

UNIVERSITÉ OUAGA 1 PR JOSEPH KI ZERBO

MÉMOIRE DE MASTER

---

# Optimisation et compression de modèles de réseaux de neurones artificiels

Application à la reconnaissance d'images de plantes  
sur terminaux de faibles capacités

---

*Auteur :*  
KABORE Abdoul Kader

*Encadreur :*  
Dr Tegawendé F. BISSYANDE

*Superviseur :*  
Pr Oumarou SIE

*Document de mémoire soumis pour satisfaire aux exigences du  
Diplôme de Master 2 Système d'Information et Réseau (SIR), option Ingénierie  
Logicielle et Système d'Information Informatisé (ILS2I)*

à

l'Unité de Formations et de Recherches en Sciences Exactes et Appliquées  
Département d'informatique

31 janvier 2022

## DECLARATION DE PATERNITE

Je soussigné, KABORE ABDOUL KADER, déclare que ce mémoire intitulé, « Optimisation et compression de modèles de réseaux de neurones artificiels - Application à la reconnaissance d'images de plantes sur terminaux de faibles capacités » et les travaux qui y sont présentés sont les miens. Je confirme que :

- Ce travail a été effectué entièrement ou principalement durant mon Master à l'Université Ouaga 1 Pr. Joseph Ki Zerbo
- Si une partie quelconque de ce mémoire a déjà été présentée en vue d'un diplôme ou de toute autre qualification de cette université, cela est clairement indiqué.
- Là où j'ai consulté le travail publié d'autres auteurs, cela est toujours clairement attribué.
- Là où j'ai cité du travail d'autrui, la source est toujours indiquée. À l'exception de telles citations, ce travail est tout à fait le mien.
- J'ai reconnu toutes les principales sources d'aide. - Lorsque ce travail est basé sur un travail effectué par moi-même conjointement avec d'autres, j'ai clairement expliqué ce que les autres ont fait et ce que j'ai moi-même apporté.

Signature :

Date :

*Dédicaces,*

*A mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études,*

*A mes chers frères et soeurs, pour leur appui et encouragement permanent,*

*A toute ma famille, à mes amis*

*Que ce travail soit l'accomplissement de vos voeux tant allégués, et le fruit de votre soutien infaillible.*

*« Some people call this artificial intelligence, but the reality is this technology will enhance us. So instead of artificial intelligence, I think we'll augment our intelligence. »*

Ginni Rometty

## *Remerciements*

Nous tenons vivement à remercier les différentes structures qui ont bien voulu nous accueillir. Ainsi nous adressons nos remerciements à tous ceux qui, d'une manière ou d'une autre, ont contribué à l'aboutissement de ce travail; et sans le concours desquels, ce travail n'aurait probablement pas eu la même considération, ni la même réussite. Notre gratitude va de ce fait :

- au Dr. Tegawendé F. BISSYANDE, qui n'a ménagé aucun effort pour nous guider et nous fournir le meilleur encadrement possible;
- au Pr Oumarou SIE et l'ensemble des enseignants du département informatique de l'UFR-SEA;
- à toute l'administration de l'Unité de Formation et de Recherches en Sciences Exactes Appliquées ainsi qu'à tout le corps professoral pour la qualité de la formation reçue;
- aux parents et amis pour tout leur soutien;
- aux camarades de classe et à tous ceux qui nous ont soutenus

# Table des matières

<b>DECLARATION DE PATERNITE</b>	<b>ii</b>
<b>Remerciements</b>	<b>v</b>
<b>1 Généralités</b>	<b>4</b>
1.1 De l'intelligence artificielle à l'apprentissage en profondeur . . . . .	4
1.2 Apprentissage en profondeur (ou <i>deep learning</i> ) . . . . .	5
1.2.1 Description et contexte . . . . .	5
1.2.2 Domaines d'application . . . . .	5
1.3 Le réseau neuronal artificiel et système visuel humain . . . . .	6
1.3.1 Introduction . . . . .	6
1.3.2 Définitions . . . . .	7
Le perceptron . . . . .	7
Le perceptron multicouche, fonction de combinaison et fonction d'activation . . . . .	9
Mode d'apprentissage d'un réseau de neurones . . . . .	10
Le surapprentissage ou <i>overfitting</i> . . . . .	11
Rétropropagation et Élagage . . . . .	11
1.4 <i>Deep learning</i> et classification d'images . . . . .	11
1.4.1 Introduction . . . . .	11
1.4.2 La notion de <i>features</i> dans une image . . . . .	12
1.4.3 Extraction des <i>features</i> . . . . .	15
Le gradient d'une image . . . . .	15
La détection des bords avec le filtre de Canny . . . . .	15
La localisation des coins avec le détecteur de Harris-Stephens .	16
SIFT . . . . .	16
1.5 Les réseaux de neurones convolutifs (ou CNN) . . . . .	17
1.5.1 Qu'est-ce que la convolution? . . . . .	17
1.5.2 Introduction aux réseaux de neurones convolutifs (CNN) .	19
1.5.3 Architecture d'un CNN . . . . .	19
Description de la partie convolutive des modèles CNN . . . . .	19
<b>2 Problématiques, motivations et objectifs de notre recherche</b>	<b>21</b>
2.1 Problématique . . . . .	21
2.1.1 <i>Deep learning</i> et ressources mises en jeu . . . . .	21
2.1.2 L'exploitation des modèles de réseaux de neurones artificiels .	22
2.2 Motivations et objectifs . . . . .	22
2.2.1 Motivations . . . . .	22
2.2.2 Objectifs de recherches . . . . .	23

