Université du Québec
à Rimouski

ACCÉLÉRATION DE L'INFÉRENCE EN PÉRIPHÉRIE : IMPLANTATION MATÉRIELLE SUR CIRCUITS FPGA DU MODÈLE DE DÉTECTION D'OBJETS YOLOV5 À L'AIDE DE L'OUTIL VITIS AI

MÉMOIRE PRÉSENTÉ

dans le cadre du programme de maîtrise en ingénierie

en vue de l'obtention du grade de maître ès sciences appliquées (M.Sc.A.)

PAR

© ACHRAF CHAKROUN

Décembre 2025

Composition du jury :

Yacine Benahmed (Ph.D.), président du jury, Université du Québec à Rimouski

Mohammed Bahoura (Ph.D.), directeur de recherche, Université du Québec à Rimouski

Martin Otis (Ph.D.), examinateur externe, Université du Québec à Chicoutimi

Dépôt initial le 21 Octobre 2025 Dépôt final le 17 Décembre 2025

Ce travail est dédié à mes parents et à tous ceux qui, de près ou de loin, m'ont soutenu durant ces années de mon parcours de maîtrise.

REMERCIEMENTS

Je tiens tout d'abord à exprimer ma profonde gratitude à mon directeur de recherche, Mohammed Bahoura, professeur à l'Université du Québec à Rimouski (UQAR), pour sa supervision durant ce projet de recherche, sa disponibilité et ses conseils. Sa rigueur scientifique, sa bienveillance et son expertise ont grandement contribué à la réussite de ce projet.

Mes remerciements s'adressent aussi au personnel du département de Mathématiques, d'Informatique et de Génie (DMIG) de l'université du Québec à Rimouski (UQAR) pour l'environnement de travail stimulant, et particulièrement à Étienne Rousseau pour les discussions enrichissantes et pour le soutien apporté.

Je souhaite également exprimer ma reconnaissance envers mes collègues du laboratoire et ami(e)s de maîtrise, notamment Malycia Mangani, pour leur soutien moral, leur relecture attentive et les moments de camaraderie qui ont rendu cette aventure plus humaine et agréable.

Un merci tout particulier à ma famille, et en particulier à mes parents et mes deux frères, pour leur amour, leur patience, leur écoute et leur soutien inconditionnel durant toutes les étapes de cette maîtrise.

RÉSUMÉ

L'objectif principal de ce mémoire est de proposer une solution efficace pour l'inférence en temps réel de modèles de détection d'objets à l'aide d'une architecture matérielle reconfigurable. Plus précisément, notre travail porte sur l'implantation matérielle du modèle de détection d'objets YOLOv5 quantifié sur la plateforme FPGA Kria KV260, à l'aide du framework Vitis AI développé par AMD/Xilinx.

Notre projet comprend deux étapes principales : l'entraînement d'un modèle de détection d'objets YOLOv5 basé sur les réseaux de neurones convolutifs (CNN), puis l'optimisation de ce modèle pour son implantation sur des plateformes matérielles comme les circuits FPGA ayant toutefois des ressources restreintes. L'hypothèse principale de cette étude repose sur le fait que l'accélération matérielle d'un modèle par quantification permettrait de maintenir une précision acceptable tout en réduisant considérablement la latence d'inférence et la consommation en puissance. Afin de valider cette hypothèse, nous avons procédé à quantifier le modèle YOLOv5n après son entraînement, en réduisant sa précision du format de virgule flottante FP32 au format de virgule fixe INT8, qui est souvent un entier de 8 bits. Ensuite, nous avons compilé le modèle pour le rendre compatible avec le processeur matériel spécialisé DPU (Deep Processing Unit) de la carte FPGA. Finalement, nous avons procédé à l'exécution de l'inférence du modèle sur la carte FPGA et effectué une comparaison entre différentes plateformes matérielles (GPU, CPU, FPGA) en termes de latence, consommation en puissance et d'efficacité globale.

Les résultats expérimentaux obtenus montrent que l'implantation matérielle sur un circuit FPGA permet une exécution en temps réel avec une consommation en puissance plus faible que les solutions GPU, tout en rivalisant au niveau de la rapidité. De plus, la précision de détection du modèle quantifié reste proche de celle du modèle codé en virgule flottante d'origine, avec une dégradation des performances très acceptable.

En conclusion, ce projet démontre la pertinence de l'utilisation des circuits FPGA pour des tâches de détection d'objets en temps réel, notamment dans des contextes où les ressources matérielles sont limitées ou où la consommation en puissance est un critère très important.

Mots clés : FPGA, modèle YOLOv5, détection d'objets, Vitis AI, quantification, inférence temps réel, efficacité énergétique, système embarqué, processeur DPU, accélération matérielle

ABSTRACT

The main objective of this work is to propose an effective solution for real-time inference of object detection models using a reconfigurable hardware architecture. More specifically, our work focuses on the hardware implementation of the quantified YOLOv5 model on the Kria KV260 FPGA platform, using the Vitis AI framework developed by AMD Xilinx.

Our project consists of two principal stages : training a YOLOv5 object detection model based on convolutional neural networks (CNN) and optimizing this model for implementation on hardware platforms such as FPGAs with limited resources. The main hypothesis of this study is that hardware acceleration of a model through quantization would maintain acceptable accuracy while significantly reducing inference latency and energy consumption. To validate this hypothesis, we quantized the trained model YOLOv5n by lowering its accuracy from floating point FP32 to fixed point INT8, which is an integer of 8 bits. We then compiled the model to make it compatible with the FPGA board's DPU. Finally, we ran the model inference on the board and compared different hardware platforms (GPU, CPU, FPGA) in terms of latency, energy consumption, and overall efficiency.

The experimental results show that the hardware implementation on the FPGA board allows for real-time execution with lower power consumption than GPU solutions while still managing a comparable speed. In addition, the accuracy of the quantized model remains close to that of the original floating-point model, with a slight but acceptable degradation.

As a conclusion, this project demonstrates the relevance of using FPGA circuits for real-time object detection tasks, particularly in contexts where hardware resources are limited or where energy efficiency is a very important criterion.

Keywords : FPGA, YOLOv5 model, object detection, Vitis AI, quantization, real-time inference, energy efficiency, embedded system, DPU processor, hardware acceleration

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS

DL	*Deep Learning*
	Apprentissage profond
ANN	*Artificial Neural Network*
	Réseau de Neurones Artificiels
YOLO	*You Only Look Once*
CNN	*Convolutional Neural Network*
	Réseau de neurones convolutifs
ML	*Machine Learning*
	Apprentissage automatique
FPGA	*Field Programmable Gate Array*
	Réseaux de portes programmables sur site
IA	*Artificial Intelligence*
	Intelligence Artificielle
DSP	*Digital Signal Processor*
	Processeur numérique du signal
FP32	*Floating Point 32-bit*
	Nombre à virgule flottante sur 32 bits
Int8	*Integer 8-bit*
	Entier sur 8 bits
DPU	*Deep Learning Processing Unit*
	Unité de traitement pour l'apprentissage profond
CPU	*Central Processing Unit*
	Unité centrale de Traitement
GPU	*Graphics Processing Unit*
	Unité de Traitement graphique

CUDA	*Compute Unified Device Architecture*
	Architecture unifiée de calcul sur périphérique
AMD	*Advanced Micro Devices*
ROM	*Read-Only Memory*
	Mémoire morte
RAM	*Random Access Memory*
	Mémoire vive
DDR4	*Double Data Rate type 4*
	Mémoire vive DDR4
PS	*Processing System*
	Système de traitement
PL	*Programmable Logic*
	Logique programmable
CLB	*Configurable Logic Block*
	Bloc logique configurable
LUT	*Look-Up Table*
	Table de correspondance
Model	*Deep Learning Model*
	Modèle d'apprentissage profond
SSD	*Single Shot Detector*
	Détecteur d'objets monocoup
PTQ	*Post-Training Quantization*
	Quantification après entraînement
QAT	*Quantization-Aware Training*
	Entraînement sensible à la quantification
ASIC	*Application-Specific Integrated Circuit*
	Circuit intégré à applications spécifiques

MAC	*Multiply-Accumulate*
	Opération de multiplication-accumulation
PANet	*Path Aggregation Network*
	Réseau d'agrégation de chemins
FPN	*Feature Pyramid Network*
	Réseau pyramidal de caractéristiques
CSP	*Cross Stage Partial*
	Connexions partielles inter-étapes
mAP	*mean Average Precision*
	Précision moyenne moyenne
AP	*Average Precision*
	Précision moyenne
mAR	*mean Average Recall*
	Rappel moyen moyen
ReLU	*Rectified Linear Unit*
	Fonction linéaire unitaire rectifiée
tanh	*Hyperbolic Tangent*
	Fonction tangente hyperbolique
AUC-ROC	*Area Under Curve - Receiver Operating Characteristic*
	Aire sous la courbe ROC
ROC	*Receiver Operating Characteristic*
	Caractéristique de fonctionnement du récepteur
IoU	*Intersection over Union*
	Intersection sur Union
PR	*Precision-Recall*
	Précision - Rappel

LISTE DES SYMBOLES

ms	Milliseconde
MB/s	Mégaoctets par seconde
A	Ampère
V	Volt
W	Watt
J	Joule
Ops/J	Opérations par joule
GOPS/W	Giga opérations par seconde par watt
IPS	Inferences per second
Inférence par seconde	
MHz	MégaHertz
Shape (N, C, H, W)	Dimensions d'un tenseur d'entrée N : nombre d'images (batch), C : canaux, H : hauteur, W : largeur

INTRODUCTION GÉNÉRALE

Contexte

Avec la forte croissance du volume de données ainsi que les besoins en traitement temps réel d'images et de vidéos dans plusieurs domaines tels que la surveillance par caméra, la conduite autonome ou la robotique, il est nécessaire d'avoir recours à des systèmes de traitement plus robustes utilisant des techniques plus sophistiquées.

Pour répondre à ces exigences, plusieurs techniques ont été développées afin de permettre le traitement des données en temps réel. Durant ces dernières années, l'intelligence artificielle et plus précisément, l'apprentissage profond DL (*Deep Learning*) s'est montré comme un outil puissant dans le traitement d'images. L'apprentissage profond consiste à utiliser les réseaux de neurones et, particulièrement, les réseaux de neurones convolutifs (CNN) dans les tâches de classification et de détection d'objets. Ces réseaux de neurones artificiels organisés en couches sont une imitation des neurones biologiques humains où chaque neurone, représenté par une unité de calcul, est capable de transmettre l'information à une unité voisine au sein du réseau.

Cela dit, même si ces réseaux sont capables de réaliser des opérations complexes, ils nécessitent une grosse charge de calcul. Au fur et à mesure que ces réseaux deviennent profonds, c'est-à-dire qu'ils contiennent de plus en plus de couches de neurones, cette charge de calcul devient plus importante.

Pour réduire le temps de calcul, une solution réside dans l'utilisation de ressources matérielles capables de traiter un volume important de données dans un temps très court. Parmi celles-ci, les circuits intégrés à réseaux de portes programmables sur site, très connus sous le nom de FPGA (Field Programmable Gate Arrays), se démarquent grâce à leur flexibilité, de leur parallélisme massif et de leur faible latence. Toutefois, les capacités d'exécution

de certains algorithmes complexes, comme les réseaux de neurones convolutifs, reposent en pratique sur l'association du FPGA avec une unité de traitement et une configuration spécifique. En particulier, l'intégration d'un processeur embarqué et le déploiement d'un micro-logiciel (firmware) permettent de piloter la logique programmable, de gérer les transferts de données et d'assurer l'exécution des tâches de pré- et post-traitement.

La détection et la classification d'objets en temps réel représentent un défi majeur en raison de la complexité des algorithmes impliqués, tels que les modèles de détection d'objets à une passe YOLO (You Only Look Once) et des contraintes de temps strictes imposées par des applications telles que la conduite autonome et la surveillance vidéo en direct. Dans ce contexte, le FPGA constitue une plateforme matérielle personnalisable, dont le potentiel est pleinement exploité lorsqu'il est combiné à un processeur et à un micro-logiciel adapté. Cette approche permet d'adresser une large gamme d'applications, notamment la détection d'objets en temps réel, comme cela sera détaillé dans la suite de ce mémoire.

Problématique de recherche

Les réseaux de neurones profonds ont révolutionné le domaine de la vision par ordinateur, notamment dans la détection d'objets, la reconnaissance de personnes et la conduite autonome (Zhu et al., 2025). Ces applications nécessitent en général une exécution temps réel avec une latence minimale. Cependant, les modèles d'apprentissage profond (DL) requierrent une large base de données afin d'améliorer les performances. En plus, ils deviennent de plus en plus gourmands en termes de charge de calcul et, par conséquent, en consommation en puissance.

Afin d'accélérer le calcul effectué par les réseaux profonds, il faut optimiser les modèles et les déployer sur des architectures dédiées à l'accélération. Certains travaux proposant d'utiliser les plateformes matérielles comme les CPU, GPU ou les ASIC. Le problème avec la plupart des plateformes matérielles traditionnelles, c'est qu'elles n'atteignent pas une bonne

efficacité énergétique. C'est un critère fondamental lorsqu'on essaie de choisir une plateforme d'accélération (Power et al., 2023). En effet, le problème avec les CPU réside dans la limitation du nombre d'opérations arithmétiques et la bande passante mémoire restreinte (Song et al., 2024). Les GPU sont bien capables d'accélérer l'entraînement et l'exécution des CNN (convolutional neural network), mais consomment beaucoup d'énergie et nécessitent un environnement disposant de ressources matérielles importantes (Guo et al., 2017).

Une solution qui assure une bonne efficacité énergétique consiste à utiliser les réseaux de portes programmables connus sous le nom de FPGA. En effet, dans leurs travaux, (Jaiswal et al., 2025) ont réussi à implanter le modèle de détection d'objets YOLOv2 sur une carte FPGA avec une réduction de 1.08x, 1.7x et 2.9x en latence et un débit de 32.7 images par seconde.

Il faut souligner que les modèles de réseaux de neurones modernes ne sont pas directement déployables sur des systèmes embarqués. En effet, ces modèles passent par un traitement d'optimisation, telles que les techniques de compression comme la quantification (Wu et al., 2020), l'élagage (Liang et al., 2021) et la factorisation à rang inférieur (Yang et al., 2020). Ces techniques permettent de réduire considérablement la taille du modèle. On retrouve aussi les techniques d'optimisation d'architecture comme la convolution séparable en profondeur (depthwise separable convolution) (Haase and Amthor, 2020), qui visent à diminuer la charge de calcul. L'utilisation de versions plus petites des modèles (Fang et al., 2020) représente également une des techniques d'optimisation.

L'architecture complexe des réseaux de neurones, comme YOLO, qui implique des calculs intensifs, nécessite des ressources matérielles performantes pour fonctionner en temps réel. Les circuits FPGA offrent un potentiel considérable pour accélérer ces calculs grâce à leur architecture parallèle et à leur capacité de traitement à faible latence (Qian et al., 2025). Cependant, l'implantation efficace du modèle YOLOv5 sur un circuit FPGA pour la détection d'objets en temps réel pose plusieurs défis techniques, notamment la gestion des contraintes de mémoire, la configuration optimale des ressources FPGA et la synchronisation

des opérations parallèles.

Ainsi, la problématique de cette recherche réside dans la réalisation de la détection d'objets en temps réel en périphérie (Edge Computing), en implantant le modèle YOLOv5 sur un circuit FPGA. Cette étude vise à explorer les possibilités offertes par les circuits FPGA pour accélérer la détection d'objets en temps réel et à proposer des solutions innovantes pour relever ce défi.

En somme, ces défis liés aux choix de la plateforme matérielle adéquate, à la complexité du réseau de neurones et aux techniques à utiliser afin d'optimiser au mieux le modèle de détection d'objets conduisent à une question fondamentale :

Quelle architecture matérielle doit-on choisir et quels sont les ajustements et les optimisations à apporter au modèle YOLOv5 afin d'atteindre un compromis optimal entre performance, précision et efficacité énergétique pour le déployer sur un circuit FPGA ?

Objectif de recherche

L'objectif principal de ce travail consiste à optimiser et à implanter le modèle de détection d'objets YOLOv5 sur circuit FPGA. En effet, cette étude vise à rendre le modèle de détection d'objets déployable sur des systèmes embarqués via l'outil Vitis-AI de AMD/Xilinx. Le but est de réaliser l'inférence en temps réel en minimisant la latence, la consommation en puissance et l'utilisation des ressources tout en maintenant les mêmes performances du modèle.

Une analyse comparative des différentes techniques d'optimisation et des plateformes matérielles utilisées pour l'accélération sera menée afin d'identifier l'approche adéquate et efficace pour l'implantation sur un circuit FPGA.

Hypothèses

Dans le cadre de ce projet, nous avons formulé les hypothèses suivantes :

— H1 : Les circuits FPGA représentent la solution la plus adaptée pour l'accélération matérielle des modèles de détection d'objets ayant une architecture complexe.

— H2 : La réduction de la précision (quantification) des paramètres d'un modèle de détection n'impacte pas significativement ses performances et augmente de façon significative la rapidité de son exécution sur un circuit FPGA.

— H3 : On devrait avoir des résultats de détection obtenus par le modèle YOLOv5 accéléré avec un GPU identiques à ceux obtenus avec un circuit FPGA.

— H4 : L'implémentation du modèle YOLOv5 sur la plateforme FPGA Kria KV260 permet de satisfaire une contrainte de temps réel souple pour un flux vidéo à 60 images par seconde, en maintenant une latence d'inférence inférieure à la période imposée par la caméra. La validation de cette hypothèse repose sur la comparaison entre la latence d'inférence mesurée et la période imposée par la caméra, définie par la fréquence d'acquisition du flux vidéo.

$$T_{\text{inférence}}^{\max} \leq T_{\text{caméra}} = \frac{1}{60} \approx 16.67 \text{ ms}. \tag{1}$$

Méthodologie

Afin de répondre à la problématique soulevée précédemment et d'atteindre les objectifs de recherche fixés, une méthodologie rigoureuse a été adoptée. Celle-ci se décompose en plusieurs étapes allant de l'analyse théorique à l'implantation pratique sur la plateforme matérielle.

— Recherche bibliographique sur les techniques d'accélération matérielle des modèles d'intelligence artificielle, en particulier sur les différentes plateformes matérielles disponibles (GPU, FPGA et ASIC).

— Analyse de l'architecture des modèles de détection d'objets de type YOLO, en détaillant les composantes principales : la dorsale (backbone), le cou (neck), et la tête (head) du réseau.

— Sélection du modèle de détection d'objets à accélérer, en tenant compte de ses performances, de sa compatibilité avec les contraintes matérielles et de son adéquation aux cas d'usage visés.

— Choix des outils d'optimisation permettant la compression et l'adaptation du modèle, notamment en termes de quantification et d'élagage (pruning).

— Formation à l'outil Vitis-AI, proposée par AMD/Xilinx, afin de maîtriser la quantification, la compilation, et le déploiement des modèles d'apprentissage profond sur circuits FPGA.

— Modification et entraînement du modèle YOLOv5 sur trois ensembles de données :

- COCO (Common Objects in Context) pour le milieu extérieur.
- HAM10000 (Human Against Machine 10000 images) pour la détection de lésions cutanées (cancer de la peau).
- KITTI pour une application de conduite autonome.

— Quantification et compilation du modèle YOLOv5 à l'aide de l'outil Vitis-AI afin de l'adapter à l'architecture du circuit FPGA de la carte Kria KV260.

— Implantation matérielle sur la carte de développement Kria KV260 du modèle quantifié.

— Développement d'un script d'inférence en C++ permettant d'exécuter des inférences en temps réel directement sur la carte de déeloppement Kria KV260.

— Évaluation des performances du modèle YOLO déployé :

- Comparaison des métriques de détection (mAP, exactitude, rappel, F1-score) entre la version logicielle sur CPU/GPU et la version matérielle sur FPGA.
- Mesure des performances système : consommation en puissance, latence, débit en images par secondes FPS (Frames per second).

Contributions

La contribution principale de ce travail réside dans le développement d'un flux de travail (workflow) optimisé pour la quantification et l'implantation matérielle du modèle YOLOv5, à l'aide de l'outil Vitis-AI sur une carte FPGA Kria KV260. Une étude comparative des performances des implantations CPU, GPU et FPGA a été faite au niveau des ressources matérielles utilisées, des latences et de la consommation en puissance. On peut structurer nos contributions comme suit :

1. Optimisation du modèle YOLOv5
   - — Adaptation du modèle YOLOv5 aux contraintes imposées par Vitis-AI, comme la gestion des opérations non supportées telles que *permute* et *view*, ainsi que la fonction d'activation SiLU.
   - — Entraînement du modèle YOLOv5n sur les bases de données HAM10000, COCO2017 et KITTI.
   - — Quantification du modèle YOLOv5n avec le module Quantizer de Vitis-AI.
2. Implantation matérielle sur la carte Kria KV260
   - — Compilation du modèle YOLOv5n quantifié afin de générer le .xmodel compatible avec l'architecture du DPU (Deep learning Processing Unit) de la carte Kria KV260.
   - — Développement du fichier *prototxt* adapté à l'architecture de YOLOv5n.
3. Inférence et mesure des performances
   - — Développement d'un exécutable (.exe) personnalisé pour l'inférence du modèle YOLOv5n quantifié, en utilisant le langage C++ et la librairie GStreamer.
   - — Développement d'un script de mesure de performance (Latence, énergie consommée, débit en FPS et fréquences)
   - — Comparaison des performances d'un modèle YOLOv5n exécuté sur un GPU avec un modèle YOLOv5n quantifié sur un FPGA.

Organisation du mémoire

Ce mémoire est organisé comme suit : le premier chapitre donne un aperçu sur les fondamentaux des modèles de réseaux de neurones, notamment le principe de fonctionnement, les éléments clés d'un réseau de neurones et enfin l'architecture des réseaux de neurones convolutifs (CNN). Dans le deuxième chapitre, une étude sur les architectures des modèles de détection d'objets sera présentée afin de mettre en évidence les éléments importants constituants ces réseaux. Nous y aborderons également les principales métriques d'évaluation des détecteurs d'objets ainsi que les modèles à une passe. Le troisième chapitre présentera une étude approfondie sur les différentes techniques et technologies utilisées pour l'accélération matérielle des modèles de détection d'objets. Le quatrième chapitre expliquera en détail les démarches adoptées pour l'élaboration du projet ainsi qu'une discussion approfondie des résultats obtenus. Nous avons opté pour la carte Kria KV260 en raison de son excellent rapport qualité/prix. Afin de bien faciliter la compréhension du mémoire au lecteur, il est important de mentionner dans ce travail que le terme temps réel est employé au sens de temps réel souple. En génie électrique, un système temps réel strict est défini comme un système cadencé par une horloge externe, pour lequel l'ensemble des calculs associés à une période donnée est réalisé de manière déterministe avant le front d'horloge suivant.

L'implémentation de YOLOv5 sur la plateforme Kria KV260 permet une exécution à faible latence compatible avec des contraintes temporelles applicatives, telles que le traitement de flux vidéo. Toutefois, en l'absence de garanties formelles sur le temps d'exécution dans le pire cas et de synchronisation stricte sur une horloge externe, cette implémentation ne peut être considérée comme un système temps réel dur au sens strict du génie électrique. Le modèle de détection d'objet YOLOv5 a été sélectionné étant donné qu'on a commencé en 2022, période durant laquelle ce modèle a fait l'objet de plusieurs travaux de recherche et est compatible avec la version 3.0 de Vitis-AI. En revanche, le modèle YOLOv8 n'est pas encore supporté sur Vitis-AI 3.0, ce qui nécessiterait une réécriture voire un réajustement des pipelines de compilation. Une conclusion générale sera tirée afin d'inclure les perspectives de recherche qui découlent de ces travaux.

CHAPITRE 1

NOTIONS DE BASES SUR LES RÉSEAUX DE NEURONES

Ce chapitre présente les fondamentaux des réseaux de neurones artificiels, les notions de base, le rôle de chaque élément au sein d'une couche, ainsi que le principe de fonctionnement d'un réseau de neurones.

1.1 Introduction

Les réseaux de neurones artificiels sont des modèles d'apprentissage inspirés du fonctionnement du cerveau humain. Ils constituent l'un des piliers de l'intelligence artificielle et sont très utilisés dans les applications de vision par ordinateur. Les réseaux de neurones permettent d'apprendre des représentations complexes à partir de données reçues. Ils sont constitués de neurones organisés en couches. Chaque neurone reçoit des entrées, les pondère et les additionne avant d'appliquer une fonction d'activation. Grâce à un processus appelé rétropropagation, le réseau ajuste ses poids pour minimiser l'erreur.

1.2 Intelligence Artificielle

1.2.1 Définition

L'intelligence artificielle (IA) représente un domaine de recherche et de développement informatique qui vise à créer des systèmes appelés "agents" capables de réaliser des tâches qui, traditionnellement, nécessitent l'intelligence humaine. Fondamentalement, l'IA cherche à doter les machines de capacités similaires à celles des êtres humains en matière de perception, de raisonnement, d'apprentissage et d'interaction avec leur environnement. Ces ca-

pacités permettent aux systèmes d'IA de comprendre et d'interpréter des données complexes, de résoudre des problèmes, de prendre des décisions autonomes et même d'apprendre à partir de l'expérience.

Russell and Norvig (2009) expliquent qu'il existe huit définitions de l'intelligence artificielle, étalées sur deux dimensions, à savoir "la pensée" et "l'action", qui sont réparties sur quatre approches, comme le démontre la figure 1.

Thinking Humanly	**Thinking Rationally**
"The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense." (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Acting Humanly	**Acting Rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998) "AI . . . is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

Figure 1: Huit définitions de l'intelligence artificielle sous 4 approches (Russell and Norvig, 2009)

Cette explication souligne la dualité entre la perspective cognitive de l'intelligence artificielle, qui se concentre sur la pensée et le raisonnement, et la perspective comportementale, qui met également l'accent sur les actions et les performances observables. Cet aspect mesure, entre autres, à quel point on se rapproche des performances humaines. De plus, elle introduit le concept de rationalité comme une mesure de l'efficacité des systèmes d'intelligence artificielle, soulignant que la rationalité est définie par la capacité d'un système à prendre de bonnes décisions en fonction des informations disponibles.

1.3 Réseau de neurones artificiels (ANN)

1.3.1 Introduction

Les réseaux de neurones artificiels (ANN) sont des modèles numériques conçus dans le but d'imiter le fonctionnement du cerveau humain. Ils se composent d'unités de traitement interconnectées, appelées neurones, organisées en plusieurs couches. Dans cette section, le fonctionnement des réseaux de neurones ainsi que leurs composants fondamentaux seront abordés. Par la suite, une attention particulière sera accordée à la structure de réseaux de neurones à base de convolution (CNN), en détaillant les différentes couches qui les composent.

1.3.2 Principe de fonctionnement

1.3.2.1 Perceptron

Un neurone est une unité de traitement d'information qui représente l'élément fondamental d'un réseau de neurones (Haykin, 2009). Inspiré du fonctionnement biologique du cerveau humain, ces unités sont capables d'apprendre à partir des données et, par la suite, réaliser une variété de tâches complexes dans différents domaines d'application. Le principe de fonctionnement d'un neurone est illustré dans la figure 2.

Chaque neurone est en effet un élément de traitement pouvant effectuer une combinaison linéaire des entrées qu'il reçoit, en ajoutant un biais, suivie d'une fonction d'activation. Les fonctions d'activation introduisent une non-linéarité, permettant ainsi de représenter la relation complexe entre l'entrée et la sortie. Dans un réseau de neurones, chaque neurone est connecté au suivant par ce que l'on appelle un poids. Ce paramètre représente l'importance de chaque connexion dans le fonctionnement du réseau. Ce paramètre peut être ajusté et optimisé pendant l'entraînement afin d'améliorer les performances du réseau de neurones. Le modèle d'un neurone inclut également un biais qui contribue à l'augmentation ou à la

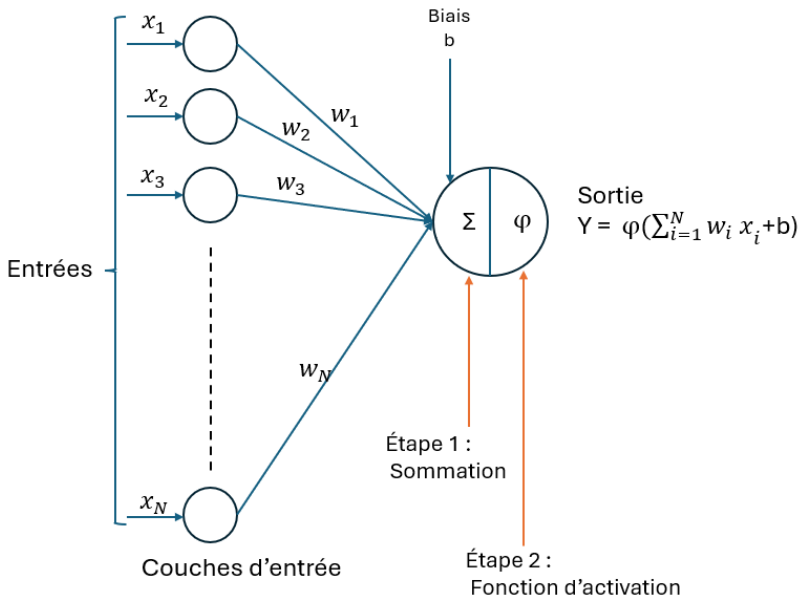


Figure 2: Modèle d'un réseau de neurone artificiel à un seul neurone (perceptron)

diminution de l'entrée nette de la fonction d'activation, selon qu'il est positif ou négatif, respectivement (Haykin, 2009).

Généralement, la sortie normalisée d'un neurone par la fonction d'activation varie soit dans une plage unitaire [0,1], soit dans une plage symétrique [-1,1]. Comme illustré à la figure 2, on peut donc résumer le fonctionnement d'un perceptron:

— L'entrée x_i au niveau de la couche d'entrée est multipliée par le poids w_i.

— Une pondération des entrées x_i avec les poids w_i est effectuée, puis ces produits sont additionnés. Le biais b est ensuite ajouté à cette somme:

$$v = \sum_i w_i \cdot x_i + b \tag{1.1}$$

— La fonction d'activation est appliquée au résultat précédent afin de produire la sortie:

$$y = \varphi(v) \tag{1.2}$$

1.3.3 Fonction d'activation

La fonction d'activation, appelée aussi fonction d'écrasement dans certains cas, permet de définir la sortie d'un neurone. Elle transforme la somme pondérée des entrées d'un neurone en une sortie en introduisant, dans la majorité des cas, une non-linéarité. Son but est d'apprendre au modèle les relations complexes et non linéaires entre l'entrée et la sortie. Il existe trois types fondamentaux de fonctions d'activation:

— **La fonction seuil**: ce type de fonction d'activation, l'une des plus basiques, se présente comme suit:

$$\varphi(v) = \begin{cases} 1 & \text{si } v \geq \theta \\ \theta & \text{sinon} \end{cases} \tag{1.3}$$

où θ est le seuil, souvent nul ($\theta = 0$). La fonction seuil est présentée graphiquement à la figure 3.

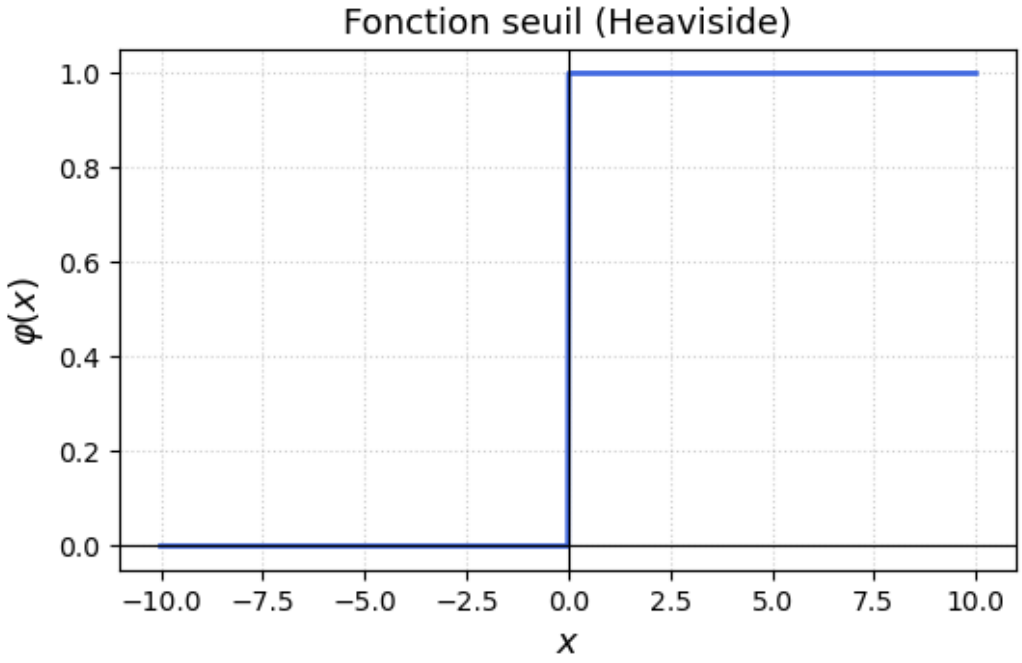


Figure 3: Fonction d'activation seuil (inspiré de (Haykin, 2009)

— **La fonction Sigmoïde**: c'est l'une des fonctions d'activation les plus utilisées dans la construction de réseaux de neurones, notamment en statistiques ou en biologie. La fonction sigmoïde est définie comme suit:

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{1.4}$$

Cette fonction est une fonction strictement croissante et présente un équilibre entre le comportement linéaire et non linéaire. Néanmoins, la fonction sigmoïde conserve aujourd'hui un rôle fondamental en tant que mécanisme de *gating* (Hu et al., 2019). Dans ce contexte, elle est employée pour moduler dynamiquement le flux d'information en produisant des coefficients de pondération compris entre 0 et 1. Ce mécanisme est largement utilisé dans les architectures à portes, telles que les réseaux de neurones récurrents à mémoire longue durée (LSTM), les unités récurrentes avec portes (GRU), ainsi que dans certains modules d'attention et de recalibrage de canaux, comme les blocs *Squeeze-and-Excitation*. La fonction sigmoïde est représentée à la figure 4.

— **La fonction linéaire par morceaux**: cette fonction est décrite comme suit :

$$\varphi(x) = \begin{cases} 1, & \text{si } x \geq +1/2, \\ x, & \text{si } -\frac{1}{2} < x < \frac{1}{2}, \\ 0, & \text{si } x \leq -\frac{-1}{2} \end{cases} \tag{1.5}$$

où le facteur d'amplification dans la région linéaire de fonctionnement est égale à 1. Elle est représentée à la figure 5.

— **La fonction tangente hyperbolique** (tanh): il s'agit d'une variante de la fonction sigmoïde. Son taux de convergence est meilleur que celui de la fonction sigmoïde, mais elle présente encore certaines limites. Cette fonction couvre une plage de valeurs centrée sur zéro. Elle est définie par la formule suivante :

$$\varphi(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \tag{1.6}$$

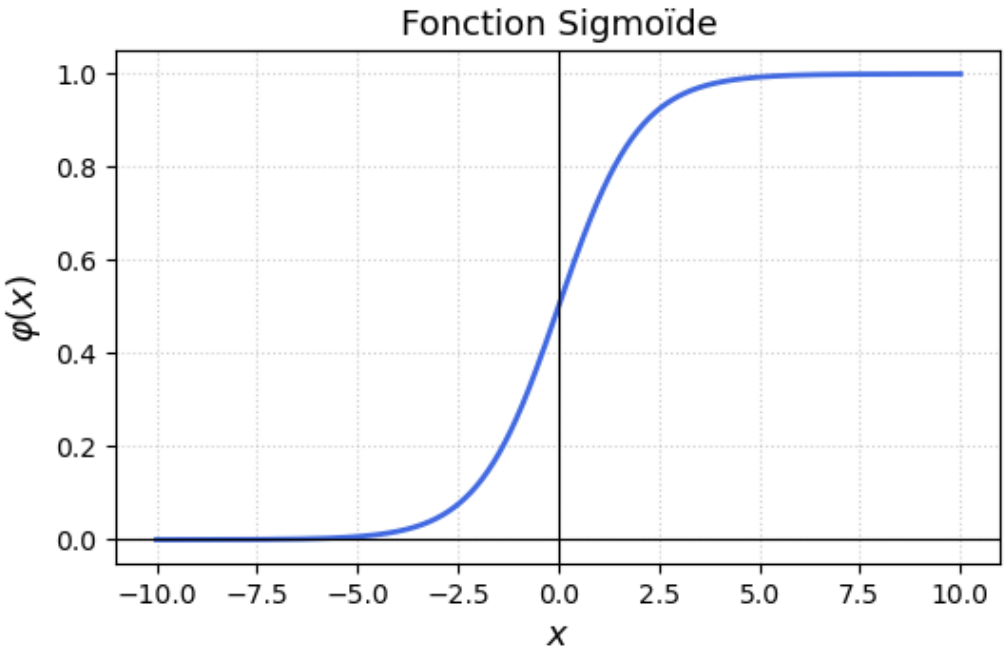


Figure 4: La fonction d'activation Sigmoïde (Haykin, 2009)

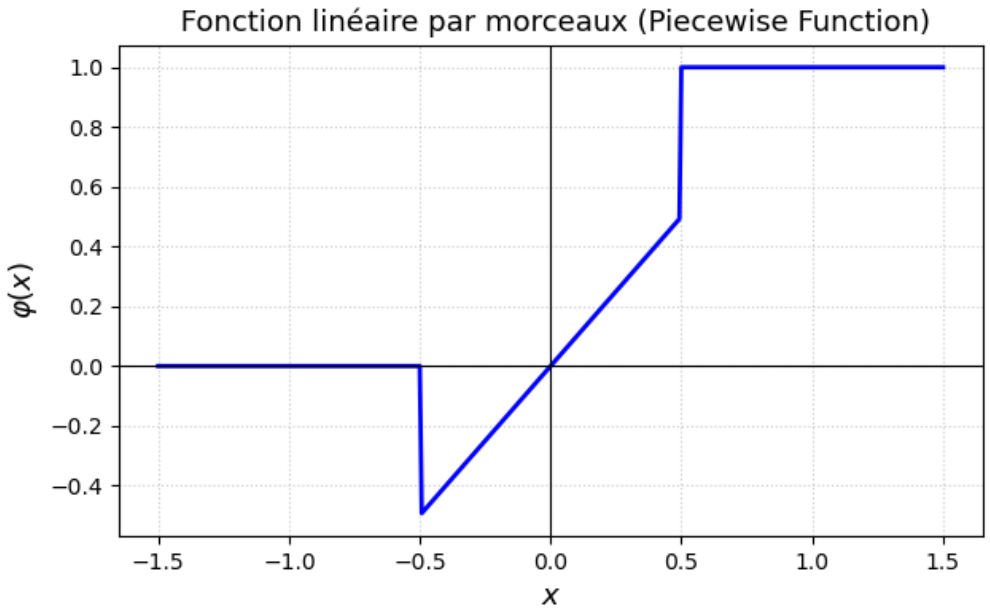


Figure 5: La fonction d'activation linéaire par morceaux (Haykin, 2009)

et elle est représentée à la figure 6.

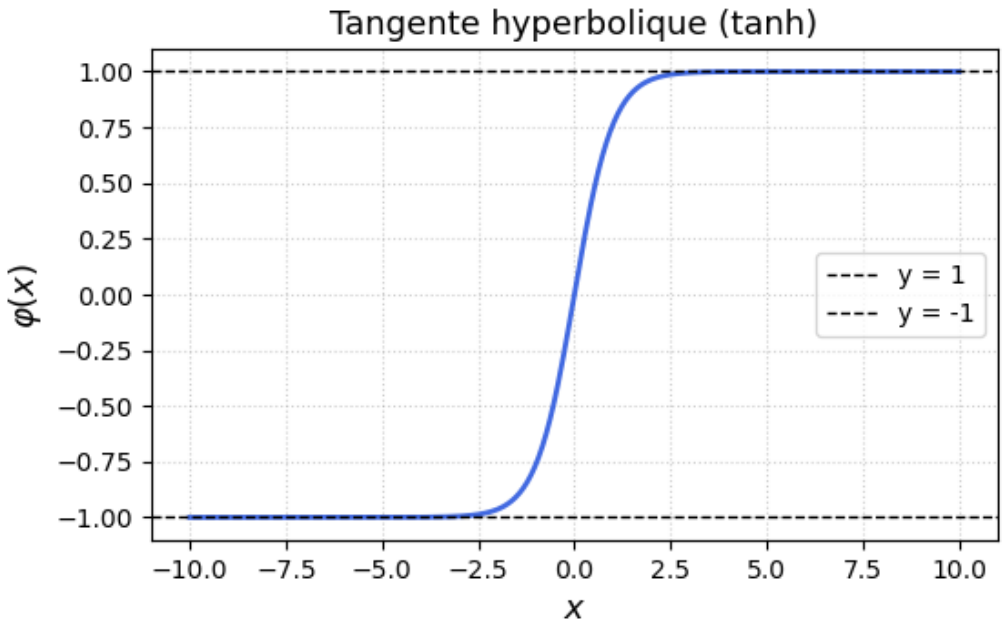


Figure 6: La fonction d'activation tangente hyperbolique (Wang et al., 2020)

Outre la fonction sigmoïde, traditionnellement utilisée dans les réseaux de neurones classiques pour des tâches de classification binaire, d'autres fonctions d'activation sont devenues centrales dans les réseaux à apprentissage profond. Parmi celles-ci, on retrouve notamment la fonction ReLU (Rectified Linear Unit) et la fonction Softplus. Wang et al. (2020) présentent en détail ces fonctions ainsi que plusieurs variantes populaires de ReLU, telles que Leaky ReLU et ELU (Exponential Linear Unit), souvent utilisées pour améliorer la convergence et éviter certains problèmes liés à la saturation des gradients (Ruder, 2017).

1.3.3.1 La fonction de l'unité linéaire rectifiée (ReLU)

La fonction ReLU est une fonction par morceaux faisant partie des fonctions d'activation les plus courantes. Elle est définie par :

$$\text{ReLU}(x) = \max(0, x) \tag{1.7}$$

Sa forme graphique est représentée à la figure 7.

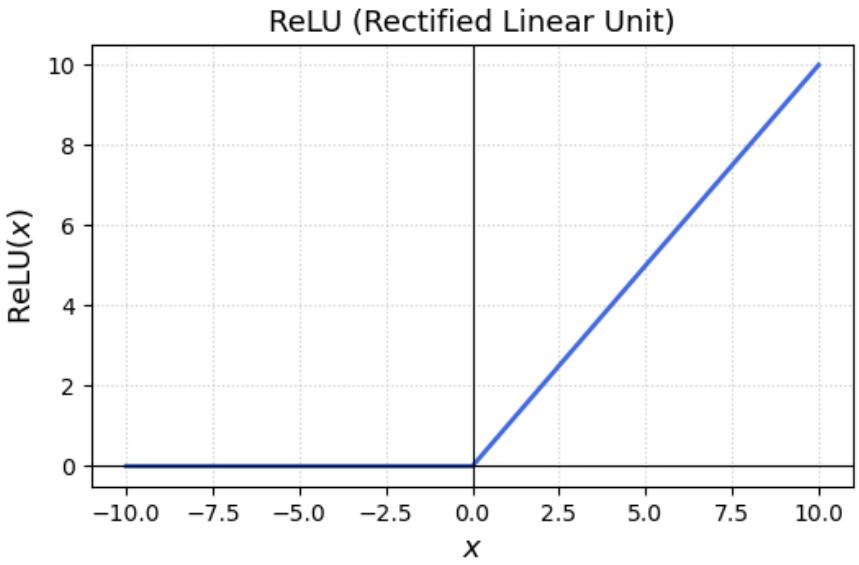


Figure 7: La fonction d'activation ReLU (Wang et al., 2020)

Cette fonction force la valeur de l'entrée à zéro si celle-ci est inférieure ou égale à zéro. Dans le cas contraire, la sortie est égale à l'entrée. En forçant certaines entrées à être nulles, cette fonction permet, en quelque sorte, de créer une caractéristique de parcimonie modérée. Elle présente de meilleurs avantages comparativement aux fonctions tangente hyperbolique et sigmoïde, étant donné qu'elle n'est pas saturée et, par conséquent, n'est pas sujette au problème de diffusion du gradient. Cependant, elle reste limitée étant donné que sa dérivée est nulle lorsque l'entrée est négative (Wang et al., 2020).

1.3.3.2 La fonction d'activation Softplus

Similaire à la fonction ReLU, cette fonction est définie par l'équation (1.8) suivante:

$$\varphi(x) = \log(1 + e^x) \tag{1.8}$$

Elle nécessite plus de calculs en raison des opérations logarithmiques et exponentielles. Une autre différence est que la fonction ReLU est exactement égale à zéro lorsque l'entrée est négative, tandis que Softplus est lisse pour les entrées proches de zéro. La fonction Softplus est représentée par la figure 8.

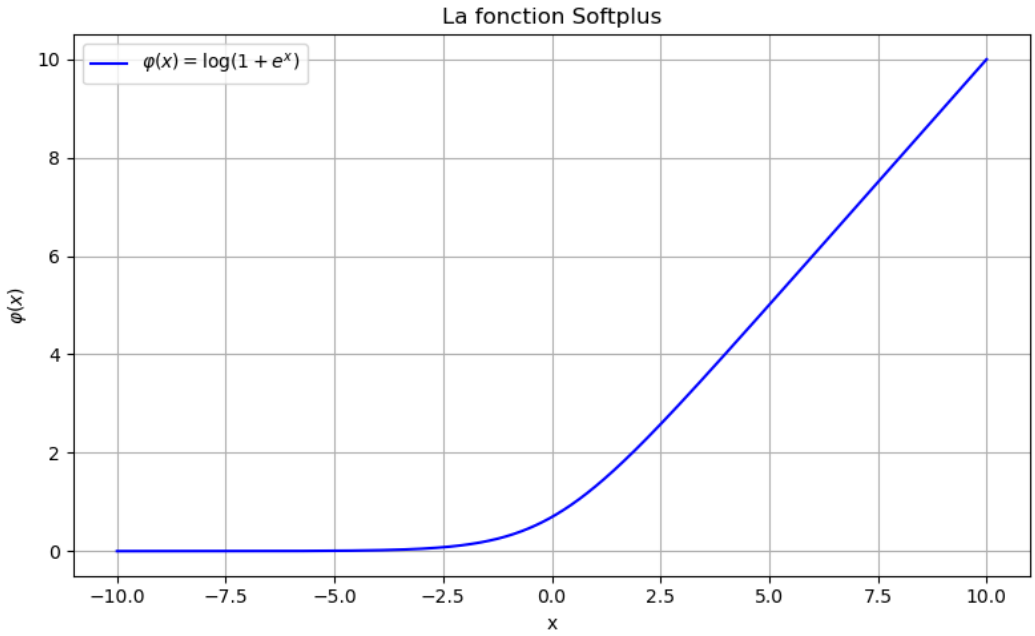


Figure 8: La fonction d'activation Softplus (Wang et al., 2020)

1.3.3.3 Variantes communes de la fonction ReLU

Depuis plusieurs années, les chercheurs ont mené des travaux visant à améliorer les fonctions d'activation. La raison est que la fonction ReLU, même si elle a été la fonction la plus avantageuse, présente un défaut dans le fonctionnement du réseau de neurones. En effet, pour les valeurs négatives, la fonction ReLU renvoie 0 en sortie. Cela peut entraîner le fait que certains neurones ne s'activent jamais ou uniquement pour ces valeurs positives. Par conséquent, certains neurones restent inactifs, étant donné que leur poids ne se mettent jamais à jour.

Ces limitations ont conduit au développement de nouvelles fonctions d'activation, telles que Leaky-ReLU, Elu, Tanh-ReLU et Softplus-ReLU.

Ces quatre fonctions sont respectivement représentées par les équations suivantes :

$$\text{LeakyReLU}(x) = \begin{cases} x, & x \geq 0, \\ \alpha x, & x < 0 \end{cases} \tag{1.9}$$

La fonction Leaky ReLU permet d'introduire une pente non nulle pour les valeurs négatives. Le paramètre α, généralement faible (dans cet exemple $\alpha = 0$), permet de maintenir un gradient non nul dans cette région, ce qui réduit ainsi le risque de neurones inactifs lors de l'apprentissage.

$$\text{ELU}(x) = \begin{cases} x, & x \geq 0, \\ \alpha\left(e^{x} - 1\right), & x < 0 \end{cases} \tag{1.10}$$

où α est un coefficient positif. La fonction ELU introduit une non-linéarité exponentielle pour les valeurs négatives, ce qui permet d'obtenir des activations moyennes plus proches de zéro. Cette propriété peut favoriser une convergence plus rapide et une meilleure stabilité numérique du processus d'apprentissage, au prix d'un coût de calcul légèrement supérieur.

$$Tanh - ReLU(x) = \begin{cases} \frac{1-e^{-2x}}{1+e^{-2x}} & \text{si } x < 0, \\ \max(0, x) & \text{sinon} \end{cases} \tag{1.11}$$

La fonction hybride Tanh-ReLU combine la continuité et la saturation de la fonction Tanh pour les valeurs négatives avec la simplicité et l'efficacité de la fonction ReLU pour les valeurs positives. Cette approche permet ainsi de lisser la transition dans la région négative tout en conservant un comportement linéaire pour les activations positives.

$$Softplus - ReLU(x) = \begin{cases} \ln(1 + e^{x}) - \ln 2 & \text{si } x < 0, \\ \max(0, x), & \text{sinon} \end{cases} \tag{1.12}$$

La fonction hybride Softplus-ReLU s'appuie sur la fonction Softplus. L'ajout du terme de décalage $ln(2)$ assure la continuité de la fonction au point $x = 0$. Cette formulation permet de réduire les discontinuités du gradient, ce qui peut être avantageux dans certains contextes d'apprentissage ou d'implémentation matérielle. Ces quatre fonctions sont représentées à la figure 9.

1.3.3.4 La fonction Softmax

La fonction Softmax est une fonction d'activation utilisée principalement dans les modèles de classification. Elle transforme un vecteur de valeurs réelles en un vecteur de probabilités dont la somme est égale à 1 (Cao et al., 2020). La fonction Softmax est définie par l'équation 1.13 suivante:

$$\text{softmax}(v_i) = \frac{\exp(v_i)}{\sum_{j=1}^{N} \exp(v_j)} \quad \text{avec } i \in \{1, \ldots, N\} \tag{1.13}$$

1.3.4 Apprentissage d'un réseau de neurones artificiel: Algorithme de rétropropagation

La rétropropagation est l'un des algorithmes fondamentaux de l'apprentissage des réseaux de neurones artificiels. L'idée derrière un tel algorithme est en effet de mettre à jour les poids qui servent de connexions entre les neurones afin de minimiser l'erreur de prédiction entre la valeur de sortie appelée prédite et la valeur réelle de la sortie (Wythoff, 1993).

