réseau de neurones convolutif

Un réseau de neurones convolutif est composé de deux parties : une partie extraction et une partie classification comme le montre la figure 2.3 :

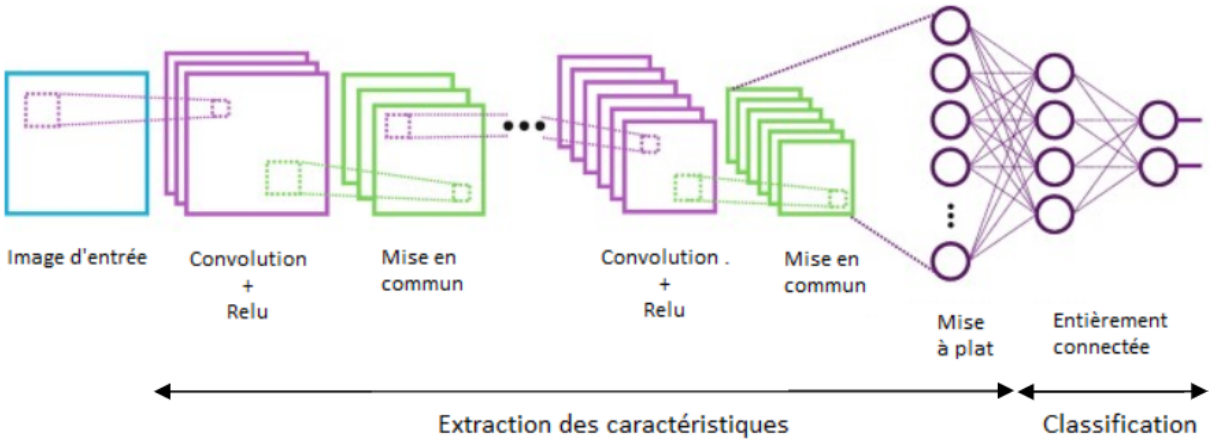



FIGURE 2.3 – Architecture d'un réseau de neurones convolutif

### 2.4.1 Partie extraction

C'est la première partie d'un réseau de neurones convolutif, elle est constituée par une succession de plusieurs couches. On cite : la couche de convolution, la couche de correction ReLU, la couche de mise en commun (*pooling*) et la couche de mise à plat (*flattening*).

#### 2.4.1.1 Couche de convolution

La couche de convolution est le bloc fonctionnel principal d'un réseau de neurones convolutif.

La convolution est une opération mathématique simple. Elle consiste à combiner deux fonctions pour en créer une nouvelle. Cela se fait en superposant une fonction sur une autre et en effectuant un décalage progressif de la fonction superposée.

Cet outil est largement utilisé dans le domaine du traitement de l'image dans le but de repérer la présence d'un ensemble de caractéristiques dans les images reçues en entrée, en appliquant une succession de filtrages par convolution. Cette opération s'effectue selon les étapes suivantes :

1. Choisir la taille du filtre représentant la caractéristique souhaitée.
2. Initialiser les coefficients du filtre.
3. Positionner le noyau du filtre sur le $1^{er}$ pixel de l'image.
4. Calculer le produit de convolution du filtre et de la partie de l'image balayée selon la relation suivante :

$$Y = \sum X[i,j] * K[m,n] \tag{2.1}$$

Avec :

**Y :** Résultat du produit de convolution.

**X[i,j] :** Élément [i,j] de l'image.

**K[m,n] :** Élément [m,n] du filtre.

5. Glisser le filtre d'un certain pas comme le montre la figure 2.4, et refaire les étapes précédentes.

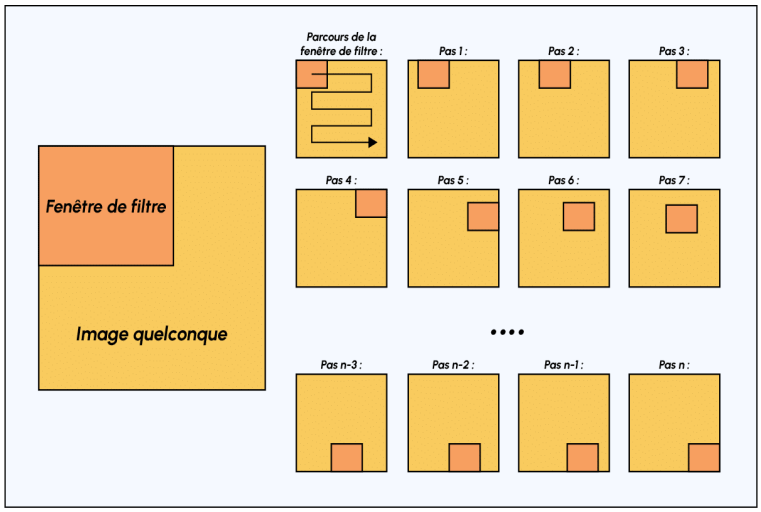



FIGURE 2.4 – Parcours de la fenêtre de filtre sur l'image.

Voici un exemple de convolution, donné par la figure 2.5 :

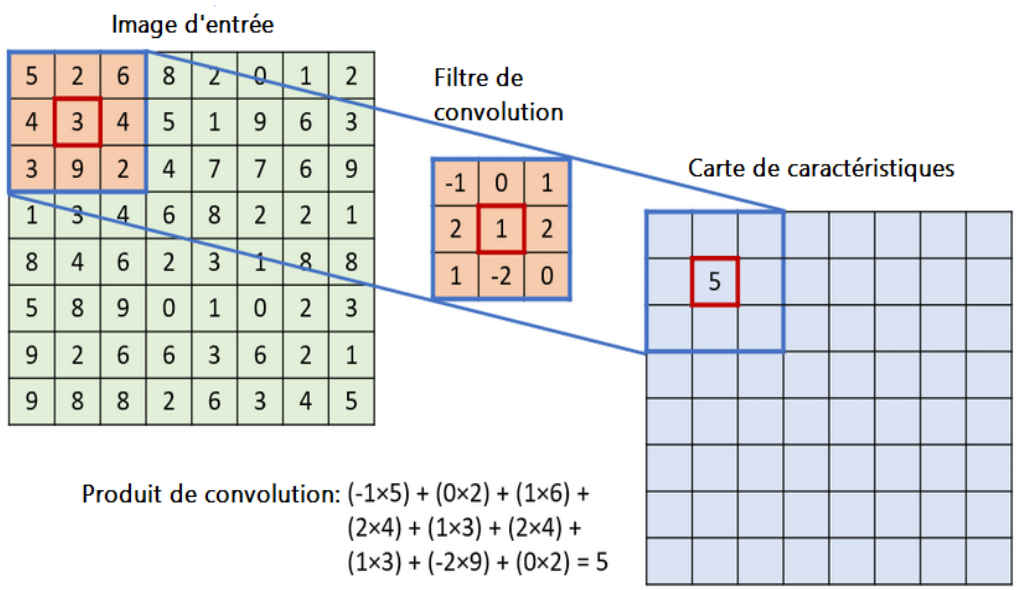



FIGURE 2.5 – Exemple de convolution

Le résultat du produit de convolution entre le filtre et l'image est appelé carte de caractéristiques. Le nombre de cartes de caractéristiques obtenues en sortie de la couche est le même que le nombre de filtres appliqués.

Un modèle de réseau de neurones convolutif peut contenir plusieurs couches de convolution, offrant ainsi une structure hiérarchique au réseau (de la même manière que dans le cortex visuel, expliqué dans la section 2.2).

#### 2.4.1.2 Couche de correction

Dans cette couche, toutes les valeurs négatives reçues en entrées sont remplacées par une valeur nulle grâce à l'application d'une fonction d'activation du type ReLU.

L'intérêt de ces couches de correction est de rendre le modèle non linéaire et de ce fait plus complexe.

Voici l'allure de la fonction ReLU, donnée par la figure 2.6 :

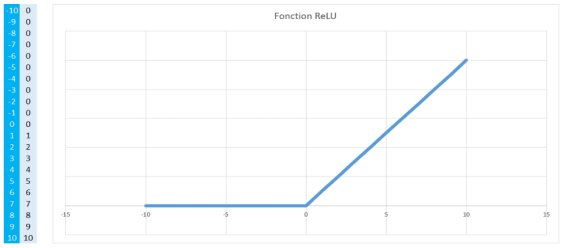



FIGURE 2.6 – Fonction d'activation ReLU

#### 2.4.1.3 Couche de mise en commun

La couche de mise en commun ou *pooling* permet de réduire la taille de l'image tout en conservant ses caractéristiques phares. Elle réduit également le nombre de calculs dans le réseau améliorant ainsi son efficacité. Elle se trouve souvent entre deux couches de convolution. Cette couche prend en entrée les cartes de caractéristiques obtenues à la sorties des couches de convolution qui la précédent.

Le *pooling* se fait en parcourant l'image par une fenêtre de *pooling*, selon un pas prédéfini. Il consiste à remplacer la partie de l'image parcourue par le filtre, par une valeur précise déterminée par le type de *pooling* utilisé, car il en existe plusieurs à savoir : *max pooling, average pooling, sum pooling.*

- ***Max pooling :*** C'est le type le plus utilisé, car il est rapide. Il prend en compte la valeur maximale des pixels de la cellule sélectionnée.

Un exemple d'application du *max pooling* est illustré par la figure 2.7 :

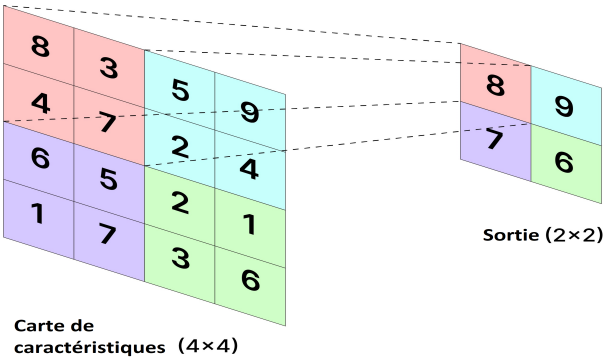



FIGURE 2.7 – Exemple d'application du *max pooling*

* ***Average (mean) pooling :*** Il s'agit de prendre la valeur moyenne des pixels de la cellule en question.
* ***Sum pooling :*** Consiste à sélectionner la somme des valeurs des pixels de la cellule de l'image.

Après application du *pooling* sur les cartes de caractéristiques, on aura en sortie de la couche le même nombre de cartes qu'en entrée mais de taille plus réduite.

#### 2.4.1.4 Couche de mise à plat

La couche de mise à plat (*flattening*) se trouve à la fin de la partie extraction, après une succession de couches de convolution et de *pooling*. Ce processus consiste simplement à empiler (concaténer) les valeurs des cartes de caractéristiques afin de les représenter sous forme d'un vecteur colonne unidimensionnel qui sera transmis à la partie classification.

Un exemple de *flattening* est montrée par la figure 2.8 :

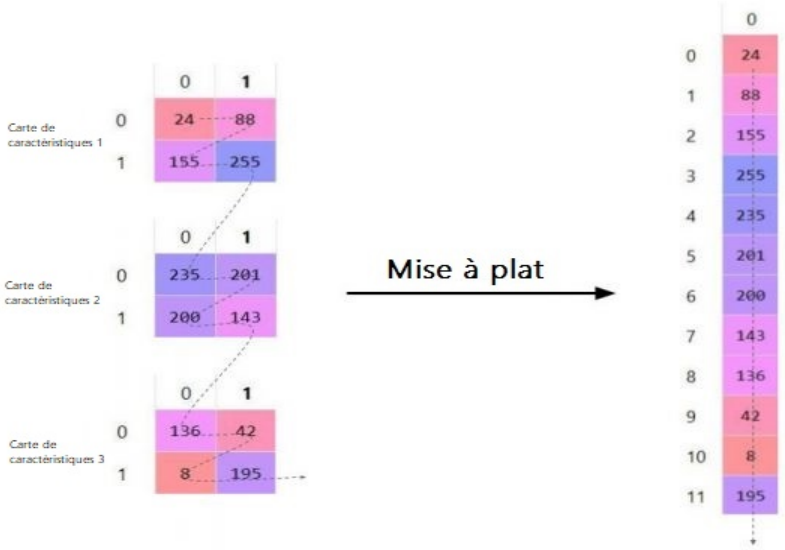



FIGURE 2.8 – Exemple de *flattening*

### 2.4.2 Partie classification

Cette seconde partie se trouve à la fin des réseaux de neurones convolutifs. Elle est composée d'une seule couche appelée couche entièrement connectée (*fully connected*), considérée comme un Perceptron multicouches. Comme pour les réseaux de neurones classiques, cette couche reçoit en entrée un vecteur de caractéristiques. Il s'agit du vecteur obtenu en sortie de la couche de *flattening* de la partie extraction, comme montré sur la figure 2.9 :

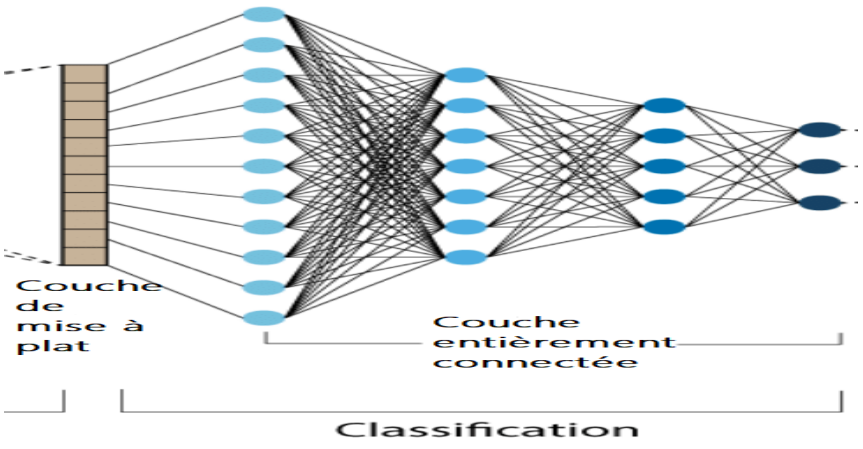



FIGURE 2.9 – Couche entièrement connectée

Le terme "entièrement connectée" vient du fait que chaque neurone d'une couche est connecté aux neurones de la couche suivante (chaque neurone reçoit en entrée les sorties de tous les neurones de la couche précédente).

Le traitement dans cette couche se fait de la même manière que dans un Perceptron multicouches.

La sortie de cette couche est un vecteur de taille N, où chaque élément correspond à la probabilité pour l'image en entrée d'appartenir à une des N classes à prédire. (La taille du vecteur de sortie= le nombre de classes).

Après le passage des caractéristiques à travers cette couche, une fonction d'activation (généralement softmax) est appliquée pour obtenir des probabilités d'appartenance à chaque classe, qui sont comprises dans l'intervalle [0,1].

## 2.5 Apprentissage des réseaux de neurones convolutifs

L'apprentissage d'un réseau de neurones convolutif est un apprentissage supervisé qui consiste à adapter ses paramètres de manière à rapprocher la sortie estimée de la sortie désirée. L'adaptation des paramètres se fait par la méthode de la descente du gradient qui consiste à minimiser l'erreur en utilisant une fonction de perte qui mesure la qualité des prédictions du modèle. Les fonctions de pertes se divisent en deux catégories selon les types de problèmes que l'on rencontre : problèmes de classification et problèmes de régression. La classification consiste à prédire les probabilités des différentes classes dans un modèle, tandis que la régression vise à prédire une valeur numérique en se basant sur un ensemble de caractéristiques. Parmi les fonctions de perte utilisées en classification, on trouve l'entropie croisée (*Cross-Entropy*) et la perte de charnière (*Hinge Loss*). En régression, les fonctions couramment utilisées sont la perte quadratique moyenne (*Mean Squared Error - MSE*) et la perte absolue moyenne (*Mean Absolute Error - MAE*). L'entropie croisée et la perte quadratique moyenne sont les deux fonctions de perte les plus fréquemment employées dans leurs domaines respectifs [12].

* **Entropie croisée (*Cross-Entropy*) :** C'est la fonction de perte qui mesure la divergence entre la distribution prédite par le modèle et la distribution réelle des classes. La formule de l'entropie croisée varie légèrement selon le type de classification ; binaire (donnée par l'équation 2.2) ou multi-classes (donnée par l'équation 2.3).

$$L(y, \hat{y}) \quad = \quad -[y * log(\hat{y}) + (1 - y) * log(1 - \hat{y})] \tag{2.2}$$

Avec :

**y :** l'étiquette réelle.

$\hat{y}$ **:** la probabilité prédite pour la classe positive.

$$L(y, \hat{y}) \quad = \quad -\sum_{i=1}^{N} y_i * log(\hat{y}_i) \tag{2.3}$$

Avec :

**N :** le nombre total de classes.

$y_i$ : l'étiquette réelle pour la classe $i$ (1 si la classe est correcte, 0 sinon).

$\hat{y}_i$ : la probabilité prédite pour la classe $i$.

* **Perte quadratique moyenne (*Mean Squared Error - MSE*)** : Il s'agit d'une fonction de perte qui calcule l'écart quadratique moyen entre les prédictions du modèle et les valeurs réelles. Elle est donnée par la formule 2.4 :

$$L \quad = \quad \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \tag{2.4}$$

Avec :

**N :** Nombre d'échantillons dans l'ensemble de données.

$y_i$ **:** La valeur réelle pour le $i^{ème}$ échantillon.

$\hat{y}_i$ **:** La valeur prédite pour le $i^{ème}$ échantillon.

### 2.5.1 Sur-apprentissage et sous-apprentissage

Le sur-apprentissage et le sous-apprentissage sont deux phénomènes qui peuvent apparaître durant la phase d'entraînement du réseau.

Le sur-apprentissage (*overfitting*) est le cas où le modèle s'est ajusté de manière excessive aux données d'entraînement, au point de ne pas pouvoir répondre efficacement à de nouvelles données.

D'autre part, le sous-apprentissage (*underfitting*) se produit lorsque le modèle n'a pas appris suffisamment à partir des données d'entraînement et ne parvient donc pas à se généraliser sur d'autres données.

Le problème de généralisation engendré par ces deux phénomènes peut être limité grâce à la technique de régularisation.

#### 2.5.1.1 Technique de régularisation

La régularisation est un procédé visant à améliorer les performances de généralisation d'un algorithme d'apprentissage, autrement dit à diminuer son erreur sur les échantillons de test.

La régularisation vise à contrôler la flexibilité du modèle en empêchant ses paramètres de devenir trop complexes. Parmi les méthodes de régularisation on trouve le *dropout*, la normalisation par lot (*batch normalization*), le *drop-weights* et l'augmentation des données (*data augmentation*).

**2.5.1.1.1 *Dropout*** Le *dropout* consiste à désactiver temporairement, de manière aléatoire et à chaque époque (*epoch)*, un certain nombre de neurones ainsi que leurs connexions associées comme montré sur la figure 10.b. De cette manière, seuls les neurones activés participent à la propagation avant et à la rétropropagation de l'information. Cette méthode permet au modèle d'être moins dépendant d'un ensemble particulier de caractéristiques, le forçant ainsi à être plus robuste et plus généralisable sur de nouvelles données.

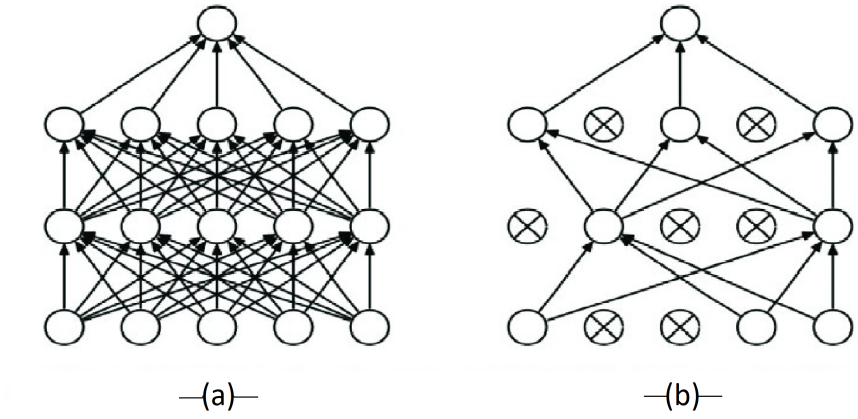



FIGURE 2.10 – Réseau de neurone sans dropout (a), avec dropout (b)

**2.5.1.1.2 Normalisation par lot** Le réajustement constant des paramètres du réseau durant la phase d'apprentissage engendre le problème de décalage de distribution (*covariant shift*). Ce phénomène fait référence au changement de la distribution des activations à chaque nouvelle itération de l'entraînement. Cela rend l'apprentissage plus difficile, car le réseau doit constamment s'adapter à ces variations. C'est là où la normalisation par lot s'avère utile. La normalisation des données consiste à mettre toutes les données à une échelle commune afin de faciliter leur traitement tandis que la normalisation par lot effectue cette tâche au sein du réseau en normalisant les activations sur un lot (un sous-ensemble des données d'entraînement) plutôt que sur un seul exemple à la fois, contribuant ainsi à accélérer l'apprentissage et à améliorer les performances du modèle.

Ces deux techniques sont souvent utilisées ensemble pour améliorer la régularisation et la

performance des modèles de réseaux de neurones. En effet, *dropout* réduit la corrélation entre les couches en désactivant de manière aléatoire certains neurones, tandis que la normalisation par lots réduit la variation des activations.

**2.5.1.1.3** ***Drop-weights*** Cette méthode est très similaire à celle du *dropout*. À chaque itération d'apprentissage, les connexions entre les neurones (les poids) sont désactivées au lieu de désactiver les neurones ; c'est la seule différence entre *drop-weights* et *dropout* [13].

**2.5.1.1.4 Augmentation des données** Entraîner le modèle sur une quantité de données importante est la manière la plus facile pour éviter le sur-apprentissage. La méthode d'augmentation des données consiste à générer plus de données à partir de celles déjà existantes, en utilisant plusieurs techniques comme la rotation, le changement de couleur, l'addition de bruit ,... [13]

## 2.6 Avantages et inconvénients

Les CNN ont révolutionné le domaine de la vision par ordinateur grâce à leur flexibilité et à leur aptitude à gérer des volumes massifs de données. Cependant, il est important de savoir que, malgré leur nombreux avantages, ces réseaux ont également des limitations et défis.

### 2.6.1 Avantages

* **Extraction automatique des caractéristiques :** Les CNN apprennent et extraient automatiquement des caractéristiques pertinentes des données d'entrée, ce qui leur permet d'obtenir une haute précision dans la classification d'images en identifiant des motifs et des relations complexes.
* **Utilisation des mêmes connaissances pour toutes les positions de l'image :** Les CNN exploitent le principe du partage des paramètres, ce qui leur permet d'utiliser les mêmes paramètres appris pour différentes régions d'une image. Cette efficacité les rend adaptables aux applications en temps réel ou aux ressources limitées.
* **Capacité à traiter de grands ensembles de données :** Les CNN sont capables de traiter de grands ensembles de données comportant un grand nombre d'exemples d'entraînement. Cela leur permet d'apprendre efficacement des motifs complexes et de généraliser correctement aux données non vues.
* **Apprentissage hiérarchique :** Les CNN sont composés de plusieurs couches qui apprennent des représentations hiérarchiques des données. Cet apprentissage hiérar-

chique leur permet de capturer des motifs et des relations complexes à différents niveaux d'abstraction [14].

### 2.6.2 Inconvénients

* **Exigences de calcul élevées :** Les CNN nécessitent des ressources informatiques importantes pour l'apprentissage et l'inférence. Cela peut rendre leur utilisation coûteuse ou même impossible pour les systèmes ayant des ressources informatiques limitées.
* **Besoins en grande quantité de données étiquetées :** Les CNN nécessitent généralement une grande quantité de données étiquetées pour entraîner des modèles de haute qualité. La collecte et l'étiquetage de ces données peuvent être coûteux et chronophages.
* **Espace mémoire important :** Les CNN peuvent nécessiter un grand espace mémoire pour stocker les paramètres du modèle et les données intermédiaires, ce qui peut limiter leur utilisation sur les systèmes ayant une mémoire limitée.
* **Défis d'interprétabilité :** Les CNN peuvent être difficiles à interpréter en termes de compréhension de la manière dont le modèle prend des décisions pour l'inférence.
* **Efficacité limitée pour les données séquentielles :** Les CNN ne se prêtent pas bien à la modélisation de données séquentielles telles que les séries chronologiques.
* **La formation prend beaucoup de temps :** Les CNN peuvent prendre beaucoup de temps pour être formés efficacement, en particulier avec des ensembles de données très grands ou complexes.