<b>3 Etat de l'art</b>	<b>24</b>
3.1 Les méthodes d'optimisation de modèle de <i>deep learning</i> . . . . .	24
3.1.1 La régression linéaire univariée en rappel . . . . .	24
3.1.2 La descente du gradient . . . . .	26
3.1.3 La descente de gradient par mini-lot . . . . .	27
Avantages . . . . .	28
Inconvénients . . . . .	28
3.1.4 Momentum . . . . .	28
3.2 Les travaux de recherches en optimisation de modèle de <i>deep learning</i> .	28
3.3 La compression de modèles de <i>deep learning</i> . . . . .	29
3.3.1 L'élagage des réseaux de neurones profonds pour les rendre rapides et petits . . . . .	29
3.3.2 Le partage de poids : le partage pondéral . . . . .	30
3.3.3 Le codage de Huffman . . . . .	32
Fonctionnement . . . . .	32
3.3.4 Les travaux de recherche sur la compression de modèles de <i>deep learning</i> . . . . .	33
<b>4 Approche</b>	<b>34</b>
4.1 La méthode Adam pour l'optimisation de modèle de <i>deep learning</i> . . . . .	34
4.1.1 Le choix de Adam . . . . .	34
4.1.2 Fonctionnement et points clés de Adam . . . . .	35
4.2 Elagage massif et progressif, partage pondéral et codage de Huffman .	35
4.2.1 Présentation . . . . .	36
4.2.2 Algorithme et équations . . . . .	37
<b>5 Réalisations</b>	<b>38</b>
5.1 Constitution du jeu de données . . . . .	38
5.1.1 La compétition " <i>iNaturalist</i> " . . . . .	38
5.1.2 <i>ImageNet</i> . . . . .	39
5.2 Les outils . . . . .	39
5.2.1 Les outils provenant d'éditeurs tierces . . . . .	39
Keras . . . . .	39
Tensorflow . . . . .	39
keras_to_tensorflow.py . . . . .	39
Deep Playground . . . . .	39
TOCO : <i>TensorFlow Lite Optimizing Converter</i> . . . . .	39
Android Studio . . . . .	40
Android Development Kit . . . . .	40
Android Native Development Kit . . . . .	40
Android Neural Networks . . . . .	40
5.2.2 Outils développés dans le cadre de notre travail . . . . .	40
Téléchargement automatique d'image . . . . .	40
Comparison d'algorithmes d'optimisation de <i>deep learning</i> . . . . .	40
5.3 Démarche . . . . .	40
5.3.1 Entrainement du modèle . . . . .	41
5.3.2 Compression : études expérimentales et résultats . . . . .	41
Elagage "aveugle" et monitoring des performances . . . . .	42
Un compromis entre taux de compression et précision . . . . .	43
5.3.3 Conversions du modèle . . . . .	44

5.3.4 Application mobile de reconnaissance d'images de plantes sur Android . . . . .	45
<b>6 Résultats et évaluations</b>	<b>46</b>
6.1 Résultats . . . . .	46
6.1.1 Performances . . . . .	46
6.1.2 Utilisation des ressources par notre application . . . . .	46
6.1.3 Evaluation de notre approche . . . . .	47
6.2 Difficultés . . . . .	48

# Table des figures

1.1	Orange . . . . .	6
1.2	Orange_Set . . . . .	7
1.3	Simple_Example_Perceptron . . . . .	8
1.4	MLP_Example . . . . .	9
1.5	Neural_Structure . . . . .	10
1.6	Fruits . . . . .	12
1.7	Pixel . . . . .	13
1.8	Tour_Eiffel . . . . .	14
1.9	Tour_Matching . . . . .	14
1.10	Coin . . . . .	16
1.11	Convolution1 . . . . .	18
1.12	Convolution2 . . . . .	18
1.13	Conv . . . . .	19
1.14	Conv2 . . . . .	20
3.1	Régression linéaire univariée . . . . .	25
3.2	Visualisation de la descente du gradient . . . . .	26
3.3	Illustration de la descente du gradient . . . . .	27
3.4	Architecture TVM.ai . . . . .	30
3.5	Principe d'élagage . . . . .	31
3.6	Notion de partage pondéral . . . . .	32
3.7	Arbre . . . . .	33
4.1	Adam . . . . .	35
4.2	Algorithme de Adam . . . . .	36
5.1	Logo de Keras . . . . .	41
5.2	Précision atteinte - Training (couleur orange) et Testing (couleur bleue) . . . . .	42
5.3	Taille du modèle . . . . .	42
5.4	... . . . .	43
5.5	... . . . .	44
5.6	Analyse de notre APK (version debug) . . . . .	45
6.1	Captures d'écran de notre application . . . . .	47
6.2	Capture d'écran des détails de notre application . . . . .	48
6.3	Modèle d'origine et modèle compressé . . . . .	49
6.4	Modèle compressé et modèle final . . . . .	49

# Liste des tableaux

3.1 Codage de Huffman - Occurrence des lettres . . . . .	32
3.2 Codage de Huffman - Codes de correspondance . . . . .	32
5.1 Résultats de l'évaluation lors de tests . . . . .	44

# Liste des Abbreviations

<b>API</b>	Application Programming Interface
<b>CPU</b>	Central Processing Unit
<b>GPU</b>	Graphics Processing Unit
<b>IA</b>	Intelligence Artificielle
<b>ILS2I</b>	Ingénierie Logicielle et Système d'Information Informatisé
<b>NDK</b>	Native Development Kit.
<b>NNAPI</b>	Neural Network APIde
<b>SDK</b>	Software Development Kit.
<b>SEA</b>	Sciences Exactes et Appliquées
<b>SIR</b>	Système d'Information et Réseau
<b>TOCO</b>	Tensorflow lite Optimizing COnverter
<b>TPU</b>	Tensorflow Processing Unit
<b>UFR</b>	Unité de Formations et de Recherches



