REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE YAHIA FARES DE MEDEA



Faculté de **T**echnologie

Mémoire de fin d'études de Master

Filière : télécommunications
Spécialité : systèmes de télécommunication

Implémentation d'un réseau de neurones convolutionnel sur une plateforme embarquée

---

Présenté par :

M^lle^ BELAZZOUGUI Louiza

M^lle^ MELIANI Siham

Proposé et Dirigé par :

M BENGHARBIA Billel

M CHADLI Samir

Promotion 2020

Remerciements

Nous tenons tout d'abord à remercier ALLAH le tout puissant et miséricordieux, qui nous a donné la force et la patience d'accomplir ce Modeste travail.

En second lieu, nous avons eu un grand honneur d'avoir comme promoteur M BENGHERBIA Billel. Nous tenons à le remercier vivement pour ses précieux conseils et son aide pour réaliser ce projet de fin d'études.

Nous remercions également M KHEDDAR H, M CHOUCHAN A, M BAILICHE M A, MM HEDROUG R et MM SAADOUNE F Z pour nous avoir dispensé les différents cours en méthodologie qui nous ont été bénéfiques pour la rédaction de notre mémoire.

Nos vifs remerciements vont également aux membres du jury, DR. TOUBAL Abdelmoughni, le président de jury, et DR. MAAZOUZ Mohamed, l'examinateur, d'avoir accepté d'examiner notre travail et de l'avoir enrichi par leurs remarques et leurs propositions.

Nous tenons à exprimer nos sincères remerciements à tous les professeurs qui ont contribué à notre formation.

Enfin, nous tenons également à remercier toutes les personnes qui nous ont soutenu durant la réalisation de ce travail.

Dédicaces

À la mémoire de mon grand-père

À ma chère grand-mère

À mes chers parents

À mes frères et sœurs

À toute ma famille

À tous mes amis et leurs familles

À mes adorables neveux et nièces

Siham

Je dédie ce mémoire

À mes uniques parents que j'aime tellement trop fort

Pour leur patience, leur amour, leur soutien et leurs encouragements.

À mon grand-père.

À mes deux frères.

À mon adorable sœur et son mari.

À ma belle-sœur.

À tous mes ami(e)s et camarades.

Sans oublier tous les professeurs que ce soit du primaire, du moyen, du secondaire ou de l'enseignement supérieur.

Louiza

Résumé

Dans ce mémoire nous présentons une implémentation d'un réseau de neurones convolutionnel sur une plateforme embarquée. Le CNN fait une classification des chiffres numériques en utilisant un ensemble de données contenants des images de digits. Pour ceci, nous proposons un modèle d'un CNN écrit en Python dans l'environnement de développement SPYDER. Le modèle proposé subit un apprentissage puis un test pour l'implémenter sur notre plateforme Raspberry Pi 3 B+. Ceci forme un système de reconnaissance de chiffres qui est renforcé avec l'accélérateur NCS 2 d'Intel compatible avec la distribution d'OpenVINO toolkit.

Mots clés : Apprentissage Profond, CNN, Raspberry Pi 3 B+, NCS2, OpenVINO.

ملخص

في هذه المذكرة نقدم تنفيذا لشبكة عصبية التفافية على منصة مدمجة. تقوم الشبكة بتصنيف الأرقام باستخدام مجموعة بيانات تحتوي على صور للأرقام. لهذا، نقترح نموذجًا لشبكة CNN مكتوبة بلغة Python في بيئة التطوير SPYDER. يخضع النموذج المقترح للتعلم ثم للاختبار حتى يتم تنفيذه على منصتنا Raspberry Pi 3 B+. هذا يشكل نظام التعرف على الأرقام المعزز بمسرع NCS 2: Intel المتوافق مع مجموعة أدوات OpenVINO.

الكلمات المفتاحية: التعليم العميق، الشبكات العصبية الالتفافية، Raspberry Pi 3 B+، NCS 2، OpenVINO.

Abstract

In this dissertation, we present an implementation of a convolutional neural network on an embedded platform. The CNN classifies digits using a dataset containing images of digits. For this, we propose a model of a CNN written in Python in the SPYDER development environment. The proposed model is trained and tested to be implemented on our platform Raspberry Pi 3 B+. This forms a digit's recognition system that is reinforced with Intel's NCS 2 accelerator compatible with the distribution of OpenVINO toolkit.

Key words : DeepLearning, CNN, Raspberry Pi 3 B+, NCS 2, OpenVINO.

Table des matières

Liste des figures I

Liste des tableaux III

Liste des acronymes IV

Notations VII

INTRODUCTION GÉNÉRALE 1

CHAPITRE I : Apprentissage profond 2

1. Introduction 3

2. Intelligence Humaine et Intelligence Artificielle 3

3. Apprentissage automatique et apprentissage profond 5

4. Vision par ordinateur 7

5. Vision par ordinateur et apprentissage profond 11

6. Vision par ordinateur et l'informatique en périphérie 12

7. Détection d'objets en temps réel sur un système embarqué 12

8. Conclusion 13

CHAPITRE II : Réseaux de neurones convolutionnels 14

1. Introduction 15

2. Bases biologiques 15

2.1- Définition du neurone 15

2.2- Structure du neurone 16

2.3- Codage des messages nerveux 16

2.4-Transmission neuronale 17

3. Neurones formels (artificiels) 17

4. Réseaux de neurones artificiels 18

4.1- Aperçu de l'architecture de l'ANN 18

4.2- Exemples des deux réseaux de neurones les plus utilisés 19

4.2-1 Perceptron mono couche 19
4.2-2 Perceptron multicouche 21
4.3- Aperçu sur l'algorithme du réseau de neurones 22
5. Réseaux de neurones convolutionnels 24
5.1- Aperçu de l'architecture du CNN 25
5.2 Couches du CNN 26
5.2-1 Couche convolutionnelle 26
5.2-2 Couche de « Pooling » 28
5.2-3 Couche de démarrage 29
5.2-4 Couche de correction ReLU 30
5.2-5 Couche entièrement connectée 30
5.2-6 Réseau résiduel 31
5.3- Normalisation par lots 32
6. Conclusion 34
CHAPITRE III : Résultats d'implémentation 35
1. Introduction 36
2. Environnements et bibliothèques 36
2.1- Anaconda (Python Distribution) 36
2.2- SPYDER 37
2.3- TensorFlow 37
2.4- NumPy 38
2.5- OpenCV 38
3. Le langage Python 39
4. Matériel et Logiciels 39
4.1- Neural Compute Stick 2 d'Intel 39
4.2- Distribution Intel® d'OpenVINO™ toolkit 40
4.2-1 Inférence du Deep Learning 40

4.3- Raspberry Pi 3 model B+ 42
5. Étapes d'installation d'OpenVINO 42
5.1- Environnement de travail 42
5.1-1 Édition Windows 42
5.1-2 Système (facultatif) 43
5.1-3 Logicielles périphériques et requis 43
5.2- Mettre à jours les variables d'environnements Windows 43
5.3- Installer OpenVINO™ 45
5.4- Définir les variables d'environnement 47
5.5- Configurer l'optimisateur de modèle (Model Optimizer) 48
5.6- Utilisation des scripts de vérification pour vérifier l'installation 49
5.6-1 Exécution du script de vérification de la classification des images 49
5.6-2 Exécution du script de vérification du pipeline d'inférence 50
6. Ensemble de données 50
7. Implémentation 51
7.1- Phase d'apprentissage 51
7.1-1 Résultats et interprétations 54
7.2- Phase de test et de validation 55
7.2-1 TF_model et IR_model 56
7.2-2 Organigrammes 58
7.2-3 Résultats et interprétations 61
8. Conclusion 63
CONCLUSION GENERALE 64
PERSPECTIVES DE RECHERCHE 64
Références 65
ANNEXES 70
Annexe A 71

Annexe B 72
Annexe C 73
Annexe D 73
Annexe E 74
Annexe F 75
Annexe G 76
Annexe H 77
Annexe I 78

Liste des figures

Chapitre I

Figure I.1 Relation entre : AI, ML et DL 7
Figure I.2 Transfert du style nerveux 8
Figure I.3 Filtres de détection de contour 9
Figure I.4 Convolution 3 dimensions 10

Chapitre II

Figure II.1 Structure du neurone biologique 16
Figure II.2 Neurone biologique et Neurone artificiel 18
Figure II.3 Réseau de neurone 19
Figure II.4 Perceptron : structure et comportement 20
Figure II.5 Fonctions d'activation 20
Figure II.6 Modèle du perceptron multicouche 21
Figure II.7 Réseau de neurones convolutionnel 25
Figure II.8 Architecture d'un CNN 26
Figure II.9 Opération de convolution en 2D 26
Figure II.10 Opération de max pooling 29
Figure II.11 Couche de démarrage 30
Figure II.12 Allure de la fonction ReLU 30
Figure II.13 Connexion résiduelle 32

Chapitre III

Figure III.1 Neural Compute Stick 2 39
Figure III.2 Diagramme de déroulement des opérations d'OpenVINO toolkit 40
Figure III.3 Raspberry Pi 3 model B+ 42
Figure III.4 Fenêtre «Propriétés système » 43
Figure III.5 Fenêtre « Variables d'environnement » 44
Figure III. 6 Fenêtre « modifier la variable d'environnement » 44
Figure III. 7 Choix du répertoire d'installation et des composants 45
Figure III.8 Consentement pour collecter des informations 46
Figure III.9 Avertissements détectés 46
Figure III.10 Fin de la première partie de l'installation 47
Figure III.11 Configuration des variables d'environnement d'OpenVINO 47
Figure III.12 Configuration de l'optimisateur de modèle 49

Figure III.13 Script de vérification de la classification des images 49
Figure III.14 Fenêtre de visualisation d'image 50
Figure III.15 Exemples de chiffres pour chaque classe 51
Figure III.16 Digramme de la phase d'apprentissage 52
Figure III.17 Architecture du CNN 53
Figure III.18 Distribution des images d'apprentissage 54
Figure III.19 Résultats d'apprentissage 54
Figure III.20 Graphe de précision pour un CNN avec « Fashion-MNIST » 55
Figure III.21 Processus de l'implémentation 55
Figure III.22 Organigramme de conversion en modèle TF 57
Figure III.23 Couches personnalisées de l'optimisateur du modèle 57
Figure III.24 Couches personnalisées du moteur d'inférence 58
Figure III.25 Organigramme du programme test 59
Figure III.26 Organigramme de la phase prédiction 60
Figure III.27 Matériel utilisé dans le projet 61
Figure III.28 Exécution du programme de prédiction avec RPI3 B+ et NSC2 62
Figure III.29 Classification avec RPI 3 B+ et NSC2 62

Liste des tableaux

Chapitre I

Tableau I.1 Intelligence Humaine vs Intelligence Artificielle.....4
Tableau I.2 Machine Learning vs Deep Learning 6

Chapitre II

Tableau II.1 Transformations de la normalisation par lots.....34

Chapitre III

Tableau III.1 Résumé des résultats des performances.....61
Tableau III.2 Spécifications techniques de NCS2 d'Intel® 76
Tableau III.3 Spécifications techniques du RPI 3 B+.....77

Liste des acronymes

AI	Artificial Intelligence (fr : Intelligence Artificielle)
ANN	Artificial Neural Network (fr : réseau de neurones Artificiel)
API	Application Programming Interface (fr : interface de programmation d'application
APSoC	All Programmable System on Chip (fr : Tout les Systèmes Programmable sur Puce)
ARM	Acorn Risc Machine
BLE	Bluetooth Low Energy (fr : Bluetooth à basse énergie)
BN	Bach Normalization (fr : Normalisation des Lots)
BSD	Berkeley Software Distribution (fr : la Distribution de Logiciels de Berkeley)
CNN	Convolutional neural network (fr : Réseau de neurones Convolutionnel)
CV	Computer Vision (fr : Vision par Ordinateur)
DL	Deep Learning (fr : Apprentissage Profond)
FPGA	Field Programmable Arrays (fr : Réseau Gate de Portes Programmables)
GPU	Graphics Processing Unit (fr : Processeur Graphique)
HDR	High-Dynamic Range (fr : Large Gamme Dynamique)
HI	Human Intelligence (fr : Intelligence Humaine)
IDE	Integrated Development Environment (fr : Environnement de Développement Intégré)
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoT	Internet of Things (fr : Internet des Objets)
IR	Intermediate Representation (fr : Représentation Intermédiaire)

ML	Machine Learning (fr : Apprentissage Automatique)
MLP	MultiLayer Perceptron (fr : Perceptron Multicouche)
MNIST	Modified National Istitute of Satndards and Technology
NCS	Neural Compute Stick
NCS2	Neural Compute Stick 2
NLP	Natural Language Processing (fr :Traitement du Langage Naturel)
NST	Neural Style Transfer (fr : Transfert du Style Nerveux)
OpenCV	Open Source Computer Vision Library (fr : bibliothèque de logiciels de vision par ordinateur à code source ouvert)
OS	Operating System (fr : Système d'exploitation)
PA	Potentiels d'Action
PDSF	Prix de Détail Suggéré par le Fabricant
PR	Potentiels de Repos
ReLU	Rectified Linear Units (fr : Unité de Rectification Linéaire)
RGB	Red, Green, Blue (fr : Rouge, Vert, Bleu)
RNN	Recurrent Neural Network (fr : réseau de neurones récurrents)
RPI	Raspberry Pi
RPI 3	Raspberry Pi 3
SBC	Single Board Computer (fr : ordinateur mono-carte)
SoC	System on Chip (fr : Système sur Puce)
SOM	System-on-Module (fr : System Sur Module)

SPYDER	Scientific PYthon Development EnviRonment (fr : Environnement de Développement Scientifique de Python)
STL	Standard Template Library
TF	TensorFlow
TPU	Tensor Processing Unit (fr : Unité de Traitement de Tenseur)
USB	Universal Serial Bus
VPU	Vision Processing Unit (fr : Unité de Traitement de la Vision)
WLAN	Wireless Local Area Network (fr : réseau local sans fil)

Notations

Symbole	Définition
$\boldsymbol{\varepsilon}$	Constante ajoutée à la variance du mini-lot pour la stabilité numérique
$\boldsymbol{\partial}/_{\boldsymbol{\partial x}}$	Dérivée partielle par rapport à x
$\mathbb{R}$	Ensemble des nombres réels
$\boldsymbol{E}(.)$	Espérance mathématique
$\boldsymbol{\Phi}(.)$	Fonction d'activation
$\boldsymbol{\mu_{\mathcal{B}}}(.)$	Fonction de la moyenne du mini-lot
$\boldsymbol{BN}(.)$	Fonction de la normalisation par lots
$\boldsymbol{\sigma_{\mathcal{B}}^2}$	Fonction de la variance du mini lot
$\boldsymbol{y_i}$	Fonction de mise à l'échelle et décalage
$\widehat{\boldsymbol{x}}_{\boldsymbol{i}}$	Fonction de normalisation du mini-lot
$\boldsymbol{\mathcal{L}}$	Fonction de perte appliquée
$\boldsymbol{\mathcal{B}}$	Mini-lot
$\lfloor . \rfloor$	Operateur de l'arrondi vers le bas
$\circledast$	Operateur de produit de convolution
$\|\|.\|\|$	Valeur absolue d'un vecteur
$\boldsymbol{Var}(.)$	Variance statistique

INTRODUCTION GÉNÉRALE

Un système de télécommunication, que ce soit simple ou complexe, est toujours composé d'un émetteur, un récepteur et un canal de transmission (canal physique ou l'air libre) à travers lequel l'information est transmise. La taille de celle-ci ne cesse d'augmenter considérablement. Donc elle ne peut pas être traitée avec la même technologie et les mêmes systèmes. Ces systèmes ont eu le besoin d'être améliorés pour faire face aux avancées technologiques. En effet, les télécommunications ont fait un grand pas dès qu'ils se sont associés aux réseaux informatiques et à l'électronique. Ces derniers, ont beaucoup apporté en termes de débit, performances, efficacité et surtout en ce qui concerne le fonctionnement en temps réel. C'est à partir de là que viennent les systèmes bourrés d'électronique plus ou moins complexe et d'informatique plus ou moins évoluée. Ce sont des systèmes autonomes, qui sont dédiés à faire une tâche bien précise. C'est la définition qu'on puisse donner à un système embarqué dont le succès est lié à celui des processeurs qui sont de plus en plus rapide. Cela est confirmé par la loi de Gordon MOORE, cofondateur d'Intel, stipulant que pour une surface de silicium donnée, le nombre de transistors qui y sont intégrés est doublé tous les 18 mois. Sachant que depuis 2016, l'usage de cette loi est abandonné : désormais on mise sur les performances des microprocesseurs plutôt que sur leur nombre. En effet, un microprocesseur 10ème génération d'Intel (10th Gen Intel® Core™ mobile processors) regroupe d'incroyables capacités, notamment jusqu'à 5,3 GHz, la technologie Intel® Wi-Fi 6 (Gig+), le HDR 4K, l'optimisation intelligente des systèmes, et bien plus encore[1]. Ceci nous a poussé à songer à la manière dont on pourrait accroître les capacités d'un système embarqué sans pour autant modifier ses caractéristiques. La finalité en est de pouvoir l'utiliser pour les applications d'Intelligence Artificielle (Artifictial Intelligence), en particulier celles d'Apprentissage Profond (Deep Learning). Intel répond à notre besoin en proposant l'accélérateur Neural Compute Stick 2 (NCS2).

L'objectif de notre projet de fin d'études consiste en la réalisation d'un système qui fera une reconnaissance des digits grâce à un réseau de neurones convolutionnel (CNN). L'implémentation de ce système se fera sur une plateforme embarquée Raspberry Pi 3 B+ (RPI 3 B+) et une clé d'Intel Neural Compute Stick 2 (NCS2).

Le travail présenté dans ce mémoire est organisé comme suit :

Le premier chapitre est une introduction à l'apprentissage profond en se référant à l'Intelligence Humaine et Artificielle ainsi qu'à l'apprentissage automatique (Machine Learning) ;

Le deuxième chapitre concernera les réseaux de neurones artificiels, en particulier les réseaux de neurones convolutionnels. Il décrira aussi les parties matérielle et logicielle nécessaires pour la réalisation de notre projet ;

Le troisième chapitre exposera les résultats d'implémentation du programme sur le RPI 3 B+.

CHAPITRE I : Apprentissage profond

1. Introduction

L'apprentissage profond (Deep Leaarning : DL) est l'un des types d'apprentissage automatique (Machine Learning : ML) qui se base sur les réseaux de neurones profonds (Deep Neural Network : DNN)[2]. Il permet aux modèles de calcul, composés de plusieurs couches de traitement, d'apprendre à représenter des données en plusieurs niveaux. Cela a considérablement amélioré la reconnaissance vocale, la reconnaissance visuelle des objets et leur détection ainsi que de nombreux autres domaines. Les réseaux convolutionnels profonds ont permis des percées dans le traitement des images, de la vidéo, de la parole et de l'audio, tandis que les réseaux récurrents ont mis en lumière des données séquentielles telles que le texte et la parole[3]. Pour que tout cela soit possible, le DL nécessite que la machine soit puissante et rapide pour qu'elle puisse gérer le grand nombre d'opérations mathématiques effectué. Parmi ses nombreuses utilisations on trouve : la classification, le renforcement et la régression[2].

Dans ce chapitre on parlera d'abord de la relation du DL avec les différents types d'intelligence. Ensuite, on fera une comparaison entre le DL et le ML. Puis, on fera une introduction sur l'informatique en périphérie dans le domaine de la Vision par Ordinateur. Enfin, on définira la détection d'objet en temps réel sur un système embarqué.

2. Intelligence Humaine et Intelligence Artificielle

Pour mieux comprendre le DL il est nécessaire de connaitre d'où il vient et à quel domaine il appartient. Pour ce faire, on présentera brièvement en premier lieu l'Intelligence Humaine (Human Intelligence : HI), ensuite l'Intelligence Artificielle (Artificial intelligence : AI).

La HI est une combinaison complexe des différents modes de traitement de l'information par les systèmes naturels (cerveau, neurones, individus) qui sont à la fois objectifs et subjectifs. Les processus objectifs comprennent les communications orales et verbales, le raisonnement, la formation des concepts, la prise des décisions et la résolution des problèmes. Chacun de ces éléments intègre la subjectivité qui s'illustre dans les émotions, la motivation, la conscience kinesthésique, la perception et les préjugés. Pour développer une telle intelligence, un enseignement de qualité élevée est nécessaire pour qu'elle soit meilleure. Cela s'explique du fait que la HI se base sur l'imposition des vérités sans fournir des informations (données) convaincantes qui permettent l'auto vérification et la crédibilité des concepts, ce qui veut dire que l'apprentissage sera focalisé sur la mémorisation des connaissances sans compréhension profonde. Cela n'est pas efficace lorsqu'on essaie de résoudre de nouveaux problèmes car ils

nécessiteront probablement beaucoup plus d'expérience pour pouvoir manipuler davantage d'idées et de concepts.

De cette HI est créée l'AI qu'on peut définir comme étant n'importe qu'elle technique permettant aux machines (ordinateurs) d'imiter le comportement humain[3]. Elle est conçue pour effectuer une tâche spécifique avec un niveau de performance surhumain. Elle réunit de nombreux domaines scientifiques dont la philosophie, la psychologie, les neurosciences, l'informatique, les statistiques, la linguistique et d'autres encore. Ce type d'intelligence est généralement abordé de deux manières. La première a été traditionnellement utilisée pendant des années dans le passé où un système d'algorithmes basé sur des règles est codé en dur, bien que cela puisse toujours être considéré comme le cas dans les applications intelligentes d'aujourd'hui. L'autre approche moderne est l'auto-apprentissage (apprentissage automatique), où les algorithmes sont conçus pour examiner les données et apprendre les modèles sous-jacents, les relations et les transformations tous seuls sans qu'ils soient explicitement informés de quoi que ce soit au préalable. Afin d'améliorer l'AI, dans l'approche moderne, les efforts sont consacrés à des structures de système sophistiquées ainsi qu'à des algorithmes d'optimisation avancés permettant de capturer des informations complexes cachées dans la nature des données. Le niveau d'intelligence d'un système peut être évalué à l'aide de métriques de performance qui définissent plusieurs composants liés aux domaines pour lesquels le système est conçu[4]. Le tableau I.1 ci-dessous montre la différence entre la HI et l'AI :

Tableau I.1 Intelligence Humaine vs Intelligence Artificielle[5]

	Intelligence Humaine	Intelligence Artificielle
Efficacité énergétique	25 Watts pour le cerveau humain	2 Watts pour une machine moderne, machine d'apprentissage
Général	Les humains apprennent généralement à gérer des centaines de compétences différentes au cours de leur vie	Pendant qu'elle consomme des kilowatts d'énergie, une machine est généralement conçue pour quelques tâches
Multitâche	Un homme travailleur assume de multiples responsabilités	Le temps nécessaire pour atteindre le système pour chaque responsabilité est considérablement élevé
La prise de décision	Les humains ont la capacité d'apprendre la prise de décision à partir de leurs expériences	Même les robots les plus avancés peuvent difficilement rivaliser, en mobilité, avec un enfant de 6 ans
État	Les cerveaux sont analogiques	Les ordinateurs sont numériques

3. Apprentissage automatique et apprentissage profond

L'apprentissage automatique ou Machine Learning (ML) est l'ensemble des techniques d'intelligence artificielle qui permettent à un ordinateur d'apprendre sans qu'il en soit explicitement programmer pour le faire[2]. Ou bien comme l'avait défini Arthur Samuel, un pionnier de la recherche en intelligence artificielle : « le domaine d'étude qui donne aux ordinateurs la capacité d'apprendre sans être explicitement programmé ». Plus tard, Tom Mitchell, un informaticien scientifique, a fourni une définition plus moderne qui est : « un programme informatique est censé apprendre de l'expérience E en ce qui concerne une classe de tâches T et une mesure de performance P, si sa performance dans les tâches T, mesurée par P, s'améliore avec l'expérience E. ». Bien qu'il existe depuis longtemps, ce n'est qu'aujourd'hui que le ML est devenu efficace. Cela est dû à l'évolution des nouvelles technologies informatiques qui sont désormais plus rapides en traitement de données et cela avec une quantité beaucoup plus importante que jamais. Par conséquent, le ML est de plus en plus recherché et est devenu d'une grande importance dans de nombreux documents de recherche adoptant tous les aspects de vie dont les finances, les soins de santé, la vision par ordinateur, la reconnaissance vocale robotique. En général, tout sujet d'apprentissage automatique peut être classé en deux catégories : apprentissage supervisé et apprentissage non supervisé. Sachant les données d'entrée et celles de sortie, l'idée de l'apprentissage supervisé est de trouver sous forme d'équation mathématique une relation qui fait une correspondance entre eux. Il y a deux catégories dans ce type d'apprentissage qui sont la régression et la classification. Un problème est classé comme régression si les résultats prévus sont continus. Sinon, le problème est classé comme une classification : les sorties prévues sont des valeurs discrètes (catégories). Dans le cas d'apprentissage non supervisé, l'objectif de prédiction est de dériver la structure des données en les regroupant sur la base des relations entre les variables plutôt que de considérer l'effet de chaque variable. Par conséquent, il n'y a pas de rétroaction sur les résultats de prédiction. La plupart des applications technologiques convergent vers l'utilisation de l'apprentissage supervisé, en particulier des méthodes de l'apprentissage profond. Ils comprennent généralement une architecture de réseau d'inspiration neuronale (réseau de neurones multicouches) conçue avec diligence et optimisée par un algorithme avancé qui permet au réseau d'apprendre (trouver le meilleur ensemble de règles) et de prédire des données structurées (c'est-à-dire étiquetées). Le réseau s'améliore en étant formé de manière itérative sur une énorme quantité de données avec une grande diversité dans le domaine du ciblage pour en acquérir de l'expérience. Par conséquent, non seulement les systèmes d'apprentissage profond ont soif de données, ils nécessitent également une quantité considérable de temps de formation

et dépendent fortement de grandes ressources de calcul. Pour cette raison, ils ont tendance à être développés sur une machine très performante avec un matériel puissant et déployés sur un système Cloud pour utilisation. L'inconvénient d'un tel système est qu'il nécessite une connexion réseau stable et une faible latence. Donc, il est préférable d'utiliser des appareils individuels puissants qui supportent les systèmes de DL[4]. Le tableau I.2 suivant montre quelques différences entre le ML et le DL :

Tableau I.2 Machine Learning vs Deep Learning[6-8]

Caractéristiques	Deep Learning	Machine Learning
Comment ça marche	Intercepte les caractéristiques des données et leur relations à l'aide de réseaux de neurones qui transmettent les informations pertinentes à travers plusieurs étapes de traitement des données	Utilise des types d'algorithmes automatisés qui apprennent à prédire les décisions futures et à modéliser les fonctions en utilisant les données qui lui sont fournies
Quantité de données	Nécessite une grande quantité de données d'apprentissage	Nécessite peu de données d'apprentissage
Dépendance aux données	Performances exceptionnelles sur un grand ensemble de données	Grandes performances sur un ensemble de données de petite à moyenne taille
Gestion de données	Une fois mis en œuvre, les algorithmes sont généralement autogérés pour l'analyse des données pertinentes	Les différents algorithmes sont gérés par l'analyse pour examiner les différentes variables dans les ensembles de données
Extraction des fonctionnalités	Effectue l'extraction automatique des fonctionnalités	L'extraction des fonctionnalités n'est pas automatique, elle nécessite des paramètres étiquetés
Gestion des fonctionnalités	Il n'est pas nécessaire de bien comprendre les fonctionnalités pour représenter les données	La compréhension des fonctionnalités est nécessaire pour représenter les données
Création des fonctionnalités	Peut créer de nouvelles fonctionnalités	A besoin de caractéristiques bien identifiées manuellement
Nombre de couches	Nécessite au moins trois couches	Il suffit d'une entrée, une sortie et seulement une seule couche cachée
Dépendance au matériel	Nécessite un matériel de haute performance	Ne nécessite pas un matériel de haute performance
Temps d'apprentissage	Prend beaucoup de temps	Prend moins de temps
Temps d'exécution	Peu prendre des semaines plus le processus est long	Va de quelques minutes à quelques heures
Interopérabilité	Très difficile	Certain algorithmes peuvent être interprétés facilement tandis que d'autres sont impossibles

Selon le type d'algorithme, la sortie d'un système utilisant le ML ou le DL diffère. L'apprentissage, que ce soit en ML ou en DL, peut être :

1. Supervisé
    - Algorithmes de classification : classent des objets en catégories ;
    - Algorithmes de régression : prédissent une valeur numérique.
2. Non-supervisé
    - Algorithmes de regroupement : classent les données en groupes ;
    - Algorithmes d'association : découvrent les lois unissant les données d'un ensemble de données[9].