Pour remédier à certains de ces inconvénients, par exemple le temps d'apprentissage, on fait souvent appel au transfert d'apprentissage.

## 2.7 Transfert d'apprentissage

Le transfert d'apprentissage, en apprentissage automatique, consiste à utiliser les poids d'un modèle préalablement entraîné sur une tâche A pour les adapter à une tâche B, différente mais présentant des similitudes. Former un réseau neuronal à partir de zéro requiert généralement un grand volume de données, ce qui n'est pas toujours réalisable. C'est dans ce contexte que le transfert d'apprentissage devient précieux. En réutilisant un modèle préentraîné comme point de départ, il est possible de développer un modèle performant avec un volume de données d'entraînement limité, tout en réduisant significativement le temps de calcul. Cette approche est particulièrement efficace dans des domaines comme la vision par ordinateur, pour des applications telles que la classification d'images, la détection d'objets,

et bien d'autres.

## 2.8 Types des réseaux de neurones convolutifs

Il existe plusieurs architectures de réseaux de neurones convolutifs dont la différence principale réside dans le nombre de couches. Certaines d'entre elles sont résumées dans l'ordre chronologique dans le tableau 2.1 [15] :

| Types | Nombre de couches | Date | Taille de l'image d'entrée | Base de données |
|---|---|---|---|---|
| LeNet | 7 | 1998 | 32×32×1 | MNIST |
| AlexNet | 8 | 2012 | 227×227×3 | ImageNet |
| ZFNet | 8 | 2014 | 224×224×3 | ImageNet |
| VGG (visual geometry group) | 16 ou 19 | 2014 | 224×224×3 | ImageNet |
| GoogleNet | 22 | 2015 | 224×224×3 | ImageNet |
| ResNet | 153 | 2016 | 224×224×3 | ImageNet |
| Darknet19 | 26 | 2016 | 224×224×3 | ImageNet |
| DenseNet | 201 | 2017 | 224×224×3 | ImageNet,CIFAR-10, CIFAR-100 |
| EfficientNet | - | 2019 | 224×224×3 (B0) | ImageNet |

TABLE 2.1 – Types des réseaux de neurones convolutifs

### 2.8.1 LeNet

C'est une référence importante dans le développement des réseaux de neurones convolutifs qui a pavé la voie pour de nombreuses avancées ultérieures dans le domaine de la vision par ordinateur. Cependant la capacité de traiter des images à haute résolution nécessite plus de couches de convolution que cette architecture, donnée par la figure 2.11, ne possède pas. Il a depuis été surpassé en termes de performances par des architectures plus récentes et plus complexes telles que AlexNet, ResNet, VGG...

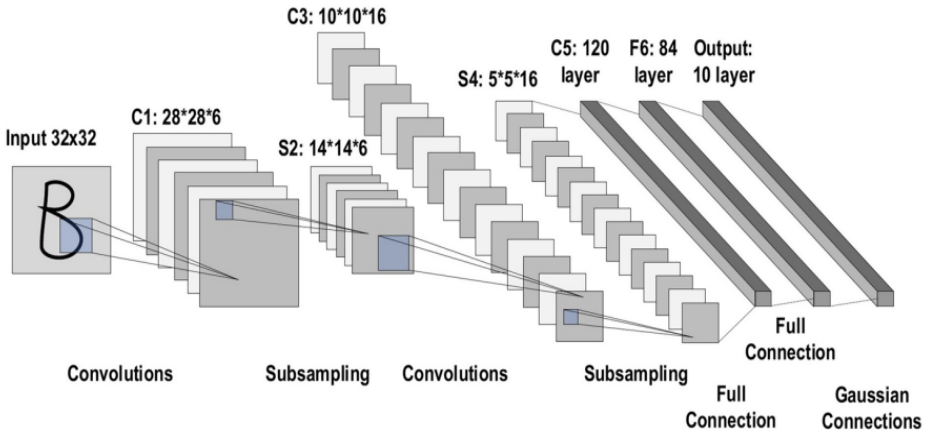



FIGURE 2.11 – Architecture LeNet

### 2.8.2 AlexNet

La spécificité d'AlexNet consiste en l'introduction de plusieurs concepts novateurs tels que :

1. L'utilisation de la fonction d'activation ReLU (*Rectified Linear Unit*) qui a conduit à une formation plus rapide et plus précise du réseau neuronal.

2. La technique de régularisation *Dropout* qui a amélioré la généralisation du modèle en réduisant le sur-apprentissage.

3. L'utilisation fortement recommandée des unités de traitement graphique GPUs (*Graphics Processing Units*) qui sont particulièrement adaptés aux calculs intensifs nécessaires à l'entraînement des réseaux de neurones.

Son architecture est donnée par la figure 2.12.

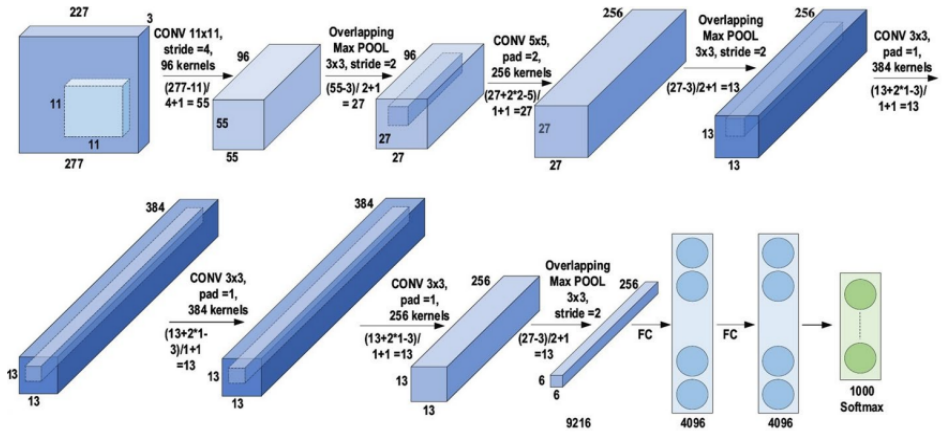



FIGURE 2.12 – Architecture AlexNet

### 2.8.3 ZFNet

L'architecture ZFNet est une variante améliorée d'AlexNet (figure 2.13). Elle utilise des filtres de grande taille dans les premières couches du réseau pour capturer des caractéristiques de plus grande taille dans les images d'entrée, et des filtres de petite taille dans les couches profondes pour capturer des caractéristiques plus détaillées. Cette approche permet de capturer des caractéristiques à différentes échelles, ce qui permet d'améliorer la qualité de la représentation des images et la précision de la classification.

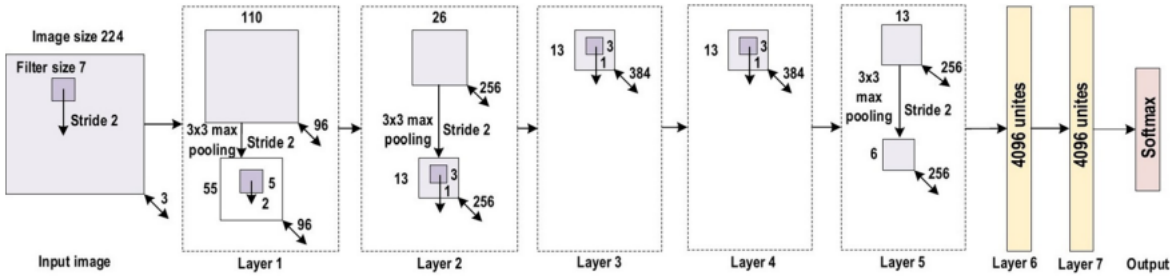



FIGURE 2.13 – Architecture ZFNet

### 2.8.4 VGG

VGG (*Visual Geometry Group*) est une architecture de réseau de neurones convolutif qui a introduit des réseaux de convolution très profonds pour la classification d'images (voir figure 2.14).

VGG utilise des filtres de taille 3x3 contrairement aux autres modèles qui utilisent des couches de convolution avec des filtres de taille plus grande comme ZFNet. Il utilise également des blocs de convolution à plusieurs couches de suite, ce qui en fait un réseau de convolution très profond.

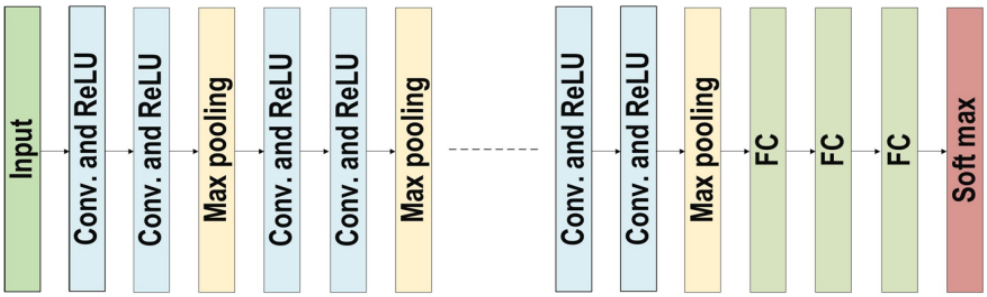



FIGURE 2.14 – Architecture VGG

### 2.8.5 GoogleNet

GoogleNet est inspiré de LeNet mais avec un nouvel élément appelé module de création "Inception". Ce module est basé sur plusieurs très petites convolutions dans le but de réduire considérablement le nombre de paramètres. L'architecture de GoogleNet (voir figure 2.15) consiste en un réseau CNN profond mais qui réduit le nombre de paramètres de 60 millions (AlexNet) à 4 millions [16].

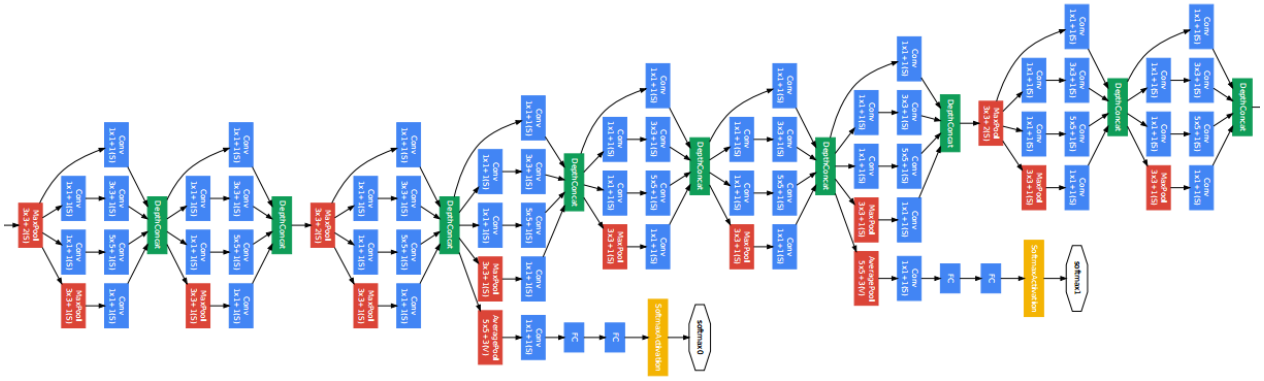


FIGURE 2.15 – Architecture GoogleNet

### 2.8.6 ResNet

ResNet (*Residual Network*) est une architecture de réseau de neurones convolutif qui permet de contourner la contrainte de la disparition du gradient en utilisant des connexions résiduelles (voir figure 2.16). Ces connexions résiduelles sont des liens directs qui sautent une ou plusieurs couches de convolution, afin de permettre aux signaux de circuler directement entre les couches.

ResNet a été l'un des premiers CNN à utiliser la fonction de normalisation par lots.

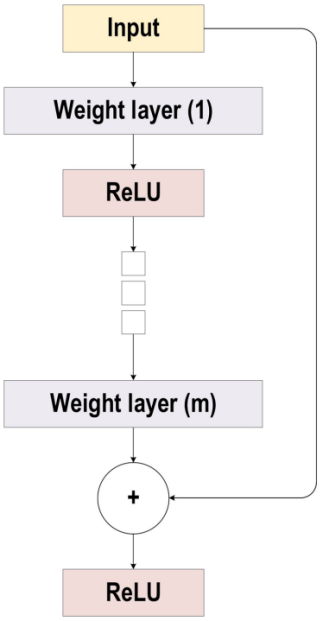



FIGURE 2.16 – Architecture ResNet

### 2.8.7 Darknet19

Darknet-19 est un réseau comprenant 19 couches de convolution utilisées pour extraire efficacement les caractéristiques des images (voir figure 2.17). Il utilise des fonctions d'activation de type ReLU et la technique de normalisation par lot. Ce réseau est utilisé comme base pour le modèle YOLOv2, un modèle de détection d'objets en temps réel [17]. Il a été amélioré par la suite avec l'ajout d'autres couches de convolution et l'intégration de plusieurs techniques. Cette évolution a conduit à la création de Darknet-53, un réseau plus profond et plus performant, qui a servi de point de départ crucial pour les modèles de détection d'objets modernes tels que YOLOv4.

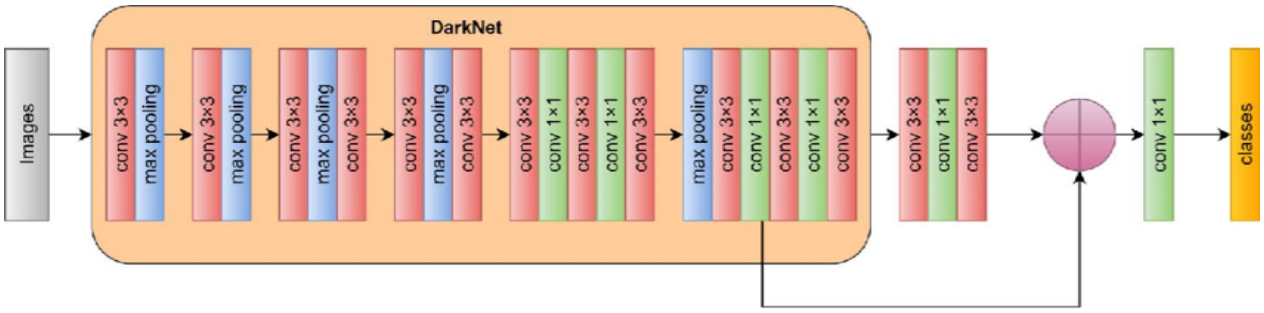



FIGURE 2.17 – Architecture Darknet19

### 2.8.8 DenseNet

La particularité du réseau DenseNet réside dans sa structure de connexion dense (voir figure 2.18). Cette architecture où chaque couche est reliée à toutes les couches précédentes dans un bloc dense, favorise la réutilisation maximale des caractéristiques et permet aux informations de circuler facilement, évitant ainsi le problème de la disparition du gradient. Cela en fait une architecture de réseau populaire dans de nombreux domaines de vision par ordinateur.

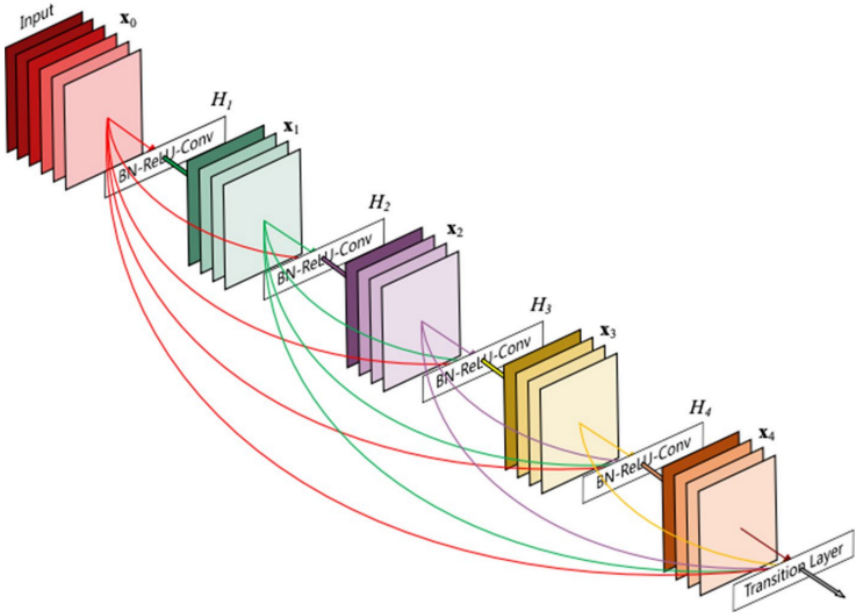



FIGURE 2.18 – Architecture DenseNet

### 2.8.9 EfficientNet

EfficientNet est une famille d'architectures de réseaux de neurones convolutifs, développée par Google en 2019 [18]. Cette famille comprend plusieurs versions, nommées de B0 à B7, chacune étant une version améliorée de celle de base B0. EfficientNet a introduit un concept appelé *compound scaling*, qui lui a permis de remédier au problème d'équilibre entre la précision et la consommation de ressources que rencontrent les modèles traditionnels de *deep learning*. Cette technique consiste à ajuster simultanément la profondeur (nombre de couches), la largeur (nombre de filtres par couche) et la résolution de l'image d'entrée du réseau, en fonction des ressources disponibles, offrant ainsi une meilleure performance tout en utilisant moins de ressources. La version B0 utilise des *Mobile Inverted Bottleneck Convolution (MBConv)* (voir figure 2.19), des blocs conçus pour réduire le nombre de paramètres tout en maintenant une bonne précision.


FIGURE 2.19 – Architecture EfficientNet

## 2.9 Conclusion

Ce chapitre consacré à l'étude des réseaux de neurones convolutifs (CNN) nous permet de conclure que ces architectures sont particulièrement adaptées pour le traitement des images. Leur capacité à extraire automatiquement des caractéristiques, à offrir une haute précision dans la classification d'images et à traiter de grands ensembles de données, les rend indispensables dans de nombreux domaines de recherche et d'application. Ce potentiel est constamment exploré et exploité, notamment dans la détection d'objets, qui sera traitée dans le chapitre suivant. Nous y examinerons comment les CNN peuvent être utilisés pour classifier et localiser des objets dans des images complexes, approfondissant ainsi les applications spécifiques de ces réseaux.

# Chapitre 3

# Détection d'objets

## 3.1 Introduction

La détection d'objets est un domaine de recherche très actif et d'un grand intérêt applicatif dans la vision par ordinateur et l'intelligence artificielle. Il s'agit d'un processus essentiel qui permet de localiser et de classifier les objets d'intérêt présents dans une image ou une vidéo. Ce domaine de recherche connaît des progrès considérables en termes de vitesse et de précision grâce aux avancées technologiques.

Au cours de ce chapitre, nous plongerons dans l'univers fascinant de la détection d'objets. Nous commencerons par une brève définition et explorerons les principes fondamentaux de cette technologie et ses domaines d'application. Enfin nous soulignerons l'évolution des approches utilisées, allant des méthodes classiques aux approches modernes basées sur des réseaux de neurones convolutifs (CNN) telles que YOLO (*You Only Look Once*).

## 3.2 Détection d'objets

La détection d'objets fait référence à la tâche de localiser et de classifier des objets spécifiques dans une image ou une vidéo. Elle utilise des techniques d'apprentissage automatique et de vision par ordinateur pour analyser les données visuelles et détecter la présence d'objets d'intérêt. Le processus de détection d'objets implique donc deux approches principales : la localisation et la classification, comme le montre la figure 3.1 :

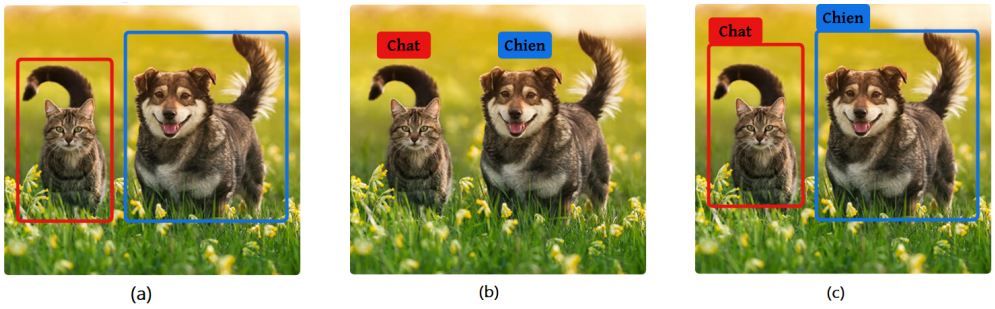



FIGURE 3.1 – Localisation (a), Classification (b), Détection d'objets (c)

### 3.2.1 Localisation

Il s'agit de déterminer la position spatiale exacte des objets dans l'image. Cela est souvent réalisé en encadrant les objets avec des formes rectangulaires appelés boîtes englobantes (*bounding boxes*).

### 3.2.2 Classification

La classification est le processus de regroupement des éléments ou des données en classes distinctes basées sur leurs caractéristiques similaires. Elle constitue une étape clé dans la détection d'objets car elle permet de différencier les objets d'intérêt des autres éléments qui constituent soit l'arrière plan, soit d'autres objets différents de ceux d'intérêt.

## 3.3 Domaines d'application de la détection d'objets

La détection d'objets existe depuis des années, mais elle devient plus visible dans un large éventail d'industries maintenant plus que jamais, ouvrant la voie à une variété d'applications innovantes. Cette approche consistant à localiser et à classifier les objets dans des images ou des vidéos a révolutionné plusieurs secteurs, tels que la conduite autonome, l'industrie et la robotique, la sécurité et la surveillance et la médecine.

### 3.3.1 Conduite autonome

Un véhicule autonome doit prendre des décisions en temps réel concernant les actions qu'il doit effectuer, telles que freiner, tourner ou accélérer. Pour ce faire, il doit être capable de localiser et de classifier tous les objets qui l'entourent, comme les piétons, les panneaux de signalisation, les autres véhicules, etc. C'est ici que la détection d'objets entre en jeu ;

elle permet d'entraîner le véhicule à reconnaître ces objets et à réagir en temps réel en conséquence.

### 3.3.2 Industrie et robotique

La détection d'objets est largement utilisée en industrie pour l'automatisation de nombreuses tâches, comme le tri des produits en fonction de leur forme, taille, couleur etc, la détection des défauts tels que les fissures, les rayures, ou les erreurs de marquage, etc. Cette technologie est également nécessaire dans le domaine de la robotique, où les robots doivent traiter les données visuelles en temps réel et s'adapter rapidement aux changements dans l'environnement.

### 3.3.3 Sécurité et surveillance

La détection d'objets est devenue un outil essentiel dans les domaines de la sécurité et de la surveillance. Elle permet par exemple d'identifier rapidement des objets potentiellement dangereux, tels que des colis suspects ou des armes, et d'alerter les autorités en temps réel. De plus, elle facilite le suivi des mouvements dans les foules, la surveillance d'individus suspects et la détection de comportements inhabituels, comme des rassemblements non autorisés. En matière de sécurité routière, elle assure une surveillance continue du trafic, détecte les accidents en temps réel et identifie automatiquement les infractions, telles que les excès de vitesse ou les infractions du code de la route. Ainsi, ces applications renforcent la réactivité et l'efficacité des systèmes de sécurité.

### 3.3.4 Médecine

La détection d'objets est très présente dans le domaine de la santé. Elle est utilisée pour l'analyse des images médicales afin de détecter des structures anatomiques spécifiques, comme les organes, les vaisseaux sanguins et les articulations, ainsi que la détection des tumeurs et anomalies. Cette technologie joue un rôle cruciale dans la téléchirurgie. Les robots chirurgicaux utilisés sont équipés de systèmes de détection d'objets qui permettent de reconnaître et de manipuler des tissus spécifiques. Par exemple, un robot peut détecter et différencier les tissus sains des tissus malades, facilitant ainsi des interventions chirurgicales et réduisant les risques pour le patient.