# Résumé

Les modèles d'apprentissage en profondeur, notamment les réseaux de neurones, sont connus pour être gourmands en ressources informatiques (mémoire, stockage, puissance de calcul). L'entraînement des réseau de neurones n'étant une opération continue, nous pouvons le faire sur des ordinateurs très performants. Le problème réside dans l'exploitation de ces réseaux qui peuvent occuper beaucoup d'espace de stockage et nécessiter une grande puissance de calcul. Dans le but de pouvoir fournir des applications intelligentes grand public et plus particulièrement aux populations qui disposent de terminaux aux capacités limitées, nous avons travaillé sur l'optimisation et compression des réseaux de neurones. C'est ainsi que nous proposons dans ce mémoire une approche pour la compression qui atteint un taux de 98,13%. Afin de tester notre approche, nous avons mis en place une application de reconnaissance d'images de plantes sur Android. Cette application atteint une précision de 93%.

# Introduction générale

Apparues dans les années 1950, les recherches en intelligence artificielle ont été vite abandonnées ayant constaté les ressources de calculs limitées à cette période. Des dizaines d'années plus tard, ce domaine et surtout l'apprentissage en profondeur renaissent et des applications dotées de capacités intelligentes prennent le contrôle de nombreux aspects de notre vie. Si, ailleurs, les ressources informatiques sont facilement accessibles au grand public pour des applications dotées de capacités intelligentes, dans notre contexte, le problème est différent. Ainsi donc, entre le manque d'infrastructures et la faible couverture réseau, Dans ce mémoire, nous étudions les possibilités d'optimisation et de compression de modèles issus de l'apprentissage en profondeur pour une utilisation dans un environnement contraint : pas d'accès à Internet et capacités limitées (mémoire, stockage et puissance de calcul).

Nous avons particulièrement travaillé sur les réseaux de neurones artificielles qui sont une approche de l'apprentissage en profondeur. La construction de ces réseaux de neurones, en plus de faire appel à d'énormes quantités de données, nécessitent également une grande puissance de calcul. Bien que cette opération puisse être effectuée sur des ordinateurs à grande capacité, l'exploitation des réseaux résultants dans des applications grand public reste problématique. Les terminaux cibles ont généralement une capacité modeste. Grâce à ce mémoire, nous faisons les contributions suivantes :

- Une motivation de la recherche de solutions pour promouvoir l'inclusion d'applications intelligentes au sein de nos communautés.
- La proposition d'une approche beaucoup plus orientée pour la compression de modèles d'apprentissage en profondeur avec l'application de la reconnaissance d'images de plantes.
- Pour finir, nous montrons que notre approche peut favoriser la naissance d'une autre manière d'utiliser nos appareils mobiles et systèmes aux capacités limitées et qui constituent les seules ressources de calculs disponibles dans un grand nombre de communautés.

La suite de cet mémoire est organisé comme suit : nous allons d'abord faire un tour d'horizon sur les fondamentaux de l'apprentissage en profondeur et plus généralement les réseaux de neurones artificiels. Ces notions nous permettrons de mieux situer notre travail de recherche qui porte sur le thème "**Optimisation et compression de modèles de réseaux de neurones artificiels : application à la reconnaissance d'images de plantes sur terminaux de faibles capacités**". Par la suite, nous reviendrons plus en profondeur sur les problématiques, motivations et objectifs de notre travail avant de vous parler de l'état de l'art des connaissances sur le sujet. Après cela, nous présenterons notre approche pour finir en vous faisant le point sur les différentes réalisations et résultats.

## Chapitre 1

# Généralités

Ce premier chapitre introduit les concepts généraux ayant trait à notre travail. Notamment, nous présentons dans cette partie de manière sommaire l'historique de l'intelligence artificielle, la mise en oeuvre d'un concept plus avancé de cette dernière : l'apprentissage en profondeur et particulièrement les réseaux de neurones.

### Sommaire

<b>1.1</b>	<b>De l'intelligence artificielle à l'apprentissage en profondeur</b>	4
<b>1.2</b>	<b>Apprentissage en profondeur (ou <i>deep learning</i>)</b>	5
1.2.1	Description et contexte	5
1.2.2	Domaines d'application	5
<b>1.3</b>	<b>Le réseau neuronal artificiel et système visuel humain</b>	6
1.3.1	Introduction	6
1.3.2	Définitions	7
<b>1.4</b>	<b><i>Deep learning</i> et classification d'images</b>	11
1.4.1	Introduction	11
1.4.2	La notion de <i>features</i> dans une image	12
1.4.3	Extraction des <i>features</i>	15
<b>1.5</b>	<b>Les réseaux de neurones convolutifs (ou CNN)</b>	17
1.5.1	Qu'est-ce que la convolution?	17
1.5.2	Introduction aux réseaux de neurones convolutifs (CNN)	19
1.5.3	Architecture d'un CNN	19

### 1.1 De l'intelligence artificielle à l'apprentissage en profondeur

Le terme **intelligence artificielle** remonte à 1956 lors de la Conférence de Dartmouth. Ce terme faisait référence à l'écriture de programmes informatiques qui tenterait d'imiter l'intelligence humaine [4]. Plus tard, au sein de celui-ci, un nouveau domaine appelé **l'apprentissage automatique** a émergé. C'est ainsi au lieu d'écrire un programme pas à pas pour résoudre un problème donné (ce qui est une approche traditionnelle de l'intelligence artificielle [27]), il s'agira maintenant de comprendre le problème à partir d'un grand nombre de données collectées sur le sujet.