La figure I.1 qui suit montre la relation entre l'AI le ML et DL :

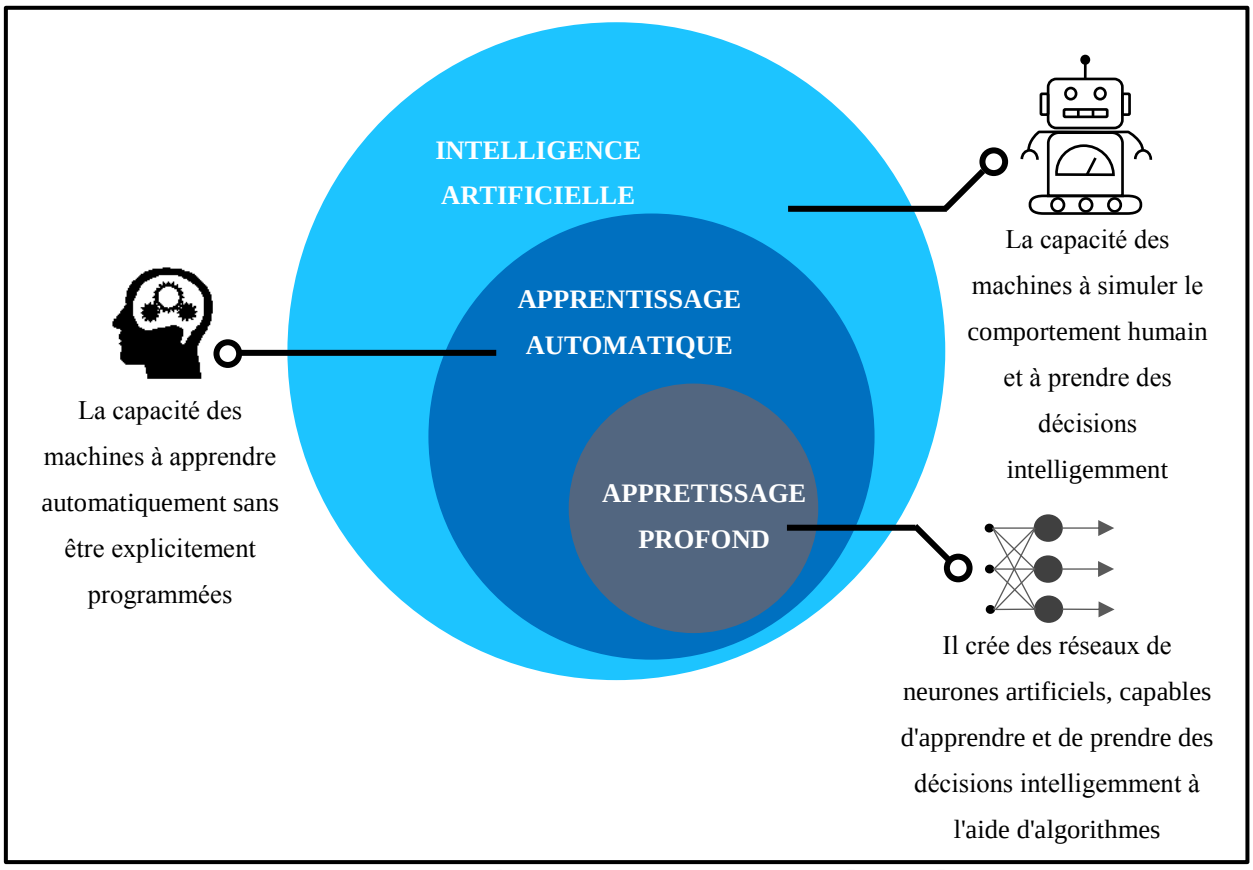


Figure I.1 Relation entre : AI, ML et DL[10, 11]

4. Vision par ordinateur

La vision par ordinateur ou Computer Vision (CV) est une branche de l'AI qui permet à une machine d'analyser, traiter et comprendre une ou plusieurs images prises par un système d'acquisition (par exemple : une caméra). Elle est utilisée dans plusieurs domaines :

- Classification d'image : l'algorithme précise le contenu de l'image ;

- Détection d'objets : l'algorithme fait la localisation de l'objet, précise son emplacement et sa distance par rapport à la caméra, puis l'entour par un carré. Il est possible que plusieurs objets soient présents dans la scène. Dans ce cas l'algorithme suit le même processus sur chaque objet ;
- Transfert du style nerveux : ou Neural Style Transfer (NST) fait référence à une classe d'algorithmes logiciels qui manipulent des images numériques ou des vidéos, pour adopter l'apparence ou le style visuel d'une autre image. Les algorithmes NST sont caractérisés par leur utilisation de réseaux de neurones profonds pour effectuer la transformation d'image.

  La figure I.2 suivante schématise le processus :



Figure I.2 Transfert du style nerveux

La vision par ordinateur utilise l'opération de la détection de contour. Cela se fait en effectuant un scan horizontal et un scan vertical à l'image de taille $(N \times N)$ grâce à des filtres de taille $(F \times F)$ ensuite en faisant une convolution selon l'équation (1.1) suivante :

$$(N \times N) \circledast (F \times F) = (L \times L) \qquad (1.1)$$

Avec :

N - dimension de l'image originale ;

F - dimension du filtre ;

L - dimension de l'image après convolution (image résultante) avec : $L = (N - F) + 1$;

$\circledast$ - operateur du produit de convolution.

D'après l'équation (1.1), on remarque bien que l'image résultante est de taille réduite par rapport à l'image originale.

En programmation, la convolution se fait avec une instruction selon le langage ou la bibliothèque utilisée :

- Python : « conv-forward » ;
- Tensorflow : « tf.nn.conv2d » ;
- Keras : conv2d.

Dans la figure I.3, les filtres des contours verticaux qui sont représentés par la matrice (a) et les filtres des contours horizontaux qui sont représentés par la matrice (b) sont recommandés. D'autres filtres utilisant des coefficients différents en été mis en œuvre comme celui de SOBEL (matrice (c)) et celui de SHARR (matrice (d)).

1	0	-1
1	0	-1
1	0	-1

a. Contour vertical

1	1	1
0	0	0
-1	-1	-1

b. Contour horizontal

1	0	-1
2	0	-2
1	0	-1

c. Filtre de SOBEL

3	0	-3
10	0	-10
3	0	-10

d. Filtre de SHARR

Figure I.3 Filtres de détection de contour

Afin d'obtenir les dimensions de l'image résultante identiques à celles de l'image originale, on utilise l'opération de Padding qui consiste à l'ajout du même nombre de lignes et de colonnes (ou couches) autour de l'image. En fonction de la valeur du padding on distingue trois types de convolution :

- La convolution valide (valid convolution) : c'est un type de convolution qui n'utilise aucun padding sur l'image d'entrée. La convolution dans ce cas s'effectue selon l'équation (1.1).
- La convolution égale (same convolution) : applique un padding sur l'image d'entrée de sorte qu'elle soit de même taille que l'image de sortie. Dans ce cas la valeur de padding est : $P = \frac{F-1}{2}$ et l'équation de convolution devient :

$$(N + 2P) \times (N + 2P) \circledast (F \times F) = [(N + 2P - F + 1) \times (N + 2P - F + 1)]|_{P=\frac{F-1}{2}}$$

$$= N \times N \qquad (1.2)$$

- La convolution en escalier (strided convolution) : Dans ce type de convolution, au lieu de scanner complètement la matrice d'origine et de se déplacer horizontalement et verticalement pas à pas, le filtre peut se déplacer de deux ou plusieurs pas S au lieu d'un

seul. Dans ce cas, la matrice (image) de sortie sera d'une taille réduite par rapport à l'originale. En effet, le produit de convolution donne l'image résultante de dimensions :

$$\left[\frac{N+2P-F}{S}+1\right]\times\left[\frac{N+2P-F}{S}+1\right]$$

Pour scanner toute la matrice, les dimensions de l'image résultante se calculent par la valeur arrondie vers le bas c'est-à-dire :

$$\left\lfloor\frac{N+2P-F}{S}+1\right\rfloor\times\left\lfloor\frac{N+2P-F}{S}+1\right\rfloor$$

Dans le cas des images RGB composées de trois matrices donc de trois dimensions : hauteur, largeur et canaux, la matrice du filtre doit avoir la même profondeur (nombre de canaux) pour que la convolution soit possible. La matrice résultante de ce produit de convolution sera donc en deux dimensions (figure I.4).

Parfois, on a besoin de faire un filtre pour une couleur précise, dans ce cas, on choisit un canal dans le filtre pour la couleur voulue selon sa position et les autres canaux seront remplacés par des matrices nulles. Si on veut faire la détection des contours horizontaux puis verticaux, on aura besoin d'un filtre pour chaque opération. Dans ce cas, la matrice résultante sera composée de deux couches. Plus on augmente le nombre de filtres plus le nombre de ces couches augmentent. On obtient ainsi l'équation de convolution suivante :

$$(N\times N\times Nc)\circledast(\mathrm{F}\times\mathrm{F}\times Nc)=(N-F+1)\times(N-F+1)\times Nf \qquad (1.3)$$

Avec :

Nc - nombre de canaux ;

Nf - nombre de filtres.

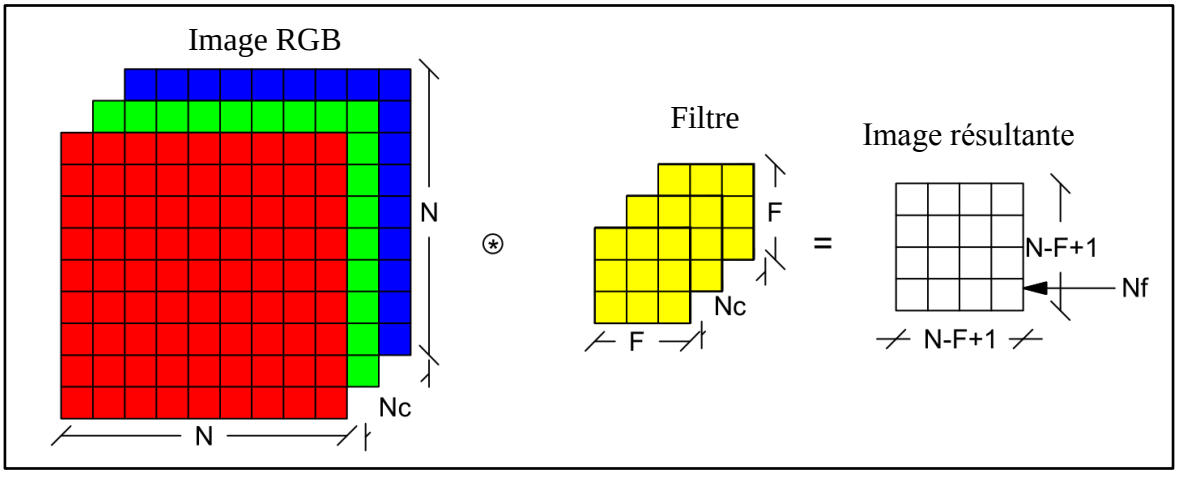


Figure I.4 Convolution 3 dimensions[12]

L'inconvénient des algorithmes de la vision par ordinateur est qu'ils ne traitent que des images de petite taille[12].

5. Vision par ordinateur et apprentissage profond

L'apprentissage profond a fait d'énormes progrès dans les différents secteurs du domaine de la vision par ordinateur, tels que la classification d'images, la détection d'objets, la segmentation sémantique, la génération d'images et l'estimation de position. Généralement, les tâches dans ses secteurs s'effectuent en utilisant les réseaux de neurones profonds : convolutionnels (CNN) ou récurrents (RNN). Ces réseaux s'entraînent sur un grand ensemble de données d'images au niveau des pixels. Plus la taille de l'image est grande, mieux est la précision de l'entrainement. Cependant le coût de calcul devient plus élevé. La plupart des systèmes des réseaux de neurones fonctionnent très lentement. Pour y remédier la mise en œuvre d'un matériel compatible avec le parallélisme[1] tel que le processeur graphique (Graphics Processing Unit (GPU)) est nécessaire. Cela augmente la vitesse d'exécution du système et optimise davantage les algorithmes utilisés. Ce qui a permis aux systèmes de vision par ordinateur d'être plus performants en temps réel. De nombreux fabricants de matériel ont de leur part apporté de grandes améliorations en termes de puissance de traitement pour soutenir l'avancement du DL[4]. Par exemple, la société Intel a introduit la série Neural Compute Stick (NCS) qui est un accélérateur de calcul d'apprentissage en profondeur très efficace doté de sa propre unité de traitement de la vision (Vision Processing Unit (VPU)). NCS permet le développement d'applications pour les périphériques et la compatibilité entre le matériel d'Intel avec celui d'autres fabricants[13]. Nvidia a lancé des GPU de plus en plus puissants, avec plus de cœurs CUDA qui augmentent la capacité de calcul parallèle. Elle a aussi fourni des ordinateurs Jetson System-on-module (SOM). Ces ordinateurs regroupent ces performances GPU pour alimenter toutes les applications graphiques et de vision par ordinateur pour les systèmes embarqués. Les appareils Jetson sont programmés par le SDK Jetpack[14]. De même, Google propose une unité de traitement de tenseur (Tensor Processing Unit (TPU)) sur son infrastructure Cloud ainsi que Google Coral avec des cœurs Edge TPU dédiés. Edge TPU est une puce ASIC conçue sur mesure par Google pour exécuter des solutions d'AI à la périphérie du réseau. Elle offre de hautes performances avec un minimum d'encombrement et de consommation énergétique[15]. Le Coral contient une carte de développement similaire au Raspberry Pi (RPI) mais avec un SOM amovible et un accélérateur USB similaire au NCS et avec d'autres composants d'extension. Il fonctionne sur son système d'exploitation officiel,

[1] En informatique, le parallélisme consiste à mettre en œuvre des architectures d'électronique numérique et des algorithmes permettant de traiter des informations de manière simultanée dans le but de réaliser le plus grand nombre d'opérations en un temps le plus court possible.

Mendel Linux de Google. Il peut également fonctionner sur toutes les distributions Linux basées sur Debian[2] et est programmé dans le cadre TensorFlow de Google.

6. Vision par ordinateur et l'informatique en périphérie[3]

Classiquement, les systèmes ML sont déployés dans un paradigme de l'informatique centralisé Cloud où les services côté utilisateur envoient des demandes d'inférence au serveur et attendent de recevoir le résultat. Cela permet de tirer profit des ressources de calcul du côté serveur et permet l'évolutivité des logiciels qui utilisent le système ML pour une gamme plus large d'appareils : PC, appareils mobiles et en particulier les systèmes embarqués. Cependant, cette configuration nécessite une connexion Internet stable. Or les protocoles de communication et les applications fonctionnant sur un flux constant d'entrée vidéo pénalisent fortement la latence. En effet, ils surchargent le support réseau pendant les processus de transfert de données et ralentissent les performances d'exécution. Afin de surmonter cette difficulté, de grands efforts doivent être consentis pour étendre la puissance du traitement en informatique de périphérie et la décentralisation des processus ML vers la périphérie du réseau. Cela bénéficiera considérablement aux applications de missions critiques et sensibles aux retards telles que la conduite autonome. En effet, dans les voitures autonomes, la prise de décision immédiate est extrêmement cruciale. Elles doivent être pleinement conscientes de l'environnement et de l'état de toutes les autres voitures à proximité pour exécuter les bonnes décisions. Par exemple, en présence d'un feu de circulation rouge ou de piétons traversant la route, la réaction d'arrêt et la notification au véhicule doivent être exécutées immédiatement. Cette décentralisation est devenue un centre d'intérêt pour des principales organisations technologiques. Ces dernières offrent des solutions qui, lorsqu'elles seraient fusionnées avec des modèles de vision par ordinateur en temps réel, deviendraient un module très puissant. Ainsi sera ouverte la voie à de nombreuses innovations technologiques, dans lesquelles la robotique aurait un impact significatif dans le développement de la perception visuelle artificielle[4].

7. Détection d'objets en temps réel sur un système embarqué

Actuellement, les applications d'AI basées sur les contours et les applications du DL optimisées pour la CV deviennent plus efficaces et plus performantes en ce qui concerne le

[2] Un système d'exploitation et une distribution de logiciels libres lancé en août 1993 par Ian MURDOCK[16] Debian. (2020). *Qu'est-ce que Debian finalement ?* Disponible: https://www.debian.org/intro/about.fr.html

[3] L'informatique en périphérie ou informatique en périphérie de réseau (edge computing), est une méthode d'optimisation employée dans le Cloud computing qui consiste à traiter les données à la périphérie du réseau, près de la source des données.

fonctionnement en temps réel. Ceci a entrainé le développement de systèmes de détection d'objets en temps réel dans des environnements tels que SPYDER sur une carte RPI 3 B+ avec un accélérateur NCS2 d'Intel. Par exemple dans une application basée sur les réseaux de neurones, l'implémentation de l'algorithme correspondant comprend les étapes suivantes :

- La reconstruction du modèle Keras dans SPYDER ;
- L'enregistrement de ce modèle en un modèle « .p » ;
- Sa conversion en un modèle TF ;
- En extraire le graphique de modèle de représentation intermédiaire (Intermediate Representation (IR)) compatible avec OpenVINO toolkit.

Les programmes de détection d'objets développés utilisent OpenVINO toolkit et NCS afin de traiter les prédictions sur des images d'entrée[4].

8. Conclusion

Dans ce chapitre, on a vu deux branches de l'Intelligence Artificielle qui sont l'apprentissage automatique et l'apprentissage profond. Ces deux derniers se diffèrent en termes de quantité de données, puissance du matériel requis, temps (d'apprentissage et d'exécution), approche de résolution de problèmes, … Néanmoins, les deux utilisent les mêmes techniques d'apprentissage : supervisé et non-supervisé. On a également palé d'une autre branche de l'AI qui est la vision par ordinateur. On a spécifié ses différents domaines : la classification d'image, le transfert du style nerveux ainsi que la détection d'objets en temps réel sur les systèmes embarqués.

CHAPITRE II : Réseaux de neurones convolutionnels

1. Introduction

Dans le chapitre précédent, nous avons vu la notion de fonctionnalités en vision et les méthodes classiques utilisées pour faire de la classification d'images. Dans ce chapitre nous allons présenter les types de réseaux de neurones mis en œuvre pour améliorer les résultats de cette classification.

En 2012, lors de la compétition annuelle de vision par ordinateur ILSVRC[4], un nouvel algorithme de DL a été mis en œuvre. Il s'agit d'un réseau de neurones convolutionnel appelé AlexNet[5]. La méthodologie qu'utilisent les réseaux de neurones convolutionnels est similaire à celle des méthodes traditionnelles d'apprentissage supervisé. Ces réseaux reçoivent des images en entrée, détectent les fonctionnalités (caractéristiques) de chacune d'entre elles, puis y entraînent un classificateur. Cependant, l'apprentissage des fonctionnalités se fait automatiquement. Les CNN réalisent eux-mêmes l'extraction et la description de ces fonctionnalités. Lors de la phase d'entraînement, l'erreur de classification est minimisée afin d'optimiser les paramètres du classificateur et les fonctionnalités. De plus, l'architecture spécifique du réseau permet d'extraire les caractéristiques d'images de différentes complexités ; des plus simples au plus sophistiquées. Leur extraction et leur hiérarchisation automatiques, qui s'adaptent au problème donné, constituent une des forces des réseaux de neurones convolutionnels. De ce fait, on a plus besoin d'implémenter manuellement un algorithme d'extraction[17].

2. Bases biologiques

2.1- Définition du neurone

Les neurones représentent l'unité structurale et fonctionnelle du système nerveux. La particularité du neurone est que le potentiel de membrane peut varier au cours du temps. Il passe d'un état de repos caractérisé par un **potentiel de repos** (PR) à un état d'activité caractérisé par un **potentiel d'action** (PA) en un temps extrêmement bref, avant de retrouver son potentiel d'équilibre. Le neurone est caractérisé par deux propriétés inséparables : l'excitabilité et la conductibilité. La première lui permet d'émettre un potentiel d'action à la suite d'une excitation supraliminaire, la seconde de propager ce même potentiel d'action jusqu'à l'arborisation terminale de l'axone sans atténuation[18].

[4] ILSVRC est une compétition phare de vision par ordinateur, qui se tient chaque année depuis 2010. Organisé par le projet ImageNet, qui produit une large base d'images annotées pour faire avancer la recherche en CV.

[5] AlexNet a été développé par des chercheurs de Toronto en 2012, mais est largement inspiré du réseau de neurones convolutionnel LeNet, développé en 1998 par le chercheur français Yann LeCun.

2.2- Structure du neurone

Le neurone est une cellule spécialisée dans le traitement et la transmission de l'information, ce qui reflète sa morphologie très particulière. La figure II.1 représente la structure du neurone biologique. Ce dernier se compose d'un corps cellulaire, le soma, et d'expansions : les dendrites d'une part, et l'axone d'une autre part. Les dendrites forment des ramifications qui rentrent en contact avec d'autres neurones, typiquement de l'ordre de 10000. Leur rôle est de recevoir des informations (électriques ou chimiques) provenant d'autres neurones. L'axone est un prolongement de quelques millimètres qui conduit un signal électrique jusqu'à son arborisation terminale, où il peut alors entrer en contact avec les dendrites d'autre neurones. La jonction axone-dendrite est dite synapse[19].

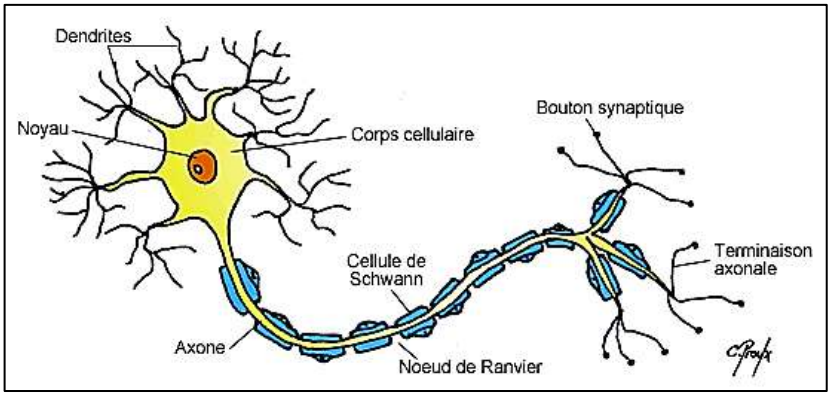


Figure II.1 Structure du neurone biologique[20]

2.3- Codage des messages nerveux

Selon leur origine et/ou leur destination, les messages nerveux véhiculés par les axones sont de trois ordres. Ils peuvent être :

- Sensitifs (issus d'un récepteur sensoriel) ;
- Moteurs (à destination d'une cellule effectrice) ;
- Centraux (entre deux neurones à l'intérieur du névraxe).

Ils présentent tous le même système de codage. En effet, l'amplitude des potentiels d'action ne variant pas en raison de la loi du tout ou rien et la vitesse de propagation étant constante pour une fibre donnée, l'information est codée par :

- ✓ Le nombre de potentiels émis ;
- ✓ Leur fréquence ;
- ✓ Leur organisation en salves ;
- ✓ La durée du message[21].

2.4-Transmission neuronale

Les neurones sont des cellules indépendantes qui communiquent entre elles au niveau des régions particulières dénommées synapses (*sunapsis* en grec = point de jonction). Toutes les synapses inter-neuronales (ou neuro-neuroniques) présentent invariablement la même organisation de base :

- Un élément présynaptique appartenant à un premier neurone ;
- Une fente synaptique séparant les deux neurones ;
- Un élément post-synaptique appartenant à un deuxième neurone.

Sur un plan physiologique, on distingue :

1. Des synapses électriques au fonctionnement très simple ;
2. Des synapses chimiques au fonctionnement beaucoup plus complexe.

La transmission de l'information au sein du système nerveux repose sur :

a) L'émission et la propagation de signaux électriques (**potentiels d'action**) sur toute la longueur de l'axone ;
b) Le transfert de ce signal électrique à d'autres cellules par l'intermédiaire de synapses.