Durant le processus d'entraînement, illustré dans la figure 10 et détaillé au chapitre 4, la rétropropagation se divise en deux étapes (Hou et al., 2023). La première correspond à la propagation avant: En effet, la couche d'entrée reçoit les entrèes (données brutes) sans effectuer de prétraitement. Son rôle consiste à transmettre ces données aux neurones de la première couche cachée (Wythoff, 1993). Chaque entrée est, par la suite, pondérée par les poids du réseau, et un biais est ajouté avant l'application d'une fonction d'activation (par

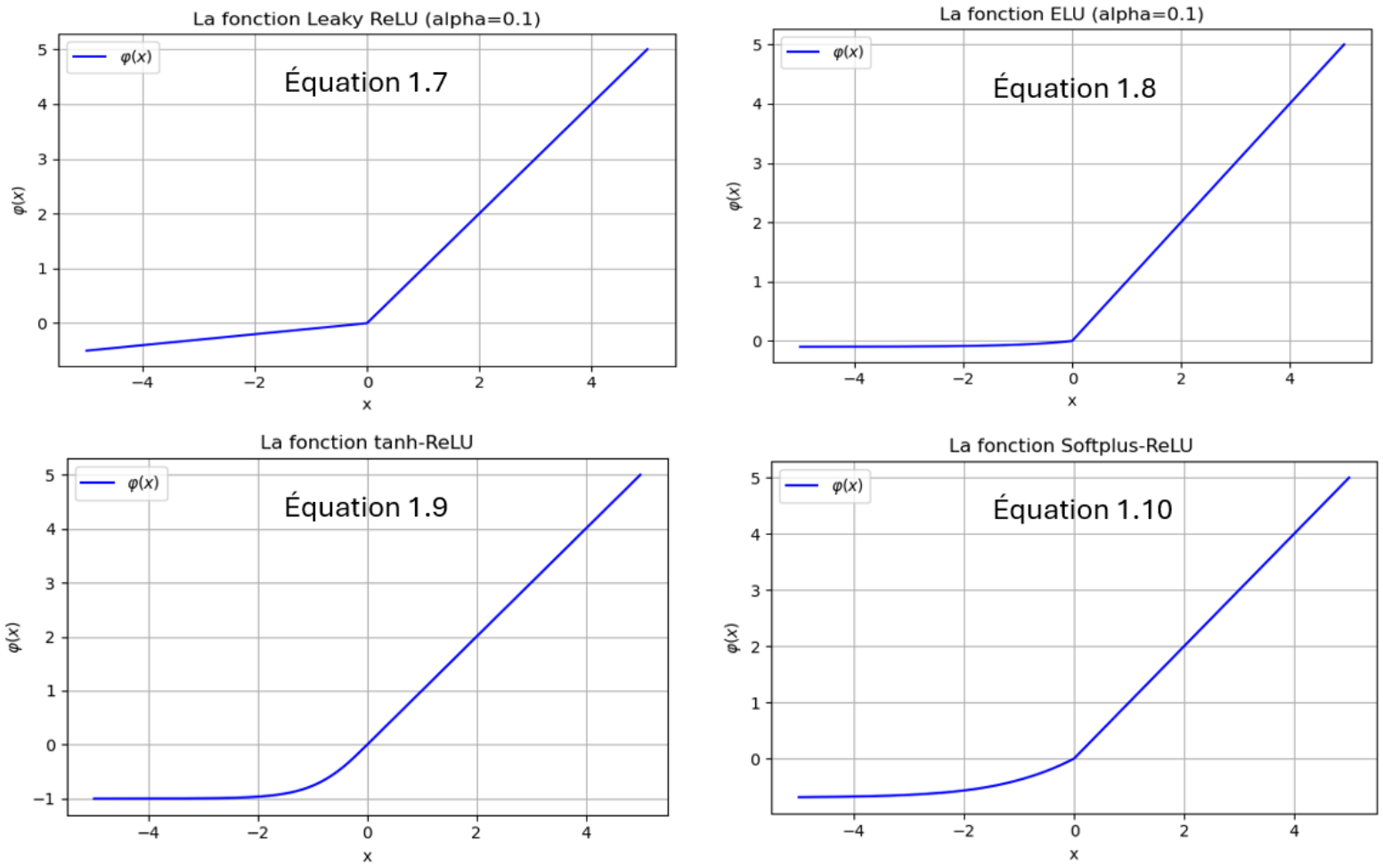


Figure 9: Les principales variantes de la fonction d'activation ReLU

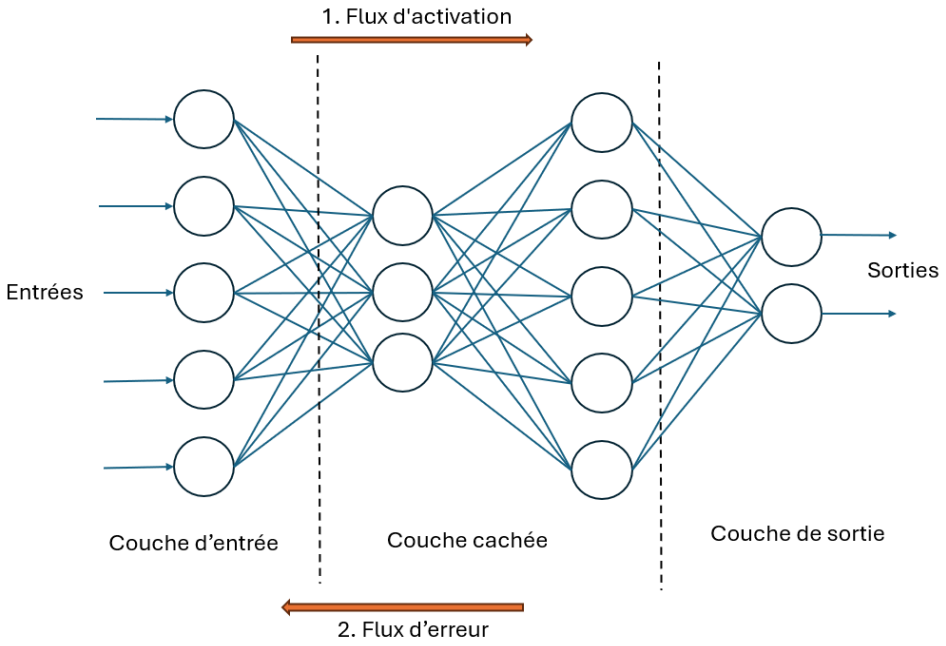


Figure 10: Schéma de l'algorithme de rétropropagation dans un réseau entièrement connecté, (inspirée de (Wythoff, 1993))

exemple, Sigmoïde), produisant une valeur comprise entre 0 et 1. Ce mécanisme se répète couche après couche jusqu'à la dernière qui génère la prédiction finale. Cette prédiction est ensuite comparée à la valeur réelle qui sert de référence, permettant de mesurer la capacité du modèle à reproduire la sortie attendue. Afin de bien évaluer les performances du modèle, on introduit la fonction de perte. Durant la propagation avant, les poids du réseau restent constants et chaque couche transmet uniquement sa propre activation à la suivante, sans rétroaction (Wythoff, 1993).

La seconde étape est la rétropropagation, essentielle à l'apprentissage supervisé. Durant cette phase, on calcule les gradients de la fonction de perte par rapport aux paramètres du réseau (poids et biais) en utilisant la règle de la chaîne. Ces gradients sont ensuite propagés dans le sens inverse, de la sortie vers l'entrée, afin de déterminer la contribution de chaque connexion à l'erreur globale. Ils servent ensuite à ajuster les poids à l'aide d'un algorithme d'optimisation, comme la descente de gradient, qui a pour but de réduire progressivement

l'erreur et d'améliorer la précision du modèle. La fonction de perte permet ainsi de quantifier l'erreur pour un exemple donné, tandis que la fonction de coût, définie comme la moyenne des pertes sur l'ensemble du jeu de données, mesure la performance globale du réseau. L'objectif de l'entraînement est de minimiser cette fonction en ajustant les poids au fil des itérations.

La mise à jour des poids à la n-ième itération s'exprime de la façon suivante (Wythoff, 1993):

$$W_{i,j}(n) \leftarrow W_{i,j}(n) - \eta \cdot \frac{\partial \text{Coût}(n)}{\partial W_{i,j}(n)} \tag{1.14}$$

— $W_{i,j}(n)$ est le poids à mettre à jour.

— η est le taux d'apprentissage.

— $\frac{\partial \text{Coût}(n)}{\partial W_{i,j}(n)}$ est la dérivée partielle de la fonction de coût par rapport au poids $W_{i,j}(n)$.

Les valeurs initiales des poids et des biais sont définies de manière aléatoire. Ces deux étapes vont se répéter durant le processus d'entraînement jusqu'à ce que l'erreur passe en dessous d'un certain seuil ou que le nombre d'itérations soit atteint.

1.4 Fonction de perte

La fonction de perte est un outil d'évaluation très utilisé en apprentissage profond afin d'optimiser les paramètres du modèle. Intégrée dans l'architecture du réseau afin d'optimiser les paramètres, cette fonction permet de quantifier l'écart entre la valeur réelle et la valeur prédite de la sortie pour chaque exemple d'apprentissage. Cet écart détermine l'ajustement des paramètres (poids et biais) (Terven et al., 2025).

Une fonction de perte possède certaines propriétés dont il faut tenir compte lors de la réalisation d'une tâche en apprentissage profond (Terven et al., 2025).

— **Convexité**: Si un minimum local est aussi un minimum global, la fonction de perte est dite convexe.

— **Différentiabilité**: Si la dérivée de la fonction de perte existe et est continue, la fonction de perte est dite différentiable. Si cette propriété est vérifiée, on peut utiliser les méthodes d'optimisation qui se basent sur le gradient.

— **Robustesse**: Une fonction de perte ne doit pas être impactée par les valeurs abérrantes (outliers).

— **Sparsité**: Un modèle doit être capable de fournir des sorties éparses si la fonction de perte favorise la sparsité, surtout lorsque le nombre de caractéristiques importantes est faible et que les données sont de haute dimension (grand nombre de variables).

— **Monotomie**: Une fonction de perte est monotone si sa valeur décroît au fur et à mesure que la valeur prédite par le modèle tend vers la valeur réelle. La monotonie permet d'assurer que le modèle tend vers la bonne solution.

Comme mentionné précédemment, le choix des fonctions de perte varie en fonction de la tâche réalisée en apprentissage profond. Dans notre contexte, nous nous intéresserons davantage à la classification multiclasses et à la détection d'objets.

1.4.1 Fonctions de perte pour la classification d'objets

1.4.1.1 L'entropie croisée binaire BCE (Binary Cross Entropy)

L'entropie croisée binaire est une technique couramment utilisée dans la tâche de classification binaire. Cette fonction quantifie la dissemblance entre la valeur prédite et la valeur réelle (Ruby and Yendapalli, 2020).

Dans le cas d'une classification binaire où les étiquettes de classes sont des 0 et des 1, la fonction BCE est définie comme:

$$L(y, p) = -\left(y \cdot \log(p) + (1 - y) \cdot \log(1 - p)\right) \tag{1.15}$$

où y est la vraie valeur et p est la probabilité prédite pour la classe 1. Si $y = 1$, la perte est

réduite à $-\log(p)$; si $y = 0$, elle est égale à $-\log(1-p)$. Cette propriété permet de bien pénaliser les erreurs faites avec une forte confiance (Terven et al., 2025).

En présence d'un déséquilibre entre les classes, c'est-à-dire qu'une des classes présentes est sous-représentée, on utilise la fonction entropie croisée binaire pondérée définie comme suit :

$$L_{\text{pondérée}}(y, p) = -\left(w_1 \cdot y \cdot \log(p) + w_0 \cdot (1-y) \cdot \log(1-p)\right) \tag{1.16}$$

avec w_1 et w_0 sont les coefficients de pondération des classes négatives et positives, respectivement.

Donc, pour toutes les données présentes, N échantillons, on obtient (Pytorch, 2020) :

$$L_{\text{BCE}}(y, p) = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log(p_n) + (1-y_n)\log(1-p_n)\right] \tag{1.17}$$

1.4.1.2 L'entropie croisée catégorique (CCE)

La perte d'entropie croisée catégorique CCE (Categorical Cross Entropy) est une extension de la fonction d'entropie croisée binaire sur plusieurs classes. Elle mesure la divergence entre la distribution réelle des classes et les probabilités prédites par le modèle. Un encodage à un seul actif (one-hot) (Samuels, 2024) est utilisé afin de représenter les valeurs réelles, renforçant ainsi les valeurs prédites correctes et pénalisant les fausses prédictions. Cette fonction vérifie la propriété de différentiabilité, la rendant ainsi pratique pour l'optimisation basée sur le gradient.

Pour une seule donnée (un seul échantillon) avec C classes, la fonction perte d'Entropie Croisée Catégorique (CCE) s'écrit :

$$\mathcal{L}_i(\hat{p}_i, y_i) = -\sum_{j=1}^{C} y_j \log(p_j) \tag{1.18}$$

où :

— y_j : valeur réelle pour la classe j (1 si c'est la bonne classe, 0 sinon),

— p_j : probabilité prédite que l'échantillon appartienne à la classe j.

Pour un ensemble de N échantillons et C classes, la CCE est définie comme :

$$\mathcal{L}_{\text{CCE}} = -\frac{1}{n}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\log(p_{i,j}) \tag{1.19}$$

où N est le nombre d'échantillons, C le nombre de classes, $y_{i,j}$ est la vraie valeur au format one-hot de la classe j pour l'échantillon i et $p_{i,j}$ est la probabilité prédite par le modèle de la classe j pour l'échantillon i (Terven et al., 2025).

1.4.1.3 Perte d'Entropie Croisée Catégorique Sparse (sparse CCE)

La perte d'Entropie Croisée Catégorique Sparse (sparse categorical cross-entropy) est l'une des variantes de l'entropie croisée catégorique standard. Comparé à la version classique qui nécessite un encodage one-hot des étiquettes, la Sparse CCE utilise directement des étiquettes entières représentant les classes, ce qui permet de réduire l'utilisation mémoire et de simplifier l'étape de prétraitement des données, notamment lorsque le nombre de classes est élevé. La perte peut être calculée individuellement pour chaque classe:

$$L_{\text{sparse}}(y_i, \hat{p}_i) = -\log(\hat{p}_{i,y_i}) \tag{1.20}$$

où $\hat{p}_i, y_i$ représente respectivement la probabilité assignée à la classe correcte y_i. Cette perte peut également être calculée de manière globale pour l'ensemble des données:

$$L_{\text{sparseCCE}} = -\frac{1}{n}\sum_{i=1}^{n}\log(\hat{y}_{i,y_i}) \tag{1.21}$$

Cette fonction vérifie la propriété de différentiabilité et est, par conséquent, efficace pour l'optimisation basée sur le gradient. Elle est particulièrement adaptée aux données de

grandes dimensions, où l'encodage sous forme d'entiers simplifie le processus (Terven et al., 2025).

1.4.1.4 Autres fonctions de perte pour la classification

Outre les fonctions de perte de base, telles que l'entropie croisée, il existe plusieurs variantes et alternatives qui ont été développées afin de mieux répondre aux besoins spécifiques de certaines tâches de classification. Ces fonctions visent notamment à améliorer la généralisation, à réduire le sur-apprentissage (overfitting) et à traiter efficacement les bases de données déséquilibrées. Le tableau 1 présente un aperçu de quelques-unes de ces fonctions de perte, en soulignant leurs avantages respectifs.

1.4.2 Fonctions de perte pour les détecteurs d'objets

Dans le contexte de la détection d'objets, les modèles utilisent plusieurs combinaisons de fonctions de perte. On peut les regrouper en deux catégories, à savoir, la localisation, c'est-à-dire une régression des coordonnées des boîtes englobantes des objets, et la reconnaissance, c'est-à-dire la classification d'objets.

1.4.2.1 Fonction de perte Smooth L1 et Balanced L1

Smooth L1 est une fonction de perte très populaire en détection d'objet introduite avec le modèle Fast R-CNN (Terven et al., 2025; Girshick, 2015). Elle est définie par l'équation 1.22 suivante :

$$L(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & \text{if } |y - \hat{y}| < \beta, \\ |y - \hat{y}| - \frac{1}{2}\beta, & \text{Sinon} \end{cases} \tag{1.22}$$

où y est la valeur réelle, $\hat{y}$ est la valeur prédite et β une valeur seuil.

Table 1: Autres fonctions perte pour la classification (Terven et al., 2025)

fonction de perte	**Description**
Cross-Entropy with Label Smoothing	— Redistribue la confiance des labels entre les classes afin d'éviter un excès de confiance. — Efficace pour améliorer la généralisation.
Negative Log-Likelihood (NLL)	— Favorise les probabilités pour les classes correctes. — Identique à la CCE au niveau de l'encodage one-hot.
PolyLoss	— Pratique pour la classification où les bases de données sont déséquilibrées. — Cette fonction peut être calibrée en ajustant les coefficients polynomiaux.
Poly-1 Loss	— Variante de la fonction PolyLoss. Elle admet un hyperparamètre unique et efficace pour réduire le surapprentissage pour les bases déséquilibrées. — Utile pour les tâches nécessitant une bonne précision et une configuration simple des hyperparamètres.
Hinge Loss	— Utilisé généralement pour les tâches à marge maximale notamment les SVM. — Efficace dans les scénarios nécessitant une marge de classification robuste et peut être adapté aux problèmes multi-classes.
Squared Hinge Loss	— Variant de la fonction de perte Hinge Loss, favorisant fortement une précision de décision plus robuste.

La fonction de perte *balanced L1* est une version améliorée de la fonction Smooth L1 traditionnelle. Cette fonction permet de réguler la régression des coordonnées des boîtes englobantes, surpassant ainsi le MSE traditionnel. Cette fonction vise à améliorer la précision de la localisation dans les tâches de régression des boîtes englobantes (Terven et al., 2025).

La fonction *Balanced L1* est déifnie par la formule 1.23 suivante :

$$L_{\text{balanced}}(x) = \begin{cases} \beta \cdot \ln\left(\frac{|x|}{\beta} + 1\right), & \text{if } |x| \geq \beta, \\ \frac{x^2+\beta^2}{2\beta}, & \text{Sinon} \end{cases} \tag{1.23}$$

où x est l'erreur entre la valeur prédite et la valeur réelle et β est le paramètre seuil qui détermine le passage entre le comportement logarithmique et quadratique.

1.4.2.2 Fonction perte Intersection sur Union (IoU)

L'intersection sur l'union IoU (Intersection Over Union) est une métrique d'évaluation utilisée dans le contexte de détection d'objets qui sert à mesurer l'intersection entre la boîte englobante prédite et la boîte englobante réelle appelée aussi boîte réelle. Comme le montre la figure 11, cette métrique est calculée comme le rapport de l'intersection sur l'union (Terven et al., 2025).

$$\text{IoU} = \frac{\text{Area}(B_p \cap B_t)}{\text{Area}(B_p \cup B_t)} \tag{1.24}$$

où B_p est la boîte prédite et B_t boîte englobante réelle (vraie). La fonction de perte IoU est calculée comme suit:

$$L_{IoU} = 1 - \text{IoU} \tag{1.25}$$

IoU favorise les valeurs d'intersection élevée, mais elle est limitée quand les boîtes englobantes ne se chevauchent pas. Ceci est dû au fait que le rapport est égale à zéro. Dans ce cas, le modèle ne peut pas ajuster la position de la boîte prédite durant l'entraînement

(Terven et al., 2025).

Figure 11: Exemples de calcul d'Intersection sur Union (IoU)

1.4.2.3 Fonction perte IoU généralisé (GIoU)

L'intersection sur l'union généralisée (GIoU) est une autre forme de IoU qui rajoute la section non couverte par l'union de la boîte prédite et la boîte réelle (vraie). Cette fonction est définie comme:

$$L_{\text{GIoU}}(B_p, B_t) = 1 - \text{IoU} + \frac{\text{Area}(C - (B_p \cup B_t))}{\text{Area}(C)} \tag{1.26}$$

où C est la plus petite zone rectangulaire contenant B_p et B_t.

Cette fonction est utile pour remédier aux limitations de la fonction de perte IoU quand les deux boîtes prédite et réelle ne se chevauchent pas du tout ou très partiellement. Cependant, la fonction GIoU a aussi ses limitations. En effet, vu que cette fonction ne traite pas directement la distance entre les centroïdes des boîtes englobantes, elle peut mener à un désalignement spatial. Prenons un cas qui met la limitation de l'IOU en évidence, lorsque l'IOU est nulle, on sait que les boîtes ne se chevauchent pas. En revanche, on ne sait pas si

elles sont alignées ou très éloignées. Le GIoU (illustré dans la figure 12) diminue lorsque les boîtes sont éloignées. Plus C est grand, plus la pénalité est forte. En outre, cette fonction ne prend pas en compte la différence de dimension qui existe entre la boîte prédite et la boîte réelle (vraie), surtout dans les scénarios où il y a une variété d'objets présents et où les boîtes englobantes peuvent se chevaucher. Ceci cause des résultats de régression inexacts (Terven et al., 2025).

1.4.2.4 Les fonctions pertes Distance-IoU et Complete-IoU

Afin de résoudre les limitations d'exactitude et de régression de la fonction GIoU, deux fonctions Distance-IoU (DIoU) et Complete-IoU (CIoU) ont été introduites afin de rajouter respectivement la distance entre les centroïdes dans la perte et la cohérence des ratios de dimensions entre les boîtes englobantes (Terven et al., 2025). La fonction DIoU intègre dans la perte la distance entre les centroïdes des boîtes prédites et des boîtes réelles, ce qui permet de mieux gérer le désalignement spatial. Elle est définie comme suit :

$$L_{\text{DIoU}}(B_p, B_t) = 1 - \text{IoU} + \frac{d^2(B_p, B_t)}{c^2} \tag{1.27}$$

où B_p et B_t étant respectivement la boîte prédite et la boîte réelle. $d(B_p, B_t)$ est la distance euclidienne entre les deux centroides des boîtes et c est la longueur diagonale de la plus petite boîte englobante contenant B_p et B_t.

La fonction CIoU est une extension de la fonction DIoU qui rajoute un terme supplémentaire pour prendre en compte la cohérence des proportions (rapport d'aspect) entre les boîtes englobantes (Terven et al., 2025). Sa formulation est la suivante:

$$L_{\text{CIoU}}(B_p, B_t) = L_{\text{DIoU}}(B_p, B_t) + \alpha \cdot v \tag{1.28}$$

où v mesure la cohérence des dimensions entre les boîtes B_p et B_t, α est un facteur de

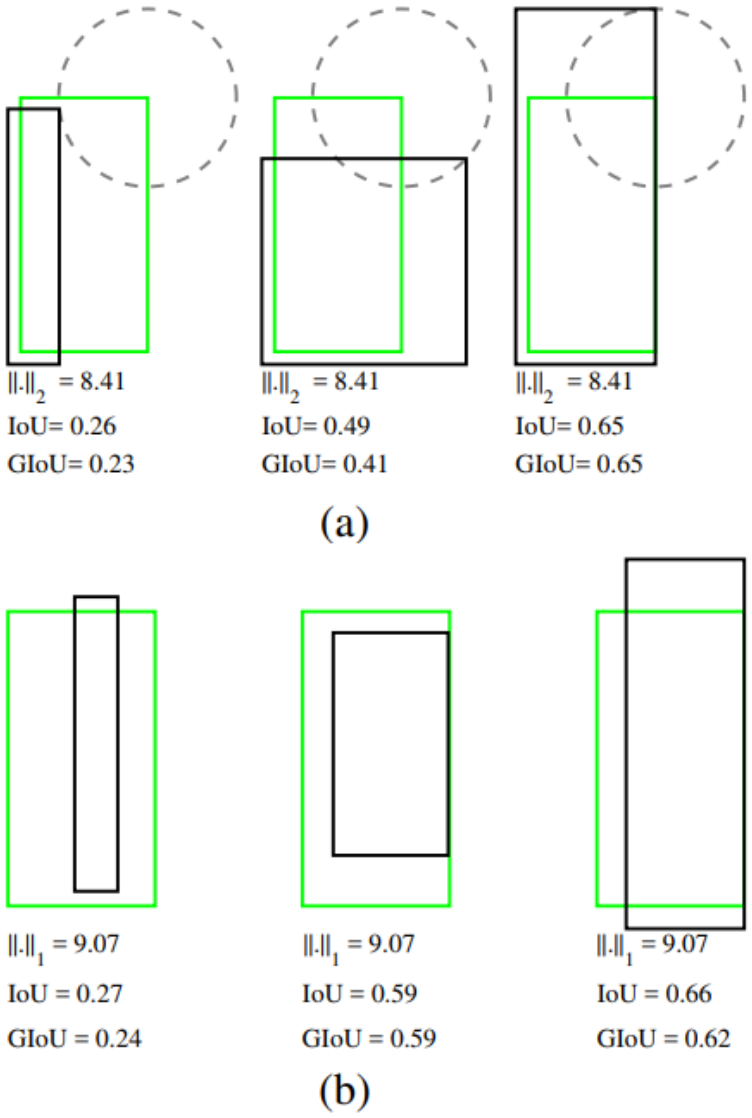


Figure 12: Différences entre distances géométriques et métriques IoU/GIoU. Dans les exemples (a) et (b), les boîtes englobantes sont représentées respectivement soit par leurs coins (x_1, y_1, x_2, y_2), soit par leur centre et leur taille (x_c, y_c, w, h). Les distances ℓ_2 (cas a) et ℓ_1 (cas b) entre les représentations des deux rectangles sont exactement les mêmes pour les trois exemples, ce qui signifie que, dans l'espace des paramètres, les situations semblent équivalentes. Cependant, IoU et GIoU sont différents. IoU mesure le chevauchement, tandis que GIOU mesure la distance spatiale entre les boîtes.

pondération qui ajuste l'importance de ce terme dans la perte:

$$\alpha = \frac{v}{(1 - \text{IoU}) + v} \tag{1.29}$$

1.4.2.5 La perte focale (Focal Loss)

Un problème très connu en classification et en détection d'objet est le déséquilibre des classes. Ce problème est généralement causé par une sous-représentation de certaines classes par rapport à d'autres. Ce problème a tendance à forcer le modèle à tenir compte que des classes dominantes et avoir de la difficulté à reconnaître les classes minoritaies. Une fonction de perte appelée *Focal Loss* (Lin et al., 2020) a été conçue dans l'objectif de résoudre ce problème. Il s'agit d'une amélioration de l'entropie croisée standard qui vise à se focaliser sur les classes mineures. Sa formule est la suivante:

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t) \tag{1.30}$$

où p_t est la probabilité de la classe réelle prédite par le modèle, α_t est un facteur de pondération qui ajuste l'importance accordée à chaque classe et γ est un facteur de focalisation qui modifie le taux de contribution des cas faciles à la valeur de la perte (Terven et al., 2025).

1.4.2.6 Fonction perte YOLO

La fonction de perte YOLO, spécialement conçue pour les modèles de détection d'objets YOLO, est une combinaison de différentes fonctions de perte qui sert à traiter les deux défis de localisation et de classification. Cette fonction de perte mesure trois éléments d'échelles importantes pour le modèle de détection d'objets à savoir:

— **La perte de localisation**: Cette fonction calcule la différence entre les valeurs prédites des coordonnées de la boîte englobante réelle et les vraies valeurs comme une somme

des carrées d'erreurs.

$$L_{\text{loc}} = \sum_{\text{pred box}} \lambda_{\text{coord}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (w_i - \hat{w}_i)^2 + (h_i - \hat{h}_i)^2 \right], \tag{1.31}$$

où $(\hat{x}_i, \hat{y}_i), \hat{w}_i$ et $\hat{h}_i$ sont respectivement les coordonnées du centre, la largeur et la hauteur de la boîte prédite, $(x_i, y_i), w_i, h_i$ sont respectivement les coordonnées du centre, la largeur et la hauteur de la boîte réelle et λ_{coord} est un facteur de pondération (Terven et al., 2025).

— **La perte d'objectivité**: Cette perte évalue le score de confiance entre la classe prédite et la vraie étiquette de la classe. C'est généralement d'une perte d'entropie croisée ou d'une erreur quadratique moyenne :

$$L_{\text{obj}} = \sum_{i \in \text{box}} \left(C_i - \hat{C}_i \right)^2, \tag{1.32}$$

où C_i et $\hat{C}_i$ sont respectivement les scores de confiances réelles et prédite (Terven et al., 2025).

— **La perte de classification** Cette fonction calcule la différence entre la probabilité de la classe prédite et l'étiquette de classe réelles. Cette fonction est identique à une entropie croisée:

$$L_{\text{class}} = - \sum_{\text{pred box}} \sum_{c=1}^{C} p_i(c) \cdot \log \left(\hat{p}_i(c) \right), \tag{1.33}$$

où $\hat{p}_i(c)$ et $p_i(c)$ et sont respectivement les probabilités prédites par le modèle et les probabilités réelles pour la classe c (Terven et al., 2025).

1.5 Réseaux de neurones convolutifs (CNN)

Les réseaux de neurones profonds DNN (Deep Neural Network) sont des réseaux constitués d'un grand nombre de couches cachées entre l'entrée et la sortie. Les architectures de

réseaux de neurones profonds varient en fonction des connexions entre les couches (Carini, 2023). Dans une architecture de réseaux de neurones, on définit :

— **Une couche entièrement connectée** FC (Fully Connected layer) c'est-à-dire, toute couche dont les sorties sont une somme pondérée des poids de toutes les entrées.

— **Une couche faiblement connecté** SC (Sparsly Connected): Toute couche dont les sorties sont une somme pondérée des poids d'un sous ensemble des entrées.

En raison du nombre assez important de paramètres et d'opérations liés à une couche FC, l'implantation de celle-ci pose un défi majeur.

Un réseau de neurones convolutionnel CNN (Convolutionnal Neural Network), illustré à la figure 13, est un modèle d'apprentissage profond spécialisé dans la reconnaissance de motifs au sein des données, notamment des images. Il exploite des couches convolutives pour extraire automatiquement des caractéristiques pertinentes. Cette capacité à capturer les structures locales rend les CNN particulièrement efficaces pour des tâches de reconnaissance d'images, de classification et de détection d'objets (Carini, 2023).

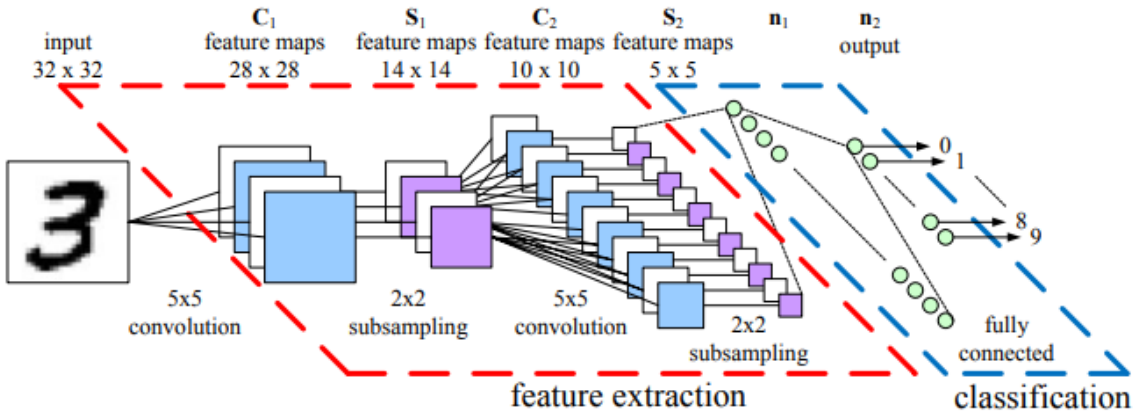


Figure 13: Exemple d'architecture d'un réseau de neurones convolutif CNN (Peemen et al., 2011)

Ces réseaux se basent sur une structure hiérarchique différente de celle des réseaux de neurones artificiels ANN (Artificial Neural Network) classique. En effet, la partie en amont

se charge de l'extraction des caractéristiques, tandis que la partie en aval, qui représente un modèle ANN classique, effectue la classification. Les réseaux CNN sont très robustes pour les tâches de détection d'objets et de reconnaissance d'images numériques. Une image numérique est une représentation visuelle de l'image sous forme numérique, généralement binaire. Une image numérique est constituée de petits éléments appelés pixels. Les pixels présents sur l'image sont placés dans une grille qui contient les valeurs des pixels. On obtient alors une matrice qui représente notre image. La valeur d'un pixel contenue dans cette matrice indique la luminosité et la couleur.

Afin de classifier une image, un réseau CNN prend en entrée une matrice de pixels représentant l'image. Cette matrice est transformée à travers plusieurs couches convolutives en une ou plusieurs cartes de caractéristiques (feature maps) qui représentent les motifs extraits à différents niveaux. Pendant l'apprentissage, les poids des filtres (et éventuellement les biais) de toutes les couches sont ajustés par rétropropagation afin d'optimiser la performance du modèle.

On distingue plusieurs types de couches dans l'architecture d'un CNN. Les plus connues sont:

— Les couches de convolution

— Les couches de régularisation par abandon aléatoire

— Les couches de sous-échantillonnage

— Les couches entièrement connectées

— Les couches d'activation

Les couches entièrement connectées et les fonctions d'activation ont été détaillées lors de la présentation des réseaux ANN. Nous détaillerons par la suite uniquement les couches propres aux réseaux CNN.

1.5.1 Couches de convolution

Une couche de convolution est fondée sur l'opération de convolution. La convolution dans le traitement d'images s'effectue via un filtre (kernel) (Carini, 2023). Considérons une matrice A pour l'image et une matrice K pour le filtre, toutes deux de dimensions 3×3.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad K = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \tag{1.34}$$

La convolution est donnée par :

$$A * K = \sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} \cdot k_{ij} \tag{1.35}$$

Les images qui représentent l'entrée du réseau passent par une couche d'extraction de caractéristiques. Les filtres ayant une taille plus petite que celle de la matrice qui représente l'image parcourent cette dernière en glissant verticalement et horizontalement (Teo et al., 2021). Les couches convolutives dans ces réseaux utilisent des noyaux de dimensions 3x3 ou 5x5. Les filtres glissent verticalement et horizontalement avec un pas (**stride**) de 1 pixel, par exemple. Une multiplication élément par élément s'effectue entre le filtre et la région couverte par le filtre sur l'image.

Prenons le cas de la figure 13: le filtre est placé dans le coin supérieur gauche de l'image, en appliquant la convolution, on obtient 1x1+1x7+1x6+1x5=19. Ensuite, en utilisant le pas de déplacement(stride), on déplace le noyau du kernel vers le bas et la nouvelle opération donne $1 \times 7 + 1 \times 5 + 1 \times 8 = 24$. Cette opération se répéte jusqu'à ce que tous les éléments de la première colonne soient traités. Ensuite, le même processus s'applique aux colonnes suivantes jusqu'au point où tous les pixels de l'image ont été parcourus (Wu, 2017). Les résultats de ces multiplications sont additionnés afin d'obtenir une valeur finale

assignée au pixel de la carte de caractéristiques. Le filtre se déplace ensuite vers la droite et le processus se répète jusqu'à ce que tous les éléments de l'image soient couverts par le filtre.

À la fin de la convolution, on obtient une nouvelle matrice de taille:

$$\text{Taille de sortie} = \left\lfloor \frac{(m + 2p) - n}{s} \right\rfloor + 1, \tag{1.36}$$

où :

— m est la taille de l'image d'entrée

— n est la taille du filtre

— p est la taille du padding ajouté

— s est le stride (pas de déplacement du filtre sur l'image).

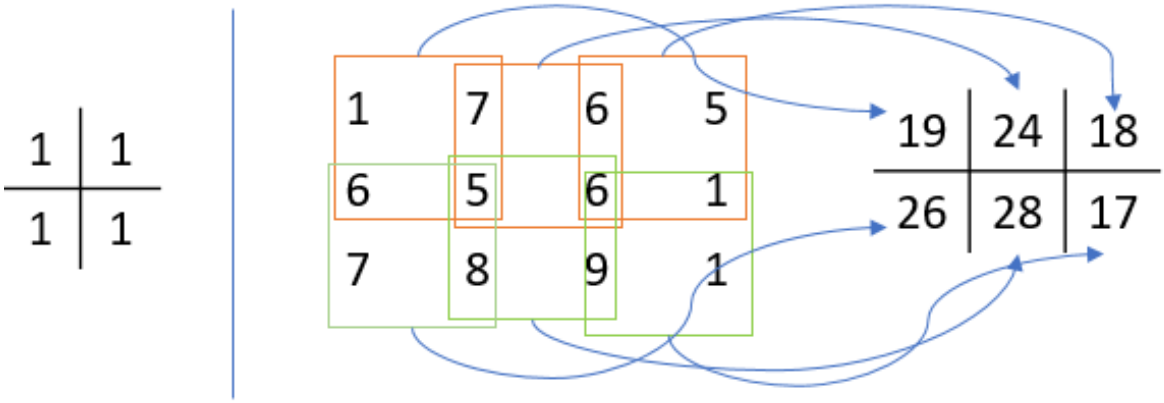


Figure 14: Opération de convolution simple avec un filtre 2x2, (inspiré de Wu (2017))

1.5.1.1 Remplissage

Dans certains cas, il serait plus important de préserver la dimension de l'image après avoir effectué plusieurs opérations de convolution. Le remplissage, appelé aussi *padding*, permet de réaliser cet objectif. Le remplissage, illustré dans la figure 15 consiste à ajouter

des pixels dans l'image au niveau des bordures (généralement des pixels nuls). Le padding sert aussi à recueillir les caractéristiques que l'on trouve au niveau des bords de l'image.

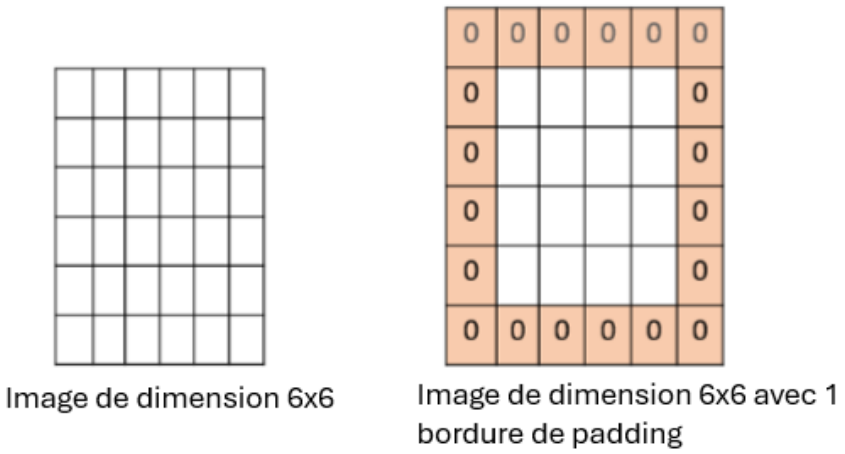


Figure 15: Exemple de remplissage dans une image numérique (Carini, 2023)

1.5.1.2 Pas de décalage

Lors d'une opération de convolution sur une image numérique, le filtre se déplace d'un certain nombre de pixels. Ce nombre de pixels est défini comme étant le pas de décalage, appelé aussi *stride*, illustré dans la figure 16. Ce paramètre permet d'ajuster la dimension de la carte des caractéristiques.

1.5.2 Couches de sous-échantillonnage

Les couches de sous-échantillonnage sont des couches intermédiaires dans les réseaux CNN. Ces couches ont pour objectif de réduire la dimension spatiale de la carte des caractéristiques produites par les couches de convolution. En réduisant la dimension, le nombre de paramètres et la complexité du modèle, les couches de sous-échantillonnage contribuent à la régularisation dans les CNN. Parmi les différentes techniques de pooling, on distingue

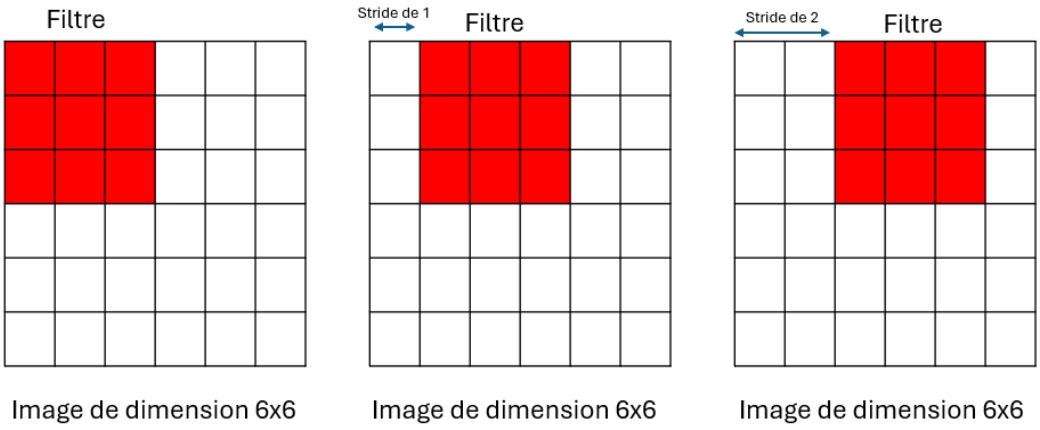


Figure 16: Illustration du déplacement du filtre pour deux valeurs du pas (stride)

le *pooling maximal* et le *pooling moyen*. Le max pooling sélectionne les valeurs maximales dans la fenêtre de pooling, tandis que le pooling moyen calcule la moyenne des valeurs dans cette fenêtre. Ces opérations permettent de réduire la taille de l'image tout en préservant les caractéristiques importantes de la carte de caractéristiques. Le principe de fonctionnement du pooling est illustré dans la figure 17:

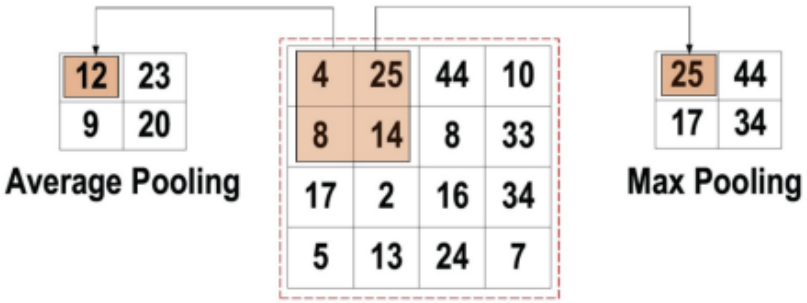


Figure 17: Opération de pooling maximal et pooling (Alzubaidi et al., 2021)

1.5.3 Couche d'activation

Après l'application du filtre sur l'image d'entrée, la sortie est transmise à une fonction mathématique appelée fonction d'activation. Parmi celles-ci, la fonction ReLU (Rectified Linear Unit) est l'une des plus utilisées dans les réseaux de neurones convolutifs (CNN) pour l'extraction de caractéristiques. Il s'agit d'appliquer la fonction à chaque valeur contenue dans la matrice, illustrée dans la figure 18.

La fonction ReLU agit en remplaçant toutes les valeurs négatives par zéro tout en conservant inchangées les valeurs positives, ce qui s'écrit :

$$\text{ReLU}(x) = \max(0, x) \tag{1.37}$$

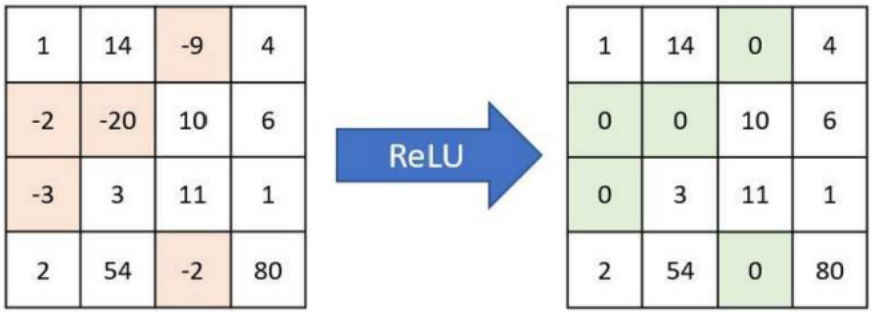


Figure 18: Extraction des caractéristiques avec la fonction d'activation ReLU

1.6 Conclusion

Les réseaux de neurones sont un concept fondamental qui a profondément révolutionné l'intelligence artificielle. Dans ce chapitre, un aperçu des composants principaux d'un réseau de neurones est présenté. Une étude des différentes couches et de leur paramètrage a été menée. Grâce à l'enchaînement judicieux des différentes couches d'un réseau convolutif (remplissage, pooling et fonctions d'activation), les CNN parviennent à extraire éfficacement

les caractéristiques abstraites présentes sur une image. Ce type de structure de réseau est l'élément clé pour la conception et l'optimisation des détecteurs d'objets, qui seront abordés dans le chapitre suivant.

CHAPITRE 2

ARCHITECTURE DES MODÈLES DE DÉTECTION D'OBJETS

Dans ce chapitre, la structure des modèles de détection d'objets sera présentée en détail. Les métriques utilisées pour évaluer les performances des modèles de détection d'objets seront ensuite introduites. Enfin, les architectures des modèles de détection d'objets à une seule passe seront abordées, avec une discussion sur le choix de la structure la plus adaptée.

2.1 Structure d'un modèle de détection d'objets

La détection d'objets est l'une des applications courantes en vision par ordinateur, qui consiste à classifier, mais aussi à localiser les objets présents dans une image. Dans un réseau de neurones profond capable d'effectuer des tâches de vision par ordinateur, notamment la détection d'objets, trois éléments majeurs constituent l'architecture du réseau:

— La dorsale (backbone).

— Le cou (neck).

— La tête (head).

L'architecture d'un détecteur d'objets typique est illustrée dans la figure 19:

2.1.1 La dorsale (Backbone)

La dorsale, en tant que premier élément qui reçoit les entrées, est chargée de l'extraction des caractéristiques et de fournir des informations importantes relatives à la position ou à la structure de l'objet. La dorsale est généralement constituée d'un classifieur dont on exclue les couches de sortie dédiées à la classification (Kateb et al., 2021). Le choix d'une dorsale dans

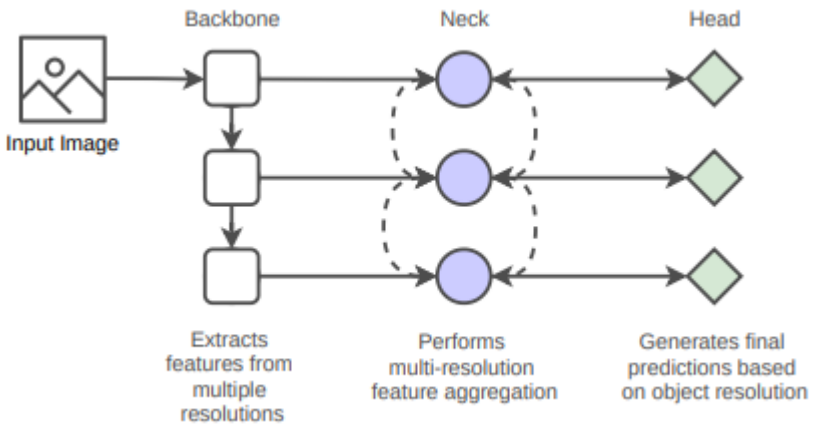


Figure 19: Architecture d'un modèle de détection d'objets typique (Kateb et al., 2021)

un détecteur d'objets à temps réel dépend du temps d'inférence (Kateb et al., 2021). Il est important de mentionner l'influence de la qualité des caractéristiques extraites par la dorsale (Liang et al., 2022). Cette étude cite plusieurs travaux qui portaient sur la conception de nouvelles dorsales dont le but est d'améliorer la précision des modèles de détection d'objets (Liang et al., 2022). Parmi les architectures proposées, on trouve les dorsales basées sur les CNN et celles basées sur des transformateurs (Vaswani et al., 2023). Ces travaux avaient pour objectif de permettre une extraction des caractéristiques multi-échelles des images, ce qui a mené à une amélioration continue de la précision des modèles de détection d'objets en mettant l'accent sur l'importance de l'extraction de caractéristiques.

2.1.2 Le cou (Neck)

Le cou représente la partie intermédiaire d'un détecteur d'objets. Situé entre la dorsale et la tête, il a pour rôle de collecter les cartes de caractéristiques provenant de différentes profondeurs de la dorsale (Bouraya and Belangour, 2021). Une fois ces cartes récoltées, sa principale fonction est de les fusionner afin de combiner les informations provenant des différents niveaux de résolution des couches de la dorsale. Cette partie est souvent implantée

en utilisant différentes variantes de réseaux pyramidaux (Kateb et al., 2021). Parmi les principales architectures de pyramides de caractéristiques, on trouve:

— Réseau pyramidal de caractéristiques FPN (Feature Pyramid Network) (Lin et al., 2017b).

— Réseau d'agrégation de chemins PAN (Path Aggregation Network) (Liu et al., 2019).

— Réseau pyramidal de caractéristiques bidirectionnel BiFPN (Bi-directional Feature Pyramid Network) (Lin et al., 2017b).

Comme illustré dans la figure 20, le cou est généralement constitué de connexions de type top-down (de haut en bas) et bottom-up (de bas en haut) permettant une meilleure propagation de l'information à travers les différentes échelles du réseau.

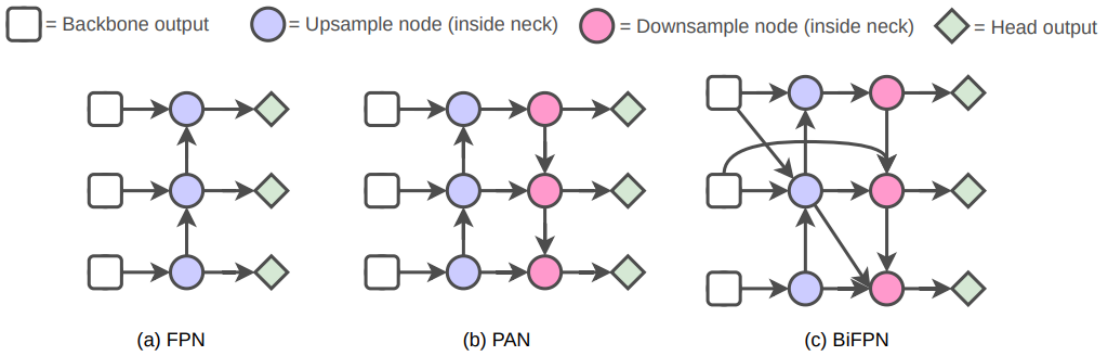


Figure 20: Différentes architectures pyramidales de cou dans un détecteur d'objet (Kateb et al., 2021)

Le but d'une telle agrégation de caractéristiques est de permettre une interaction plus directe entre les caractéristiques de bas niveau et les caractéristiques de haut niveau à travers plusieurs couches.

2.1.3 La tête (Head)

La tête d'un détecteur d'objets constitue le dernier élément du modèle. Elle a pour fonction de prédire les boîtes englobantes (bounding boxes) ainsi que les classes associées aux objets détectés. Selon l'architecture du détecteur d'objets, la prédiction peut être:

— Dense (Dense prediction) pour les modèles de détection à une étape comme YOLO (Redmon et al., 2016), SSD (Liu et al., 2016) ou CenterNet (Duan et al., 2019)

— Clairsemée (Sparse prediction) pour les modèles de détection à deux étapes comme Mask R-CNN (He et al., 2018) et Faster R-CNN (Ren et al., 2017) (Bouraya and Belangour, 2021)

Ces différents modèles seront abordés dans la prochaine section. On peut retrouver plusieurs types de têtes dans un même détecteur d'objets afin de détecter avec précision des objets de différentes résolutions. L'architecture complète d'un modèle de détection d'objet est représentée dans la figure 21.

2.2 Métriques d'évaluation

Dans le domaine de la détection d'objets, les métriques d'évaluation constituent un outil essentiel permettant de mesurer les performances d'un modèle de détection. Dans cette section, nous allons présenter les métriques d'évaluation les plus couramment utilisées. Des notions de bases communes de ces métriques seront introduites afin de mieux expliquer le fonctionnement de chacune d'elles. Ces concepts reposent sur la comparaison entre les valeurs prédites et les valeurs réelles de sorties (Terven et al., 2025; Padilla et al., 2020).

— **Vrai positif** (True Positive): une prédiction est considérée comme un vrai positif lorsque la classe estimée correspond à la classe réelle et que la boîte englobante prédite recouvre suffisamment la boîte réelle, selon un seuil d'IoU (Intersection over Union) prédéfini. En détection d'objets, cette évaluation repose donc directement sur la valeur de l'IoU.

— **Faux positif** (False Positive): un faux positif survient quand le modèle prédit l'existence d'un objet sur l'image alors que l'objet n'est pas réellement présent.

— **Faux Négatif** (False Negative): un faux négatif apparaît lorsque le modèle n'arrive pas à prédire l'existence d'un objet pourtant bien présent dans l'image.

— **Vrai Négatif** (True Negative): le modèle détecte correctement l'absence d'un objet présent sur l'image.