L'apprentissage automatique est un domaine très vaste avec beaucoup d'applications. Appelé "reconnaissance de modèle" [23] à l'origine, ses algorithmes sont devenus très nombreux et complexes sur le plan mathématique allant jusqu'à s'inspirer du fonctionnement du cerveau donnant ainsi naissance à l'apprentissage en profondeur. Fondamentalement, l'apprentissage en profondeur est une partie de l'apprentissage automatique et l'apprentissage automatique, une partie de l'IA.

## 1.2 Apprentissage en profondeur (ou *deep learning*)

Le *deep learning* ou **apprentissage en profondeur**, dérivé du "*machine learning*" où la machine est capable d'apprendre par elle-même, contrairement à la programmation où elle se contente d'exécuter à la lettre des règles pré-déterminées. Néanmoins cette apprentissage autonome s'appuie sur d'énormes quantité de données. Les techniques de l'apprentissage en profondeur ont permis des progrès importants et rapides dans les domaines de l'analyse du signal sonore ou visuel, notamment la reconnaissance faciale, la reconnaissance vocale, la vision par ordinateur, ainsi que le traitement automatisé du langage. Depuis les années 2000, les progrès de l'apprentissage en profondeur suscitent des investissements privés, universitaires et publics importants, notamment de la part des GAFA (Google, Apple, Facebook, Amazon).

### 1.2.1 Description et contexte

Le *deep learning* s'appuie sur d'énormes quantités de données pour permettre à la machine d'apprendre d'elle-même. Vu dans ce sens, une des perspectives des techniques de l'apprentissage en profondeur est le remplacement de certains travaux, encore relativement laborieux pour les techniques les plus avancées (*machine learning* notamment)

C'est ainsi que les recherches dans le domaine du *deep learning* s'efforcent de construire de meilleures représentations du réel à partir de données non labellisées à grande échelle en s'inspirant des dernières avancées en neuroscience - les neurosciences sont les études scientifiques du système nerveux, tant du point de vue de sa structure que de son fonctionnement, depuis l'échelle moléculaire jusqu'au niveau des organes, comme le cerveau, voire de l'organisme tout entier.

### 1.2.2 Domaines d'application

L'apprentissage en profondeur s'applique à divers secteurs des TIC, notamment :

- la reconnaissance visuelle — par exemple, d'un panneau de signalisation par un robot ou une voiture autonome — et vocale);
- la robotique ;
- la bioinformatique, par exemple, pour l'étude de l'ADN ;
- la reconnaissance ou la comparaison de formes ;
- la sécurité, comme analyser les émotions révélées par un visage photographié ou filmé dans un aéroport ;
- la santé pour un diagnostic médical (ex. : reconnaissance automatique d'un cancer en imagerie médicale) ;
- la pédagogie assistée par l'informatique ;
- l'art, par exemple reproduire une oeuvre artistique à partir d'une photo à l'ordinateur.

Une application de l'apprentissage en profondeur à la santé publique est le projet Horus de la société Eyra [7]. Il s'agit d'un appareil portable utilisant la plate-forme NVidia Jetson, qui aide les malvoyants ou les aveugles à s'orienter et à reconnaître des personnes ou des objets, en retranscrivant en audio une image captée par une caméra.

En physique, l'apprentissage en profondeur est utilisé pour la recherche sur les particules exotiques [5].

## 1.3 Le réseau neuronal artificiel et système visuel humain

*"Réseau de neurones en anglais neural network, un magnifique paradigme de programmation d'inspiration biologique qui permet à un ordinateur d'apprendre à partir de données d'observation." [19]*

(Michael Nielsen)

### 1.3.1 Introduction

Le système visuel est une machine complexe dont le fonctionnement sophistiqué tient les chercheurs en admiration. Considérons la photo suivante :




---

FIGURE 1.1 – Une orange.

La plupart des gens reconnaissent sans effort l'image ci-dessus comme étant celle d'un fruit : l'orange. Cette facilité est trompeuse. Dans chaque hémisphère de notre cerveau, nous avons un cortex visuel primaire, également appelé V1, contenant 140 millions de neurones, avec des dizaines de milliards de connexions entre eux. Et pourtant, la vision humaine implique non seulement V1, mais toute une série de cortex visuels [29] - V2, V3, V4 et V5 - effectuant un traitement d'images de plus en plus complexe. Nous avons dans nos têtes un supercalculateur parfaitement adapté à la compréhension du monde visuel. Reconnaître une orange n'est pas donc une chose facile. Au contraire, nous, les humains, sommes incroyablement bons pour comprendre ce que nos yeux nous montrent. Mais presque tout ce travail est fait inconsciemment. Ainsi, nous ne voyons généralement pas la difficulté d'un problème que nos systèmes visuels résolvent.

La difficulté de la reconnaissance visuelle devient évidente si nous essayons d'écrire un programme informatique pour reconnaître une orange. Ce qui semble facile lorsque nous le faisons nous même devient soudainement extrêmement difficile. Des intuitions simples sur la façon dont nous reconnaissions cette orange - "sa couleur, sa forme, ..." - ne sont pas si simples à exprimer de manière algorithmique. Lorsque vous essayez de rendre ces règles précises, nous nous perdons rapidement dans un tas d'exceptions, de mises en garde et de cas particuliers (une orange coupée en deux, une orange de couleur verte ...). Les réseaux de neurones abordent le problème différemment. L'idée est de prendre un grand nombre de d'images d'oranges, appelés données d'entraînement, puis développer un système pouvant tirer parti de ces données. En d'autres termes, le réseau neuronal utilisera ces données pour déduire automatiquement des règles de reconnaissance. En outre, en augmentant la qualité et la quantité de données 1.2, le réseau peut en apprendre davantage sur les oranges et améliorer ainsi sa précision.