Le potentiel d'action (PA) représente le signal électrique fondamental qu'émettent les cellules nerveuses. Il s'agit d'une onde électrique qui s'auto-régénère et se propage depuis son site d'origine : le cône axonique, jusqu'à l'extrémité de l'axone. L'information codée par la fréquence des PA est transmise à la cellule cible par transmission synaptique. Le PA est lié à des flux d'ions extrêmement rapides à travers la membrane cellulaire. Ces signaux électriques reflètent des modifications transitoires de la perméabilité membranaire à certains ions[21].

3. Neurones formels (artificiels)

Précédemment, un aperçu a été donné sur le neurone biologique, sa composition et son fonctionnement. Dans cette section, nous allons faire l'analogie entre le neurone biologique et le neurone formel. La figure II.2 suivante fait la comparaison entre ces deux types de neurone.

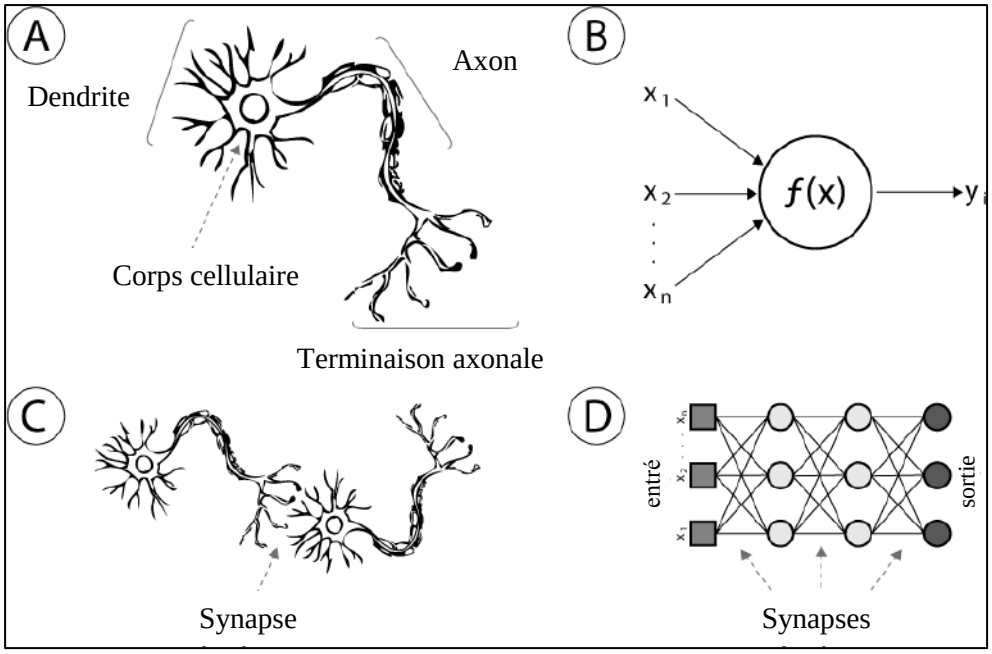


Figure II.2 Neurone biologique et Neurone artificiel[22]

4. Réseaux de neurones artificiels

Les réseaux de neurones artificiels (ANN) sont des paradigmes de traitement de l'information inspirés par le système cérébral humain, avec la motivation de reproduire ses remarquables capacités de cognition humaine pour résoudre des tâches informatiques. Le tout premier réseau neuronal, appelé perceptron, a été inventé par le psychologue Frank ROSENBLATT en 1957. Les réseaux neuronaux étaient alors capables de traiter de nombreux problèmes trop complexes voire impossibles à résoudre par des méthodes de calcul et de statistique standards, cela les a rendus populaire auprès des chercheurs. À la fin des années 1980, de nombreuses institutions ont adopté le perceptron dans diverses applications. La structure des réseaux de neurones stimule le réseau de neurones du cerveau. Biologiquement, un neurone contient un corps cellulaire et est connecté par des dendrites (c'est-à-dire des fils d'entrée) pour recevoir des signaux électriques (c'est-à-dire des pics) d'autres neurones. Ce neurone a un fil de sortie appelé axone pour se connecter aux autres neurones et envoyer des signaux. L'axone a de nombreuses bornes auxquelles sont connectés d'autres neurones à travers des synapses[4].

4.1- Aperçu de l'architecture de l'ANN

La figure II.3 illustre un réseau de neurones à trois couches avec deux couches cachées de cinq neurones chacune et une couche de sortie avec un seul neurone.

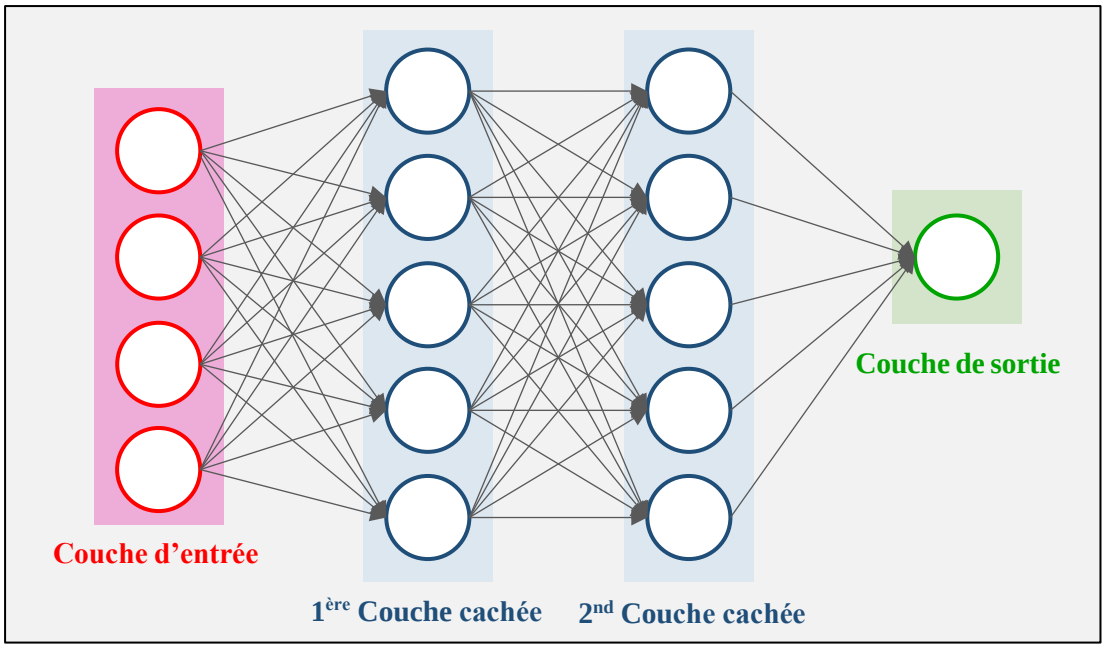


Figure II.3 Réseau de neurone

Dans les réseaux de neurones, chaque neurone correspond à un nœud de calcul qui reçoit des informations (caractéristiques) d'un certain nombre de nœuds d'entrée multipliées par un ensemble de poids. Ces derniers simulent le rôle des synapses qui permettent le transfert de signaux entre les neurones. Ceux-ci effectuent certains calculs non-linéaires, puis envoient des sorties à d'autres nœuds. Les réseaux neuronaux sont généralement constitués de plusieurs couches contenant un ensemble de neurones. Plus il y a de couches, plus les réseaux sont profonds. La dernière couche est généralement la couche de sortie et dépend des problèmes que les réseaux neuronaux tentent de résoudre. Cette couche de sortie peut avoir soit :

- Un nœud de sortie, si le problème est la classification ;
- Plusieurs nœuds de sortie si le problème est la régression[4].

4.2- Exemples des deux réseaux de neurones les plus utilisés

4.2-1 Perceptron mono couche

Avant d'aborder le comportement collectif d'un ensemble de neurones, nous allons présenter le Perceptron (un seul neurone) en phase d'apprentissage. Ce dernier est réalisé avec des poids fixes. Le perceptron de la figure II.4 réalise une simple addition. La somme pondérée des entrées x_i par les poids w_i associés aux entrées est appelée potentiel, noté p.

$$p = \sum_{i=1}^{n} w_i x_i \qquad (2.1)$$

Ce potentiel est alors soumis à une fonction seuil Φ de type Heavyside :

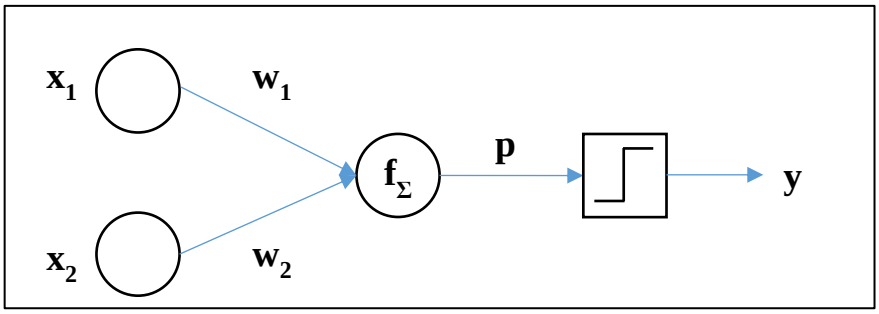


Figure II.4 Perceptron : structure et comportement

La sortie y est donnée par l'équation (2.2) suivante :

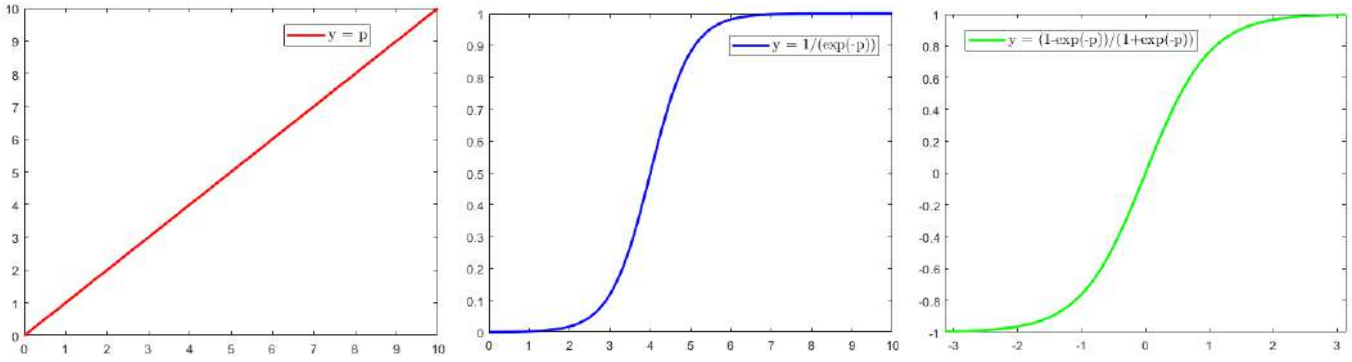
$$y = \begin{cases} 0 \; si \; p < 0 \\ 1 \; si \; p \geq 0 \end{cases} \qquad (2.2)$$

Le problème de ce modèle réside dans la détermination des poids. De ce fait, on a mis au point l'algorithme de retro-propagation du gradient. Cet algorithme consiste à calculer le gradient de la sortie, ensuite la retro-propagation de ce dernier, puis celle des poids. Par conséquent, il faut que la fonction de seuil soit dérivable. La fonction de Heavyside est donc remplacée par des fonctions d'activation qui lui sont similaires mais dérivables. La figure II.5 donne les courbes des principales fonctions d'activation Φ définies par les équations ci-dessous[23-25] :

Linéaire $\qquad y = \Phi(p) = p \qquad (2.3)$

Sigmoïde $\qquad y = \Phi(p) = \dfrac{1}{1 + e^{-cp}} \quad c > 0 \qquad (2.4)$

Tangente hyperbolique $\qquad y = \Phi(p) = \dfrac{1 - e^{-cp}}{1 + e^{-cp}} \qquad (2.5)$



a- Linéaire b- Sigmoïde c- Tangente hyperbolique

Figure II.5 Fonctions d'activation

4.2-2 Perceptron multicouche

Un perceptron multicouche (Multi Layer Perceptron : MLP) est généralement constitué d'une couche d'entrée, une ou plusieurs couches cachées chargées de faire les calculs intermédiaires et possédant une fonction d'activation et une couche de sortie. La couche d'entrée représente toujours une couche virtuelle associée aux entrées du système. Elle ne contient aucun neurone contrairement au reste des couches. La couche de sortie correspond aux sorties du système.

Dans le perceptron multicouche, chaque neurone à l'intérieur d'une couche reçoit des signaux provenant de la couche précédente. Ensuite il effectue un calcul et transmet le résultat à la couche suivante, si elle est présente. Il n'existe pas d'interconnexions entre les neurones situés à l'intérieur d'une même couche. Les activations des différents neurones sont seulement propagées de la couche d'entrée vers la couche de sortie à travers tous les neurones constitutifs du réseau.

Les MLP sont capables de traiter des données qui ne sont pas linéairement séparables. L'avènement des algorithmes de retro-propagation ont fait d'eux le type de réseaux de neurones le plus utilisé. La figure II.6 illustre la structure d'un MLP présentant quatre neurones en entrée, trois neurones sur la couche cachée et deux en sortie[23, 26].

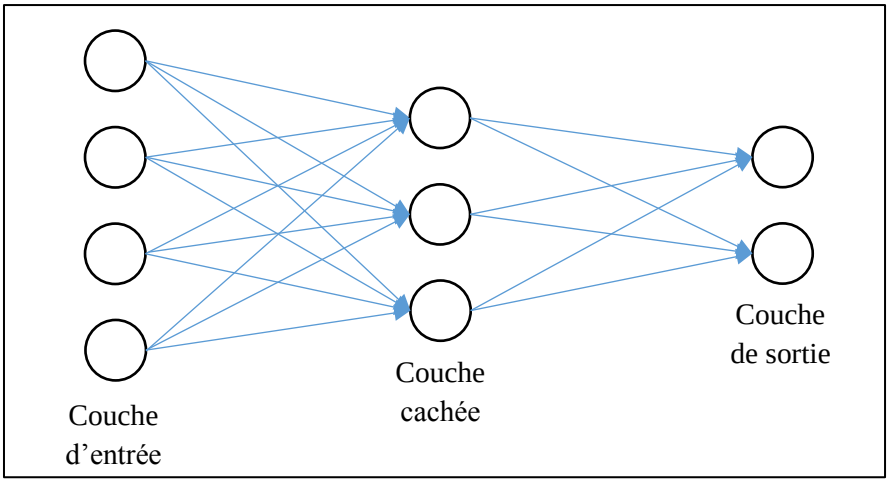


Figure II.6 Modèle du perceptron multicouche

Lorsque tous les neurones d'une couche sont connectés aux neurones de la couche suivante, on parle alors de couches complètement connectées. Cependant, le passage d'une couche à l'autre peut être formalisé sous forme matricielle. Soit un MLP dont le nombre de neurones sur la couche d'entrée est n_0, n_1 sur la couche cachée et n_2 sur la couche de sortie. Les poids de la couche cachée peuvent s'écrire sous la forme d'une matrice $W \in \mathbb{R}^{n_0 \times n_1}$. Pour

un vecteur $X \in \mathbb{R}^{n_0}$ présenté en entrée, le vecteur potentiel V et le vecteur de sortie Y pour la couche cachée s'écrivent comme suit [24]:

$$V = WX \quad (2.6)$$

$$Y = \Phi(V) \quad (2.7)$$

a. Rétro-propagation du gradient pour un MLP

La rétro-propagation du gradient s'applique d'une manière récursive de la couche de sortie à celle de l'entrée. L'initialisation se fait donc par le calcul de l'erreur à la sortie de la dernière couche. Soit l'échantillon d'apprentissage $X \in \mathbb{R}^{n_0}$ et le vecteur de sortie désiré $D \in \mathbb{R}^{n_2}$. L'erreur L sur la couche de sortie est définie par l'équation (2.8) :

$$L = \frac{1}{2}\|Y_2 - D\|^2 \quad (2.8)$$

Où le vecteur Y_2 représente la sortie du MLP (c'est-à-dire la sortie de la couche 2). De manière immédiate, le gradient de l'erreur en sortie peut être calculé par :

$$\partial L / \partial Y_2 = Y_2 - D \quad (2.9)$$

Pour une couche donnée dont X est le vecteur d'entrée, V le potentiel, W la matrice du poids et Y la sortie, la retro-propagation s'écrit donc[24] :

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial V} = \frac{\partial L}{\partial Y}\Phi'(V) \quad et \quad \frac{\partial L}{\partial X} = \frac{\partial L}{\partial V}\frac{\partial V}{\partial X} = \frac{\partial L}{\partial V}\mathrm{W}^T \quad (2.10)$$

b. Mise à jour des poids

Une que fois la retro-propagation du gradient est effectuée pour toutes les couches, les matrices de poids seront mises à jour :

$$W \leftarrow W - \lambda \times d \times W \quad (2.11)$$

Où :

λ - est le taux d'apprentissage ;

d - nombre des entrées [24, 25].

4.3- Aperçu sur l'algorithme du réseau de neurones

La représentation mathématique d'un ou plusieurs neurones est exprimée dans l'ensemble de fonctions suivant :

1. Fonction d'Hypothèse :

$$z = h_{w,b}(x) = w^T x + b \quad (2.12)$$

Avec :

w - la matrice de poids bidimensionnelle du neurone ;

b - la constante de biais du neurone ;

x - la matrice d'entrée bidimensionnelle qui arrive au neurone ;

z - la matrice de sortie bidimensionnelle de la fonction d'hypothèse.

Considèrerons une couche actuelle i . La fonction d'Hypothèse (2.12) simule le flux d'informations provenant des neurones de la couche $i-1$ vers ceux de la couche i. Ce flux d'information est représenté dans le modèle biologique par les signaux électriques circulant d'un ensemble de neurones vers un autre ensemble à travers les synapses. La présence du biais b permet aux neurones de faire un bon apprentissage d'un certain type d'information. Si les réseaux neuronaux mettaient en œuvre un algorithme de normalisation par lots (BN) pour normaliser la sortie z, b deviendrait inutile car il serait annulé.

2. Fonction d'activation :

$$z = \Phi(p) \qquad (2.13)$$

Avec :

Φ - la fonction d'activation ;

z - la matrice de sortie bidimensionnelle de la fonction d'activation.

La sortie z de la fonction d'Hypothèse (2.12) est appliquée par une fonction d'activation $\Phi(p)$ (voir figure II.5-a) pour effectuer une transformation de non-linéarité qui donne des valeurs probabilistes. Il existe d'autres fonctions d'activation qui sont utilisées telles que leaky-ReLU et Softmax.

3. Fonction de perte définie par[27] :

$$j(w,b) = \frac{1}{m}\sum_{i=0}^{m} \mathcal{L}(\hat{y}_i, y_i) \qquad (2.14)$$

Avec :

m - le nombre d'exemples d'apprentissage ;

$\mathcal{L}$ - la fonction de perte appliquée, qui dépend du type de problème à résoudre (régression ou classification) ;

$\hat{y}_i$- la prédiction du réseau de neurones correspondant à la sortie de la couche i ;

y_i - la valeur exacte correspondant à la sortie de la couche i.

La fonction de perte est utilisée pour former les réseaux neuronaux. Chaque couche dans le réseau de neurones effectue des calculs sur la sortie de la couche précédente. Pour la première couche le calcul se fait sur l'entrée. Ceci se fait de gauche à droite. Cette activité est appelée propagation vers l'avant. La prédiction du réseau neuronal (désignée par $\hat{y}$) est la sortie de la

fonction d'activation de tous les neurones de la dernière couche. Lors de l'apprentissage des réseaux neuronaux, la propagation vers l'avant se produit plusieurs fois sur m exemples d'apprentissage. Ce nombre de fois est appelé "époque".

Le calcul des réseaux neuronaux s'effectue sur un ensemble de matrices et de constantes de forme identique. La mise en œuvre de la programmation se fait par la vectorisation pour tirer profit de l'architecture matérielle de manière plus efficace et accélérer la vitesse de calcul. Par conséquent, si la fonction d'hypothèse (équation 2.12) est vectorisée par rapport au nombre de neurones dans la couche i, w deviendra une matrice de poids tridimensionnelle et b deviendra un vecteur de biais. La vectorisation a été déjà intégrée par la plupart des bibliothèques du ML (comme TF) au cœur de leurs API (Application Programming Interface). De plus, des unités de traitement dédiées ont également été créées pour accélérer encore le calcul du ML[4].

5. Réseaux de neurones convolutionnels

Un réseau de neurones convolutionnel (CNN) est une classe des réseaux de neurones artificiels (ANN) qui utilise des couches de convolution afin de filtrer des entrées pour y extraire les informations utiles. L'opération de convolution consiste à combiner des données d'entrées (carte de caractéristiques) avec un noyau de convolution (filtre) pour former une carte de caractéristiques transformée. Les filtres dans les couches convolutionnelles (conv layers) sont modifiés en fonction des paramètres appris afin d'extraire les informations les plus utiles pour une tâche spécifique. Les réseaux convolutionnels s'ajustent automatiquement pour trouver la meilleure caractéristique en fonction de la tâche[28]. Le CNN est capable de filtrer des informations sur la forme et la texture d'un objet lorsqu'il est confronté à une tâche générale de reconnaissance d'objets. Dans cette opération différentes catégories d'objets tendent à avoir des formes différentes. Parfois certains objets différents ont la même forme et sont caractérisés par leur apparence et le CNN est suffisamment puissant pour bien généraliser ces informations. Les applications des réseaux de neurones convolutionnels comprennent divers systèmes de traitement d'images (reconnaissance d'images, classification d'images, étiquetage vidéo, analyse de textes) et de la parole (reconnaissance vocale, traitement du langage naturel, etc…). Les CNN sont aussi appliqués dans des systèmes d'AI tels que les robots, les assistants virtuels et les voitures à conduite autonome[4]. La figure II.7 illustre un réseau de neurone convolutionnel qui fait la reconnaissance du chiffre 5 en manuscrit.

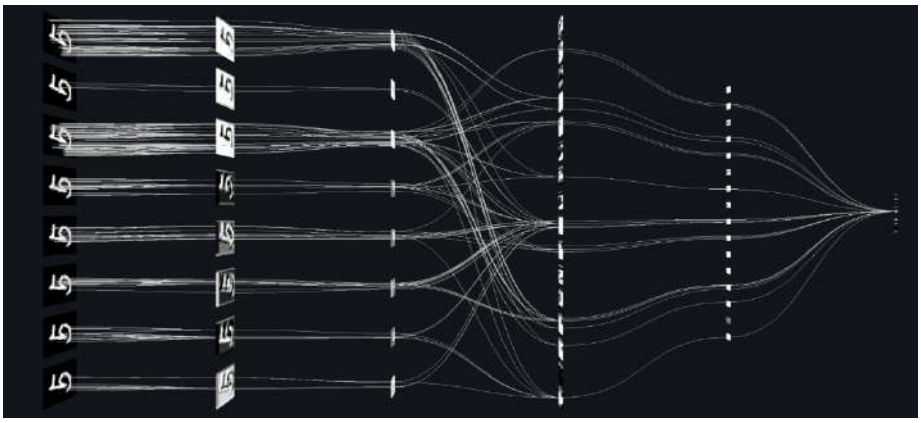


Figure II.7 Réseau de neurones convolutionnel[29]

5.1- Aperçu de l'architecture du CNN

En général, les CNN ressemblent aux ANN car les deux sont basés sur l'idée de couches de neurones artificiels portant des poids et des biais désignés à l'apprentissage. Chaque neurone réalise un produit ponctuel avec quelques entrées et est éventuellement suivi d'une transformation de non-linéarité. L'ensemble du réseau est toujours exprimé par une fonction de score différenciable (fonction de perte).

Dans l'ANN traditionnel, la structure des neurones entièrement connectés ne s'adapte pas bien aux données d'entrée tridimensionnelles telles que les images complètes. De plus, comme il faudrait plusieurs autres couches cachées avec un nombre de neurones plus important, les paramètres totaux s'additionneraient encore plus. Il est clair que cette conception d'architecture conduit à un coût de calcul important. De plus, le nombre énorme de paramètres conduirait rapidement le modèle à un sur-ajustement[4]. Ce dernier peut être réduit par la technique de décrochage (dropout)[30].

L'architecture du CNN est conçue de manière à encoder certaines propriétés spécifiques des images qui rendent le traitement des données multidimensionnelles plus efficace. Ceci améliore considérablement l'efficacité du calcul et réduit le nombre de paramètres dans le réseau. Une architecture CNN en plus de l'entrée et de la sortie, comporte généralement trois couches principales : la couche convolutionnelle, la couche de regroupement et la couche traditionnelle du réseau neuronal (couche entièrement connectée). Spécifiquement, les couches convolutionnelles sont constituées de neurones de volume tridimensionnels appelés "noyaux" (filtres). Chaque filtre possède trois dimensions : largeur, hauteur et profondeur[4].

La figure II.8 donne un exemple d'architecture d'un CNN.

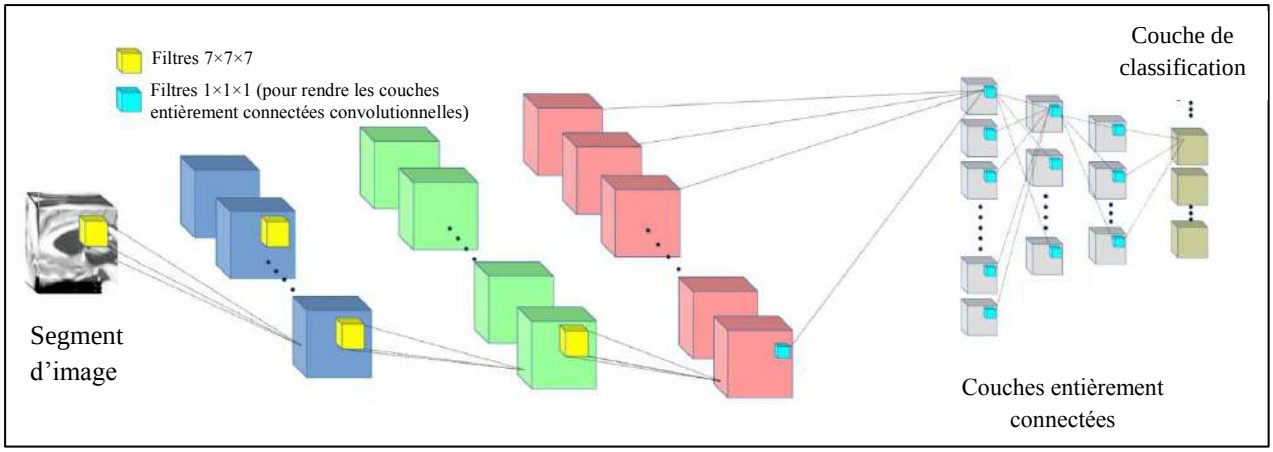


Figure II.8 Architecture d'un CNN[31]

5.2 Couches du CNN

5.2-1 Couche convolutionnelle

Dans cette couche s'effectue la majorité du calcul informatique. Une couche convolutionnelle comprend un ensemble de filtres dont chacun est petit en largeur et en hauteur mais s'étend sur toute la profondeur en fonction du volume d'entrée. Le CNN utilise la stratégie de partage des paramètres qui permet de réduire considérablement leur nombre en adoptant l'hypothèse que si un filtre est utile pour calculer une image à un endroit donné, il sera également utile à d'autres endroits[4].