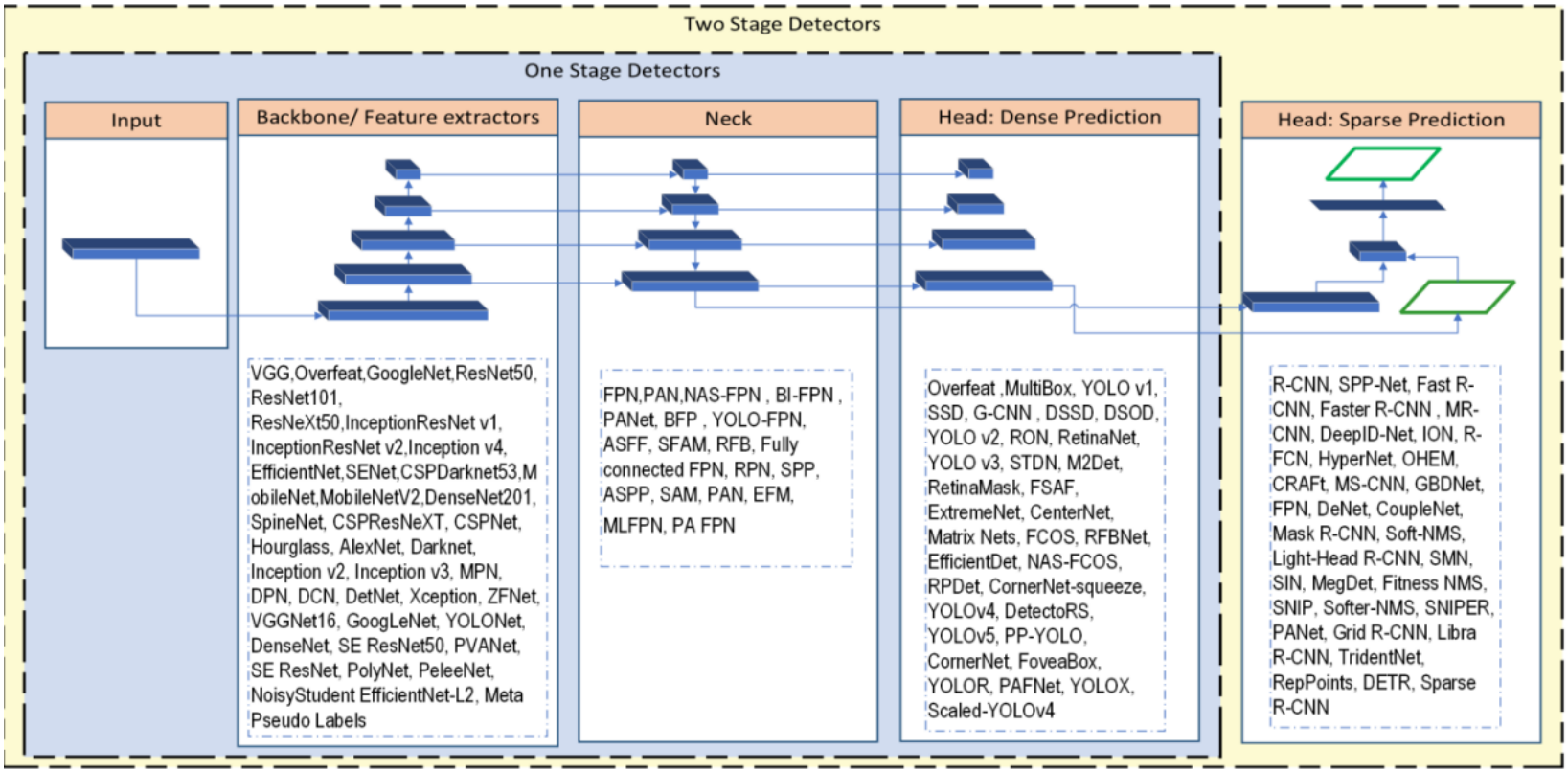


Figure 21: Exemple d'architecture d'un modèle de détection d'objets (Bouraya and Belangour, 2021)

Il est important de noter que, dans le contexte de la détection d'objets, les vrais négatifs représentent théoriquement les boîtes englobantes non présentes dans l'image. Par conséquent, cette métrique n'est pas prise en considération, étant donné qu'il existe une infinité de boîtes englobantes possibles (Padilla et al., 2020). La table 2 présente une matrice de confusion pour une problématique à deux classes.

Table 2: Matrice de confusion

		Classes prédites	
		Classe positive	**Classe négative**
Classes réelles	**Positive**	Vrai positif (TP)	Faux négatif (FN)
	Négative	Faux positif (FP)	Vrai négatif (TN)

2.2.1 Précision

La précision mesure la proportion de prédictions positives qui sont correctes. En d'autres termes, elle indique, parmi toutes les détections réalisées par le modèle, lesquelles correspondent réellement à un objet de la bonne classe. Elle est définie par:

$$\text{Précision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.1}$$

où TP désigne les vrais positifs et FP les faux positifs.

Dans le contexte de la détection d'objets, nous ne tenons pas compte des valeurs Vraies Négatives, car il est impossible de recenser toutes les zones « vides » correctement ignorées par le modèle.

La précision est particulièrement importante dans les applications où un faux positif peut avoir un coût élevé, comme en médecine (diagnostic de maladies), en détection de fraude ou en sécurité.

Dans un problème multi-classe, la précision peut être calculée de deux manières (Terven et al., 2025; Google, 2025):

— **Précision macro-moyenne**: Calculée séparément pour chaque classe puis moyennée.

— **Précision micro-moyenne**: Calculée en regroupant tous les vrais positifs et faux positifs de l'ensemble des classes avant d'appliquer la formule

2.2.2 Rappel

Le rappel, appelé aussi taux de vrais positifs TPR (True Positive Rate), mesure la proportion de prédictions positives correctes par rapport à l'ensemble des cas réellement positifs. Il est défini par la formule suivante :

$$\text{Rappel} = TPR = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.2}$$

où TP désigne les vrais positifs et FN les faux négatifs. Les faux négatifs sont les cas positifs que le modèle n'a pas détectés correctement.

Un rappel élevé indique que le modèle est capable d'identifier la majorité des cas positifs. Cette métrique est essentielle dans l'évaluation des applications critiques comme en médecine, par exemple, où manquer un cas positif peut avoir des conséquences sévères (Terven et al., 2025; Google, 2025).

Dans une classification multi-classes, on distingue deux types de rappels :

— **Rappel macro-moyen** : calculé séparément pour chaque classe puis moyenné.

— **Rappel micro-moyen** : calculé globalement sur l'ensemble des classes.

2.2.3 Compromis entre Précision et Rappel

Dans des situations réelles de classification, il est nécessaire de trouver un compromis entre la précision et le rappel. Ces deux métriques sont généralement inversement proportionnelles; améliorer l'une peut engendrer une dégradation de l'autre. Dépendamment du contexte de l'application, il est essentiel de tenir compte de cette relation. En effet, un faible score de rappel signifie que la valeur de seuil de détection est élevée, ce qui réduit le nombre de valeurs faux positifs et par conséquent, entraîne une précision élevée. Cependant, dans certaines applications comme la détection de fraude, une très haute précision peut mener à ce que des cas positifs, mais de très faibles probabilités soient ignorés par le modèle et classés à tort négatifs. Par conséquent, les tentatives de fraude pas très évidentes ne sont pas détectées par le modèle. À l'inverse, si le rappel est très élevé, c'est-à-dire que la précision est faible, le modèle pourrait identifier à tort des cas innocents comme frauduleux.

On peut donc conclure qu'un bon détecteur d'objets doit avoir un rappel qui augmente progressivement pendant l'entraînement, tout en maintenant une précision très élevée (Terven et al., 2025; Padilla et al., 2020).

Il est également possible de générer un graphique illustrant le compromis entre la précision et le rappel pour différents seuils de décision. Comme expliqué précédemment, un modèle de détection d'objets idéal présente à la fois une précision élevée et un rappel important. La figure 22 illustre différents cas d'entraînement d'un modèle sur une base de données, évaluée avec la métrique Précision vs Rappel. Plus le modèle tend vers le coin supérieur à droite, plus ses performances se rapprochent de l'idéal. Par conséquent, quand l'aire sous la courbe, appelée AUC (Area under Curve), est élevée, meilleur est le modèle en termes d'équilibre entre la précision et le rappel. À l'inverse, si le rappel diminue significativement tandis que la précision reste élevée et stable, le modèle est considéré comme peu performant. Nous n'allons pas aborder l'approche pour l'aire sous la courbe AUC (AUC-ROC), car la courbe de caractéristiques de fonctionnement du récepteur ROC (Receiver Operating Char-

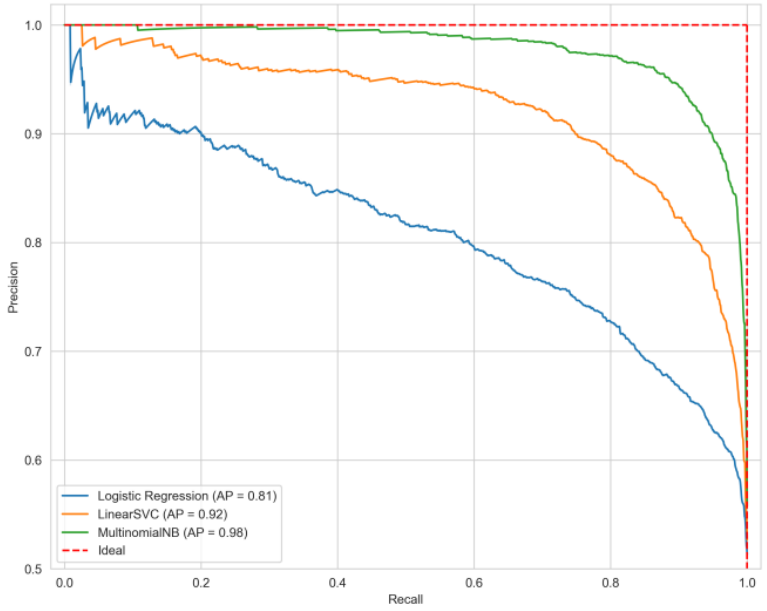


Figure 22: Évaluation de 3 modèles de détection d'objets entraînés sur la même base de données selon la courbe PR (Precision-Recall)

acteristic) se base principalement sur la prise en compte des vrais négatifs (TN), ce qui est moins pertinent dans le contexte de la détection d'objets, où l'accent est principalement mis sur les cas positifs (Kaur and Singh, 2023; Terven et al., 2025).

2.2.3.1 Score F1

Le score F1 est une métrique d'évaluation largement utilisée en apprentissage automatique, notamment pour les problèmes de classification et de détection d'objets. Il correspond à la moyenne harmonique de la précision et du rappel, et permet d'évaluer de manière équilibrée la capacité d'un modèle à produire des prédictions correctes tout en détectant un maximum d'instances pertinentes. Le score F1 est particulièrement pertinent dans les contextes où les classes sont déséquilibrées ou lorsque les faux positifs et les faux négatifs en-

traînent des coûts comparables (Terven et al., 2025). Il est défini par l'équation suivante:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.3}$$

2.2.4 Précision moyenne (AP) et précision moyenne généralisée (mAP)

En détection d'objets, on doit évaluer la classification ainsi que la localisation des objets. La métrique de précision moyenne AP (Average Precision) calcule la précision moyenne de chaque classe individuellement. Par la suite, la moyenne de ces valeurs donne la précision moyenne des moyennes mAP (mean Average Precision) utilisée comme indicateur globale de performance de toutes les classes. La précision moyenne AP est calculée à partir de plusieurs autres métriques, à savoir: IoU, la matrice de confusion, la précision et le rappel (Padilla et al., 2020). La précision moyenne (AP) peut donc être obtenue en utilisant deux approches principales :

1. **Interpolation sur 11 points AP11**

   L'AP11 (Average Precision sur 11 points) est une méthode classique utilisée pour calculer la précision moyenne AP en échantillonnant la courbe Précision–Rappel à 11 niveaux de rappel fixes, allant de 0 à 1.

$$\tilde{R} \in \{0.0,\ 0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.7,\ 0.8,\ 0.9,\ 1.0\} \tag{2.4}$$

   Pour chaque niveau de rappel $\tilde{R}$, on ne prend pas directement la précision mesurée. On procède plutôt par une interpolation maximale, c'est-à-dire qu'on prend la plus grande précision obtenue pour tout rappel R_0 supérieur ou égal à R :

$$P_{\text{interp}}(\tilde{R}) = \max_{R_0 \geq \tilde{R}} P(R_0). \tag{2.5}$$

Une fois les précisions interpolées pour chaque niveau de rappel, on calcule la moyenne:

$$AP_{11} = \frac{1}{11} \sum_{\tilde{R} \in \{0, 0.1, 0.2, \ldots, 1.0\}} P_{\text{interp}}(\tilde{R}). \tag{2.6}$$

où $P_{\text{interp}}(\tilde{R})$ est la précision interpolée au niveau de rappel $\tilde{R}$.

2.2.5 Rappel Moyen (AR)

Le Rappel Moyen AR (Average Recall) est une métrique largement utilisée pour l'évaluation des modèles de détection d'objets, notamment la base de données COCO. Pour chaque seuil d'IOU, on calcule le rappel pour l'ensemble des objets. Ensuite, on effectue la moyenne des Rappels obtenus sur tous les seuils considérés. Ces étapes se répètent un bon nombre de fois afin d'augmenter le rappel, mais risquent également d'augmenter le nombre de faux positifs (Terven et al., 2025). La formule du rappel moyen est définie comme suit:

$$\text{AR} = \frac{1}{N} \sum_{i=1}^{N} \text{Recall}(\text{IoU}_i), \tag{2.7}$$

avec N étant le nombre total de seuils d'IoU considérés (généralement entre 0.5 et 0.95 avec un pas de 0.05) et $R(\text{IoU}_i)$ étant le Rappel calculé au i-ième seuil d'IoU.

2.3 Modèles à une passe

Les modèles de détection d'objets à une étape (One-stage detectors) sont des modèles qui effectuent la détection en une seule passe du réseau, c'est-à-dire prédire directement les classes et les boîtes englobantes sur l'image entière, sans générer de régions candidates. Parmi les modèles de détection d'objets populaires, on retrouve YOLO, SSD, RetinaNet.

Les modèles à deux étapes (Two-stage detectors) divisent la détection en deux phases.

D'abord, ils génèrent des propositions de régions d'intérêt, puis ils classifient ces régions et affinent la localisation des boîtes englobantes avec une meilleure précision.

2.3.1 Modèle à une passe (single stage detector)

Dans l'état de l'art, les approches traditionnelles utilisées dans les modèles de détection d'objet consistaient à formuler des hypothèses sur les boîtes englobantes, re-échantillonner les pixels et les caractéristiques de chaque boîte et enfin appliquer un classificateur de haute qualité. Bien que ces modèles comme Faster R-CNN, soient précis, leur calcul était très intensif et lent à exécuter. Par conséquent, il n'était pas très convenable de les utiliser dans les applications temps réel, surtout dans les systèmes embarqués (Liu et al., 2016).

Pour pallier à ces limitations, Liu et al. (2016) ont proposé un modèle de détection qui n'effectue pas de proposition de boîtes englobantes ni le ré-échantillonnage des pixels ou des caractéristiques. Ils ont également introduit des filtres et des couches de prédiction à différentes échelles. Ces améliorations ont permi d'atteindre à une précision moyenne de 74.3% et un taux de rafraîchissement de 59 IPS (Images par seconde) sur la base de données PASCAL VOC comparé à une précision moyenne de 73.2 % et un taux de rafraîchissement de 7 IPS pour Faster R-CNN.

2.3.1.1 Single Shot MultiBox Detector (SSD)

Single shot Multibox detector est un modèle de détection d'objets temps réel, conçu en 2016 par Liu et al. (2016), qui appartient à la catégorie des modèles de détection à une étape (One Stage detector). L'architecture de ce modèle est représentée dans la figure 23:

Ce modèle prend en entrée des images de dimensions 300x300 ou 512x512 et utilise VGG-16 ou ResNet comme dorsale (Backbone) pour l'extraction des caractéristiques.

Pour la génération des boîtes englobantes (voir figure 24), le modèle SSD utilise une

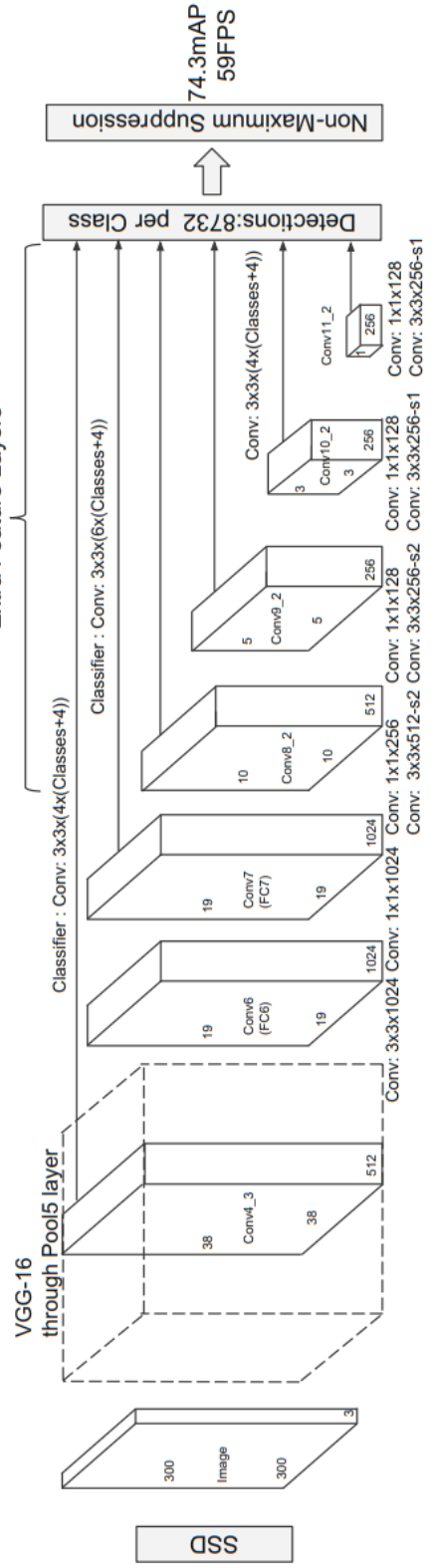


Figure 23: Architecture d'un modèle de détection à une étape (One Stage Detector) (Liu et al., 2016)

approche basée sur les boîtes d'ancrage (anchor boxes). Cette méthode consiste à placer, à chaque pixel de la carte de caractéristiques, plusieurs boîtes prédéfinies (ou boîtes par défaut) de tailles et de rapports d'aspect différents qui servent de point de départ pour la prédiction. Le modèle apprend ensuite à prédire un décalage (offset) permettant d'ajuster la taille de ces boîtes d'ancrage pour mieux les adapter à la taille de l'objet présent sur l'image. L'idée derrière cet algorithme est d'éviter de prédire les boîtes englobantes à partir de zéro (Liu et al., 2016). Le modèle utilise les cartes de caractéristiques issues de plusieurs couches présentes dans le réseau. Ceci lui permet de détecter des objets de différentes tailles, qu'ils soient petits ou grands.

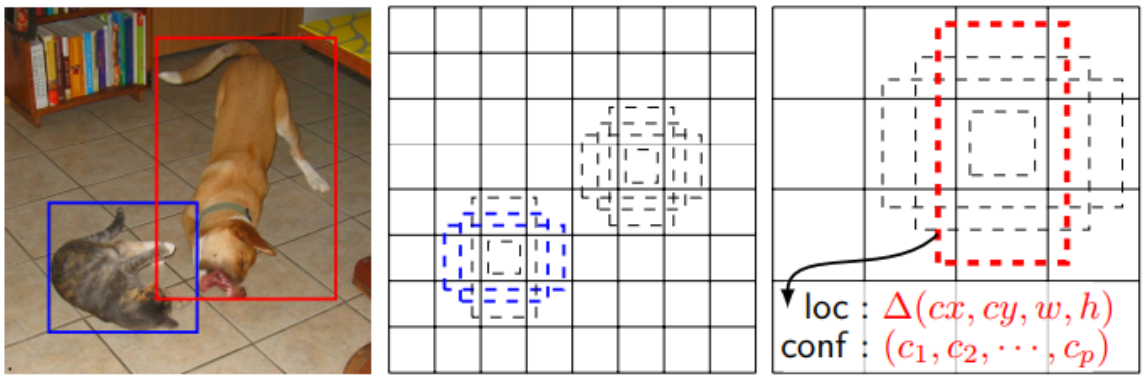


Figure 24: Génération des boîtes englobantes de différentes tailles avec un modèle de détection d'objets à une étape (Liu et al., 2016)

Afin d'éviter d'avoir plusieurs boîtes autour d'un même objet, le modèle SSD utilise une technique de post-traitement appelée Non-Max Suppression (NMS), permettant de retirer les boîtes englobantes redondantes. Les boîtes englobantes redondantes peuvent être soit de faibles probabilités de détection, soit de multiples boîtes qui se chevauchent. La technique NMS procède en ordonnant toutes les boîtes englobantes en fonction de leur probabilité (score) de détection. La boîte ayant la probabilité la plus élevée est sélectionnée comme étant la meilleure boîte. Pour le reste des boîtes, un calcul de chevauchement IoU (Intersection

over Union) est réalisé entre la meilleure boîte et chacune des autres boîtes.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{|A \cap B|}{|A \cup B|} \tag{2.8}$$

Si cette valeur, comprise entre 0 et 1, dépasse un seuil prédéfini, la boîte est supprimée car elle est considérée comme redondante. Ce processus se répète jusqu'à ce que toutes les boîtes redondantes soient éliminées, car elles sont considérées comme des doublons.

Dans (Yamaguchi et al., 2022), un modèle SSD a été conçu pour la détection automatique du carcinome du sein dans les micrographies histopathologiques. Le modèle a atteint une exactitude de 88.5% sur une base de données de 3 classes. Dans (Cai et al., 2022), un modèle SSD, présent dans la figure 25, a été utilisé afin de classer des abeilles à l'aide d'une base de données créée manuellement en intégrant plusieurs techniques d'augmentation de données.

Ce modèle présente une bonne précision dans différentes applications. Néanmoins, il est important de mentionner les limitations liées à l'utilisation de ce modèle de détection. En effet, un problème majeur réside dans la détection d'objets de petite taille, qui dépend de la carte de caractéristiques générée par le réseau. En effet, lorsqu'une image est de petite taille (coarse spatial resolution), chaque pixel représente une grande portion de l'image, ce qui complique l'extraction de caractéristiques pertinentes. En revanche, pour les images de grande taille, chaque pixel représente une petite zone de l'image, facilitant ainsi l'extraction de caractéristiques des détails fins.

Pour les réseaux de neurones convolutifs CNN, à cause des opérations de pooling mentionnées dans le chapitre précédent, les cartes de caractéristiques ont tendance à devenir de plus en plus petites à mesure que la profondeur du réseau augmente (en raison de la présence de plusieurs couches de pooling). Par conséquent, les petits objets présents dans ces images de faible résolution deviennent de plus en plus difficiles à détecter ou à classifier. Certaines techniques permettent toutefois de remédier à ce problème. Dans (Liu et al., 2016), les au-

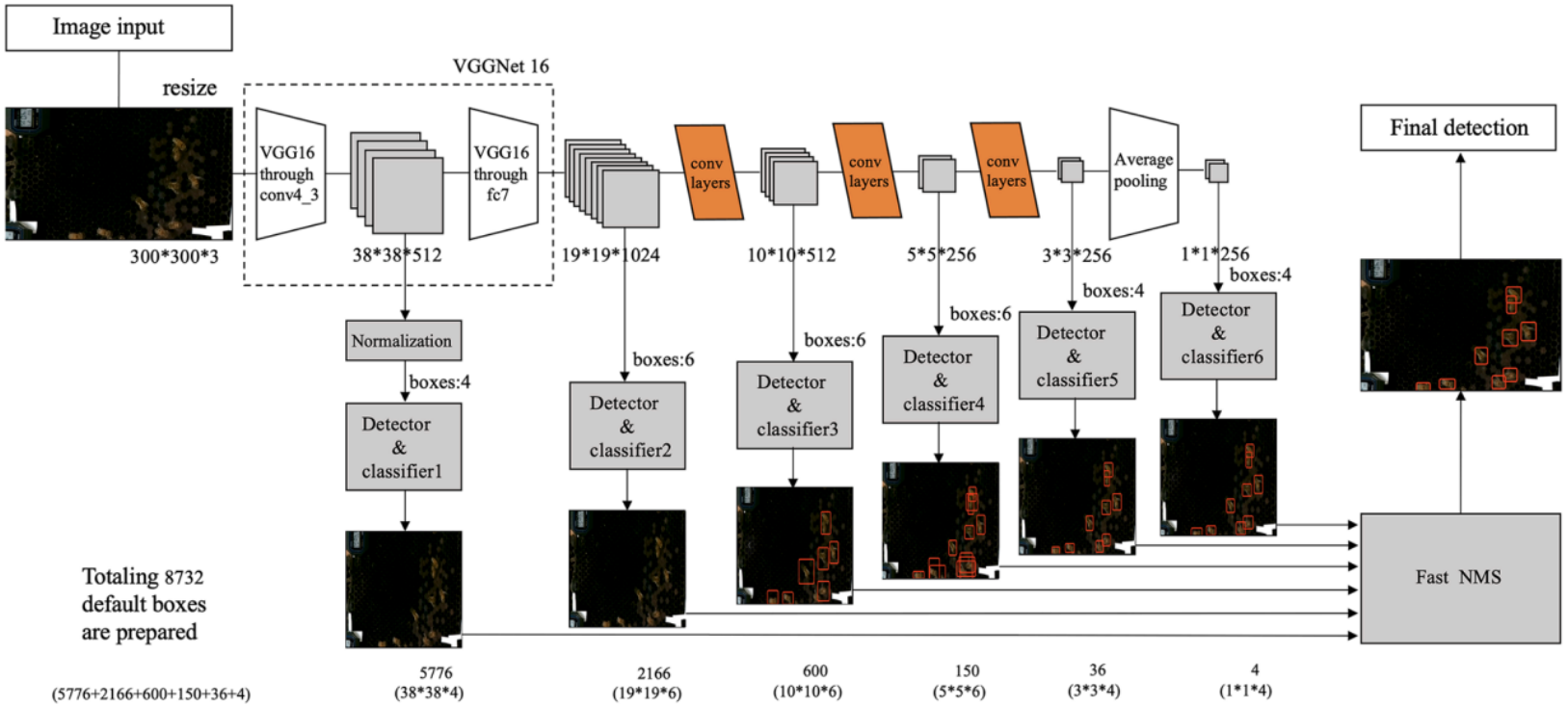


Figure 25: Modèle de détection d'objets à une étape utilisé pour la détection d'abeilles (Cai et al., 2022)

teurs ont appliqué plusieurs techniques d'augmentation de données sur différentes bases de données populaires, telles que VOC2007, PASCAL VOC2012 et COCO. Ces techniques ont contribué à une amélioration de la précision moyenne (mAP) d'environ 3%. Néanmoins, le modèle reste inadéquat pour les images contenant plusieurs objets de petite taille.

2.3.1.2 You Only look Once (YOLO)

You Only Look Once (YOLO), tel qu'illustré dans la figure 26, est un modèle de détection d'objets qui a été proposé pour la première fois en 2016 par Redmon et al. (2016) et présenté lors de la conférence IEEE CVPR. Étant également un détecteur à un seul étage (single stage detector), YOLO effectue la détection en une seule étape, ce qui le rend particulièrement rapide.

L'approche utilisée pour la détection d'objets est basée sur une grille. En effet, l'image est divisée en une grille de cellules, et chaque cellule a pour rôle de prédire si un objet est contenu dans celle-ci. Lorsqu'un objet est en effet présent dans la cellule, cette dernière donnerait en sortie l'emplacement, la taille de l'objet ainsi qu'un score de classe, c'est-à-dire la probabilité que l'objet appartienne à une classe bien spécifique. Plusieurs boîtes d'ancrage (Bounding boxes) sont également générées et ajustées. Ces boîtes sont évaluées durant le processus de prédiction afin de mieux les faire correspondre aux objets détectés (Redmon et al., 2016).

Ce modèle YOLO excelle particulièrement par sa vitesse. En effet, il peut traiter jusqu'à 45 images par seconde (FPS), tandis que sa version rapide, Fast YOLO, atteint une vitesse de 155 FPS. Le fait de faire la détection en une seule étape rend YOLO facile à optimiser et à entraîner, comparé à d'autres modèles de détection d'objets plus complexes. Le tableau 2 présente les résultats de la comparaison de YOLO avec différents modèles de détection d'objets.

YOLO présente cependant certaines limitations. En effet, comme tout modèle SSD,

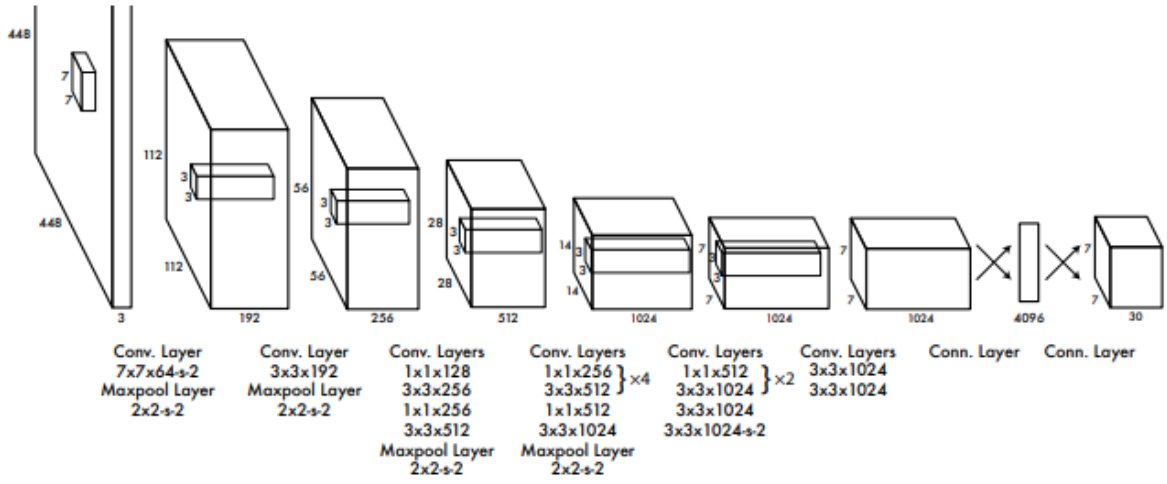


Figure 26: Architecture du modèle de détection d'objets YOLO (Redmon et al., 2016)

Table 3: Comparison entre YOLO et les autres modèles de détection d'objets (Redmon et al., 2016).

Modèle	mAP (VOC 2007)	FPS (Frames Per Second)	Key Features	Real-Time Capability
YOLO	63.4%	45 FPS	Single-stage detector, grid-based approach	Yes
Fast YOLO	52.7%	155 FPS	Faster variant of YOLO with reduced layers	Yes
Faster R-CNN	73.2%	7 FPS	Two-stage detector, region proposals	No
R-CNN	66.0%	0.5 FPS	Region proposals, high accuracy	No
DPM	33.7%	0.07 FPS	Deformable part models	No

YOLO a des difficultés avec les objets de petite taille. Ceci est dû à la limitation de l'approche basée sur une grille, qui est peu adaptée à la détection des objets de petite taille. YOLO éprouve également des difficultés lorsque plusieurs objets sont présents dans une même zone de l'image, étant donné que chaque cellule de la grille n'est capable de détecter qu'un seul objet. Par conséquent, lorsque plusieurs petits objets se retrouvent dans la même cellule, certains d'entre eux ne peuvent pas être détectés.

2.3.1.3 Évolution du modèle YOLO

1. **YOLOv2**

   En 2017, Redmon and Farhadi, (2017) ont introduit une nouvelle version de YOLO, appelée YOLOv9000 ou YOLOv2. Ce nouveau modèle est capable de détecter jusqu'à 9000 catégories d'objets également appelés classes. Comparé à la première version de YOLO, YOLOv2 présente différentes techniques introduites dans son architecture qui ont amélioré ses performances. Parmi celles-ci, on trouve la normalisation par lots (Batch normalization), appliquée après chaque couche de convolution pour normaliser chaque entrée de la couche suivante. Le but est de stabiliser et d'accélérer le processus d'entraînement du modèle, de façon globale, sans rajouter des calculs supplémentaires.

   YOLOv2 adopte la même approche que le modèle Faster R-CNN pour la génération des boîtes englobantes, en ajustant des boîtes prédéfinies au lieu de générer des boîtes englobantes à partir de zéro. Cette approche est particulièrement efficace pour la détection d'objets de petite taille, l'un des points faible de la première version de YOLO.

   Afin d'améliorer la résolution de la carte de caractéristiques, YOLOv2 utilise un classifieur de meilleure résolution (448x448) au lieu de 224x224. Par conséquent, le modèle obtenait davantages de détails visuels sur les objets présents dans l'image, ce qui se traduit par une amélioration de la précision de détection, avec un gain de +4% en mAP (mean Average Precision).

   Pendant l'entraînement, la dimension de l'image d'entrée et la taille du réseau sont modifiés toutes les 10 itérations. Cela permet au même réseau de réaliser des détections à différentes résolutions. YOLOv2 est ainsi capable de détecter des objets sur des images de dimension qui varient de 288x288 à 544x544, tout en conservant une très bonne précision en temps réel (Redmon and Farhadi, 2017).

   On peut dire que YOLO9000 est plus performant que la première version de YOLO.

En effet, cette nouvelle version est capable de détecter jusqu'à 9000 catégories d'objets tout en maintenant une très bonne vitesse et une précision élevée. En comparaison, le modèle YOLO original ne détecte que 20 catégories, correspondant à celles de la base de données PASCAL VOC. De plus, YOLO9000 opère avec une fréquence de 67 images par seconde (IPS) sur le jeu de données PASCAL VOC avec une précision moyenne (mAP) de 76.8%.

Table 4: Performances de YOLO9000 sur deux bases de données (Redmon et al., 2016)

Base de données	YOLO9000 mAP	YOLO mAP	YOLO9000 FPS	Principales différences
VOC 2007	76.8%	63.4%	67 FPS	Résolution plus élevée, boîtes d'ancrage, batch norm
COCO	Non spécifié	N/A	40+ FPS	Détection de 9000 classes

2. **YOLOv3**

En 2018, une nouvelle version de YOLO, appelée YOLOv3, a été introduite. Ce nouveau modèle présente plusieurs améliorations comparé à ses prédécesseurs. Parmi ces mises à jour, on peut citer l'utilisation d'une nouvelle dorsale, Darknet-53 (Toğaçar, 2022). Ce backbone Darknet-53 est plus robuste que Darknet-19 (utilisé dans YOLOv2), car il possède 53 couches de convolution avec des connexions résiduelles, ce qui le rend ainsi très efficace pour l'extraction de caractéristiques (Redmon and Farhadi, 2018).

Un autre point important est que YOLOv3 est capable de générer des prédictions à 3 différentes échelles lui permettant ainsi de détecter des objets de différentes tailles. Les versions précédentes de YOLO étaient peu performantes avec les objets de petites tailles. Cependant, malgré les améliorations rapportées face aux objets de petites tailles, YOLOv3 connaît une légère baisse de performance relativement aux objets de taille moyenne ou grande (Redmon and Farhadi, 2018).

Au niveau de la classification, YOLOv3 n'utilise plus la fonction d'activation softmax. Ceci est dû au fait que cette fonction n'est pas très adaptée aux cas multilabel, c'est-à-dire lorsqu'une image ou une boîte englobante peut contenir plusieurs étiquettes (par

exemple oiseau et animal). En effet, la fonction Softmax suppose que chaque boîte englobante renferme un seul objet, ce qui limite sa capacité dans un contexte où plusieurs peuvent co-exister dans la même zone. À la place, les auteurs ont utilisé des classifieurs logistiques indépendants, plus performants dans le cas où il y a présence de plusieurs classes dans une seule boîte englobante (Redmon and Farhadi, 2018).

Ces changements font de YOLOv3 un modèle à la fois plus rapide et plus précis que ses prédécesseurs. En effet, comme illustré dans la figure 27, YOLOv3 affiche de meilleures performances que les variantes de SSD, et reste comparable à plusieurs modèles d'état de l'art, tout en étant plus rapide que tous les autres modèles qui l'ont précèdé (Redmon and Farhadi, 2018).

3. **YOLOv4**

La quatrième version de YOLO, appelé YOLOv4, a été conçue en 2020. En se basant sur l'hypothèse selon laquelle la dorsale (backbone) d'un modèle devrait être sélectionnée en fonction de la taille du champ réceptif. Pour cela, le modèle utilise CSPDarknet53 comme dorsale qui a démontré de meilleures performances que CSPResNext50 et EfficientNet-B3 pour la détection d'objets (Bochkovskiy et al., 2020).

Des travaux de pré-traitements de données ont été réalisés afin d'améliorer les performances du modèle. Ces techniques d'augmentation consistent à faire des ajustements précis au niveau des pixels, tout en préservant les informations contenues dans celui-ci. Les techniques qui ont été utilisées sont:

— **CutMix** (Yun et al., 2019): Cette technique sert à recouvrir l'image recadrée par une région recangulaire d'une autre image tout en ajustant les étiquettes correspondantes (Bochkovskiy et al., 2020).

— **MixUp** (Zhang et al., 2017): Cette technique mélange deux images en les superposant selon un certain ratio, en ajustant également les étiquettes (Bochkovskiy et al., 2020).

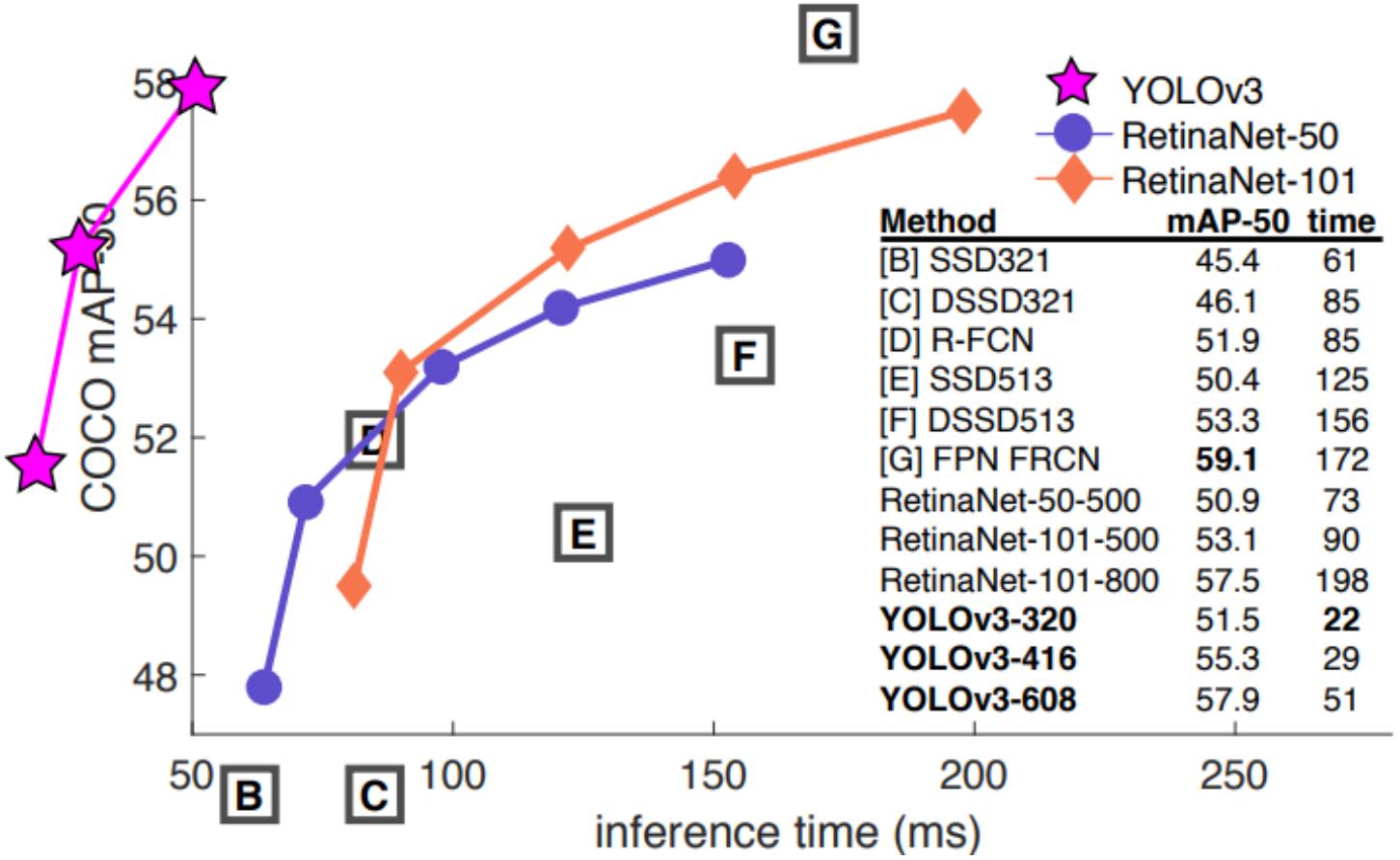


Figure 27: Comparaison du temps d'inférence (ms) entre YOLOv3, RetinaNet-50 et RetinaNet-101 (Redmon and Farhadi, 2018)

— **Cutout** (DeVries and Taylor, 2017): cette technique masque de façon aléatoire une région rectangulaire d'une image en la remplissant avec une valeur aléatoire ou nulle (Bochkovskiy et al., 2020).

Ces méthodes ont pour but d'augmenter la diversité des données d'entraînement sans pour autant augmenter la taille réelle de la base de données. YOLOv4 utilise également PANet (Path Aggregation Network) comme extracteur de caractéristiques. Introduit en 2018 dans les modèles de détection d'objets, PANet permet d'améliorer la précision de détection en combinant les différentes couches du réseau, ce qui permet à YOLOv4 d'offrir de meilleures performances, notamment pour la détection d'objets de petite taille (Bochkovskiy et al., 2020).

Un autre point fort de YOLOv4 est l'utilisation de la fonction d'activation Mish (Misra, 2020), qui permet d'atténuer le problème de disparition du gradient (Vanishing gradient) (Bochkovskiy et al., 2020).

La combinaison de ces techniques d'augmentation, d'architecture de la dorsale et de la fonction d'activation a permis à YOLOv4 d'atteindre des résultats parmi les meilleurs en matière de détection d'objets.

Enfin, YOLOv4 se distingue par son efficacité en termes de ressources et contrairement aux versions précédentes, ne requiert pas l'utilisation de plusieurs GPU pour l'entraînement. Par conséquent, cela rend le modèle plus accessible aux chercheurs ou aux développeurs disposant de ressources matérielles limitées et à améliorer la stabilité et la convergence du modèle (Bochkovskiy et al., 2020).

Les figures 28, 29 et 30 résument une comparaison, en termes de rapidité et de précision, entre YOLOv4 et d'autres modèles de détection d'objets de l'état de l'art, évalués sous différents GPU (Maxwel, Volta et Pascal de Nvidia).

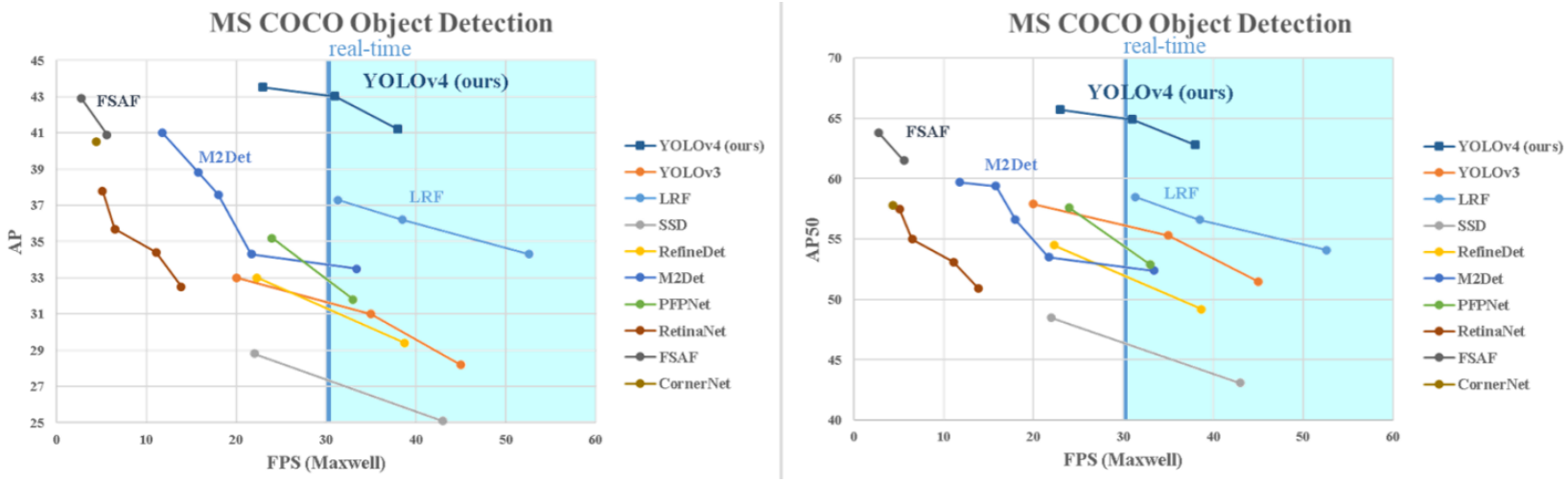


Figure 28: Comparaison de la rapidité et la précision de YOLOv4 avec les autres modèles de détection d'objets (le GPU utilisé est l'architecture Maxwell de Nvidia) (Bochkovskiy et al., 2020)

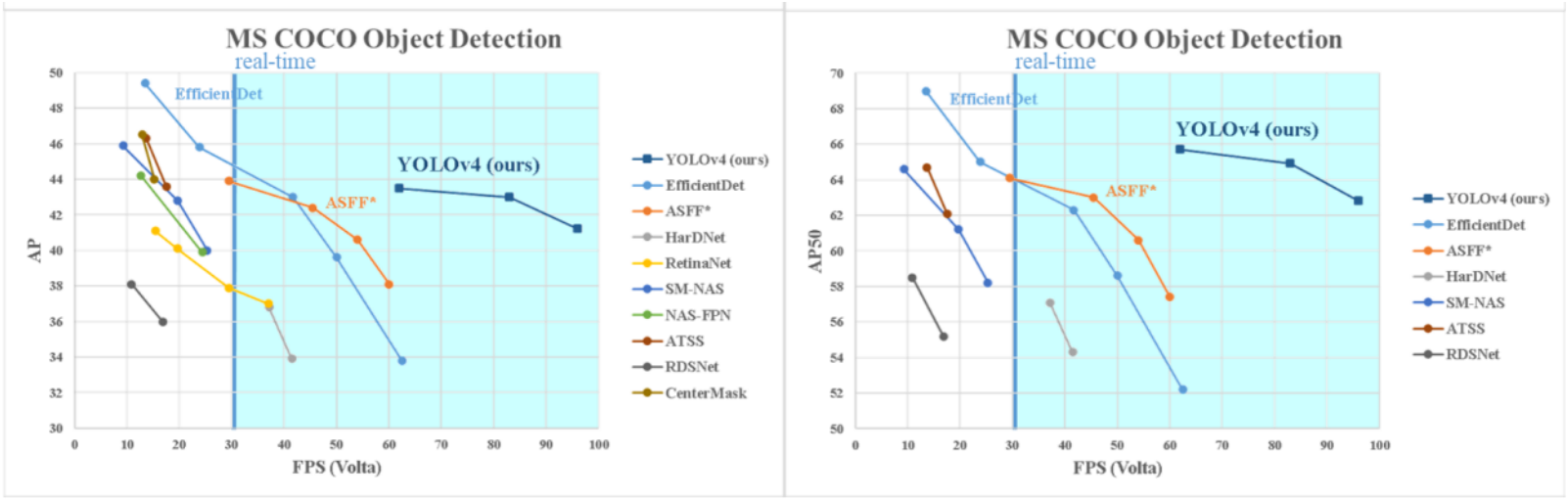


Figure 29: Comparaison de la rapidité et la précision de YOLOv4 avec les autres modèles de détection d'objets (le GPU utilisé est l'architecture Volta de Nvidia (Bochkovskiy et al., 2020)

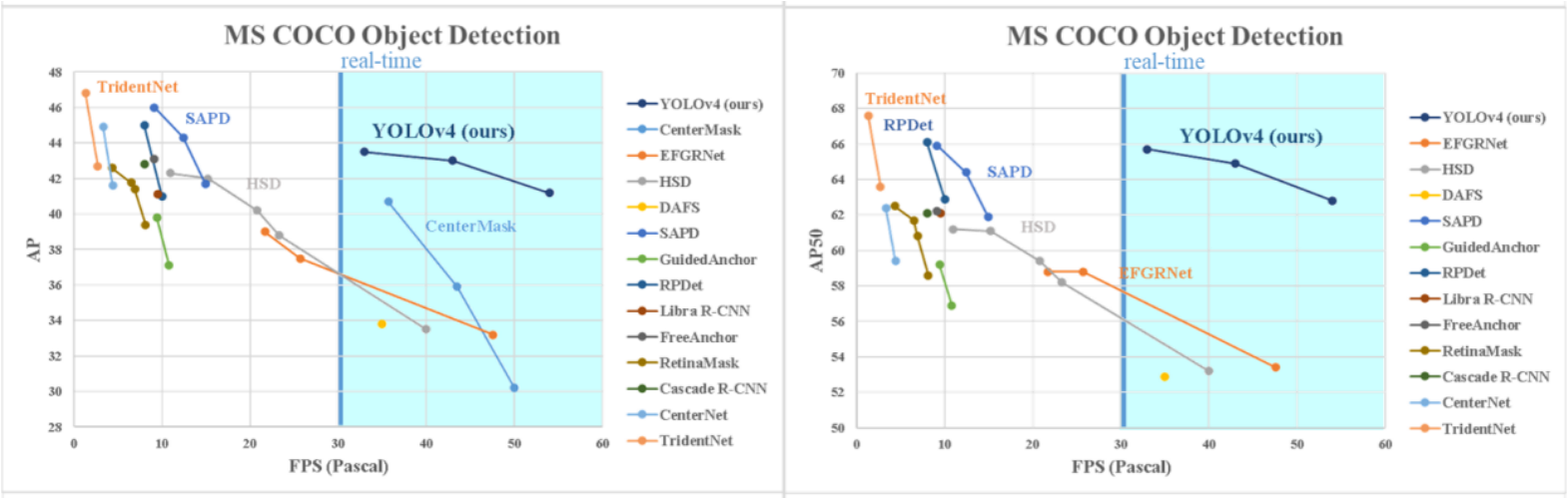


Figure 30: Comparaison de la rapidité et la précision de YOLOv4 avec les autres modèles de détection d'objets (le GPU utilisé est l'architecture Pascal de Nvidia) (Bochkovskiy et al., 2020)

Table 5: Influence des différents paramètres sur les performances du modèle AP_{50} et AP_{75} désignent la précision moyenne (Average Precision) calculée avec des seuils d'IoU respectifs de 0.5 et 0.75) (Bochkovskiy et al., 2020)

Model (with optimal setting)	Size	AP	AP_{50}	AP_{75}
CSPResNeXt50-PANet-SPP	512x512	42.4	64.4	45.9
CSPResNeXt50-PANet-SPP (BoF-backbone)	512x512	42.3	64.3	45.7
CSPResNeXt50-PANet-SPP (BoF-backbone + Mish)	512x512	42.3	64.2	45.8
CSPDarknet53-PANet-SPP (BoF-backbone)	512x512	42.4	64.5	46.0
CSPDarknet53-PANet-SPP (BoF-backbone + Mish)	512x512	43.0	64.9	46.5

4. **YOLOv5**

YOLOv5 est la cinquième version de l'algorithme YOLO, développée par Ultralytics et implantée sous le framework PyTorch. Ce modèle se distingue par sa capacité d'exécution en temps réel ainsi que sa compatibilité avec plusieurs formats d'exportation, tels que TFLite, ONNX, CoreML et TensorRT (Ultralytics, 2022). YOLOv5 est proposé en plusieurs variantes (YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x), chacune optimisée pour un équilibre entre vitesse et précision. Cette version incorpore des améliorations au niveau de l'architecture.

Au niveau de l'entrée, YOLOv5 applique une technique d'augmentation de données appelée mosaïque. Cette technique consiste à fusionner 4 images d'entraînement en une seule afin de créer une base de données plus variée et améliorer la capacité du modèle à détecter les objets de petites tailles.