## 3.4 Méthodes de détection d'objets

Grâce à l'adoption de diverses techniques, la détection d'objets connaît des progrès considérables en terme de précision et de vitesse au fil du temps. Cette approche comprend plusieurs méthodes, divisées en deux grandes catégories : classiques et récentes, comme illustré sur la figure 3.2 :

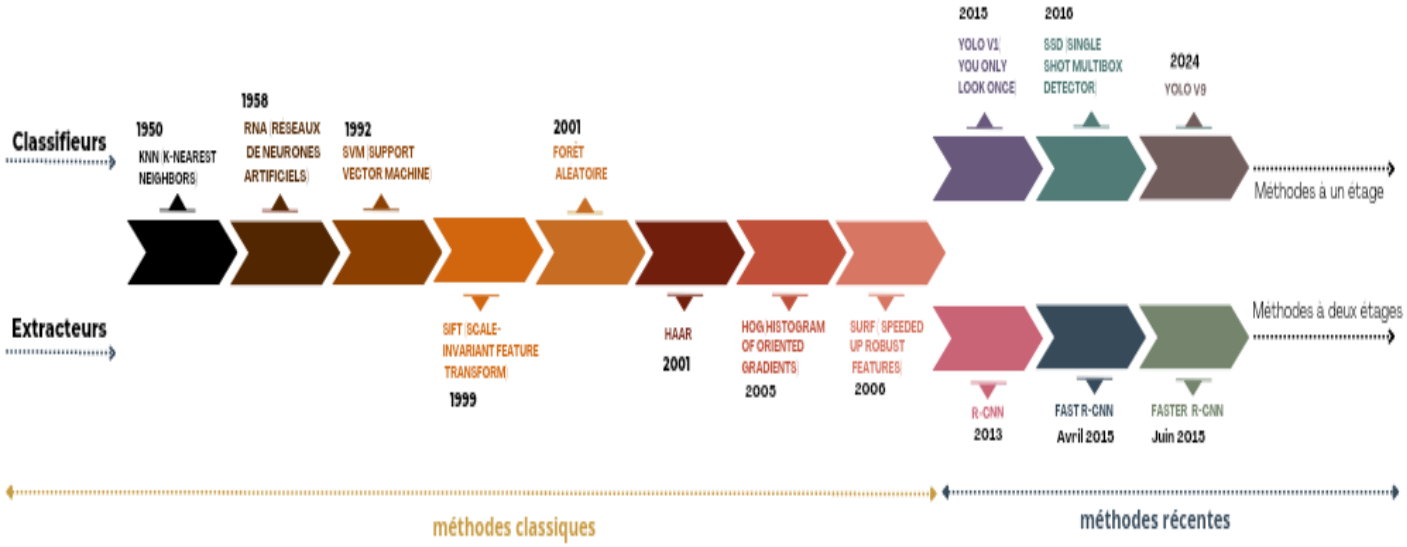



FIGURE 3.2 – Différentes méthodes de détection d'objets

### 3.4.1 Méthodes classiques

Les méthodes classiques de détection d'objets utilisent la technique de la fenêtre glissante, qui permet de parcourir l'image entière à différentes échelles et positions. Toutes les fenêtres issues de ce balayage sont représentées par des vecteurs de caractéristiques. Ces derniers sont soumis à l'entrée d'un classifieur pour décider de la présence ou non d'un objet d'intérêt dans ces fenêtres. Cependant, ces techniques sont souvent très coûteuses en termes de temps de calcul, car elles nécessitent de nombreuses itérations pour couvrir toutes les positions possibles, ce qui ralentit considérablement le processus de détection.

#### 3.4.1.1 Fenêtre glissante

La technique de la fenêtre glissante consiste à «faire glisser» une fenêtre généralement rectangulaire sur toute l'image, en examinant chaque position et échelle possible pour détecter la présence d'un objet.

La technique de la fenêtre glissante utilise deux approches principales pour détecter des objets de différentes tailles : la première consiste à faire glisser des fenêtres de différentes tailles à travers une image de taille fixe, tandis que la seconde approche applique une fenêtre de taille fixe sur plusieurs versions redimensionnées de l'image.

Après avoir choisi la taille de la fenêtre, on la positionne sur le coin supérieur gauche de l'image. La fenêtre ensuite glisse de manière itérative pour parcourir toute l'image, en se déplaçant horizontalement et verticalement selon un certain pas. À chaque position, les caractéristiques de la région sont extraites, puis analysées pour décider de la présence ou non de l'objet d'intérêt dans la fenêtre correspondante, comme illustré sur la figure 3.3.


FIGURE 3.3 – Principe de la fenêtre glissante

#### 3.4.1.2 Extracteurs de caractéristiques

Le processus de détection d'objets est conditionné par une étape très importante et judicieuse qui est l'extraction de caractéristiques en utilisant des extracteurs.

Les extracteurs de caractéristiques sont des algorithmes qui permettent d'extraire les caractéristiques représentant au mieux l'image telles que la texture, les contours, les coins, etc. Ces extracteurs se basent sur l'information de voisinage, on cite : SIFT, Haar, HOG, SURF. Le choix adéquat de l'extracteur est essentiel pour garantir une bonne détection des objets.

**a- SIFT**

L'algorithme SIFT (*Scale-Invariant Feature Transform*) a été proposé pour la première fois en 1999 par le chercheur David G.Lowe [19], puis amélioré en 2004. Cet extracteur est connu pour son invariance à l'échelle et à la rotation, permettant ainsi une extraction efficace des caractéristiques.

D'abord, SIFT procède à la construction d'une pyramide d'échelles. Il s'agit d'une représentation de l'image à différentes échelles. Ceci est réalisé en filtrant l'image afin de la flouter (grâce à un filtre gaussien), puis en réduisant sa taille (en appliquant des techniques

de *pooling*). Cette opération est réalisée à plusieurs reprises, jusqu'à ce qu'on obtienne une pyramide d'images à différents niveaux, chacun représentant l'image à une échelle différente. Le but de cette étape est de cerner les caractéristiques qui sont invariantes à l'échelle, c'est-à-dire qui peuvent être détectées indépendamment de la taille ou de la résolution de l'image. Afin de repérer ces caractéristiques (représentés par des point d'intérêt), pour chaque pixel de chaque niveau de la pyramide, les extrema d'échelle (valeurs maximales ou minimales locales) sont calculés. Ceci est réalisé en utilisant la technique de différence des Gaussiennes DoG, qui consiste à soustraire les images floutées consécutives à différentes échelles. Après cette étape, les points d'intérêt sont localisés précisément en appliquant la méthode de l'interpolation. À ce stade, il ne reste plus qu'à décrire ces points. Cela revient à assigner une orientation à chaque point (en calculant les orientations des gradients), puis à construire les descripteurs SIFT représentés par des vecteurs (en utilisant le calcul des histogrammes d'orientations des gradients)[20]. Ce processus est illustré par la figure 3.4 :

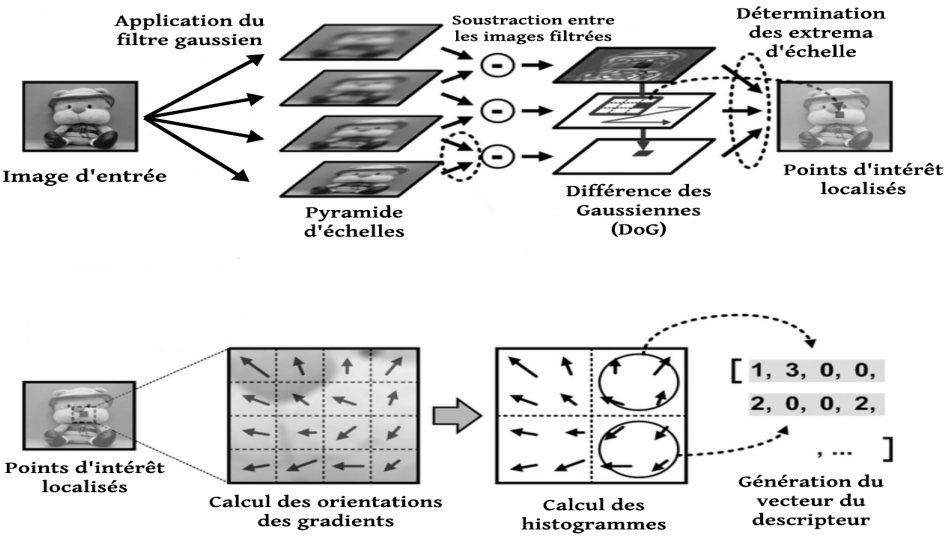



FIGURE 3.4 – L'algorithme SIFT

**b- Haar**

Les caractéristiques de Haar sont des motifs rectangulaires simples utilisés pour représenter les variations d'intensité des pixels dans une image. Ce type de caractéristiques a été exploité dans le détecteur de Viola-Jones qui a révolutionné le domaine de la détection de visages [21]. Les trois types des caractéristiques de Haar sont : Caractéristiques de bord (*Edge features*), caractéristiques de ligne (*Line features*) et caractéristiques de quatre rectangles (*Four rectangle features*). Ils sont représentés par la figure 3.5.

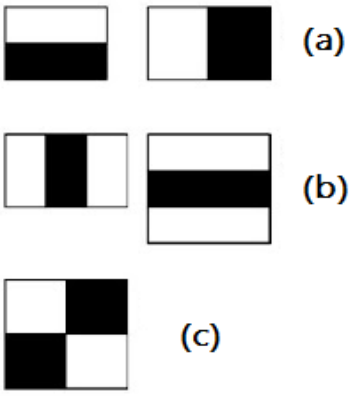



FIGURE 3.5 – Caractéristiques de Haar : caractéristiques de bord (a), caractéristiques de ligne (b), caractéristiques de quatre rectangles (c)

Une fenêtre glissante parcourt l'image à différentes échelles et positions, pour appliquer dans chaque région parcourue les caractéristiques de Haar. La réponse de chaque caractéristique dans chaque région est donnée par la différence entre la somme des pixels dans les zones sombres et claires, générant ainsi un ensemble de valeurs numériques représentant la réponse des caractéristiques à la région d'intérêt.

Pour diminuer le temps de calcul des caractéristiques, une représentation optimisée de l'image originale est utilisée ; il s'agit de " l'image intégrale " introduite pour la première fois dans le détecteur de Viola-Jones. Au lieu de faire des calculs complexes, l'utilisation de l'image intégrale permet de calculer rapidement les sommes des pixels pour différentes régions en effectuant simplement quelques opérations arithmétiques [22].

**c- HOG**

HOG (*Histogram Of Oriented gradients*) est un extracteur proposé par Navneet Dalal et Bill Triggs [23] en 2005. Il est basé sur le calcul des orientations des gradients dans différentes régions de l'image. Les orientations des gradients représentent les directions du changement d'intensité des pixels dans une image.

L'algorithme commence par diviser l'image en cellules. Ensuite, dans chaque cellule, les orientations des gradients des pixels sont calculées, puis regroupées dans un histogramme qui compte le nombre d'occurrences de chaque orientation des gradients dans la cellule. Les histogrammes des cellules sont ensuite normalisés (en utilisant la technique de normalisation par blocs), et enfin concaténés, pour former un long vecteur de caractéristiques qui représente l'ensemble de l'image, qu'on appelle la "caractéristique HOG" et qui sera présenté à un classifieur. Ce processus est représenté par la figure 3.6.

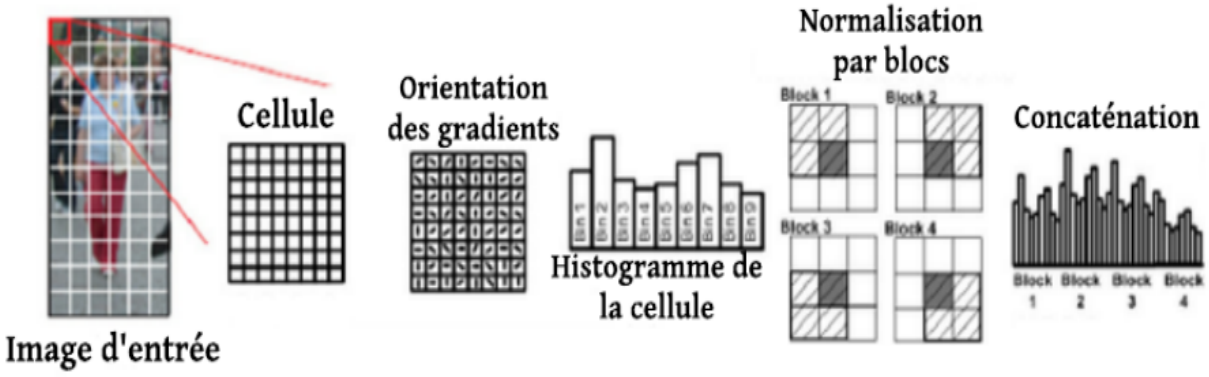



FIGURE 3.6 – L'algorithme HOG

**d- SURF**

Le descripteur SURF (*Speeded Up Robust Features*) a été proposé en 2006 par Herbert Bay et al.[24]. SURF est basé sur les principes fondamentaux de SIFT, cependant, il utilise des techniques plus améliorées, ce qui font de lui un extracteur trois fois plus rapide que son prédécesseur, et donc plus adapté à des applications en temps réel et au traitement d'une grande quantité d'images [25][26]. Parmi ces améliorations, on trouve l'introduction de la technique de l'approximation de la différence de boîte (*Box Filter Approximation*) au lieu de la différence gaussienne DoG (*Difference of Gaussian*) pour le calcul des extrema d'échelle. On cite également l'utilisation d'une matrice hessienne pour détecter les caractéristiques à différentes échelles contrairement à SIFT qui utilise l'approche de la pyramide d'échelle. (plus de details : [27])

#### 3.4.1.3 Méthodes de classification

Après l'étape d'extraction de caractéristiques, le vecteur de caractéristiques de chaque région de l'image est présenté à l'entrée d'un classificateur pour déterminer la présence ou non d'un objet d'intérêt et attribuer l'étiquette correspondante à celui-ci. Plusieurs types de classifieurs peuvent être utilisés, tels que les machines à vecteurs de support (*Support Vector Machine,* SVM), les k-plus proches voisins (*K-Nearest Neighbors,* KNN), les forêts aléatoires (*random forest*), et les réseaux de neurones artificiels (RNA).

* **Machines à vecteurs de support :** est un algorithme de *machine learning* supervisé très populaire, efficace pour les tâches de classification. Il vise à trouver un hyperplan optimal qui sépare deux classes dans un espace de caractéristiques. Dans le cas où les données sont linéairement séparables, cet algorithme vise à trouver l'hyperplan qui

maximise la marge séparant les deux classes. Cette marge représente la distance entre les points de données les plus proches de la frontière, appelés "vecteurs de support" et l'hyperplan lui-même, comme montré sur la figure 3.7. Dans le cas où les données ne sont pas linéairement séparables, SVM est accompagné de "*kernel trick*", une technique qui ramène les données non linéairement séparables vers un espace de dimension plus élevé où elles peuvent être linéairement séparables [28].

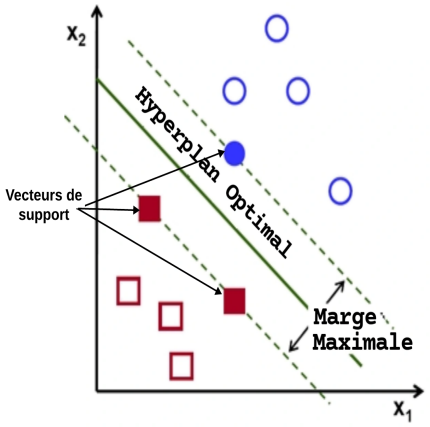



FIGURE 3.7 – Machines à vecteurs de support

∗ **K-plus proches voisins :** est une méthode de classification simple, efficace, et non paramétrique [29]. Le fonctionnement de cet algorithme supervisé est plutôt simple ; Il cherche à attribuer une classe à une donnée en se basant sur sa similarité avec des données déjà classifiées. Ceci se fait en calculant la distance entre le nouvel exemple et les autres exemples existants, puis sélectionner les k exemples voisins, ayant le plus de similarités avec ce dernier. La classe la plus représentée parmi les k voisins est celle à laquelle l'algorithme attribue le nouvel exemple. Le paramètre k est choisi par l'utilisateur. Ce processus est donné par la figure 3.8.

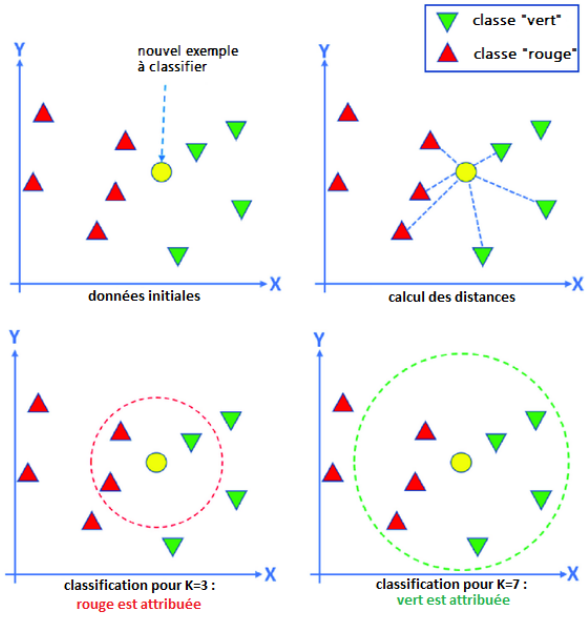



FIGURE 3.8 – Processus de l'algorithme K-plus proches voisins

* **Forêt aléatoire :** Il s'agit d'un algorithme d'apprentissage supervisé utilisé pour des tâches de classification. Une forêt aléatoire est constituée d'un ensemble d'arbres de décision indépendants. Ces arbres sont construits de manière aléatoire à partir des données d'entraînement, via une méthode appelée *bootstrap* (ou *bagging*). Cette méthode implique la formation d'un échantillon d'entraînement avec remplacement pour chaque arbre, ce qui signifie qu'une donnée pourrait figurer dans l'ensemble d'entraînement de plusieurs arbres. Chaque arbre comporte plusieurs nœuds pour lesquelles des sous-ensembles aléatoires de caractéristiques sont choisis. L'arbre sélectionne ensuite la meilleure caractéristique parmi ces sous-ensembles pour effectuer la division des nœuds assurant ainsi la diversification des arbres. Une fois que tous les arbres ont fait leur prédiction, la classe la plus courante sera assignée à la donnée ; c'est ce qu'on appelle vote majoritaire [30]. Ce processus est illustré dans la figure 3.9.

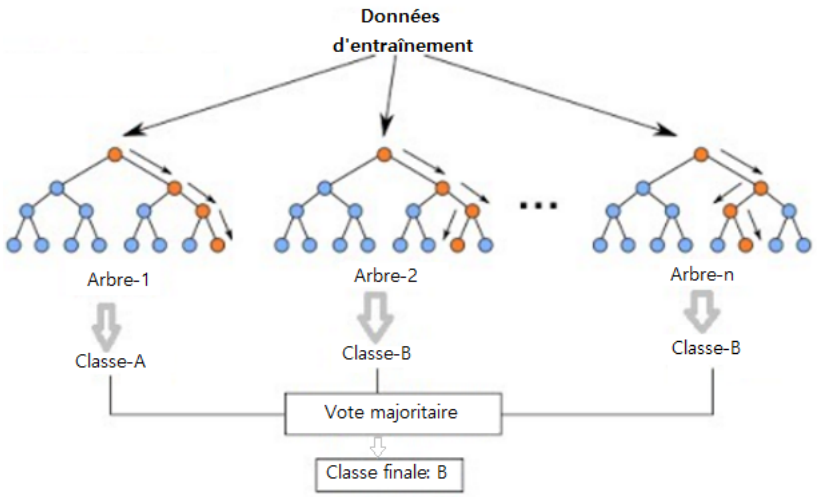



FIGURE 3.9 – Processus de la forêt aléatoire

* **Réseaux de neurones artificiels :** Les réseaux de neurones artificiels sont de puissants modèles utilisés pour de nombreuses tâches y compris la classification. Un RNA est composé de couches interconnectées, à savoir des couches cachées qui reçoivent les données d'entrées et qui effectuent des calculs sur ces dernières, suivies d'une couche de sortie qui génère la prédiction finale, tel que présenté dans le chapitre 1.

### 3.4.2 Méthodes récentes de détection d'objets

Bien que les méthodes classiques soient utilisées, ces dernières peuvent être coûteuses en termes de temps de calcul, car de nombreuses positions et échelles doivent être explorées. Par conséquent, des méthodes plus récentes basées sur les réseaux de neurones convolutifs ont été développées, pour une exécution rapide, à savoir détecter plusieurs objets simultanément dans une image, permettant ainsi leur traitement en temps réel. Cependant, le choix de l'architecture du CNN à utiliser demeure une difficulté à surmonter. Ces méthodes sont classées en deux catégories principales : les méthodes à un étage et les méthodes à deux étages, comme montré sur la figure 3.10.

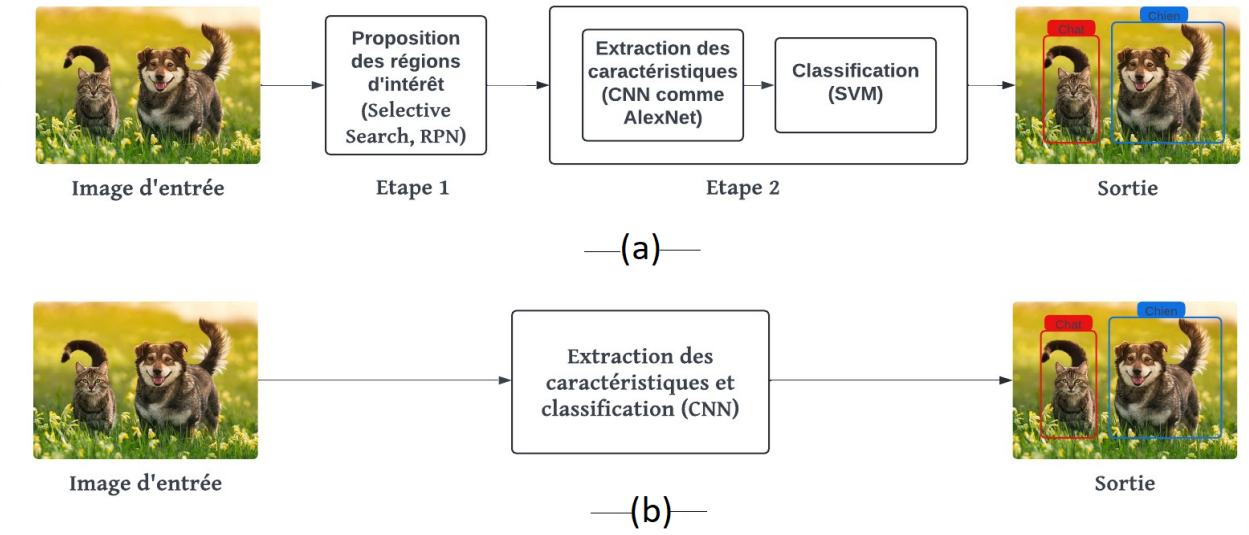



FIGURE 3.10 – Méthodes de détection d'objets à deux étages (a) et à un étage (b)

### 3.4.2.1 Méthodes à deux étages

Les méthodes à deux étages basées sur les CNN ont été une nouvelle avancée pour le processus de détection d'objets l'effectuant en deux étapes distinctes comme sur la figure 3.10.a. La première étape consiste à proposer des régions d'intérêt (*Region Proposals*) qui sont susceptibles de contenir des objets. La deuxième étape consiste à extraire les caractéristiques de ces régions d'intérêt et à soumettre ces caractéristiques à un classifieur. Parmi ces méthodes on trouve : les réseaux de neurones à convolution basés région (R-CNN : *Region-based Convolutional Neural Networks*), les réseaux de neurones à convolution basés région rapide (Fast R-CNN), et les réseaux de neurones à convolution basés région plus rapide (Faster R-CNN).