La figure II.9 donne un aperçu sur l'opération de convolution en deux dimensions.

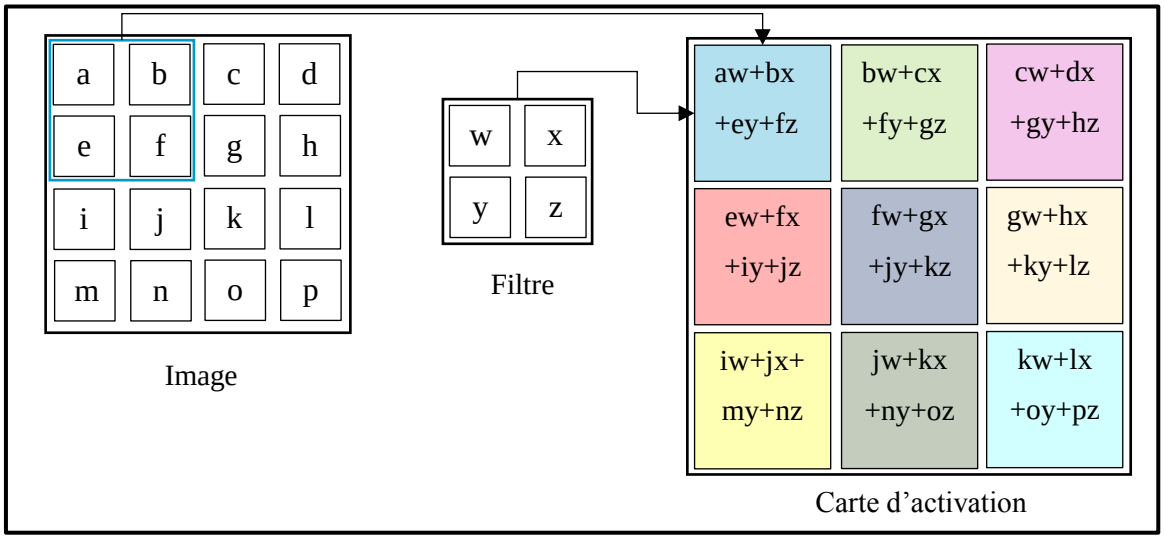


Figure II.9 Opération de convolution en 2D[32]

Au cours de la propagation vers l'avant, le filtre effectue une convolution (glissement) sur un plan de l'image (entrée) de la dimension en largeur à la dimension en hauteur et calculent le produit des points avec les entrées de l'image. Le processus s'effectue sur toute la profondeur de l'image. La sortie finale de cette opération de convolution est un volume tridimensionnel

constitué d'un ensemble de tranches de caractéristiques bidimensionnelles empilées le long de la dimension de profondeur. Chaque tranche contient les réponses d'un filtre à chaque plan du volume de l'image[4]. Ce qui correspond à une carte d'activation (feature map), qui nous indique où se situent les caractéristiques dans l'image. Plus la valeur indiquée dans la carte est élevée, plus il y existence d'une caractéristique de l'image à l'endroit où s'est effectuée la convolution[33]. Comme pour l'ANN, toute la valeur du volume de sortie sera éventuellement normalisée, puis transformée par une fonction de non-linéarité, qui est généralement une fonction ReLU (Rectified Linear Units). Lors de l'opération de retour en arrière (retro-propagation du gradient), chacun des neurones du volume de sortie tridimensionnel calculera le gradient des poids dans ses filtres. Les gradients sont additionnés sur chaque tranche de caractéristiques et sont utilisés pour mettre à jour les poids dans le filtre qui a généré cette tranche de caractéristiques. Les couches convolutionnelles du CNN formées sont un ensemble de filtres appris qui peuvent extraire certains types d'informations telles que :

- Les bords dans différentes orientations ;
- Les taches (gouttes) de couleur dans les couches peu profondes ;
- La forme et la texture générales des objets d'entraînement dans les couches profondes.

Il convient de mentionner qu'une couche convolutive est spécifiée par les hyperparamètres[6] suivants :

- La taille du filtre ;
- Le nombre de filtres ;
- Le pas que les filtres font à chaque balayage ;
- Le type du remplissage à zéro (zero-padding).

Ces paramètres doivent être soigneusement sélectionnés.

Un pas signifie que les filtres vont glisser d'un pixel, et plus le pas est grand, plus la taille de la sortie sera petite. La technique du « Zero-padding » consiste à remplir le volume d'entrée avec une bordure de valeurs zéro supplémentaires. Cette technique est efficace car elle permet aux filtres d'effectuer la convolution plus fréquemment sur les bords de l'entrée et de

[6] Des propriétés qui régissent l'ensemble du processus de formation. Ils comprennent les variables qui déterminent la structure du réseau (par exemple, le nombre d'unités cachées) et les variables qui déterminent la façon dont le réseau est formé (par exemple, le taux d'apprentissage). Les hyperparamètres du modèle sont définis avant l'entraînement (avant l'optimisation des poids et des biais)[34] J. Leonel. (2019). *Hyperparameters in Machine /Deep Learning*. Disponible: https://medium.com/@jorgesleonel/hyperparameters-in-machine-deep-learning-ca69ad10b981

mieux capturer les informations à ces endroits. En outre, cette technique est également pratique car elle permet de contrôler la taille de la sortie. Le Zero-padding peut s'effectuer de deux manières : pas de remplissage ou bien un remplissage par des zéros qui donne une taille de sortie similaire à celle de l'entrée. Comme la disposition des hyperparamètres est mutuellement contraignante, une dimension de la taille de sortie, à l'exception de la dimension de profondeur, peut être formulée comme suit :

Dimension de la sortie : $$\frac{n + 2p - f}{s} + 1 \qquad (2.14)$$

Où :

n - la dimension de l'entrée ;

p - le nombre de zéro-padding à utiliser ;

f - la dimension correspondante du filtre ;

s - le pas du filtre[4].

5.2-2 Couche de « Pooling »

La couche de pooling (couche de sous-échantillonnage) est souvent placée entre deux couches de convolution. Elle reçoit en entrée plusieurs cartes de caractéristiques (feature maps), et applique à chacune d'elles l'opération de pooling. Cette opération (voir figure II.10) consiste à réduire progressivement la taille spatiale de l'entrée (l'image), tout en préservant ses caractéristiques importantes. Cela en utilisant son filtre convolutif pour réduire les divisions de l'entrée à une seule valeur (sous-échantillonnage). Pour cela, on découpe l'image en cellules régulières, puis on garde au sein de chaque cellule la valeur maximale. En pratique, on utilise souvent des cellules carrées de petite taille pour ne pas perdre trop d'informations. Les choix les plus communs sont des cellules adjacentes de taille 2 × 2 pixels qui ne se chevauchent pas, ou des cellules de taille 3 × 3 pixels, distantes les unes des autres d'un pas de 2 pixels (qui se chevauchent donc). On obtient en sortie le même nombre de cartes de caractéristiques qu'en entrée, mais celles-ci sont bien plus petites. La couche de pooling permet de réduire le nombre de paramètres dans le réseau, ce qui réduit la consommation de la mémoire et de calcul. On améliore ainsi l'efficacité du réseau et on évite le sur-apprentissage. La procédure pooling offre également une immuabilité simple pour l'inclinaison, les translations et le cisaillement et améliore donc la faculté de détection des objets d'image de CNN. Les valeurs maximales sont repérées de manière moins exacte dans les cartes de caractéristiques obtenues après pooling que dans celles reçues en entrée – c'est en fait un grand avantage ! En effet, lorsqu'on veut reconnaître un chien par exemple, ses oreilles n'ont pas besoin d'être localisées le plus

précisément possible : savoir qu'elles se situent à peu près à côté de la tête suffit ! Ainsi, la couche de pooling rend le réseau moins sensible à la position des caractéristiques. Le fait qu'une caractéristique se situe un peu plus en haut ou en bas, ou même qu'elle ait une orientation légèrement différente ne devrait pas provoquer un changement radical dans la classification de l'image. Plus la taille du filtre est importante, plus les informations sont concentrées, ce qui conduit à des réseaux plus fins qui s'intègrent plus facilement dans la mémoire du GPU ou du processeur mobile. Cependant, il y a une difficulté à concevoir une taille convenable de filtre (pas trop grade pas trop petite). Une taille trop grande conduit à une suppression d'informations qui conduit à de mauvaises performances de prédiction. En revanche, une taille trop petite ne suffit pas pour extraire les informations nécessaires pour une bonne classification[4, 33].

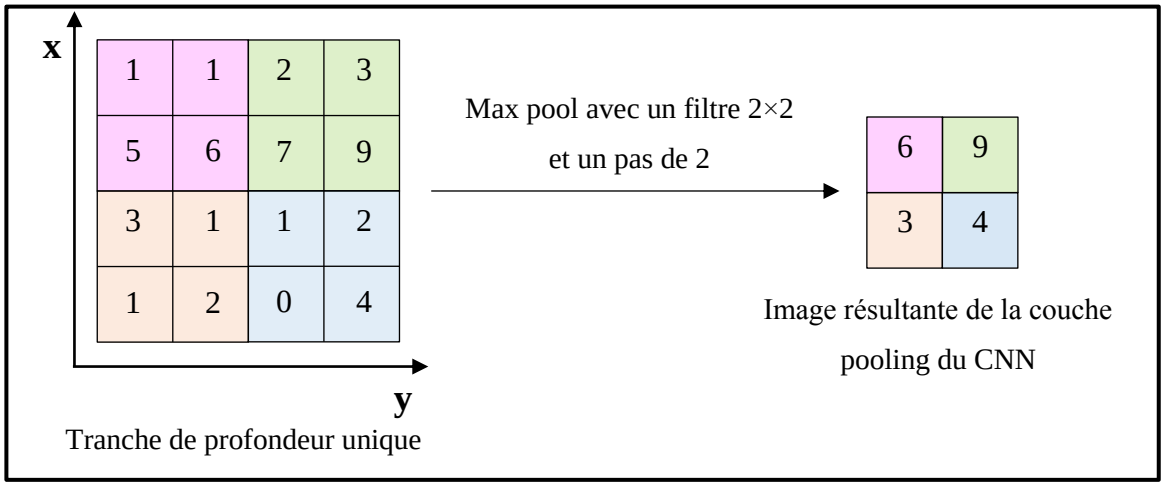


Figure II.10 Opération de max pooling

5.2-3 Couche de démarrage

Dans les couches de démarrage du CNN sont utilisées plusieurs convolutions avec plusieurs filtres pour regrouper les sorties au sein d'une même couche. Par exemple, la figure II.11 illustre l'architecture d'une couche de départ avec l'utilisation de trois opérations convolutionnelles différentes et une opération de regroupement moyenne. La première opération de convolution utilise des filtres 1×1, puis transmet la sortie à deux opérations convolutionnelles avec des filtres 5×5 et 3×3, ensuite s'effectue une opération de mise en commun moyenne. Les sorties de toutes ces couches sont concaténées ensemble le long de la dimension de profondeur et sont transmises à la couche suivante. L'intention de ces couches de démarrage est de permettre au réseau d'apprendre les meilleurs poids pour ces filtres lors de la formation et d'extraire automatiquement des informations plus utiles. Ce qui permet également d'obtenir des couches convolutionnelles plus profondes et plus grandes. Ainsi on obtient de meilleures performances tout en maintenant un coût de calcul raisonnable[4].

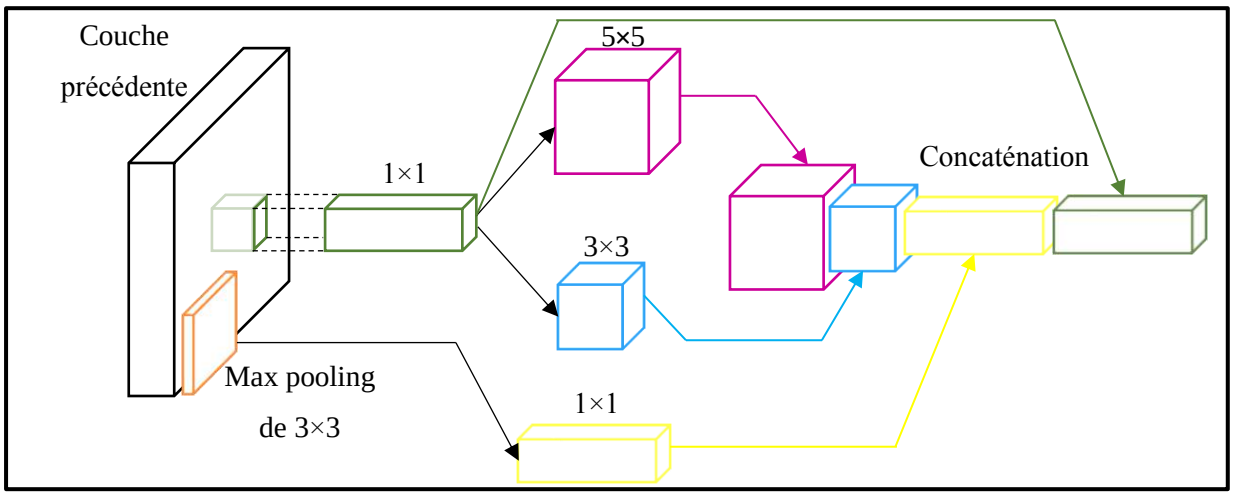


Figure II.11 Couche de démarrage[35]

5.2-4 Couche de correction ReLU

La couche de correction ReLU remplace toutes les valeurs négatives reçues en entrées par des zéros en utilisant la fonction Unité de rectification linéaire (ReLU : Rectified Linear Units) (figure II.12) définie par l'équation (2.14) suivante :

$$ReLU(x) = \max(0, x) \qquad (2.14)$$

Cette fonction joue le rôle de fonction d'activation[33].

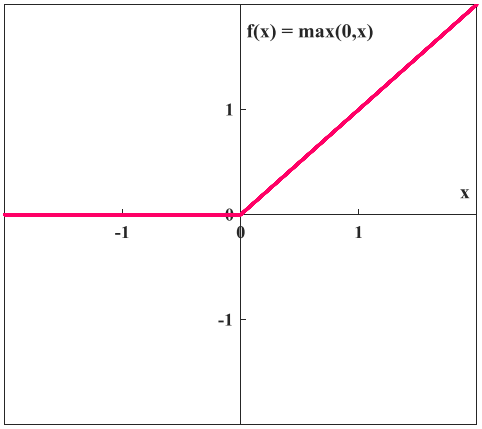


Figure II.12 Allure de la fonction ReLU

5.2-5 Couche entièrement connectée

La couche entièrement connectée (fully-connectecd layer) constitue toujours la dernière couche d'un réseau de neurones, convolutif ou non. Elle n'est donc pas caractéristique d'un CNN. Ce type de couche reçoit un vecteur en entrée et produit un nouveau vecteur en sortie. Pour cela, elle applique une combinaison linéaire puis éventuellement une fonction d'activation aux valeurs reçues en entrée. Cette couche permet de classifier l'image en entrée du réseau. Elle renvoie un vecteur de taille N correspondant au nombre de classes dans notre problème de classification d'images. Chaque élément du vecteur indique la probabilité pour l'image en entrée d'appartenir à une classe. Par exemple, si le problème consiste à distinguer les chats des chiens,

le vecteur final sera de taille 2 : le premier élément donne la probabilité d'appartenir à la classe « chat » et le deuxième donnera la probabilité d'appartenir à de la classe « chien ». Ainsi, le vecteur [0,9 0,1] signifie que l'image a 90% de chances de représenter un chat. Chaque valeur du tableau en entrée "vote" en faveur d'une classe. Les votes n'ont pas tous la même importance. La couche leur accorde des poids qui dépendent de l'élément du tableau et de la classe. Pour calculer les probabilités, la couche fully-connected multiplie donc chaque élément en entrée par un poids, fait la somme, puis applique une fonction d'activation (sigmoïde si N=2, Softmax si N>2). Ce traitement revient à multiplier le vecteur en entrée par la matrice contenant les poids. Le fait que chaque valeur en entrée soit connectée avec toutes les valeurs en sortie explique le terme fully-connected. La couche fully-connected détermine le lien entre la position des caractéristiques dans l'image et une classe. En effet, le tableau en entrée, étant le résultat de la couche précédente, correspond à une carte d'activation pour une caractéristique donnée. Les valeurs élevées dans ce tableau indiquent la localisation (plus ou moins précise selon le pooling) de cette caractéristique dans l'image. Si la localisation d'une caractéristique à un certain endroit de l'image caractérise certaine classe, alors on accorde un poids important à la valeur correspondante dans le tableau[33].

5.2-6 Réseau résiduel

Les chercheures dans le domaine de l'AI ont démontré l'efficacité de l'architecture profonde des réseaux de neurones. Ces derniers peuvent effectivement augmenter en précision en empilant raisonnablement plus de couche. Cependant si le nombre de couches est excessivement grand, les couches deviennent plus profondes et les valeurs du gradient deviennent plus petites et tendent vers zéro. Ce qui diminue la précision des couches et entraîne une dégradation des performances[4]. Kaiming He et ses collègues à « Microsoft Research Asia »[36] furent les premiers à étudier ce problème. Ils proposèrent la solution du réseau résiduel qui a permis la formation de plus de 2000 couches. Le réseau résiduel contient essentiellement des connexions résiduelles. Elles sont destinées à acheminer la sortie de certaines couches précédentes vers la sortie d'une couche ultérieure et à effectuer l'ajout à ces dernières. Ce principe a été étendu à de nombreux autres domaines du DL tels que les systèmes de traitement de la parole[4]. La figure II.13 suivante montre la structure d'un élément de base dans l'apprentissage résiduel :

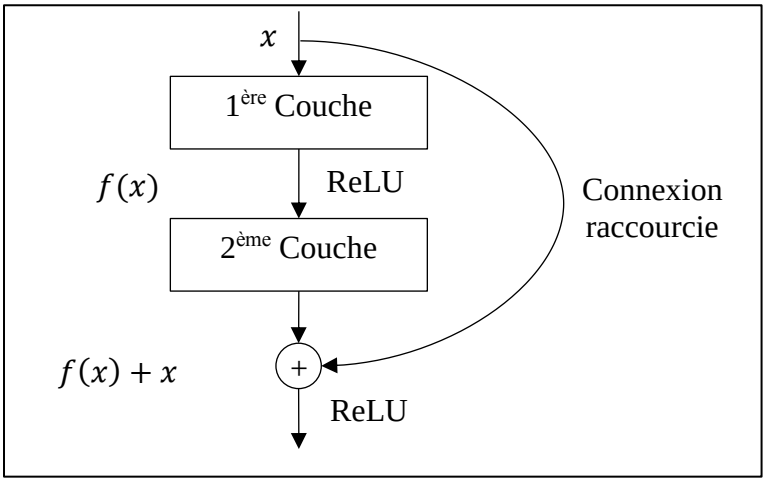


Figure II.13 Connexion résiduelle[36, 37]

Supposons qu'on a un réseau de n couches et qu'on a une configuration résiduelle, on ne fait pas que passer la sortie de la couche 1 à la couche 2 et ainsi de suite, mais on additionnera également les sorties de la couche 1 aux sorties de la couche 2. Si on désigne chaque couche par $f(x)$ la sortie y sera :

- $y = f(x)$ dans un réseau standard ;
- $y = f(x) + x$ dans un réseau résiduel[37].

La formulation de $y = f(x) + x$ peut être réalisée par un réseau de neurones à propagation avant avec des "connexions de raccourci" (figure II.13). Les connexions de raccourci sont celles qui sautent une ou plus de couches. Leurs sorties sont ajoutées aux sorties des couches empilées. Ces connexions n'ajoutent ni paramètre supplémentaire ni complexité de calcul. [36].

5.3- Normalisation par lots

La normalisation par lots (Bach Normalization : BN) fait référence au changement de la distribution des données d'entrée utilisées pour l'algorithme d'apprentissage qui, à son tour, affecte le comportement de l'algorithme ML. Son but est de trouver une solution au problème du décalage de covariance entre deux couches successives. Cette covariance se manifeste lorsque les ensembles de données d'entraînement et de test proviennent de sources différentes et ont des distributions différentes. Ce que fait BN, c'est essayer de limiter le décalage de covariance en normalisant les activations de chaque couche à la moyenne nulle et à la variance unité. Cela permet à chaque couche d'apprendre sur une répartition plus stable des entrées et d'utiliser un taux d'apprentissage plus important, accélérant ainsi la formation du réseau[4]. En pratique, le fait de limiter les activations de chaque couche à une moyenne strictement nulle et à une variance unitaire peut limiter la puissance expressive du réseau [38]. Pour une couche

d'entrée de dimension d, l'activation $x = (x^{(1)} - x^{(d)})$ nous allons normaliser chaque dimension suivant l'équation (2.15) suivante :

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}|x^{(k)}|}{\sqrt{\mathrm{Var}|x^{(k)}|}} \qquad (2.15)$$

On introduit pour chaque activation $x^{(k)}$ une paire de paramètres $\gamma^{(k)}$et $\beta^{(k)}$ qui mettent à l'échelle et font un décalage à la valeur normalisée $\hat{x}_i$ selon l'équation (2.16) ci-dessous :

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)} \qquad (2.16)$$

Ces paramètres sont appris par le réseau. En posant $\gamma^{(k)} = \sqrt{\mathrm{Var}|x^{(k)}|}$ et $\beta^{(k)} = \mathrm{E}|x^{(k)}|$ on retrouve l'activation original. Comme nous avons dit plutôt BN s'intéresse à la covariance. Cependant, ceci n'est pas pratique lorsqu'on utilise l'optimisation stochastique. Donc nous optons pour une simplification qui est l'utilisation des mini lots car ils utilisent les variances par dimension plutôt que par les covariances conjointes. Chaque mini lot produit des estimations de la moyenne et de la variance de chaque activation. Considérons un mini lot $\mathcal{B} = \{x_1 \cdots m\}$. Soit les valeurs normalisées $\hat{x}_1 \cdots m$ et leur transformation linéaires $y_1 \cdots m$. La transformation de la normalisation par lots est définie par l'équation (2.17) suivante :

$$BN_{\gamma,\beta}: x_1 \cdots m \rightarrow y_1 \cdots m \qquad (2.17)$$

Avec :

Entrée : les valeurs de x par un mini-lot $\mathcal{B} = \{x_1 \cdots m\}$;

Paramètres à apprendre : γ et β ;

Sortie : $y_i = BN_{\gamma,\beta}(x_i)$.

Le tableau II.1 suivant résume les transformations utilisées par l'algorithme de normalisation par lots appliquée sure une activation x par un mini-lots[39]:

Tableau II.1 Transformations de la normalisation par lots

Transformation	Formule
1- La moyenne du mini-lot	$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
2- La variance du mini-lot	$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$
3- Normalisation	$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}}$
4- Mise à l'échelle et décalage	$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$

Où ε : Constante ajoutée à la variance du mini-lot pour la stabilité numérique.

6. Conclusion

Dans ce chapitre, nous avons présenté les réseaux de neurones artificiels qui sont basées sur les neurones biologiques et leur fonctionnement. Nous avons ensuite étudié le tout premier type de réseau de neurones qui est le perceptron (monocouches et multicouches). Puis nous avons vu les réseaux de neurones convolutionnels (architecture, algorithme et les différentes couches).

CHAPITRE III : Résultats d'implémentation

1. Introduction

Dans cette partie on va exposer les différents outils de notre implémentation. D'abord, on va donner un aperçu sur les environnements et les bibliothèques utilisés pour la programmation. Ensuite, on présentera le matériel et les logiciels requis dont les logiciels complémentaires et nécessaires pour assurer la compatibilité du programme avec notre accélérateur d'Intel NCS2. Puis on va décrire brièvement l'ensemble de données choisi. On va également expliquer les différentes étapes de notre implémentation : étape d'apprentissage et étape de test et de validation. Enfin, on va exposer les différents programmes ainsi que les résultats de leur exécution et leur interprétation.

2. Environnements et bibliothèques

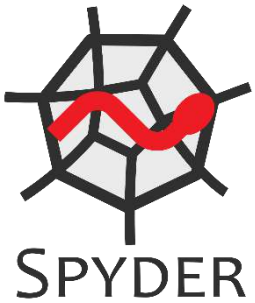
2.1- Anaconda (Python Distribution)



Anaconda est une distribution libre et open source des langages de programmation Python et R. Elle est appliquée au développement d'applications dédiées à la science des données et à l'apprentissage automatique (traitement de données à grande échelle, analyse prédictive, calcul scientifique), qui vise à simplifier la gestion des paquets et de déploiement. Les versions de paquetages sont gérées par le système de gestion de paquets Conda. La distribution Anaconda comprend plus de 250 paquets populaires en science des données adaptés pour Windows, Linux et MacOS. Ces paquets sont automatiquement installés lors de l'installation d'Anaconda, et de nombreux autres peuvent être installés en tapant simplement « conda install [nom du logiciel] » dans l'invite de commande d'Anaconda.

Conda forge : est un effort communautaire qui fournit des paquets Conda pour une large gamme de logiciels. Ces paquets peuvent être trouvés en ligne[40]. Pour les types de technologies Anaconda voire **Annexe A**.

2.2- SPYDER



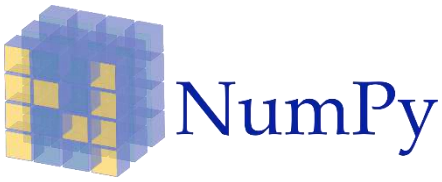
SPYDER (The Scientific Python Development Environment) est un environnement scientifique puissant écrit en Python, pour Python, et conçu par et pour les scientifiques, les ingénieurs et les analystes de données. Il offre une combinaison unique des fonctionnalités avancées d'édition, d'analyse, de débogage et de profilage d'un outil de développement complet avec les capacités d'exploration des données, d'exécution interactive, d'inspection approfondie et de visualisation d'un progiciel scientifique. De plus, Spyder offre une intégration intégrée avec de nombreux progiciels scientifiques populaires, dont NumPy, SciPy, Pandas, IPython, QtConsole, Matplotlib, SymPy, et bien d'autres encore. Au-delà de ses nombreuses fonctionnalités intégrées, ses capacités peuvent être étendues encore plus loin grâce à son système de plugins et à son API[41]. Autres utilisations et caractéristiques de SPYDER se trouvent dans l'**Annexe B**.