Comme illulstré dans la figure 31, au niveau de la dorsale (backbone), YOLOv5 se base sur un CSPDarknnet53, une version améliorée de l'architecture Darknet. L'étape d'extraction de caractéristiques comporte 3 composantes importantes:

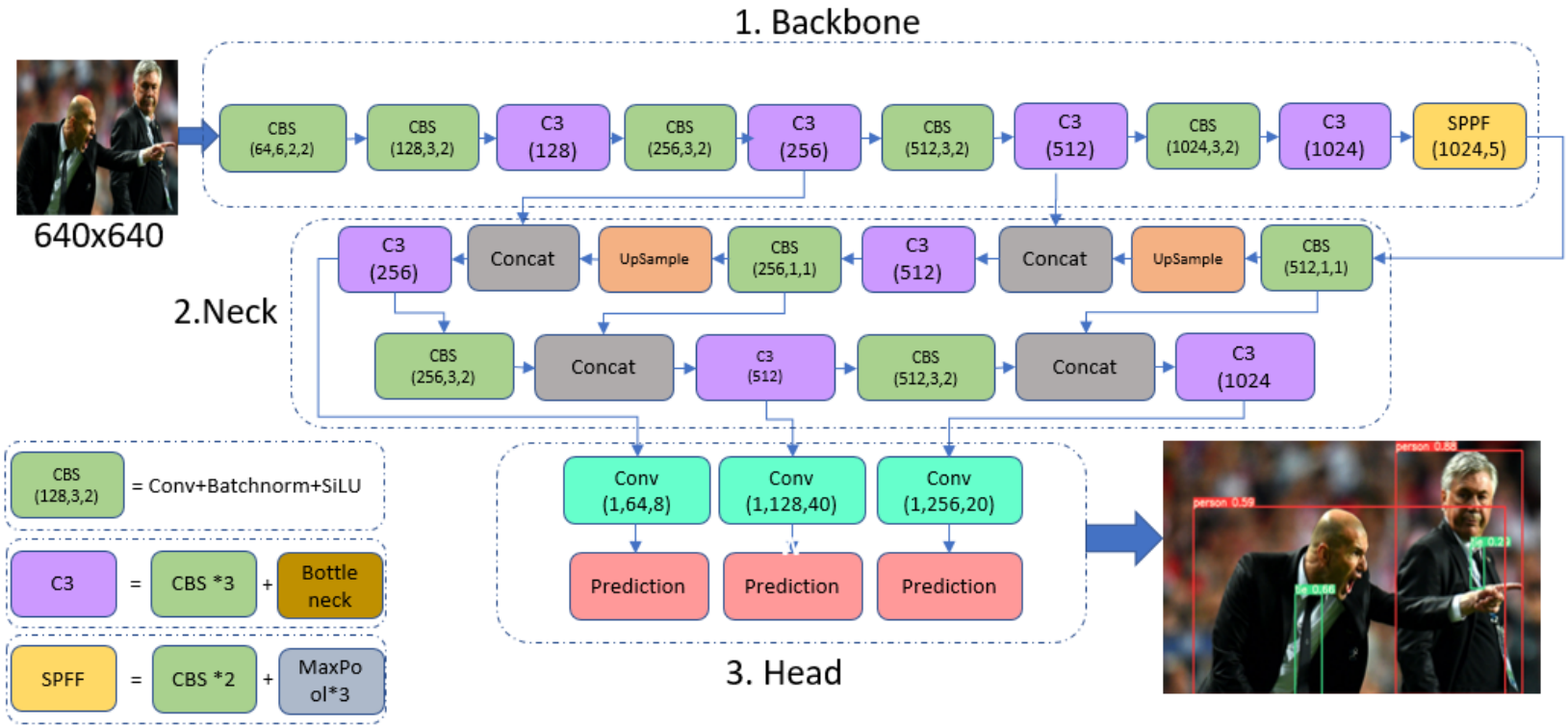


Figure 31: Architecture du modèle de détection d'objets YOLOv5

Table 6: Comparaison des tailles (en millions de paramètres) des différentes architectures/versions du modèle YOLOv5 (Ultralytics, 2022)

Modèle	**Dimension de l'entrée (pixels)**	**Paramètres (Millions)**
YOLOv5n	640x640	1.9
YOLOv5s	640x640	7.2
YOLOv5m	640x640	21.2
YOLOv5l	640x640	64.5
YOLOv5x	640x640	86.7

— **La couche Focus** : cette couche découpe l'image en sous-parties par échantillonnage spacial afin de minimiser la perte d'informations.

— **Le module BottleNeckCSP** (Bochkovskiy et al., 2020): il réduit la charge de calcul tout en extrayant des informations détaillées à partir de la carte de caractéristiques.

— **La couche SPP (Spatial Pyramid Pooling)** (He et al., 2014): elle améliore le champ récepteur du modèle en transformant les entrées de taille variable en des cartes de caractéristiques de taille fixe, permettant ainsi au modèle de récupérer les informations des différentes régions de l'image.

Comme mentionné précédemment, le cou se charge de fusionner les caractéristiques extraites par la dorsale. YOLOv5 utilise une combinaison de deux structures: FPN (Feature pyramid Network) et PANet (Path aggregation network).

— **FPN** (Lin et al., 2017b) agrège les caractéristiques multi-échelles en combinant des caractéristiques à faible résolution mais sémantiquement fortes (provenant des couches profondes) avec des caractéristiques à haute résolution mais sémantiquement faibles (provenant des couches moins profondes). Ceci rend le modèle plus performant face aux objets de petites tailles et de grandes tailles.

— **PANet** (Liu et al., 2019) est une amélioration de FPN qui renforce davantage la capacité du modèle à détecter des objets de différentes tailles, en améliorant la trans-

mission d'informations entre les couches. Il permet de regrouper les informations de haut niveau afin d'améliorer la localisation des objets sur l'image.

Enfin, le dernier composant qui représente la tête (head) du réseau a pour tâche de prédire les boîtes englobantes ainsi que les probabilités des classes détectées. Cette partie du réseau est composée de couches de détection, responsables notamment la génération de cadres d'ancrages et la prédiction des score de détection d'objets. YOLOv5 utilise un classificateur unique, propore à son architecture, capable de produire 3 types de sorties (Khanam and Hussain, 2024):

— Les coordonnées de la boite englobante (bounding box).

— Les score d'objectivité (probabilité qu'un objet soit présent).

— Les probabilités de classe (pour chaque classe possible).

5. **RetinaNet**

RetinaNet est le tout premier modèle à intégrer le réseau FPN (Feature Pyramid Network) au-dessus du réseau de la dorsale. Son fonctionnement repose sur l'utilisation de deux sous-réseaux spécialisés. Durant la convolution, les cartes de caractéristiques générées sont transférées à trois niveaux de mise à l'échelle correspondant à différentes résolutions. Ces cartes de caractéristiques sont par la suite concaténées à chaque niveau de la pyramide pour produire une représentation multi-échelle de l'image de l'entrée.

Les cartes de caractéristiques sont ensuite parcourues par des boîtes d'ancrage afin de générer des propositions d'objets. Ces propositions sont transmises vers les deux sous-réseaux (Lin et al., 2020).

— Le premier qui est chargé de la classification des objets.

— le deuxième gère la régression de la boîte englobante, c'est-à-dire leur localisation précises dans l'image.

La figure 32 représente l'architecture du modèle RetinaNet. Grâce à l'utilisation de FPN et des deux sous-réseaux, le modèle a permis une amélioration significative de

la précision moyenne (mAP) sur la base de données COCO. Cependant, comparé à d'autres modèles de détection d'objets tels que SSD et YOLO, RetinaNet reste très lent durant l'inférence (Lin et al., 2020).

Une contribution majeure introduite avec la conception de RetinaNet est l'utilisation de la fonction de perte *Focal Loss*, conçue pour gérer le déséquilibre entre les classes dans les tâches de détection d'objets. Cette fonction de perte est définie par l'équation 1.4 :

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t) \tag{1.4}$$

où α_t est l'hyperparamètre d'équilibrage de classe, $(1 - p_t)^\gamma$ est le terme focalisation et γ représente le paramètre de focalisation (généralement égal à 2).

Ce terme ajuste dynamiquement la contribution de chaque exemple à la perte en se basant sur la probabilité de prédiction. Par exemple, pour une prédiction correcte avec une probabilité élevée, soit $p_t = 0.99$ et avec $\gamma = 2$, on obtient :

$$(1 - p_t)^\gamma = (1 - 0.99)^2 = 0.01^2 = 0.0001 \tag{2.9}$$

Dans ce cas, la contribution à la perte de cet exemple bien classé est fortement réduite. Cela permet au modèle de moins se concentrer sur les exemples faciles et de focaliser l'apprentissage sur les exemples difficiles ou mal classés (Lin et al., 2020).

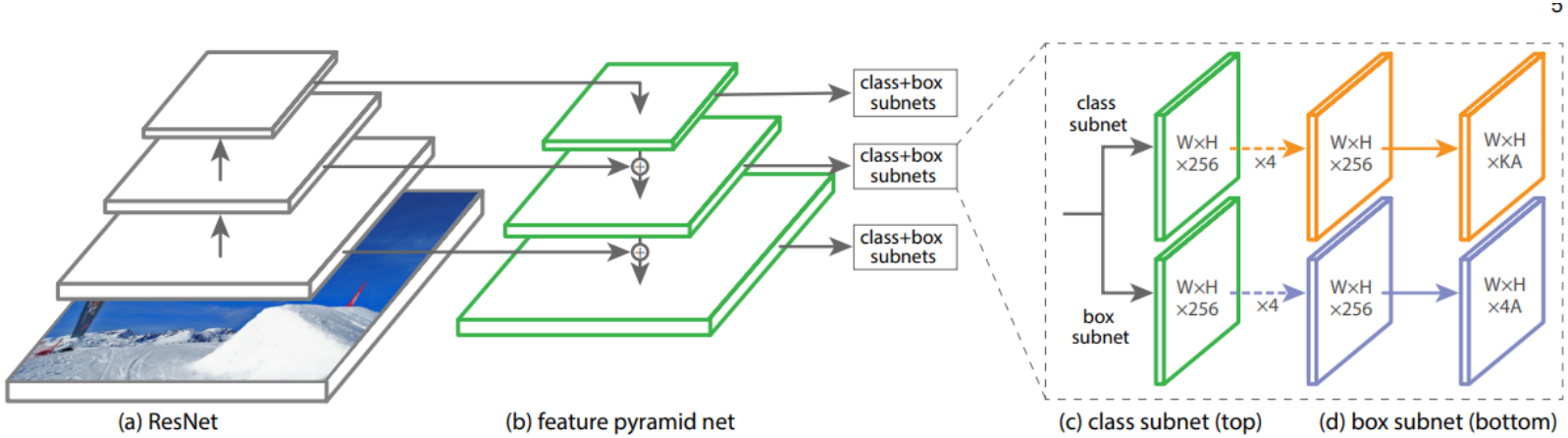


Figure 32: Architecture du modèle de détection d'objet RetinaNet (Lin et al., 2020)

2.3.2 Conclusion

Dans ce chapitre, la structure des modèles de détection d'objets a été présentée en détail. Une attention particulière a été accordée aux rôles de chacun des composants et aux travaux qui ont permis de concevoir ces composants. Une étude bibliographique a été menée sur les métriques d'évaluation afin de mieux apprécier les performances des modèles et savoir choisir celui qui est plus adapté à nos besoins. Nous avons choisi de nous focaliser sur les modèles de détection d'objets à une seule passe (one stage Detectors) pour leur rapidité de traitement. Une panoplie de ces modèles a été présentée pour mieux comprendre les architectures existantes, ainsi que leurs avantages et limites. Le prochain chapitre sera consacré aux techniques d'accélération des modèles d'intelligence artificielle.

CHAPITRE 3

ACCÉLÉRATION MATÉRIELLE DES MODÈLES DE DÉTECTION D'OBJETS

Ce chapitre aborde l'accélération matérielle des modèles de détection d'objets en mettant l'accent sur les différentes techniques d'optimisation des réseaux de neurones. Il détaille en profondeur la quantification des paramètres du modèle et l'élagage des réseaux de neurones. Il examine aussi les différentes plateformes matérielles disponibles pour implanter de tels modèles.

3.1 Introduction

Durant les dernières années, les applications conçues en lien avec la vision par ordinateur exploitent de plus en plus les concepts d'intelligence artificielle (IA) et d'apprentissage automatique ML (Machine Learning). Quel que soit le domaine en question, à savoir la santé, la sécurité des personnes ou l'éducation, l'intelligence artificielle ne cesse de se démontrer comme un outil hors du commun.

Cependant, ces applications nécessitent généralement un traitement massif de données, ce qui signifie qu'il faut impérativement des ressources matérielles et logicielles efficaces (Kalapothas et al., 2022). Les réseaux de neurones convolutifs (CNN) deviennent de plus en plus profonds, c'est-à-dire qu'ils comportent un nombre croissant de couches intermédiaires. Donc, le temps de calcul effectué par ces modèles est impacté négativement. Plusieurs travaux ont démontré que le calcul convolutif est principalement la partie qui nécessite le plus de temps dans le processus des CNN (Li et al., 2021; Wang et al., 2022).

Par conséquent, plusieurs architectures d'accélération des réseaux de neurones CNN ont été proposées afin de réduire le temps de calcul des couches convolutives. Parmi celles-

ci, les cartes graphiques (GPU), les circuits à base de portes programmables sur site (FPGA) et les circuits intégrés à application spécifique (ASIC) sont les accélérateurs matériels les plus populaires.

3.2 Notions sur l'accélération des modèles d'apprentissage profond

Les modèles utilisés en apprentissage profond sont très connus pour avoir des centaines de millions voir des milliards de paramètres à ajuster ou à calculer. Par exemple, Chat-GPT 3 compte environ 175 milliards de paramètres (Brown et al., 2020) et les nouvelles versions chat-gpt 4 et chat-gpt 5 ont une estimation de 1 à 1.8 billion de paramètres et 5 à 10 billion de paramètres respectivement. Une nouvelle tendance en intelligence artificielle, qui a fait son apparition durant les trois dernières années, consiste à recourir à des périphériques de pointe afin d'utiliser des modèles de plus petite taille, mais qui peuvent fonctionner en temps réel. Ces modèles sont généralement employés dans plusieurs domaines, notamment la surveillance médicale, la conduite autonome et la sécurité des personnes (Zhu et al., 2024). Toutefois, leur utilisation présente certains défis, car ils dépendent de plusieurs contraintes importantes (faible latence, mémoire limitée et ressources de calcul restreintes). Par conséquent, il est impératif d'avoir un compromis entre la précision et le temps de calcul. Des travaux ont été menés afin de proposer des techniques d'accélération. Parmi celles-ci, trois sont assez populaires:

— La distillation des connaissances.

— L'élagage de modèle.

— La quantification.

3.3 Techniques de compression de réseaux de neurones

3.3.1 Distillation de connaissances

La distillation des connaissances est une technique en apprentissage automatique qui vise à transférer les connaissances d'un modèle pré-entraîné notamment large et complexe, vers un modèle de plus petite taille. Cette technique fait partie des approches de compression des modèles d'apprentissage profond. L'idée derrière une telle approche est de faire en sorte qu'un modèle de petite taille puisse imiter un modèle plus complexe, tout en facilitant l'interprétation de ce dernier (Bucilă et al., 2006). Dans ce contexte, le modèle complexe et de grande taille sert de maître au modèle élève, qui est plus simple et de petite taille. Durant l'entraînement, un modèle d'apprentissage profond peut générer plusieurs prédictions préliminaires (soft targets) avant d'aboutir à une prédiction finale (hard targets).

En distillation de connaissances, les prédictions préliminaires (soft targets) sont utilisées pour entraîner le modèle élève, ce qui lui permet de fournir des informations plus riches et plus cohérentes que de lui transmettre les prédictions finales (hard targets). Ceci permet au modèle élève d'apprendre en utilisant moins d'exemples et avec un taux d'apprentissage plus élevé.

Les fonctions de perte utilisées dans cette technique sont une fonction de perte standard sur les prédictions finales (hard-targets) et une fonction de perte de distillation sur les prédictions préliminaires (soft-targets). La fonction de perte de distillation mesure la différence de probabilités entre le maître et l'élève.

Il existe différents types de connaissances en distillation.

— **Les connaissances basées sur les réponses**: Il s'agit de la distillation de prédictions individuelles de classes. Ces prédictions représentent des « logits» (Bergmann, 2024)

— **Les connaissances basées sur les caractéristiques**: Cette approche consiste à distiller des fonctions d'activation ou bien les poids présents dans les couches intermédiaires.

— **Les connaissances basées sur les relations**: Il s'agit de distiller les relations entre les différents éléments du réseau de neurones.

On peut procéder à une distillation des connaissances par deux méthodes. La première est une distillation hors ligne (Off-line), où le maître est un modèle pré-entraîné et ses paramètres (poids) sont figés. Pendant la distillation des connaissances, ces paramètres restent fixes afin de ne pas perdre les connaissances acquises grâce au modèle maître. Cette technique se concentre davantage sur l'entraînement du modèle élève et est très courante dans un grand modèle de langage LLM (Large Language Model).

La deuxième méthode est la distillation en ligne (On-line). Durant la distillation de connaissances (KD), les deux modèles (maître et élève) sont entraînés simultanément. Cette méthode est généralement utilisée lorsqu'aucun modèle maître dédié à une tâche spécifique n'est disponible. Dans ce cas, nous devons adapter le modèle au contexte de la problématique en l'entraînant avec une base de données appropriée.

3.3.2 Élagage de modèle (Pruning)

L'élagage de modèle est une technique de compression qui cible les architectures de réseaux de neurones complexes. Le but de cette technique est d'éliminer les neurones, les filtres ou même des couches entières qui ont peu contribué à l'obtention de la prédiction par le modèle. Cette technique réduit la dimension du réseau tout en maintenant une bonne précision de détection d'objets (Pachón et al., 2023; Dong and Yang, 2019).

On peut classer les techniques d'élagage des réseaux de neurones selon différentes approches. La première méthode consiste à éliminer les poids individuels sans suivre un cheminement particulier (Poyatos et al., 2023). Cette méthode s'appelle *élagage non structuré* (Liu et al., 2019). La deuxième approche consiste à supprimer systématiquement certains paramètres du modèle en instaurant des règles et des contraintes prédéfinies. On parle alors d'*élagage structuré* (Anwar et al., 2017). Cette seconde méthode est plus avantageuse,

puisqu'elle permet d'obtenir des modèles plus robustes et efficaces en termes d'implantation matérielle (Hardware).

On peut aussi classer les techniques d'élagage de données en *élagage des neurones* et *élagage des connexions*. L'élagage des neurones consiste à désactiver (geler) les neurones qui ont peu contribué à l'obtention des résultats. En revanche, l'élagage de connexions se focalise davantage sur l'élimination temporaire des poids ou des connexions bien spécifiques qui lient les neurones ayant moins contribué à l'obtention de la sortie.

Une autre approche pour classifier les techniques d'élagage de réseau est de les diviser en *élagage statique* et *élagage dynamique*. L'élagage statique est effectué hors ligne (offline), avant la phase d'inférence (donc avant le déploiement du modèle), alors que *l'élagage dynamique* se déroule en cours d'exécution (runtime). Dans ce dernier cas, le modèle ajuste sa propre structure en fonction des données d'entrées ou des contraintes de calcul (Liang et al., 2021).

3.3.2.1 Élagage statique

L'élagage statique est une technique d'optimisation qui s'applique entre la phase d'entraînement et la phase d'inférence. Elle consiste à retirer les neurones, les poids ou les connexions du réseau, de façon hors ligne (off-line). L'objectif d'une telle méthode est de réduire la complexité du modèle, le temps de calcul requis par celui-ci, ainsi que la consommation de mémoire, tout en maintenant une précision acceptable.

Selon Liang et al. (2021), l'élagage statique est réalisé en trois étapes principales:

— **Sélection des paramètres à élaguer**: Identification des paramètres (neurones, poids ou connexions) ayant le moins contribué à l'obtention de la sortie du réseau.

— **Élagage des paramètres**: Application d'une des techniques d'élagage de réseau pour supprimer les paramètres identifiés

— **Ajustement fin (Fine-tuning) et réentraînement du modèle:** Cette étape, facultative, consiste à réajuster les paramètres restants du modèle afin de réduire la perte.

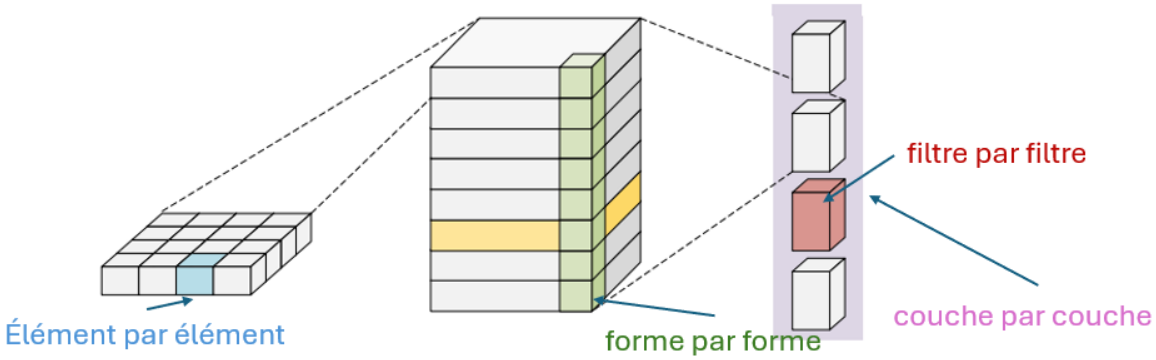


Figure 33: Exemples d'élagage (pruning) à différentes granularités. Selon le niveau de granularité choisi, la sparsité du réseau varie : un élagage par élément offre une suppression très fine, tandis qu'un élagage par canal ou par filtre introduit une structure plus marquée. Enfin, un élagage par couche est plus grossier, simplifiant davantage le modèle, inspiré de (Liang et al., 2021)

La figure 33 illustre que l'élagage (pruning) peut se faire à différents niveaux de granularité. À gauche, on supprime des éléments individuels (element-wise pruning), créant ainsi une sparsité très fine. Au fur et à mesure que la granularité augmente, on supprime des ensembles entiers, comme des canaux, puis des filtres complets et enfin des couches entières. Cela permet d'obtenir une structure plus simple et une accélération potentielle de l'inférence sur des plateformes matérielles bien spécialisées. Parmi les techniques employées pour l'élagage statique, on retrouve *l'élagage basé sur l'amplitude*. Cette technique repose sur l'hypothèse selon laquelle les poids ayant une amplitude importante contribuent de façon significative à la sortie du modèle. Pour déterminer l'importance d'un poids, à savoir s'il doit être conservé ou supprimé, des normes ℓ_p ont été introduites. Pour un vecteur x contenant n éléments, la norme l_p est calculée comme suit:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} \tag{3.1}$$

Avec p= 1 et p=2, la norme ℓ_p prend deux formes particulières correspondant respectivement à la norme de Manhattan et à la norme euclidienne (Tibshirani, 1996; Liang et al., 2021).

Pour la norme LASSO (least absolute shrinkage and selection operator), sa régularisation s'effectue selon la formule suivante (Liang et al., 2021):

$$\arg\min_{\alpha,\beta} \sum_{i=1}^{N} \left(y_i - \alpha - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{avec} \quad \sum_{j=1}^{p} |\beta_j| \leq t \tag{3.2}$$

En partant d'un ensemble de données de taille N où chaque élément possède p caractéristiques normalisées et une seule sortie y_i, le vecteur $\beta = (\beta_1, \ldots, \beta_p)^T$ représente les coefficients du modèle, et t est un paramètre de réglage qui permet de contrôler combien de coefficients/poids seront diminués vers zéro. Le paramètre α peut être fixé à 0 si la moyenne des y_i est nulle. Dans le contexte de la régularisation, une contrainte représente la limite imposée sur la norme des coefficients du modèle. Dans le cas de la régularisation LASSO (least absolute shrinkage and selection operator), la contrainte est $\sum_{j=1}^{p} |\beta_j| \leq t$. Cette contrainte garantit que la somme des valeurs absolues des coefficients du modèle reste toujours inférieure ou égale à t. Si cette contrainte était imposée directement sur les valeurs des coefficients au lieu de leurs valeurs absolues, nous obtiendrions, dans ce cas, une régression de Ridge. L'équation 3.2 peut aussi être reformulée sous forme Lagrangienne:

$$\arg\min_{\beta \in \mathbb{R}^p} [\frac{1}{N} ||y - X\beta||_2^2 + \lambda ||\beta||_1] \tag{3.3}$$

où $\lambda \geq 0$ est le paramètre de régularisation. Dans cette nouvelle formulation, nous introduisons un nouveau paramètre appelé multiplicateur de Lagrange. En plus, en intégrant la contrainte $g(x) = 0$ directement dans la fonction objectif $f(x)$, le problème de minimisation avec contrainte devient un problème de minimisation sans contrainte, grâce à la fonction de Lagrange.

$$L(x, \lambda) = f(x) - \lambda g(x) \tag{3.4}$$

où $\|\cdot\|_p$ est la norme ℓ_p standard, X est la matrice des co-variables contenant les x_{i_j} et k est un paramètre dépendant et lié à t.

Il existe aussi des techniques d'*élagage basé sur les pénalités*. Ces techniques modifient la fonction de perte en y ajoutant des termes supplémentaires, appelés termes de pénalisation, dans le but de minimiser les poids. Ainsi, un poids qui tend vers zéro est plus facile à éliminer (Liang et al., 2021).

Dans l'élagage de modèle, le concept de parcimonie fait référence au nombre de coefficients dans un modèle qui ont été réduits à zéro. Par conséquent, un modèle parcimonieux aura tendance à mieux se concentrer sur le petit nombre de paramètres qui contribuent de manière significative à l'obtention de la sortie du modèle.

Le terme LASSO (least absolute shrinkage and selection operator) correspond à un terme de régularisation agissant comme un biais. Il tend à améliorer la parcimonie en réduisant les poids correspondants des caractéristiques de valeur absolue. Selon Muthukrishnan et Rohini (2016), le LASSO s'est révélé être une meilleure technique de sélection de caractéristiques que les techniques traditionnelles, telles que la régression par moindres carrés ordinaires (MCO), avec une amélioration des performances d'environ 60%.

Cependant, l'élagage par élément (Element-wise pruning) présente un inconvénient majeur: les réseaux obtenus ne sont plus bien structurés ni optimisés. Pour remédier à ce problème, Yuan and Lin (2006) ont proposé une technique innovante appelée Groupe LASSO. Cette dernière permet de remédier au problème mentionné en figeant un ensemble de paramètres au lieu de les traiter un par un, tout en maintenant une bonne structure du modèle. Dans l'équation 3.3 présentée, (Liang et al., 2021), X et β sont remplacés par des groupes de plus grande dimension, notés respectivement X_j et β_j:

$$\arg\min_{b\in\mathbb{R}^p} \left\| y - \sum_{j=1}^{J} X_j\beta_j \right\|_2^2 + \lambda \sum_{j=1}^{J} \|\beta_j\|_{K_j} \tag{3.5}$$

Cette technique permet de réduire le nombre de paramètres afin d'obtenir des modèles plus compacts, consommant moins d'espace mémoire. Par conséquent, la bande passante nécessaire est également réduite. De plus, un petit nombre de paramètres diminue le risque de surajustement (Overfitting) durant l'entraînement. En outre, le fait de réduire le nombre de paramètres permet également de diminuer le nombre de calculs à effectuer, ce qui accélère le temps d'inférence. Des modèles moins complexes en termes d'architecture et ayant moins de paramètres sont plus faciles à implanter.

Liang et al. (2021) abordent également plusieurs techniques d'élagage qui se basent sur le *Group LASSO* (ou LASSO en groupe) et qui visent à neutraliser les poids, les neurones ou même les filtres, telles que:

— Structured Sparsity Learning (Wen et al., 2016).

— Group-wise Brain Damage (Lebedev and Lempitsky, 2016).

— Sparse Convolutional Neural Networks (SCNN) (Liu et al., 2015).

Ces nouvelles approches innovantes ont démontré leur efficacité pour l'accélération matérielle en fournissant en sortie des modèles bien structurés et facilement implantables sur des plateformes matérielles. En effet, ces approches ont contribué à une accélération du temps d'inférence de 5x sur un CPU et de 3x sur un GPU pour le modèle AlexNet.

D'autres techniques, comme Network Slimming (Liu et al., 2017) et Sparse Structure Selection (Huang and Wang, 2018), utilisent également l'approche LASSO (least absolute shrinkage and selection operator) pour l'élagage. Elles ont permis d'obtenir des modèles plus compacts, comme VGG, avec une réduction de taille de 82.5%, sans perte de précision, ainsi qu'une accélération de 4x pour le défi ILSVRC-2012, avec une perte de précision limitée à 3.93%.

Le dropout, illustré dans la figure 34, n'étant pas à la base une technique d'élagage de modèle, mais plus une technique de régularisation stochastique. Il permet de réduire la taille du modèle et d'éviter ainsi le surapprentissage (overfitting). Cette technique tend à désactiver aléatoirement jusqu'à 50% des neurones d'un modèle, le forçant ainsi à ne pas

dépendre excessivement de certaines connexions. Toutefois, cette technique augmente la charge d'entraînement vu qu'elle entraîne des sous-modèles du réseau en question, mais elle n'augmente pas le temps d'inférence, car tous les neurones sont actifs pendant la phase de prédiction (Liang et al., 2021).

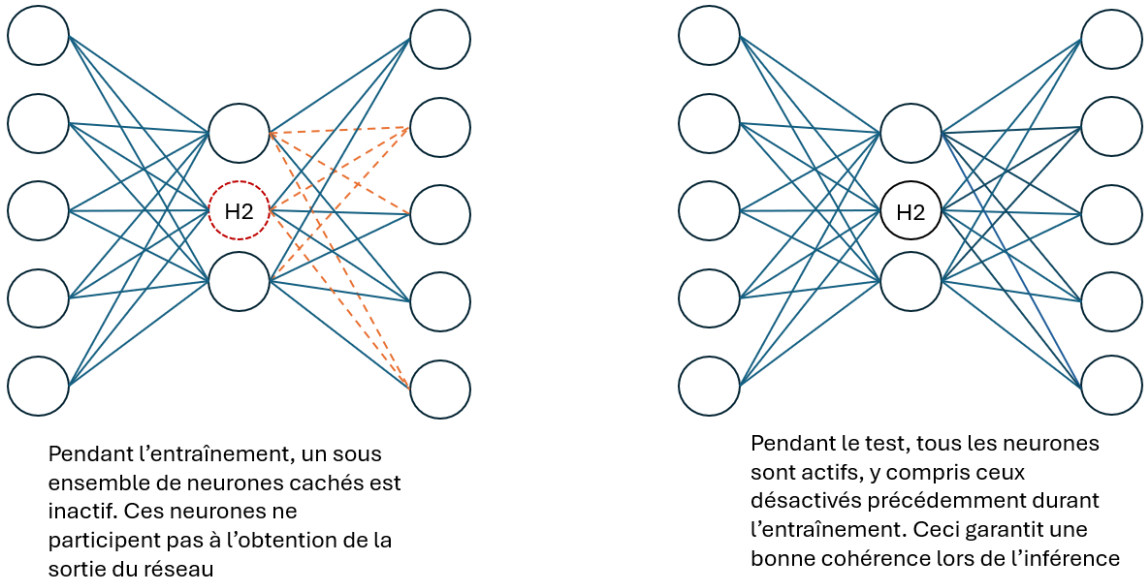


Figure 34: Principe de l'abandon (Dropout) dans les réseaux de neurones (Srivastava et al., 2014)

Dans (Molchanov et al., 2017), une nouvelle technique appelée Sparse Variational dropout a été introduite. Cette technique ajoute un nouvel hyperparamètre de Dropout appelé Taux de Dropout, permettant de compresser les modèles et de réduire leur taille jusqu'à 68%.

Un point important à noter est que l'élagage statique peut cependant nuire à la structure du réseau en supprimant des paramètres importants, ce qui peut entraîner une perte assez significative de précision. Afin de remédier à ce problème et de corriger cette perte, il est souvent nécessaire de reentraîner le modèle ou de procéder à un réglage fin (fine tuning). Plusieurs travaux ont proposé des solutions à ce problème. Par exemple, Han et al. (2016) a introduit une approche appelée compression profonde, structurée en trois étapes:

1. Un élagage statique sur le modèle permettant une réduction de taille allant de 9x à 13x.
2. Une quantification des poids, suivie d'un codage Huffman sur la représentation des données, était effectuée afin de réduire la taille globale du réseau.
3. Le réentraînement du modèle n'intervenait qu'après ces deux premières étapes.

Cette méthode a donné des résultats impressionnants sur plusieurs réseaux. Sur AlexNet entraîné avec la base de données ImageNet, la taille du réseau a été réduite de 35x, passant de 240 MB à 6.9 MB. Sur VGG-16, une compression de 49x a été obtenue sans perte de précision. De plus, cette technique a également permis d'obtenir une accélération des couches de 3 à 4 fois et une amélioration de l'efficacité énergétique de 3 à 7 fois.

Cependant, les changements apportés par un élagage sont généralement irréversibles, ce qui rend nécessaire le recours à des algorithmes d'élagage récupérables (Liang et al., 2021). À cet effet, Guo et al. (2016) ont introduit une nouvelle approche d'élagage appelée chirurgie dynamique des réseaux (Dynamic Network Surgery). Elle s'applique aux réseaux de neurones profonds (DNN) et se base sur un mécanisme d'épissage de connexions permettant au réseau de restaurer des paramètres précédemment élagués, en cas de besoin. Le principal défi de cette approche réside dans l'interdépendance des paramètres du modèle. En effet, un paramètre peut paraître sans importance jusqu'à ce que d'autres paramètres soient retirés. Dès lors, il peut devenir essentiel et crucial pour le réseau. Cette méthode (voir équation 3.6) consiste à appliquer des masques binaires T_k à chaque couche k, en s'appuyant sur une fonction discriminante $h(k)$. Celle-ci vaut 0 si le poids considéré est jugé non important et 1 dans le cas contraire. Le masque T_k est une matrice qui encode l'état des poids de la couche k.

$$h_k(W_k^{(i,j)}) = \begin{cases} 0 & \text{if } |W_k^{(i,j)} < a_k \\ T_k^{(i,j)} & \text{if } a_k \leq |W_k^{(i,j)}| < b_k \\ 1 & \text{if } |W_k^{(i,j)} \geq b_k \end{cases} \tag{3.6}$$

avec $\mathbf{W}_k$ est la matrice de poids dans la couche k, a_k et b_k étant des seuils définis, et t une

marge telle que $b_k = a_k + t$. Les seuils a_k et b_k permettent de déterminer l'importance des poids. Si la matrice des poids est inférieure à a_k, ces poids seront élagués. Dans le cas où $|W_k^{(i,j)}| \geq b_k$, ces paramètres seront retenus comme importants. Dans le cas où $a_k \leq |W_k^{(i,j)}| < b_k$, la fonction h_k prend la valeur de T_k. On peut dire que T_k sert de mécanisme de mémoire qui retient la décision précédente. Les valeurs binaires de $T_k^{(i,j)}$ ne sont pas fixées manuellement : elles sont mises à jour automatiquement pendant l'entraînement, en fonction de la magnitude des poids $|W_k^{(i,j)}|$ et des seuils a_k et b_k.

Pour résumer l'algorithme, nous commençons par un modèle avec tous ses paramètres activés et entraînés. Ensuite, on procède à une propagation vers l'avant, suivie d'un calcul de la perte, puis une propagation vers l'arrière permet d'obtenir le gradient. Une mise à jour des poids est effectuée pour déterminer si les poids seront élagués ou non. Enfin, la matrice W_k est ajustée en fonction de la structure modifiée. Ce processus se répétera jusqu'à la convergence du gradient.

Cette technique est considérée comme innovante, car elle rend le processus de compression de modèle adaptatif et dynamique. D'autres techniques ont été proposées afin de récupérer les paramètres perdus suite à l'élagage, comme l'élagage progressif des filtres SFP (Soft Filter Pruning) (He et al., 2018).

L'élagage peut avoir des effets positifs sur la densité inhérente du modèle, ce qui permet de réduire les risques de sur-apprentissage et ainsi d'améliorer l'efficacité du modèle. Il faut cependant concevoir soigneusement les contraintes de sparsité afin de garantir des résultats prometteurs (Liang et al., 2021). L'un des travaux mentionnés consistait à combiner l'application de contraintes parcimonieuses (Zhou et al., 2016) avec des décompositions tensorielles de faible rang (Liu et al., 2013), ainsi que des groupes peu denses (Deng et al., 2013). Cette combinaison a démontré des résultats significatifs. En effet, on constate une réduction de 70% du nombre de neurones d'AlexNet avec seulement 0.57% de perte de précision top-1, c'est-à-dire la métrique où le modèle est considéré comme correct si sa prédiction la plus probable correspond à la classe réelle.

Un autre point important à mentionner est que le réglage manuel des ratios d'élagage représente un défi majeur dans la compression des modèles. En effet, choisir un ratio élevé engendre un élagage assez agressif, et le modèle peut nécessiter un réglage fin. Une solution pour remédier à ce problème est d'utiliser des techniques automatisées qui garantissent une sparsité adaptative avec des décisions automatisées au niveau des couches. La sparsité peut ainsi être atteinte dans différentes parties d'un réseau de neurones, à savoir au niveau des poids, des noyaux, des canaux ou même des couches entières (Liu et al., 2017). Cependant, il existe un compromis entre la flexibilité et la facilité d'implantation dont il faut tenir compte.

Liu et al. (2017) expliquent aussi que la sparsité au niveau des poids offre la meilleure flexibilité et permet d'atteindre le taux de compression le plus élevé. Cependant, cette approche nécessite des logiciels dédiés à l'implantation matérielle et des accélérateurs matériels pour une inférence rapide du modèle. À l'inverse, la sparsité au niveau des couches, quant à elle, n'exige pas de logiciels spécifiques afin d'accélérer l'inférence du modèle, mais peut nécessiter l'élagage de toute une couche. Cette stratégie reste envisageable lorsque les réseaux sont suffisamment profonds (50 couches et plus). Les auteurs présentent également une technique appelée l'amincissement des réseaux (*Network slimming*), qui vise à réaliser une sparsité au niveau des canaux. En effet, cette sparsité constitue un bon compromis entre flexibilité et facilité d'implantation. Elle peut aussi être appliquée à n'importe quel réseau convolutionnel (CNN) ou entièrement connecté.

L'amincissement d'un réseau (Network slimming) est une technique qui tire parti de la normalisation par lots (*Batch normalization*) et de la pénalité induite par la rareté afin de réduire le nombre de canaux ou de paramètres tout en maintenant la performance du modèle (Liu et al., 2017).

La normalisation par lots (Batch Normalization) est une technique très employée dans les réseaux de neurones convolutifs (CNN) qui vise à améliorer la stabilité de l'entraînement et à accélérer la convergence. Considérons z_{in} et z_{out} respectivement l'entrée et la sortie d'une couche de BN (batch normalization).

La normalisation des lots applique la transformation représentée par l'équation 3.7 (Liu et al., 2017):

$$\hat{z} = \frac{z_{\text{in}} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \quad , \quad z_{\text{out}} = \gamma \hat{z} \; + \; \beta, \tag{3.7}$$

où $\hat{z}$ est l'activation normalisée, μ_B et σ_B sont respectivement la moyenne et l'écart-type calculés sur le mini-lot B. Les paramètres γ et β sont entraînables et représentent respectivement l'échelle et le décalage. Le terme ε est une constante de faible valeur qui sert à éviter une division par zéro. Batch normalization (BN) reçoit en entrée une activation z_{in}, la normalise en utilisant des paramètres statistiques de mini-lots (moyenne μ_B et écart-type σ_B) et applique les deux paramètres γ et β. Il est important de noter que, dans une couche de BN, chaque canal possède son facteur d'échelle (γ). En effet, pour C canaux, il existe C paramètres d'échelle distincts $\gamma_1, \gamma_2, ..., \gamma_C$ chacun contrôlant l'importance du canal correspondant à l'étape de la normalisation. Par conséquent, les auteurs ont trouvé qu'il n'est pas nécessaire d'ajouter des paramètres d'échelle supplémentaires, puisqu'on peut tirer avantage des facteurs d'échelle déjà existants dans les couches de normalisation par lot. En effet, si l'on ajoute un paramètre d'échelle malgré l'existence des paramètres γ des couches de BN, ce paramètre supplémentaire sera également normalisé par cette dernière, annulant ainsi cette mise à l'échelle. Le paramètre γ est suffisant en tant que critère d'élagage. Lorsque γ est ramené vers 0, par exemple, via une régularisation L1, cela indique que ce canal apporte peu d'information et peut donc être élagué sans perte significative de performance (Liu et al., 2017).

Par conséquent, cette approche d'amincissement de réseau (Network slimming), qui consiste à tirer profit des couches de BN, n'admet pas de calcul supplémentaire. Elle est simple, directe et efficace. Une fois la phase d'entraînement terminée, il suffit d'utiliser une technique de sparsité, comme la pénalité L1, pour vérifier si un canal est pertinent à conserver ou s'il doit être élagué.

Pour résumer, l'algorithme du Network slimming, illustré par la figure 35, commence par un entraînement initial durant lequel une pénalité L1 est appliquée sur les facteurs de mise à échelle γ. Cela a pour effet de ramener les faibles valeurs de γ vers zéro. Une fois que l'entraînement est terminé, les canaux ayant un facteur γ proche de zéro sont élagués et, par conséquent, sont retirés du réseau, ce qui conduit à une architecture plus mince. Ensuite, pour éviter toute perte de précision, un réglage fin est employé afin de s'assurer que le réseau désormais élagué conserve sa précision. Enfin, ce processus peut être répété plusieurs fois afin d'obtenir un modèle de plus en plus compact (Liu et al., 2017).

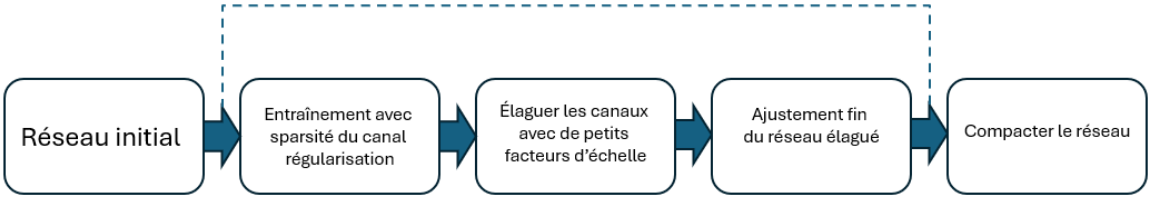


Figure 35: Étape d'amincissement des réseaux de neurones, inspiré de (Liu et al., 2017)

3.3.2.2 Élagage dynamique

La partie précédente aborde l'élagage statique, qui consiste à retirer de façon permanente des paramètres du modèle, conduisant ainsi à un changement de sa structure et, potentiellement, à un impact sur ses performances. Malgré le fait que l'élagage statique permette de réduire l'espace mémoire, le temps de calcul et la consommation d'énergie du modèle, il présente un inconvénient majeur: la suppression de paramètres du modèle est irréversible et peut engendrer une perte d'informations. Des techniques avancées ont été élaborées afin de récupérer la précision du modèle en restaurant certaines connexions perdues, mais leurs résultats restent limités (Liang et al., 2021). Par ailleurs, Gao et al. (2019) mentionnent que l'importance des neurones est loin d'être dépendante des données d'entrée. Par conséquent, retirer une connexion jugée inutile pour une entrée donnée peut être problématique lorsque la prochaine entrée peut en bénéficier.

Afin de résoudre ce problème, l'élagage dynamiquem illustré dans la figure 36, propose une approche différente. L'élagage dynamique permet de déterminer de manière adaptative quels paramètres ou éléments du réseau de neurones ne contribueront pas de manière significative à la sortie. En utilisant les entrées du réseau comme éléments de décision, l'élagage dynamique peut exploiter ou observer les changements au niveau de l'entrée et ainsi désactiver les éléments qui n'ont pas d'influence sur le résultat, sans pour autant les retirer de façon permanente du réseau. Par conséquent, l'élagage dynamique permet de réduire le temps de calcul, la bande passante et la consommation d'énergie sans avoir recours à un réglage fin après chaque décision d'élagage (Liang et al., 2021).

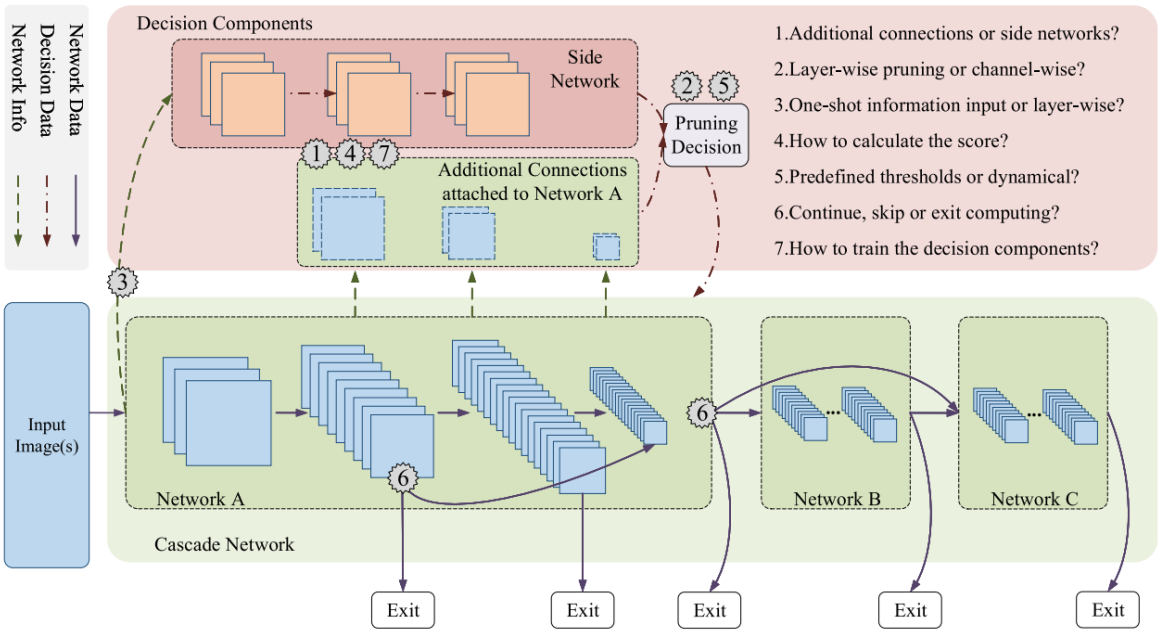


Figure 36: Système de décision pour l'élagage dynamique, inspiré de (Liang et al., 2021)

Comparé aux différents algorithmes utilisés dans l'élagage statique pour déterminer l'importance des éléments, l'élagage dynamique propose un système de décision plus riche qui prend en compte plusieurs facteurs. La figure 36 illustre les éléments ciblés par le système de décision. En effet, celui-ci se base sur des connexions supplémentaires attachées au réseau, des attributs de connexions appris par rétro-propagation (Gao et al., 2019), ou encore sur un

réseau supplémentaire basé sur l'apprentissage par renforcement. Par exemple, Lin et al. (2017a) présente le framework Runtime Neural Pruning, qui effectue de l'élagage en temps réel en fonction de l'entrée (input) et des cartes de caractéristiques intermédiaires. Cela assure le fonctionnement du réseau à sa capacité maximale en n'utilisant que les éléments importants. Ce framework utilise un processus de prise de décision de Markov, combiné avec un réseau entraîné par apprentissage par renforcement. Le réseau supplémentaire agit comme un optimiseur de performance. Il apprend quelles couches sont importantes pour chaque image et lesquelles peuvent être désactivées temporairement. Par conséquent, le modèle de base ne perd ni sa taille ni ses paramètres, ce qui le rend plus facile à ajuster après l'entraînement. Cette flexibilité lui permet d'être utilisée dans différentes applications.

D'autres défis importants pour ce système de décision concernent l'approche à utiliser pour le calcul du score de la décision (Liang et al., 2021). Certains travaux, comme celui de Gao et al. (2019), choisissent la norme l_p pour évaluer l'importance des éléments. Cette valeur de décision est ensuite comparée à une valeur seuil, qui peut être réglée manuellement (Leroux et al., 2017) ou automatiquement (Huang et al., 2017). Un autre point à mentionner est la définition du critère d'arrêt. En effet, certains algorithmes d'élagage suppriment toute une couche ou un sous-réseau (Wu, 2017), tandis que d'autres sélectionnent de façon dynamique le chemin de données (Figurnov et al., 2017). Toutefois, il est possible d'entraîner l'élément de décision même en utilisant l'apprentissage par renforcement, ce qui rend l'élagage encore plus flexible et adaptatif.

Parmi les différentes stratégies utilisées pour élaguer dynamiquement un modèle, on peut citer le calcul conditionnel (Davis and Arel, 2013). En effet, les réseaux de neurones profonds ont tendance à contenir des redondances dans leurs paramètres que l'on peut exploiter pour réduire la compléxité du modèle. Par exemple, on peut approximer la matrice de poids W par une décomposition de rang plus faible (Low-Rank Approximation) sans trop impacter les performances, ce qui nous permet de retirer les paramètres redondants. Il faut aussi mentionner que ces redondances ont plus de chances d'apparaître au niveau des acti-

vations. Par conséquent, identifier ces redondances et ainsi éviter de les calculer permet de réduire dynamiquement la taille du modèle.

La technique d'approximation consiste à incorporer un modèle de contrôle chargé de décider quels neurones devraient être calculés lors du prochain calcul, en tenant compte de l'état actuel des fonctions d'activation. Les neurones considérés comme non importants, c'est-à-dire ceux dont les sorties sont nulles ou négatives après l'application d'une fonction ReLU, seront désactivés pour le prochain calcul.

L'approximation de la matrice des poids se fait par décomposition en valeurs singulières (SVD), comme le propose Davis and Arel (2013). Supposons une matrice $A \in \mathbb{R}^{m \times n}$ qui est exprimée sous la forme :

$$A = U\Sigma V^T, \tag{3.8}$$

où $U \in \mathbb{R}^{m \times m}$, $\Sigma \in \mathbb{R}^{m \times n}$ et $V \in \mathbb{R}^{n \times n}$. D'après (Davis and Arel, 2013), il est possible d'approximer A par une matrice de rang réduit $\hat{A}_r$, en résolvant le problème d'optimisation contraint :

$$\min_{\hat{A}_r} \|A - \hat{A}_r\|_F, \tag{3.9}$$

sous la contrainte :

$$\text{rang}(\hat{A}_r) = r < \text{rang}(A), \tag{3.10}$$

où $\|\cdot\|_F$ désigne la norme de Frobenius.

La solution consiste à conserver uniquement les r premières colonnes de U notées U_r, les r premières valeurs diagonales de Σ notées Σ_r, ainsi que les r premières colonnes de V notées V_r. La matrice approximée s'écrit alors :

$$\hat{A}_r = U_r \Sigma_r V_r^T. \tag{3.11}$$

Dans le contexte d'un réseau de neurones, on définit l'approximation de rang réduit des poids sous la forme :

$$\hat{W}_r = UV \quad \text{où} \quad U = U_r, \quad V = \Sigma_r V_r^T, \tag{3.12}$$

où $\hat{W}_r$ est une matrice approximative de rang r inférieure à celle de W. Cette approximation est plus facile à traiter et permet d'obtenir une approximation plus simple du signe (positif ou négatif) de chaque activation. Si l'on considère une activation $a^{(l+1)}$ dans la couche $l+1$, calculée à partir de l'activation $a^{(l)}$ de la couche précédente, on obtient:

$$a^{(l+1)} \;=\; \sigma((a^{(l)}\, W^{(l)}) + b^l), \tag{3.13}$$

où σ est une fonction d'activation ReLU définie par:

$$\sigma(x) \;=\; \max(0,\, x), \tag{3.14}$$

qui force toute valeur négative à zéro, permettant ainsi de sauter le calcul de la prochaine activation. En remplaçant $\hat{W}_r$ par son approximation obtenue dans l'équation 3.15, nous obtenons:

$$a^{(l+1)} \;\approx\; \sigma\Big(\big(a^{(l)}\,(U^{(l)}\, V^{(l)})\big)\Big) + b^l). \tag{3.15}$$

Une deuxième technique d'élagage dynamique se base sur des réseaux adaptatifs avec des sorties anticipées. En effet, cette technique a pour objectif de minimiser le temps d'inférence en introduisant plusieurs points de contrôle. Ces points permettent au modèle de générer une sortie avant que l'entrée n'ait parcouru tout le réseau.