FIGURE 1.2 – Collages d'images d'orange.

### 1.3.2 Définitions

Qu'est-ce qu'un réseau neuronal artificiel ? Pour commencer, nous allons d'abord expliquer un type de neurone artificiel appelé perceptron, puis le **perceptron multicouche**

#### Le perceptron

Il a été développé dans les années 1950 par le scientifique Frank Rosenblatt [30], inspiré par les travaux antérieurs de Warren McCulloch [32] et Walter Pitts [31]. Un perceptron prend plusieurs entrées binaires,  $x_1, x_2, \dots$  et produit une seule sortie binaire :

Dans l'image (figure 1.3) présentée ici, le perceptron a trois entrées,  $x_1, x_2, x_3$ . En général, il pourrait y avoir plus ou moins d'intrants. Rosenblatt a proposé une règle simple pour calculer la sortie. Il a introduit des poids,  $w_1, w_2, \dots$  des nombres réels exprimant l'importance des entrées respectives dans la sortie. La sortie du neurone, 0 ou 1, est déterminée par le fait que la somme pondérée  $\sum_j w_j x_j$  est inférieure ou supérieure à une valeur seuil. Tout comme les poids, le seuil est un nombre réel qui est un paramètre du neurone. Pour le dire en termes algébriques plus précis :

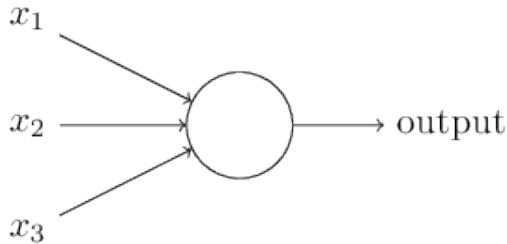


FIGURE 1.3 – Exemple simple d'un Perceptron.

$$\text{sortie} = \begin{cases} 0 & \text{si } \sum_j w_j x_j \leq \text{seuil} \\ 1 & \text{si } \sum_j w_j x_j > \text{seuil} \end{cases}$$

C'est tout ce qu'il y a à savoir du fonctionnement d'un perceptron ! C'est le modèle mathématique de base. Une façon de penser au perceptron est qu'il s'agit d'un dispositif qui prend des décisions en pondérant les preuves. Nous vous proposons l'exemple (pas très réaliste) qui suit pour plus de compréhension. Supposons que le week-end arrive, et vous avez entendu dire qu'il y aura un festival de musique dans votre ville. Vous aimez la musique et essayez de décider si vous voulez ou non aller au festival. Vous pouvez prendre votre décision en évaluant trois facteurs :

1. La météo est-elle bonne ?
2. Est-ce que y'aura t-il une personne pour vous accompagner ?
3. Le festival est-il près du transport en commun ? (Vous ne possédez pas de moyen de déplacement).

Nous pouvons représenter ces trois facteurs par les variables binaires correspondantes  $x_1, x_2$  et  $x_3$ . Par exemple, nous aurions  $x_1 = 1$  si le temps est bon et  $x_1 = 0$  si le temps est mauvais. De même,  $x_2 = 1$  si vous seriez accompagné et  $x_2 = 0$  sinon. Et de même pour  $x_3$  et le transport en commun.

Maintenant, supposons que vous adorez la musique, à tel point que vous êtes heureux d'aller au festival même si personne d'autre n'est intéressée pour vous accompagner et que le festival est difficile à atteindre. Mais peut-être détestez-vous vraiment le mauvais temps, et il n'y a pas moyen d'aller au festival si le temps est mauvais. Vous pouvez utiliser des perceptrons pour modéliser ce type de prise de décision. Une façon de faire est de choisir un poids  $w_1 = 6$  pour le temps et  $w_2 = 2$  et  $w_3 = 2$  pour les autres conditions. La valeur plus élevée de  $w_1$  indique que le temps compte beaucoup pour vous, beaucoup plus que le fait d'être accompagné ou la proximité des transports en commun. Enfin, supposons que vous choisissiez un seuil de 5 pour le perceptron. Avec ces choix, le perceptron implémente le modèle de prise de décision souhaité, en produisant 1 chaque fois que la météo est bonne et 0 chaque fois que le temps est mauvais. Que vous soyez accompagné ou que le transport en commun soit à proximité, cela ne fait aucune différence.

En faisant varier les poids et le seuil, nous pouvons obtenir différents modèles de prise de décision. Par exemple, supposons que nous choisissons plutôt un seuil de 3. Le perceptron déciderait alors que vous devriez aller au festival chaque fois que le temps était clément ou que le festival était proche des transports en commun et

qu'une autre personne était prête à vous rejoindre. En d'autres termes, ce serait un modèle différent de prise de décision. La suppression du seuil signifie que vous êtes plus enclin à aller au festival.

A travers cet exemple, si nous voulons mettre en place un perceptron pour décrire si une image est celle d'une orange ou non, nous devons procéder également de la sorte en répondant à des questions telles que :

1. Quels facteurs caractérisent une orange ? La forme ? La couleur ? Autres ?
2. Quel poids donné à chaque facteur ?
3. Etc ...