2.3- TensorFlow



TensorFlow (TF) est une plate-forme multiplateforme à code source libre pour le ML. Elle contient des bibliothèques et des ressources communautaires qui permettent d'expérimenter rapidement de nouvelles idées, de transformer des concepts en code de modèles de référence et en publication. TF permet aux développeurs de créer et de déployer facilement des applications de ML. À partir de la version 2.0, TensorFlow est devenu capable de construire facilement des modèles d'apprentissage automatique à plusieurs niveaux d'abstraction avec une grande vitesse et de bonnes performances. Pour commencer, TensorFlow fournit les API Keras séquentielles de haut niveau qui présentent les avantages suivants : convivialité, modularité, compossibilité et extensibilité[42]. Pour plus de détails consulter l'**Annexe C**

2.4- NumPy



NumPy est la bibliothèque mathématique de calcul scientifique pour le langage de programmation Python. Elle prend en charge les objets de tableaux multidimensionnels (matrices) avec une large collection de fonctions d'opérations sophistiquées (diffusion). NumPy peut être utilisé de manière pratique comme un gestionnaire multidimensionnel efficace de données génériques et définir de nouveaux types de données arbitraires. Cela permet une intégration rapide et transparente avec une grande variété de frameworks et de bases de données populaires[4]. Les tâches effectuées par NumPy se trouvent dans l'**Annexe D**.

2.5- OpenCV



OpenCV (Open Source Computer Vision Library) est une bibliothèque de logiciels de vision par ordinateur et de ML à code source ouvert. OpenCV a été construite pour fournir une infrastructure commune pour les applications de vision par ordinateur et pour accélérer l'utilisation de la perception des machines dans les produits commerciaux. Étant un produit sous licence BSD[7], OpenCV permet aux entreprises d'utiliser et de modifier facilement le code. La bibliothèque contient plus de 2500 algorithmes optimisés parmi lesquels se trouve un ensemble complet d'algorithmes classiques de vision par ordinateur et de ML. Ces algorithmes peuvent être utilisés dans plusieurs applications[43] (voir **Annexe E**).

[7] La licence BDS (Berkeley Software Distribution License) est une licence libre utilisée pour la distribution des logiciels. Elle permet de réutiliser soit l'intégralité ou bien une partie du logiciel sans restriction, qu'il soit intégré dans un logiciel libre ou propriétaire.

3. Le langage Python

Python est un langage de programmation interprété, orienté objet, de haut niveau et doté d'une sémantique dynamique. Ce langage est développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles[44]. Ses structures de données intégrées de haut niveau, combinées à un typage et une liaison dynamiques le rendent très avantageux pour le développement rapide d'applications. La syntaxe de Python, simple et facile à apprendre, facilite la lisibilité et réduit le coût de la maintenance du programme. Python prend en charge des modules et des paquets qui encouragent la modularité des programmes et la réutilisation du code[45]. Les caractéristiques de ce langage ce trouvent dans l'**Annexe F**.

4. Matériel et Logiciels

4.1- Neural Compute Stick 2 d'Intel

La figure III.1 représente NTSC 2 qui est un kit de développement "Plug and Play" pour l'inférence AI.



Figure III.1 Neural Compute Stick 2[46]

Il est maintenant plus facile que jamais de mettre la vision par ordinateur et l'intelligence artificielle au service d'Internet des Objets (Internet of Things : IoT) et des prototypes de périphériques grâce à ses capacités améliorées. Elle offre des prototypes plus intelligents pour le développement des appareils intelligents telles que :

- Une caméra intelligente ;
- Un drone doté de capacités de reconnaissance de gestes ;
- Un robot industriel ;
- Un dispositif intelligent à domicile.

Ce qui ressemble à une clé USB standard cache beaucoup plus à l'intérieur. Elle est basée sur le tout dernier VPU Intel® Movidius™ Myriad™ X qui comprend le moteur de calcul neuronal, un accélérateur matériel dédié pour les inférences de réseaux neuronaux profonds. Avec plus de noyaux de calcul que la version originale (NCS) et l'accès à la Distribution d'OpenVINO™ toolkit d'Intel®. NCS2 offre des performances 8 fois supérieures à celles de la génération précédente. Elle regroupe plusieurs caractéristiques[47](voir **Annexe G**).

4.2- Distribution Intel® d'OpenVINO™ toolkit

Une demande fréquente pour le NCS 2 d'Intel® est qu'il peut se connecter à n'importe quel ordinateur mono-carte (Single Board Computer : SBC) disponibles sur le marché. La distribution Intel® d'OpenVINO™ toolkit fournit une installation binaire qui prend en charge un grand nombre de développement et de plates-formes en fournissant des outils et une API pour l'Intel® NCS 2. Mais de nombreux utilisateurs qui réalisent des prototypes avec le NCS 2 d'Intel® veulent choisir leur plateforme SBC préférée afin de se rapprocher le plus possible de leur configuration finale du système. Pour répondre aux besoins de ces développeurs, Intel a ouvert l'API du moteur d'inférence OpenVINO™ toolkit et le plug-in VPU Intel® Movidius™ Myriad™[46].

4.2-1 Inférence du Deep Learning

OpenVINO™ toolkit permet aux développeurs de déployer des modèles d'apprentissage profond performants grâce à une API de haut niveau du moteur d'inférence C++ ou Python* intégrée à la logique applicative. L'inférence du DL prend en charge plusieurs plateformes Intel® et est inclus dans la distribution Intel® d'OpenVINO™. Les opérations dans OpenVINO™ toolkit se déroulent comme le montre la figure III.2.

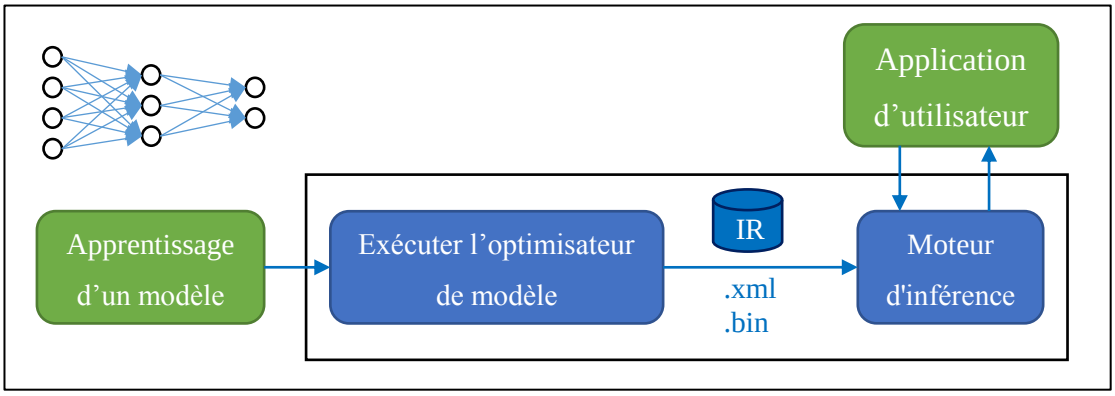


Figure III.2 Diagramme de déroulement des opérations d'OpenVINO toolkit

La distribution Intel® d'OpenVINO™ comprend les deux éléments suivants[48] :

a. L'optimisateur de modèle

C'est un outil de ligne de commande basé sur Python. Il importe des modèles formés à partir de frameworks d'apprentissage profond populaires tels que Caffe*, TensorFlow*, Apache MXNet*, et Open Neural Network Exchange (ONNX*).

Les tâches de l'optimisateur de modèle sont :

- Fonctionne sur plusieurs systèmes d'exploitation : Windows*, Linux*, et macOS* ;
- Effectue des analyses et des ajustements pour une exécution optimale sur les dispositifs cibles en utilisant des modèles statiques et entraînés ;
- Sérialise et ajuster le modèle dans un format de représentation intermédiaire (IR) d'Intel ;
- Prend en charge de plus de 100 modèles publics pour Caffe, TensorFlow, MXNet et ONNX.

Les frameworks standards ne sont pas nécessaires pour générer des fichiers IR pour les modèles constitués de couches standards. Lors du traitement de couches personnalisées dans des modèles originaux, l'Optimisateur de modèle fournit un mécanisme flexible d'extensions[48].

b. Le moteur d'inférence

Ce moteur utilise une API commune pour fournir des solutions d'inférence sur la plateforme de votre choix : CPU, GPU, VPU ou FPGA.

Les tâches du moteur d'inférence sont :

- Exécuter différentes couches sur différentes cibles (par exemple, un GPU et des couches sélectionnées sur un CPU) ;
- Implémenter des couches personnalisées sur un CPU tout en exécutant la topologie restante sur un GPU - sans avoir à réécrire les couches personnalisées ;
- Optimiser les charges de travail (analyse des graphiques de calcul, ordonnancement et compression des modèles) pour le matériel cible grâce à une solution de notation intégrée et conviviale ;
- Exploiter la nouvelle exécution asynchrone pour améliorer les performances de la fréquence d'images tout en limitant les cycles gaspillés ;
- Utiliser une API pratique en C, C++ ou Python pour travailler sur les fichiers IR et optimiser l'inférence[48].

4.3- Raspberry Pi 3 model B+

Le Raspberry Pi est un nano ordinateur monocarte à processeur ARM[8]. Ce système compact se présente en plusieurs modèles de différente tailles et formes et qui sont amélioré en puissance de calcul[4]. Le Raspberry Pi 3 model B+ (voir figure III.3) est le tout dernier produit de la gamme Raspberry Pi 3. Il est doté d'un processeur Quad-core de 64 bits fonctionnant à 1,4 GHz. Ceci lui donne un avantage au niveau de la connectivité réseau. En effet Cette puce combine :

1. Un réseau local sans fil (WLAN) bi-bande à 2,4 GHz et 5 GHz ;
2. Bluetooth 4.2/Bluetooth à basse énergie (BLE) ;
3. Gigabit Ethernet[49].

Les caractéristiques techniques de la carte Le Raspberry Pi 3 model B+ se trouvent l'**Annexe H**.



Figure III.3 Raspberry Pi 3 model B+ [50]

5. Étapes d'installation d'OpenVINO[51]

5.1- Environnement de travail

Pour pouvoir installer OpenVINO™ toolkit sur windows, il est nécessaire que notre ordinateur ait les caractéristiques adéquates et les logiciels périphériques requis

5.1-1 Édition Windows

Windows 10 professionnel

[8] Processeurs ARM : (Acorn Risc Machine) développés par la société Britannique Acorn Compouters. ARM [Ltd] ne fabrique pas et ne vend pas ses processeurs sous forme de circuits intégrés, elle les invente et vend leurs licences à d'autre fabricants.

5.1-2 Système (facultatif)

- Processeur : Intel® Core™ i3-6006U CPU @ 2,00GHz 2,00GHz
- Mémoire installée (RAM) : 4,00 Go (3,87 Go utilisable)
- Type du système : Système d'exploitation 64 bit, processeur ×64

5.1-3 Logicielles périphériques et requis

- Microsoft Visual Studio avec C++ 2019
- CMake 3.4 64-bit
- Python 3.6.5 64-bit

5.2- Mettre à jours les variables d'environnements Windows

1. Dans la fenêtre de recherche Windows tapez **modifier les variables d'environnement système** cliquez sue **Entrée** une fenêtre semblable à celle-ci apparaitra :

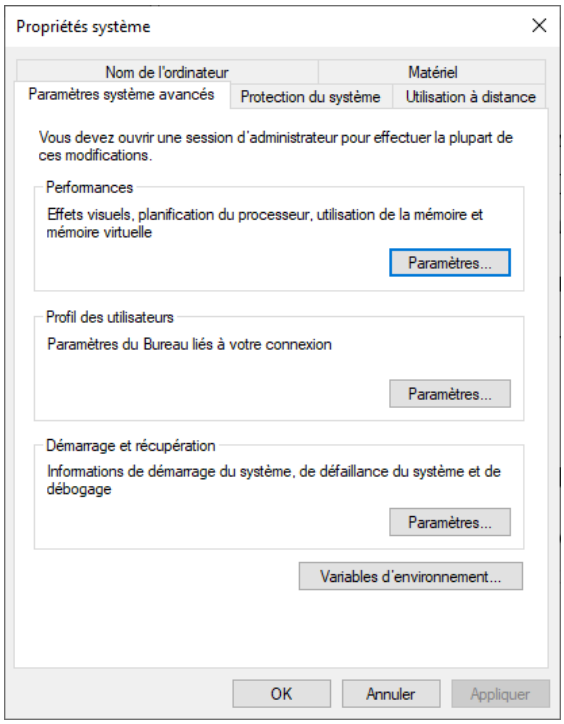


Figure III.4 Fenêtre «Propriétés système »

2. Cliquez sur **Variables d'environnement**
3. Dans **Variables système**, cliquez sur **Path** puis **Modifier**

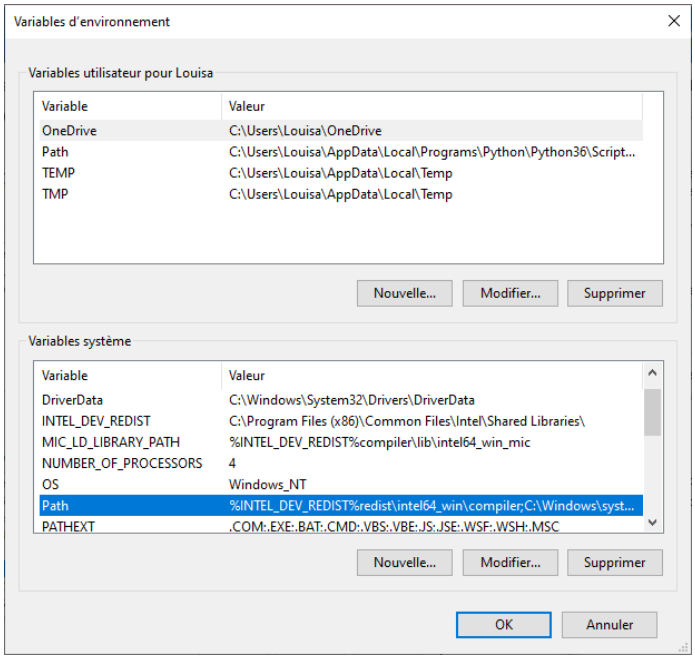


Figure III.5 Fenêtre « Variables d'environnement »

4. Dans la fenêtre ouverte, cliquez sur **Parcourir**, une autre fenêtre s'ouvre :

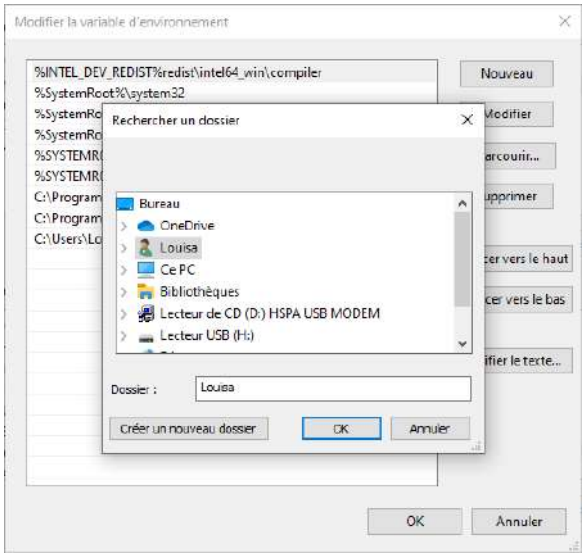


Figure III. 6 Fenêtre « modifier la variable d'environnement »

5. Si vous avez besoin d'ajouter CMake au PATH, accédez au répertoire ou c'est installé CMake, le répertoire par défaut est : C:\Program Files\CMake.

6. Si vous avez besoin d'ajouter Python au PATH, accédez au répertoire ou c'est installé, le répertoire par défaut est : C:\Users\Louisa\AppData\Local\Programs\Python\Python36

7. Cliquez sur OK à plusieurs reprises pour fermer chaque fenêtre.

5.3- Installer OpenVINO™

Installer la distribution Intel® des composants principaux d'OpenVINO ™toolkit

1. Si vous n'avez pas téléchargé la distribution Intel® d'OpenVINO ™ toolkit, télécharger la dernière version. Par défaut, le fichier est sauvegardé dans le répertoire « Téléchargements » autant que : « w_openvino_toolkit_p_2019.3.379 »
2. Allez à Téléchargements
3. Faites double clic sur « w_openvino_toolkit_p_2019.3.379 ». Une fenêtre s'ouvre pour vous permettre de choisir votre répertoire d'installation et vos composants. Le répertoire par défaut est : C:\Program Files (x86)\IntelSWTools\openvino_2019.3.379, C:\Program Files (x86)\IntelSWTools\openvino, pour plus de simplicité, un raccourci vers la dernière installation est également créé : C:\Program Files (x86)\IntelSWTools\openvino. Si vous choisissez un répertoire d'installation différent, le programme d'installation créera le répertoire pour vous :

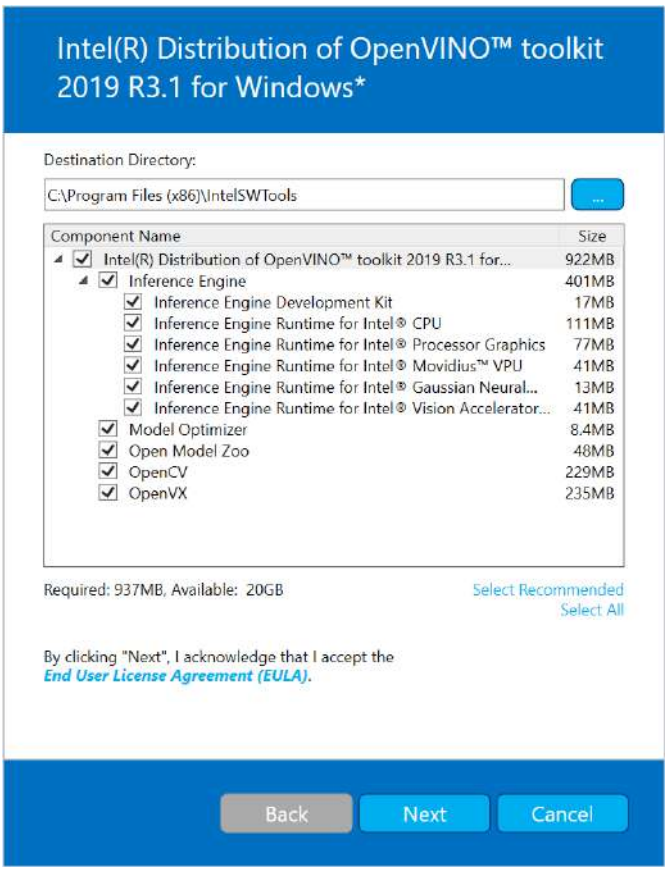


Figure III. 7 Choix du répertoire d'installation et des composants

8. Cliquez sur **Next**
9. On vous demande si vous souhaitez donner votre consentement pour collecter des informations. Choisissez l'option de votre choix. Cliquez sur **Next**.

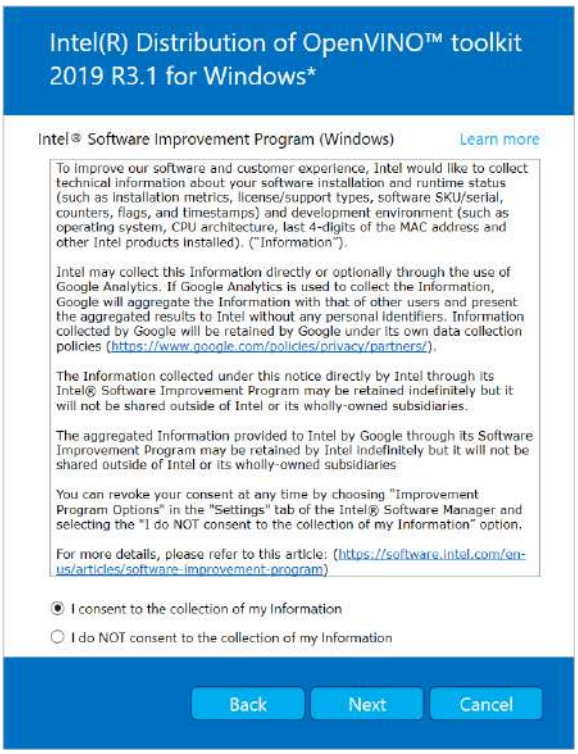


Figure III.8 Consentement pour collecter des informations

10. Si vous installez les logiciels de l'étape **2.1-3 (Logicielles périphériques et requis)**, aucun avertissement n'apparaitra. Sinon, la fenêtre suivante apparaitra et vous dévierez faire l'étape **2.1-3** :

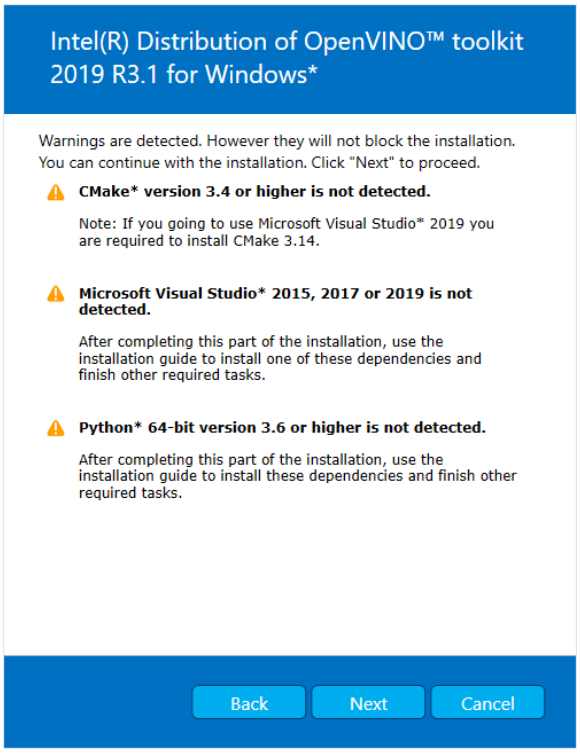


Figure III.9 Avertissements détectés

11. Cliquez sur **Next**.

12. Lorsque la première partie de l'installation est terminée, la fenêtre finale vous informe que les composants principaux ont été installés et que des étapes supplémentaires sont encore nécessaires :

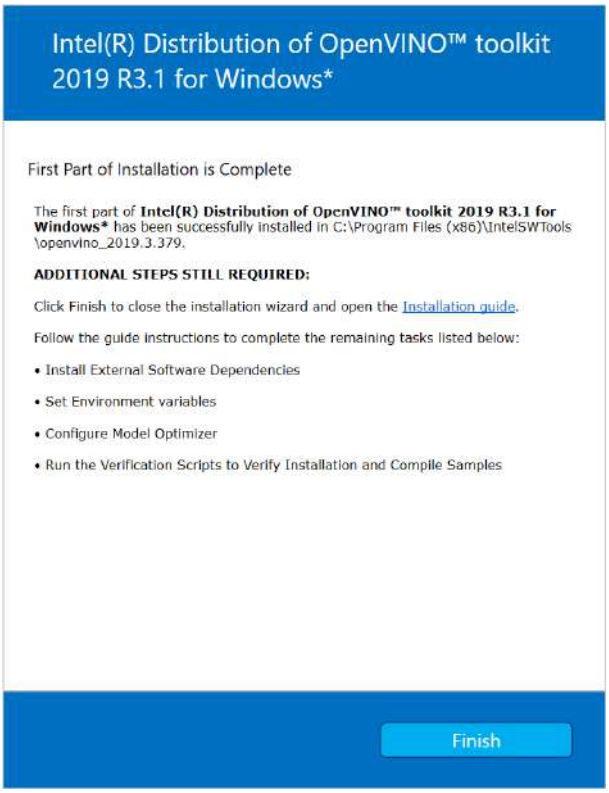


Figure III.10 Fin de la première partie de l'installation

5.4- Définir les variables d'environnement

Vous devriez mettre à jours plusieurs variables d'environnement avant que vous puissiez compiler et exécuter les applications d'OpenVINO™. Ouvrez l'invite de commandes en l'exécuter autant qu'administrateur. Puis exécutez ces deux instructions :

```
1- cd C:\Program Files (x86)\IntelSWTools\openvino\bin\
2- setupvars.bat
```



Figure III.11 Configuration des variables d'environnement d'OpenVINO

5.5- Configurer l'optimisateur de modèle (Model Optimizer)

Vous pouvez configurer l'optimisateur de Modèle soit pour tous les frameworks pris en charge à la fois, soit pour un seul framework à la fois :

Option 1 : Configurez l'Optimisateur de modèle pour tous les frameworks pris en charge en même temps :

1. Ouvrez l'invite de commandes en l'exécutant autant qu'administrateur
2. Allez dans le répertoire des prérequis du Model Optimizer avec l'instruction :

```
1. cd C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\model_optimizer\install_prerequisites
```

3. Exécutez le fichier batch suivant pour configurer l'optimisateur de modèle pour Caffe*, TensorFlow*, MXNet*, Kaldi* et ONNX* :

```
install_prerequisites.bat
```

Option 2 : Configurez l'optimisateur de modèle pour chaque framework séparément :

1. Répétez les étapes 1 et 2 de l'option 1 ;

2. Exécutez le fichier batch pour le framework que vous utiliserez avec le Model Optimizer. Vous pouvez en utiliser plusieurs :

- Pour Caffe :

  ```
  install_prerequisites_caffe.bat
  ```

- Pour TensorFlow :

  ```
  install_prerequisites_tf.bat
  ```

- Pour MXNet :

  ```
  install_prerequisites_mxnet.bat
  ```

- Pour Kaldi :

  ```
  install_prerequisites_kaldi.bat
  ```

- Pour ONNX :

  ```
  install_prerequisites_onnx.bat
  ```

La fenêtre suivante apparaitra pour indiquer que la configuration de l'optimisateur de modèle est réussie :