**3.4.2.1.1 R-CNN :** Alors que les CNN s'étaient déjà imposés dans les tâches de classification d'images, la détection d'objets nécessitait une solution plus complexe. C'est là que l'algorithme R-CNN est apparue, offrant une approche innovante pour la détection d'objets, en combinant la force des CNN à extraire les caractéristiques les plus pertinentes et celle de la méthode de la proposition des régions. Cette méthode se déroule en suivant les étapes montrées dans la figure 3.11.

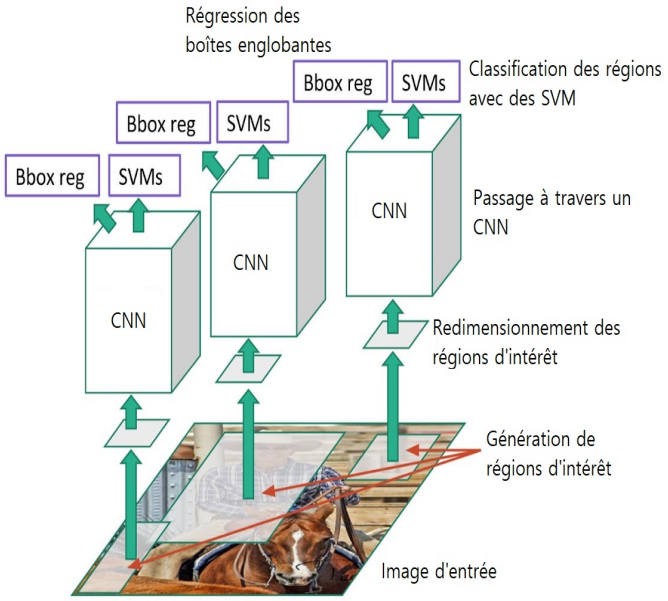



FIGURE 3.11 – Principe de la méthode R-CNN

**a. Génération de propositions de régions :** R-CNN commence en divisant l'image en plusieurs régions appelées d'intérêt (*Regions Of Interest)*. Différentes techniques peuvent être utilisées pour générer ces propositions de régions, telles que la méthode de la recherche sélective *(Selective Search)* qui cherche des régions de l'image présentant des similitudes de texture, de couleur ou de forme. Cette méthode génère un ensemble restreint de propositions de régions, d'environ 2000 [31].

**b. Extraction de caractéristiques :** Chaque région d'intérêt générée précédemment est redimensionnée, et est ensuite présentée à un réseau de neurones convolutif pré-entraîné, tel qu'AlexNet pour extraire le vecteur de caractéristiques de chaque région.

**c. Classification :** Le vecteur de caractéristiques est introduit dans un classifieur binaire du type SVM, afin de classifier chaque région selon la présence ou l'absence d'un type spécifique d'objet.

**d. Régression des boîtes englobantes :** Les régions contenant des objets sont ensuite ajustés avec la régression du cadre de sélection en utilisant le principe de suppression non maximale pour produire des boites englobantes finales pour les emplacements exactes d'objets.

Il est à noter que l'étape d'extraction de caractéristiques de chaque proposition de région séparément prend beaucoup de temps, en plus de l'utilisation d'autant de classifieurs SVMs que d'objets recherchés. Bien que cette approche fonctionne efficacement en terme de précision, son calcul reste toujours coûteux. Ainsi des variantes telles que Fast R-CNN et Faster R-CNN ont été développées pour améliorer la vitesse de détection tout en maintenant une

précision élevée.

**3.4.2.1.2 Fast R-CNN :** Fast R-CNN est une version améliorée de la méthode R-CNN, elle a introduit des améliorations qui ont permit d'augmenter ses performances en termes de vitesse et de précision.

Les étapes de ce processus sont illustrées par la figure 3.12.

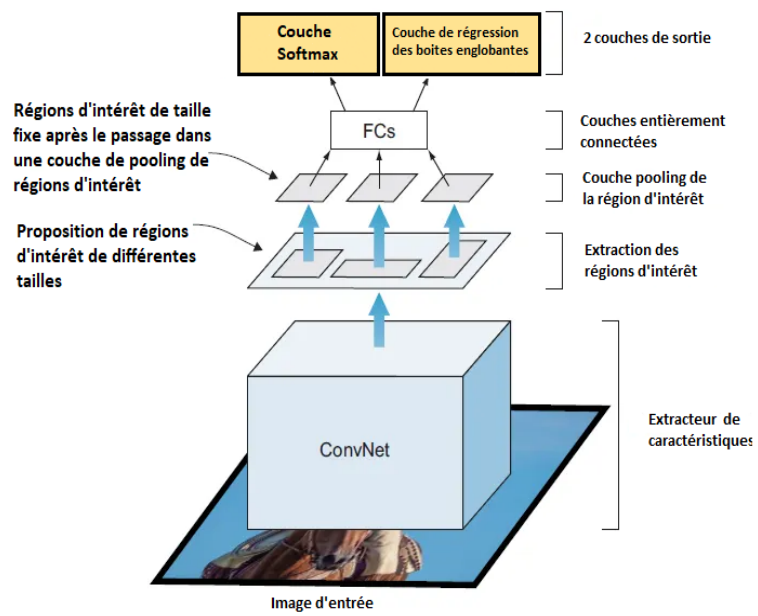



FIGURE 3.12 – Principe de la méthode Fast R-CNN

Un réseau Fast R-CNN prend en entrée une image entière ainsi qu'un ensemble de régions d'intérêt. Le réseau traite d'abord l'image entière avec plusieurs couches de convolution et de max pooling pour produire une carte des caractéristiques. Ensuite, à partir de la carte de caractéristiques produite, une couche de *pooling* de la région d'intérêt extrait un vecteur de caractéristiques de longueur fixe. Chaque vecteur de caractéristiques est ensuite alimenté dans une séquence de couches entièrement connectées (FC) qui se divisent finalement en deux couches de sortie :

* Une couche softmax qui produit des probabilités pour les K classes d'objets ainsi qu'une classe supplémentaire pour les images qui ne correspondent à aucune des classes spécifiques, appelée classe "arrière-plan".
* Une couche qui effectue la régression des boîtes englobantes, qui consiste à prédire leurs coordonnées qui sont données par quatre nombres réels [32].

Au lieu de traiter chaque région d'intérêt séparément, Fast R-CNN qui est une version améliorée de R-CNN partage des fonctionnalités convolutives entre différentes régions, accélérant

considérablement le processus de détection.

**3.4.2.1.3 Faster R-CNN :** Faster R-CNN a apporté des améliorations significatives par rapport aux versions précédentes. Cette méthode a introduit le RPN (*Region Proposal Network)*. Il s'agit d'un réseau de neurones convolutifs, qui analyse les caractéristiques de l'image et génère automatiquement les régions d'intérêt dans l'image sans avoir besoin d'utiliser des méthodes externes de proposition de régions. Ce système est donc constitué de deux module : le RPN et le détecteur Fast R-CNN qui utilise les régions proposées [33]. L'ensemble du système est un réseau unique et unifié pour la détection d'objets comme montré sur la figure 3.13.

Bien que très précis et plus rapide que ses prédécesseurs (RCNN et Fast RCNN), Faster R-CNN peut être plus lent que les méthodes à un étage en raison de l'étape de proposition de régions.

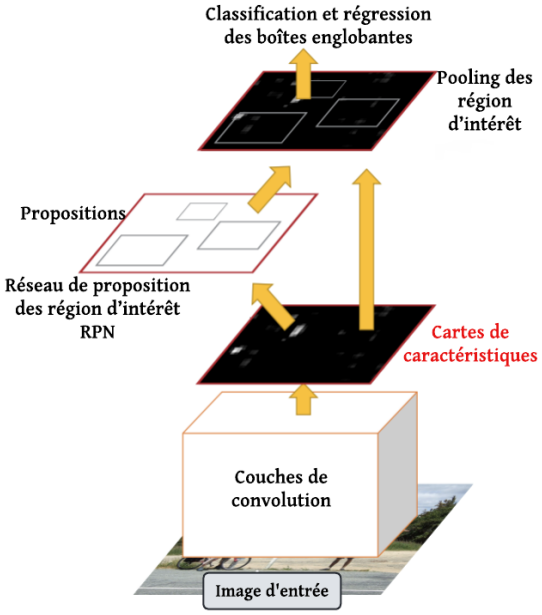



FIGURE 3.13 – Principe de la méthode Faster R-CNN

### 3.4.2.2 Méthodes à un étage

Ces méthodes, à savoir SSD (*Single Shot Detector*) et YOLO (*You Only Look Once*) effectuent la détection d'objets en une seule étape (voir la figure 3.10.b), directement via des caractéristiques extraites à partir de l'image d'entrée sans passer par l'étape de proposition de régions d'intérêt. Ces méthodes à un étage qui privilégient la vitesse à la précision sont préférées pour les applications en temps réel, contrairement aux méthodes à deux étages qui

sont adaptées aux tâches qui privilégient la précision à la vitesse. Dans notre travail, nous nous sommes intéressés à l'étude de YOLO.

## 3.5 Modèle YOLO

YOLO a été introduit pour la première fois en 2015 par Joseph Redmon, Santosh Divvala, Ross Girshick et Ali Farhadi [34] et depuis subi plusieurs améliorations. C'est le premier détecteur capable de traiter des vidéos en temps réel, étant donner qu'il ne considère pas l'étape de proposition de régions des méthodes précédentes. En effet, YOLO est composé d'un seul réseau neuronal convolutif appliqué sur l'image entière en un seul passage, d'où l'appellation "vous ne regardez qu'une fois" (*You Only Look Once*).

### 3.5.1 Architecture de YOLO

YOLO est conçu pour la détection d'objets en temps réel. L'architecture de sa première version, YOLOv1, prend en entrée une image redimensionnée à 448 x 448 pixels. Ce modèle se compose de 24 couches convolutives utilisant des filtres de 7x7 et 3x3, ainsi que des couches de *max pooling* pour réduire la taille des cartes de caractéristiques. Les 20 premières couches sont pré-entraînées sur l'ensemble de données de classification ImageNet (1000 classes), suivies de 4 couches convolutives supplémentaires. Deux couches entièrement connectées sont ajoutées [34], comme montré sur la figure 3.14.

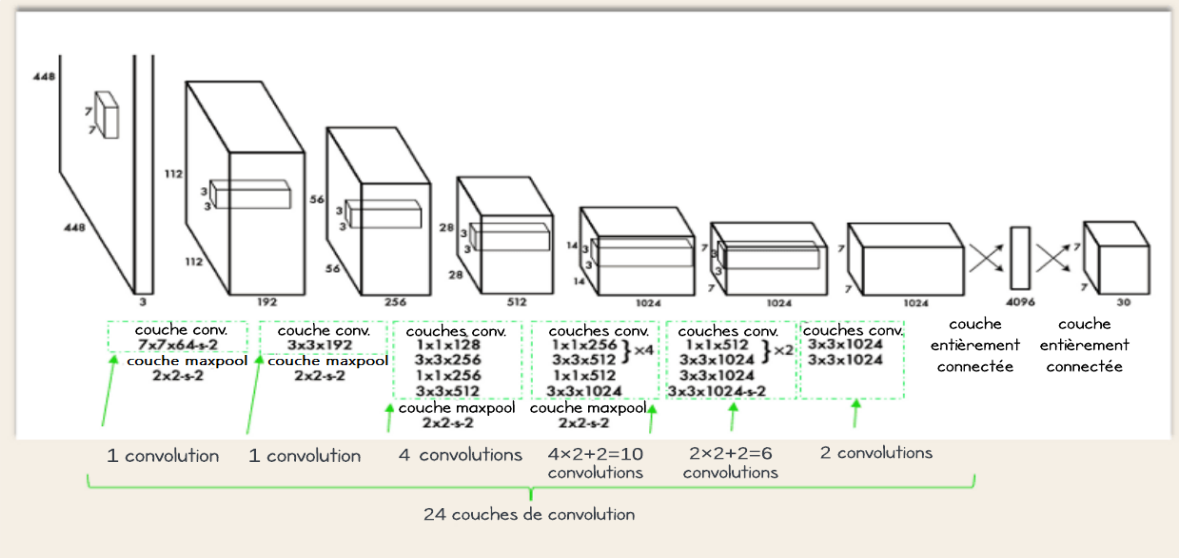



FIGURE 3.14 – Architecture YOLOv1

### 3.5.2 Fonctionnement de YOLO

YOLOv1 propose une approche unique pour effectuer à la fois la localisation et la classification des objets dans une image. YOLO exploite les caractéristiques de l'image pour générer simultanément les boîtes englobantes de tous les objets présents dans l'image. Ceci fait de YOLO est un détecteur d'objets en temps réel, capable d'analyser 45 images par seconde [34]. Le fonctionnement de YOLO repose sur la division d'une image en une grille de SxS. Pour chaque cellule de la grille, il génère B boîtes englobantes. Pour chacune d'elles, le réseau prédit un score de confiance de présence d'objets dans la boîte ainsi que les coordonnées de celle-ci. Les boîtes englobantes dont le score de confiance dépasse un seuil donné sont sélectionnées et utilisées pour localiser les objets dans l'image. Son fonctionnement implique les étapes suivantes : le passage de l'image à travers le CNN, le filtrage des prédictions, la Suppression Non-Maximale et l'affichage du résultat de la détection.

#### 3.5.2.1 Passage à travers le CNN

Dans le modèle YOLOv1, les images sont redimensionnées à une taille fixe, typiquement 448x448 pixels. Une fois la taille de l'image d'entrée est ajustée, celle-ci est propagée à travers un CNN composé de plusieurs couches de convolution et de *pooling*. Ces couches extraient des caractéristiques à différents niveaux d'abstraction, allant des détails fins aux motifs plus complexes. À la sortie de la couche de convolution, plusieurs cartes de caractéristiques sont générées, chacune capturant des aspects essentiels pour la classification et la localisation des objets dans l'image. Ces cartes de caractéristiques sont combinées via la couche entièrement connectée par une simple régression linéaire, produisant en sortie une grille de cellules SxS, où chaque cellule prédit les coordonnées des B boîtes englobantes, les scores de confiance et les probabilités d'appartenance aux C classes des objets présents dans ces boites. Les prédictions de chaque cellule sont organisées en un vecteur puis affinées grâce à des post-traitements, comprenant le filtrage des prédictions et la suppression non-maximale. Ce processus intégré confère à YOLOv1 sa rapidité et son efficacité dans la détection d'objets.

#### 3.5.2.2 Filtrage des prédictions

Après la détection initiale, un seuil de confiance, également appelé critère de rejet, est appliqué pour filtrer les prédictions en fonction de leur score de confiance. Chaque prédiction reçoit un score entre 0 et 1, représentant la probabilité de présence de l'objet. Si ce score est inférieur au seuil défini, la prédiction est rejetée. Ce mécanisme améliore la précision du modèle en éliminant les détections moins fiables. Le choix du seuil est crucial : un seuil élevé améliore la précision mais peut exclure des détections valables, tandis qu'un seuil bas détecte

plus d'objets mais peut inclure des erreurs.

#### 3.5.2.3 Suppression Non-Maximale

Après filtrage, plusieurs boites englobantes sont détectées autours d'un même objet. Pour éliminer les détections redondantes, YOLOv1 utilise la technique de suppression non maximale (NMS). Cette méthode conserve uniquement les boîtes englobantes les plus pertinentes en supprimant celles qui se chevauchent excessivement avec des boîtes ayant un score de confiance plus élevé. La mesure de *Intersection over Union* (IoU) est utilisée pour évaluer le chevauchement entre deux boîtes en calculant le rapport entre leur intersection et leur union comme montré sur la figure 3.15. Si l'IoU dépasse un seuil prédéfini, la boîte avec le score de confiance le plus bas est éliminée. Ainsi, NMS assure que chaque objet est représenté par une seule boîte englobante, réduisant les doublons et optimisant la précision des détections finales.

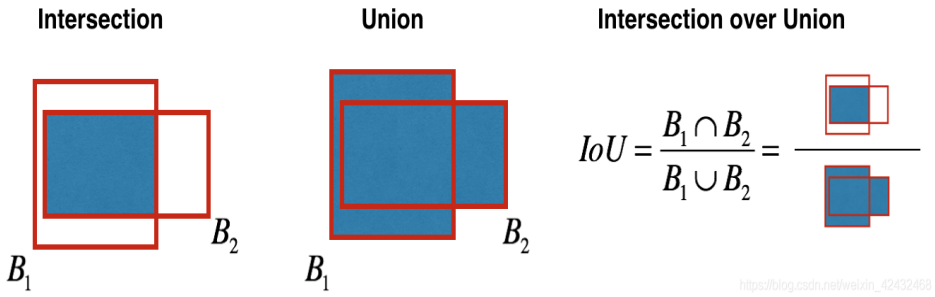



FIGURE 3.15 – La mesure d'Intersection over Union

#### 3.5.2.4 Résultat de détection

Les boîtes englobantes finales et les classes correspondantes à leurs scores de confiance sont renvoyées en sortie (voir la figure 3.16).

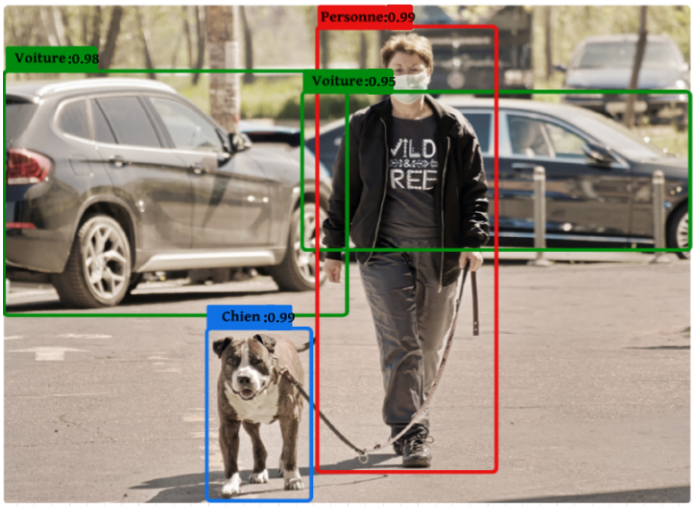



FIGURE 3.16 – Résultats de détection

## 3.6 Variantes de YOLO

Le modèle YOLO de base, aussi connu comme YOLOv1 a été amélioré plusieurs fois depuis sa sortie initiale en 2015, créant ainsi une famille YOLO de plusieurs versions, chacune s'appuie sur son prédécesseur et l'améliore. Voici une chronologie présentant le développement de YOLO au cours des dernières années :

### 3.6.1 YOLOv2 (YOLO9000)

YOLOv2 est la première version améliorée du modèle d'origine, avec une capacité de détecter plus de 9000 catégories d'objets. Il a atteint une précision de 76.8 % pour 67 images par seconde (FPS) sur la base de données PASCAL VOC2007 [17]. Parmi les améliorations que cette nouvelle version a apporté on cite :

1. Normalisation par lots : L'utilisation de cette technique améliore la convergence et réduit le sur-apprentissage du modèle, tout en éliminant le besoin d'utiliser d'autres formes de régularisation, augmentant ainsi le mAP de 2% [17].
2. Entraînement multi-échelles : Le modèle est entraîné sur différentes tailles d'images d'entrée, ce qui le force à apprendre à prédire des objets efficacement à différentes résolutions.
3. YOLOv2 abandonne l'utilisation des couches entièrement connectées pour la prédiction. Au lieu de cela il utilise les couches convolutives finales pour la classification. Les cartes de caractéristiques produites par les couches convolutives sont directement utilisées pour prédire les classes d'objets et les boîtes englobantes.

4. YOLOv2 introduit les boîtes d'ancrage, un ensemble de boîtes englobantes prédéfinies de différentes tailles et formes pour mieux gérer les variations de taille et d'aspect des objets. Cela permet une prédiction plus précise des boîtes englobantes par rapport aux méthodes de YOLOv1.

### 3.6.2 YOLOv3

YOLOv3 a été annoncé dans l'article YOLOv3 : *An Incremental Improvement* en 2018 par les mêmes concepteurs de YOLO. Il a apporté quelques améliorations aux versions précédentes, entraînant plus de précision et de vitesse. Parmi les améliorations introduites on cite [35] [36] :

1. Détection à différentes échelles : YOLOv3 effectue la détection d'objets à trois échelles différentes, et utilise le concept de la pyramide des caractéristiques (*Feature Pyramid Network* (FPN)) pour extraire les caractéristiques de ces différentes échelles.

2. Introduction d'un nouveau réseau CNN : Darknet-53 est le nouveau réseau d'extraction des caractéristiques utilisé dans YOLOv3. Ce réseau possède 53 couches convolutives, et certaines de ses connexions sont résiduelles, faisant de lui une approche hybride entre les deux réseaux Darknet-19 et ResNet, qui est également plus forte et performante.

3. Prédiction des scores de confiance : YOLOv3 utilise la technique de régression logistique pour la prédiction des scores de confiance des boîtes englobantes.

### 3.6.3 YOLOv4

YOLOv4 est une version améliorée de YOLOv3, apportant des améliorations significatives qui seront vues en détails dans le prochain chapitre.

### 3.6.4 Autres variantes de YOLO

Le reste des versions de YOLO sont résumées dans le tableau 3.1.

| Variantes | Améliorations apportées |
|---|---|
| YOLOv5[36] | * Développé en utilisant PyTorch au lieu de Darknet. |
| Scaled-YOLOv4 [37][36] | * Nouvelles techniques de mise en échelle. |
| YOLOR[38][36] | * Nouvelle approche en un seul modèle multi-tâches. |
| YOLOX[39][36] | * Nouvelle approche sans boîtes d'ancrage. |
| YOLOv6 [40][36] | * Approche sans boîtes d'ancrage.<br>* Fournit des modèles de tailles différentes pour les applications industrielles. |
| YOLOv7 [41][36] | * Performances remarquables.<br>* Amélioration des méthodes *BOF* de manière à augmenter la précision sans augmenter le coût. |
| DAMO-YOLO [42] [36] | * Introduction de nouvelles technologies pour renforcer la détection en temps réel. |
| YOLOv8 [36] | * Prédiction de moins de boîtes englobantes.<br>* Approche sans boîtes d'ancrage.<br>* Processus de suppression non-maximale très rapide. |
| PP-YOLO [43][36] | * Série de 3 modèles, basée sur YOLOv3.<br>* Utilisation de 10 astuces pour augmenter la précision sans sacrifier la vitesse. |
| YOLOv9 [44] | * Programmable Gradient Information (PGI)<br>* *Generalized Efficient Layer Aggregation Network* (GELAN) |
| YOLOv10 [45] | * Affectations doubles cohérentes pour une formation sans NMS<br>* Stratégie de conception de modèle holistique axée sur l'efficacité et la précision. |

TABLE 3.1 – Différentes versions de YOLO

## 3.7 Conclusion

Ce chapitre nous permet de conclure que la détection d'objets est un domaine de recherche très actif qui attire de plus en plus l'attention en raison de son large éventail d'applications. Cette technologie, qui consiste à localiser et à classifier les objets dans des images ou des vidéos, est en constante évolution grâce à des algorithmes de pointe qui ont considérablement amélioré sa précision et sa vitesse en devenant de plus en plus robustes et adaptables à différents scénarios. Parmi ces algorithmes nous trouvons YOLO que nous avons traité en détails dans ce chapitre et dont les résultats de l'implémentation de sa version 4 (YOLOv4) feront l'objet d'étude dans le chapitre suivant.