Un exemple typique est le réseau en cascade, illustré dans la figure 37. Ce type de réseau se compose de plusieurs sous-réseaux. Si un niveau de précision pré-défini est atteint à l'une de ses sorties, le reste des calculs effectués par les couches suivantes peut être ignoré. Cependant, ce fonctionnement présente certaines difficultés. En effet, définir les valeurs de précision seuil n'est pas trivial. Par conséquent, il faut trouver un compromis entre la vitesse

du modèle et sa précision. Une valeur seuil trop faible engendre une faible précision globale du modèle. Tandis qu'une valeur trop élevée peut ralentir la vitesse du modèle. Leroux et al. (2017) montrent que, sur la base de données CIFAR-10, baisser le seuil de 0.9 à 0.5 fait passer l'erreur de test de 15.74% à 28.75%.

Par ailleurs, l'élagage dynamique présente un inconvénient en ce qui concerne le calcul. En effet, le critère de sélection pour l'élagage doit être calculé pendant l'exécution. Par conséquent, ce calcul supplémentaire rajoute une charge de calcul importante, ainsi qu'une latence et une consommation d'énergie. Afin de remédier à ce problème, un compromis doit être trouvé entre le calcul effectué pendant l'élagage, la charge du réseau élagué et la perte de précision.

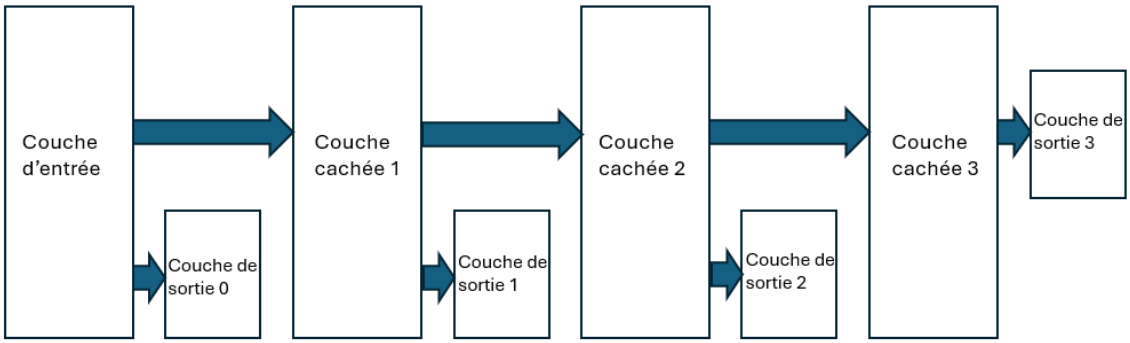


Figure 37: Architecture d'un réseau de neurones en cascade, inspiré de (Leroux et al., 2017)

3.3.3 Quantification

En apprentissage machine, le processus de développement et d'utilisation de modèles se décompose en deux étapes importantes:

1. La phase d'entraînement: Le modèle apprend les caractéristiques importantes à partir des ensembles de données (batch) fournis au modèle sur plusieurs itérations (epochs).
2. La phase d'inférence: Le modèle entraîné est exécuté afin de faire des prédictions, généralement sur des données nouvelles ou différentes de celles fournies durant la phase d'entraînement.

En accélération matérielle, il est important d'examiner les composants matériels qui rendent la plateforme performante. La phase d'inférence dépend fortement des opérations de multiplication matrice-vecteur sous la forme (Nagel et al., 2021):

$$y = W \cdot x + b \tag{3.16}$$

où W représente les poids du modèle, x les entrées et b les biais. Ces opérations sont implantées en utilisant les éléments de traitement PE (Processing Elements) et effectuent les opérations de multiplication et d'accumulation (MAC). Enfin, les accumulateurs rassemblent les résultats.

Dans les réseaux de neurones standards, les calculs sont effectués en utilisant une précision en virgule flottante de 32 bits (32 bits Floating point ou FP32). Ces opérations sont très gourmandes en ressources matérielles et, par conséquent, en termes d'énergie également. Une solution pour remédier à ce problème est d'utiliser une précision plus faible, par exemple un entier de 8 bits (INT8). Cette conversion se fait par un processus que l'on appelle la quantification. Concrètement, la quantification permet de simplifier des calculs complexes comme 7.3x6.4 en une opération plus simple 7x6. Plusieurs travaux, comme celui de Vanhoucke et al. (2011), ont démontré que le fait de baisser la précision des poids, des fonctions d'activation ou des biais n'a pas d'impact significatif sur les performances du modèle. Cependant, cette technique permet de réduire le temps d'inférence et la consommation d'énergie, rendant ainsi le système plus efficace (Liang et al., 2021).

Guo et al. (2017) soulignent quelques points pertinents concernant la quantification. Cette dernière peut également être linéaire (INT8 ou SmoothQuant), ce qui représente le choix le plus courant. La quantification se base sur une simple mise à l'échelle, suivie d'un écrêtage, mais elle peut rencontrer des difficultés lorsque les données ont une large plage dynamique. De plus, on retrouve également des méthodes de quantification non linéaire permettant de réduire davantage la taille des données (par exemple, 4 bits, binaire), bien qu'elles soient plus complexes à mettre en œuvre. Ces méthodes utilisent le regroupement de

poids (Nock and Nielsen, 2006) ou le mappage via les tables de correspondance LUT (Look-Up Table) apprises afin de réduire la largeur de bits sans perte significative de précision.

Il est important de noter que, parmi les avantages offerts par la quantification, notamment la réduction de la charge de calcul, celle-ci réduit également la bande passante et l'utilisation de l'espace mémoire. Par conséquent, cela permet une meilleure utilisation des hiérarchies matérielles de mémoire (Guo et al., 2017).

Dans la littérature, la quantification est souvent divisée en deux catégories : la *quantification après l'entraînement* ou PTQ (post-training quantization) et *l'entraînement tenant compte de la quantification*, ou QAT (quantization-aware training).

3.3.3.1 Quantification après l'entraînement (PTQ)

La quantification après l'entraînement PTQ (Post-Training Quantization) est une technique qui consiste à réduire la précision des poids et des activations, typiquement de 32 bits en virgule flottante (FP32) à une précision plus faible, comme des entiers à 8 bits (INT8). Cette étape se fait après que le modèle a déjà été entraîné et ne requiert pas une nouvelle phase d'entraînement. L'objectif d'une telle méthode est de réduire la consommation de mémoire et les charges de calcul du modèle, tout en conservant des performances acceptables (Sharify et al., 2024).

On retrouve plusieurs techniques de quantification après l'entraînement. Parmi elles, *SmoothQuant* (Xiao et al., 2024) est conçue pour améliorer la précision des modèles quantifiés (INT8) sans avoir besoin de les reentraîner. En utilisant cette technique, on s'assure d'équilibrer l'échelle entre les poids et les activations, et ce, avant la quantification. Les techniques de quantification traditionnelles sont très souvent impactées négativement par des valeurs aberrantes d'activation, qui peuvent provoquer d'importantes pertes de précision ou de faibles plages de quantification. La plage d'activation correspond à l'intervalle minimum-maximum d'un tenseur d'activation, c'est-à-dire la sortie d'une fonction d'activation (comme

SiLU, ReLU, etc.). La quantification consiste à faire correspondre (ou ”mapper” en anglais) des valeurs en virgule flottante de bits (FP32) à un ensemble de valeurs entières (INT8), généralement comprises entre -128 et 127. Lorsqu'une valeur aberrante est présente dans les activations, le quantificateur doit étendre la plage des valeurs pour l'inclure, ce qui a pour effet de détériorer la précision de la représentation des valeurs proches de zéro (Xiao et al., 2024). SmoothQuant introduit un terme qui lisse la plage d'activation ainsi que les valeurs des poids, permettant ainsi une meilleure quantification en réduisant l'influence des extrêmes. L'équation 3.17 illustre le fonctionnement de base d'un réseau de neurones.

$$Y = W \cdot X + b \tag{3.17}$$

SmoothQuant introduit un terme l afin de lisser l'opération de multiplication:

$$Y = \left(\frac{W}{l}\right) \cdot (l \cdot X) \tag{3.18}$$

— W étant les poids (weights),

— X étant les activations,

— l est un facteur d'échelle calculé pour chacun des canaux.

Cette technique présente plusieurs avantages. Elle permet notamment de réduire la perte de précision lors du passage d'une représentation à virgule flottante de 32 bits (FP32) à une représentation à entiers fixes de 8 bits (INT8). De plus, elle évite de réentraîner le modèle après l'opération de quantification.

D'autres techniques de quantification après l'entraînement (PTQ) ont également été élaborées, telles que GPTQ (Frantar et al., 2023) et AWQ (Lin et al., 2023).

3.3.3.2 Entraînement tenant compte de la quantification (QAT)

L'entraînement tenant compte de la quantification QAT (Quantization Aware Training) consiste à intégrer la quantification pendant la phase d'entraînement. Contrairement au PTQ (Post training quantization) qui quantifie un modèle pré-entraîné, le QAT simule la quantification (INT8) pendant l'apprentissage. Cette technique a pour but de permettre aux poids et aux activations du modèle de s'adapter dès le départ à une précision réduite. Pour cela, un bruit de quantification est ajouté pendant l'entraînement afin de rendre le modèle plus robuste aux représentations à faible nombre de bits.

Pendant le QAT, toutes les entrées, les poids, les activations et les sorties sont quantifiés en une représentation à point fixe (Fixed point representation 8 bits integers) lors de la passe avant (Forward pass). Cependant, ces paramètres demeurent inchangés lors de la passe arrière (Backward pass) en raison de l'utilisation du Straight-Through Estimator (STE), qui traite la dérivée de la fonction de quantification comme une identité. Ce qui rend, par conséquent, le QAT compatible avec des flux d'entraînement existants tout en tenant compte de la quantification sur le modèle (Kirtas et al., 2023).

On fait correspondre à tout signal $h^{(i)}$, qu'il s'agisse d'une entrée, d'un poids ou d'une sortie, sa version quantifiée $h_q^{(i)}$, définie par:

$$h_q^{(i)} = \text{clip}\left(\left\lfloor \frac{h^{(i)}}{s_h^{(i)}} + f_h^{(i)} \right\rfloor, q_{\min}^{(i)}, q_{\max}^{(i)}\right) \tag{3.19}$$

où $s_h^{(i)} \in \mathbb{R}^+$ est le facteur de mise à l'échelle calculé à partir de $[h_{\min}^{(i)}, h_{\max}^{(i)}]$, $f_h^{(i)} \in \mathbb{N}$ est le zéro-point, c'est-à-dire le décalage pour centrer la plage de quantification autour de zéro, $q_{\min}^{(i)}, q_{\max}^{(i)} \in \mathbb{N}$ représente les limites de valeur de quantification, par exemple, des entiers non signés sur 8 bits [0, 255], et clip est une fonction d'écrêtage qui garantit que la valeur reste comprise dans la plage valide de quantification (Kirtas et al., 2023).

La fonction inverse de quantification permet de restituer une approximation du modèle dans son format original en virgule flottante (FP32)

$$h_f^{(i)} = s_h^{(i)} \left(h_q^{(i)} - f_h^{(i)} \right) \tag{3.20}$$

Dans (Jacob et al., 2017), les auteurs ont proposé une approche de quantification qui affine le processus de correspondance (mapping) où les poids et les activations sont représentés par des entiers de 8 bits (INT8), tandis que les biais sont représentés par des entiers de 32 bits (INT32). La phase d'inférence a été exécutée en utilisant uniquement l'arithmétique des nombres entiers. Pendant la phase d'entraînement, la quantification a été simulée grâce à des nœuds de fausse quantification (False quantization) afin d'imiter le comportement du temps d'inférence, tout en préservant les poids au format virgule flottante (FP32) pour mettre à jour de façon précise le gradient.

L'évaluation de l'entraînement tenant compte de la quantification (QAT) sur les modèles ResNet a démontré que le modèle quantifié en 8 bits présente des résultats presque similaires à ceux du modèle original en 32 bits (FP32) avec seulement 1.5% de perte de précision. Cette représentation en 8 bits était meilleure que d'autres approches de quantification, telles que les Binary Weight Networks (BWN) (Leng et al., 2018) et la Fine-Grained Quantization (FGQ) (Mellempudi et al., 2017). Quant au modèle InceptionV3 (Szegedy et al., 2016), les représentations en 7 bits et 8 bits ont réussi à maintenir la précision du modèle original lorsque la fonction d'activation ReLU6 a été utilisée. Concernant le compromis Précision-vitesse, la version quantifiée uniquement en entiers de MobileNets, testée sur les jeux de données ImageNet, était plus performante que la version originale FP32 lorsqu'elle était exécutée sur des processeurs Qualcomm Snapdragon.

Par ailleurs, une évaluation du modèle MobileNet SSD, entraîné avec la base de données COCO, a démontré jusqu'à 50% de gain en vitesse avec seulement 1.8% de perte de précision. Le fait de changer les convolutions standards par des convolutions séparables en

profondeur (dephtwise seperable convolutions) a aidé le modèle à maintenir ses performances tout en réduisant sa compléxité (Jacob et al., 2017).

3.4 Plateformes d'accélération matérielle

Parmi les solutions proposées pour relever le défi du calcul intensif effectué par les réseaux de neurones, notamment les opérations de convolution, figure la conception de plateformes matérielles spécialisées. Le choix de la plateforme matérielle dépend généralement de plusieurs facteurs:

— **Débit (throughput)**

Le débit (throughput) exprimé en IPS (Images per Second) ou en GOPS (Giga Operations Per Second) est une métrique qui mesure le nombre d'opérations effectuées par seconde. Il est défini par:

$$\text{IPS} = \frac{f \times P \times \eta}{W} \tag{3.21}$$

où f est la fréquence de fonctionnement (Hz), P est le nombre d'unités de calcul, η est le taux d'utilisation des unités de calcul et W est le nombre total d'opérations requises pour une tâche donnée.

— **Latence**

La latence, exprimée en milli-secondes (ms), représente le temps nécessaire pour traiter une seule entrée. Elle est définie par:

$$L = \frac{C}{\text{IPS}} \tag{3.22}$$

où C représente la complexité de calcul liée à l'entrée (nombre total des opérations nécessaires).

— **Efficacité énergétique**

L'efficacité énergétique (Energy Efficiency), mesurée en Ops/Joule ou GOPS/W,

représente la quantité d'opérations exécutées par unité d'énergie consommée. Cette métrique est exprimée par:

$$\text{Eff} = \frac{W}{E_{\text{total}}} \tag{3.23}$$

où E_{total} est l'énergie totale consommée, exprimée en Joules (J), estimée par:

$$E_{\text{total}} \approx W \times E_{\text{op}} + N_{\text{SRAM}}^{\text{acc}} \times E_{\text{SRAM}} + N_{\text{DRAM}}^{\text{acc}} \times E_{\text{DRAM}} + E_{\text{static}} \tag{3.24}$$

où W est le nombre total d'opérations effectuées, E_{op} est l'énergie consommée par opération, $N_{\text{SRAM}}^{\text{acc}}$ et $N_{\text{DRAM}}^{\text{acc}}$ sont respectivement le nombre d'accès à la mémoire SRAM et DRAM, E_{SRAM}, E_{DRAM} sont les énergies par accès à la mémoire et E_{static} représente l'énergie statique consommée par le système. Plusieurs plateformes matérielles sont conçues pour offrir un compromis optimal entre la puissance de calcul, la consommation d'énergie et la flexibilité, en fonction des besoins spécifiques des applications.

3.4.1 Unité centrale de traitement (CPU)

Une unité centrale de traitement CPU (Central Processing Unit) est le processeur principal d'un ordinateur. Cette unité est capable d'exécuter des instructions arithmétiques, logiques et de contrôle, ainsi que des opérations d'entrées/sorties (lecture et écriture). Le CPU est particulièrement efficace pour exécuter une large gamme de tâches générales. Cependant, l'inconvénient majeur de cette plateforme est qu'elle exécute les tâches de manière séquentielle. Ce mode de fonctionnement n'est pas adapté aux calculs requis par les réseaux de neurones. En effet, le calcul effectué par les réseaux de neurones, surtout les CNN, est massivement parallèle. Par conséquent, les CPUs ne constituent pas une solution efficace pour une accélération matérielle dans ce contexte, où le besoin est axé davantage sur un calcul utilisant des millions d'opérations de manière simultanée (Maurizio et al., 2024; Gschwend, 2020).

3.4.2 Unité de traitement graphique (GPU)

Les unités de traitement graphiques GPU (Graphics Processing Unit) sont des composants essentiels dans le calcul à haute performance moderne HPC (High Performance Computing). Ces cartes sont très connues pour leur architecture massivement parallèle, leur permettant d'effectuer des milliers d'opérations de calcul simultanément. Leur très large bande passante de mémoire leur permet de transférer efficacement des volumes importants de données notamment à partir de grandes bases de données. Un autre avantage majeur des GPUs est leur capacité à gérer les calculs intensifs en virgule flottante, un critère essentiel pour les simulations scientifiques (Das and Gupta, 2024). Les GPUs ont été initialement utilisés pour des tâches de nature graphique, comme le rastering, le traitement de sommets (vertex processing) dans les pipelines 3D et l'ombrage de pixels. Comparés aux CPUs, généralement optimisés pour la latence et le parallèlisme limité, les GPU sont plus axés sur le débit (throughput) (Mišić et al., 2012).

Les GPU sont aujourd'hui utilisés dans plusieurs domaines: le traitement d'images et de vidéos, l'intelligence artificielle, l'apprentissage machine, les systèmes robotiques multi-agents et les systèmes embarqués temps-réel. Dans (Youvan, 2023), les auteurs ont présenté la contribution des plateformes GPU à l'accélération de l'intelligence artificielle. L'introduction de CUDA par (Nvidia, 2025) a largement démocratisé l'utilisation des GPUs dans l'accélération matérielle. Ambuj and Machavaram (2024) a présenté un framework d'optimisation multi-objectifs accéléré par un GPU, conçu pour minimiser les dépenses énergétiques des voitures autonomes en agriculture. Cette accélération a permis d'obtenir jusqu'à 5 332x de gain en vitesse pour le calcul de la fonction de fitness, 390x pour la mise à jour des positions et 257x pour les mises à jour de vitesse. Leur implantation n'a pas seulement permis d'accélérer la vitesse de calcul, mais aussi de réduire l'erreur quadratique moyenne de 0.04 à 0.009.

3.4.3 Circuit intégré à application spécifique (ASIC)

Les circuits intégrés à application spécifique ASIC (Application-Specific Integrated Circuit) sont des circuits intégrés conçus pour exécuter une tâche ou une application particulière. Contrairement aux processeurs généralistes, ceux-ci offrent des performances optimisées, une consommation réduite et une efficacité supérieure pour un usage ciblé. Des ASIC spécialisés ont également été conçus pour l'accélération des réseaux neuronaux, ciblant généralement l'inférence à la périphérie, par exemple dans les caméras de sécurité ou les appareils mobiles (Liang et al., 2021). Jouppi et al. (2017) ont proposé leur système ASIC appelé TPU (Tensor Processing Unit), qui a été déployé dans les centres de données de Google depuis 2015. Ce TPU avait atteint jusqu'à 92 TOPS (Tera Operations Per Second), tout en conservant une faible consommation d'énergie. Comparé aux processeurs Intel Haswell et aux GPUs Nvidia K80 contemporains, le TPU offre des performances 15 à 30 fois supérieures et une efficacité énergétique 30 à 80 fois supérieure. Ces gains sont obtenus en grande partie grâce à ses chemins de données quantifiés en entiers et à son modèle d'exécution déterministe, qui répondent mieux aux contraintes de latence du 99^{e} centile, essentielles pour les charges de travail d'inférence.

3.4.4 Réseau de portes Programmable sur site (FPGA)

Les réseaux de portes programmables sur site FPGA (Field Programmable Gate Array) sont des circuits intégrés reconfigurables capables d'implanter des conceptions logiques numériques. Contrairement aux circuits à fonctions fixes, tels que les CPUs ou les GPUs, les FPGAs conventionnels sont constitués d'une matrice de blocs logiques configurables (CLB), comme le montre la figure 38. Ces blocs sont reliés par des interconnexions programmables et des blocs d'entrées/sorties IOB (Input/output Blocks). Les connexions entre les blocs CLB sont assurées par des commutateurs (switchs) programmables (transistors), eux-mêmes pilotés par des cellules de RAM de 1 bit (Brown and Vranesic, 2014).

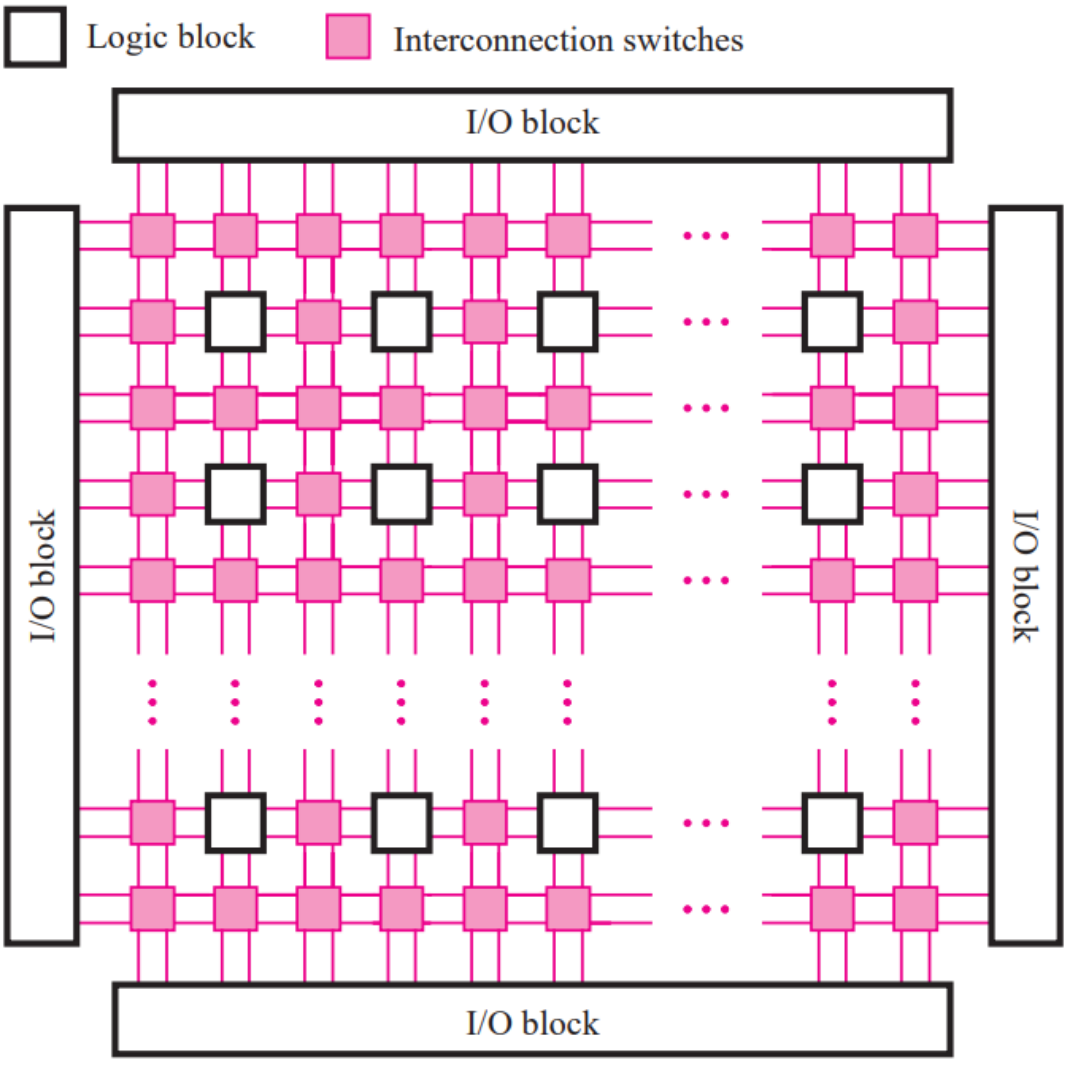


Figure 38: Architecture d'un circuit FPGA conventionnel (Brown and Vranesic, 2014)

Les blocs CLB sont constitués de deux tables de correspondance LUT (Look-up tables) à 4 entrées. Chaque table est capable d'implanter toutes sortes de fonctions logiques allant jusqu'à quatre variables. En opérant séparément, ces deux LUTs permettent d'exécuter deux fonctions logiques distinctes. Elles peuvent être associées à des bascules intégrées (flip-flops) pour prendre en charge également la logique séquentielle (Brown and Vranesic, 2014). Enfin, ces blocs peuvent également être utilisés comme mémoire d'une taille de 16x1 RAM (ou 32x1 RAM en utilisant les deux tables LUT). Au fil des années, les FPGA ont connu une évolution technologique remarquable, passant de simples réseaux logiques programmables à des systèmes hétérogènes sur puce (SoC).

Les premières générations de FPGA se limitaient à des blocs CLB et à un réseau d'interconnexions programmables, principalement destinés au prototypage de circuits logiques, comme l'illustre la figure 37. Avec l'augmentation de la densité des transistors et l'amélioration des fréquences d'horloge, les FPGA ont progressivement gagné en performance et en capacité, rendant possibles des applications embarquées complexes. Entre 1900 et 2000, l'intégration de blocs matériels spécialisés tels que les multiplicateurs matériels (DSP slices), les blocs de mémoire embarqués (BRAM) et les interfaces haut débit (PCIe, Ethernet, DDR) a permis aux FPGA de devenir de véritables plateformes de calcul haute performance (Brown and Rose, 1996).

Plus tard, au début des années 2000, Xilinx a introduit le microprocesseur MicroBlaze (Xilinx, 2002). MicroBlaze (voir figure 39) est un processeur RISC 32 bits configurable, implanté dans la logique programmable en utilisant les LUT/FF/BRAM. Comparé à un processeur traditionnel, Microblaze est considéré comme un processeur logiciel (soft processor), étant donné qu'il ne s'agit pas d'un composant physique fixe. Il s'agit d'une conception de processeur implanté à l'aide de la logique programmable d'un FPGA. Microblaze possède une fonctionnalité personnalisable et peut être configuré pour optimiser ses performances de traitement, sa fréquence de fonctionnement ou une combinaison entre ces éléments (Crockett et al., 2014).

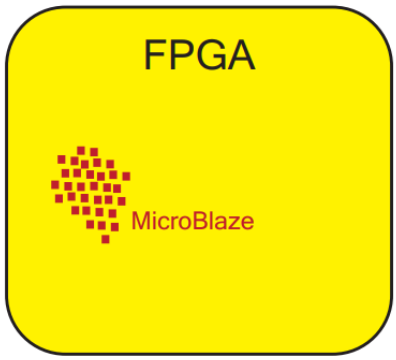


Figure 39: FPGA avec processeur logiciel (par exemple MicroBlaze) (Crockett et al., 2014)

L'étape majeure et marquante de l'histoire des FPGA est survenue au cours des années 2010. Xilinx a integré des cœurs de processeur physiques sur la même puce que la structure programmable. La gamme Zynq-7000 de Xilinx (puis, plus tard, le Zynq UltraScale+ MPSoC) intégrait des processeurs d'application ARM (Cortex-A9 / A53, etc.) ainsi que la structure FPGA, formant ainsi un véritable SoC FPGA. Ce modèle PS (système de traitement) + PL (logique programmable) a permis une co-conception étroite entre le matériel et le logiciel. En effet, les tâches de calcul intensif s'exécutent sur le matériel (PL), tandis que les tâches de contrôle et le système d'exploitation s'exécutent sur les cœurs ARM physiquement intégérs (PS) (voir la figure 40).

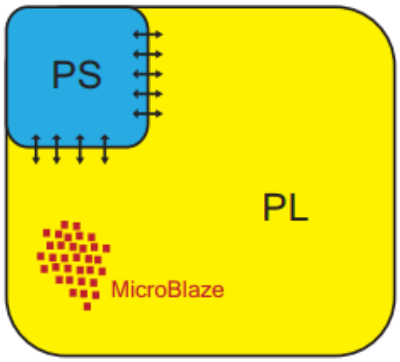


Figure 40: Architecture d'un circuit Zynq (le microblaze est optionnel) (Crockett et al., 2014)

Cette architecture permet aux développeurs de définir, avec une grande flexibilité, des fonctionnalités matérielles après la fabrication du circuit. La figure 41 démontre une comparaison entre les trois plateformes classiques (CPU, GPU, FPGA) en termes de rapport flexibilité/efficacité énergétique.

L'architecture d'un FPGA est qualifiée de spatiale. En effet, chaque élément de traitement (Processing Element, PE) y fonctionne de manière autonome, utilisant sa propre mémoire locale ainsi que sa propre logique de contrôle. Les PEs peuvent s'interconnecter directement entre eux et sont généralement organisés sous forme de grille, ce qui facilite l'acheminement des données et permet une exécution hautement parallèle et pipelinée, notamment pour des opérations intensives telles que les multiplications–accumulations (MAC). En revanche, l'architecture des CPU/GPU, quant à elle, est qualifiée de *temporelle*. Dans ces architectures, les PEs ne communiquent pas directement entre eux et dépendent d'une unité de contrôle centralisée ainsi que d'une hiérarchie mémoire (registres, cache, DRAM) chargée d'ordonner les opérations et de gérer le flux de données. Le parallélisme est obtenu principalement par l'exécution simultanée d'une même instruction sur plusieurs données, plutôt que par une spécialisation matérielle distribuée comme dans les architectures spatiales (Capra et al., 2020).

Le tableau comparatif 6 résume les avantages et les inconvénients de chacune des plateformes matérielles abordées précédemment.

Table 7: Comparaison des plateformes matérielle: CPU, GPU, FPGA et ASIC.

Plateforme	**Avantages**	**Inconvénients**
CPU (Central Processing Unit)	- Très flexible, adapté à des tâches générales - Large écosystème logiciel et support - Facile à programmer (C/C++, Python, etc.) - Bonne performance pour calculs séquentiels	- Performance limitée pour le calcul massivement parallèle - Consommation énergétique plus élevée que FPGA/ASIC pour IA - Moins adapté aux applications temps réel intensives
GPU (Graphics Processing Unit)	- Excellente performance pour calcul parallèle massif (Deep Learning, vision par ordinateur) - Large support logiciel (CUDA, OpenCL, PyTorch, TensorFlow) - Idéal pour l'entraînement de modèles IA	- Consommation énergétique élevée - Latence parfois élevée - Moins efficace pour l'inférence en périphérie (edge computing)

Plateforme	**Avantages**	**Inconvénients**
FPGA (Field Programmable Gate Array)	- Reconfigurable matériellement (flexibilité) - Haute efficacité énergétique (par rapport au GPU) - Faible latence (idéal pour l'inférence en temps réel) - Bon compromis entre performance et consommation	- Plus difficile à programmer (VHDL, Verilog, etc) - Développement long et coûteux en expertise - Moins performant que GPU pour l'entraînement
ASIC (Application-Specific Integrated Circuit)	- Performances maximales pour une application donnée - Très haute efficacité énergétique - Faible latence - Solution idéale pour déploiement massif (ex: Google TPU)	- Non reconfigurable (rigide) - Développement extrêmement coûteux - Temps de conception long (plusieurs mois/années) - Pas adapté à la recherche ou au prototype

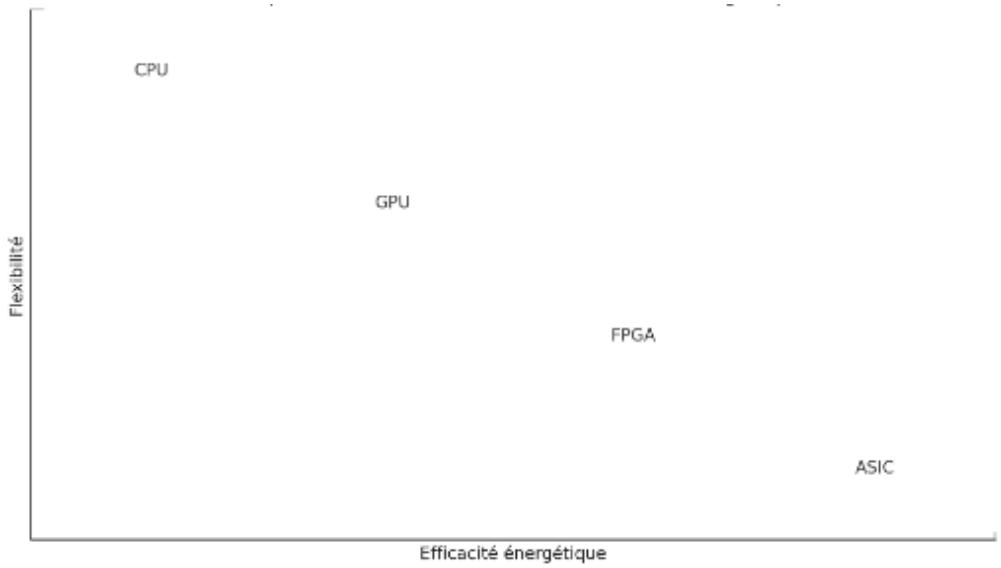


Figure 41: Compromis entre flexibilité et efficacité énergétique (Maurizio et al., 2024)

3.5 Conclusion

Dans ce chapitre, les différentes techniques d'optimisation des réseaux de neurones ont été abordées. L'objectif de ces méthodes est de réduire considérablement la taille du modèle afin de le rendre plus rapide sans trop impacter ses performances. L'avantage majeur de ces techniques est de pouvoir déployer ces modèles optimisés sur différentes plateformes matérielles.

Une étude comparative de trois plateformes très populaires dans la littérature a été menée afin de comprendre leurs spécificités et de choisir celle qui serait la plus adaptée à ce projet. Le chapitre suivant détaillera les différentes étapes entreprises pour la réalisation du projet, ainsi que les résultats obtenus.

CHAPITRE 4

MISE EN ŒUVRE MATÉRIELLE ET RÉSULTATS

Ce chapitre abordera les différentes étapes de réalisation du projet, allant de l'entraînement du modèle de détection d'objets à son implantation matérielle sur le circuit FPGA de la carte Kria KV260. Nous présenterons les différentes bases de données utilisées ainsi que les techniques d'optimisation choisies afin d'assurer une implantation matérielle efficace. Les résultats obtenus seront présentés à l'aide des courbes et synthétisés dans des tableaux. Une évaluation des performances du modèle à la fois sur la machine hôte et sur le circuit FPGA, sera abordée. Finalement, une analyse comparative des différentes plateformes matérielles utilisées durant ce projet sera menée.

4.1 Choix du modèle de détection d'objets

Dans le contexte de ce projet de recherche, qui prend en compte la contrainte des ressources matérielles et l'exigence d'exécution en temps réel, nous avons opté pour le modèle de détection d'objets YOLOv5, et plus particulièrement sa version YOLOv5n, pour diverses raisons. D'une part, en termes de capacité d'exécution en temps réel, YOLOv5 fait partie des modèles de détection à une passe (Single Stage detector), c'est-à-dire qu'il réalise la détection en traitant l'image d'entrée en une seule étape. Ce critère est essentiel pour les applications en temps réel. À l'inverse, les modèles de détection à deux passes, comme Faster R-CNN (Ren et al., 2017), présentent une latence plus élevée, car ils nécessitent une première étape de génération de propositions de boîtes englobantes avant de passer par le réseau de classification. D'autre part, en termes d'efficacité dans un contexte de ressources limitées, YOLOv5s a démontré son efficacité dans plusieurs travaux. Par exemple, Yan et al. (2024) ont démontré qu'une quantification en 4 bits du modèle YOLOv5s permet de réduire

considérablement l'utilisation de ressources matérielles (50% de blocs DSP en moins) avec un impact minimal sur les performances du modèle. En outre, YOLOv5n a été spécialement conçu avec une taille qui peut facilement être implantée sur des circuits embarqués, sans perte significative de précision. En effet, la version nano YOLOv5n ne possède que 1.9 million de paramètres, contre 86.7 millions pour la version extra large YOLOv5x. La différence de taille réside principalement dans la largeur et la profondeur des modules BottleCSP utilisés. Enfin, YOLOv5 s'est imposé comme un modèle de référence dans l'état de l'art, étant donné le grand nombre de travaux qui se basent sur son architecture.

4.2 Choix des bases de données

Pour ce projet, nous avons opté initialement pour les deux bases de données COCO (Common Objects in Context) et HAM10000 (Human Against Machine). COCO est une base de données à grande échelle, conçue par Microsoft pour la détection et la segmentation d'objets. Elle contient 330 000 images, parmi lesquelles 200 000 possèdent des annotations, couvrant 80 catégories d'objets du quotidien (Lin et al., 2014). Elle est connue pour la diversité des objets figurant dans ses images. Cette base de données est utilisée dans plusieurs travaux de recherche et constitue l'une des bases de données de référence pour l'entraînement de modèles d'apprentissage profond (Lin et al., 2014). Il faut noter que dans cette base de données, il existe plusieurs splits. Nous avons choisi le split COCO2017 qui contient en tout 164000 images. La répartition des données dans la base de données COCO est la suivante :

— Le dossier **Train2017**, contenant 118 000 images, est utilisé pour l'entraînement des modèles de détection d'objets.

— Le dossier **Val2017**, contenant 5 000 images, est utilisé pour la validation.

— Le dossier **Test2017**, contenant 40670 images, est utilisé pour la phase d'évaluation ou de test.

Cette base de données est accompagnée de son propre fichier YAML (Yet Another Markup Language), qui définit de façon claire et structurée la configuration de la base de

données. Ce type de fichier est couramment utilisé dans les projets de machine learning pour simplifier la gestion des données.

La base de données HAM10000 (Human Against Machine) est une base dermatoscopique qui contient 10 000 images destinées à l'entraînement et couvrant 7 classes de diagnostics dermatologiques communs. Ses images proviennent de deux sources: l'Université médicale de Vienne, en Autriche (groupe ViDIR), et la clinique de dermatologie Cliff Rosendahl's Skin Cancer Practice, située dans le Queensland, en Australie. Cette base de données publique a été conçue pour remédier au problème de taille et de manque de diversité des anciennes bases de données qui se focalisaient uniquement sur les lésions mélanocytaires (Tschandl et al., 2018). Cependant, cette base de données ne contient malheureusement pas de données annotées avec des boîtes englobantes (bounding boxes). Pour répondre à cette limitation, nous avons utilisé une base dérivée de la base HAM10000 annotée manuellement à l'aide de l'outil Roboflow.

Roboflow (Nelson and Dwyer, 2020) est une plateforme conçue pour la préparation et l'annotation de données pour des applications de vision par ordinateur. Dans le cadre de ce projet, nous avons utilisé cette plateforme uniquement pour obtenir des images annotées. La base annotée extraite de HAM10000 contient :

— 6184 images pour l'entraînement.

— 643 images pour la validation.

— 410 images pour le test.

Plusieurs techniques d'augmentation de données ont été appliquées à cette base de données afin de renforcer la robustesse du modèle de détection. Parmi celles-ci, des retournements aléatoires à l'horizontale/verticale et des rotations aléatoires comprises entre -20 degrés et +20 degrés ont été effectués afin de visualiser plusieurs perspectives. Au niveau des couleurs, une modification de la saturation a été appliquée afin d'ajuster l'intensité des couleurs entre -25% et +25%. Une variation visuelle, notamment au niveau de l'éclairage (entre -8% et +8%) et une variation de la focalisation (jusqu'à 2 pixels de flou gaussien) ont

été simulées sur les images afin de rendre le modèle plus robuste et généraliste. Un bruit a également été introduit dans les images (jusqu'à 5% des pixels).

Une troisième base de données a été utilisée lors de la réalisation de ce projet: la base KITTI, considérée comme l'une des plus couramment utilisées dans le domaine de la conduite autonome. Créée en 2012 par l'Institut de technologie de Karlsruhe (Allemagne), en collaboration avec Toyota, cette base a été conçue dans le but d'évaluer des algorithmes de détection d'objets (voitures, piétons, cyclistes, etc.), de segmentation sémantique et d'estimation de profondeur dans un contexte urbain réel. Pour notre projet, nous avons adapté les images de KITTI à la dimension 224x640 acceptée par le modèle YOLOv5.

Le répertoire `Training` contient 7 481 images qui seront réparties grâce au script `train.py` en 80% (5 984 images) pour l'entraînement et 10% (748 images) pour la validation. Nous avons utilisé les 10% restantes (749 images) pour les tests.

Il est important de tenir compte de deux points importants lors de l'entraînement du modèle YOLOv5.

— L'hiérarchie de la base de données.

— Le format des données d'annotation.

En effet, pour l'hiérarchie des données, YOLOv5 exige une structure définie des répertoires de données. Le modèle utilise les images contenues dans le dossier train pour l'entraînement, tandis que les images du répertoire Val/ seront utilisées pour la validation. Les annotations servent généralement à fournir les informations nécessaires contenues dans les images. Dans le cas de détection d'objets, les annotations sont généralement stockées dans des fichiers texte, où chaque ligne représente un objet détecté (représenté par les coordonnées de la boîte englobante). YOLOv5 utilise un format spécifique dans ses fichiers, où chaque ligne doit contenir cinq informations, séparées par des espaces :

```
<class_id> <x_center> <y_center> <width> <height>
```

— *class_id*: L'identifiant correspondant à la classe, qui devrait correspondre à celui crée dans le fichier YAML ou attribué manuellement.

— < *x_center* > < *y_center* >: Les coordonnées du centre de la boîte englobante, généralement normalisées (entre 0 et 1) par rapport à la hauteur et à la largeur de l'image.

— < *width* > et < *height* >: La largeur et la hauteur de la boîte englobante sont normalisées aux dimensions de l'image.

Exemple :

```
1 0.410 0.59 0.100 0.400
```

Ceci signifie que l'objet appartenant à la classe 1 (chien, par exemple) est représenté par une boîte englobante dont le centre est situé à 41% de la largeur et à 59% de la hauteur de l'image ; la largeur correspond à 10% de la largeur de l'image; la hauteur correspond à 40% de la hauteur de l'image.

4.3 Entraînement et validation du modèle

Pour entraîner le modèle YOLOv5, il est tout d'abord nécessaire de choisir la taille (variante) du modèle. Ultralytics (2022) met à disposition un dépôt *Github* contenant des scripts Python afin d'importer et de réentraîner le modèle selon le contexte choisi. Nous avons opté pour la version *nano* du modèle (YOLOv5n), étant donné que l'objectif du projet est son deploiement sur un circuit FPGA. La version YOLOv5n se distingue par un nombre de paramètres très réduit (1 million de paramètres contre 867 millions de paramètres pour YOLOv5x), ce qui permet une exécution rapide et peu coûteuse en ressources matérielles. Bien que ses performances soient légérement inférieures à celles de la version la plus large, elles restent suffisantes pour satisfaire les contraintes de ce projet, particulièrement celles liées à l'accélération matérielle.

L'entraînement se fait à l'aide du script train.py fourni dans le dépôt officiel de YOLOv5 (Ultralytics, 2022). Il est possible de lancer ce script via une commande Linux en fournissant les arguments nécessaires. Par exemple, pour entraîner le modèle YOLOv5n avec la base de données COCO2017 pendant 300 epochs avec une taille de lot de 128 images, la commande est la suivante :

```
# Train YOLOv5n on COCO2017 for 300 epochs
python train.py --data coco.yaml --epochs 300 --weights '' \
                --cfg YOLOv5n.yaml --batch-size 128
```

Cette fonction admet plusieurs paramètres :

— **data** < *fichier.yaml* >: Contient l'architecture et la description de la base de données. (le path vers les images d'entraînement et de validation, le nombre de classes nc et les noms des classes par exemple).

— **cfg** < *fichier.yaml* >: Définit l'architecture du modèle qu'on souhaite créer (l'empilement de convolutions, modules C3, spatio-temporels...). Dans notre cas, le fichier yaml correspond à celui du YOLOv5n. Ce fichier précise les paramètres du réseau de convolution (nombre de canaux, nombre de détecteurs par grille, stride, etc.).

— **weights** < *chemin* >: Spécifie le chemin vers les poids pré-entraînés. Ne pas fournir de valeur pour ce paramètre signifie que le modèle sera entraîné sans poids pré-entraînés. Il est recommandé d'utiliser des poids pré-entraînés car cela accélère l'entraînement et stabilise la convergence.

— **batch-size** < *int* >: Correspond au nombre d'images traitées simultanément à chaque itération. Un batch large (64–128) permet de mieux stabiliser le gradient (moyenne sur plus d'exemples) au prix d'une plus grande consommation de VRAM.

— **epochs** < *int* >: Correspond au nombre de passages complets pendant l'entraînement. Pendant chaque passage, le modèle reçoit un lot de données de taille égale au batch size.

Pendant l'entraînement, l'objectif principal est de trouver les bons paramètres optimaux (valeurs des poids/biais) qui minimisent la fonction de perte. Cette dernière mesure l'écart

entre les prédictions du modèle et les valeurs vraies (réelles). Pour cela, on utilise souvent l'algorithme de descente du gradient (Ruder, 2017; Maurizio et al., 2024). L'idée derrière cet algorithme est de suivre pas à pas la pente (gradient) de la surface de la fonction de perte jusqu'à ce qu'elle atteigne un minimum, idéalement global, mais souvent local.

Une fois ces paramètres initialisés, le script `Train.py` passe à la phase de préparation des données. Durant cette phase, le fichier YAML (correctement formaté) est utilisé afin de créer deux DataLoaders (Pytorch, 2020): le premier DataLoader est dédié à l'entraînement, le second à la validation. Plusieurs techniques d'augmentation sont utilisées durant l'entraînement (mosaic, mixup, variations aléatoires de teinte/saturation/luminosité, flip horizontal/vertical, et même changement d'échelle (multi-scale) si cela est actif). Les DataLoaders ont aussi pour tâche de répartir les images en lots (mini-batches), qui peuvent éventuellement être mis en cache en RAM ou sur disque afin d'accélérer les accès. Les images et leurs annotations sont ensuite redimensionnées à la taille spécifiée, dans notre cas 640x640, avant d'être normalisées dans l'intervalle [0, 1]. Une fois le fichier YAML chargé, on vérifie sa compatibilité avec l'Automatic Mixed Precision (AMP) (Pytorch, 2020), une technique qui permet d'exécuter les convolutions en float16. Cela permet de réduire l'utilisation de la mémoire tout en conservant une précision de détection suffisante.

Ensuite, en fournissant un fichier checkpoint.pt, celui-ci est chargé dans l'architecture YOLOv5n composée de blocs (CSP, SPPF et têtes d'ancrage) afin de réentraîner le modèle. Lors de l'entraînement, les premières centaines `epochs` vont servir à calibrer le taux d'apprentissage (Learning rate) et le momentum pour éviter les risques d'instabilité. Par la suite, lors de la passe avant (Forward pass), chaque mini-batch est envoyé au processeur graphique GPU pour réaliser la prédiction (Classification et calcul de la boîte englobante). Ensuite, la perte combinée (box loss + objectness loss + classification loss), comme mentionné dans la section 3.5.2, est calculée. La rétro-propagation (Backward pass) s'effectue en semi-précision (float16 au lieu de float32) grâce à l'utilisation d'AMP (Automated Mixed Precision). Cette technique réduit considérablement le temps de calcul tout en maintenant

la précision des opérations critiques. Une fois la rétropropagation terminée, une limitation de la norme du gradient est appliquée afin d'éviter une explosion du gradient (Philipp et al., 2018) et ainsi stabiliser l'entraînement des poids. La figure 42 illustre le mécanisme de descente de gradient : la fonction de coût est représentée en jaune, et l'on part d'un poids initial situé à droite. La pente (ou dérivée du coût) détermine la direction de mise à jour à l'aide du gradient (ligne en pointillés). À chaque itération, les poids sont ajustés (points bleus) jusqu'à atteindre le minimum global (point vert), qui correspond à la valeur optimisant la fonction de coût. Finalement, l'optimiseur met à jour les poids, et le script évalue le modèle en utilisant les données présentes dans le dossier `Val2017`. Cette évaluation implique un calcul de `mAP (Mean Average Precision) à IoU=0.5 et à IoU=0.5:0.95`, selon les métriques COCO. À la fin de l'entraînement, les meilleurs résultats sont enregistrés dans un fichier `best.pt` pour une évaluation finale. Dans le cas où le script constate qu'il n'y a plus d'améliorations à faire, un mécanisme d'arrêt anticipé, paramétré via l'argument `--patience`, interrompt l'entraînement et enregistre le modèle dans un fichier `Last.pt`.

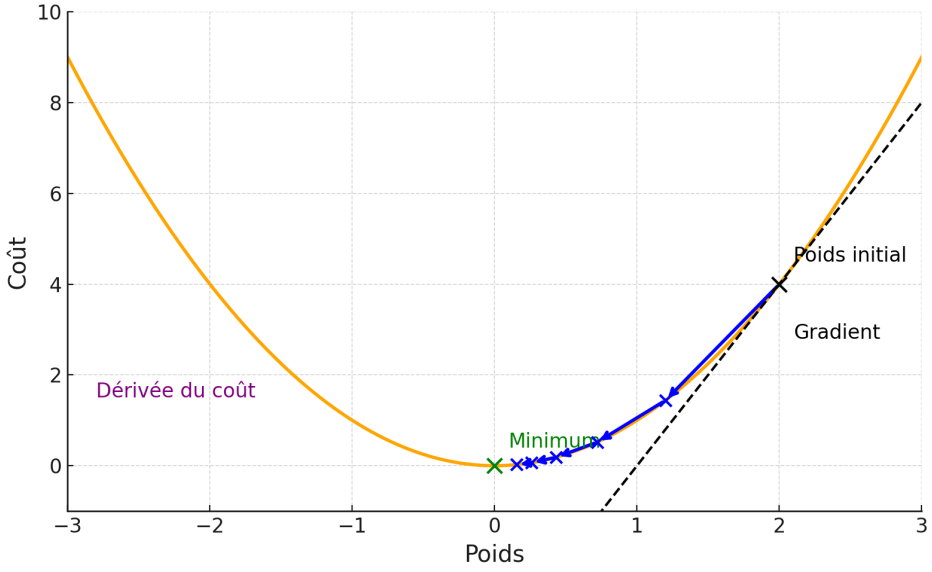


Figure 42: Algorithme de la descente du gradient en apprentissage machine

Un dernier point important concerne l'utilisation d'opérateurs pris en charge par Vitis AI 3.0. En effet, la version 3.0 de Vitis AI (AMD-XILINX, 2023) ne supporte qu'un nombre limité de fonctionnalités. Par défaut, YOLOv5 est entraîné avec la fonction d'activation SiLU, ainsi que des opérations de manipulation de tenseurs telles que `permute` et `view`, qui ne sont malheureusement pas prises en charge par Vitis AI 3.0. Il faut donc les retirer ou les modifier. Dans ce projet, l'entraînement de YOLOv5n a été effectué avec la fonction d'activation Leaky ReLU, l'une des meilleures fonctions d'activation supportées par Vitis-AI 3.0. Si l'on décide de garder les paramètres de base du modèle, les opérations non prises en charge seront transmises directement au CPU lors de la compilation, ce qui peut engendrer une latence très élevée. Or, l'objectif principal de l'accélération matérielle est d'envoyer toutes les opérations au DPU (Deep Processing Unit) integré à la plateforme Kria KV260. Nous expliquerons le fonctionnement ainsi que le rôle du DPU dans la sous-section suivante.

Par ailleurs, trois versions du modèle YOLOv5 ont été entraînés dans le cadre de ce projet: `YOLOv5n` (nano), `YOLOv5s` (small) et `YOLOV5l` (large), pour comparer les performances respectives et choisir le modèle à déployer.

4.3.1 Résultats de l'entraînement de `YOLOv5n` avec la base de données dérivée de HAM10000

Pour la base de données dérivée de HAM10000 et annotée avec Roboflow, la commande `val.py` permet d'évaluer le modèle entraîné en utilisant les données du répertoire `val`. Les résultats obtenus sont présentés dans les tableaux 7 et 8 et dans les figures 43 à 47.

Dans le tableau 7, la colonne Images indique le nombre total d'images utilisées pour tester le modèle. Par exemple, on observe une valeur de 410 aussi bien pour « Toutes classes » que pour « Cas bénin » et « Cas malin » : cela signifie que l'évaluation a été réalisée sur les mêmes 410 images au total, indépendamment de la classe considérée.