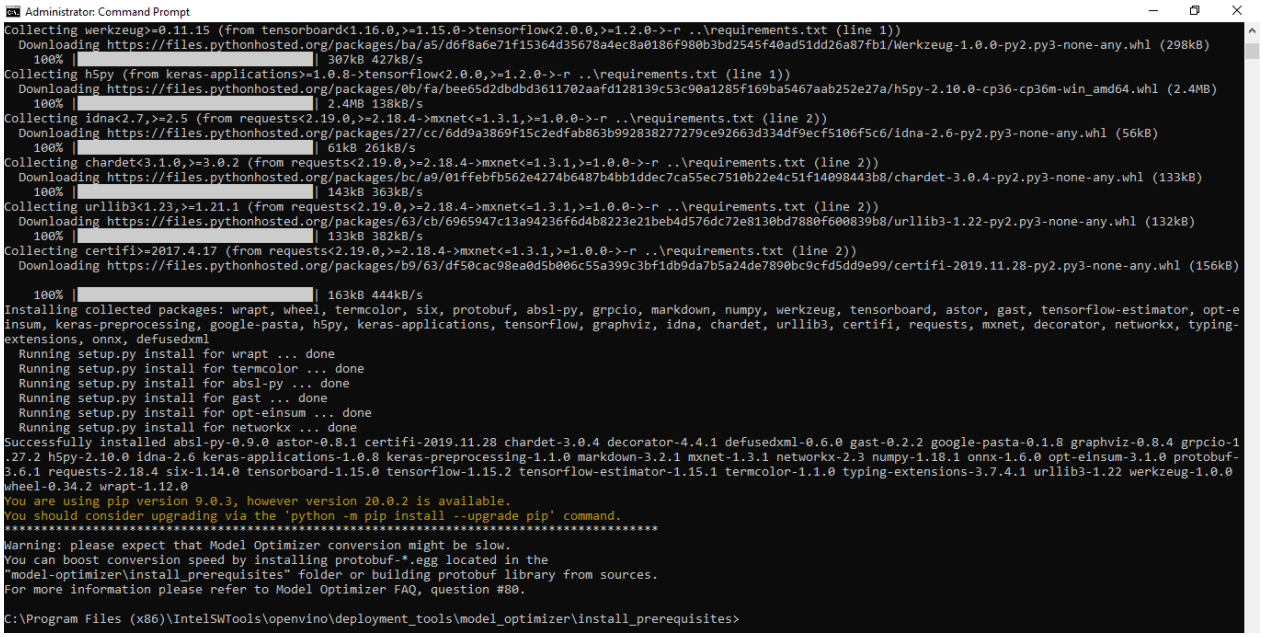


Figure III.12 Configuration de l'optimisateur de modèle

5.6- Utilisation des scripts de vérification pour vérifier l'installation

Pour vérifier l'installation et compiler deux échantillons, exécutez les applications de vérification fournies avec le produit sur l'unité centrale (CPU) :

1. Ouvrir l'invite de commande en l'exécutant autant qu'administrateur
2. Aller au répertoire de la démo du moteur d'inférence :

```
1. cd C:\Program Files (x86)\IntelSWTools\openvino\deployment_tools\demo\
```

3. Exécuter les scripts de vérification en suivant les instructions de la section suivante.

5.6-1 Exécution du script de vérification de la classification des images

Pour exécuter le script, lancez l'instruction: `demo_squeezenet_download_convert_run.bat`

Lorsque le script de vérification sera terminé, une fenêtre contenant les noms et les certifications des top 10 des catégories :

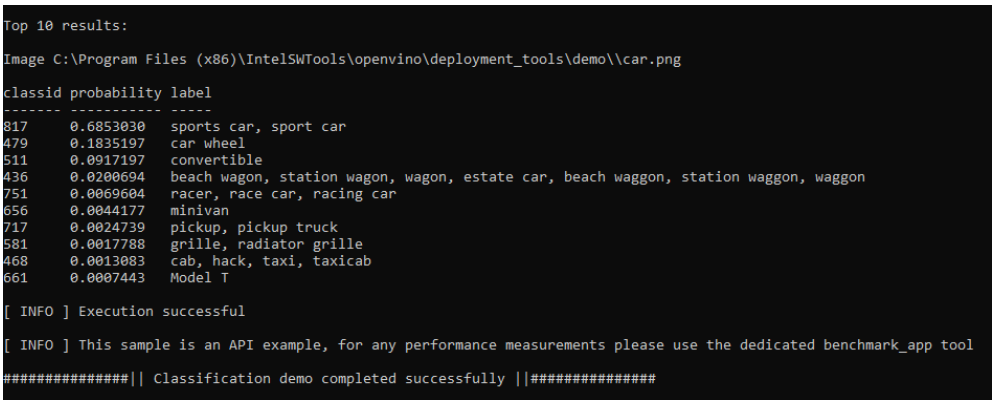


Figure III.13 Script de vérification de la classification des images

Cette démo est terminée. Laissez la console ouverte et passez à la section suivante pour lancer la démo « Inference Pipeline ».

5.6-2 Exécution du script de vérification du pipeline d'inférence

Pour exécuter le script, lancez l'instruction : `demo_security_barrier_camera.bat` tout en restant dans la même console.

Lorsque la démo est terminée, deux fenêtres s'ouvrent :

1. Une fenêtre de console qui affiche des informations sur les tâches effectuées par la démo ;
2. Une fenêtre de visualisation d'image qui affiche un cadre résultant avec des détections rendues sous forme de boîtes englobantes, comme ci-dessous :



Figure III.14 Fenêtre de visualisation d'image

6. Ensemble de données

L'ensemble de données utilisée dans notre programme est composé de 10160 images de 128×128 pixels en noir et blanc. Ces images sont des chiffres de 0 à 9 de différentes écritures organisées dans 10 classes. Chaque classe contient 1016 images. La figure III.15 suivante représente quelques exemples de chaque classe :

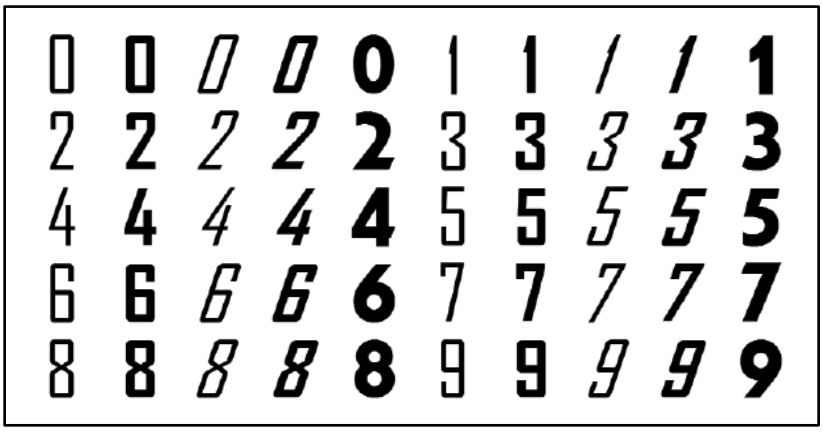


Figure III.15 Exemples de chiffres pour chaque classe

7. Implémentation

7.1- Phase d'apprentissage

Nous allons créer un CNN pour faire la classification des chiffres de 0 à 9 :

1. Importation l'ensemble de données : images représentants les chiffres organisés dans des dossiers de 0 à 9 ;
2. Lire le dossier de chaque classe qui contient 1016 images chacun ;
3. Mettre toutes ces images dans une seule liste et créer une liste correspondante qui contient toutes les étiquettes des images ;
4. Diviser les données en trois catégories : Apprentissage, Test et Validation
5. Tracer la distribution des images d'apprentissage pour vérifier que les données de chaque classe sont distribuées d'une manière quasi équiprobable (voir figure III.18) ;
6. Les images seront traitées, restructurées et optimisées pour le processus d'apprentissage ensuite, elles seront augmentées pour généraliser les données, puis on va utiliser des fonctions telles que la rotation, la translation, le zoom et plus ;
7. On va représenter chaque matrice sous forme de vecteur, c'est ce que l'on appelle le « One Hot encoding » ;
8. Créer le modèle est commencer le processus d'apprentissage ;
9. Une fois le processus terminé, on va tracer les résultats (figure III.19) ;
10. Sauvegarder le modèle d'apprentissage et l'importer dans un script de test.

La figure III.16 ci-dessous représente le diagramme de déroulement de la phase « apprentissage » de notre programme et la figure III.17 illustre l'architecture de son CNN.

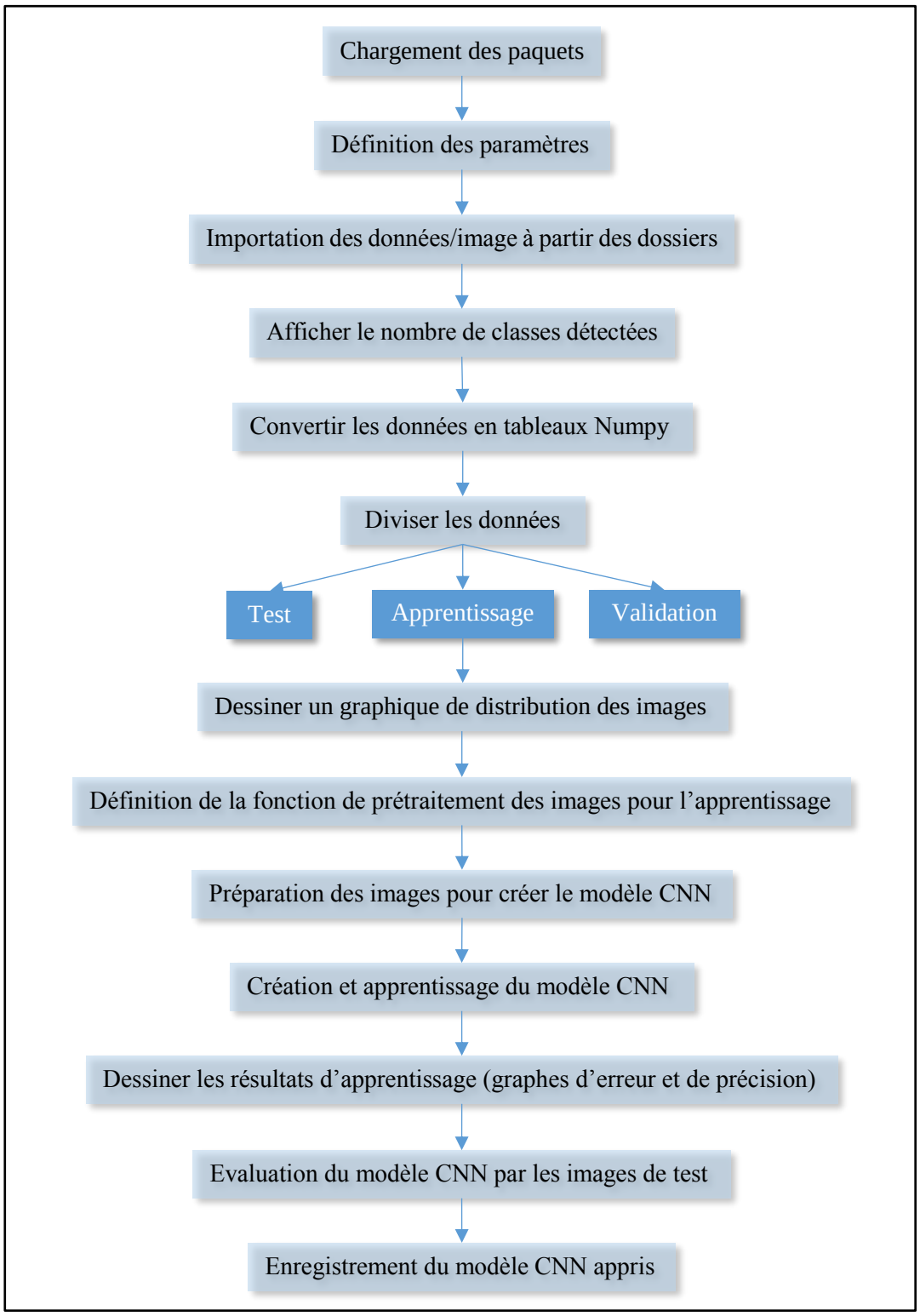


Figure III.16 Digramme de la phase d'apprentissage

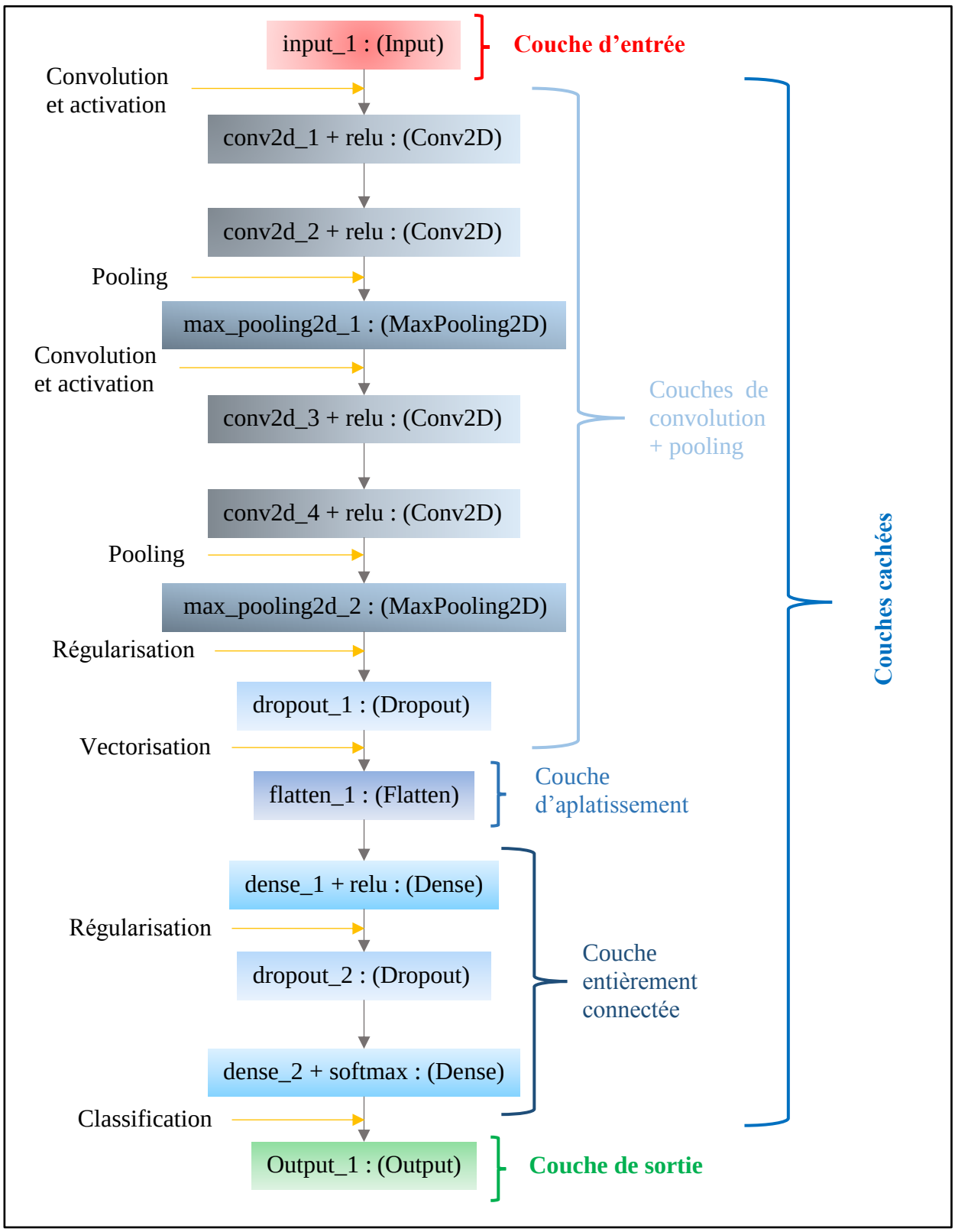


Figure III.17 Architecture du CNN

7.1-1 Résultats et interprétations

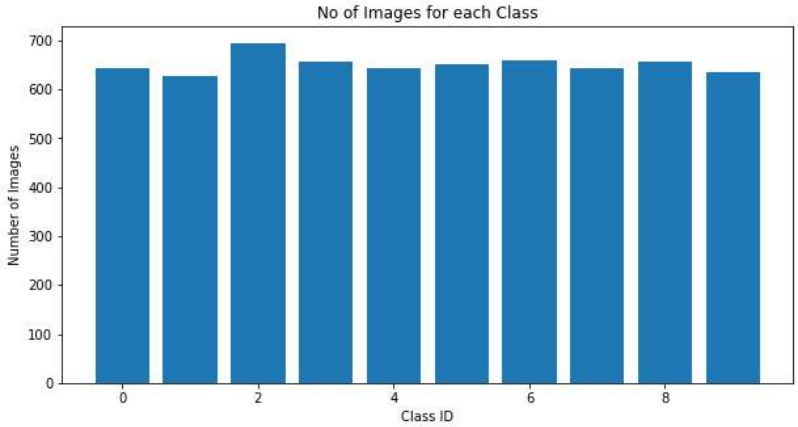
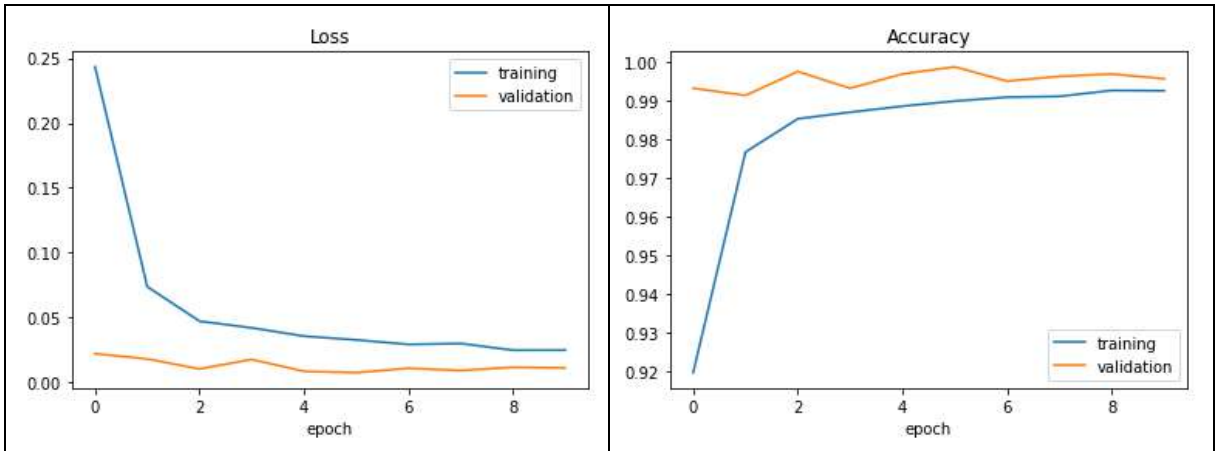


Figure III.18 Distribution des images d'apprentissage

D'après le graphique à colonnes de la figure III.18 ci-dessus on remarque qu'il y a environ 600 images d'apprentissage dans chaque classe.



a) Graphe d'erreur | b) Graphe de précision

Figure III.19 Résultats d'apprentissage

Dans les deux graphes de la figure III.19, on représente la variation des données d'apprentissage par rapports aux données de validation en fonction des époques. Ceci, le long du processus d'apprentissage. Pour le graphe d'erreur, on remarque qu'au début de l'apprentissage, l'erreur pour les données d'apprentissage atteint sa valeur maximale de 0,25. A partir de l'époque 2, les valeurs d'erreur pour les données d'apprentissage et celle de validation se rapprochent. A la fin de l'apprentissage, l'erreur de validation arrivent à 0,0114 et celle d'apprentissage 0,0252. Dans le graphe de précision, la valeur de celle-ci pour les données d'apprentissage est à sa valeur minimale de 92% pour arriver aux environs de 99% à partir de l'époque 2. Notre CNN atteint une précision de 99,45% en fin d'apprentissage. Cette précision

est aussi due à l'ensemble de données utilisé. En effet, avec l'ensemble de données « Fashion-MNIST » la précision ne dépasse pas 89,1% (figure III.20).

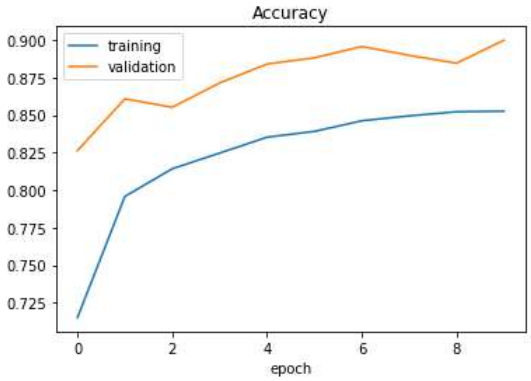


Figure III.20 Graphe de précision pour un CNN avec « Fashion-MNIST »

7.2- Phase de test et de validation

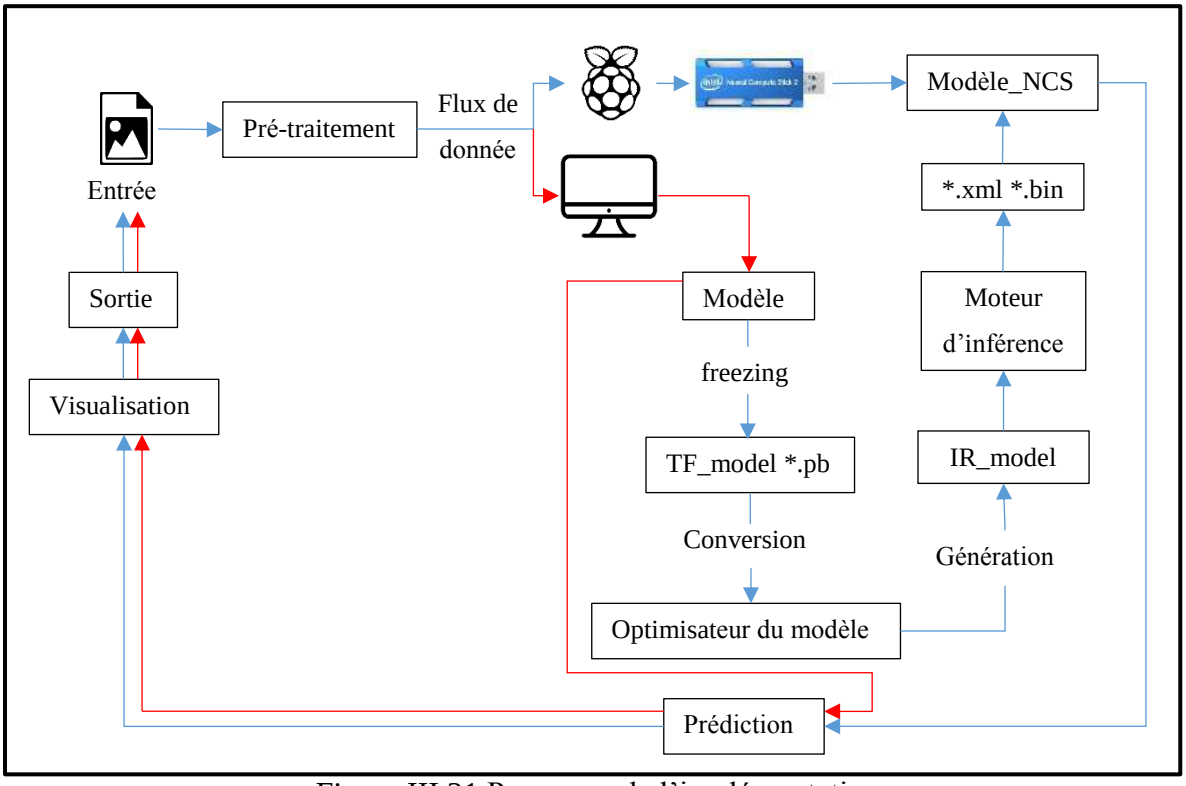


Figure III.21 Processus de l'implémentation

La figure III.21 ci-dessus montre un schéma descriptif du déroulement de la phase « test » de notre travail. Ce dernier est divisé en deux parties :

a) Optimisation avec l'ordinateur : Après la phase d'apprentissage de notre modèle, nous allons le tester.

1. Lire le modèle et injecter une image d'entrer ;
2. Faire un prétraitement a cette dernière ;
3. Faire la prédiction à partir de notre modèle ;

4. Visualiser l'image de sortie qui va contenir l'indice de la classe prédite ainsi que la probabilité d'appartenance à celle-ci.

A la fin, notre modèle va subir un « freezing » pour l'utiliser dans la seconde partie.

b) Implémentation sur RPI 3 B+ avec NSC2 : le modèle de la première partie est nécessaire pour générer le modèle IR

1. Utiliser l'optimisateur du modèle d'OpenVINO avec le « frozen model » pour générer le graphe du modèle IR ;
2. Faire la prédiction à partir de ce dernier ;
3. Visualiser l'image de sortie ;
4. Comparer les deux parties en termes de performance (tableau III.1).

7.2-1 TF_model et IR_model

Pour pouvoir passer de l'étape d'optimisation sur ordinateur vers l'étape d'implémentation sur plateforme embarquée, il est nécessaire de faire une conversion du programme pour en extraire les deux modèles : TensorFlow (TF) et Représentation Intermédiaire (IR)

a. TF_model : Le modèle TensorFolw peut être obtenu à partir d'un modèle sauvegardé d'extension «. H5[9] » ou «.p ». Dans notre cas on a utilisé une fonction « freeze_session » sur le modèle keras avec l'extension « .H5 » afin de le convertir en un modèle TF. L'organigramme de la figure III.22 explique de déroulement du programme (**Annexe I**) utilisé pour la conversion.

b. IR_model : On utilise le script « mo.py » du répertoire <INSTALL_DIR>/deployment _tools/model_optimizer pour exécuter l'optimisateur du modèle et convertir le modèle en représentation intermédiaire (IR). La façon le plus simple est d'exécuter mo.py en précisant le chemin du modèle d'entrée : `python mo.py --data_type FP16 --framework tf --input_model -input_dir/TF_model/model_tf.pb --model_name IR_model --output_dir/IR_model/ --input_shape [1,32,32,1] --input conv2d_1_input --output dense_2/Softmax`. Le script mo.py est le point d'entrée universel qui permet de déduire le framework qui a produit le modèle d'entrée par une extension standard du fichier de modèle (dans notre cas « .pb » puisqu'il s'agit d'un modèle TensorFlow). Deux fichiers

[9] Le type de fichier H5 est principalement associé à HDF5 par le Centre National des Applications Super-informatique. HDF5 est une bibliothèque à usage général (general purpose) et un format de fichier pour stocker des données scientifiques[52] FILExt. (2020). *All about H5 Files*. Disponible: https://filext.com/file-extension/H5.

seront extraits de cette conversion dont les extensions sont « .xml » et « .bin ». La figure III.23 représente le processus de génération du modèle IR pour l'extraction des fichiers « .xml » et « .bin » à partir du moteur d'inférence [53].