# Chapitre 4

# Implémentation et résultats de YOLOv4

## 4.1 Introduction

La détection d'objets est un domaine clé de la vision par ordinateur, ayant bénéficié d'avancées majeures grâce à l'évolution des algorithmes de l'apprentissage profond. Parmi les méthodes les plus innovantes, l'algorithme YOLO qui se distingue par sa capacité à réaliser des détections en temps réel avec une bonne précision. Depuis son introduction en 2015 par Joseph Redmon, YOLO a subit différentes améliorations. Par conséquent, plusieurs versions ont été développées. Dans le cadre de notre étude, nous avons choisi de nous concentrer sur YOLOv4.

Ce chapitre présente les différentes étapes de son implémentation, y compris les outils, les paramètres et les bases de données utilisées pour mener notre étude. Nous expliquerons également l'environnement de développement et le langage de programmation choisis pour réaliser nos expériences. Enfin, nous discuterons des résultats obtenus lors de notre expérimentation, nous analyserons les performances de la technique utilisée et nous explorerons les perspectives d'avenir pour améliorer YOLOv4 dans la détection d'objets.

## 4.2 YOLOv4

Conçu en 2020, YOLOv4 apporte beaucoup d'améliorations et de concepts innovants à la famille YOLO tout en préservant son fonctionnement de base. En combinant vitesse, précision, et accessibilité en termes de ressources, il offre des performances de pointe sans nécessiter une infrastructure coûteuse. L'architecture optimisée de YOLOv4 ainsi que les techniques innovantes qu'il intègre lui permettent de fonctionner efficacement en temps réel avec un seul GPU, contrairement à d'autres modèles comme Faster R-CNN et SSD qui nécessitent des configurations et ressources plus complexes pour des performances comparables [46].

#### 4.2.0.1 Architecture et fonctionnement

Comme tous les autres détecteurs d'objets, YOLOv4 est composé d'un réseau de base (*backbone*) pour l'extraction de caractéristiques, une partie intermédiaire (*neck*) pour créer des représentations de caractéristiques à différentes échelles et une partie détection (*head*) pour effectuer la détection des objets [46], comme montré sur la figure 4.1.

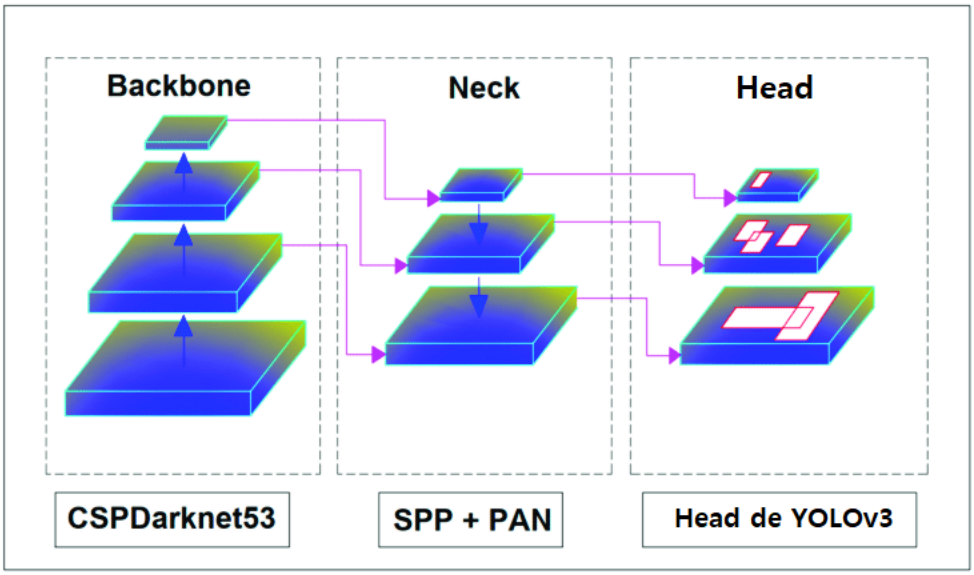



FIGURE 4.1 – Architecture de YOLOV4

* ***Backbone*** **:** CSPDarknet53 est le *backbone* de YOLOv4 utilisé comme extracteur de caractèristiques. Il s'agit d'une version améliorée du réseau Darknet-53, auquel on a ajouté le concept CSP (*Cross Stage Partial*) qui lui même est une amélioration du réseau DenseNet. CSP consiste à diviser les caractéristiques extraites en deux parties, puis à combiner ces deux parties dans des étapes ultérieures du réseau, ce qui permet de réduire la redondance des gradients et d'améliorer la capacité d'apprentissage. En combinant les caractéristiques de ces deux réseaux, YOLOV4 bénéficie d'un *backbone* puissant qui permet une extraction des caractéristiques efficace.
* ***Neck*** **:** Le *neck* de YOLOv4 intègre à la fois SPP (*Spatial Pyramid Pooling*) et PA-Net (*Path Aggregation Network*), qui sont des couches supplémentaires utilisées pour extraire différentes cartes de caractéristiques à différentes échelles du réseau *backbone*. SPP utilise plusieurs niveaux de pooling avec différentes tailles de fenêtres pour extraire des caractéristiques de l'image à différentes résolutions et générer une sortie de taille fixe, et PANet fusionne les caractéristiques extraites à différentes échelles en intégrant des chemins d'agrégation entre les couches du réseau.
* ***Head*** **:** Le *head* de YOLOv4 est similaire à celui de YOLOv3. Il utilise des couches

de convolutions à 3 échelles, pour effectuer les prédictions qui sont données par des boîtes englobantes, des scores de confiance, et des classes pour chaque objet détecté. Cette approche de détection multi-échelles offre au modèle la possibilité de détecter des objets de tailles variées (petis, moyens, grands).

#### 4.2.0.2 Techniques innovantes

YOLOv4 se distingue des autres modèles de détection d'objets par de nombreuses techniques innovantes, renforçant ainsi son architecture et optimisant ses performances. Ces techniques sont réparties à travers les différentes couches du réseau et sont regroupées en deux catégories : *Bag of Freebies* (BoF) et *Bag of Specials* (BoS).

* ***Bag of freebies*** **:** Comporte les différentes techniques qui modifient seulement la stratégie d'entraînement sans augmenter le temps d'inférence. Les méthodes incluses dans BoF comme l'augmentation de données (par exemple CutMix et Mosaic) et la régularisation (comme DropBlock), améliorent les performances du modèle en réduisant le sur-apprentissage et en augmentant la diversité des données d'entraînement.
* ***Bag of Specials*** **:** BoS fait référence à des techniques avancées qui augmentent légèrement le coût d'inférence mais améliorent significativement la précision de la détection d'objets[46]. Cela inclut l'utilisation de blocs de convolution spécialisés comme CSPDarknet53, ainsi que d'autres mécanismes tels que SPP et PANet , qui sont conçus pour capturer les détails importants et fusionner efficacement les caractéristiques multi-échelles, ce qui améliore la qualité globale des prédictions.

## 4.3 Environnement et langage d'implémentation

Pour l'implémentation de YOLOv4, la préparation de l'environnement de travail et le choix du langage de programmation sont cruciaux. Nous allons voir dans cette partie comment ces aspects sont préparés pour garantir l'efficacité et la performance du modèle.

### 4.3.1 Google Colab

Google Colab, ou Colaboratory, est une plateforme gratuite de Google qui permet d'exécuter du code Python dans un environnement de notebook Jupyter hébergé dans le cloud. Développé par Google Research en 2017, Google Colab offre un accès gratuit à des ressources de calcul puissantes, telles que des GPU (*Graphic Processing Units*) et des TPU (*Tensor Processing Unit*), facilitant ainsi la formation et la recherche en apprentissage automatique, et l'analyse de données, sans nécessiter de configuration matérielle ou logicielle complexe.

Les utilisateurs peuvent écrire et exécuter du code Python directement depuis leur navigateur, avec de nombreuses bibliothèques préinstallées comme TensorFlow, PyTorch, NumPy, et pandas. Colab est également intégré à Google Drive, facilitant le stockage, le partage et la collaboration en temps réel sur des notebooks.

Sa capacité à gérer des projets complexes, combinée à ses fonctionnalités de collaboration et à son stockage en nuage, en fait un outil puissant et accessible pour les chercheurs, les développeurs et les étudiants [47] [48] [49].

**GPU T4 :** Fait référence à un type de GPU de la série T4 de NVIDIA, disponible sur Google Colab. Il est conçu pour offrir un excellent compromis entre performance et coût. Il est équipé de cœurs Tensor et CUDA, qui optimisent les opérations de calcul en parallèle et accélèrent les tâches complexes, en particulier dans les domaines de l'apprentissage automatique. Grâce à cette architecture avancée, le GPU T4 permet des gains significatifs en termes de vitesse rendant les environnements comme Google Colab particulièrement puissants pour les chercheurs et les développeurs [50].

### 4.3.2 Python

Python, créé par Guido van Rossum en 1991, est un langage de programmation de haut niveau, interprété et polyvalent, réputé pour sa syntaxe claire et lisible, ce qui facilite son apprentissage et la compréhension du code. Ce langage est utilisé dans divers domaines, notamment le développement web, le développement logiciel, la science des données et l'intelligence artificielle. Python se distingue particulièrement par son support des *frameworks* de *deep learning* et sa vaste collection de bibliothèques spécialisées telles que TensorFlow, PyTorch, Keras et OpenCV, qui sont essentielles pour créer, entraîner et évaluer des modèles de détection d'objets comme YOLO. Grâce à une communauté mondiale de développeurs très active, Python offre une abondance de ressources, de tutoriels et de forums d'entraide, permettant ainsi de trouver facilement des solutions aux problèmes courants et d'accéder à des implémentations existantes. Cette richesse en outils et en support contribue à la croissance continue de Python en tant que langage open source de premier plan pour les projets complexes en apprentissage automatique.

#### 4.3.2.1 Bibliothèques

Pour travailler efficacement avec Python dans le contexte de la vision par ordinateur, notamment avec le modèle YOLO, il est essentiel de connaître certaines bibliothèques et *frameworks* clés. Voici les principaux :

* **NumPy :** est une bibliothèque cruciale pour les opérations mathématiques et le traitement des tableaux multidimensionnels. Elle est souvent utilisée en conjonction avec OpenCV pour manipuler les images et les données dans le contexte de la vision par ordinateur.
* **Pandas :** est une bibliothèque qui facilite la manipulation et l'analyse de données tabulaires.
* **Matplotlib :** est une bibliothèque qui permet la création de graphiques, ainsi que l'affichage d'images et de boîtes englobantes pour visualiser la sortie de notre modèle YOLO.
* **OpenCV :** est une bibliothèque open-source largement utilisée qui permet de manipuler des images, de détecter des objets et de reconnaître des formes, tout en facilitant la création de boîtes englobantes autour des objets détectés. Elle est souvent employée en conjonction avec des modèles de détection d'objets comme YOLO pour le traitement des images et la gestion des flux vidéo, ce qui en fait un outil indispensable pour les applications de vision par ordinateur.
* **Keras :** est une bibliothèque Python, offrant un excellent support pour le traitement des données et la visualisation. Elle prend en charge divers types de réseaux neuronaux avec une structure modulaire, permettant une grande flexibilité. Keras est une API de haut niveau qui fournit une interface pour TensorFlow, PyTorch ou Theano, et peut être utilisée pour construire et entraîner notre modèle YOLO.
* **Tensorflow :** est une bibliothèque open-source développée par Google Brain, lancée en novembre 2015. Conçu pour l'apprentissage en profondeur et les réseaux de neurones. Il est largement utilisé et s'est rapidement imposé comme l'un des frameworks les plus populaires dans ce domaine. Le nom « TensorFlow » provient des tenseurs, des tableaux multidimensionnels utilisés pour représenter les données dans les opérations sur les réseaux de neurones. TensorFlow s'intègre bien avec Google Cloud et prend en charge les CPU, GPU, ainsi que les TPU pour le calcul intensif. Bien que sa prise en main puisse être complexe, sa flexibilité et ses capacités étendues en font un outil puissant et incontournable. Il est devenu un pilier pour de nombreux produits de Google, tels que Gmail, Google Photos, et la reconnaissance vocale, en offrant des fonctionnalités robustes pour des tâches comme la classification d'images.
* **Darknet :** est un *framework* open-source de *deep learning*, développé en C et CUDA, réputé pour sa simplicité et son efficacité, particulièrement dans le domaine de la vision par ordinateur, notamment pour la détection d'objets. Connu pour sa légèreté et ses performances, Darknet est souvent associé aux modèles YOLO, qui sont utilisés pour la détection d'objets en temps réel. Bien que principalement écrit en C, Darknet peut

être intégré avec Python à l'aide de *wrappers*, offrant ainsi une flexibilité accrue. Il est apprécié pour sa rapidité, sa facilité d'installation et sa capacité à tirer parti des GPU grâce à CUDA, ce qui le rend particulièrement adapté aux applications nécessitant des performances en temps réel.

Parmi toutes les bibliothèques évoquées précédemment, celles que nous allons utiliser incluent OpenCV, NumPy, Matplotlib et darknet. Elles fourniront des fonctionnalités essentielles pour l'implémentation de notre modèle et l'analyse de ses résultats.

### 4.3.3 LabelMe

LabelMe est un outil open source dédié à l'annotation d'images, utilisé dans les projets de vision par ordinateur. Il permet de créer des annotations manuelles pour toutes images sous forme de boites englobantes enregistrées dans un fichier texte au format JSON, à l'aide d'une interface web simplifiée. Ce fichier JSON contient les propositions des boîtes englobantes des objets d'intérêt présents dans l'image, exprimées en coordonnées du coin supérieur gauche et du coin inférieur droit. Il inclut également les numéros des classes correspondantes à ces objets, ainsi que d'autres informations complémentaires, comme montré sur la figure 4.2.

LabelMe est apprécié pour sa facilité d'utilisation et sa capacité à gérer plusieurs annotations dans une seule image, ce qui en fait un outil précieux pour la création de bases de données annotés.

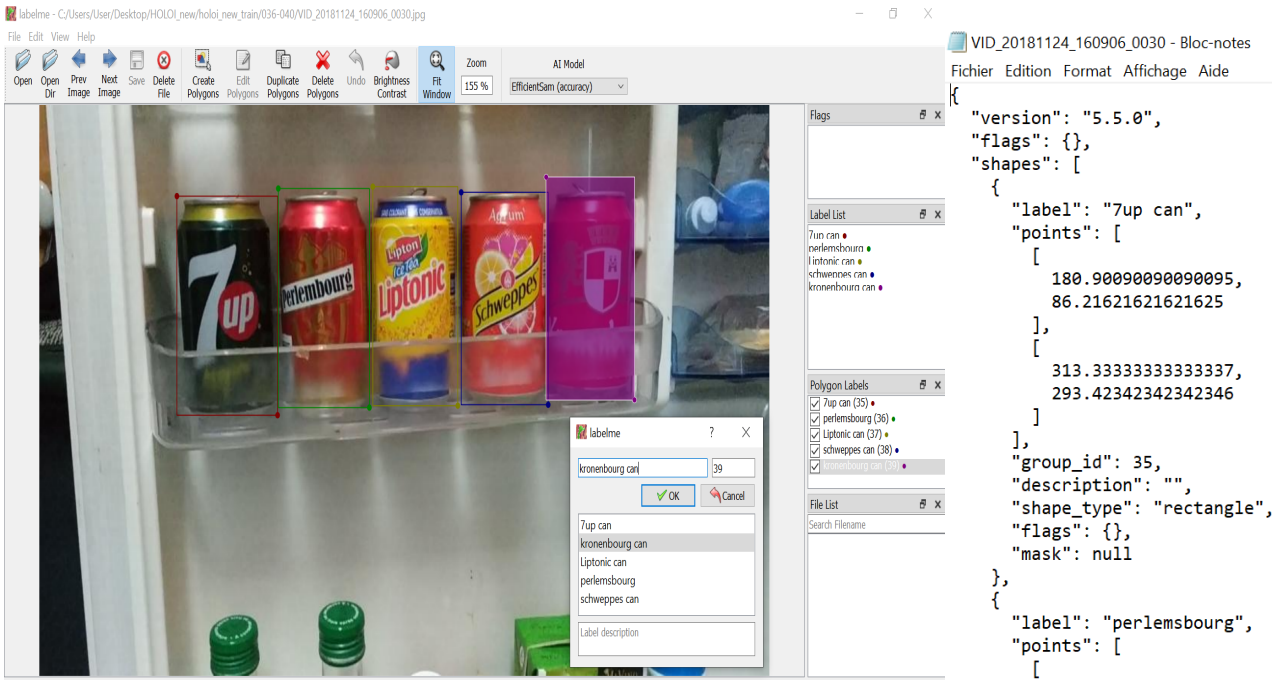



FIGURE 4.2 – Aperçu d'une fenêtre LabelMe et d'un fichier JSON

Pour implémenter YOLO sur une quelconque base d'images, ces dernières doivent être annotées au préalable sous le format YOLO. Ainsi la conversion du format JSON au format YOLO est inévitable. Chaque ligne, du fichier d'annotation sous format YOLO, décrit un objet de la manière suivante (voir figure 4.3) :

" « class_id » « x_center » « y_center » « width » « height » "

**avec :**

class_id : numéro de la classe.

x_center, y_center : les coordonnées du centre de la boîte englobante.

width : la largeur de la boîte englobante.

height : la hauteur de la boîte englobante.

Les coordonnées et la taille de la boîte englobante sont normalisés par rapport à la taille de l'image. Le fichier d'annotation est conçu pour être entièrement compatible avec les exigences de YOLO, ce qui simplifie le processus d'intégration des données dans les systèmes d'entraînement du modèle.

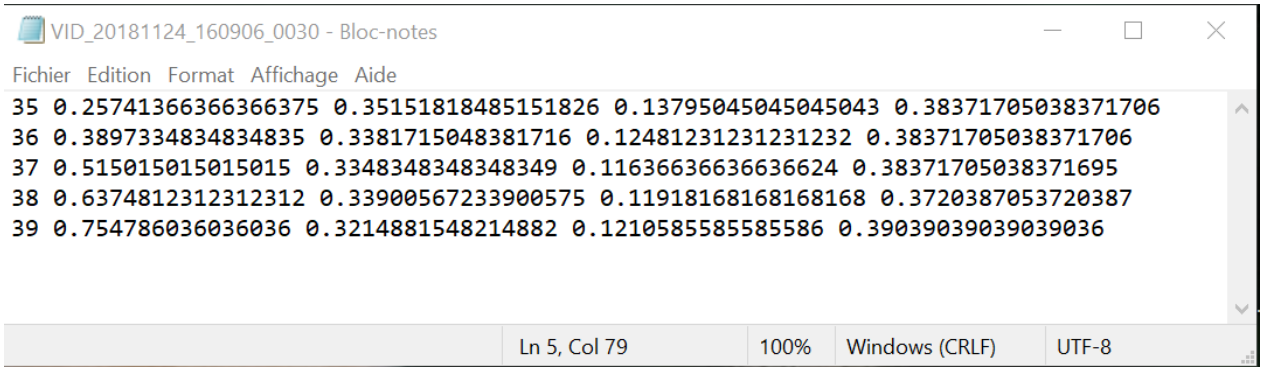



FIGURE 4.3 – Exemple d'un fichier texte sous format YOLO

## 4.4 Bases utilisées

Pour l'évaluation de YOLOv4, nous avons utilisé quatre bases de données distinctes, chacune ayant été soigneusement choisies pour représenter des classes variées et permettre un apprentissage robuste du modèle. Ces bases contiennent des images annotées pour la détection d'objets, couvrant différents contextes et scénarios d'utilisation. Les bases varient en taille et en complexité, avec certaines axées sur des objets spécifiques et d'autres offrant une diversité plus large d'éléments. Les bases exploitées sont : Document, Carrot, Vehicule et HOLOI (*Home & Office Library Object Images*).

### 4.4.1 Base Document

Cette base de donnée à une seule classe d'objets, à savoir "Document", comprend un total de 300 images couleurs [51]. Ces images sont de dimensions 4128 x 2322 pixels ou 2322 x 4128 pixels, représentant différents types de documents. Chaque image de cette base peut contenir un ou plusieurs documents, présentés dans divers environnements et sous différentes perspectives comme montré sur la figure 4.4, ce qui permet d'entraîner le modèle à détecter les documents dans des conditions variées. Les images ont été préalablement annotées, chaque image étant accompagnée d'un fichier texte au format YOLO représentant sa vérité terrain, ce qui a permis d'utiliser la base de données directement.

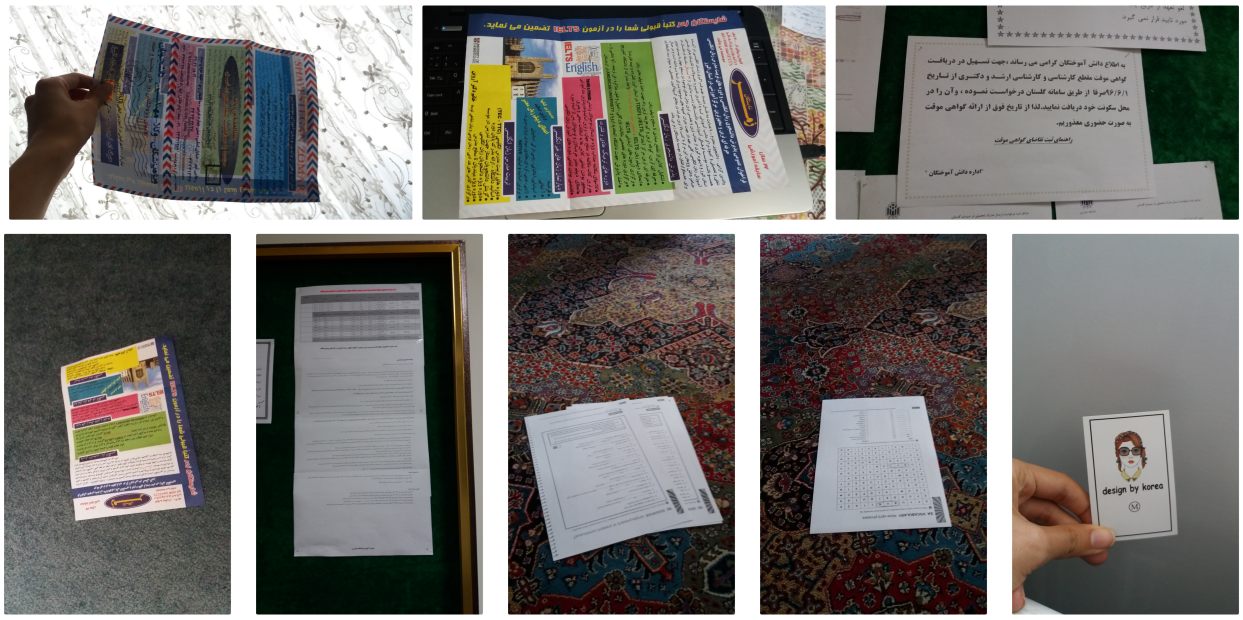



FIGURE 4.4 – Quelques images de la base Document

### 4.4.2 Base Carrot

Notre base de données, intitulée Carrot, est dérivée d'une base plus vaste nommée Vegetables [52], qui regroupe sept classes de légumes. Toutefois, en raison de l'absence d'informations sur les annotations de cette dernière, nous avons choisi de nous concentrer uniquement sur la classe des carottes. Pour ce faire, nous avons pris l'initiative d'annoter nous-mêmes les images de cette classe au format YOLO, en utilisant LabelMe.

Cette base Carrot comprend un total de 299 images couleurs de différentes dimensions que nous avons redimensionnées par la suite à une taille uniforme de 576 x 768 pixels. Chaque image peut contenir une ou plusieurs carottes, présentées dans divers environnements et sous différentes perspectives. Les variations incluent des bruits gaussiens, des rotations, des

chevauchements de carottes, des changements de luminosité et de teinte, ainsi que différents angles de vue (voir figure 4.5). Ces divers éléments permettent d'entraîner le modèle à détecter les carottes dans des conditions variées

FIGURE 4.5 – Quelques images de la base Carrot

### 4.4.3 Base Vehicule

Cette base de données est conçue pour la détection de véhicules et comprend 5 classes de véhicules : Ambulance, Bus, Car, Motorcycle et Truck [53]. Elle contient un total de 502 images couleurs de dimensions uniformes de 416 x 416 pixels et qui ont été préalablement annotées sous format YOLO, permettant ainsi une utilisation directe de cette base. Chacune de ces images peut contenir un ou plusieurs véhicules, représentés dans divers environnements et sous différentes perspectives (voir la figure 4.6).