### Le perceptron multicouche, fonction de combinaison et fonction d'activation

Le perceptron est, essentiellement, un modèle linéaire. Sa capacité de modélisation est donc très limitée. Le Perceptron multicouche introduit des couches intermédiaires (ou couches cachées, *hidden layers* en anglais) entre la couche d'entrée et celle de sortie. Chaque neurone d'une couche est connecté à tous les neurones de la couche au-dessus de lui. D'où l'appellation de perceptron multi-couche (ou *multi-layer perceptron*, souvent abrégé *MLP*, en anglais).

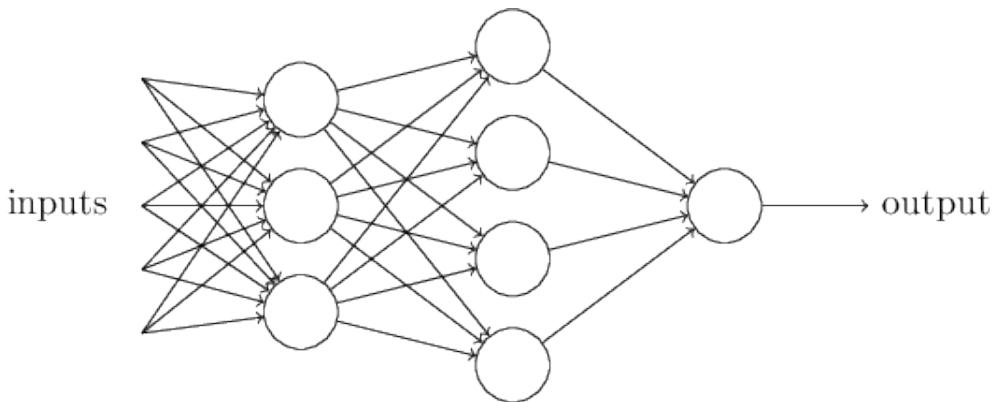


FIGURE 1.4 – Exemple de perceptron multicouche.

Considérons un neurone quelconque, il reçoit des neurones en amont un certain nombre de valeurs via ses connexions synaptiques, et il produit une certaine valeur en utilisant une fonction de combinaison. Cette fonction peut donc être formalisée comme étant une fonction vecteur-à-scalaire, notamment : Les réseaux de type MLP (*multi-layer perceptron*) calculent une combinaison linéaire des entrées, c'est-à-dire que la fonction de combinaison renvoie le produit scalaire entre le vecteur des entrées et le vecteur des poids synaptiques.

La fonction d'activation (ou fonction de seuillage, ou encore fonction de transfert) sert à introduire une non-linéarité dans le fonctionnement du neurone.

Les fonctions de seuillage présentent généralement trois intervalles :

1. En dessous du seuil, le neurone est non-actif (souvent dans ce cas, sa sortie vaut 0 ou -1);

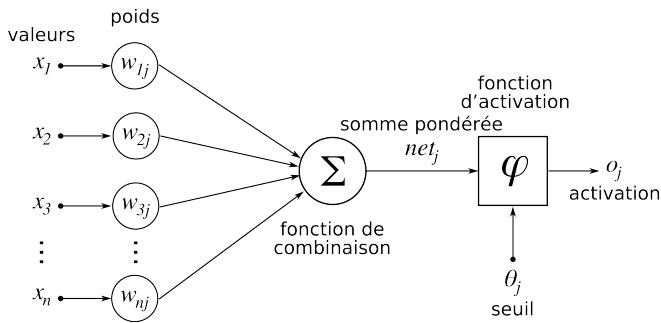


FIGURE 1.5 – Structure d'un neurone artificiel. Le neurone calcule la somme de ses entrées puis cette valeur passe à travers la fonction d'activation pour produire sa sortie.

2. Aux alentours du seuil, le neurone est en phase de transition;
3. Au-dessus du seuil, le neurone est actif (souvent dans ce cas, sa sortie vaut 1).

Dans le domaine des réseaux de neurones artificiels, la fonction d'activation est une fonction mathématique appliquée à un signal en sortie d'un neurone artificiel. Le terme de "fonction d'activation" vient de l'équivalent biologique "potentiel d'activation", seuil de stimulation qui, une fois atteint entraîne une réponse du neurone. La fonction d'activation est souvent une fonction non-linéaire. Un exemple de fonction d'activation est la fonction de Heaviside [8], qui renvoie tout le temps 1 si le signal en entrée est positif, ou 0 s'il est négatif. D'autres exemples classiques de fonctions d'activation sont La fonction sigmoïde [21] et La fonction tangente hyperbolique [26]

### Mode d'apprentissage d'un réseau de neurones

La large majorité des réseaux de neurones possède un algorithme "d'entraînement" qui consiste à modifier les poids synaptiques en fonction d'un jeu de données présentées en entrée du réseau. Le but de cet entraînement est de permettre au réseau de neurones d'"apprendre" à partir des exemples. Si l'entraînement est correctement réalisé, le réseau est capable de fournir des réponses en sortie très proches des valeurs d'origines du jeu de données d'entraînement. Mais tout l'intérêt des réseaux de neurones réside dans leur capacité à généraliser à partir des données de test. Il est donc possible d'utiliser un réseau de neurones pour réaliser une mémoire ; on parle alors de mémoire neuronale.

La vision topologique d'un apprentissage correspond à la détermination de l'hyper surface sur  $\mathbb{R}^n$  où  $\mathbb{R}$  est l'ensemble des réels, et  $n$  le nombre d'entrées du réseau. Un apprentissage peut être supervisé ou non supervisé.

Un apprentissage est dit supervisé lorsque le réseau est forcé à converger vers un état final précis, en même temps qu'un motif lui est présenté.