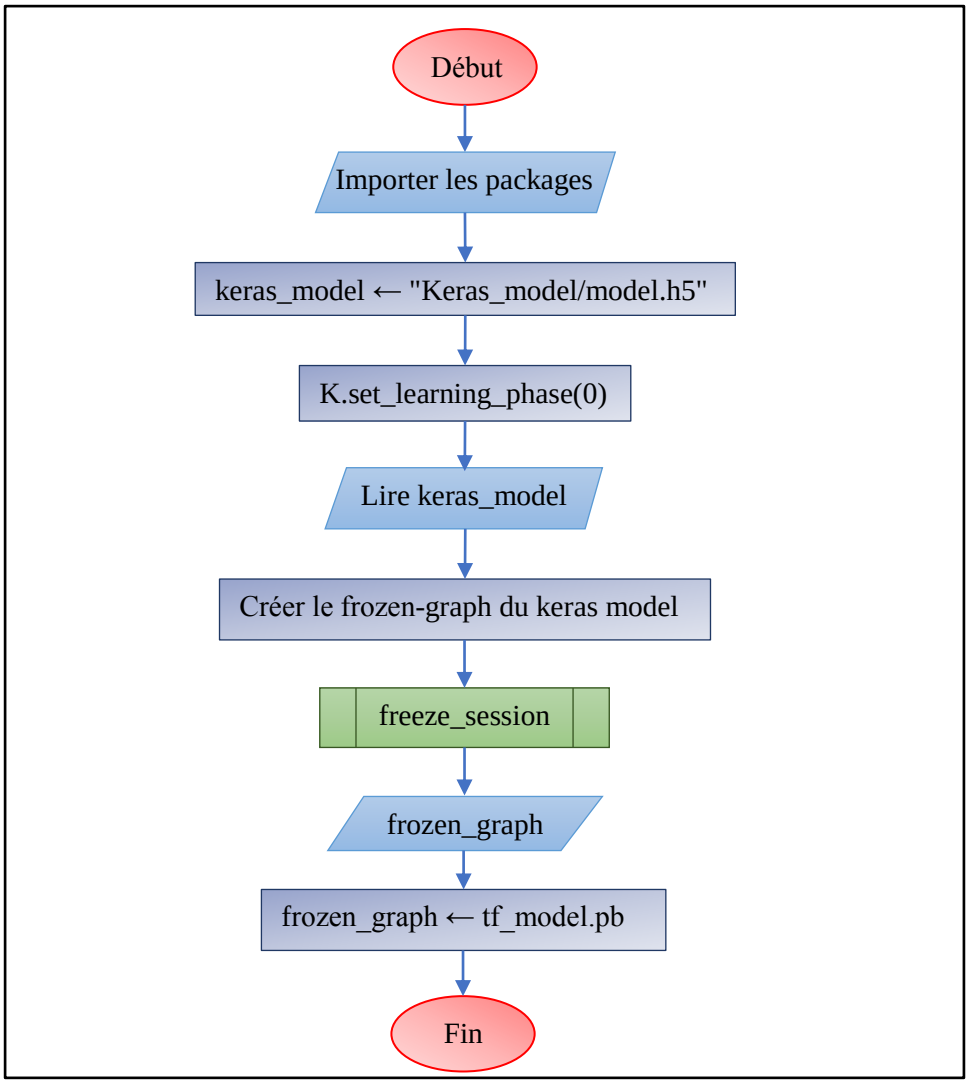


Figure III.22 Organigramme de conversion en modèle TF

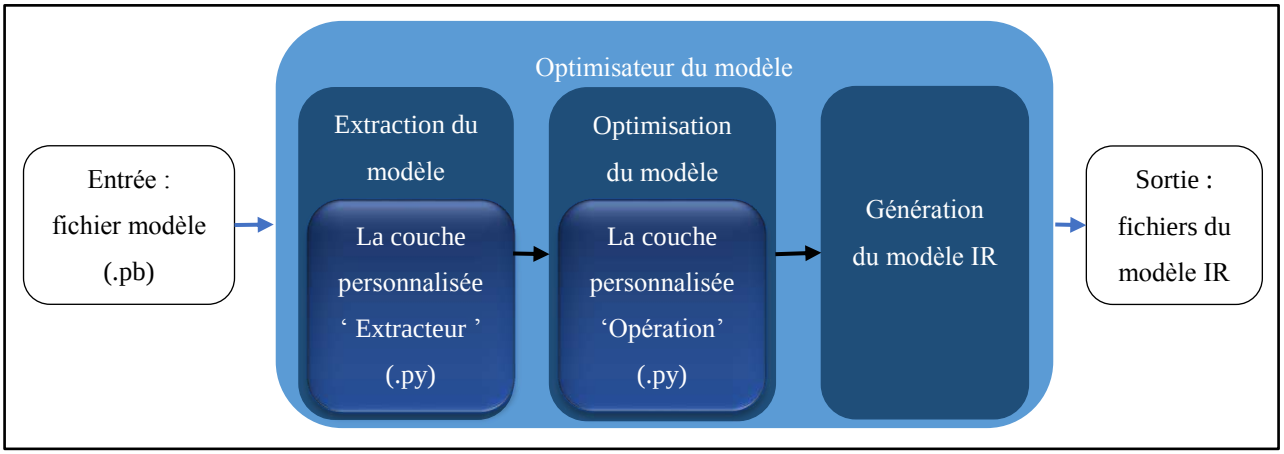


Figure III.23 Couches personnalisées de l'optimisateur du modèle

L'optimiseur de modèle extrait d'abord les informations du modèle d'entrée (la topologie des couches, les paramètres, le format des entrées et des sorties, …). Le modèle est ensuite optimisé à partir des diverses caractéristiques connues des couches, des interconnexions et du flux de données qui proviennent, par partie, de la couche « opération ». Celle-ci fournira des détails, qui incluent la forme de la sortie pour chaque couche. Enfin, le modèle optimisé génère les fichiers IR du modèle nécessaires au moteur d'inférence pour faire fonctionner le modèle.

La figure III.24 suivante montre le fonctionnement du moteur d'inférence avec les plugins du CPU et du GPU en mettant en évidence les extensions des couches personnalisées pour chaque appareil (CPU ou GPU). Chaque plugin comprend une bibliothèque d'implémentations optimisées pour exécuter des opérations de couche connues. Celles-ci doivent être étendues pour exécuter une couche personnalisée. L'extension de la couche personnalisée est implémentée en fonction de l'appareil cible [53].

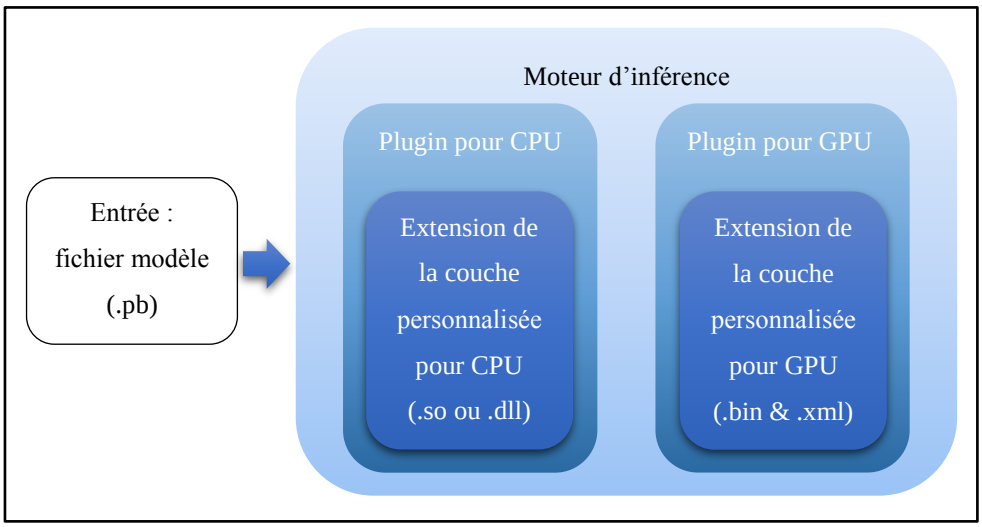


Figure III.24 Couches personnalisées du moteur d'inférence

7.2-2 Organigrammes

Afin de tester notre plateforme, on exécute deux programmes :

- Le premier est pour tester le modèle CNN appris (organigramme de la figure III.25) ;
- Le second est pour faire la prédiction des entrées selon notre CNN (organigramme de la figure III.26).

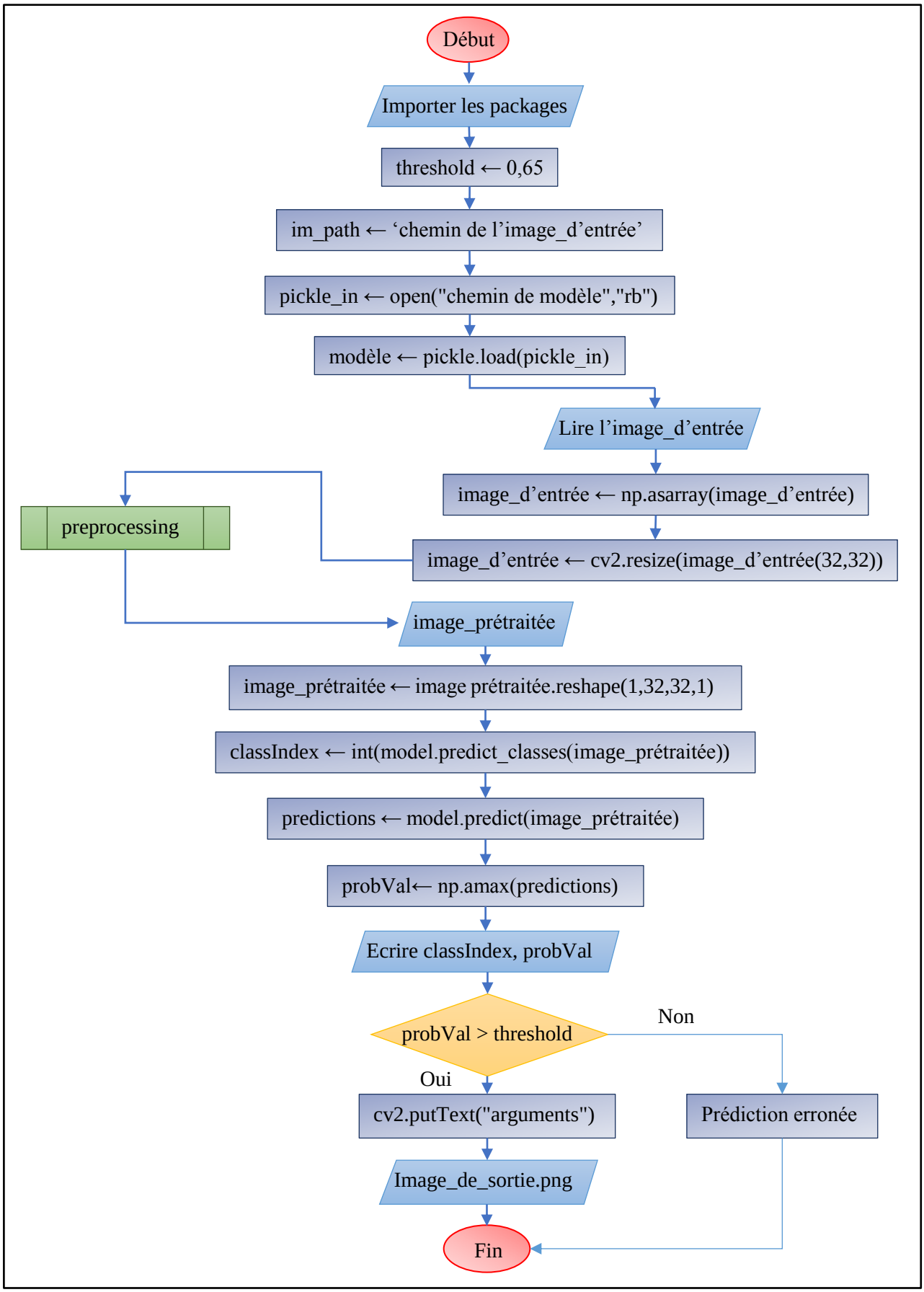


Figure III.25 Organigramme du programme test

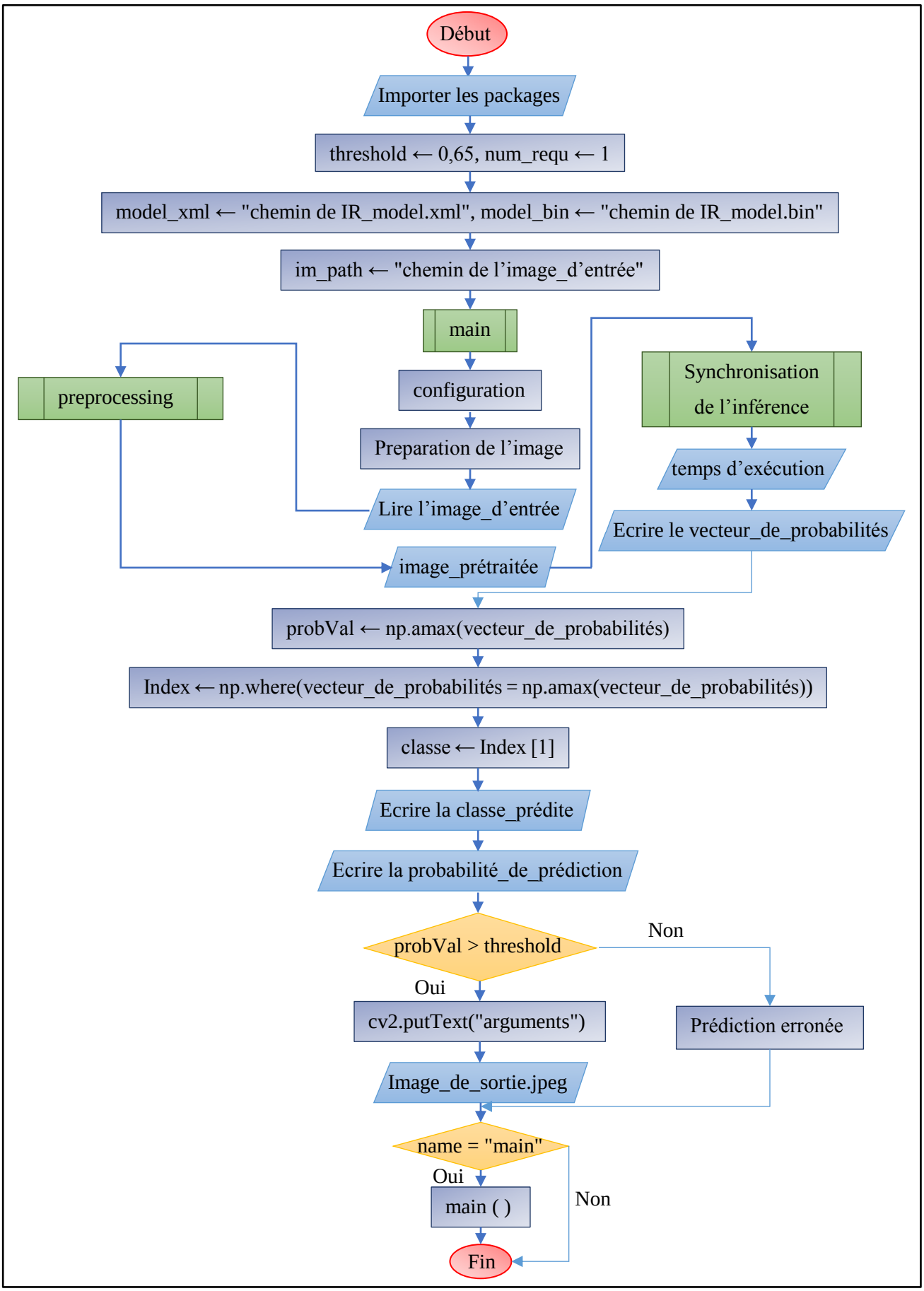


Figure III.26 Organigramme de la phase prédiction

7.2-3 Résultats et interprétations

Le tableau III.1 suivant compare le temps d'exécution pour prédire la classe d'un chiffre et la probabilité de précision de cette prédiction :

Tableau III.1 Résumé des résultats des performances

Hardware		Caractéristiques techniques						Temps d'exécution
		Fréquence		N° de cores		Mémoire		
Pc portable		2 GHz		2		32 GO		Environ 200 ms
NCS2	**RPI3 B+**	700MHz	1,4 GHz	16	4	4 GO	1 GO	Environ 3 ms

On remarque que les performances du Raspberry, accélérées à l'aide du NSC2, sont environ 70 fois plus rapide que ceux de notre ordinateur.

La figure III.27 montre le matériel qui constitue notre système NSC2 + RPI 3 B+. La fenêtre d'exécution du programme test sur notre plateforme RPI 3 B+ avec la clé NSC2 est dans la figure III.28.

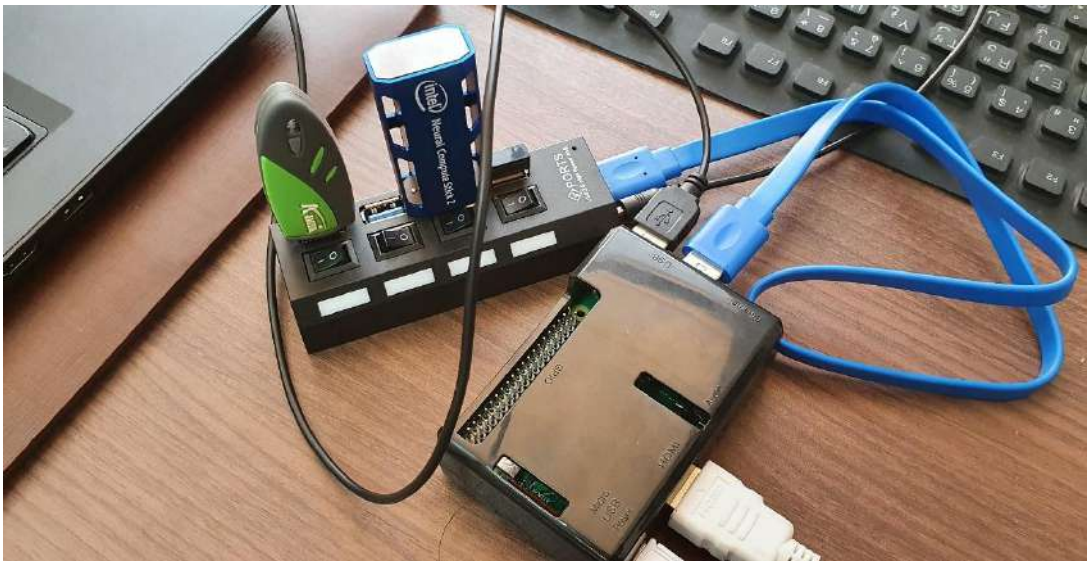


Figure III.27 Matériel utilisé dans le projet

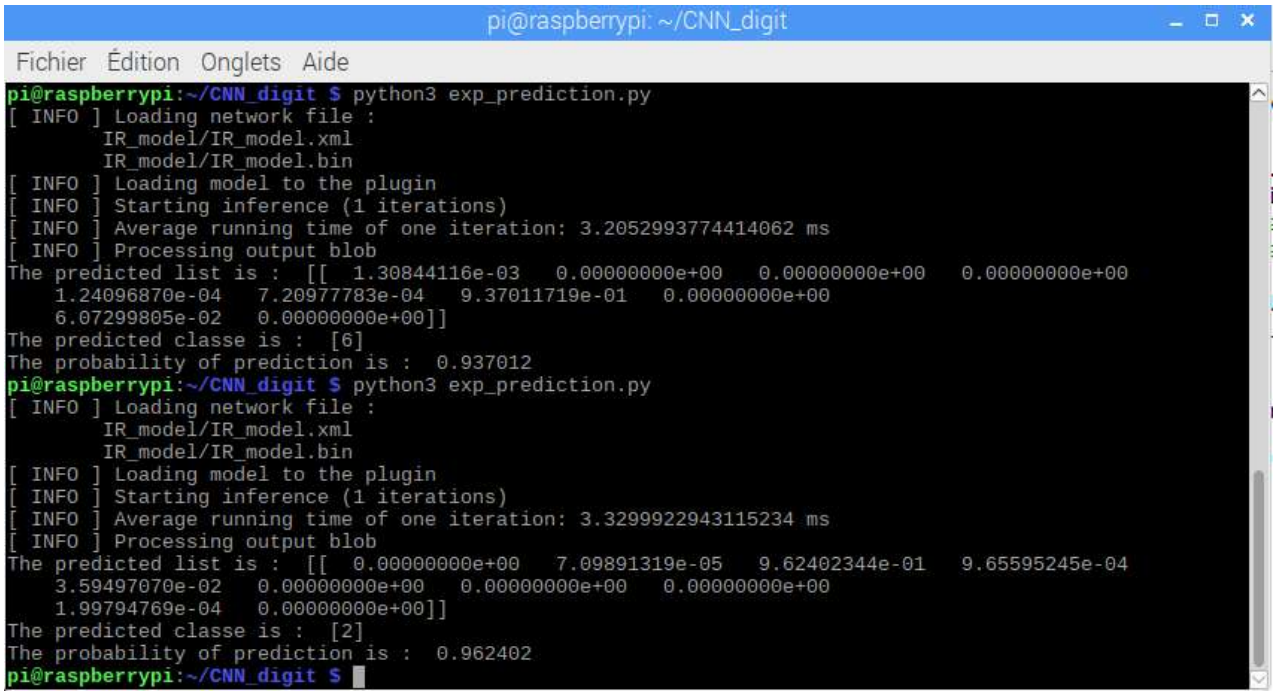


Figure III.28 Exécution du programme de prédiction avec RPI3 B+ et NSC2

Pour la première image entrée, on remarque que la valeur la plus élevée dans est le vecteur de probabilité est $9{,}37 \times 10^{-1}$ qui est dans la 7ème position du vecteur. Cette position est l'équivalente à la classe du chiffre 6. C'est la classe prédite par notre système. Pour la deuxième entrée, la classe prédite est celle du chiffre 2. Les résultats de l'exécution sont dans la figure III.29 qui suit :

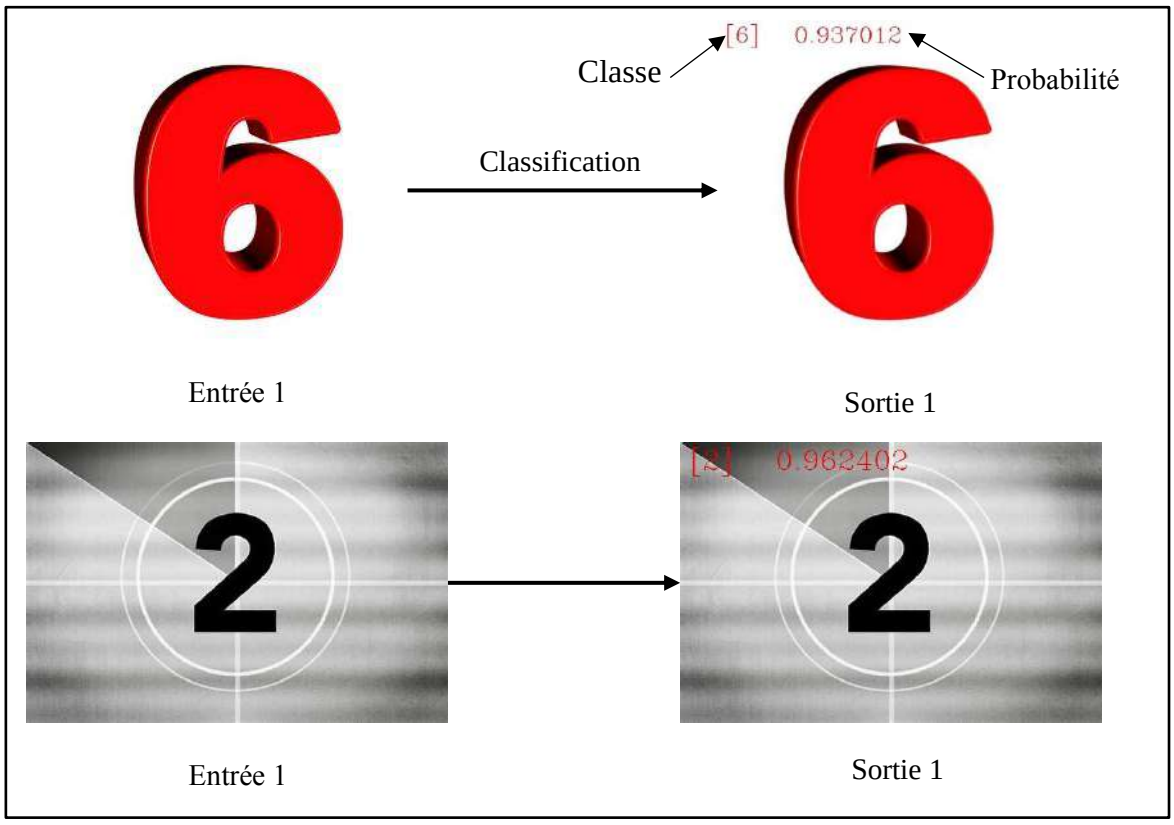


Figure III.29 Classification avec RPI 3 B+ et NSC2

8. Conclusion

Dans ce dernier chapitre, nous avons cité quelques bibliothèques et environnement utilisés dans la phase de programmation, ainsi que le matériel et les logiciels nécessaires pour la réalisation de notre projet.

Nous avons aussi présenté l'implémentation du CNN dans notre plateforme embaquée. Nous avons, tout d'abord, précisé les spécifications de notre ordinateur ainsi que les logiciels complémentaires et nécessaires à notre travail. Ensuite, nous avons parlé notre programme qui se déroule en deux phases principales. Phase d'apprentissage : elle s'agit de créer un CNN en utilisant une base de données de chiffres en plusieurs types d'écriture ; Phase de test et de validation : dans cette phase, le programme d'apprentissage est testé puis utilisé afin de faire des prédictions sur différentes images d'entrées. Ces images sont enfin classées selon la probabilité d'appartenance à la classe prédite. Enfin nous avons exposé les résultats de l'implémentation sur RPI 3 B+ qui est renforcé avec l'accélérateur NSC 2 d'Intel.