FIGURE 4.6 – Quelques images de la base Vehicule

### 4.4.4 Base HOLOI

La base HOLOI, créée à l'Université de Haute Alsace de Mulhouse, regroupe des images couleurs qui représentent en tout 500 objets. Pour notre travail, nous avons sélectionné uniquement les images incluant 50 de ces objets illustrés dans la figure (voir figure 4.8). Cette sélection contient 917 images de dimensions 960 x 540 pixels, avec environ 40 images par objet. Ces images, ont été prises avec un appareil photo numérique Nikon et divers téléphones portables, dans différentes conditions : changement d'arrière plan, d'éclairage et d'angle de prise des images comme montré sur la figure 4.7. Cela permet au modèle de s'adapter à une large gamme de scénarios. Les images étaient accompagnées de vérités terrain dont le format était inconnu. Nous avons donc effectué nous-mêmes l'annotation avec LabelMe, comme pour la base Carrot.


FIGURE 4.7 – Quelques images de la base Holoi


FIGURE 4.8 – 50 objets utilisés de la base HOLOI

Le tableau 4.1 résume les caractéristiques phares des 4 bases de données utilisées dans notre étude.

| Base | Document | Carrot | Vehicule | Holoi |
|---|---|---|---|---|
| Nombre de classes | 1 | 1 | 5 | 50 |
| Taille de l'image (pixels) | 4128 x 2322, 2322 x 4128 | 576 x 768 | 416 x 416 | 960 x 540 |
| Variation d'angle de prise | ✓ | ✓ | ✓ | ✓ |
| Variation de la taille | ✓ | ✓ | ✓ | × |
| Changement d'arrière plan | ✓ | ✓ | ✓ | ✓ |
| Changement de luminosité | ✓ | ✓ | ✓ | ✓ |
| Flou | ✓ | ✓ | × | ✓ |
| Chevauchement | ✓ | ✓ | ✓ | ✓ |

TABLE 4.1 – Récapitulatif des caractéristiques des bases utilisées.

## 4.5 Tests et résultats

Après avoir sélectionné les bases de données et l'environnement requis, il est essentiel d'adapter les images utilisées pour l'apprentissage et le test de YOLOv4, ainsi que d'ajuster certains paramètres de ce modèle en fonction de ces bases. Ensuite, nous examinerons les tests effectués et les résultats obtenus pour évaluer la performance et la précision du modèle.

### 4.5.1 Préparation des bases

Pour évaluer la performance de YOLOv4 sur nos 4 bases de données, il est essentiel de passer par plusieurs étapes de configuration avant de commencer l'apprentissage du modèle.

Tout d'abord, les bases de données doivent être divisées en deux ensembles distincts : l'ensemble d'apprentissage, destiné à entraîner le modèle, et l'ensemble de test, utilisé pour évaluer sa performance. Certaines de nos bases, à savoir Vehicule et HOLOI, sont déjà préalablement divisées en ensembles d'apprentissage et de test, que nous pouvons utiliser directement. En revanche, pour les bases Document et Carrot, il est nécessaire de réaliser cette division nous-mêmes. La répartition de ces bases est illustrée dans le tableau 4.2.

| | Nombre d'images | | |
|---|---|---|---|
| Base | Total | Apprentissage | Test |
| Document | 300 | 240 (80%) | 60 (20%) |
| Carrot | 299 | 239 (80%) | 60 (20%) |
| Vehicule | 502 | 439 (87%) | 63 (13%) |
| HOLOI | 917 | 635 (70%) | 282 (30%) |

TABLE 4.2 – Répartition des différentes bases utilisées

Ensuite, il est nécessaire de préparer les fichiers texte **data** et **names**.

Le fichier **data** spécifie le nombre total de classes, les chemins vers les fichiers train, valid et names et le répertoire backup où les poids du modèle seront sauvegardés.

Le fichier **names** contient la liste des noms des classes d'objets à détecter. Chaque ligne du fichier correspond à une classe d'objet, numérotée à partir de zéro.

### 4.5.2 Critères d'évaluation

Afin d'évaluer les performances de notre modèle, nous allons nous appuyer sur des métriques clés telles que la précision (*percision*), le rappel (*recall*), la précision moyenne *(Average Precision -AP)* et la moyenne des précisions moyennes (*Mean Average Precision -mAP).* Le calcul de ces métriques se basent sur les mesures suivantes :

Vrai positif (TP) : le nombre de cas où le modèle a correctement prédit un résultat positif (le résultat réel était positif).

Vrai négatif (TN) : le nombre de cas où le modèle a correctement prédit un résultat négatif (le résultat réel était négatif).

Faux positif (FP) : le nombre de cas où le modèle a prédit de manière erronée un résultat positif (le résultat réel était négatif).

Faux négatif (FN) : le nombre de cas où le modèle a prédit de manière erronée un résultat négatif (le résultat réel était positif).

* **Précision :** mesure la proportion de prédictions correctes parmi toutes les prédictions positives. Cette métrique est donnée par la formule suivante :
$$Précision = \frac{TP}{TP + FP} \tag{4.1}$$
* **Rappel** : mesure la proportion de prédictions correctes parmi toutes les prédictions de la vérité terrain. Cette métrique est donnée par la formule suivante :
$$Rappel = \frac{TP}{TP + FN} \tag{4.2}$$
* **Précision moyenne :** est une mesure de la précision d'un modèle pour une classe spécifique dans un problème de classification. Elle est calculée en prenant la moyenne des précisions à chaque niveau de rappel, de 0 à 1, où un nouvel élément de la classe est trouvé dans la liste des prédictions triées. L'AP peut être déterminée pour chaque classe individuellement dans un problème de classification multiclasse.
* **Moyenne de la précision moyenne :** est une mesure plus globale, principalement utilisée dans les tâches de détection d'objets. La mAP est obtenue en faisant la moyenne des AP calculées pour chaque classe d'objet présente dans la base de données, selon la formule 4.3 ci-dessous. Elle évalue la capacité d'un modèle de détection d'objets à localiser et classifier correctement plusieurs classes d'objets dans une image.

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{4.3}$$

Où N est le nombre total de classes.

### 4.5.3 Paramètres de YOLOv4

Pour détecter des objets dans les différentes bases sélectionnées, nous commencerons par entraîner YOLOv4 sur celles-ci. Pour ce faire, nous utiliserons l'apprentissage par transfert en chargeant les poids pré-entraînés du modèle YOLOv4 sur la base MSCOCO [54], ce qui permettra d'accélérer l'entraînement de ce modéle. Cependant, avant de commencer l'apprentissage, plusieurs hyperparamètres doivent être soigneusement ajustés dans le fichier **cfg** pour améliorer les performances et assurer la convergence du modèle. Voici les principaux à considérer :

* **Taille du lot (*Batch size*) :** est un hyperparamètre qui détermine combien d'exemples d'apprentissage sont traités ensemble avant de mettre à jour les poids du modèle. Par exemple, *batch*=64 signifie que 64 images sont traitées en parallèle avant chaque mise à jour. Un petit *batch size* offre des mises à jour plus fréquentes mais peut rendre l'apprentissage instable, tandis qu'un grand *batch size* fournit des gradients plus stables mais consomme plus de mémoire et peut ralentir la convergence. Le choix du *batch size* nécessite souvent des tests empiriques pour trouver un bon équilibre. Dans la plupart des cas, une taille de batch de 32 ou 64 est recommandée comme point de départ. Cependant, il est important de noter que la taille du batch peut varier en fonction de la nature de l'ensemble de données et des ressources disponibles [55].
* ***Subdivisions* :** est un hyperparamètre en *machine learning* qui détermine le nombre de sous-ensembles dans lesquels un lot (*batch*) est divisé pendant l'apprentissage. Par exemple, avec subdivisions=8, un lot de 64 images sera divisé en 8 sous-ensembles de 8 images chacun. Cette division permet de traiter des parties plus petites du lot à la fois, réduisant ainsi les exigences en mémoire GPU. Il est souvent nécessaire de faire des ajustements expérimentaux pour trouver le compromis optimal pour la configuration spécifique du modèle.
* ***Max Batches* :** est un hyperparamètre qui fixe le nombre total d'itérations d'apprentissage d'un modèle. Par exemple, max_batches=500500 signifie que l'apprentissage s'arrêtera après 500500 itérations. Ce paramètre détermine la durée totale de l'apprentissage ; un nombre élevé permet un apprentissage plus approfondi mais augmente le temps et les ressources nécessaires, tandis qu'un nombre plus bas peut réduire ces besoins au risque d'une qualité de modèle moindre. Il est déterminé en fonction du

nombre de classes dans le jeu de données, comme suit : **max_batches= nombre de classes x 2000**. Cette formule permet de s'assurer que le modèle s'entraîne suffisamment longtemps pour apprendre les caractéristiques des différentes classes.

* ***Steps*** **:** est un hyperparamètre qui définit les moments où le taux d'apprentissage est modifié pendant l'apprentissage du modèle. Par exemple, steps=400000,450000 indique que le taux d'apprentissage sera ajusté lors des itérations 400000 et 450000. Ces modifications permettent au modèle de mieux s'adapter et d'améliorer ses performances à mesure qu'il converge. Steps est définis comme deux valeurs correspondant à 80% et 90% de max_batches. Une bonne configuration des steps optimise l'apprentissage, tandis qu'une mauvaise configuration peut le ralentir ou limiter les performances du modèle.
* ***Filters*** **:** est un hyperparamètre qui détermine le nombre de filtres utilisés dans une couche de convolution. Par exemple, filters=255 signifie que la couche de convolution appliquera 255 filtres différents pour extraire les caractéristiques des images d'entrée. La valeur de *filters* dépend du nombre de classes du jeu de données, et est donnée par la formule suivante : **filters=(num_classes+5)×3**. Un nombre élevé de filtres permet de capturer plus de détails, mais augmente la complexité et les besoins en calcul, tandis qu'un nombre trop faible peut limiter la capacité du modèle à apprendre des représentations riches.
* ***Anchors*** **:** représentent des boîtes d'ancrage prédéfinies utilisées pour la détection des objets. Ces ancres définissent des tailles et proportions spécifiques pour les boîtes de détection, permettant au modèle de mieux s'adapter à la diversité des objets dans les images. Par exemple, les valeurs *anchors* = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401 spécifient 9 paires de dimensions (largeur, hauteur) pour ces boîtes. Les ancres aident à initialiser et ajuster les prédictions des boîtes englobantes pendant l'apprentissage, améliorant ainsi la précision des détections. En effet, les boîtes d'encrage sont recalculées pour chaque base d'images en utilisant l'algorithme de classification *k-means*.
* ***Classes*** **:** Désigne le nombre total de catégories d'objets que le modèle peut détecter. Ce paramètre est crucial pour adapter le réseau aux spécificités des données d'apprentissage. Par exemple, si votre ensemble de données comporte trois classes (chat, chien, oiseau), vous définirez classes = 3. Ce paramètre influence directement la taille de la sortie du modèle, car le nombre de classes détermine le nombre de prédictions faites par le réseau. Une configuration incorrecte de ce paramètre peut entraîner des erreurs dans les prédictions finales, rendant impossible la classification correcte des objets.

Voici quelques paramètres supplémentaires qui sont cruciaux pour saisir le processus d'ap-

prentissage :

* **Un algorithme d'apprentissage (*optimizers*) :** est une méthode utilisée pour ajuster les paramètres d'un modèle d'apprentissage automatique (*machine learning*) afin de minimiser l'erreur et améliorer la performance du modèle. Ces algorithmes, comme la descente de gradient, les variantes comme Adam ou RMSprop, jouent un rôle crucial en guidant le processus d'apprentissage en ajustant les poids du modèle en réponse aux erreurs. Ils ne sont pas toujours explicitement listés dans les paramètres d'apprentissage de modèles comme YOLO, mais sont essentiels pour optimiser les performances. Les paramètres d'apprentissage de YOLO, tels que la taille du lot (*batch size*) et les fonctions de perte (*loss*), influencent également l'efficacité de l'apprentissage mais ne définissent pas directement l'algorithme d'apprentissage utilisé [56].
* **Époque (*Epoch*) :** Dans le *machine learning*, une époque désigne un cycle complet au cours duquel l'ensemble des données d'apprentissage traverse l'algorithme. Ce paramètre clé influence l'ajustement des poids du modèle et sa précision, nécessitant parfois des milliers de répétitions pour réduire le taux d'erreurs. Le nombre idéal d'époques varie en fonction des données et des performances du modèle, et est souvent déterminé en testant différentes valeurs pour trouver celle qui convient le mieux à la résolution du problème spécifique [57].
* **La fonction de perte (*loss*) :** est une mesure utilisée pour évaluer l'écart entre les prédictions d'un modèle et les valeurs réelles. Pendant l'apprentissage, l'objectif est de minimiser cette fonction pour améliorer la précision du modèle [58].

### 4.5.4 Présentation et analyse des résultats

Dans cette section, nous présenterons les fichiers de configuration **data** et **names** utilisés pour chaque base de données ainsi que les ajustements effectués dans les fichiers **cfg** pour adapter le modèle. Nous détaillerons ensuite les résultats obtenus pour chaque base de données, suivis d'une analyse et d'une interprétation spécifiques à chaque cas.

#### 4.5.4.1 Base Document

Les fichiers **data** et **names** relatifs à la base d'images document sont illustrés dans la figure 4.9. Les hyperparamètres modifiés dans le fichier cfg sont précisés comme suit :

partie [net] : max_batches=2000, steps= 1600,1800

partie [convolutional] avant [yolo] : filters= 18

partie [yolo] : classes= 1 , anchors= 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142,

110, 192, 243, 459, 401

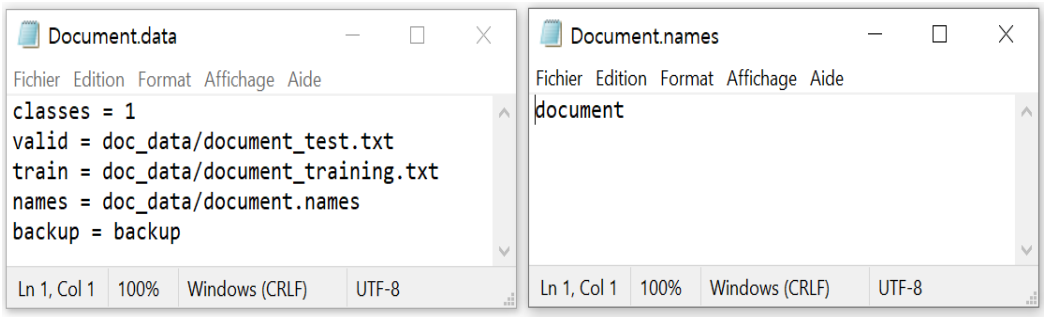



FIGURE 4.9 – Fichiers data et names de la base Document

L'apprentissage du modèle YOLOv4 sur la base Document (ensemble d'apprentissage) a duré environ 12 heures. Les résultats de la détection sur l'ensemble des images test ont été évalués à l'aide de différentes métriques, comme l'indique la figure 4.10, où nous soulignons :

Précision (Precision) : 79%

Rappel (Recall) : 61%

mAP : 67.80 %

Temps de détection total : 75 secondes

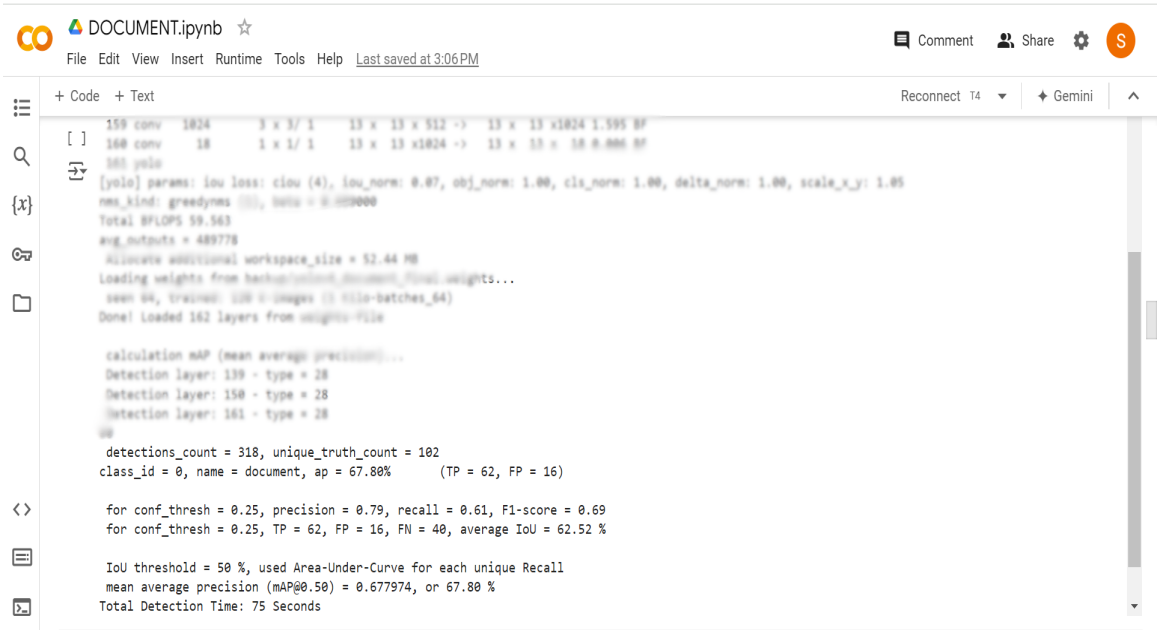



FIGURE 4.10 – Résultats du test de la base Document

Les résultats de cette base révèlent une mAP de 67,80 %, indiquant une performance modérée avec des marges d'erreur significatives, avec une précision de 79 % et un rappel de 61 %.

Étant donné que cette base ne contient qu'une seule classe de documents mais avec des catégories très variées (forte variation intra-classe), les résultats insatisfaisants illustrés dans la figure 4.11 peuvent être dûs soit à une sous-représentation de certaines catégories, soit au changement d'arrière plans par rapport à ceux des images d'apprentissage. Ce manque de diversité dans les données d'apprentissage peut nuire à la capacité du modèle à reconnaître certains types de documents. En revanche, les bons résultats présentés dans la figure 4.12 montrent que, dans les cas où le modèle a réussi à détecter correctement les documents, les prédictions étaient accompagnées d'un score de confiance élevé. Cela est probablement dû à la disponibilité d'un nombre suffisant d'images d'apprentissage pour certaines catégories. Cette capacité à s'adapter à différents environnements renforce l'efficacité du modèle dans la détection des documents, mais il reste essentiel d'améliorer le rappel pour maximiser l'identification des objets pertinents.

Le temps moyen de prédiction est de 37.86 millisecondes.

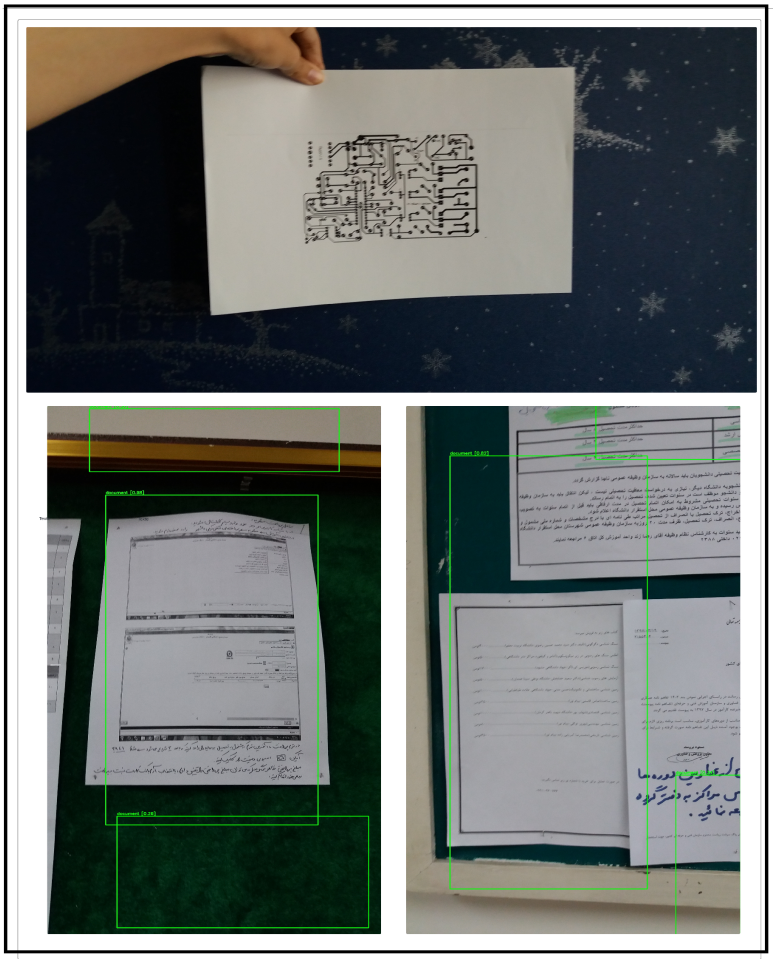


FIGURE 4.11 – Images résultantes du test de la base Document : Mauvaises prédictions

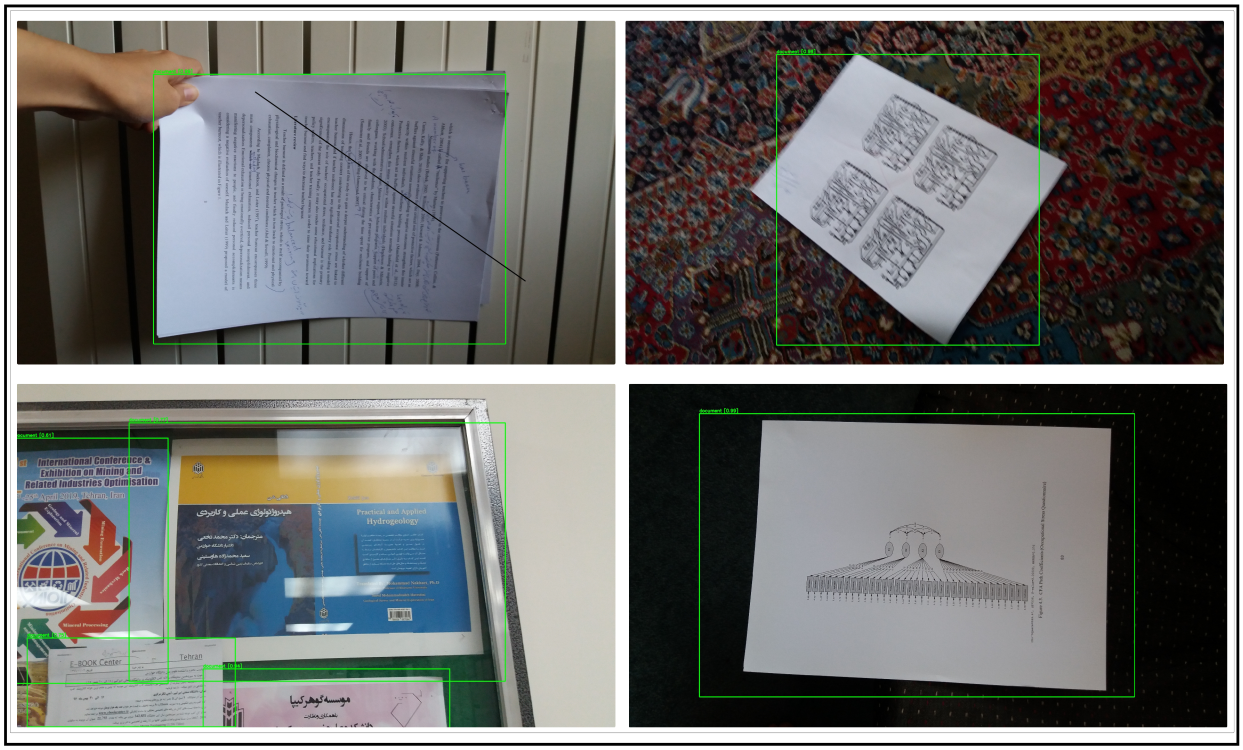



FIGURE 4.12 – Images résultantes du test de la base Document : Bonnes prédictions