La colonne Instances précise le nombre d'objets annotés avec les boîtes englobantes

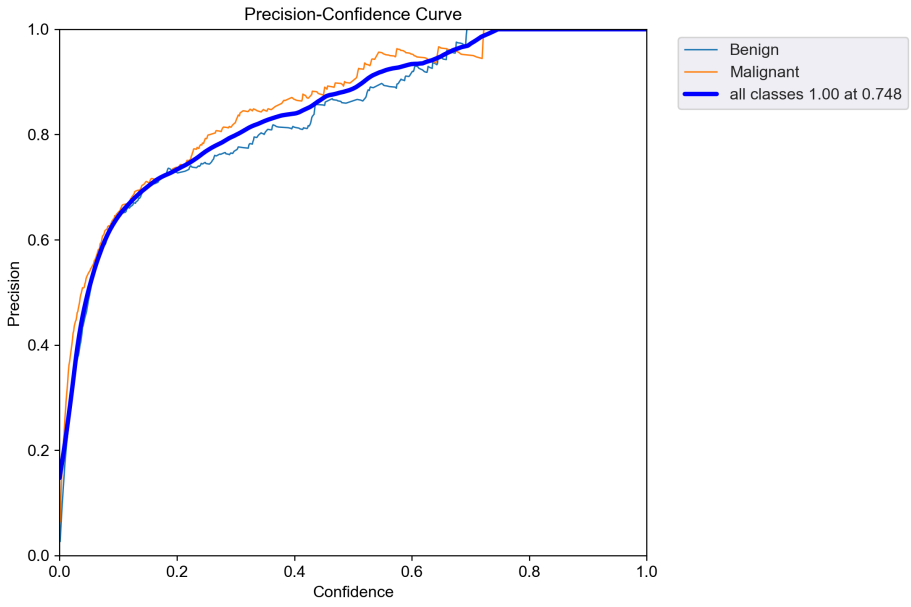


Figure 43: Courbe de précision du modèle `YOLOv5n` sur la base de données du cancer de la peau dérivée de HAM10000.

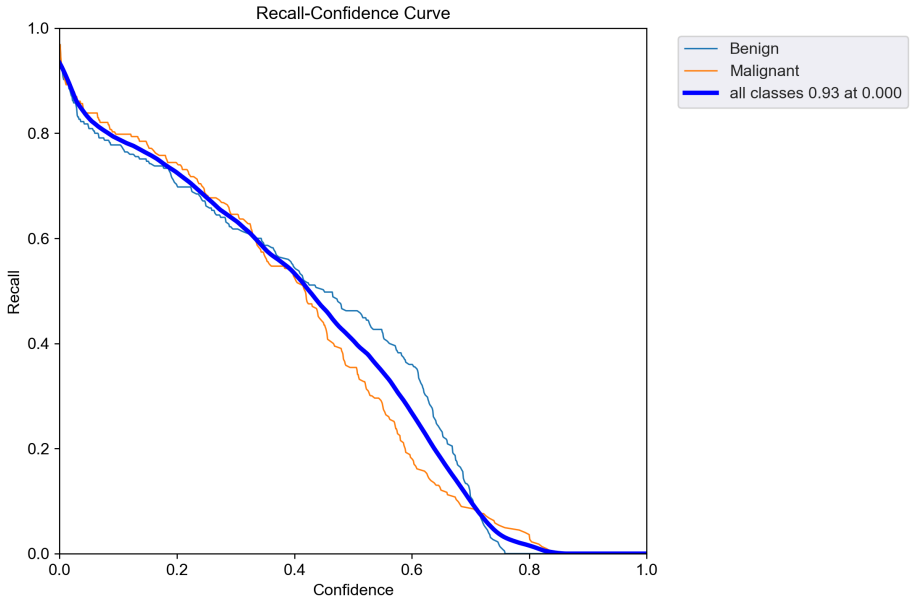


Figure 44: Courbe de rappel du modèle `YOLOv5n` sur la base de données du cancer de la peau dérivée de HAM10000.

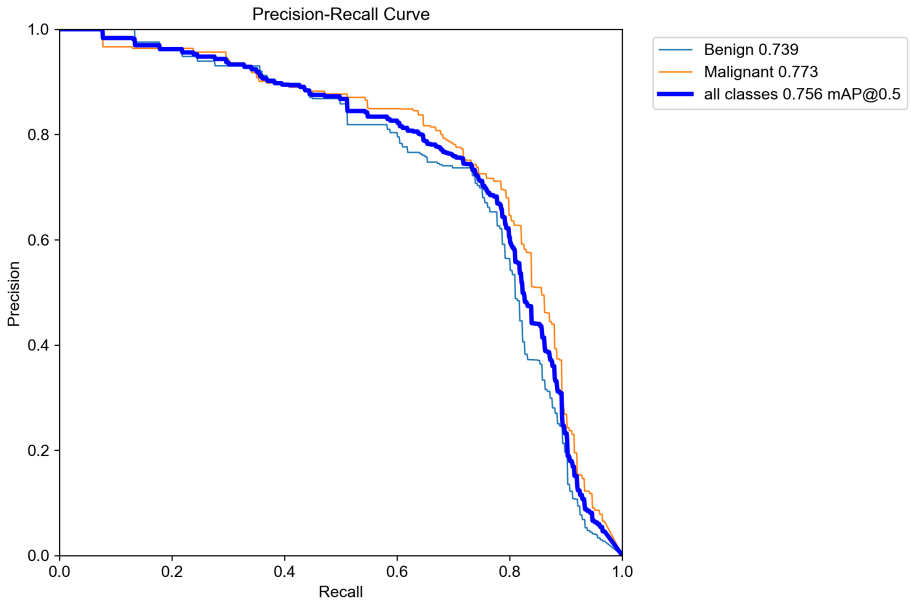


Figure 45: Courbe de précision-rappel du modèle `YOLOv5n` sur la base de du cancer de la peau dérivée de HAM10000.

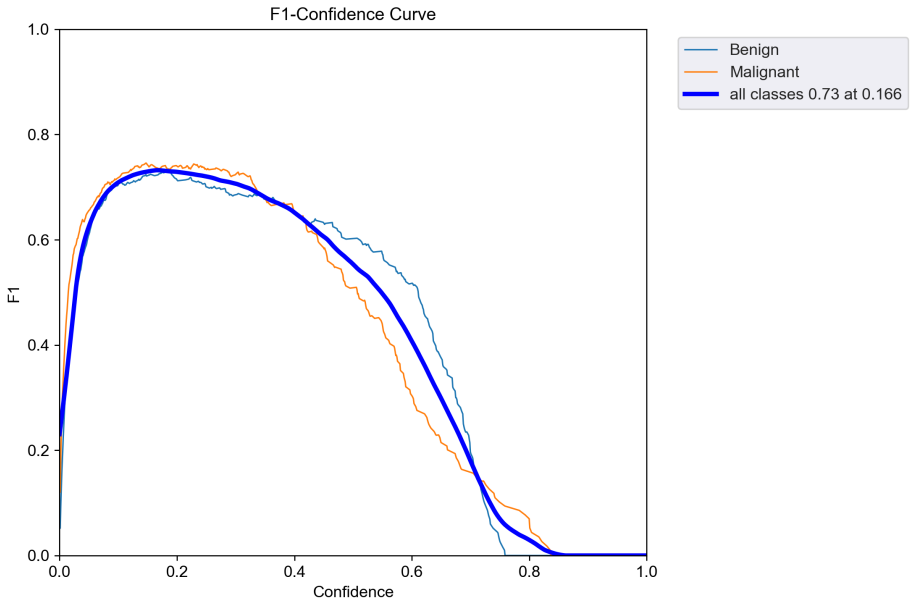


Figure 46: Courbe du score F1 du modèle `YOLOv5n` sur la base de données du cancer de la peau dérivée de HAM10000.

Table 8: Performances du modèle YOLOv5n sur la base de données du cancer de la peau dérivée de HAM10000.

Classe	**Images**	**Instances**	**Précision**	**Rappel**	**F1 @0.166**	**mAP@0.5**	**mAP@0.5–0.595**
Toutes classes	410	448	0.826	0.790	0.73	0.818	0.431
Cas bénin	410	225	0.835	0.760	0.72	0.790	0.360
Cas malin	410	223	0.817	0.821	0.74	0.846	0.502

Table 9: Temps de traitement de YOLOv5n sur la base de données du cancer de la peau dérivée de HAM10000.

Pré-traitement (ms)	**Inference (ms)**	**NMS (ms)**	**Shape (N, C, H, W)**
0.1	0.9	0.6	(32, 3, 416, 416)

présentes dans ces images. On compte ainsi 448 lésions au total, dont 225 bénignes et 223 malignes. Ces valeurs permettent de comprendre la répartition des annotations dans l'ensemble de test.

Étant donné que c'est la première discussion des résultats, il est important de comprendre comment analyser les courbes des métriques d'évaluation. En effet, les deux premières courbes (figures 43 et 46), nommées Precision-Confidence (P) et Recall-Confidence (R), montrent l'évolution de la précision et du rappel en fonction du seuil de confiance défini. Lorsque le seuil est proche de zéro, le modèle accepte toutes les prédictions, à savoir beaucoup de vrais positifs, ce qui explique pourquoi le rappel est élevé. Cependant, plusieurs faux positifs sont acceptés, ce qui explique pourquoi la précision est proche de zéro. Par la suite, plus le seuil de confiance augmente, plus le modèle devient confiant. Par conséquent, il aura tendance à filtrer les faux positifs, ce qui explique pourquoi la précision augmente, tandis que certains vrais positifs ne sont plus détectés, ce qui explique la diminution du rappel. Afin de synthétiser le compromis entre la précision et le rappel, le score F1 est couramment utilisé. Ce score est défini comme la moyenne harmonique de la précision et du rappel, permet de résumer le compromis entre ces deux métriques. Il atteint généralement son maximum pour

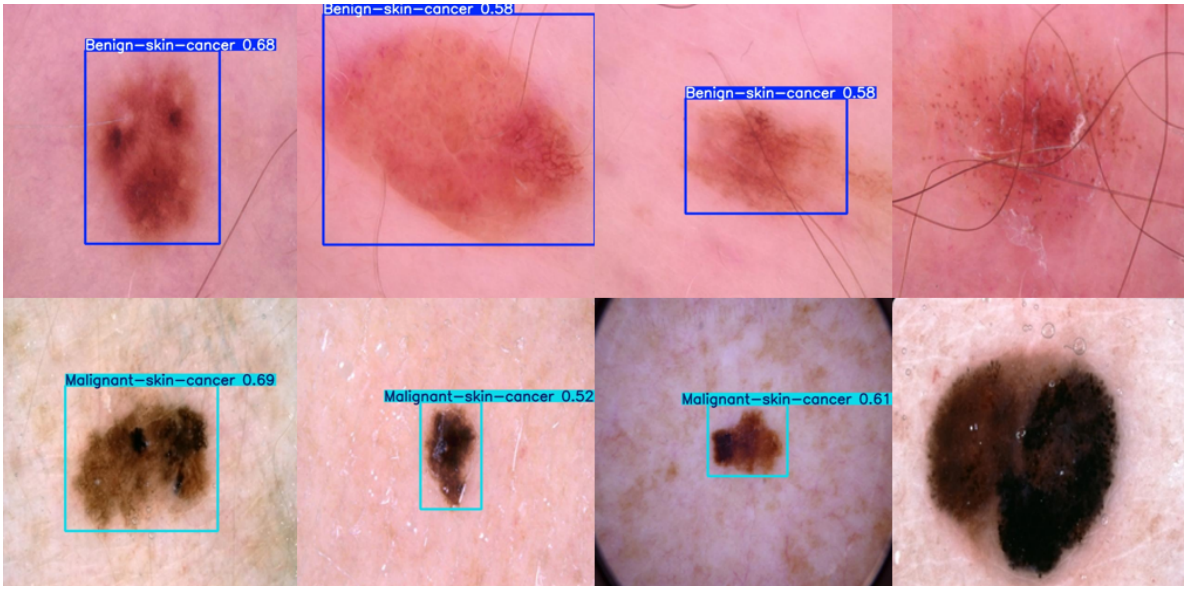


Figure 47: Exemple de prédiction du modèle `YOLOv5n` sur la base de données du cancer de la peau dérivée de HAM10000.

un seuil de confiance intermédiaire, correspondant à un point de fonctionnement optimal du modèle. Un score F1 élevé indique que le modèle parvient à détecter correctement un grand nombre d'instances tout en limitant les fausses détections. Le maximum du score F1 est généralement atteint pour un seuil de confiance intermédiaire, représentant ainsi un point de fonctionnement optimal du modèle selon ce critère.

La courbe Precision-Recall (figure 45) est une autre représentation importante: l'axe des abscisses est le rappel et l'axe des ordonnées est la précision. Ici encore, l'évolution est basée sur le seuil de confiance. Pour un seuil de confiance élevé, le rappel est nul et la précision est élevée. Lorsque le seuil de confiance diminue, le rappel augmente, au détriment de la précision. D'où l'allure de la courbe obtenue. Il faut aussi garder en tête que ces courbes sont discontinues, car elles sont calculées sur un nombre limité de seuils. Cependant, nous relions les points pour mettre en évidence l'évolution des métriques.

Sur l'ensemble de données de validation (`val`), le modèle YOLOv5n atteint un mAP@0.5 de 0.818. Autrement dit, pour un seuil IoU supérieur ou égal à 50%, le modèle est

capable de détecter environ 82% des lésions.

— Pour la classe ”Malignant”, on obtient une précision (P) de 0.817 et un rappel (R) de 0.821, ce qui représente un très bon compromis. La majorité des cas de cancer sont détectés, avec un faible taux de faux négatifs (FN), et les fausses alertes sont considérablement limitées.

— Pour la classe ”Benign”, la précision est de 0.835 tandis que le rappel est un peu plus faible, avec une valeur de 0.76. Cela signifie qu'une partie des cas bénins n'est pas correctement détectée (voir figure 47) ou bien classée à tort comme Malins, ce qui représente un point à améliorer afin d'éviter les faux diagnostics de cancer (considérer des cas bénins comme atteints de cancer).

Par ailleurs, lorsqu'on calcule le mAP@0.5:0.95, c'est-à-dire une évaluation plus sévère sur une plage de seuils IoU allant jusqu'à 0.95, le score chute 0.431. Cet écart entre mAP@0.5 et mAP@0.5–0.95 signifie que les boîtes englobantes générés sont suffisamment précises pour un seuil modéré, mais insuffisantes pour un seuil très sévère. En pratique, un seuil IoU très élevé n'est pas facile à atteindre, mais, dans le cas d'une chirurgie médicale assistée par l'IA, il est important de respecter les seuils sévères. Par conséquent, il faudra retravailler cette partie, c'est-à-dire obtenir plus de données afin d'améliorer les performances du modèle. Les résultats obtenus montrent un score F1 maximal de 0.73 pour un seuil de confiance de 0.166. Ce niveau de performance traduit un compromis satisfaisant entre la précision et le rappel pour la classification des lésions bénignes et malignes. Dans le contexte de cette étude, ce résultat est jugé pertinent pour un modèle de détection léger et optimisé pour un fonctionnement en temps réel. Toutefois, bien que ce score démontre la capacité du modèle à identifier efficacement les lésions, il demeure insuffisant pour une utilisation clinique autonome.

Du point de vue de la vitesse, `YOLOv5n` se montre très rapide: environ 0.1 ms pour le prétraitement, 0.9 ms pour l'inférence et 0.6 ms pour la suppression des doublons (NMS) par image de taille 416×416 pixels. Pour résumer, cela représente un temps de traitement de 1.6 ms par image, ce qui constitue une excellente vitesse d'exécution en temps réel. Il

est donc possible d'obtenir des performances significatives tout en maintenant une latence négligeable.

4.3.2 Résultats de l'entraînement de YOLOv5s avec la base de données dérivée de HAM10000

Les résultats obtenus avec YOLOv5s sur la même base de données de cancer de la peau montrent une amélioration globale par rapport à la version YOLOv5n utilisée précédemment. Ces résultats sont présentés dans les tableaux 10 et 11 et les figures 48, 49, 50, 51 et 52.

Table 10: Performances du modèle YOLOv5s sur la base de données du cancer de la peau dérivée de HAM10000.

Classe	**Images**	**Instances**	**Précision**	**Rappel**	**F1 @0.267**	**mAP@0.5**	**mAP@0.5–0.595**
Toutes classes	410	448	0.836	0.797	0.74	0.825	0.435
Bénin	410	225	0.841	0.778	0.73	0.799	0.346
Malin	410	223	0.831	0.816	0.75	0.850	0.524

Table 11: Temps de traitement de YOLOv5s sur la base de données de cancer de la peau dérivée de HAM10000

Pré-traitement(ms)	**Inférence(ms)**	**NMS(ms)**	**Shape (N, C, H, W)**
0.1	1.4	0.5	(32, 3, 416, 416)

En validant sur 410 images de taille 416x416, le modèle YOLOv5s présente une amélioration globale au niveau des trois métriques d'évaluation. En effet, la précision moyenne (mAP@0.5) obtenue est de 0.825 pour l'ensemble des classes, avec une précision de 0.836 et un rappel de 0.797. Pour plus de détails, la classe Benign présente une précision de 0.841, un rappel de 0.778 et un mAP@0.5 de 0.799, tandis que la classe Malignant atteint une précision de 0.831, un rappel de 0.816 et un mAP@0.5 de 0.850. Pour un seuil IoU plus

sévère (mAP0.5-0.95), la classe Benign atteint une valeur de 0.346, tandis que la classe Malignant atteint la valeur de 0.524. Du point de vue de la vitesse, le temps moyen de traitement est de 2 ms par image (sur GPU) réparti comme suit: 0.1 ms pour le prétraitement, 1.4 ms pour l'inférence et 0.5 ms pour la phase de NMS.

Cependant, il est important de mentionner que le modèle `YOLOv5s` a des difficultés lorsque le seuil IoU est élevé. Cette faiblesse se ressent surtout avec la classe Benign, étant donné que son mAP0.5-0.95 reste limité à 0.346. On remarque aussi sur la figure 52 que des cas non détectés par le modèle YOLOv5n sont correctement détecté par la version YOLOv5s. Le score F1 atteint sa valeur maximale de 0.74 pour un seuil de confiance de 0.267. Ce seuil correspond au point de fonctionnement optimal du modèle, où le compromis entre la précision et le rappel est le plus équilibré pour l'ensemble des classes. Par rapport au modèle YOLOv5n, cette légère amélioration du score F1 reflète la capacité accrue du modèle YOLOv5s à mieux discriminer les lésions bénignes et malignes, au prix d'une complexité légèrement supérieure.

Cette amélioration globale des performances s'explique en grande partie par la différence significative du nombre de paramètres entre les deux versions de YOLOv5. La version `YOLOv5s` possède 7 millions de paramètres, tandis que la version `YOLOv5n` en possède seulement 1.7 million.

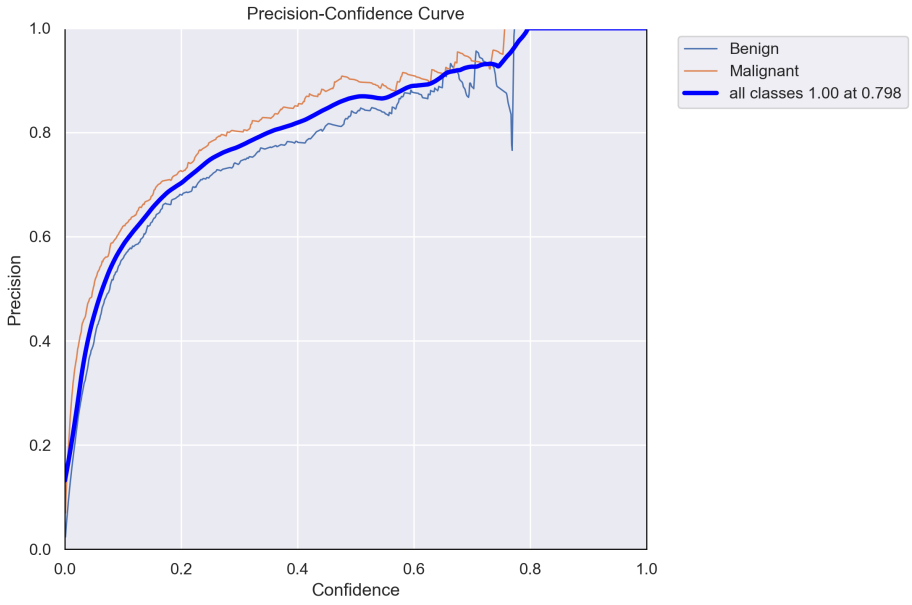


Figure 48: Courbe de précision du modèle `YOLOv5s` sur la base de données dérivée de HAM10000

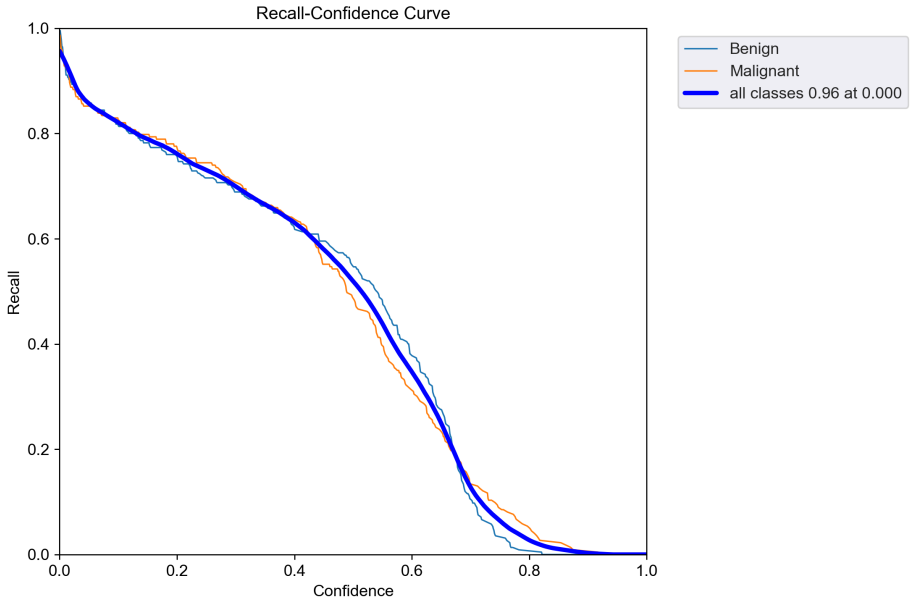


Figure 49: Courbe de rappel du modèle `YOLOv5s` sur la base de données dérivée de HAM10000

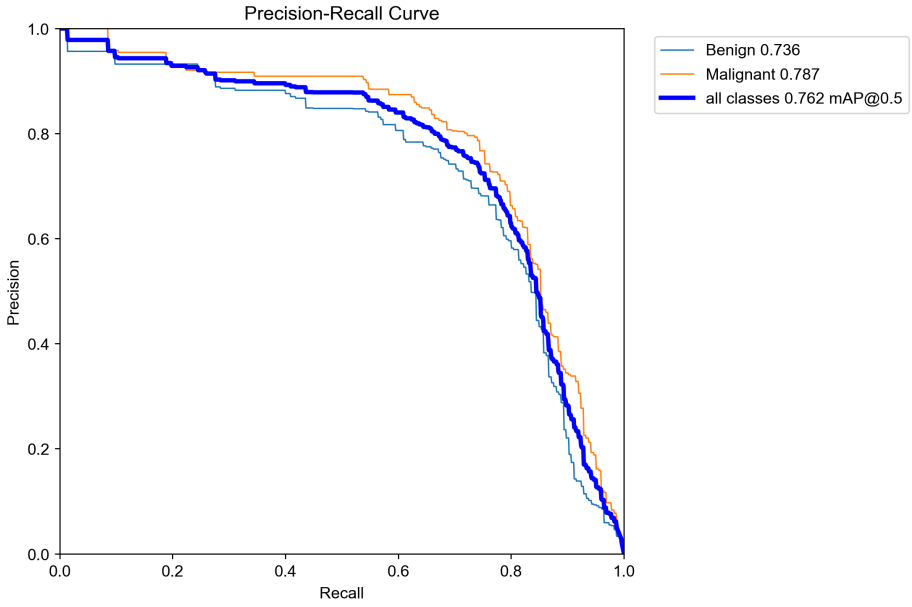


Figure 50: Courbe de précision-rappel du modèle `YOLOv5s` sur la base de données dérivée de HAM10000

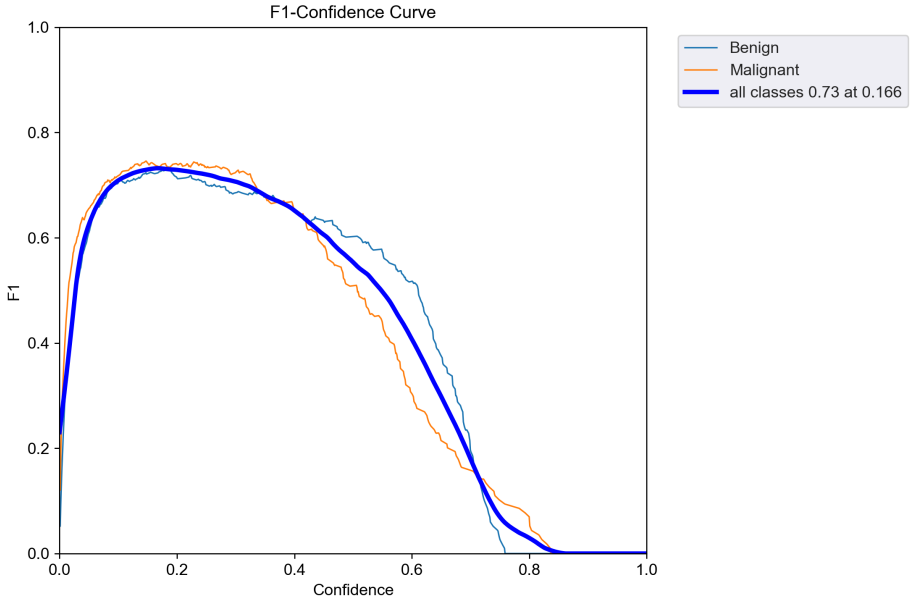


Figure 51: Courbe du score F1 du modèle `YOLOv5s` sur la base de données du cancer de la peau dérivée de HAM10000.

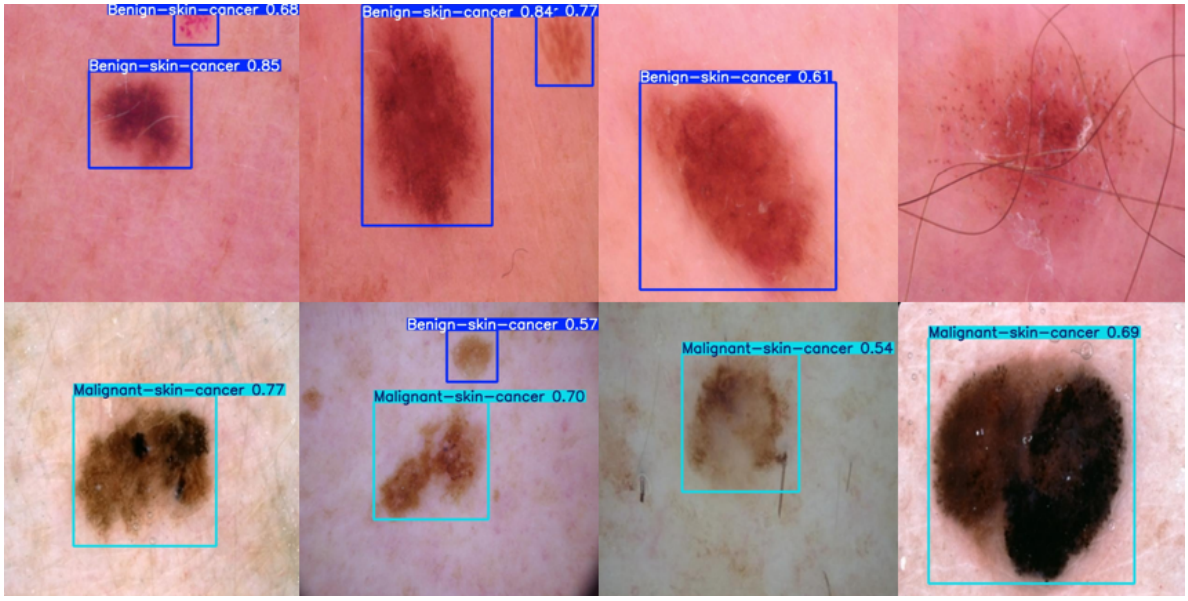


Figure 52: Exemple de prédiction du modèle `YOLOv5s` sur la base de données dérivée de HAM10000

4.3.3 Résultats de l'entraînement de `YOLOv5n` avec la base de données KITTI convertie au format YOLOv5

Dans cette sous-section, nous présentons les résultats de validation du modèle YOLOv5n évalué sur la base de données KITTI, préalablement convertie au format compatible avec YOLOv5. Les résultats correspondants sont illustrés dans les tableaux 12 et 13, ainsi que dans les figures 53, 54, 55, 56 et 57. Les métriques d'évaluation utilisées sont la précision moyenne à un IoU de 50% (mAP@0.5), le rappel (Recall) et la précision (Precision) pour chaque classe présente dans la base de données.

Table 12: Performances du modèle YOLOv5n sur la base de données KITTI

Classe	**Images**	**Instances**	**P**	**R**	**F1 @0.40**	**mAP@0.5**	**mAP@0.5–0.95**
Toutes	748	4072	0.898	0.773	0.82	0.858	0.541
Car	748	2873	0.935	0.885	0.91	0.959	0.711
Van	748	283	0.862	0.860	0.86	0.913	0.639
Truck	748	142	0.885	0.901	0.89	0.944	0.717
Pedestrian	748	415	0.854	0.713	0.78	0.818	0.393
Person_sitting	748	10	0.971	0.400	0.57	0.578	0.276
Cyclist	748	197	0.888	0.853	0.87	0.909	0.508
Tram	748	43	0.917	0.907	0.91	0.964	0.629
Misc	748	109	0.870	0.661	0.75	0.782	0.453

Table 13: Temps de traitement de YOLOv5n sur la base de données KITTI

Pré-traitement(ms)	**Inférence (ms)**	**NMS (ms)**	**Shape (N, C, H, W)**
0.1	1,5	0.3	(32, 3, 1216, 1216)

Sur l'ensemble de données KITTI, avec une taille d'entrée de 1216x1216, YOLOv5n offre un compromis intéressant entre précision et rapidité, ce qui le rend adapté aux applications embarquées en temps réel. Globalement, le modèle atteint une précision moyenne (mAP@0.5) de 85.8% et un rappel moyen de 77.3%, ce qui signifie que la majorité des objets sont correctement détectés avec un nombre limité de fausses alertes. Les classes *Car* et *Truck* détiennent les meilleures performances, avec respectivement 95.9% et 94.4% en mAP@0.5. Ce résultat s'explique par la forte présence et la relative homogénéité de ces véhicules dans la base de données KITTI. De même, la classe *Tram* affiche d'excellents scores avec un mAP@0.5 de 96.4% et un rappel de 90.7%. On peut dire que YOLOv5n parvient à détecter correctement les objets de grande taille et aux contours nets. En revanche, certaines classes, notamment *Person_sitting*, n'atteignent que 57.8% en mAP@0.5 et surtout 40% de rappel, ce

qui indique que plusieurs personnes assises ne sont pas détectées. Cette limitation est principalement dûe au faible nombre d'instances disponibles pour cette classe dans les données d'entraînement, ce qui implique, par conséquent, une faible performance. Le score F1 atteint une valeur maximale de 0.82 pour un seuil de confiance de 0.40. Ce seuil correspond au point de fonctionnement optimal du modèle. Ce résultat reflète de bonnes capacités de généralisation du modèle YOLOv5n sur une tâche de détection multi-classes en environnement routier, malgré la complexité et le déséquilibre inhérents aux différentes catégories d'objets (par exemple Car vs Person_sitting).

Au niveau de la vitesse, YOLOv5n se distingue par un temps de traitement (latence) extrêmement faible: 0.1 ms pour le prétraitement, 1.5 ms pour l'inférence et 0.3 ms pour la NMS par image, avec un batch-size de 16. Ces résultats confirment que le modèle est apte à un déploiement sur une plateforme embarquée telle que la carte Kria KV260, où la latence représente un facteur critique.

En résumé, YOLOv5n offre un excellent compromis entre la vitesse et la précision concernant les classes prédominantes de la base de données KITTI. Toutefois, afin d'améliorer la robustesse de détection pour les classes minoritaires, notamment les piétons et les personnes assises, un ajustement des données d'entraînement et des hyper-paramètres est donc nécessaire.

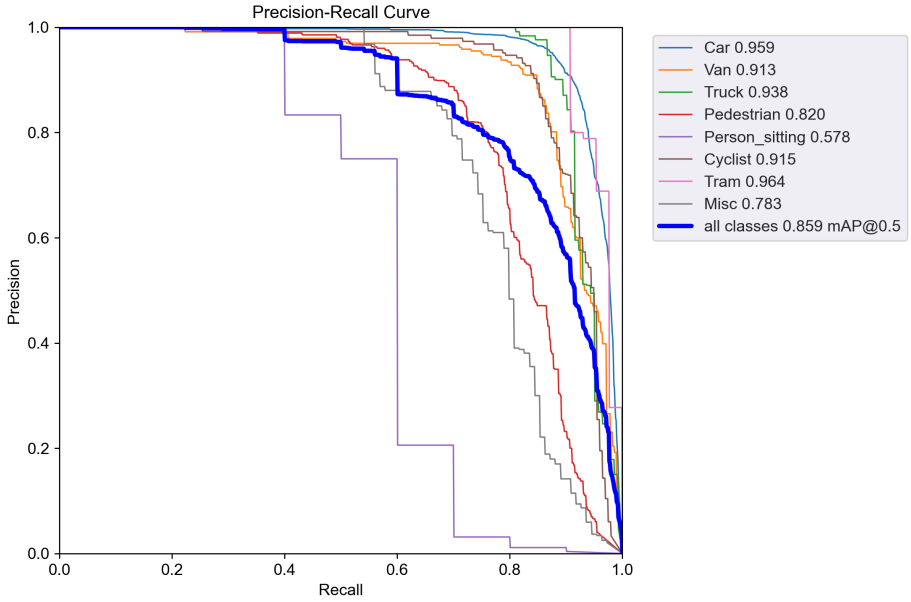


Figure 53: Courbe de précision du modèle `YOLOv5n` sur la base de données KITTI

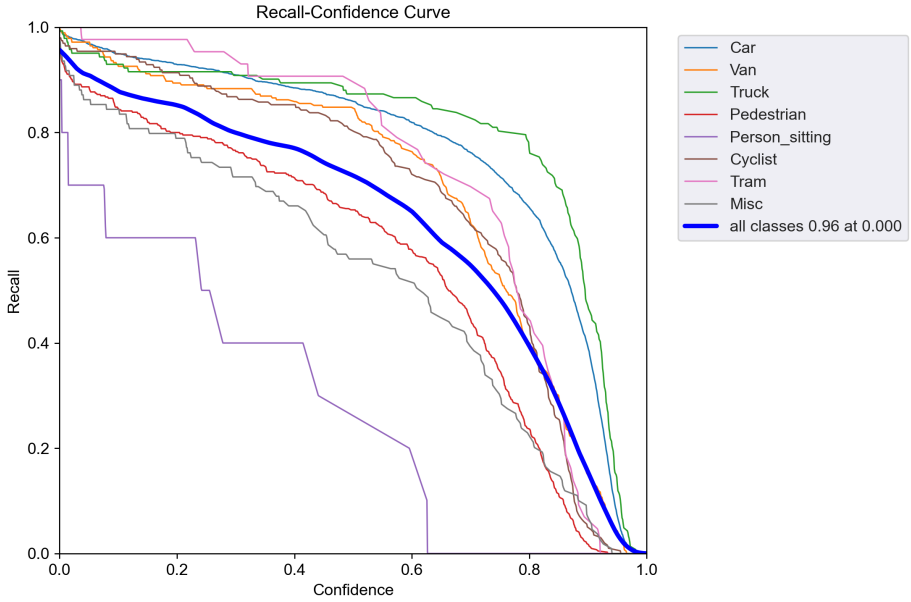


Figure 54: Courbe de rappel du modèle `YOLOv5n` sur la base de données KITTI

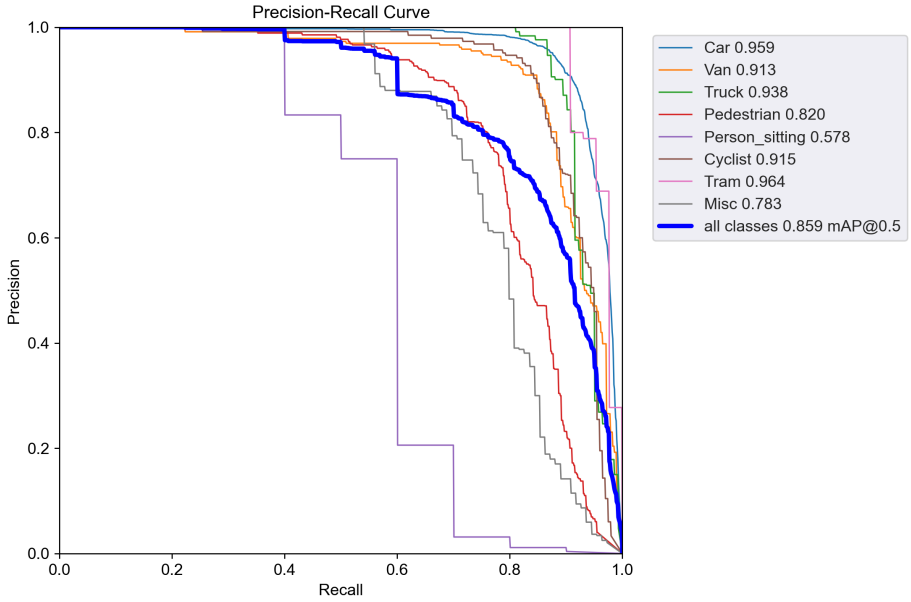


Figure 55: Courbe de précision-rappel du modèle `YOLOv5n` sur la base de données KITTI.

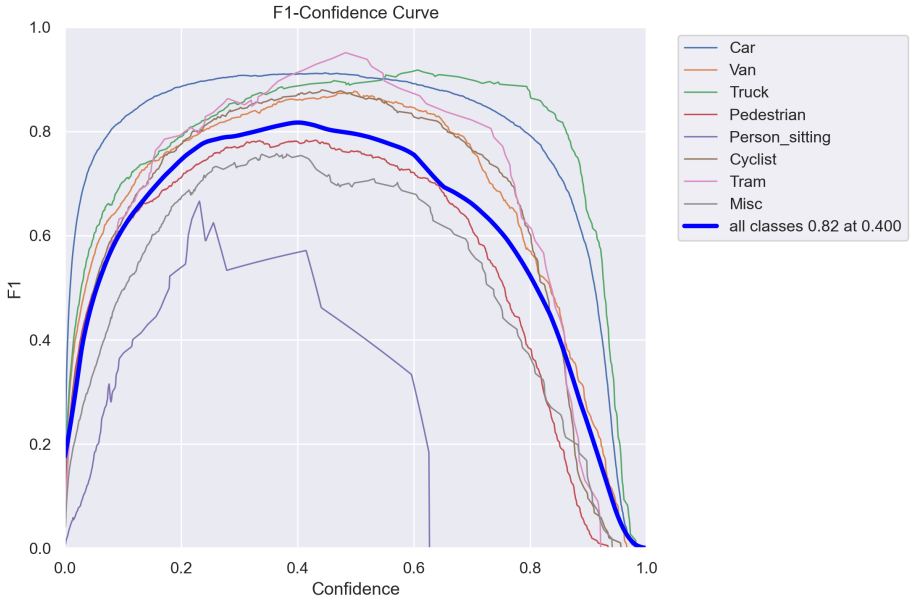


Figure 56: Courbe du score F1 du modèle `YOLOv5n` sur la base de données KITTI.

Figure 57: Exemple de prédiction du modèle `YOLOv5n` sur la base de données KITTI

4.3.4 Résultats de l'entraînement de `YOLOv5l` avec la base de données KITTI convertie au format YOLOv5

Dans cette sous-section, nous présentons les résultats de validation du modèle YOLOv5l évalué sur la base de données KITTI, préalablement convertie au format compatible YOLOv5. Les métriques d'évaluation utilisées sont la précision moyenne au seuil d'Intersection over Union (mAP@0.5), le rappel (Recall) et la précision (Precision) pour chaque classe présente dans la base de données.

D'après les résultats obtenus, illustrés aux figures 58 à 61, l'hypothèse selon laquelle le nombre de paramètres a un impact sur les performances du réseau est vérifiée. En effet, le modèle YOLOv5l ayant une résolution de 1216×1216, atteint une mAP@0.5 globale très élevée (91,5%), avec des scores élevés sur la majorité des classes, comme:

— **Car** : mAP@0.5 = 96,8 %, rappel = 92.6 %

— **Truck** : mAP@0.5 = 98,2 %, rappel = 95.3 %

— **Tram** : mAP@0.5 = 98,0 %, rappel = 97.7 %

Ces valeurs démontrent une excellente capacité du modèle à localiser et classifier les objets fréquemment présents dans les images, une capacité renforcée par une haute résolution d'entrée et la profondeur importante du réseau (107,8 GFLOPs). Cependant, certaines classes

Table 14: Performance du modèle `YOLOv5l` sur la base de données KITTI (taille d'entrée 1216x1216)

Classe	**Images**	**Instances**	**P**	**R**	**F1 @0.471**	**mAP@0.5**	**mAP@0.5–0.95**
Toutes	748	4072	0.834	0.769	**0.86**	0.843	0.549
Car	748	2873	0.877	0.913	0.89	0.949	0.732
Van	748	283	0.797	0.835	0.82	0.882	0.630
Truck	748	142	0.884	0.852	0.87	0.940	0.713
Pedestrian	748	415	0.846	0.682	0.75	0.788	0.417
Person_sitting	748	10	0.820	0.459	0.59	0.623	0.283
Cyclist	748	197	0.891	0.832	0.86	0.899	0.532
Tram	748	43	0.777	0.809	0.79	0.848	0.561
Misc	748	109	0.781	0.771	0.78	0.816	0.522

Table 15: Temps de traitement de `YOLOv5l` sur la base de données KITTI

Pré-traitement (ms)	**Inférence (ms)**	**NMS (ms)**	**Shape (N, C, H, W)**
0.1	11.9	0.3	(32, 3, 1216, 1216)

moins représentées dans la base de données KITTI posent problème. Par exemple, la classe *Person_sitting* affiche une précision parfaite (100 %), mais un rappel faible (43.3%), ce qui signifie que le modèle ne détecte qu'une partie des rares instances disponibles. De même, pour les classes *Pedestrian* (mAP@0.5 = 84.2%, rappel = 72.3%) et *Cyclist* (mAP@0.5 = 90.8%, rappel = 83.8%), cela traduit une difficulté du modèle à détecter les objets moins fréquents, probablement due à un nombre insuffisant d'exemples d'entraînement. Le score F1 atteint une valeur maximale de 0.86 pour un seuil de confiance de 0.471. Comparativement aux versions plus légères du modèle, YOLOv5l bénéficie d'une capacité de représentation accrue, ce qui se traduit par des performances globales élevées, en particulier pour les classes dominantes telles que Car, Truck et Tram. Les performances restent toutefois plus limitées

pour les classes faiblement représentées, comme Person_sitting, ce qui est cohérent avec le déséquilibre du jeu de données.

La comparaison entre la version Yolov5l et Yolov5n met en évidence le compromis entre la rapidité et la précision. Le modèle YOLOv5n (4.2 GFLOPs, 1216×1216) obtient une mAP@0.5 de 85.8% et un rappel de 77.3%. Les classes *Car* (mAP@0.5 = 95.9%, rappel = 88.5%) et *Truck* (mAP@0.5 = 94.4%, rappel = 90.1%) sont bien détectées, mais les performances baissent pour la classe *Pedestrian* (mAP@0.5 = 81.8%, rappel = 71.3%), la classes *Misc* (mAP@0.5 = 78.2%, rappel = 66.1%) et la classe *Van* (mAP@0.5 = 91.3%, rappel = 86.0%). Cette dégradation des performances s'explique par le fait qu'un réseau de petite taille a des difficultés à extraire des représentations suffisamment discriminantes, surtout pour des objets moins fréquents ou de petite taille, en raison du faible nombre de paramètres. En revanche, YOLOv5n présente un très faible temps d'inférence (1.5 ms), ce qui le rend adapté aux contraintes en temps réel, particulièrement sur une plateforme matérielle embarquée comme la carte Kria KV260.

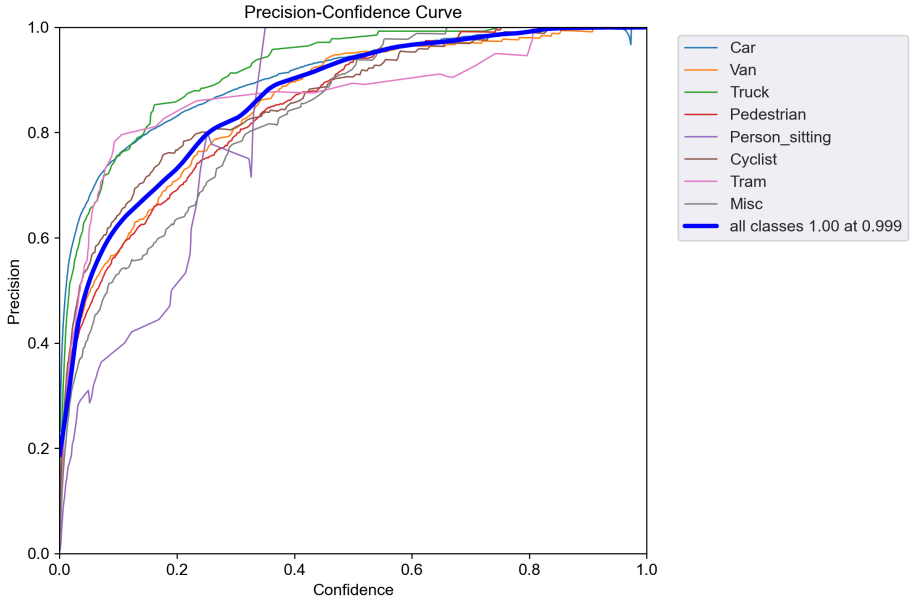


Figure 58: Courbe de précision du modèle `YOLOv5l` sur la base de données KITTI

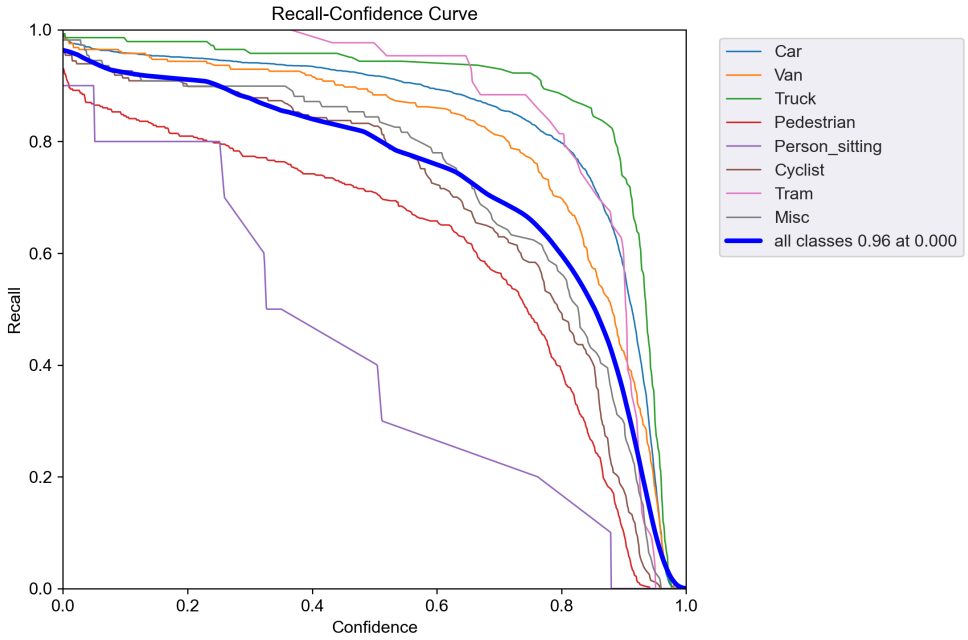


Figure 59: Courbe de rappel du modèle `YOLOv5l` sur la base de données KITTI

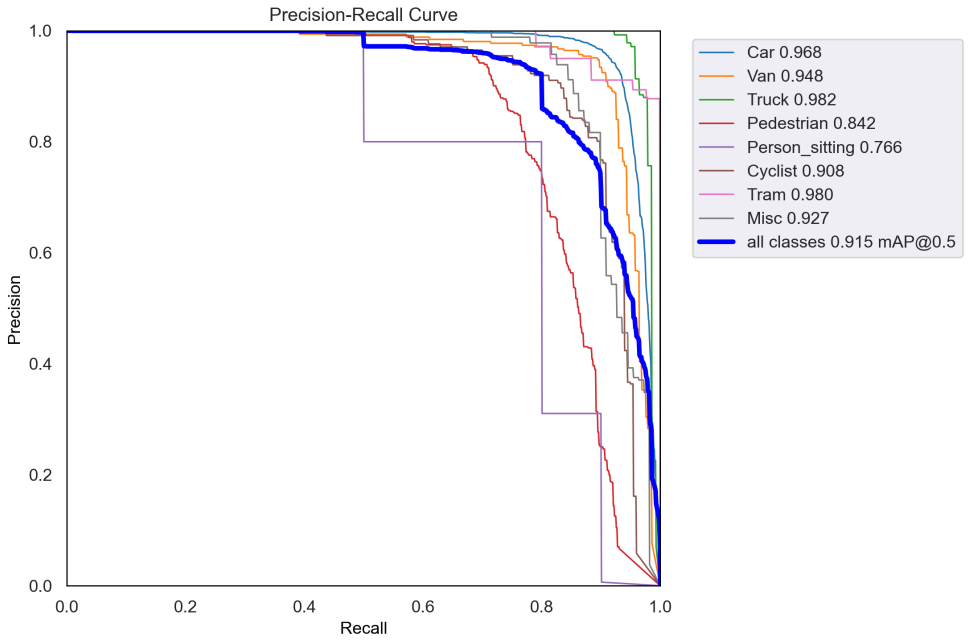


Figure 60: Courbe de précision-rappel du modèle `YOLOv5l` sur la base de données KITTI

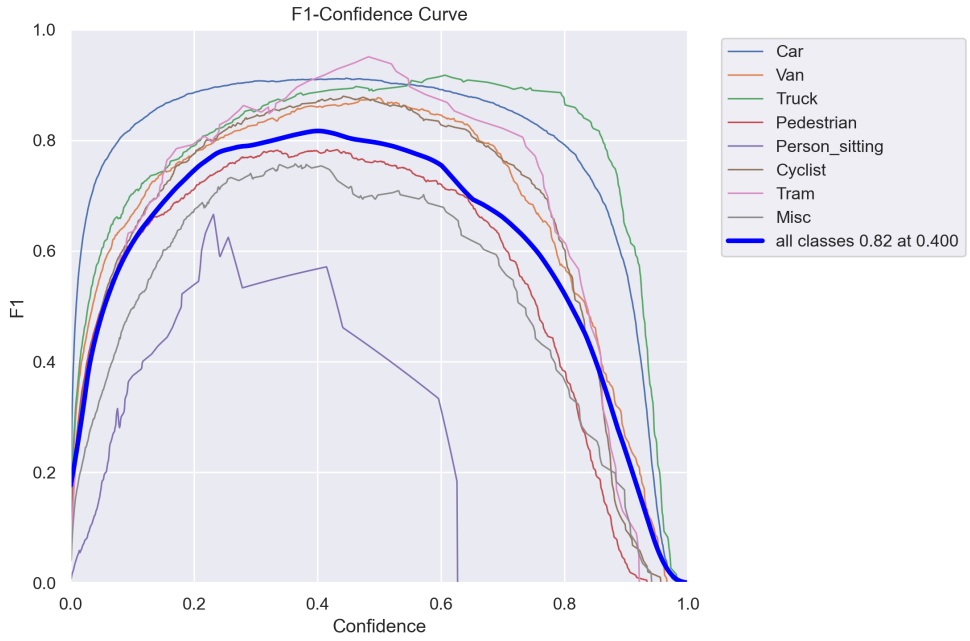


Figure 61: Courbe du score F1 du modèle `YOLOv5l` sur la base de données KITTI.