CONCLUSION GENERALE

Dans ce projet, nous avons brièvement abordé le domaine de l'intelligence artificielle. Nous avons expliqué deux de ses branches les plus utilisées qui sont l'apprentissage automatique et l'apprentissage profond, tout en se basant sur les réseaux de neurones artificiels, en particulier les réseaux de neurones convolutionnels. Nous avons présenté les différentes couches qui constituent l'algorithme des réseaux de neurones convolutionnels : couche convolutionnelle, couche de pooling, couche de démarrage, couche de correction, couche entièrement connectée et le réseau résiduel. Nous avons également fait un aperçue sur les bibliothèques et frameworks utilisées pour la programmation, ainsi qu'au matériel et logiciels. Le matériel s'agit du Raspberry Pi 3 B+ et Neural Compute Stick 2 d'Intel. Les logiciels, précisément distribution Intel d'OpenVINO toolkit qui comprend : l'optimisateur du modèle et le moteur d'inférence, permettent essentiellement de faire fonctionner les NSC 2 avec le RPI 3 B+. Parmi ceux-ci, la. Nous avons exposé les différentes étapes de l'implémentation commençant de la réalisation d'un CNN qui fait la classification des chiffres manuscrits à son implémentation sur le Raspberry Pi utilisé avec le dispositif d'accélération d'Intel Neural Compute Sick. Ceci est basé sur le framework d'OpenVINO toolkit d'Intel qui permet de convertir et de charger des modèles sur l'accélérateur pour qu'il puisse fonctionner avec le Raspberry Pi. A la fin, nous avons pu atteindre notre objectif qui vise principalement à se familiariser avec les nouvelles plateformes embarquées et à faciliter l'utilisation des applications de celles-ci.

PERSPECTIVES DE RECHERCHE

Notre projet ne se limite pas juste à implémenter un CNN dans une plateforme embarquée. Au contraire, on peut l'améliorer davantage du côté matériel aussi bien que du côté programmation. D'abord, du côté matériel, nous pouvons ajouter un module de camera compatible avec le RPI 3 B+ pour changer le type d'entrée de notre système d'une image a une vidéo. On peut également ajouter une autre clé NSC2 pour augmenter la performance. En outre, du côté programmation, on peut changer quelques paramètres : on peut par exemple choisir un ensemble de données plus complexe comme les caractères du Kanji chinois ou ceux du Hiragana et Katakana japonais au lieu des chiffres manuscrits. Ou bien, pour un traitement d'images plus avancé, on peut utiliser « fashion-MNIST », l'ensemble de données qui regroupe des images de 28×28 pixels en niveaux de gris dont chacune est associée avec une étiquette se trouvant parmi 10 classes différentes de vêtements, chaussures et sacs.

Références

[1] Intel. (2020). *The power of 10*. Disponible: https://www.intel.com/content/www/us/en/products/docs/processors/core/10th-gen-processors.html

[2] H. Asem, 2019 فن تعليم الآلة deep learning Deep Learning هذه المحاضرة هي جزء من سلسلة محاضرات التعلم العميق. Disponible: https://www.youtube.com/watch?v=UKk3K0g7cP8&list=PL6-3IRz2XF5WyBLsw6yJYWIiFJ1OmmRyK

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, pp. 436-444, 2015. Disponible: https://www.nature.com/articles/nature14539#Abs1

[4] N. H. A. Minh, "Computer Vision Deployment on Edge Devices," Technology and Communication, VAASAN AMMATTIKORKEAKOULU UNIVERSITY OF APPLIED SCIENCES, 2019. Disponible: https://www.theseus.fi/bitstream/handle/10024/261705/thesis.pdf?sequence=2&isAllowed=y

[5] sketchbubble. (2020). *Artificial Vs Human Intelligence*. Disponible: https://www.sketchbubble.com/en/presentation-artificial-vs-human-intelligence.html

[6] S. Raj. (2019). *Machine Learning vs Deep Learning*. Disponible: https://mc.ai/machine-learning-vs-deep-learning-6/

[7] BoTreeTechnologies. (2018). *Decoding the Differences between Machine Learning & Deep Learning*. Disponible: https://mc.ai/decoding-the-differences-between-machine-learning-deep-learning/

[8] A. BADKAR. (2020). *Difference Between Machine Learning And Deep Learning*. Disponible: http://www.iamwire.com/2017/11/difference-between-machine-learning-and-deep-learning/169100

[9] A. Suarez. (2019, Date) Title. What Is the Difference Between Machine Learning and Deep Learning?,bismart. Pages. Disponible: https://blog.bismart.com/en/difference-between-machine-learning-deep-learning

[10] darqube. (2019). *Machine Learning Basics*. Disponible: https://darqube.io/machine-learning-basics.html

[11] H. Yates,2019, "Human Behavior,head,flower - Black And White Clipart Thinking, HD Png Download,"3-33074_human-behavior-head-flower-black-and-white-clipart,PngItem,PngItem,36 KB,PNG. Disponible:

https://www.pngitem.com/middle/hxTxb_human-behavior-head-flower-black-and-white-clipart/

[12] H. Asem, 2019 H. Asem فن تعليم الآلة youtube video deep learning deep learning هذه المحاضرة هي جزء من سلسلة محاضرات التعلم العميق Disponible: https://www.youtube.com/watch?v=a1lvQ6cCom4&list=PL6-3IRz2XF5XHyhWNs-jxFtfERv4NlZmm

[13] Intel. (2020). *Intel® Neural Compute Stick 2*. Disponible: https://software.intel.com/en-us/neural-compute-stick

[14] NVIDIA. (2019). *Embedded Systems for Next-Generation Autonomous Machines*. Disponible: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/

[15] G. cloud. (2020). *Edge TPU, IA en périphérie*. Disponible: https://cloud.google.com/edge-tpu?hl=fr

[16] Debian. (2020). *Qu'est-ce que Debian finalement ?* Disponible: https://www.debian.org/intro/about.fr.html

[17] K. N. Pascal MOASSE. (2020). *Qu'est ce qu'un réseau de neurones convolutif (ou CNN) ?* Disponible: https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5082166-quest-ce-quun-reseau-de-neurones-convolutif-ou-cnn

[18] J. Coget. (2009). *PHYSIOLOGIE DU NEURONE*. Disponible: http://passeport.univ-lille1.fr/site/biologie/scbio/Neurone/Neurone_web.publi/web/co/Intro.html

[19] R. Brette, "Modeles impulsionnels de réseaux de neurones biologiques," Doctorat, Ecole Doctorale Cerveau - Cognition - Comportement, Université Pierre et Marie Curie-Paris VI, 2003. Disponible: http://www.snv.jussieu.fr/brette/these.pdf

[20] Amélimélo. (2018). *Gymnastique pour neurone gréviste !* Disponible: https://www.amelimelo.fr/2038/

[21] D. P. Sauleau. (2017). *Cours Neurophysiologie*. Disponible: https://sites.google.com/site/aphysionado/home

[22] I. LILJEQVIST. (2016). *The Essence of Artificial Neural Networks*. Disponible: https://medium.com/@ivanliljeqvist/the-essence-of-artificial-neural-networks-5de300c995d6

[23] H. A. MAHDJOUBI Abderrahmane, "Implémentation d'un réseau de neurones convolutionnel pour la reconnaissance automatique des mots manuscrits en arabe " Master Technoogie, Electrical Engeneering Dr Yahia Fares Medea Algerie, 2019.

[24] P. Buyssens, "Fusion of different capture modes for face recognition applied to e_transactions," Sciences, Université de Caen, 2011. Disponible: https://hal.archives-ouvertes.fr/tel-01079134

[25] C. M. Bishop, *Neural networks for pattern recognition*, illustrée, réimprimée ed.: Oxford university press, 1995. Disponible: http://home.elka.pw.edu.pl/~ptrojane/books/Bishop%20-%20Neural%20Networks%20for%20Pattern%20Recognition.pdf

[26] B. H. Amel, "Vers Un Modèle De Classification Neuronale Des Données Médicales A Base De La Technologie FPGA," Doctorat, Génie Biomédical, Aboubakr Belkaïd – Tlemcen – Faculté de TECHNOLOGIE, 2017. Disponible: http://dspace.univ-tlemcen.dz/bitstream/112/11459/1/Doct.EBM.Baba%20Hamed.pdf

[27] K. Koidl, "Loss functions in classification tasks," *School of Computer Science and Statistic Trinity College Dublin, ADAPT Research Centre*, 2013. Disponible: https://www.scss.tcd.ie/Kevin.Koidl/cs4062/Loss-Functions.pdf

[28] N. developer. (2020). *Convolutional Neural Network (CNN)*. Disponible: https://developer.nvidia.com/discover/convolutional-neural-network

[29] J. Le. (2018, Date) Title. The 8 Neural Network Architectures Machine Learning Researchers Need to Learn, kdnuggets.com, Architecture, Deep Learning, Machine Learning, Neural Networks, Disponible: https://www.kdnuggets.com/2018/02/8-neural-network-architectures-machine-learning-researchers-need-learn.html

[30] G. H. Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929-1958, 2014. Disponible: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

[31] C. D. Jose Dolza, Ismail Ben Ayeda, "3D fully convolutional networks for subcortical segmentation in MRI: A large-scale study," Technologie, Ecole de technologie supérieure (ETS), Montreal, QC, Canada, 2017. Disponible: https://arxiv.org/pdf/1612.03925.pdf

[32] Y. B. Ian Goodfellow, Aaron Courville, "convolutional networks," MIT Press2016. Disponible: http://papers.nips.cc/paper/8824-macow-masked-convolutional-generative-flow.pdf

[33] K. N. Pascal MOASSE. (2020). *Découvrez les différentes couches d'un CNN*. Disponible: https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5083336-decouvrez-les-differentes-couches-dun-cnn

[34] J. Leonel. (2019). *Hyperparameters in Machine /Deep Learning*. Disponible: https://medium.com/@jorgesleonel/hyperparameters-in-machine-deep-learning-ca69ad10b981

[35] J. R. Gerbscheid, "People detection on the Pepper Robot using Convolutional Neural Networks and 3D Blob detection.," Bachelor, Faculty of Science, University of Amsterdam,2017.Disponible:https://staff.fnwi.uva.nl/a.visser/research/athome/JonathanGerbscheidBachelorThesis.pdf

[36] X. Z. Kaiming He, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition," Cornell university 2015. Disponible: https://arxiv.org/pdf/1512.03385.pdf

[37] J. OLAFENWA. (2018). *UNDERSTANDING RESIDUAL NETWORKS*. Disponible: https://towardsdatascience.com/understanding-residual-networks-9add4b664b03

[38] keitakurita. (2018). *An Intuitive Explanation of Why Batch Normalization Really Works (Normalization in Deep Learning Part 1)*. Disponible: http://mlexplained.com/2018/01/10/an-intuitive-explanation-of-why-batch-normalization-really-works-normalization-in-deep-learning-part-1/

[39] C. S. Sergey Ioffe, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Cornell University2015. Disponible: https://arxiv.org/pdf/1502.03167.pdf

[40] f. d. s. By data scientists. (2020). *Who is Anaconda?* Disponible: https://www.anaconda.com/

[41] S. D. C. M. License. (2018). *Spyder : The Scientific Python Development Environment*. Disponible: https://www.spyder-ide.org/

[42] A. A. Martin Abadi, Paul Barham, "TensorFlow : Large-Scale Machine Learning on HeterogeneousDistributedSystems,"2015.Disponible:https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf

[43] O. team. (2020). *OpenCV : About*. Disponible: https://opencv.org/about/

[44] G. Swinnen, *Apprendre à programmer avec Python*: O'Reilly, 2005. Disponible: https://books.google.dz/books?id=8gb_wAEACAAJ

[45] P. S. Foundation. (2020). *What is Python? Executive Summary*. Disponible: https://www.python.org/doc/essays/blurb/

[46] N. Smith, "Intel® Neural Compute Stick 2 and Open Source OpenVINO™ Toolkit," 2019. Disponible: https://software.intel.com/en-us/articles/intel-neural-compute-stick-2-and-open-source-openvino-toolkit

[47] A. Stahl, "New Neural Compute Stick 2 by INTEL," 2019. Disponible: https://www.codico.com/en/News/neural-compute-stick-intel.htm

[48] Intel. (2016). *Deep Learning Inference*. Disponible: https://software.intel.com/en-us/openvino-toolkit/deep-learning-inference

[49] raspberry.org, "Raspberry PiModel B+," 2017. Disponible: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf

[50] A. Eames. (2018, Date) Title. New Raspberry Pi model 3B+ 1.4 GHz, 330Mbit Ethernet, 802.11ac, PoE,RasPi.TV. Pages. Disponible: https://raspi.tv/2018/new-raspberry-pi-model-3b-1-4-ghz-330mbit-ethernet-802-11ac-poe

[51] OpenVINO™. (2020). *Install Intel® Distribution of OpenVINO™ toolkit for Windows* 10*.Disponible:https://docs.openvinotoolkit.org/latest/_docs_install_guides_installing_openvino_windows.html?fbclid=IwAR29fnsp6ocXevnofIpHxx8Lw2afynv2i7lr5HWrh-p264bs5tjXyIyp-fk#run-the-demos

[52] FILExt. (2020). *All about H5 Files*. Disponible: https://filext.com/file-extension/H5

[53] I. Corporation. (2020). *Custom Layer Extensions for the Model Optimizer*. Disponible: https://docs.openvinotoolkit.org/latest/openvino_docs_HOWTO_Custom_Layers_Guide.html

ANNEXES

Annexe A

Plusieurs types de technologie appartiennent à Anaconda, on peut citer :

1. Les fondamentaux : Jupyter, Pandas, SciPy et NumPy ;
2. ML : Keras, TensorFlow, PyToch et scikit-learn ;
3. La visualisation des données : NumPy, Bokeh, Plotly, Panel, Dash, Voila, Streamlit et HoloViz ;
4. Le traitement des images : Pillow, Scikit-image et OpenCV ;
5. Le calcul évolutif : Numba, Dask, Rapids et Apache Spark ;
6. La préparation des données/ETL : Apache Airflow et Intake ;
7. Le traitement du langage naturel (NLP) : NTLK, Gensim et SpaCy[40].

Pour créer un environnement dans Anaconda (pynq dans notre cas) on procède comme suit :

a. Créer et activer l'environnement : on exécute les instructions suivantes dans l'invite de commandes d'Anaconda :
    1. conda update conda --all
    2. conda create -n pynq python=3.7
    3. conda activate pynq
b. Installer les packages dans l'environnement
    1. conda install spyder
    2. conda install numpy scipy mkl-service libpython
    3. conda install m2w64-toolchain nose sphinx pydot-ng git //conda install m2w64-toolchain
    4. conda install tensorflow
    5. conda install -c conda-forge tensorflow=1.14
    6. conda install keras
    7. pip install matplotlib
    8. pip install pandas
    9. pip install sklearn
    10. conda install pillow
    11. conda install opencv
    12. conda install cv2
    13. conda install -c https://conda.binstar.org/t/TOKEN/j14r pydot

Annexe B

SPYDER peut également être utilisé comme une bibliothèque d'extension PyQt5, permettant aux développeurs de s'appuyer sur ses fonctionnalités et d'intégrer ses composants, comme la console interactive, dans leur propre logiciel PyQt. Ses composants sont les éléments de base d'un puissant IDE :

1. Correcteur (editor) : Travaille efficacement dans un éditeur multilingue avec un navigateur de fonctions/classes, des outils d'analyse de code, la complétion automatique de code, la division horizontale/verticale et l'action « go-to-definition » ;
2. Console IPython permet :
    - d'exploiter la puissance d'autant de consoles IPython que souhaité tout en bénéficiant de la souplesse d'une interface graphique complète ;
    - exécuter le code par ligne, cellule ou fichier ;
    - rendre les tracés en ligne ;
3. Explorateur de variables permet :
    - L'interaction et la modification des variables à la volée :
    - Tracer un histogramme ou des séries temporelles,
    - Modifier une date ou un tableau Numpy,
    - Trier une collection,
    - Fouiller dans les objets imbriqués.
4. Profileur : Trouve et élimine les goulots d'étranglement pour améliorer les performances du code ;
5. Débogueur : Suit chaque étape de l'exécution du code de manière interactive ;
6. Aide : Visualise instantanément les documents de n'importe quel objet employé dans le code ou qu'on souhaite s'y informer.

SPYDER contient également des Plugins qui permettent d'étendre ses fonctionnalités telles que Notebook, Terminal, UnitTest et Reports.

Pour l'installer, il est recommandé de le faire à travers l'invite de commande d'Anaconda avec l'instruction « conda install spyder » car il est inclus par défaut dans « Anaconda Python Distribution » qui est livré avec tout ce dont on a besoin pour démarrer dans un package tout en un[41].

Pour accéder à SPYDER depuis l'invite de commandes d'Anaconda il suffit d'exécuter ses instructions :

1. (base) C:\WINDOWS\system32>cd..
2. (base) C:\Windows>cd..
3. (base) C:\>conda activate pynq
4. (pynq) C:\>spyder

Où pynq est un environnement créé dans Anaconda.

Annexe C

TensorFlow a été choisi pour le prototypage rapide, la recherche avancée et la production. Pour la formation de très grands systèmes et éventuellement dans un contexte de production, l'API Estimateur offre une capacité de distribution sur plusieurs machines (multi-serveurs) au même niveau de simplicité que les Keras. Pour la recherche et le développement où le contrôle total est une nécessité, l'API de base TensorFlow dispose désormais de la fonction Eager Execution activé par défaut pour fournir une interface définie pour ses opérations. Tensor-Flow est conçu pour fournir une voie directe vers la production, quels que soient le langage et la plate-forme utilisés. TensorFlow Extended fournit un pipeline ML de production complet, qu'il soit dans le Cloud ou dans le navigateur où il s'exécute dans des environnements JavaScript, on utilise Tensor-Flow.js ; s'il est sur des appareils mobiles et de périphérie, on fait usage de TensorFlow Lite ; s'il est sur site, on utilisera simplement toute API qui convient. TensorFlow prend également en charge des écosystèmes de bibliothèques d'extension et de modèles :

- Ragged Tensors : une implémentation TensorFlow de listes imbriquées de longueur variable pour faciliter le stockage et le traitement de données de forme non uniforme ;
- TensorFlow Probability : une bibliothèque Python pour combiner facilement des modèles probabilistes et un DL sur le matériel TPU et GPU ;
- Tensor2Tensor - une bibliothèque de modèles et d'ensembles de données conçus pour le DL ;
- BERT pour Bidirectional Encoder Representations from Transformers, une nouvelle méthode de représentation des langues qui a obtenu des résultats impressionnants dans de nombreuses tâches récentes de traitement du langage naturel (NLP) [42].

Annexe D

Dans NumPy, comme les images à canaux multiples (images couleur) sont généralement représentées sous forme de tableaux tridimensionnels, les opérations telles que l'indexation, le découpage et le masquage avec d'autres tableaux au niveau du pixel sont très

efficaces. En outre, la matrice NumPy est adoptée par la bibliothèque de vision par ordinateur OpenCV comme structure de données universelle pour le stockage des images et elle simplifie considérablement le flux de travail et le débogage. D'autre part, les limites de cette bibliothèque sont que les algorithmes non vectoriels se forment lentement car ils doivent être implémentés en Python pur. De plus, la vectorisation peut faire passer la complexité des besoins en mémoire de constante à linéaire car la taille des tableaux temporaires doit être aussi grande que les entrées[4].

Annexe E

Les algorithmes d'OpenCV peuvent être utilisés dans plusieurs applications :

- Détecter et reconnaître des visages, identifier des objets ;
- Classer les actions humaines dans les vidéos ;
- Suivre les mouvements des caméras ;
- Suivre des objets en mouvement ;
- Extraire des modèles 3D d'objets ;
- Produire des nuages de points 3D à partir de caméras stéréo ;
- Assembler des images pour produire une image haute résolution d'une scène entière ;
- Trouver des images similaires dans une base de données d'images ;
- Supprimer les yeux rouges des images prises au flash ;
- Suivre les mouvements des yeux ;
- Reconnaître un paysage et établir des marqueurs pour le superposer à la réalité augmentée.

OpenCV compte plus de 47 000 personnes dans sa communauté d'utilisateurs et le nombre de téléchargements est estimé à plus de 18 millions. La bibliothèque est largement utilisée dans les entreprises, les groupes de recherche et par les organismes gouvernementaux. Il dispose d'interfaces C++, Python, Java et MATLAB et est compatible avec Windows, Linux, Android et Mac OS. OpenCV s'oriente principalement vers les applications de vision en temps réel et tire parti des instructions MMX et SSE lorsqu'elles sont disponibles. Une interface CUDA et OpenCL complète est actuellement en cours de développement. Il existe plus de 500 algorithmes et environ 10 fois plus de fonctions qui composent ou supportent ces algorithmes. OpenCV est écrit nativement en C++ et dispose d'une interface modélisée qui fonctionne de manière transparente avec les conteneurs STL[43].

Annexe F

L'interpréteur Python et la vaste bibliothèque standard sont disponibles sous forme source ou binaire sans frais pour toutes les principales plates-formes et peuvent être distribués librement. Python procure une productivité importante. Comme il n'y a pas d'étape de compilation, le cycle édition-test-débogage est très rapide. Le débogage des programmes Python est facile. En effet, un bogue ou une mauvaise entrée ne provoquera jamais un défaut de segmentation. Au contraire, lorsque l'interpréteur découvre une erreur, il soulève une exception. Lorsque le programme ne détecte pas l'exception, l'interpréteur imprime un tracé de pile. Un débogueur de niveau source permet :

- d'inspecter les variables locales et globales ;
- d'évaluer des expressions arbitraires ;
- de définir des points d'arrêt ;
- de parcourir le code ligne par ligne.

Le débogueur est écrit en Python lui-même, ce qui témoigne de la puissance introspective de Python. D'autre part, la façon la plus rapide de déboguer un programme est souvent d'ajouter quelques instructions d'impression à la source. Le cycle rapide d'édition-test-débogage rend cette approche simple et très efficace[45].

Annexe G

NCS 2 regroupe les caractéristiques suivantes :

- Proposée par le VPU Movidius™ Myriad™ X d'Intel® ;
- Jusqu'à 8 fois les performances du Intel® Movidius™ Neural Compute Stick ;
- Soutenu par la distribution Intel® d'OpenVINO™ toolkit ;
- Travail en temps réel sur l'inférence de dispositif et n'exige pas de connexion a cloud ;
- Assure le fonctionnement de plusieurs appareils sur la même plateforme pour augmenter les performances[47].

Le tableau III.2 ci-dessous regroupe les caractéristiques techniques de NCS 2

Tableau III.2 Spécifications techniques de NCS2 d'Intel®[47]

Spécifications	Neural Compute Stick 2
Unité de traitement visuelle (VPU)	La VPU Movidius™ Myriad™ X d'Intel®
Kit de développement de logiciels	La distribution Intel® d'OpenVINO™ toolkit
Systèmes d'exploitation supportés	Ubuntu* 16.04.3 LTS (64 bit), Windows® 10 (64 bit), CentOS* 7.4 (64 bit), Raspbian*, et d'autres via la distribution libre d'OpenVINO™
Frameworks supportées	TensorFlow*, Caffe*, MXNet*, ONNX*, et PyTorch* / PaddlePaddle* via la conversion ONNX*
Connectivité	USB 3.1 Type-A, USB 2.0 Type-A
Dimensions	72.5mm X 27mm X 14mm
Température de fonctionnement	0° - 40° C
Numéro de référence du matériel	964486
PDSF	69$ Dollars Américain à partir du 14 juillet 2019
Plateformes supportées	×86_64, ARM

Annexe H

Le tableau III.3 suivant regroupe les caractéristiques techniques de la carte RPI 3 B+ :

Tableau III.3 Spécifications techniques du RPI 3 B+[49]

Spécifications	RPI 3 B+
Processeur	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Mémoire	1GO LPDDR2 SDRAM
Connectivité	- La double bande 2.4GHz et 5GHz de la IEEE 802.11.b/g/n/ac WLAN, Bluetooth 4.2, BLE - Gigabit Ethernet via USB 2.0 (debit maximal de 300Mbps) - 4 × ports USB 2.0.
Accès	En-tête GPIO étendu à 40 broches
Vidéo et Audio	- 1 × full size HDMI - Port d'affichage MIPI DSI - Port de caméra MIPI CSI - Sortie stéréo à 4 pôles et un port vidéo composite
Multimédia	H.264, décodage MPEG-4 (1080p30) ; codage H.264 (1080p30) ; graphiques OpenGL ES 1.1, 2.0
Support de la carte SD	Format Micro SD pour le chargement du système d'exploitation et le stockage des données
Alimentation d'entrée	- 5V/2,5A DC via un connecteur micro USB - 5V DC via l'entête GPIO - Alimentation par Ethernet (PoE). Activée, elle nécessite un HAT de PoE pour la séparation
Environnement	Température de fonctionnement, 0-50°C
Compatibilité	- Directive sur la compatibilité électromagnétique (CEM) 2014/30/UE - Directive sur la restriction des substances dangereuses (RoHS) 2011/65/UE[10]
Durée de vie de la production	Le Raspberry Pi 3 modèle B+ restera en production au moins jusqu'à janvier 2023.
Dimensions	85mm x 56mm x 17mm

[10] La liste complète des certifications locales et régionales du produit est sur : https://www.raspberrypi.org/documentation/hardware/raspberrypi/conformity.md

Annexe I

Le programme utilisé pour la conversion de la bibliothèque Keras vers TensorFlow

```
from keras.models import load_model
from keras import backend as K
import tensorflow as tf
keras_model = "Keras_model/model.h5"
def freeze_session(session, keep_var_names=None, output_names=None, clear_devices=True):
from tensorflow.python.framework.graph_util import convert_variables_to_constants
graph = session.graph
with graph.as_default():
    freeze_var_names = list(set(v.op.name for v in tf.global_variables()).difference(keep_var_names or []))
    output_names = output_names or []
    output_names += [v.op.name for v in tf.global_variables()]
    input_graph_def = graph.as_graph_def()
    if clear_devices:
      for node in input_graph_def.node:
        node.device = ""
    frozen_graph = convert_variables_to_constants(session, input_graph_def,
                                        output_names, freeze_var_names)
    return frozen_graph
# loading keras model
K.set_learning_phase(0)
model = load_model(keras_model)

# create a frozen-graph of the keras model
frozen_graph = freeze_session(K.get_session(),
                    output_names=[out.op.name for out in model.outputs])

# save model as .pb file
tf.train.write_graph(frozen_graph, "TF_model/", "tf_model.pb", as_text=False)
```