À l'inverse, lors d'un apprentissage non-supervisé, le réseau est laissé libre de converger vers n'importe quel état final lorsqu'un motif lui est présenté.

### Le surapprentissage ou *overfitting*

Il arrive souvent que les exemples de la base d'apprentissage comportent des valeurs approximatives ou bruitées. Si on oblige le réseau à répondre de façon quasi parfaite relativement à ces exemples, on peut obtenir un réseau qui est biaisé par des valeurs erronées.

Par exemple, imaginons qu'on présente au réseau des couples  $(x_i, f(x_i))$  situés sur une droite d'équation  $y = ax + b$ , mais bruités de sorte que les points ne soient pas exactement sur la droite. S'il y a un bon apprentissage, le réseau répond  $ax + b$  pour toute valeur de  $x$  présentée. S'il y a surapprentissage, le réseau répond un peu plus que  $ax + b$  ou un peu moins, car chaque couple  $(x_i, f(x_i))$  positionné en dehors de la droite va influencer la décision : il aura appris le bruit en plus, ce qui n'est pas souhaitable.

Il existe plusieurs manières de réduire les surapprentissages dans les modèles d'apprentissage en profondeur. La meilleure option est d'obtenir plus de données d'entraînement. Malheureusement, dans des situations réelles, nous n'avons souvent pas cette possibilité en raison de contraintes de temps, de budget ou techniques. Un autre moyen de réduire les surapprentissages consiste à réduire la capacité du modèle à apprendre des données. En tant que tel, le modèle devra se concentrer sur les schémas pertinents dans les données d'entraînement, ce qui aboutira à une meilleure généralisation [25].

### Rétropropagation et Élagage

La rétropropagation consiste à rétropropager l'erreur commise par un neurone à ses synapses et aux neurones qui y sont reliés. Pour les réseaux de neurones, on utilise habituellement la rétropropagation du gradient de l'erreur, qui consiste à corriger les erreurs selon l'importance des éléments qui ont justement participé à la réalisation de ces erreurs : les poids synaptiques qui contribuent à engendrer une erreur importante se verront modifiés de manière plus significative que les poids qui ont engendré une erreur marginale.

L'élagage *pruning*, en anglais) est une méthode qui permet d'éviter le surapprentissage tout en limitant la complexité du modèle. Elle consiste à supprimer des connexions (ou synapses), des entrées ou des neurones du réseau une fois l'apprentissage terminé. En pratique, les éléments qui ont la plus petite influence sur l'erreur de sortie du réseau sont supprimés. Les deux algorithmes d'élagage les plus utilisés sont *Optimal brain damage (OBD)* de Yann LeCun et al. [16] et *Optimal brain surgeon (OBS)* de B. Hassibi et D. G. Stork []

## 1.4 Deep learning et classification d'images

Dans cette partie, nous allons aborder une des applications du *deep learning* qui est la reconnaissance d'images.

### 1.4.1 Introduction

Un modèle de réseaux de neurones profond est capable aujourd'hui de reconnaître chaque élément d'une scène pourvu qu'il ait été entraîné pour cela. À partir

de la sémantique de cette reconnaissance, un réseau de neurones profond peut même générer une légende à cette scène.

Par exemple, aujourd’hui il est assez courant de pouvoir obtenir le résultat suivant automatiquement :

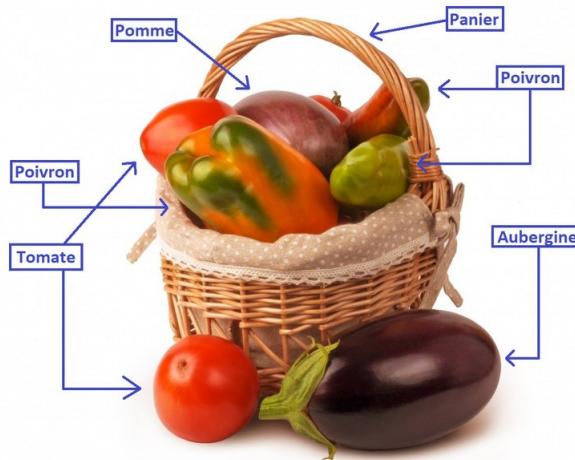


FIGURE 1.6 – Sur cette image, nous pouvons voir ce qu’un modèle de réseau de neurones profond est capable de faire automatiquement.

Une fois que l’ordinateur a identifié tous les éléments de la scène ci-dessus, il sera capable de dire en sortie : "Il y a une corbeille de fruits et légumes sur fond blanc. Dans la corbeille il y a 2 tomates, une aubergine et 3 poivrons". A partir de ces observations, beaucoup de données sont exploitables, le nombre de fruits, le nombre de légumes, la taille du panier etc.

Ce qui est important et révolutionnaire ce n’est pas le fait de savoir combien de tomates il y a dans un panier mais la compréhension d’une scène.

#### 1.4.2 La notion de *features* dans une image

Pour un être humain, une image, qu’elle soit numérique, imprimée ou sur un autre support est perçue de la même façon. Nous sommes sur-entraîné pour cet exercice, nous avons tellement l’habitude de voir et d’interpréter tout ce qui nous entoure que nous avons l’impression que de nos yeux à notre cerveau il n’y a qu’une étape.

Pour un ordinateur, même pour une image, ce qui compte, ce sont les chiffres. Ainsi donc l’ordinateur décompose une image en pixels (*picture element*), c’est la plus petite composante d’une image, c’est un carré unicolore.