4.4 Choix de la plateforme et du logiciel d'accélération matérielle

La carte Kria KV260, illustrée dans les figures 62 et 63, est une plateforme matérielle à base de FPGA qui est généralement utilisée pour des applications d'intelligence artificielle. Elle est constituée de la version K26 d'un système sur module SOM (System on Module), d'une carte porteuse et d'une solution thermique. Le SOM intègre des composants matériels numériques de base, notamment un circuit MPSoC AMD Zynq™ UltraScale+™, de la mémoire vive, des dispositifs d'amorçage non volatils, une solution d'alimentation intégrée et un module de sécurité. La carte dispose de plusieurs périphériques d'application, notamment une variété d'entrées de caméra/capteur, de sorties d'affichage vidéo, d'interfaces physiques USB, de cartes SD et d'Ethernet (AMD-XILINX, 2023).

La Kria utilise un circuit SoC (ZYNQ) qui se base sur un circuit FPGA classique, jumelé à des processeurs ARM Cortex. Faisant partie de la famille UltraScale+ MPSoC, la

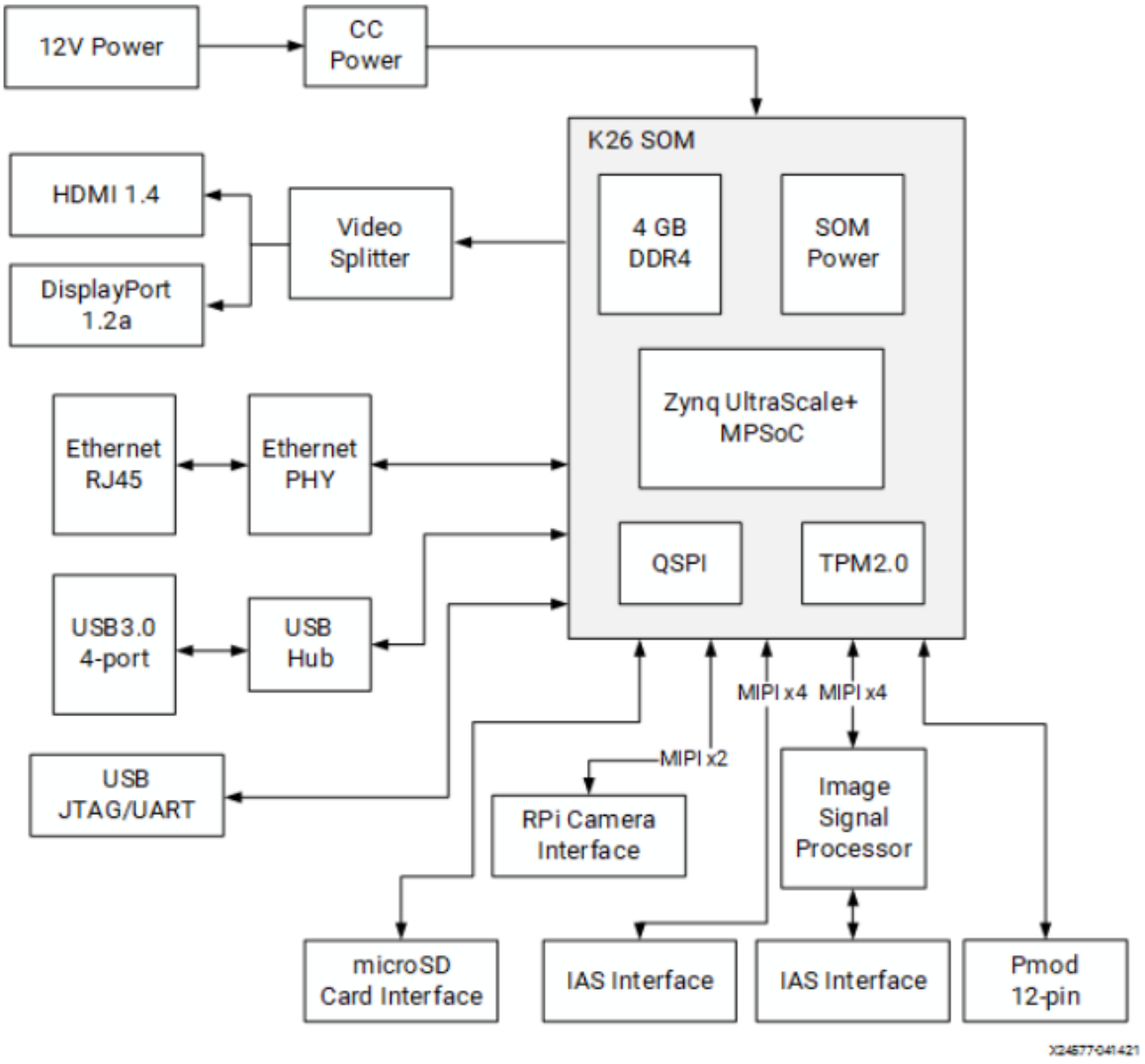


Figure 62: Diagramme de composants de la carte Kria KV260 (AMD-XILINX, 2023)

carte Kria KV260 est constituée principalement de deux sections:

— La partie PS (Processing System), qui correspond à l'unité de traitement, est un processeur composé de 4 cœurs ARM Cortex-A53 configurés pour fonctionner en mode SMP (symmetric multi-processing) Linux. Cela permet à la carte de fonctionner, ou plus précisément de démarrer (`booter`) via des images Linux. La partie PS de la carte KV260 est également chargée de fournir une horloge de fréquence égale à 100 MHz à la partie PL (Programmable Logic), ce qui est nécessaire à son fonctionnement. De plus, la partie PS comprend une interface Display Port (DP) pour la sortie vidéo, ainsi qu'un contrôleur de mémoire DRAM permettant à l'utilisateur de gérer la communication entre le SoC et la DRAM de 4 Go (AMD, 2021).

— La partie PL (Programmable Logic) constitue la logique programmable (matérielle). Elle regroupe les ressources matérielles comme la mémoire on-chip, les blocs DSP, les LUTs,etc, qui peuvent être pilotés par la partie PS.

Un dernier élément important présent sur la carte Kria KV260 et qui représente la ressource principale utilisée et exploitée dans ce projet est le DPU (Deep Learning Processor Unit) d'AMD. Il s'agit d'un bloc IP conçu et optimisé par AMD pour exécuter les noyaux clés des réseaux neuronaux convolutifs (CNN) sur circuit FPGA. La carte Kria KV260 possède un DPU appartenant à la catégorie DPUCZDX8G qui est conçu pour fonctionner avec les plateformes UltraScale+ MPSoC. Ce moteur de calcul offre un degré de parallélisme configurable ajustable dépendamment des besoins de calculs. Son objectif final est de permettre à la carte Kria d'exécuter l'inférence des réseaux de neurones convolutifs (CNN) en périphérie (edge computing). Le DPU est présent sur la partie PL du circuit et communique directement avec la partie PS via une interface AXI (Advanced eXtensible Interface). Ce bloc exécute des microcodes compilés (séquences de 0 et de 1) issus des réseaux de neurones, et nécessite des emplacements mémoire pour la lecture des entrées (images) ainsi que pour le stockage des données d'entrées/sorties temporaires (AMD-XILINX, 2023):

Le DPU utilise plusieurs ressources du circuit FPGA, telles que:

— Les slices DSP pour les opérations multiplication accumulation massivement parallèles.

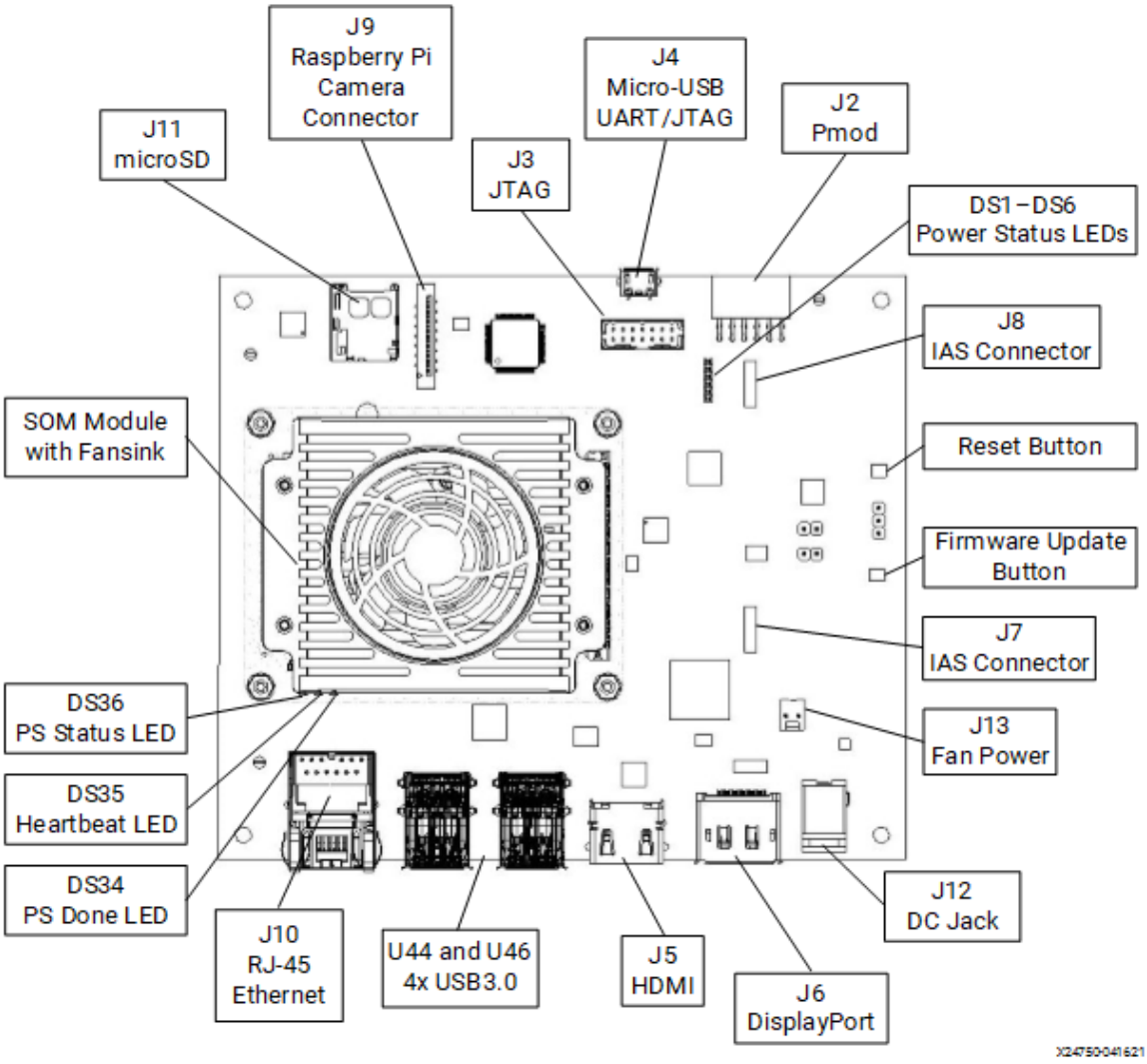


Figure 63: Interfaces et connecteurs présents sur la carte Kria KV260 ((Xilinx, 2023))

— Les blocs RAM/UltraRAM pour le stockage des cartes de caractéristiques intermédiaires et des poids.

— Les bus AXI DMA pour le transfert de données à large bande passant vers ou depuis la mémoire DDR externe.

— La logique du micro contrôleur pour distribuer les instructions micro codées.

Le DPU présente des avantages sur trois échelles:

1. Débit: il peut gérer des centaines, voire des milliers d'opérations MAC en parallèle et permet d'obtenir des taux de rafraîchissement (FPS) compatibles avec le temps réel.
2. Latence: L'utilisation de mémoires intégrées, de DSPs et de pipeline permet de réduire les délais d'inférence de bout en bout.
3. Efficacité énergétique: Comparé à l'exécution sur un CPU ou un GPU, le DPU utilise les ressources du FPGA plus efficacement, en particulier pour les CNN en virgule fixe.

Grâce à ce composant, illustré dans les figures 64 et 65, la carte peut exécuter des modèles CNN directement sur le FPGA, réduisant la dépendance à un CPU/GPU central et permettant des applications temps réel. Le DPU est exploitable via l'outil Vitis AI, qui permet de compiler des modèles CNN (PyTorch dans le cadre de ce projet) vers un format compatible avec le DPU, générant automatiquement le microcode et les fichiers nécessaires pour l'inférence sur la partie PL.

La prochaine étape consiste à choisir les outils logiciels adaptés pour optimiser notre réseau de neurones entraîné.

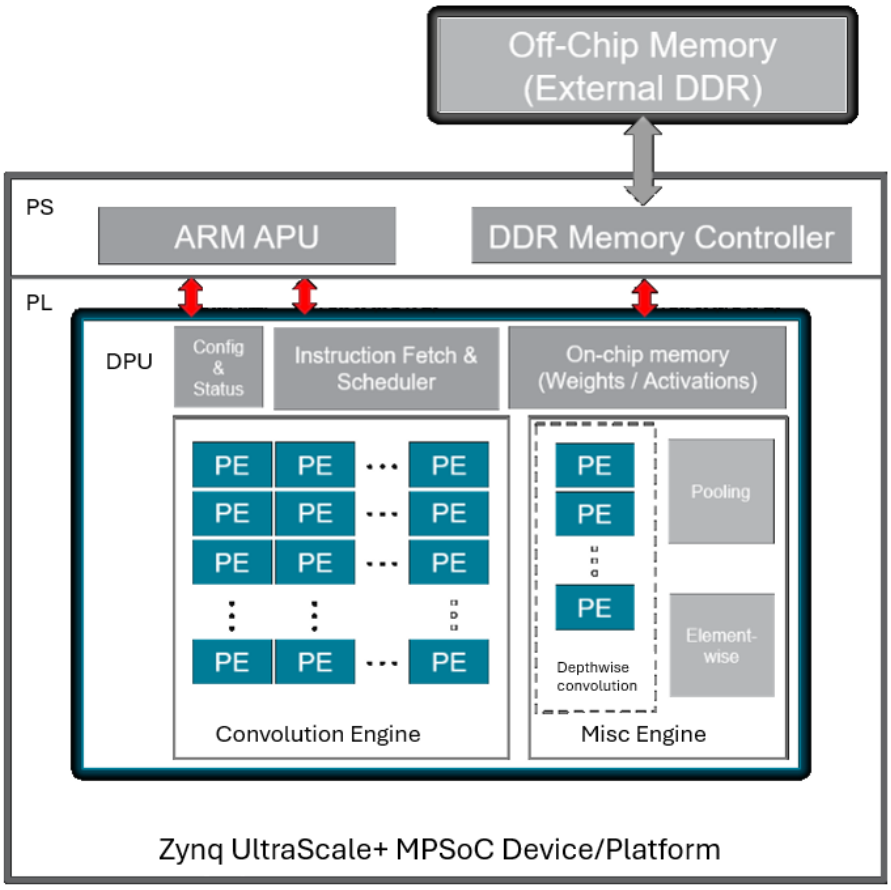


Figure 64: Schéma fonctionnel haut niveau de l'unité matérielle DPUCZDX8G (AMD-XILINX, 2023)

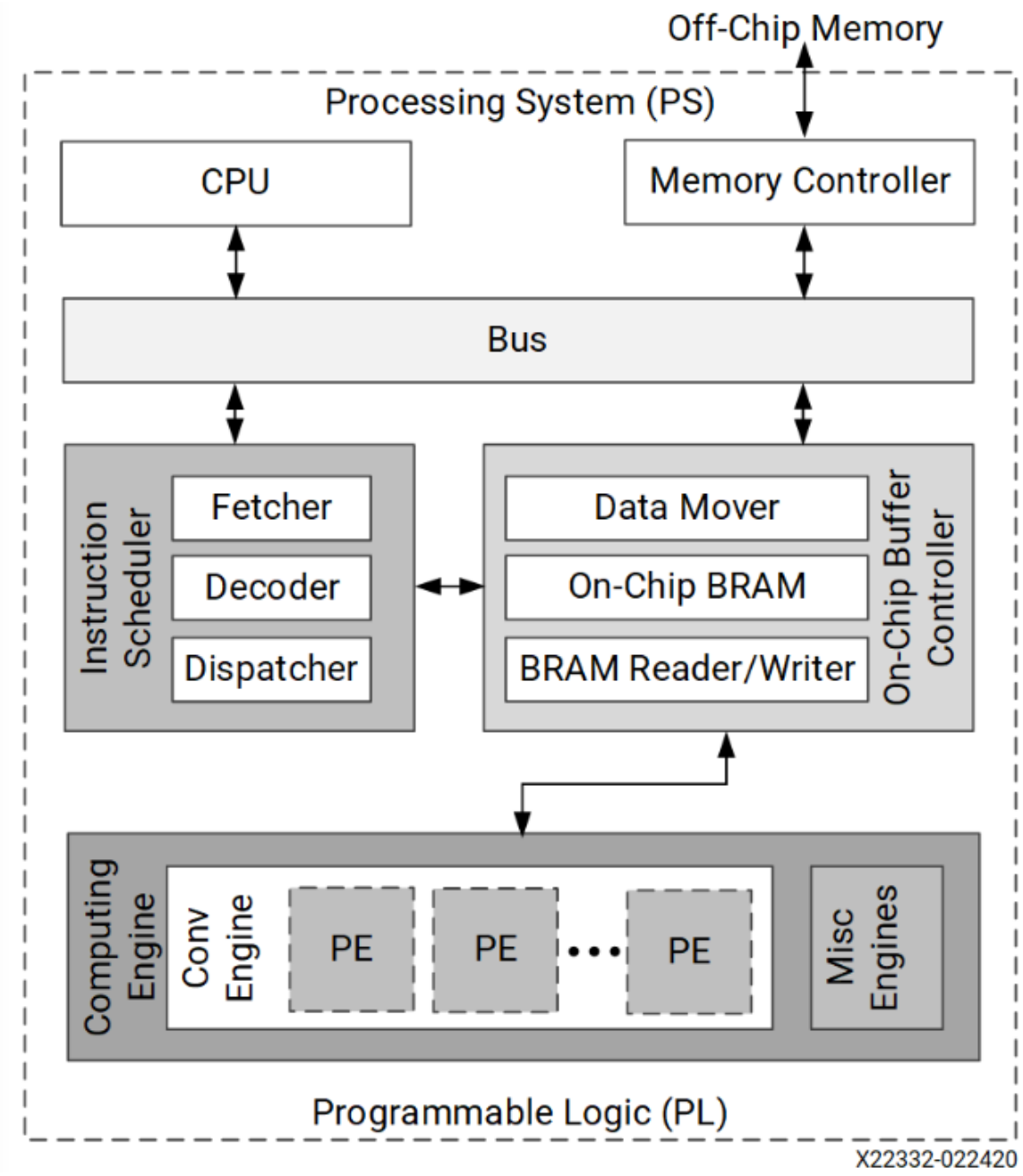


Figure 65: Architecture matérielle de l'unité DPUCZDX8G (AMD-XILINX, 2023)

4.5 Accélération matérielle du modèle YOLOv5n

4.5.1 Introduction et installation de l'environnement Vitis AI

Vitis AI est un environnement de développement logiciel (voir figure 66), développé par AMD/Xilinx pour accélérer la phase d'inférence des modèles d'intelligence artificielle sur les plateformes AMD, telles que des dispositifs de périphérie (edge device) ou les cartes accélératrices AMD Versal™. Ce logiciel comprend des fonctionnalités optimisées, des outils polyvalents, des bibliothèques avancées, ainsi que divers modèles pré-entraînés et des exemples de conception illustratifs. En mettant l'accent sur la haute efficacité et la facilité d'utilisation, Vitis AI permet de gérer l'accélération de l'IA sur les SoC (System on Chip) et les SoC adaptatifs d'AMD. Un autre avantage de Vitis AI est qu'il permet aux utilisateurs ne possédant pas de connaissances approfondies en FPGA de développer facilement des applications d'inférence d'apprentissage profond et de les déployer sur des circuits FPGA (AMD-XILINX, 2023).

Il existe trois modes d'installation de Vitis-AI selon le matériel disponible:

— CPU-only : exécution sans GPU, uniquement sur le processeur.

— CUDA : exécution optimisée pour les GPU NVIDIA via la plateforme CUDA.

— ROCm : support des GPU AMD via la plateforme ROCm.

L'ordinateur utilisé pour réaliser ce projet est doté d'une carte graphique RTX 3070 (8 Go de VRAM). Par conséquent, nous tirons parti de cette configuration en installant la version CUDA (GPU) de Vitis AI 3.0.

Il existe également deux méthodes principales pour installer Vitis AI :

— En exploitant les conteneurs préconçus disponibles sur Docker-Hub.

— En créant un conteneur personnalisé adapté à la machine hôte.

Il est également important de noter qu'il faut un ordinateur doté d'un processeur supportant la technologie AVX. L'AVX (Advanced Vector Extensions) est un ensemble

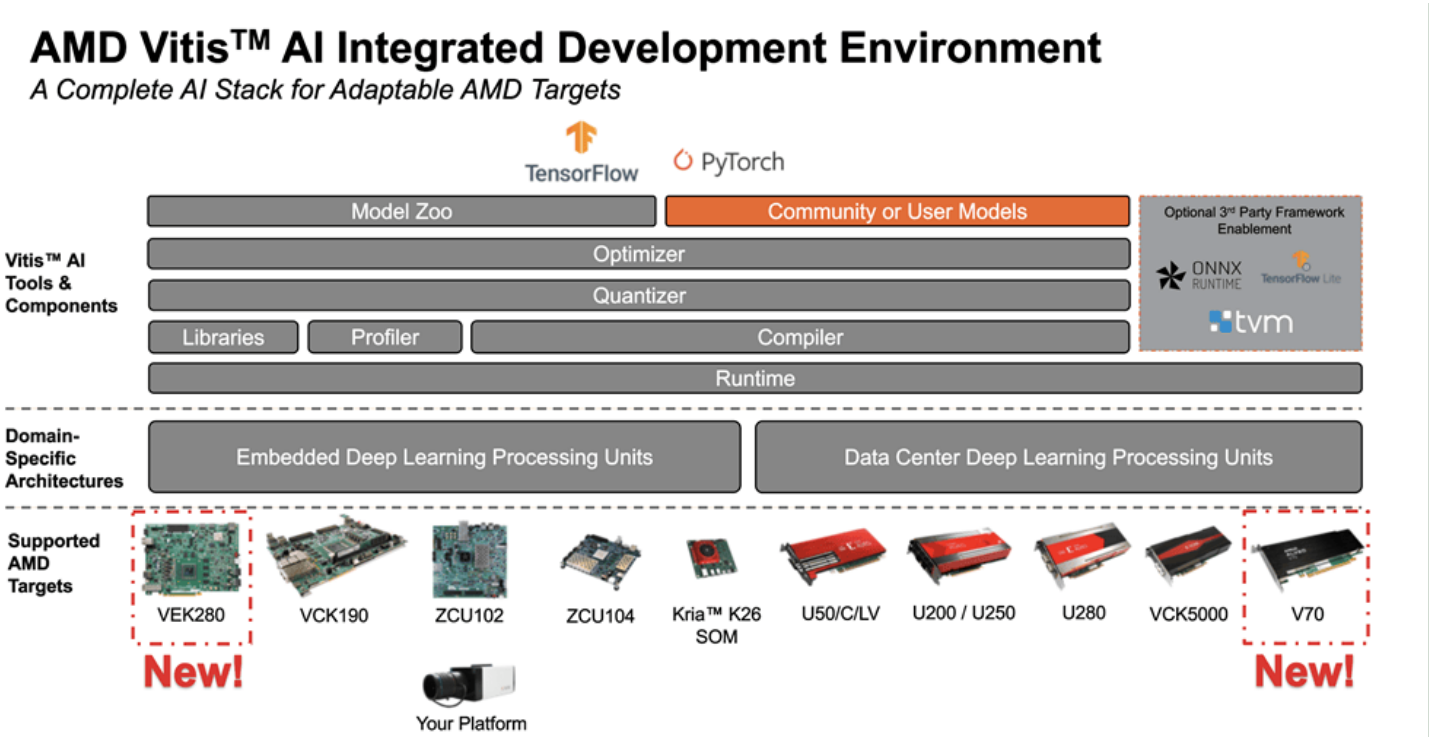


Figure 66: Environnement de développement Vitis-AI (AMD-XILINX, 2023)

d'instructions destiné à effectuer des opérations vectorielles et en virgule flottante sur les architectures modernes de processeurs. Il s'agit d'une extension de l'ensemble d'instructions x86, conçue pour améliorer les performances des calculs intensifs, en particulier ceux impliquant de grands ensembles de données. L'installation de Vitis-AI est détaillée davantage dans la section Annexe (voir Annexe1)

4.5.2 Fonctionnalités de Vitis AI

Cette partie aborde la panoplie de fonctionnalités offertes par le framework Vitis-AI. En effet, le processus d'inférence nécessite une charge de calcul importante. Comme mentionné dans le chapitre 3, il existe plusieurs techniques d'optimisation permettant de réduire cette charge de calcul. Parmi ces techniques, l'élagage (pruning) et la quantification se révélent être les techniques les plus prometteuses. Heureusement, Vitis-AI propose deux modules pour réaliser ces deux tâches.

— Vitis-AI Optimizer.

— Vitis-AI Quantizer.

4.5.2.1 Vitis-AI Optimiser

Le module Optimiser de Vitis-AI permet d'optimiser les modèles d'apprentissage profond (Deep Learning) en réduisant leur complexité. Dans la version 3.0 de Vitis-AI, il inclut un outil appelé le "pruner", qui permet d'élaguer les modèles en supprimant certaines connexions redondantes entre leurs neurones. Par conséquent, la charge de calcul est diminuée de façon significative. Le module prend en charge les frameworks Pytorch et Tensorflow. Vitis-AI Optimiser s'appuie sur le framework d'origine dans lequel le modèle a été entraîné. Le processus prend comme points d'entrée et de sortie du processus d'élagage un graphe en précision FP32 (32 bits Floating Point). Voici le flux de travail (workflow) du module

optimizer illustré dans la figure 67, qui comporte plusieurs étapes (AMD-XILINX, 2023; Xilinx-tutorials, 2023) :

1. **Analyse de sensibilité** : Pour chaque couche et chaque canal de convolution, on mesure à quel point leur suppression impacterait les prédictions du réseau.
2. **Élagage "proposé"** : Les poids des canaux identifiés comme peu sensibles sont mis à zéro, ce qui permet d'évaluer précisément leur influence sans modifier la structure du graphe.
3. **Réentraînement (finetuning)** : Le modèle partiellement élagué est ensuite ré-entraîné sur plusieurs époques afin de récupérer la précision perdue.
4. **Répétition des itérations** : On répète généralement ces deux étapes (élagage + fine-tuning) plusieurs fois. À chaque itération, on sauvegarde l'état actuel, ce qui permet de revenir en arrière en cas de besoin. On retient la version élaguée qui offre le meilleur compromis entre performance et sparsité. Sinon, on relance l'élagage à partir d'un point antérieur avec d'autres hyper-paramètres.
5. **Transformation finale** : Une fois la version optimale identifiée, on passe à l'étape de transformation, qui supprime physiquement du graphe les canaux n'ayant pas d'impact sur les performances du modèle.

À noter que le module Vitis-AI Optimizer nécessitait une licence au moment de la réalisation du projet. Toutefois, à partir de la version 3.5, le module est devenu open source et ne nécessite plus de licence.

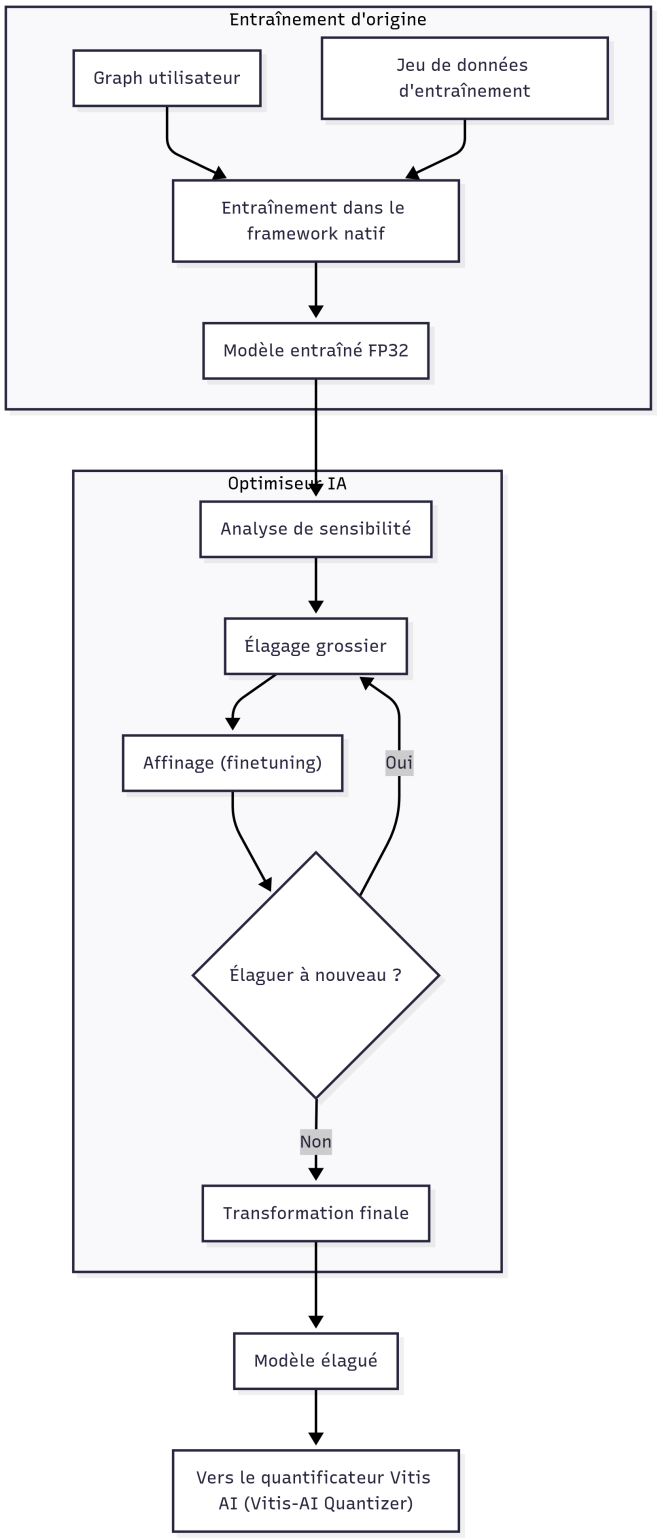


Figure 67: Flux de travail du module Vitis_AI optimizer, inspiré de (Xilinx-tutorials, 2023)

4.5.2.2 Vitis-AI Quantizer

Le second module proposé par le framework Vitis-AI est le quantifieur (Quantizer), compatible à TensorFlow ou PyTorch. Le flux de travail(workflow) du Quantizer est illustré dans la figure 68. Il utilise une base de données de calibration afin d'ajuster le modèle, de limiter la perte de performances et collecter la distribution des activations à chaque couche. La base de données de calibration est généralement dérivée de la base de données utilisée pour entraîner le modèle. Elle ne nécessite uniquement entre 100 et 1000 images non étiquetées. Une fois la calibration terminée, les paramètres du réseau (poids et activations) sont quantifiés en 8 bits (INT8) dans le cadre d'une quantification après l'entraînement PTQ (Post-Training Quantization). Une fois la quantification appliquée au modèle, on réévalue sa précision sur les données de validation. Si les performances sont jugées satisfaisantes, le processus se termine à ce stade. Cependant, comme mentionné dans le chapitre 3, la quantification peut altérer les performances du modèle. C'est pour cette raison que le quantificateur (Quantizer) de Vitis-AI prend en charge l'entraînement conscient de la quantification QAT (Quantization Aware Training). Dans ce cas, les données d'entraînement sont réutilisées pour effectuer plusieurs passes de rétro-propagation et ainsi affiner les poids quantifiés, afin de compenser la perte de précision initiale (Xilinx-tutorials, 2023).

4.5.3 Résultats de la quantification

Avant d'entamer la quantification du modèle YOLOv5n, il est important de noter que la seule méthode acceptée au niveau de la tête de l'architecture *Detect_Class* est *forward()*. Les autres méthodes sont réservées au prétraitement et au post-traitement. La version 3.0 de Vitis-AI ne supporte pas certains opérateurs comme *view* et *permute*. Une façon simple de contourner ce problème est de suivre la solution proposée par Logictronix (Logictronix, 2023), qui consiste à modifier le code de détection (voir Annexe2) en retirant la couche concernée dans la tête de détection et l'implanter dans la section de post-traitement (voir

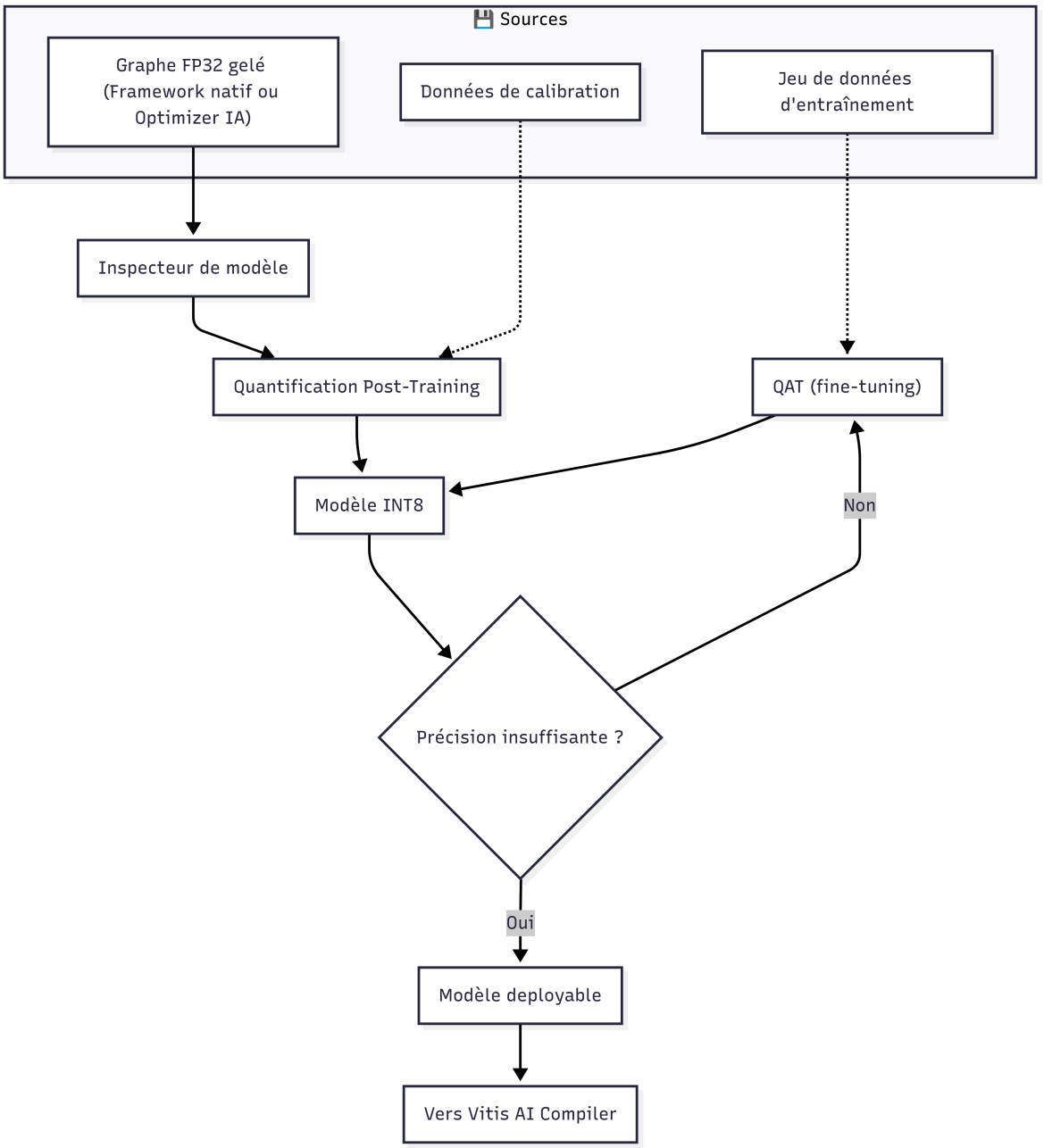


Figure 68: Flux de travail du module Vitis AI Quantizer, inspiré de (Xilinx-tutorials, 2023)

Annexe2). Si cette condition n'est pas respectée, toutes les opérations non supportées seront exécutée sur le CPU de la carte Kria KV260 au lieu du DPU, ce qui engendre une grande latence et une perte significative des performances du modèle.

Par la suite, nous avons suivi l'algorithme de Logictronix qui définit une classe personnalisée *CustomImageDataset*, qui sert à charger des images ainsi que leurs annotations au format YOLO. Dans notre cas, les versions du modèle YOLOv5 chargées grâce à la classe *DetectMultiBackend* sont YOLOv5n, YOLOv5s et YOLOv5l. Cette classe permet d'utiliser différents formats et backends de modèles (Par exemple: PyTorch, ONNX, TensorRT, etc.) avec la même interface, rendant le pipeline d'inférence compatible avec divers environnements.

L'étape suivante de ce script consiste à appliquer la quantification en utilisant la bibliothèque *pytorch_nndct* d'AMD/Xilinx. Cette bibliothèque contient plusieurs interfaces de programmation d'applications (APIs) permettant de convertir la précision des paramètres de réseau de neurones de FP32 à INT8. La quantification se fait en deux étapes principales : *Calibration* et *Test*.

— *Calibration (Calib)*: C'est la première étape de quantification. Le modèle reçoit les entrées (images) non étiquetées, et le quantificateur analyse la distribution des paramètres du réseau. À la fin de cette étape, les fichiers de calibration et de configuration sont générés. Ils servent à paramétrer la quantification tout en minimisant la perte de précision.

— *Test* : C'est la deuxième étape, où la quantification proprement dite est appliquée. La précision des paramètres est changée de FP32 à INT8 en utilisant les fichiers de configuration générés à l'étape précédente. Cette deuxième étape produit un fichier .xmodel en format 8 bits, prêt à être déployé sur le circuit FPGA.

À la fin du processus de quantification, on obtient un fichier .xmodel appelé *DetectMultiBackend.xmodel*. Ce fichier sera par la suite analysé par le module *vai-c*, qui est le compilateur de Vitis-AI, afin de générer un nouveau fichier .xmodel compatible avec le DPU

de la carte Kria KV260 et donc prêt pour un déploiement matériel.

4.5.3.1 Compilation du fichier xmodel

Le module Vitis-AI Compiler d'AMD/Xilinx (AMD-XILINX, 2023) est un outil qui sert à convertir des modèles quantifiés de format implantable sur le DPU. Le processus de compilation se déroule en deux étapes principales.

1. Le compilateur analyse et convertit le modèle quantifié en une représentation graphique interne appelée XIR (Xilinx Intermediate Representation). L'objectif de cette étape est d'éliminer les dépendances liées au Framework d'origine qui, dans notre cas, est PyTorch. La représentation XIR obtenue résume le modèle sous la forme de nœuds (opérations), de tenseurs et de sous-graphes.
2. Le compilateur effectue une série d'optimisations spécifiques au matériel, telles que: la fusion d'opérateurs (fusion de la normalisation par lots avec la convolution), la planification de la mémoire et l'ordonnancement des instructions en fonction de l'architecture DPU.

À partir du graphe de représentation XIR initial, le compilateur génère des sous-graphes optimisés pour le DPU de la carte Kria KV260, dont la configuration est décrite dans le fichier *arch.Json*.

Le résultat final de la compilation est un fichier .xmodel contenant les instructions optimisées du modèle YOLOv5, les poids quantifiés, les formats des tenseurs d'entrée/sortie, ainsi qu'une séquence d'instructions exécutables par le DPU (AMD-XILINX, 2023).

La compilation est réalisée grâce à la commande suivante :

```
vai_c_xir -xmodel quant_model/DetectMultiBackend.xmodel -a /opt
   /vitis_ai/compiler/arch/DPUCZDX8G/KV260/arch.json -o
   YOLOv5n_pt -n YOLOv5n_pt
```

Le modèle obtenu présente 3 couches de sortie:

— DetectionModel_model__Detect_model__Detect_24__Conv2d_m__ModuleList_2__9323_fix_.

— _DetectMultiBackend_DetectionModel_model__Detect_model__Detect_24__Conv2d_m__ModuleList_1__9304_fix_.

— DetectMultiBackend__DetectMultiBackend_DetectionModel_model_Detect_model__Detect_24__Conv2d_m__ModuleList_0__9285_fix_.

Nous pouvons voir les paramètres de ces couches grâce à l'outil de visualisation Netron.

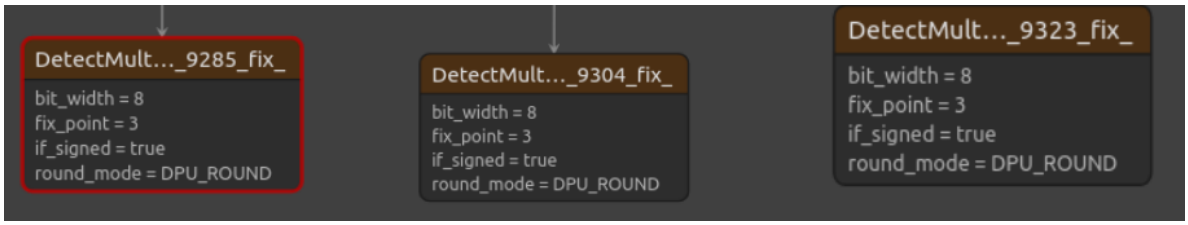


Figure 69: Couches de sortie du modèle .xmodel quantifié et compilé

Une fois le modèle compilé, il reste une dernière étape avant de le déployer sur la carte Kria KV260: La création du fichier prototxt. Ce fichier définit les paramètres spécifiques au modèle, notamment les noms des couches de sortie, les valeurs de normalisation des entrées, qui sont cruciales pour maintenir la précision de l'inférence. La normalisation des entrées dans un réseau de neurones consiste à standardiser les propriétés statistiques des pixels d'une image avant de les transmettre à la couche d'entrée. L'approche la plus couramment adoptée dans les réseaux neuronaux convolutifs (CNN) est la standardisation (z scores) qui s'obtient par la soustraction de la moyenne, suivie de la division par l'écart-type. Plus précisément, il s'agit de calculer la moyenne et l'écart-type par canal (R, G, B) sur l'ensemble des données d'apprentissage. Les valeurs introduites dans le fichier prototxt doivent absolument correspondre aux valeurs obtenues lors de l'entraînement.

La figure 69 illustre les trois couches de sortie du fichier .xmodel obtenu après la phase

de compilation. Le fichier prototxt élaboré pour ce projet est présenté en annexe. Nous avons indiqué les noms des couches extraites lors de l'analyse du modèle quantifié et compilé (9323, 9285 et 9304). Par ailleurs, les valeurs de la moyenne et de l'écart-type ont été calculées à l'issue de l'entraînement du modèle.

4.6 Déploiement du modèle YOLOv5n sur la carte Kria KV260

Pour assurer une connexion entre la carte Kria KV260 et la machine hôte, nous avons utilisé un câble Ethernet (*eth0*) et configuré une connexion SSH. Nous avons attribué l'adresse IP *192.168.0.2* à la carte et *192.168.0.1* à la machine hôte. La connexion SSH a été établie via la commande : `sudo SHH -X root@192.168.0.2`

Une fois la connexion SSH établie, la première étape consiste à transférer le modèle compilé sur la carte Kria KV260. Il est important de bien nommer le répertoire contenant le modèle avec le même nom que ce dernier. Dans notre cas, nos répertoires de modèles étaient *YOLOv5n_KITTI_640*, *YOLOv5n_pt_ coco* et *YOLOv5n_cancer*. Une fois que nos fichiers (.xmodel, prototxt et quant_info.json) sont transférés via la commande *scp*, il faut exécuter l'inférence de ceux-ci. Pour cela, nous avons développé nos propres scripts d'inférence en C++ capables de traiter des images JPEG, des vidéos AVI et des flux en temps réel. Ces scripts d'inférence utilisent le framework *GStreamer* (GStreamer, 2024), une solution open-source permettant de concevoir des pipelines audio/vidéo pour la lecture, le streaming, le transcodage, le mixage sur de multiples plateformes. Notre pipeline d'inférence utilise deux codes principaux: *test_YOLOv5_video.cpp* pour les vidéos et *test_jpeg_YOLOv5* pour les images. Avant l'inférence, chaque image est redimensionnée à la taille d'entrée attendue par le modèle (dans notre cas, 640×640 pixels), ce qui garantit la compatibilité et des performances de détection optimales.

Par la suite, le post-traitement de chaque résultat de détection se fait à l'aide de la

fonction *process_result*, qui effectue les opérations suivantes:

— Superposition des boîtes englobantes et des étiquettes de classe (avec des scores de confiance) directement sur l'image.
— L'utilisation d'une liste de classes d'objets est adaptée à la base de données utilisée.
— La génération dynamique de palettes de couleurs afin de distinguer visuellement les classes présentes sur l'image.

Finalement, l'image est redimensionnée aux dimensions de la vidéo d'origine afin de maintenir la cohérence spatiale de la vidéo d'entrée. Nos scripts enregistrent également les résultats obtenus afin de les transférer et de les visualiser sur la machine hôte grâce à la commande *scp*.

4.6.1 Résultats de l'implantation du modèle YOLOv5n pour les images

Les résultats obtenus sont présentés dans les figures 70 et 71 qui suivent. On peut remarquer qu'il n'existe pratiquement aucune différence significative entre les résultats de prédiction et de détection des deux plateformes (GPU et FPGA). La légère différence entre les scores de confiance est due à la quantification effectuée sur YOLOv5n. Les deux systèmes détectent les mêmes objets principaux, ce qui indique une bonne robustesse du modèle embarqué sur FPGA. Les boîtes englobantes génèrées sont situées aux mêmes coordonnées confirmant l'hypothèse selon laquelle la quantification n'entraîne pas une perte significative des performances du modèle a été vérifiée. Le fait que la même scène soit traitée de manière cohérente sur deux plateformes démontre à la fois la portabilité et la robustesse de l'architecture du modèle YOLOv5n, ainsi que la capacité de la carte Kria KV260 à exécuter efficacement le modèle, même après quantification pour le matériel embarqué.

Du point de vue matériel, on peut dire que la légère différence de scores de confiance rappelle que le choix du matériel, que ce soit un FPGA ou un GPU, influence non seulement

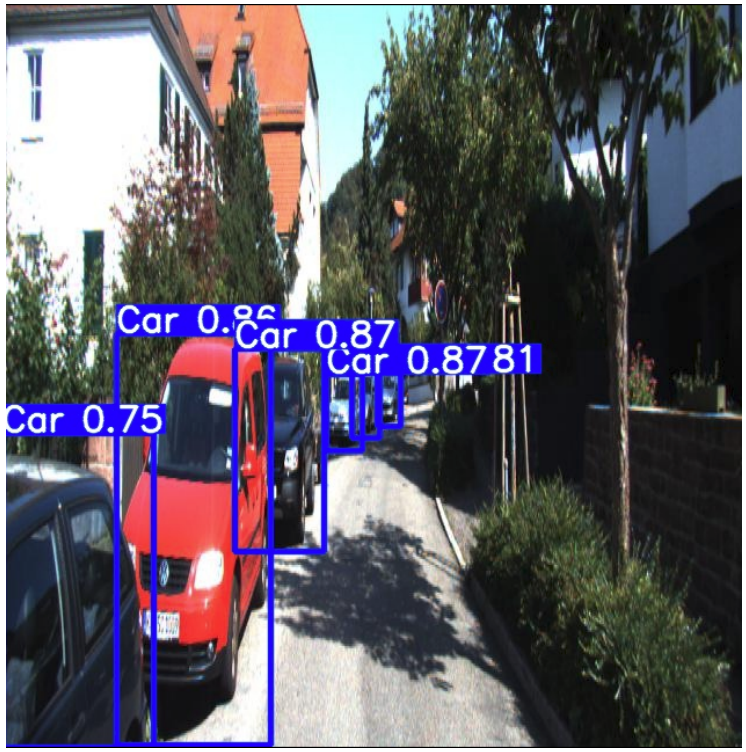
les performances (latence, débit), mais peut également impacter légèrement les résultats, principalement à cause de la quantification.

4.6.2 Résultats de l'implantation matérielle du modèle YOLOv5n pour un streaming vidéo en temps réel

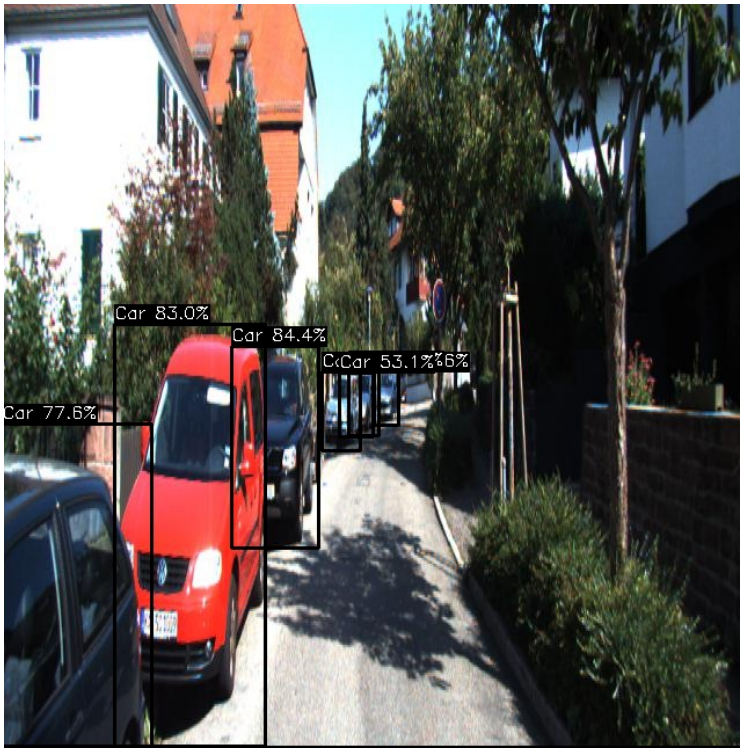
Un dernier test réalisé dans le cadre de ce projet consistait à effectuer une inférence en temps réel du modèle YOLOv5n (voir figure 72). En effet, les capacités d'inférence du modèle ont été démontrées par l'analyse de vidéos préenregistrées et par visualisation en direct. Pour ce faire, nous avons utilisé comme données d'entrée une vidéo de visite de la ville de Rimouski (flux préenregistré) et, pour le test en direct, une caméra Logitech C922X Pro. Les résultats expérimentaux obtenus lors de ces essais sont présentés ci-dessous. Ensuite, une évaluation détaillée des performances et de l'efficacité matérielle du modèle sur la plateforme Kria KV260 a été réalisée.

Afin d'analyser les performances du modèle YOLOv5n exécuté en temps réel, AMD/Xilinx propose un outil intégré au package Vitis-AI, dédié à l'analyse et à l'optimisation des performances des modèles quantifiés et déployés avec Vitis-AI. Le module *Vitis-AI Profiler* permet ainsi de profiler et de visualiser les applications d'IA. En effet, certaines opérations s'exécutent sur le matériel, comme le calcul effectué par le réseau CNN qui s'effectue sur le DPU, tandis que d'autres, comme le prétraitement d'images, s'exécutent sur un CPU en tant que fonctions programmées en C/C++. À la fin de l'analyse, trois fichiers au format .csv et un fichier .xrt sont générés. Ces fichiers contiennent des informations concernant les ressources matérielles utilisées pendant l'exécution, ainsi que les performances du modèle. Le but de cette analyse est de faire le diagnostic et l'optimisation de la chaîne complète d'inférence (AMD-XILINX, 2023).