#### 4.5.4.2 Base Carrot

Les fichiers **data** et **names** correspondants à cette base sont donnés par la figure 4.13. Les hyperparamètres modifiés dans le fichier **cfg** sont indiqués comme suit :

partie [net] : max_batches=2000, steps= 1600,1800

partie [convolutional] avant [yolo] : filters= 18

partie [yolo] : classes= 1 , anchors= 47, 33, 39, 95, 112, 69, 56,139, 180, 46, 91,133, 162, 94, 263, 72, 144,204

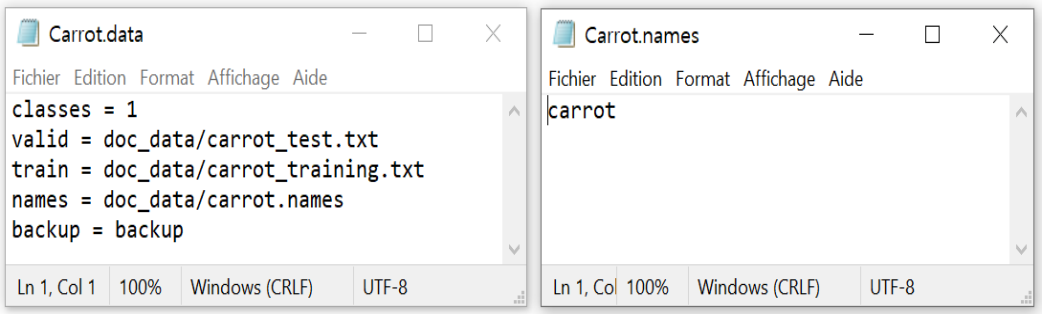



FIGURE 4.13 – Fichiers data et names de la base Carrot

L'apprentissage du modèle YOLOv4 sur la base Carrot a duré environ 4 heures. Les

résultats de la détection sur l'ensemble des images test ont été évalués à l'aide de différentes métriques, comme l'indique la figure 4.14, où nous soulignons :

Précision (Precision) : 96%

Rappel (Recall) : 95%

mAP : 95.87 %

Temps de détection total : 5 secondes

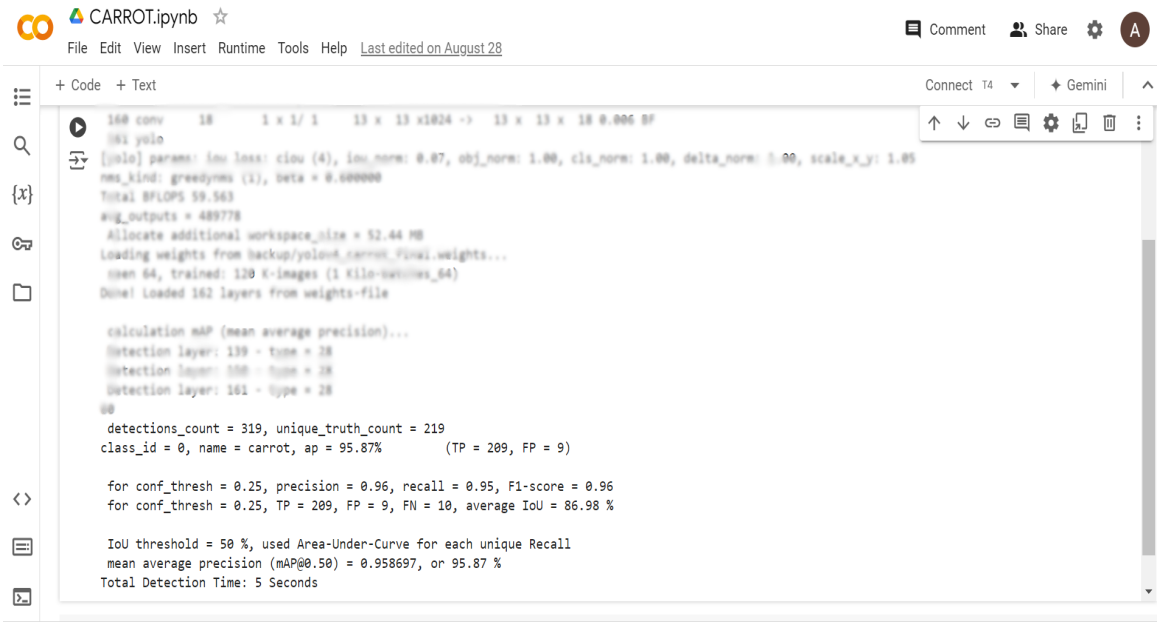



FIGURE 4.14 – Résultats du test de la base Carrot

Les résultats du modèle montrent une bonne performance, avec une mAP de 95.87% et des prédictions satisfaisantes sur l'ensemble des images test, comme illustré sur la figure ci-dessous. Cette performance élevée indique que le modèle détecte la majorité des objets avec une grande précision de 96% et un bon rappel de 95%, avec un nombre insignifiant des FP par rapport au nombre des images de test (60). Bien que des facteurs tels que le bruit gaussien, la rotation et l'effet miroir, présents dans l'ensemble Carrot puissent introduire des défis, YOLOv4 a su les surmonter efficacement. L'homogénéité de la classe Carrot a amélioré considérablement les résultats de la détection. De plus, le temps de prédiction moyen par image après le test est de 38,61 millisecondes, confirmant ainsi l'efficacité du modèle pour des applications en temps réel.

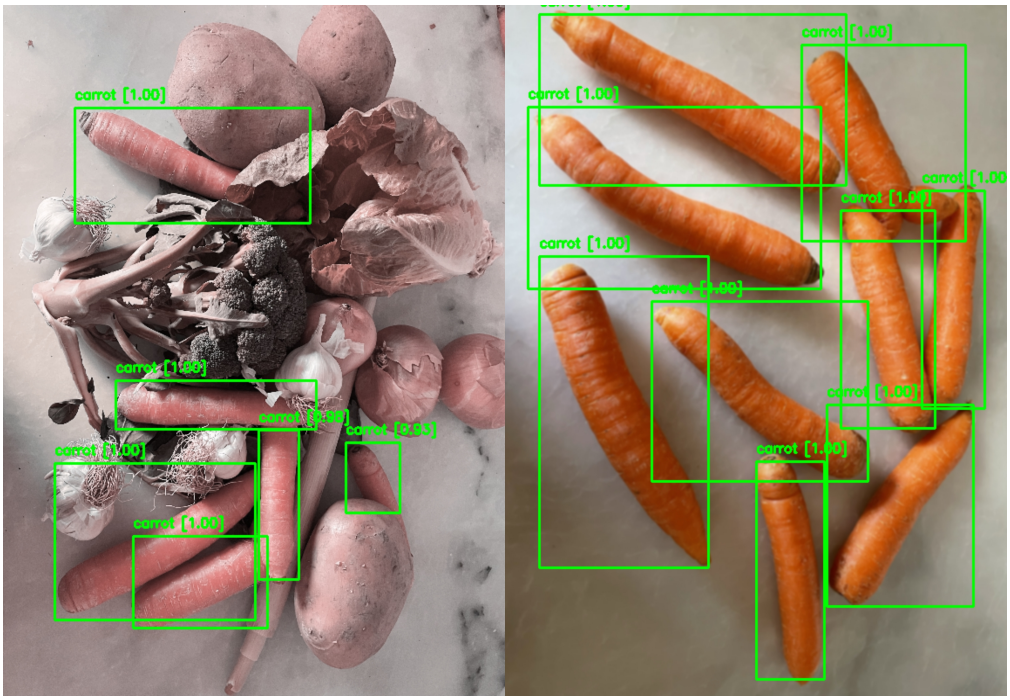



FIGURE 4.15 – Images résultantes du test de la base Carrot

#### 4.5.4.3 Base Vehicule

Les fichiers **data** et **names** correspondants à cette base sont donnés par la figure 4.16. Les hyperparamètres modifiés dans le fichier **cfg** sont indiqués comme suit :

partie [net] : max_batches=10000, steps= 8000,9000

partie [convolutional] avant [yolo] : filters= 30

partie [yolo] : classes= 5 , anchors= 23, 22, 51, 51, 45,120, 94, 75, 99,192, 172,111, 200,234, 330,222, 352,351

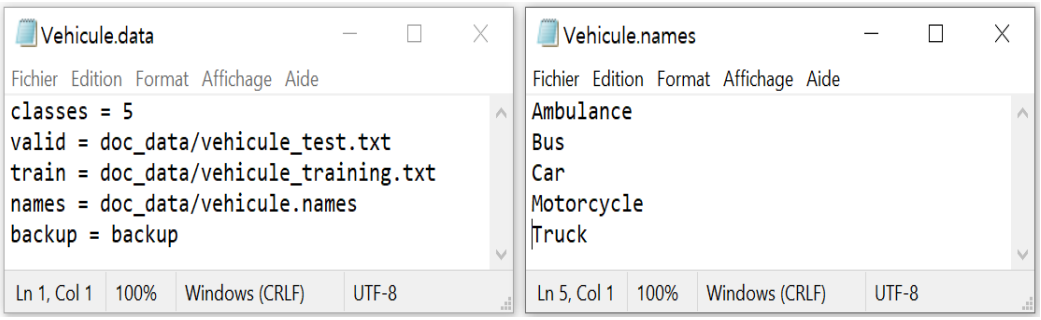



FIGURE 4.16 – Fichiers data et names de la base Vehicule

L'apprentissage du modèle YOLOv4 sur la base Vehicule a duré environ 12 heures. Les résultats de la détection sur l'ensemble des images test ont été évalués à l'aide de différentes

métriques, comme l'indique la figure 4.17, où nous soulignons :

Précision (Precision) : 81%

Rappel (Recall) : 60%

mAP : 71.64 %

Temps de détection total : 2 secondes

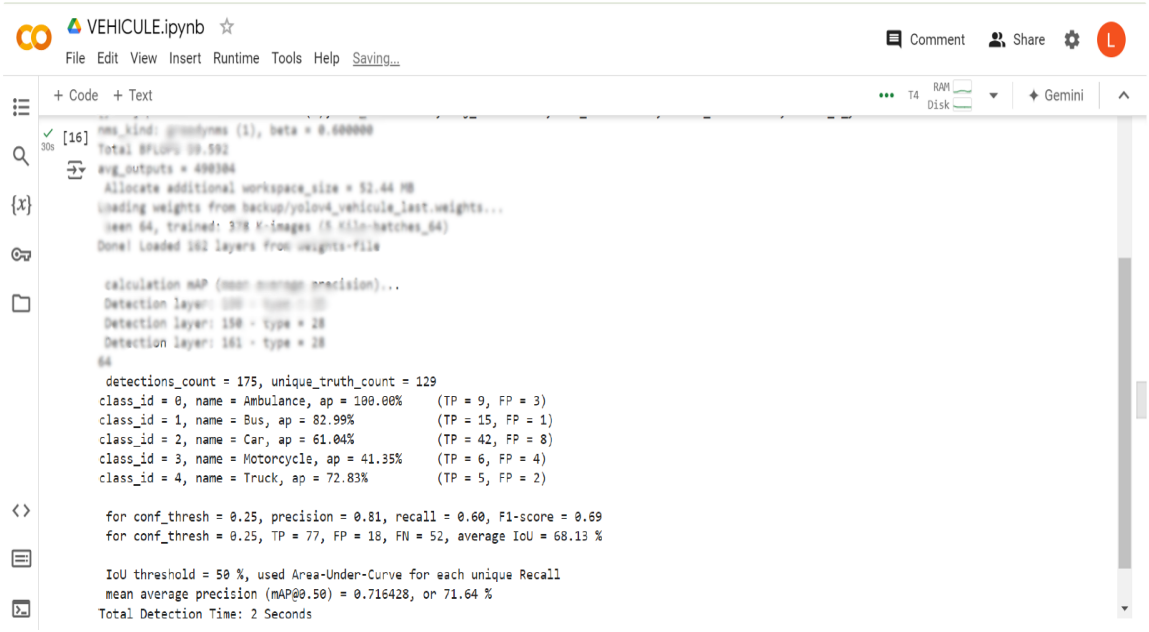



FIGURE 4.17 – Résultats du test de la base Vehicule

Un mAP de 71.64% et une précision de 81% témoignent d'une performance globalement satisfaisante. Cependant, un rappel de 60% révèle que le modèle ne parvient pas à détecter une part significative des objets d'intérêts, ce qui se traduit par 52 faux négatifs.

Les classes : Car, Motorcycle, Bus et Truck étaient déjà présentes dans la base COCO, sur laquelle le modèle a été pré-entraîné. Cependant, leur performance s'est révélée inférieure aux attentes, notamment en raison de la variabilité significative entre notre jeu de données et COCO, tant en termes d'apparence que de contexte.

Classe Motorcycle (AP de 41.35%) : Cette classe affiche la performance la plus faible, en grande partie à cause de son hétérogénéité, rendant l'extraction de caractéristiques spécifiques à cette classe difficile.

Classe Car (AP de 61.04%) : Le modèle détecte un nombre significatif de voitures, mais fait face à des défis liés à l'occlusion, avec de nombreuses voitures partiellement masquées par d'autres objets. La diversité des modèles, couleurs et positions des voitures complique également leur identification.

Classe Truck (AP de 72.83%) : Le modèle montre une détection efficace des camions, en

raison de leur forme distinctive et uniforme. De plus, les camions sont généralement moins sujets à l'occlusion que les voitures, ce qui contribue à de meilleures performances.

Classe Bus (AP de 82.99%) : Cette classe affiche de meilleures performances grâce à la forme distinctives des bus, qui sont faciles à identifier. Les bus apparaissent souvent dans des contextes clairs, ce qui facilite leur détection.

Classe Ambulance (AP de 100%) : En tant que classe nouvelle, l'Ambulance atteint un AP de 100%, reflet d'une bonne performance attribuée à la qualité des exemples dans la base de données et à sa distinction visuelle claire (faible variation intra-classe).

Le temps moyen de prédiction par image est de 38,61 ms, assurant une rapidité adaptée aux applications en temps réel.

Les points évoqués dans cette partie sont illustrés par les figures 4.18 et 4.19.

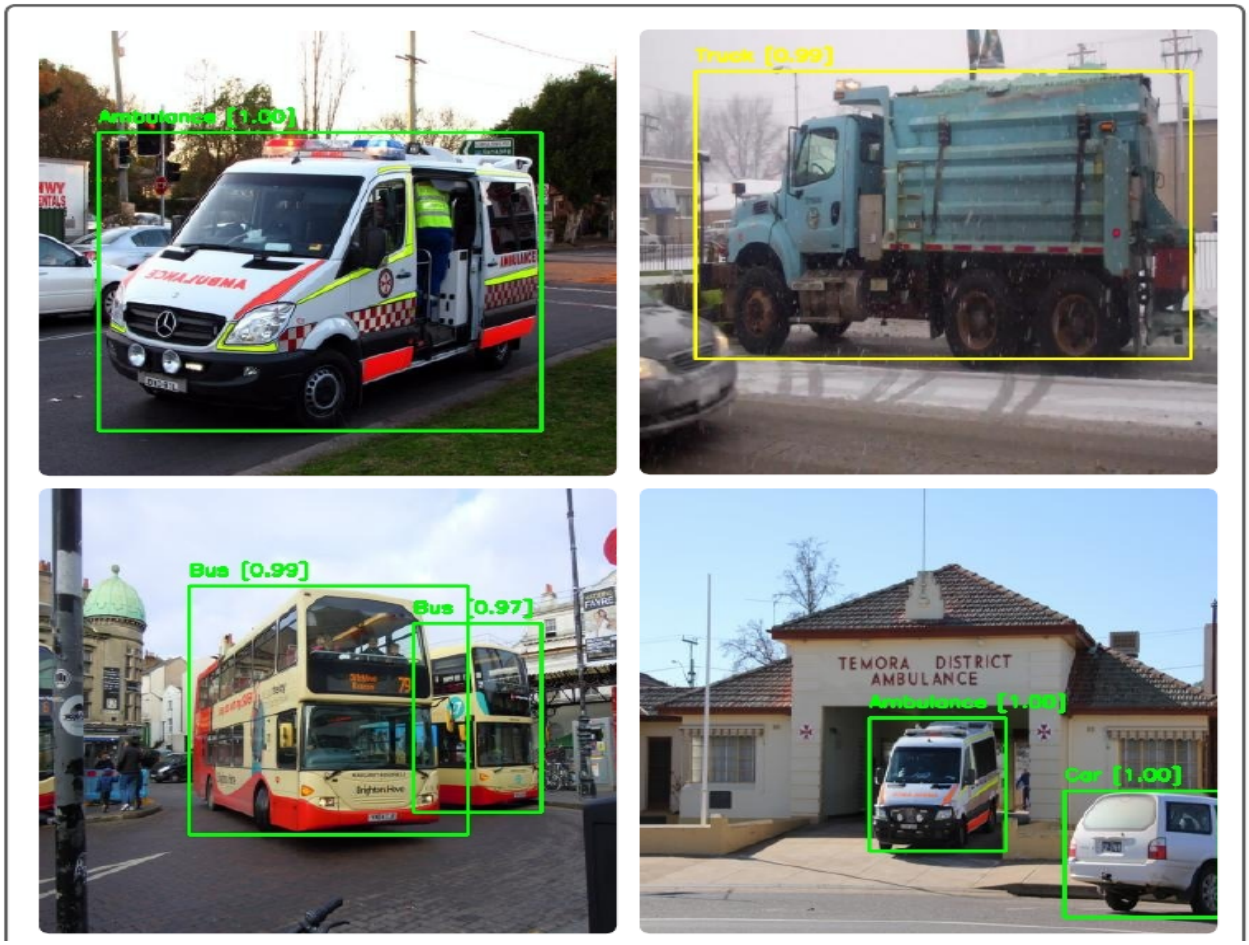



FIGURE 4.18 – Images résultantes du test de la base Vehicule : Bonnes prédictions

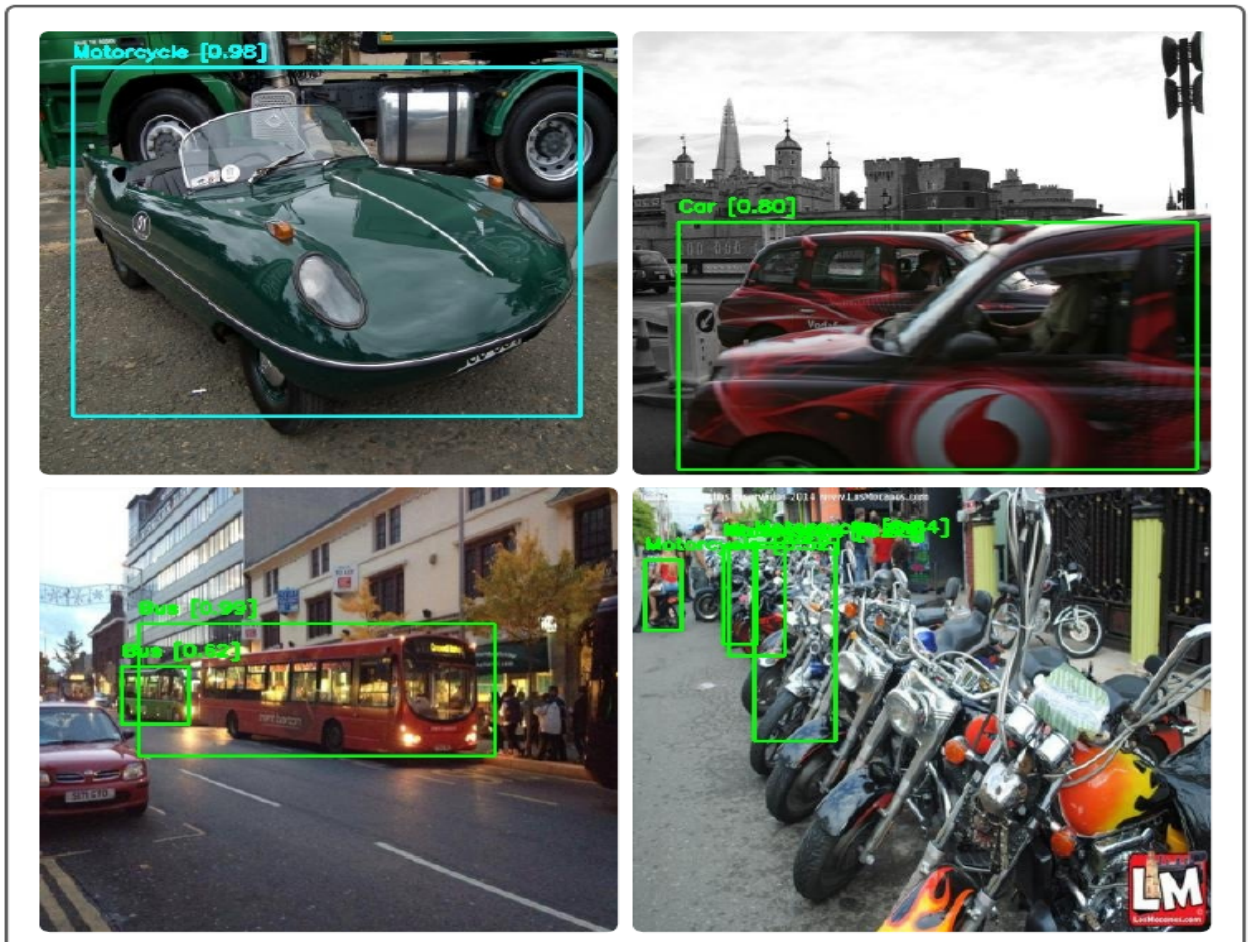



FIGURE 4.19 – Images résultantes du test de la base Vehicule : Mauvaises prédictions

#### 4.5.4.4 Base HOLOI

Les fichiers **data** et **names** correspondants à cette base sont donnés par la figure 4.20. Les hyperparamètres modifiés dans le fichier **cfg** sont indiqués comme suit :

partie [net] : max_batches=30000, steps= 24000,27000

partie [convolutional] avant [yolo] : filters= 165

partie [yolo] : classes= 50 , anchors= 21, 58, 26, 74, 33, 69, 31, 90, 42, 82, 38,105, 54,100, 48,125, 69,134

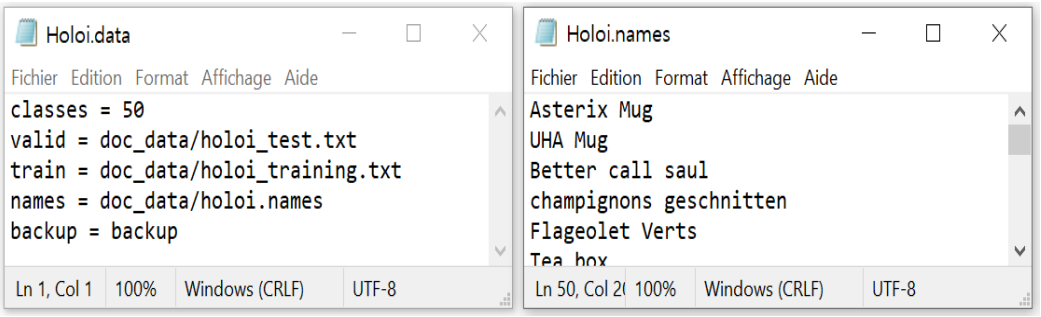



FIGURE 4.20 – Fichiers data et names de la base HOLOI

L'apprentissage du modèle YOLOv4 sur la base Vehicule a duré environ 18 heures. Les résultats de la détection sur l'ensemble des images test ont été évalués à l'aide de différentes métriques, comme l'indique la figure 4.21, où nous soulignons :