Dans le tableau 16, on observe que le sous-graphe *DetectMultiBackend*, exécuté sur le DPU de la carte Kria KV260 (*DPUCZDX8G_1:batch-1*), a été exécuté 1508 fois avec



(a) Inférence de YOLOv5n sur KITTI sur la machine hôte



(b) Inférence de YOLOv5n sur KITTI sur la carte Kria KV260

Figure 70: Comparaison entre l'inférence du modèle YOLOv5n sur la machine hôte et sur la carte Kria KV260 sur la base de données KITTI

(a) Inférence de YOLOv5n sur COCO sur la machine hôte

(b) Inférence de YOLOv5n sur COCO sur la carte Kria KV260

Figure 71: Comparaison entre l'inférence du modèle YOLOv5n sur la machine hôte et sur la carte Kria KV260, en utilisant la base de données COCO

(a) Détection de piétons et de voitures à Rimouski avec YOLOv5n sur la carte Kria KV260

(b) Détection de voitures stationnées à Rimouski avec YOLOv5n sur la carte Kria KV260

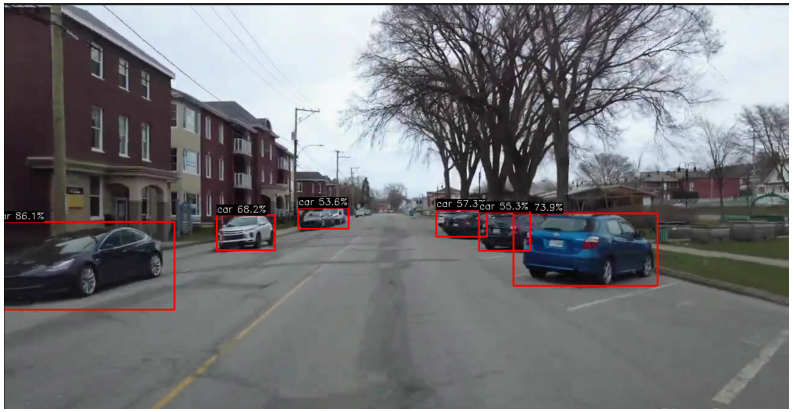


Figure 72: Exemples de détection d'objets en temps réel sur des scènes urbaines à Rimouski, à l'aide de YOLOv5n déployé sur la carte Kria KV260.

Table 16: Mesures de temps et de performance pour YOLOv5n sur Kria KV260

Kernel	Compute Unit	Runs	Min (ms)	Avg (ms)	Max (ms)	Perf. (GOP/s)
subgraph_DetectMultiBackend	DPUCZDX8G_1:batch-1	1508	6.94	7.03	16.93	53.21

Table 17: Charges mémoire et bande passante relevées lors de l'exécution

Kernel	Workload (GOP)	Mem IO (MB)	Mem BW (MB/s)
subgraph_DetectMultiBackend	4.49	29.56	4305.45

un temps moyen de 7.03 ms pour une variation comprise entre un minimum de 6.94 ms et un maximum de 16.93 ms. Cette faible variation du temps d'exécution moyen démontre la stabilité du modèle YOLOv5n, une fois déployé. En outre, le système a atteint 53.21 GOPs/s (Giga opérations par seconde), ce qui indique une capacité de traitement élevée du DPU intégré à la carte Kria KV260.

Au niveau de la charge de calcul et de l'utilisation de la mémoire associée à l'exécution du modèle YOLOv5n (voir tableau 17), on remarque cependant que le flux de travail total (workload) du kernel est évalué à 4.49 GOP pour les opérations de traitement d'images, nécessitant un transfert mémoire d'environ 29.56 MB, avec une bande passante atteignant 4305.45 MB/s. Cette utilisation élevée de la bande passante mémoire peut s'expliquer par un échange assez intensif entre la mémoire vive DDR et l'unité de traitement DPU. Ceci peut causer un goulot d'étranglement (*Bottleneck*) pour des modèles plus profonds et plus complexes. Cependant, les résultats restent tout à fait satisfaisants pour le modèle *YOLOv5n*.

Les résultats concernant le taux d'utilisation de la mémoire en lecture et en écriture des différents ports DDRC sur la carte Kria KV260 sont également présentés. Sur la figure 73, le port *DDRC_PORT_S5* (courbe noire) est le plus utilisé lors de l'exécution du modèle. Ce comportement périodique peut s'expliquer par l'utilisation régulière et intensive, avec un pic de débit proche de 900 MB/s en lecture et en écriture, et qui est lié au transfert de données pour l'inférence des images dans le DPU. À l'inverse, les ports DDRC_PORT_S1 à S3 (bleu,

orange et vert) présentent des débits bien inférieurs. Ces ports pourraient être utilisés pour des tâches secondaires, comme la gestion de l'environnement Linux ou toute autre tâche non liée directement au DPU.

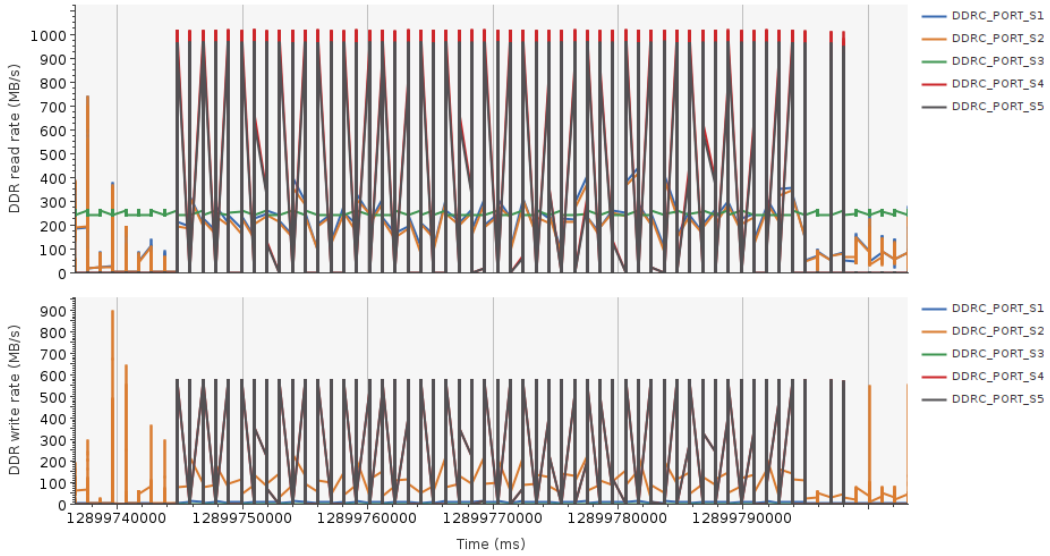


Figure 73: Graphique du taux de lectures et d'écriture obtenus sur les différents ports DDRC lors de l'éxecution du modèle YOLOv5n sur la carte Kria KV260

4.6.3 Comparaison entre les différentes plateformes lors de l'exécution du modèle YOLOv5n

Dans cette sous-section, nous présentons les résultats obtenus à partir de l'implantation matérielle du modèle YOLOv5n sur la carte Kria KV260 (voir tableau 18). Nous analysons en particulier la consommation énergétique et les performances matérielles durant la phase d'inférence. On remarque une augmentation notable de la consommation énergétique pendant la phase d'inférence. La puissance totale du SOM (System On Module) augmente d'environ 1087.50 mW, soit à peu près 20% par rapport à l'état IDLE (repos). Cette augmentation de la puissance s'accompagne bien évidemment d'une augmentation du courant consommé, tandis que la tension reste relativement stable, démontrant que la régulation de l'alimentation est effectivement efficace.

Au niveau des ressources du CPU, l'augmentation est relativement modérée, de l'ordre de $\tilde{1}5\%$ sur les deux cœurs, ce qui indique que la charge de calcul est plutôt prise en charge par le DPU (Deep Processing Unit) de la carte Kria KV260. Ces résultats illustrent clairement l'efficacité du déploiement matériel du modèle YOLOv5n sur la carte FPGA Kria KV260.

Table 18: Résumé des métriques de performance lors de l'inférence du modèle YOLOv5n sur la plateforme Kria KV260.

Mesure	**Phase IDLE**	**Phase DPU seulement**	**Phase Inférence**
Tension (V)	11.9	11.9	11.9
Courant (A)	0.42	0.51	0.73
Puissance (W)	5.0	6.1	7.7
Utilisation CPU (%)	11	17	34
Utilisation RAM (MB)	400	420	470
Fréquence DPU (MHz)	400	400	400
Fréquence CPU (MHz)	1200	1200	1200

Nous avons également regroupé les résultats obtenus dans un tableau afin de comparer les performances du modèle sur les trois plateformes différentes, à savoir le CPU, le GPU et le FPGA.

Table 19: Évaluation des plateformes matérielles pour l'inférence du modèle YOLOv5n sur la base de données COCO2017

Métrique d'évaluation	**CPU**	**GPU**	**FPGA (Kria KV260)**
Puissance (W)	–	42.61	2.7
Images par seconde (FPS)	60	158	142
Temps d'exécution (ms)	15.9	6.33	7.03
Quantification	32 FP	32 FP	8 bits
Taille de l'image (pixels)	640×640	640×640	640×640

À partir du tableau 19, on peut observer que la carte FPGA (Kria KV260) offre un com-

promis intéressant. Elle permet une exécution extrêmement efficace sur le plan énergétique, avec une consommation de seulement 2.7 W, tout en maintenant un débit de 142 images par seconde (FPS), proche de celui de la carte graphique (158 FPS). En revanche, le GPU, malgré son débit élevé (158 FPS), consomme environ 16x plus d'énergie. La quantification en 8 bits optimise la bande passante de la mémoire et réduit les charges de calcul, contribuant directement à la performance du modèle deployé sur la carte FPGA Kria KV260. Il est important de noter que le GPU est conçu pour le calcul parallèle massif, grâce à ses milliers de cœurs spécialisés (CUDA ou équivalents), ce qui lui permet d'atteindre des performances maximales, mais au prix d'une consommation importante d'énergie. De son côté, le CPU offre des performances assez limitées en traitement parallèle, avec un temps d'exécution relativement élevé (15.9 ms) et un débit modeste (60 FPS). Sa consommation énergétique n'est pas précisée, mais elle est généralement supérieure à celle d'un FPGA. Les résultats expérimentaux montrent que la latence d'inférence du modèle YOLOv5 implémenté sur la plateforme Kria KV260 est de 7,03 ms pour un flux vidéo à 60 images par seconde, soit une valeur largement inférieure à la période de 16,67 ms imposée par la caméra.

Ces résultats permettent de valider l'hypothèse H4, selon laquelle l'implémentation proposée satisfait une contrainte de temps réel souple pour des applications de vision embarquée. Toutefois, en l'absence de garantie formelle sur le temps d'exécution dans le pire cas, cette implémentation ne peut être qualifiée de système temps réel strict au sens du génie électrique.

Sur le plan matériel, la carte Kria KV260 se distingue donc par sa capacité à offrir une efficacité énergétique prometteuse, tout en maintenant un niveau de performance élevé. Le circuit FPGA se positionne ainsi comme une solution idéale pour des applications embarquées nécessitant un compromis entre performance, consommation énergétique et rapidité. À l'inverse, le GPU reste pertinent pour des environnements de calcul haute performance où la consommation énérgétique n'est pas une contrainte.

4.7 Conclusion

Dans ce chapitre, nous avons présenté l'ensemble des démarches adoptées pour la réalisation de ce projet. Dans la première partie, nous avons fait des choix concernant le modèle à entraîner, ainsi que les bases de données à utiliser pour l'entraînement. Une fois le modèle entraîné, nous avons procédé à son implantation matérielle en utilisant Vitis-AI pour la quantification et la bibliothèque GStreamer pour réaliser l'inférence. Les résultats obtenus sont prometteurs et permettent de confirmer l'hypothèse du compromis lié à l'inférence en périphérie (edge computing). En effet, le modèle consomme $\tilde{1}$6x moins d'énergie tout en maintenant une vitesse (débit d'inférence) élevée, malgré la perte non significative de performances après la quantification.

CONCLUSION GÉNÉRALE

L'objectif de ce travail de recherche était l'accélération matérielle des modèles de détection d'objets. Une analyse de l'état de l'art a été menée afin d'étudier l'architecture des modèles de détection d'objets et de comprendre le rôle de chacune de leurs composantes (dorsale, cou et tête). Par la suite, nous avons exploré les différentes méthodes d'optimisation des réseaux de neurones, allant de l'élagage, qu'il soit statique ou dynamique, à la quantification.

De plus, plusieurs plateformes d'accélération matérielle (CPU, GPU et FPGA) ont été étudiées. En effet, le CPU représente la plateforme la moins adaptée à l'accélération matérielle en raison de sa faible capacité à gérer le parallélisme intensif des pipelines requis par les modèles de détection d'objets. Le GPU est conçu pour le calcul parallèle, grâce à des milliers de cœurs spécialisés (CUDA ou équivalents), ce qui lui permet d'atteindre des performances maximales. Cependant, il présente des limitations liées à sa consommation d'énergie importante et à son coût élevé. Nous avons donc choisi la carte Kria KV260, qui offre un compromis unique. Les circuits FPGA se positionnent à mi-chemin entre flexibilité et performance, permettant un parallélisme matériel configurable pour traiter des tâches d'apprentissage profond (deep learning) avec de faibles latences et une consommation énergétique très basse.

Durant ce projet, nous avons utilisé le logiciel Vitis-AI d'AMD/Xilinx afin de quantifier la version nano de Yolov5, qui comporte le moins de paramètres (1.9 million) comparée aux autres versions, ce qui en fait la meilleure cible pour un déploiement sur un système embarqué. La quantification réduit la précision de notre modèle de virgule flottante de 32 bits (FP32) à une précision d'entier de 8 bits (INT8). Les performances du modèle n'ont pas connu de dégradation significative, permettant ainsi une compilation réussie sur la carte Kria KV260. Ayant attribué la charge de calcul la plus importante au DPU de la carte Kria KV260, nous avons exécuté l'inférence du modèle Yolov5n grâce à notre script C++ qui utilise la bibliothèque GStreamer.

Les tests de notre modèle Yolov5n, réalisés sur la carte Kria KV260 ont montré des résultats prometteurs. En effet, nous avons réussi à répliquer les mêmes performances du modèle exécuté sur notre machine hôte avec une légère différence de précision, qui s'explique par le changement de précision de FP32 à INT8 lors de la quantification. En ce qui concerne les tests en temps réel, le circuit FPGA de la carte Kria KV260 présente une faible consommation énergétique, environ 15 fois moins que celle du GPU et une latence de 7.03 ms, correspondant à 142 images par seconde (FPS). Ces résultats démontrent que les FPGA constituent une solution fiable pour exécuter une inférence en périphérie (edge computing). Ils permettent de déployer des modèles de détection d'objets complexes pour diverses applications embarquées sans nécessiter internet ou une machine hôte puissante.

Ces travaux ouvrent la voie à plusieurs perspectives, parmi lesquelles on peut citer :

— La création d'un projet Vivado permettra de générer un fichier bitstream, offrant un contrôle précis des ressources logiques et des interconnexions mémoire.

— La conception d'une application Vitis pour mieux personnaliser le DPU au niveau RTL, en modulant les pipelines.

— La création d'une image Linux personnalisée via PetaLinux au lieu d'utiliser l'image pré-conçue par AMD-Xilinx. Cela permet d'adapter l'architecture selon les besoins spécifique, rendant le système plus flexible.

ANNEXE I

INSTALLATION DE VITIS-AI

L'installation de Vitis AI nécessite environ 100 Go d'espace disque dur et un système d'exploitation Linux (dans notre cas, Ubuntu 20.04). Par la suite, il faut installer le toolkit CUDA de NVIDIA, qui permet d'activer le mode GPU dans le conteneur Docker. Le toolkit peut être installé via les commandes suivantes :

— `apt purge nvidia* libnvidia*`

— `apt install nvidia-driver-<xxx>`

— `apt install nvidia-container-toolkit`

xxx étant la version du pilote à installer (exemple nvidia-driver-510). Si vous ne connaissez pas la version du pilote adapté à votre système, il suffit simplement d'utiliser la commande :

— `ubuntu-drivers autoinstall`

Afin de vérifier que l'installation du pilote a bien été complétée, il suffit d'utiliser la commande suivante : `nvidia-smi`. Cette commande affiche les informations relatives à la carte graphique (modèle, version du pilote, mémoire utilisée, etc). Le résultat obtenu est illustré à la figure 74.

Une fois que le toolkit NVIDIA est installé, il faut installer Docker. Docker est une plateforme logicielle qui permet de concevoir, tester et déployer des applications rapidement. Il intègre les logiciels dans des unités normalisées appelées conteneurs, qui rassemblent tous les éléments nécessaires à leur fonctionnement, tels que les bibliothèques, les outils système, le code et l'environnement d'exécution. Grâce à Docker, il devient plus facile de déployer

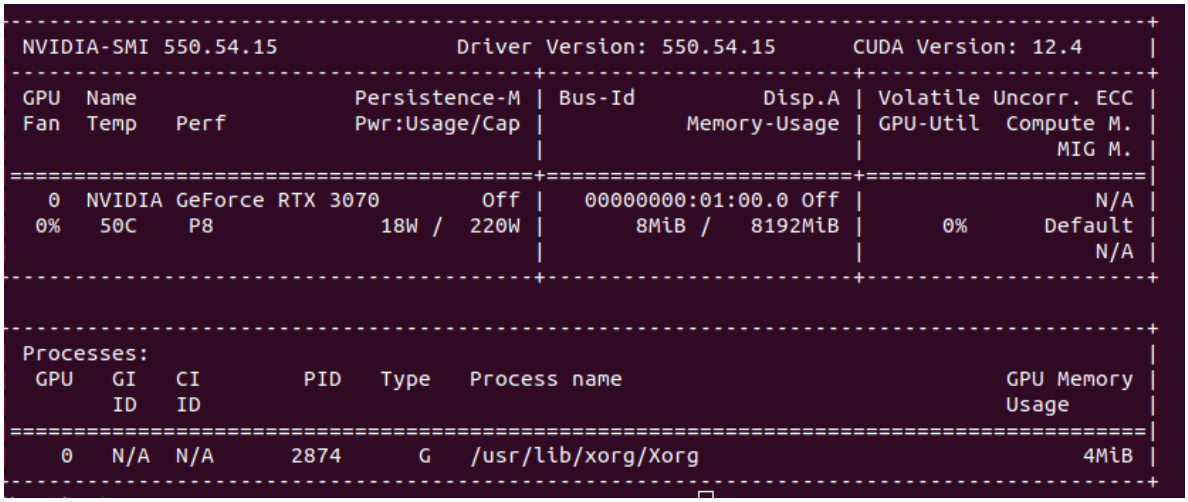


FIGURE 74: Résultat de la commande nvidia-smi

et de dimensionner des applications dans n'importe quel environnement, tout en garantissant une bonne exécution du code (Docker, 2023). Les éléments de base de Docker sont les images et les conteneurs :

— Une image Docker un paquet autonome contenant tous les éléments nécessaires à l'exécution d'un logiciel : le code, un environnement d'exécution tel que JVM (Java Virtual Machine), des pilotes, des outils, des scripts, des bibliothèques, etc.

— Un conteneur Docker est une instance en cours d'exécution d'une image Docker, isolé du système hôte mais utilisant ses ressources lorsque nécessaire.

Dans une image Docker, il n'y a pas de système d'exploitation séparé, comme illustré dans la figure 75 qui suit. Contrairement à la machine virtuelle, le conteneur Docker n'a pas besoin de système d'exploitation pour exécuter ses applications, ce qui le rend beaucoup plus léger. Selon (Docker, 2023), un conteneur nécessite quelques Mo (mégaoctets) de stockage, tandis qu'une machine virtuelle peut en nécessiter plusieurs Go (gigaoctets). Ces avantages permettent particulièrement :

— Un lancement beaucoup plus rapide des applications.

— La migration facilitée d'une machine à une autre.

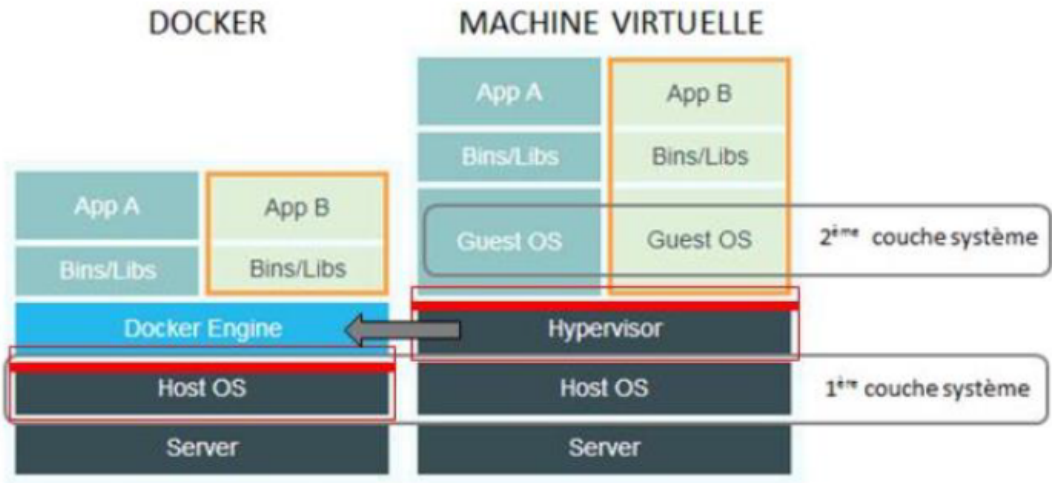


FIGURE 75: Différence au niveau de l'architecture entre une machine virtuelle et un conteneur Docker (Docker, 2023)

Une fois Docker installé, l'importation de Vitis-AI se fait via les deux commandes suivantes :

1. `git clone https://github.com/Xilinx/Vitis-AI` permet d'importer le répertoire Github de Xilinx Vitis AI.
2. `cd Vitis-AI`

Depuis la version 3.0, il est possible d'utiliser les conteneurs Docker pour les trois types d'installation :

— CPU-only

— CUDA-capable GPUs

— ROCm-capable GPUs

Pour le mode CUDA (GPU), il n'existe malheureusement pas de conteneurs préconçus. Il faudra donc concevoir un son propre conteneur. Pour ce faire, il suffit d'exécuter la commande suivante, avec les paramètres illustrés dans le tableau 15 :

— `cd <Vitis-AI install path>/Vitis-AI/docker`

— `./docker_build.sh t <DOCKER_TYPE> f <FRAMEWORK>`

TABLEAU 20: Options de construction de l'image Docker pour Vitis AI 3.0

DOCKER_TYPE (-t)	TARGET_FRAMEWORK (-f)	Desired Environment
cpu	pytorch	PyTorch cpu-only
	tf2	TensorFlow 2 cpu-only
	tf1	TensorFlow 1.15 cpu-only
gpu	pytorch	PyTorch CUDA-gpu
	opt_pytorch	PyTorch with AI Optimizer CUDA-gpu
	tf2	TensorFlow 2 CUDA-gpu
	opt_tf2	TensorFlow 2 with AI Optimizer CUDA-gpu
	tf1	TensorFlow 1.15 CUDA-gpu
	opt_tf1	TensorFlow 1.15 with AI Optimizer CUDA-gpu
rocm	pytorch	PyTorch ROCm-gpu
	opt_pytorch	PyTorch with AI Optimizer ROCm-gpu
	tf2	TensorFlow 2 ROCm-gpu
	opt_tf2	TensorFlow 2 with AI Optimizer ROCm-gpu

Dans notre cas, nous avons opté pour le type `gpu` et le framework `pytorch`.

— `./docker_build.sh -t gpu -f opt_pytorch`

La commande suivante permet de vérifier si Docker a bien été activé avec l'option CUDA.

— `docker run --gpus all nvidia/cuda:11.3.1-cudnn8-runtime-ubuntu20.04 nvidia-smi`

Une fois que le conteneur Docker est créé et que l'image Vitis-AI construite, il est possible de lancer Vitis-AI 3.0 via la commande suivante :

— `./docker_run.sh xilinx/vitis-ai-pytorch-gpu:latest`

Le résultat obtenu est illustré dans la figure 76.

FIGURE 76: Interface de Vitis AI 3.0 dans l'invite de commande Linux

ANNEXE II

MODIFICATION DU CODE DE DÉTECTION DE YOLOV5

Le code suivant présente l'implémentation de la fonction `forward` utilisée dans le modèle YOLOv5 pour le traitement des sorties du réseau de détection. Cette fonction assure la transformation des tenseurs issus des couches convolutives en prédictions exploitables, aussi bien en phase d'apprentissage qu'en phase d'inférence.

Listing 1: Fonction `forward` du modèle YOLOv5 avant les modification

```
def forward(self, x):
    """Processes input through YOLOv5 layers, altering shape
       for detection:
    x(bs, 3, ny, nx, 85)."""
    z = []  # inference output
    for i in range(self.nl):
        x[i] = self.m[i](x[i])  # convolution
        bs, _, ny, nx = x[i].shape
        x[i] = x[i].view(bs, self.na, self.no, ny, nx) \
                    .permute(0, 1, 3, 4, 2).contiguous()

        if not self.training:  # inference
            if self.dynamic or self.grid[i].shape[2:4] != x[i].
               shape[2:4]:
                self.grid[i], self.anchor_grid[i] = self.
                   _make_grid(nx, ny, i)
```

```
            if isinstance(self, Segment):  # boxes + masks
                xy, wh, conf, mask = x[i].split(
                    (2, 2, self.nc + 1, self.no - self.nc - 5),
                        4)
                xy = (xy.sigmoid() * 2 + self.grid[i]) * self.
                   stride[i]
                wh = (wh.sigmoid() * 2) ** 2 * self.anchor_grid
                   [i]
                y = torch.cat((xy, wh, conf.sigmoid(), mask),
                   4)
            else:  # boxes only
                xy, wh, conf = x[i].sigmoid().split(
                    (2, 2, self.nc + 1), 4)
                xy = (xy * 2 + self.grid[i]) * self.stride[i]
                wh = (wh * 2) ** 2 * self.anchor_grid[i]
                y = torch.cat((xy, wh, conf), 4)

            z.append(y.view(bs, self.na * nx * ny, self.no))

    return x if self.training else \
        (torch.cat(z, 1),) if self.export else (torch.cat(z, 1)
           , x)
```

Listing 2: Fonction `forward` du modèle YOLOv5 après les modification

```
def forward(self, x):
    z = []
    for i in range(self.nl):
            x[i] = self.m[i](x[i])
    return x
```

Listing 3: Post-traitement re-implante dans le script ´ Detect.py

```
def postprocessing(x):
    grid = [torch.empty(0) for _ in range(3)]
    z = []
    anchor_grid = [torch.empty(0) for _ in range(3)]
    stride = torch.tensor([ 8., 16., 32.], device='cuda:0')
    # anchors = torch.tensor([[10.,13., 16.,30.,
       33.,23.],[30.,61., 62.,45., 59.,119.],[116.,90.,
       156.,198., 373.,326.]] , device='cuda:0')
    anchors = torch.tensor([[1.25000,  1.62500, 2.00000,
       3.75000,4.12500,  2.87500],
        [1.87500,  3.81250, 3.87500,  2.81250, 3.68750,
           7.43750],
        [ 3.62500,  2.81250, 4.87500,  6.18750, 11.65625,
           10.18750]], device='cuda:0')
    anchors = torch.tensor(anchors).float().view(3,-1,2)
    # anchors[0] = anchors[0] / stride[0]
    # anchors[1] = anchors[1] / stride[1]
    # anchors[2] = anchors[2] / stride[2]



    for i in range(3):
        bs, _, ny, nx = x[i].shape  # x(bs,255,20,20) to x(bs
```

```
            ,3,20,20,85)
        x[i] = x[i].view(bs, 3, 18, ny, nx).permute(0, 1, 3, 4,
            2).contiguous()

        if grid[i].shape[2:4] != x[i].shape[2:4]:
            grid[i], anchor_grid[i] = make_grid(nx,ny,i,anchors
                ,stride)



        xy, wh, conf = x[i].sigmoid().split((2, 2, 13 + 1), 4)
        xy = (xy * 2 + grid[i]) * stride[i]  # xy
        wh = (wh * 2) ** 2 * anchor_grid[i]  # wh
        y = torch.cat((xy, wh, conf), 4)
        z.append(y.view(bs, 3 * nx * ny, 18))

    return (torch.cat(z, 1), x)
```

RÉFÉRENCES

Alzubaidi, L., Zhang, J., Humaidi, A., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M., Al-Amidie, M., Farhan, L., 2021. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. Journal of Big Data 8. doi:10.1186/s40537-021-00444-8.

Ambuj, Machavaram, R., 2024. Optimizing energy expenditure in agricultural autonomous ground vehicles through a gpu-accelerated particle swarm optimization-artificial neural network framework. Cleaner Energy Systems 9, 100130. doi:https://doi.org/10.1016/j.cles.2024.100130.

AMD, 2021. Kria kv260 vision ai starter kit. URL: https://www.xilinx.com/products/som/kria/kv260-vision-starter-kit.html.

AMD-XILINX, 2023. Deep-learning processor unit, vitis ai user-guide (ug1414), reader amd technical information portal. URL: https://docs.amd.com/r/3.0-English/ug1414-vitis-ai/Deep-Learning-Processor-Unit.

Anwar, S., Hwang, K., Sung, W., 2017. Structured pruning of deep convolutional neural networks. ACM Journal on Emerging Technologies in Computing Systems 13. doi:10.1145/3005348.

Bergmann, D., 2024. What is knowledge distillation_ _ ibm. URL: https://www.ibm.com/think/topics/knowledge-distillation.

Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M., 2020. YOLOv4: Optimal speed and accuracy of object detection. URL: http://arxiv.org/abs/2004.10934.

Bouraya, S., Belangour, A., 2021. Deep learning based neck models for object detection: A review and a benchmarking study. International Journal of Advanced Computer Science and Applications 12, 161–167. doi:10.14569/IJACSA.2021.0121119.

Brown, S., Rose, J., 1996. Fpga and cpld architectures: a tutorial. IEEE Design Test of Computers 13, 42–57. doi:10.1109/54.500200.

Brown, S., Vranesic, Z., 2014. Fundamentals of Digital Logic with VHDL Design. 3rd ed., McGraw-Hill Education, New York, NY.

Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D., 2020. Language models are few-shot learners. URL: https://arxiv.org/abs/2005.14165, arXiv:2005.14165.

Bucilă, C., Caruana, R., Niculescu-Mizil, A., 2006. Model compression, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 535–541. doi:10.1145/1150402.1150464.

Cai, J., Makita, Y., Zheng, Y., Takahashi, S., Hao, W., Nakatoh, Y., 2022. Single shot multibox detector for honeybee detection. Computers and Electrical Engineering 104. doi:10.1016/j.compeleceng.2022.108465.

Cao, Y., Jia, J., Xiao, W., Jia, J., Zhou, W., 2020. Proceedings of 2020 ieee international conference on artificial intelligence and information systems : Icaiis : March 20-22, 2020, dalian, china .

Capra, M., Bussolino, B., Marchisio, A., Masera, G., Martina, M., Shafique, M., 2020. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. IEEE Access 8, 225134–225180. doi:10.1109/ACCESS.2020.3039858.

Carini, D., 2023. Comparing hls4ml and Vitis AI for CNN Synthesis and Evaluation on FPGA: A Comprehensive Study. Ph.D. thesis. URL: https://hdl.handle.net/10589/208840.

Crockett, L.H., Elliot, R.A., Enderwitz, M.A., Stewart, R.W., 2014. The Zynq Book Embedded Processing with the ARM® Cortex®-A9 on the Xilinx® Zynq®-7000 All Programmable SoC. 1st editon ed., Strathclyde Academic Media. URL: www.zynqbook.com.

Das, R.S., Gupta, V., 2024. A systematic literature review on graphics processing unit accelerated realm of high-performance computing. International Journal of Computing and Engineering 5, 10–21. doi:10.47941/ijce.1813.

Davis, A., Arel, I., 2013. Low-rank approximations for conditional feedforward computation in deep neural networks. URL: http://arxiv.org/abs/1312.4461.

Deng, W., Yin, W., Zhang, Y., 2013. Group sparse optimization by alternating direction method, in: Wavelets and Sparsity XV, SPIE. p. 88580R. doi:10.1117/12.2024410.

DeVries, T., Taylor, G.W., 2017. Improved regularization of convolutional neural networks with cutout URL: https://arxiv.org/abs/1708.04552, arXiv:1708.04552.

Docker, 2023. Docker documentation. URL: https://docs.docker.com/guides/.

Dong, X., Yang, Y., 2019. Network pruning via transformable architecture search URL: https://arxiv.org/abs/1905.09717, arXiv:1905.09717.

Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q., Tian, Q., 2019. Centernet: Keypoint triplets for object detection, in: Proceedings of the IEEE International Conference on Computer Vision, Institute of Electrical and Electronics Engineers Inc.. pp. 6568–6577. doi:10.1109/ICCV.2019.00667.

Fang, W., Wang, L., Ren, P., 2020. Tinier-yolo: A real-time object detection method for constrained environments. IEEE Access 8, 1935–1944. doi:10.1109/ACCESS.2019.2961959.

Figurnov, M., Collins, M.D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., Salakhutdinov, R., 2017. Spatially adaptive computation time for residual networks URL: http://arxiv.org/abs/1612.02297.

Frantar, E., Ashkboos, S., Hoefler, T., Alistarh, D., 2023. Gptq: Accurate post-training quantization for generative pre-trained transformers URL: https://arxiv.org/abs/2210.17323, arXiv:2210.17323.

Gao, X., Zhao, Y., Łukasz Dudziak, Mullins, R., zhong Xu, C., 2019. Dynamic channel pruning: Feature boosting and suppression URL: https://arxiv.org/abs/1810.05331, arXiv:1810.05331.

Girshick, R., 2015. Fast r-cnn, in: 2015 IEEE International Conference on Computer Vision (ICCV), IEEE. pp. 1440–1448. URL: http://ieeexplore.ieee.org/document/7410526/, doi:10.1109/ICCV.2015.169.

Google, 2025. Classification_ justesse, rappel, précision et métriques associées _ machine learning google for developers URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall.

Gschwend, D., 2020. Zynqnet: An fpga-accelerated embedded convolutional neural network URL: https://arxiv.org/abs/2005.06892, arXiv:2005.06892.

GStreamer, 2024. Gstreamer: Open source multimedia framework. URL: https://gstreamer.freedesktop.org/.

Guo, K., Zeng, S., Yu, J., Wang, Y., Yang, H., 2017. A survey of fpga-based neural network accelerator 9, 1–26. URL: http://arxiv.org/abs/1712.08934.

Guo, Y., Yao, A., Chen, Y., 2016. Dynamic network surgery for efficient DNNs URL: https://arxiv.org/abs/1608.04493, arXiv:1608.04493.

Haase, D., Amthor, M., 2020. Rethinking depthwise separable convolutions: How intra-kernel correlations lead to improved mobilenets, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 14588–14597. doi:10.1109/CVPR42600.2020.01461.

Han, S., Mao, H., Dally, W.J., 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding URL: http://arxiv.org/abs/1510.00149.

Haykin, S.S., 2009. Neural networks and learning machines. Third ed., Pearson Education, Upper Saddle River, NJ.

He, K., Zhang, X., Ren, S., Sun, J., 2014. Spatial pyramid pooling in deep convolutional networks for visual recognition, in: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (Eds.), Computer Vision – ECCV 2014, Springer International Publishing, Cham. pp. 346–361.

He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y., 2018. Soft filter pruning for accelerating deep convolutional neural networks. URL: https://api.semanticscholar.org/CorpusID:51608028.

Hou, J., Peng, J., Liu, K., Liang, Z., Yao, J., Li, J., Tang, T., 2023. A multi-channel optical sensing system based on back propagation neural network for mixed ion solution. Optik 291, 171391. doi:https://doi.org/10.1016/j.ijleo.2023.171391.

Hu, J., Shen, L., Albanie, S., Sun, G., Wu, E., 2019. Squeeze-and-excitation networks. URL: https://arxiv.org/abs/1709.01507, arXiv:1709.01507.

Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., Weinberger, K.Q., 2017. Multi-scale dense networks for resource efficient image classification. URL: http://arxiv.org/abs/1703.09844.

Huang, Z., Wang, N., 2018. Data-driven sparse structure selection for deep neural networks URL: http://arxiv.org/abs/1707.01213.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D., 2017. Quantization and training of neural networks for efficient integer-arithmetic-only inference URL: http://arxiv.org/abs/1712.05877.

Jaiswal, M., Sharma, A., Saini, S., 2025. Hardware acceleration of tiny yolo deep neural networks for sign language recognition: A comprehensive performance analysis. Integration 100. doi:10.1016/j.vlsi.2024.102287.

Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P.l., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T.V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C.R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., Yoon, D.H., 2017. In-datacenter performance analysis of a tensor processing unit, in: Proceedings of the 44th Annual International Symposium on Computer Architecture, Association for Computing Machinery, New York, NY, USA. p. 1–12. doi:10.1145/3079856.3080246.

Kalapothas, S., Flamis, G., Kitsos, P., 2022. Efficient edge-ai application deployment for fpgas. Information 13. URL: https://www.mdpi.com/2078-2489/13/6/279, doi:10.3390/info13060279.

Kateb, F.A., Monowar, M.M., Hamid, M.A., Ohi, A.Q., Mridha, M.F., 2021. Fruitdet: Attentive feature aggregation for real-time fruit detection in orchards. Agronomy 11. doi:10.3390/agronomy11122440.

Kaur, R., Singh, S., 2023. A comprehensive review of object detection with deep learning.

doi:https://doi.org/10.1016/j.dsp.2022.103812.

Khanam, R., Hussain, M., 2024. What is YOLOv5: A deep look into the internal features of the popular object detector URL: https://arxiv.org/abs/2407.20892, arXiv:2407.20892.

Kirtas, M., Passalis, N., Oikonomou, A., Moralis-Pegios, M., Giamougiannis, G., Tsakyridis, A., Mourgias-Alexandris, G., Pleros, N., Tefas, A., 2023. Mixed-precision quantization-aware training for photonic neural networks. Neural Computing and Applications 35, 21361–21379. doi:10.1007/s00521-023-08848-8.

Lebedev, V., Lempitsky, V., 2016. Fast convnets using group-wise brain damage, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2554–2564. doi:10.1109/CVPR.2016.280.

Leng, C., Dou, Z., Li, H., Zhu, S., Jin, R., 2018. Extremely low bit neural network: Squeeze the last bit out with admm. Proceedings of the AAAI Conference on Artificial Intelligence 32. doi:10.1609/aaai.v32i1.11713.

Leroux, S., Bohez, S., Coninck, E.D., Verbelen, T., Vankeirsbilck, B., Simoens, P., Dhoedt, B., 2017. The cascading neural network: building the internet of smart things. Knowledge and Information Systems 52, 791–814. doi:10.1007/s10115-017-1029-1.

Li, H., Wang, Z., Yue, X., Wang, W., Hiroyuki, T., Meng, L., 2021. A comprehensive analysis of low-impact computations in deep learning workloads, in: Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI, Association for Computing Machinery. pp. 385–390. doi:10.1145/3453688.3461747.

Liang, T., Chu, X., Liu, Y., Wang, Y., Tang, Z., Chu, W., Chen, J., Ling, H., 2022. Cbnet: A composite backbone network architecture for object detection. IEEE Transactions on Image Processing 31, 6893–6906. doi:10.1109/TIP.2022.3216771.

Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X., 2021. Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing 461, 370–403. doi:https://doi.org/10.1016/j.neucom.2021.07.045.

Lin, J., Rao, Y., Lu, J., Zhou, J., 2017a. Runtime neural pruning, in: Neural Information Processing Systems. URL: https://api.semanticscholar.org/CorpusID:38486148.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.M., Wang, W.C., Xiao, G., Dang, X., Gan, C., Han, S., 2023. Awq: Activation-aware weight quantization for llm compression and acceleration URL: http://arxiv.org/abs/2306.00978.

Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S., 2017b. Feature pyramid networks for object detection, in: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Institute of Electrical and Electronics Engineers Inc.. pp. 936–944. doi:10.1109/CVPR.2017.106.

Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P., 2020. Focal loss for dense object detection. IEEE Transactions on Pattern Analysis and Machine Intelligence 42, 318–327. doi:10.1109/TPAMI.2018.2858826.

Lin, T.Y., Maire, M., Belongie, S.J., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L., 2014. Microsoft coco: Common objects in context. URL: https://api.semanticscholar.org/CorpusID:14113767.

Liu, B., Wang, M., Foroosh, H., Tappen, M., Penksy, M., 2015. Sparse convolutional neural networks, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 806–814. doi:10.1109/CVPR.2015.7298681.

Liu, J., Musialski, P., Wonka, P., Ye, J., 2013. Tensor completion for estimating missing values in visual data. IEEE Transactions on Pattern Analysis and Machine Intelligence 35, 208–220. doi:10.1109/TPAMI.2012.39.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C., 2016. SSD: Single shot multibox detector , 21–37doi:10.1007/978-3-319-46448-0_2.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C., 2017. Learning efficient convolutional networks through network slimming, in: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2755–2763. doi:10.1109/ICCV.2017.298.

Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T., 2019. Rethinking the value of network pruning, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=rJlnB3C5Ym.

Logictronix, 2023. YOLOv5 quantization & compilation with vitis-ai 3.0. URL: https://www.hackster.io/LogicTronix/yolov5-quantization-compilation-with-vitis-ai-3-0-for-kria-7b005d.

Maurizio, S.P., Co-Supervisor, M., Sinisi, I.C., 2024. POLITECNICO DI TORINO FPGA IMPLEMENTATION OF A VEHICLES DETECTOR Candidate Davide ALTAMORE. Master's thesis.

Mellempudi, N., Kundu, A., Mudigere, D., Das, D., Kaul, B., Dubey, P., 2017. Ternary neural networks with fine-grained quantization doi:10.48550/arXiv.1705.01462.

Misra, D., 2020. Mish: A self regularized non-monotonic activation function URL: http://arxiv.org/abs/1908.08681.

Mišić, M.J., Đurđević, M., Tomašević, M.V., 2012. Evolution and trends in gpu computing, in: 2012 Proceedings of the 35th International Convention MIPRO, pp. 289–294.

Molchanov, D., Ashukha, A., Vetrov, D., 2017. Variational dropout sparsifies deep neural networks. URL: http://arxiv.org/abs/1701.05369.

Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., van Baalen, M., Blankevoort, T., 2021. A white paper on neural network quantization URL: http://arxiv.org/abs/2106.08295.

Nelson, J., Dwyer, B., 2020. Roboflow: Computer vision tools for developers and enterprises. URL: https://roboflow.com/.

Nock, R., Nielsen, F., 2006. On weighting clustering. IEEE Transactions on Pattern Analysis and Machine Intelligence 28, 1223–1235. doi:10.1109/TPAMI.2006.168.

Nvidia, 2025. Cuda toolkit documentation 12.9. URL: https://docs.nvidia.com/cuda/https://docs.nvidia.com/cuda/.

Pachón, C.G., Ballesteros, D.M., Renza, D., 2023. An efficient deep learning model using network pruning for fake banknote recognition. Expert Systems With Applications 233, 120961. doi:10.17632/tj8kvrb.

Padilla, R., Netto, S.L., da Silva, E.A.B., 2020. A survey on performance metrics for object-detection algorithms, in: 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), pp. 237–242. doi:10.1109/IWSSIP48289.2020.9145130.

Peemen, M., Mesman, B., Corporaal, H., 2011. Efficiency optimization of trainable feature extractors for a consumer platform, in: Blanc-Talon, J., Kleihorst, R., Philips, W.,

Popescu, D., Scheunders, P. (Eds.), Advanced Concepts for Intelligent Vision Systems, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 293–304.

Philipp, G., Song, D., Carbonell, J.G., 2018. The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions URL: https://arxiv.org/abs/1712.05577, arXiv:1712.05577.

Power, K., Deva, S., Wang, T., Li, J., Eising, C., 2023. Hardware accelerators in autonomous driving. Proceedings of the Irish Machine Vision and Image Processing Conference 2023 4, 2–5.

Poyatos, J., Molina, D., Martinez, A.D., Del Ser, J., Herrera, F., 2023. Evoprunedeeptl: An evolutionary pruning model for transfer learning based deep neural networks. Neural Networks 158, 59–82. doi:https://doi.org/10.1016/j.neunet.2022.10.011.

Pytorch, 2020. BCELoss — pytorch 2.8 documentation. URL: https://docs.pytorch.org/docs/stable/generated/torch.nn.BCELoss.html.

Qian, W., Zhu, Z., Zhu, C., Zhu, Y., 2025. Fpga-based accelerator for YOLOv5 object detection with optimized computation and data access for edge deployment. Parallel Computing 124, 103138. doi:https://doi.org/10.1016/j.parco.2025.103138.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: Unified, real-time object detection, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788. doi:10.1109/CVPR.2016.91.

Redmon, J., Farhadi, A., 2017. Yolo9000: Better, faster, stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) , 6517–6525URL: https://api.semanticscholar.org/CorpusID:786357.

Redmon, J., Farhadi, A., 2018. Yolov3: An incremental improvement. ArXiv abs/1804.02767. URL: https://api.semanticscholar.org/CorpusID:4714433.

Ren, S., He, K., Girshick, R., Sun, J., 2017. Faster r-cnn: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence 39, 1137–1149. doi:10.1109/TPAMI.2016.2577031.

Ruby, U., Yendapalli, V., 2020. Binary cross entropy with deep learning technique for image classification. International Journal of Advanced Trends in Computer Science and

Engineering 9. doi:10.30534/ijatcse/2020/175942020.

Ruder, S., 2017. An overview of gradient descent optimization algorithms URL: http://arxiv.org/abs/1609.04747.

Russell, S., Norvig, P., 2009. Artificial Intelligence: A Modern Approach. 3rd ed., Prentice Hall Press, USA. URL: https://dl.acm.org/doi/book/10.5555/1671238.

Samuels, J.I., 2024. One-hot encoding and two-hot encoding: An introduction doi:10.13140/RG.2.2.21459.76327.

Sharify, S., Saxena, U., Xu, Z., Yazar, W., Soloveychik, I., Wang, X., 2024. Post training quantization of large language models with microscaling formats, in: Rezagholizadeh, M., Passban, P., Samiee, S., Partovi Nia, V., Cheng, Y., Deng, Y., Liu, Q., Chen, B. (Eds.), Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop, PMLR. pp. 241–258. URL: https://proceedings.mlr.press/v262/sharify24a.html.

Song, C., Ye, C., Sim, Y., Jeong, D.S., 2024. Hardware for deep learning acceleration. Advanced Intelligent Systems 6. doi:10.1002/aisy.202300762.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15, 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2016. Rethinking the inception architecture for computer vision. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2016-December, 2818–2826. doi:10.1109/CVPR.2016.308.

Teo, Y.S., Shin, S., Jeong, H., Kim, Y., Kim, Y.H., Struchalin, G.I., Kovlakov, E.V., Straupe, S.S., Kulik, S.P., Leuchs, G., Sanchez-Soto, L.L., 2021. Benchmarking quantum tomography completeness and fidelity with machine learning doi:10.1088/1367-2630/ac1fcb.

Terven, J., Cordova-Esparza, D.M., Romero-González, J.A., Ramírez-Pedraza, A., Chávez-Urbiola, E.A., 2025. A comprehensive survey of loss functions and metrics in deep learning. Artificial Intelligence Review 58. URL: http://dx.doi.org/10.1007/s10462-025-11198-7.

Tibshirani, R., 1996. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological) 58, 267–288. URL: http://www.jstor.org/stable/2346178.

Toğaçar, M., 2022. Using darknet models and metaheuristic optimization methods together to detect weeds growing along with seedlings. Ecological Informatics 68. doi:10.1016/j.ecoinf.2021.101519.

Tschandl, P., Rosendahl, C., Kittler, H., 2018. Data descriptor: The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Scientific Data 5, 1–9. doi:10.1038/sdata.2018.161.

Ultralytics, 2022. Github - ultralytics_yolov5 in pytorch_ onnx_ coreml_ tflite. URL: https://github.com/ultralytics/yolov5.

Vanhoucke, V., Senior, A., Mao, M., 2011. Improving the speed of neural networks on cpus. Proc. Deep Learning and ... , 1–8URL: http://research.google.com/pubs/archive/37631.pdf.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2023. Attention is all you need URL: http://arxiv.org/abs/1706.03762.

Wang, Y., Li, Y., Song, Y., Rong, X., 2020. The influence of the activation function in a convolution neural network model of facial expression recognition. Applied Sciences (Switzerland) 10. doi:10.3390/app10051897.

Wang, Z., Li, H., Yue, X., Meng, L., 2022. Briefly analysis about cnn accelerator based on fpga, in: Procedia Computer Science, Elsevier B.V.. pp. 277–282. doi:10.1016/j.procs.2022.04.036.

Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H., 2016. Learning structured sparsity in deep neural networks URL: http://arxiv.org/abs/1608.03665.

Wu, H., Judd, P., Zhang, X., Isaev, M., Micikevicius, P., 2020. Integer quantization for deep learning inference: Principles and empirical evaluation , 1–20URL: http://arxiv.org/abs/2004.09602.

Wu, J., 2017. Introduction to convolutional neural networks. URL: https://api.semant

icscholar.org/CorpusID:36074296.

Wythoff, B.J., 1993. Backpropagation neural networks: A tutorial. Chemometrics and Intelligent Laboratory Systems 18, 115–155. doi:https://doi.org/10.1016/0169-7439(93)80052-J.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., Han, S., 2024. Smoothquant: Accurate and efficient post-training quantization for large language models. URL: https://arxiv.org/abs/2211.10438, arXiv:2211.10438.

Xilinx, 2023. Vitis ai user guide. URL: https://docs.amd.com/r/3.0-English/ug1414-vitis-ai/Deep-Learning-Processor-Unit.

Xilinx, I., 2002. MicroBlaze ™ Software Reference Guide. Technical Report. URL: https://www.ee.torontomu.ca/~courses/ee8205/Data-Sheets/sopc/MicroBlaze_SW_RefrenceGuide.pdf.

Xilinx-tutorials, 2023. Developing a model — vitis™ ai 3.0. URL: https://xilinx.github.io/Vitis-AI/3.0/html/docs/workflow-model-development.html.

Yamaguchi, M., Sasaki, T., Uemura, K., Tajima, Y., Kato, S., Takagi, K., Yamazaki, Y., Saito-Koyama, R., Inoue, C., Kawaguchi, K., Soma, T., Miyata, T., Suzuki, T., 2022. Automatic breast carcinoma detection in histopathological micrographs based on single shot multibox detector. Journal of Pathology Informatics 13. doi:10.1016/j.jpi.2022.100147.

Yan, Z., Zhang, B., Wang, D., 2024. An fpga-based YOLOv5 accelerator for real-time industrial vision applications. Micromachines 15. doi:10.3390/mi15091164.

Yang, H., Tang, M., Wen, W., Yan, F., Hu, D., Li, A., Li, H., Chen, Y., 2020. Learning low-rank deep neural networks via singular vector orthogonality regularization and singular value sparsification. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops 2020-June, 2899–2908. doi:10.1109/CVPRW50498.2020.00347.

Youvan, D., 2023. Parallel precision: The role of gpus in the acceleration of artificial intelligence doi:10.13140/RG.2.2.21937.76641.

Yuan, M., Lin, Y., 2006. Model selection and estimation in regression with grouped variables. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 68. URL:

https://api.semanticscholar.org/CorpusID:6162124.

Yun, S., Han, D., Chun, S., Oh, S.J., Yoo, Y., Choe, J., 2019. Cutmix: Regularization strategy to train strong classifiers with localizable features, in: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 6022–6031. doi:10.1109/ICCV.2019.00612.

Zhang, H., Cissé, M., Dauphin, Y., Lopez-Paz, D., 2017. mixup: Beyond empirical risk minimization. ArXiv abs/1710.09412. URL: https://api.semanticscholar.org/CorpusID:3162051.

Zhou, H., Alvarez, J.M., Porikli, F., 2016. Less is more: Towards compact CNNs, in: Leibe, B., Matas, J., Sebe, N., Welling, M. (Eds.), Computer Vision – ECCV 2016. Springer International Publishing, Cham, pp. 662–677.

Zhu, C., Zhang, L., Luo, W., Jiang, G., Wang, Q., 2025. Tensorial multiview low-rank high-order graph learning for context-enhanced domain adaptation. Neural Networks 181, 106859. doi:10.1016/j.neunet.2024.106859.

Zhu, Y., Peng, H., Fu, A., Yang, W., Ma, H., Al-Sarawi, S.F., Abbott, D., Gao, Y., 2024. Towards robustness evaluation of backdoor defense on quantized deep learning models. Expert Systems with Applications 255. doi:10.1016/j.eswa.2024.124599.