Précision (Precision) : 91%

Rappel (Recall) : 94%

mAP : 92.51%

Temps de détection total : 204 secondes

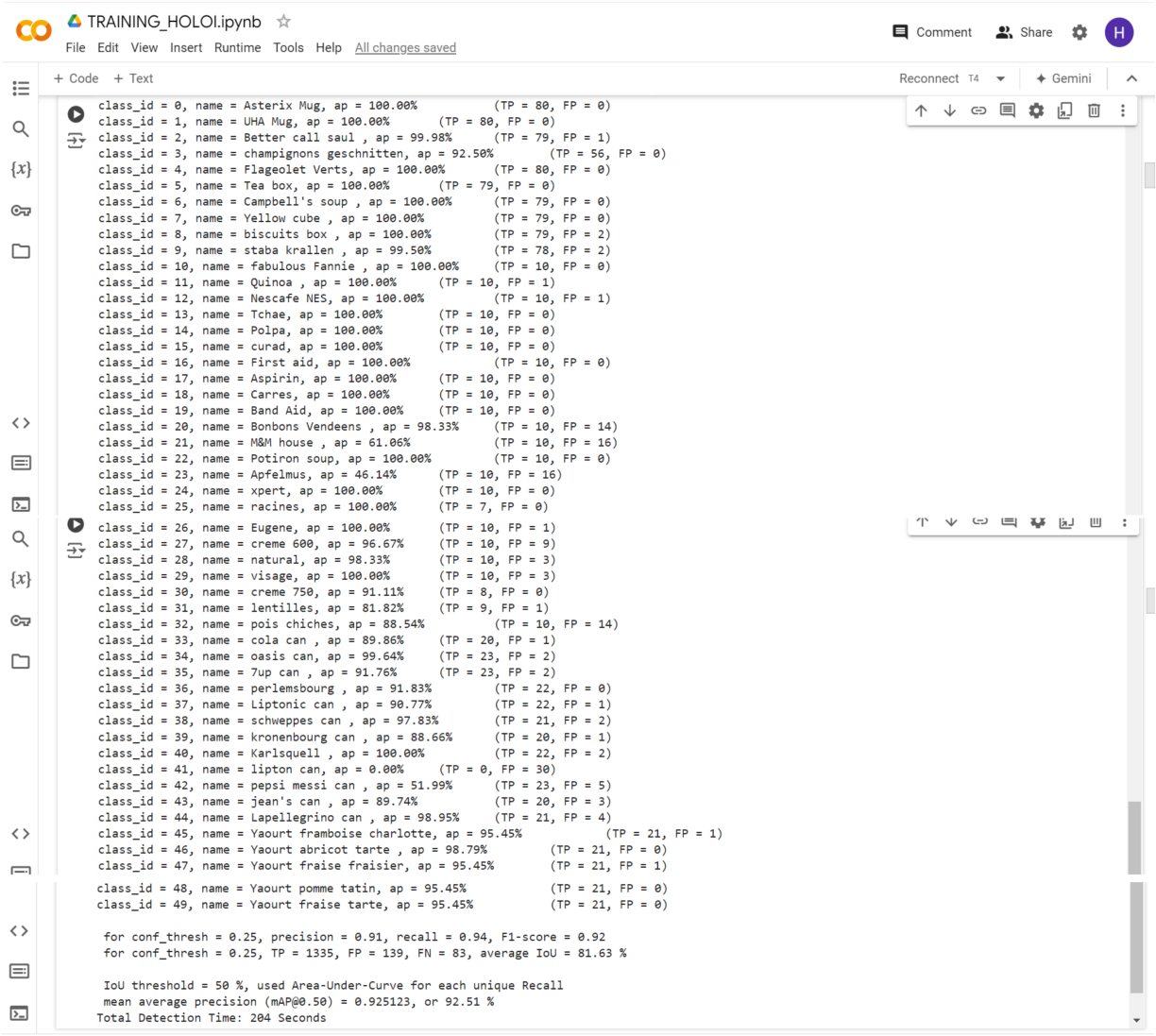



FIGURE 4.21 – Résultats du test de la base HOLOI

Les performances globales du modèle après l'apprentissage sont satisfaisantes, avec une précision de 91 % et un rappel de 94 %, assurant une bonne classification et localisation des objets dans la plupart des cas comme montré sur la figure 4.22. Cette réussite peut être attribuée en grande partie à l'homogénéité des objets au sein de chaque classe, comme observé dans la base "Carrot". YOLOv4 a particulièrement bien réussi à détecter et à distinguer les objets même lorsque certaines classes présentaient une forte ressemblance entre elles. Par exemple, les classes de pots de yaourt, canettes et boîtes de teinture pour cheveux (voir figure 4.23) ont été efficacement différenciées, malgré leurs similarités visuelles.

Cependant, la classe "Lipton Can" pose un problème particulier, affichant un AP de 0 % et générant 30 faux positifs. Cette classe souffre probablement de confusions avec d'autres

canettes aux caractéristiques visuelles proches (voir figure 4.24), malgré un nombre d'images comparable à d'autres classes. Cela pourrait être dû à une qualité et une diversité insuffisantes des données d'apprentissage. Pour remédier à cette faiblesse, il serait crucial d'augmenter la diversité des images d'apprentissage et de s'assurer de la précision des annotations. Malgré ces défis, le modèle atteint un mAP optimal de 92.51% après 10 000 itérations, démontrant sa robustesse dans des conditions variées, avec un temps moyen de prédiction de 38,61 ms, garantissant une rapidité suffisante pour des applications en temps réel.


FIGURE 4.22 – Images résultantes du test de la base HOLOI : Bonnes prédictions

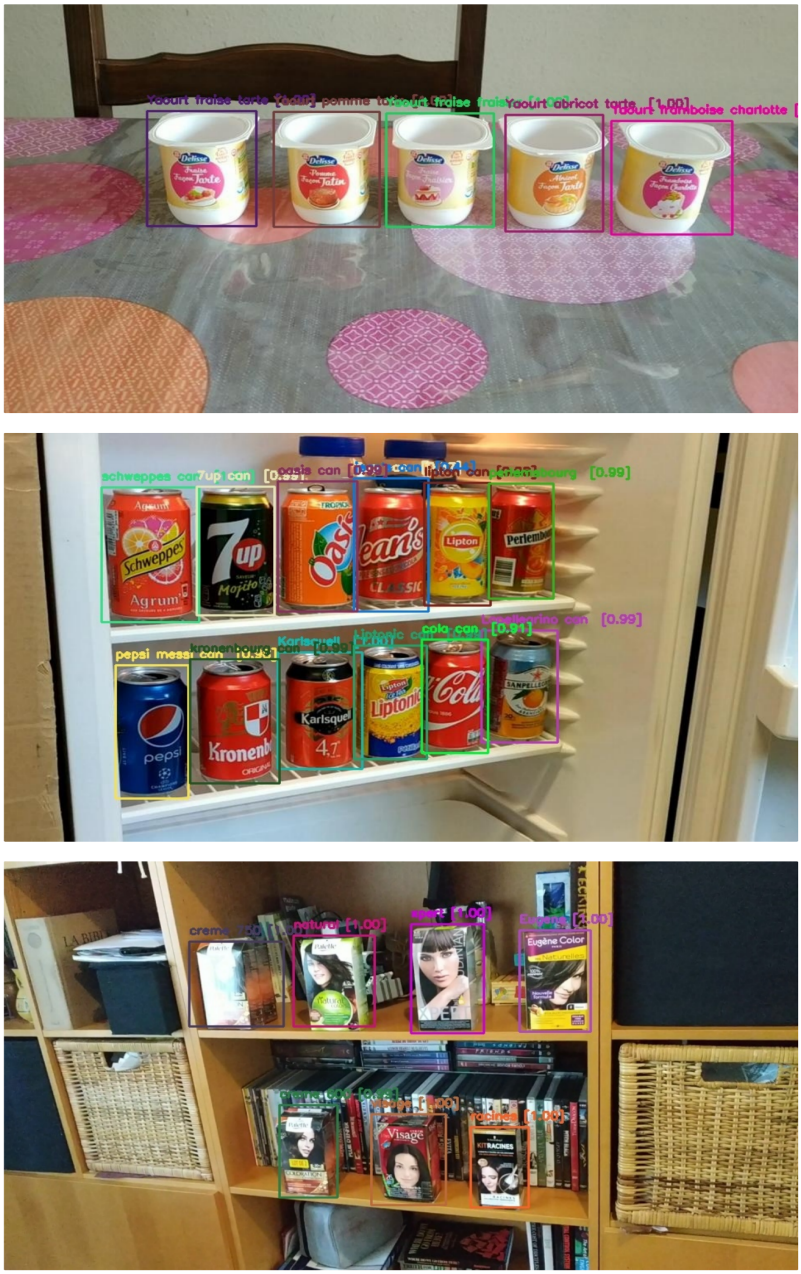



FIGURE 4.23 – Images résultantes du test de la base HOLOI : Objets présentant des similarités


FIGURE 4.24 – Images résultantes du test de la base HOLOI : Mauvaises prédictions

### 4.5.5 Problèmes rencontrés

Lors de l'apprentissage de notre modèle YOLOv4 pour la détection d'objets, nous avons rencontré plusieurs défis que voici :

* **Limites matérielles :** YOLOv4, en raison de ses exigences élevées en puissance de calcul, nécessite un GPU performant. Néanmoins, en raison de limitations matérielles nous avons opté pour Google Colab qui offre un accès gratuit à des GPU, facilitant ainsi l'apprentissage de modèles complexes comme YOLOv4. Cependant, nous avons

rencontré des problèmes liés à cet accès, qui présente les contraintes suivantes : La durée d'utilisation des GPU est limitée et un temps de gel peut survenir lorsque l'accès au GPU est temporairement suspendu après avoir atteint les quotas d'utilisation, ce qui interrompt les processus en cours. Ces restrictions nous ont posé problème, étant donné notre besoin de sessions longues et d'un accès continu.

* **Contraintes liées aux bases de données :** Nous avons rencontré plusieurs problèmes liés aux bases de données, notamment avec les annotations qui n'étaient pas au format YOLO souhaité. Ces annotations étaient souvent fournies dans des formats différents, tels que XML (pour Pascal VOC) ou JSON, nécessitant ainsi une conversion pour les adapter au format YOLOv4. De plus, nos bases d'images étaient déséquilibrées, avec certaines classes sur-représentées et d'autres sous-représentées, ce qui a conduit à une détérioration du mAP lors d'un apprentissage prolongé, indiquant que le modèle s'adaptait trop étroitement aux données d'apprentissage et perdait ainsi sa capacité à se généraliser efficacement sur de nouvelles images. Certaines de nos bases de données présentent une hétérogénéité importante, regroupant des objets très variés au sein d'une même classe. Cela complique l'apprentissage des caractéristiques communes par le modèle et réduit la précision de la détection

### 4.5.6 Solutions potentielles

Pour surmonter les problèmes rencontrés lors de l'implémentation de notre modèle YOLOv4, plusieurs pistes d'amélioration sont envisageables :

* Opter pour Colab Pro, une version payante proposée par Google Colab, qui offre un accès prioritaire aux GPU, une durée d'utilisation prolongée et d'autres avantages pour un usage intensif.
* Perfectionner les bases de données en assurant la précision et la diversité des annotations tout en rééquilibrant les classes et en augmentant le nombre des images dans les classes. De plus l'application des techniques d'augmentation des données peut enrichir les petites bases de données et améliorer la généralisation du modèle d'autant plus que les modèle YOLO nécessitent de grands ensembles d'images pour un apprentissage optimal. Outre les méthodes classiques telles que la rotation, le zoom et le recadrage, il serait bénéfique d'incorporer des techniques avancées comme CutMix, Mosaic et AutoAugment.
* Réajuster les hyperparamètres pour améliorer les performances, en expérimentant avec divers taux d'apprentissage, en affinant le choix des ancres et en ajustant la taille des lots pour obtenir de meilleurs résultats.

Ces ajustements permettront de surmonter les défis actuels et d'obtenir de meilleurs résultats

avec YOLOv4.

## 4.6 Conclusion

Ce chapitre nous permet de conclure que l'algorithme YOLOv4, qui a fait l'objet de notre étude approfondie, a démontré, grâce à sa rapidité et à sa précision, son efficacité pour détecter les objets dans les images. Les résultats de l'implémentation de YOLOv4 que nous avons obtenus dans ce chapitre montrent clairement son potentiel dans le domaine de la détection d'objets en particulier pour des applications en temps réel. En fin de compte, notre étude démontre que YOLOv4 est un algorithme robuste et adaptable qui peut être utilisé dans une grande variété de scénarios telle que la surveillance, la conduite autonome et la médecine, ouvrant ainsi la voie à de nouvelles applications et à de nouvelles perspectives dans le domaine de la vision par ordinateur.

# Conclusion Générale

Le travail présenté dans ce mémoire a pour objectif d'explorer les techniques modernes de l'intelligence artificielle, en particulier les méthodes avancées de détection d'objets telles que YOLO. Dans ce travail, nous avons déployé YOLOv4 sur plusieurs bases de données, et son implémentation a conduit à des résultats prometteurs, alliant à la fois une grande précision dans la détection des objets et une rapidité d'exécution. Ces performances soulignent le potentiel de YOLOv4 dans des applications nécessitant un traitement en temps réel.

Ce mémoire a été divisé en quatre parties, organisés comme suit :

Dans la première partie, nous avons exploré les concepts fondamentaux des RNA, en posant les bases théoriques nécessaires à la compréhension des modèles d'apprentissage profond. Dans la deuxième partie, nous avons analysé les réseaux de neurones convolutifs (CNN), qui représentent une catégorie des RNA. Nous avons démontré leur capacité à traiter les images en soulignant leur aptitude à extraire automatiquement des caractéristiques pertinentes et à offrir une précision élevée dans la classification. Dans la troisième partie, nous avons abordé la détection d'objets en examinant les approches traditionnelles, avant de nous concentrer sur les techniques modernes, en particulier l'algorithme YOLO. Nous avons montré comment YOLO, avec son approche innovante de traitement d'image en une seule étape, améliore considérablement les performances de la détection d'objets par rapport aux méthodes antérieures. Dans la quatrième partie, nous avons expérimenté YOLOv4 sur quatre bases de données pour évaluer ses performances. Nous avons expliqué en détail les étapes d'implémentation et analysé les résultats obtenus, en mettant en lumière les forces et les faiblesses de l'algorithme.

Nous constatons que l'implémentation de YOLOv4 a donné des résultats pertinents, affichant une précision généralement satisfaisante sur plusieurs bases de données ainsi qu'une rapidité optimale. Ces caractéristiques confèrent des avantages significatifs pour les applications pratiques où rapidité et précision sont cruciales. De plus, l'adaptabilité de YOLOv4 à différents ensembles de données témoigne de sa flexibilité et de sa robustesse. En somme, YOLOv4 s'avère être une solution efficace pour les systèmes de détection d'objets, combinant une précision fiable et une rapidité satisfaisante, prête à être exploitée dans des environnements

variés.

Cependant, il existe des limitations avec l'utilisation de YOLOv4 qui pourraient nécessiter des ajustements ou des alternatives dans certaines situations. Pour cela, il est recommandé d'utiliser les nouvelles versions de YOLO pour améliorer la détection d'objets. Bien que YOLOv4 offre une bonne précision et rapidité, les évolutions récentes intègrent des mécanismes avancés qui peuvent mieux gérer des environnements complexes et des objets difficiles à détecter. Ces versions améliorées promettent d'optimiser les performances et d'élargir les possibilités d'application. De plus, combiner YOLO avec d'autres méthodes de détection d'objets, comme Faster R-CNN ou SSD, pourrait être une perspective intéressante pour améliorer les résultats, bien que cela ne soit pas garanti.

# Bibliographie


[1] Réseau de neurones artificiels — Wikipédia. https ://fr.wikipedia.org.

[2] Ann Waugh Allison Grant. *Ross et Wilson ; Anatomie et physiologie normales et pathologiques*. Elsevier Masson S.A.S, 62, rue Camille-Desmoulins, 92442 Issy-les-Moulineaux cedex, 2015.

[3] Le neurone et l'influx nerveux — Alloprof aide aux devoirs. https ://www.alloprof.qc.ca/fr/eleves/bv/sciences/le-neurone-et-l-influx-nerveux-s1286.

[4] Préface du docteur Yves Morin. *Petit Larousse de la médecine*. Larousse, 2001.

[5] Réseau de neurones récurrent — Data Analytics Post. https ://dataanalyticspost.com/Lexique/reseaux-de-neurones-recurrents/.

[6] Sihem Menaoum ; Khimoud Hayat. *Classification des images texturées par le réseau ELM-LRF*. Université Mouloud Mammeri de Tizi-Ouzou, Faculté De Génie électrique Et Informatique, Département d'automatique. Mémoire de fin d'étude de master académique, 2019.

[7] Règle de Hebb — Wikipédia. https ://fr.wikipedia.org/wiki/R

[8] LAKHDARI HAKIMA KHELLOUT SIHAM. *Etude et application du réseau ELM-LRF en classification des images*. Université Mouloud Mammeri de Tizi-Ouzou, Faculté De Génie électrique Et Informatique, Département d'automatique. Mémoire de fin d'étude de master académique, 2017.

[9] Boukemoum Zakarya. *Détection des véhicules par Histogramme Orienté Gradient*. Université de 8 Mai 1945 – Guelma - Faculté des Mathématiques, d'Informatique et des Sciences de la matière, Département d'Informatique, Filière : Informatique, Option : Systèmes Informatiques. Mémoire de fin d'étude de master académique, 2019.

[10] OUDNI Louiza. *Étude théorique et implémentation de l'algorithme SIFT*. École Nationale Polytechnique,Département d'Électronique. Mémoire de fin d'étude de master académique, 2013.

[11] Boubezari Rayana. *Application de l'algorithme SURF pour le Tracking d'objets.* École Nationale Polytechnique,Département d'Électronique. Mémoire de fin d'étude de master académique, 2013.

[12] Sparsh Gupta ; mis à jour par Brennan Whitfield. 7 Most Common Machine Learning Loss Functions — Built In. https ://builtin.com/machine-learning/common-loss-functions, 2023.

[13] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning : concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8 :1–74, 2021.

[14] Convolutional neural network — Engati. https ://www.engati.com/glossary/convolutional-neural-network.

[15] Abdulazeez Alsajri and Vugar Abdullayev. Review of deep learning : Convolutional neural network algorithm. *Babylonian Journal of Machine Learning*, 2023 :19–25, 04 2023.

[16] Rabah Khermimoun ; Magmoun Houssam. *Classification des lésions de la peau à base des réseaux de neurones convolutifs.* Université Mouloud Mammeri de Tizi-Ouzou, Faculté De Génie électrique Et Informatique, Département d'automatique. Mémoire de fin d'étude de master académique, 2019.

[17] Joseph Redmon and Ali Farhadi. YOLO9000 : better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[18] Petru Potrimba. What is EfficientNet ? The Ultimate Guide. — blog.roboflow.com. https ://blog.roboflow.com/what-is-efficientnet/, 2023.

[19] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999.

[20] Classez et segmentez des données visuelles — openclassrooms.com. https ://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles.

[21] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.

[22] Great Learning Editorial Team. Face Detection using Viola Jones Algorithm — mygreatlearning.com. https ://www.mygreatlearning.com/blog/viola-jones-algorithm/.

[23] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. volume 1, pages 886–893, 07 2005.

[24] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf : Speeded up robust features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[25] L L Chen, R P Han, and Q C Meng. Clothing image feature extraction based on surf. *IOP Conference Series : Materials Science and Engineering*, 439(3) :032104, nov 2018.

[26] Frank Cabello, Yuzo Iano, Rangel Arthur, Abel Dueñas, Julio León, and Diogo Caetano. Automatic detection of utility poles using the bag of visual words method for different feature extractors. pages 116–126, 07 2017.

[27] Deepanshu Tyagi. Introduction to SURF (Speeded-Up Robust Features) — deepanshut041. https ://medium.com/@deepanshut041/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e.

[28] Yukai Yao, Yang Liu, Yongqing Yu, Hong Xu, Weiming Lv, Zhao Li, and Xiaoyun Chen. K-svm : An effective svm algorithm based on k-means clustering. *Journal of Computers*, 8, 10 2013.

[29] Jingwen Sun, Weixing Du, and Niancai Shi. A survey of knn algorithm. *Information Engineering and Applied Computing*, 1, 05 2018.

[30] Qu'est-ce que l'algorithme de forêt aléatoire (random forest) ? | IBM — ibm.com. https ://www.ibm.com/fr-fr/topics/random-forest. [Accessed 23-09-2024].

[31] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[32] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn : Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[34] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once : Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[35] Joseph Redmon and Ali Farhadi. Yolov3 : An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[36] https ://deci.ai. The history of yolo object detection models from yolov1 to yolov8.

[37] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4 : Scaling cross stage partial network. *CoRR*, abs/2011.08036, 2020.

[38] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. You only learn one representation : Unified network for multiple tasks. *CoRR*, abs/2105.04206, 2021.

[39] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YOLOX : exceeding YOLO series in 2021. *CoRR*, abs/2107.08430, 2021.

[40] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei. Yolov6 : A single-stage object detection framework for industrial applications, 2022.

[41] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7 : Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022.

[42] Xianzhe Xu, Yiqi Jiang, Weihua Chen, Yilun Huang, Yuan Zhang, and Xiuyu Sun. Damo-yolo : A report on real-time object detection design, 2023.

[43] Xiang Long, Kaipeng Deng, Guanzhong Wang, Yang Zhang, Qingqing Dang, Yuan Gao, Hui Shen, Jianguo Ren, Shumin Han, Errui Ding, and Shilei Wen. PP-YOLO : an effective and efficient implementation of object detector. *CoRR*, abs/2007.12099, 2020.

[44] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. Yolov9 : Learning what you want to learn using programmable gradient information, 2024.

[45] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yolov10 : Real-time end-to-end object detection, 2024.

[46] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4 : Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020.

[47] Tamal Das. Google Colab : Tout ce que vous devez savoir | Geekflare — geekflare.com. https ://geekflare.com/fr/google-colab/.

[48] Denis Eleyehou. Google Colab : la force du cloud pour l'apprentissage automatique — datascientest.com. https ://datascientest.com/google-colab-tout-savoir.

[49] Ultralytics. Google Colab — docs.ultralytics.com. https ://docs.ultralytics.com/integrations/google-colab/.

[50] NVIDIA Tesla T4 Specs — techpowerup.com. https ://www.techpowerup.com/gpu-specs/tesla-t4.c3316.

[51] Benoit Cayla. YOLO (Partie 5) Créer son modèle avec YOLO : Préparation des données - datacorner par Benoit Cayla — datacorner.fr. https ://datacorner.fr/yolo-custom-1/ ?$_g l$ = 1 ∗ 18$p9umj$ ∗$_g$

$a * NzQ0OTc0MjA5LjE3MTU1MDA5NzM. *_g a_RXP3T9KBWC * MTcyNzA2MTIwMS42OC4wLjE3MjcwNjEyMDEuMC4wLjA.$

[52] M. Israk Ahmed and Shahriyar Mahmud Mamun. Vegetable image dataset, 2021.

[53] Vehicles-OpenImages Object Detection Dataset — public.roboflow.com. https ://public.roboflow.com/object-detection/vehicles-openimages.

[54] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco : Common objects in context, 2015.

[55] What is batch size in neural network ? — stats.stackexchange.com. https ://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network.

[56] Keras Team. Keras documentation : Optimizers — keras.io. https ://keras.io/api/optimizers/. [Accessed 23-09-2024].

[57] Raphael Kassel. Epoch : Définition, mode de fonctionnement et utilisation — datascientest.com. https ://datascientest.com/qu-est-ce-qu-un-epoch-en-machine-learning.

[58] Djazira Habeche ; Tounsia Boukaouma. *Tumeur cérébrale IRM Deep learning CNN Segmentation Transfert Learning Data augmentation Cross validation Unet VGG16 ResNet34 Google colab Réseaux de neurones*. Université Mouloud Mammeri de Tizi-Ouzou, Faculté De Génie électrique Et Informatique, Département d'automatique. Mémoire de fin d'étude de master académique, 2022.