Ainsi donc les pixels peuvent très bien servir à décrire une image, mais ne constituent pas la meilleure manière de caractériser une image en *deep learning* pour un modèle de reconnaissance d’images. Ce problème est donc reformulé d’une autre manière : "Quels sont les éléments caractéristiques de l’image 1 ? Les retrouve-t-on dans l’image 2 ?". La stratégie consiste à trouver les éléments communs aux deux images.

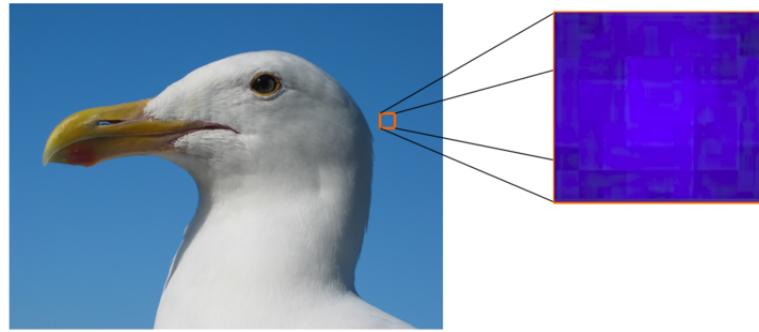


FIGURE 1.7 – Observation des pixels d'une image.

Cette tâche est également plus ou moins triviale pour notre cerveau, mais difficile pour un ordinateur : celui-ci doit parvenir à décrire la particularité d'une classe d'images, et ce en dépit de toutes les variations possibles :

- La prise de vue (résultat d'une transformation affine ou d'une projection)
- L'orientation (résultat d'une rotation)
- L'échelle (résultat d'un zoom)
- Les propriétés photométriques : la luminosité et/ou le contraste (variations dues aux différents périodes de la journée, météo, flash...)
- Occlusion : une partie de l'image est cachée
- Background clutter : une partie de l'image se confond avec les éléments en arrière-plan



FIGURE 1.8 – La Tour Eiffel dans tous ses états.

En vision par ordinateur, le terme *[local] features* désigne des zones intéressantes de l'image numérique. Ces zones peuvent correspondre à des contours, des points ou des régions d'intérêt. A chaque *feature* détecté est associé un vecteur, appelé *descripteur* (*feature descriptor* ou *feature vector*), qui, comme son nom l'indique, décrit la zone concernée.

La résolution du problème d'image matching se fait alors en deux étapes :

1. Déetecter et décrire les caractéristiques (*features* dans chaque image)
2. Trouver les paires de *features* qui se correspondent dans les deux images (*features matching*)

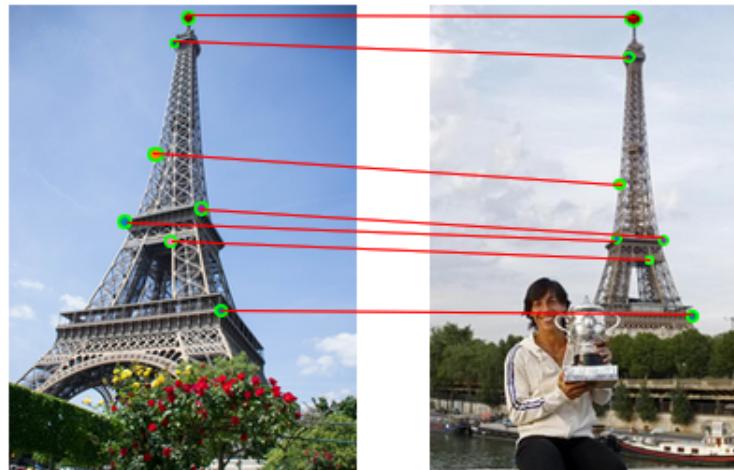


FIGURE 1.9 – Exemple d'image matching. Les features sont repérées par les ronds.

L'algorithme d'*image matching* étudie des images caractérisées par leurs *features*, donc la qualité des résultats dépend (entre autres) de la pertinence des *features* détectées. En ce sens, la première étape est fondamentale et ne doit en aucun cas être négligée.

L'extraction et la description de *features* constituent la première étape indispensable pour de nombreuses autres tâches en vision par ordinateur, comme la classification d'images.

Un mauvais choix de *features* peut entraîner plusieurs difficultés dans l'étape de *features matching* :

**Problème 1 :** deux images n'ont pas les mêmes *features* alors qu'elles représentent la même chose de manières différentes

**Problème 2 :** deux images présentent les mêmes *features*, mais trouver les paires qui se correspondent est très difficile

Ces deux problèmes rendent le *matching* impossible et doivent donc être anticipés dès la première étape, lors de l'extraction et la description de *features*. Cela nous amène à la question suivante : "Quelles *features* faut-il sélectionner ?"

En effet, une zone constitue un bon choix de *features* si elle est :

**Répétable** - une *feature* doit se retrouver dans les images représentant la même scène malgré les différences géométriques et photométriques. Une *feature* doit donc présenter des propriétés d'invariance à ces transformations.

**Distinctive** - une *feature* doit être suffisamment unique et non-ambiguë au sein d'une image pour faciliter le *matching*. Ce sont les informations contenues dans son descripteur qui doivent mettre en valeur sa particularité.

**Locale** - une *feature* doit correspondre à une zone suffisamment petite, et elle est décrite selon son voisinage uniquement. Cela permet d'éviter les difficultés de *matching* dues aux phénomènes d'occlusion et de *background clutter* (une partie de l'image se confond avec les éléments en arrière-plan).

De plus, les *features* détectées doivent être nombreuses, mais l'effectif total doit être strictement inférieur au nombre de pixels de l'image afin que l'algorithme de *matching* soit efficace.

En résumé, une bonne *feature* doit être suffisamment unique pour pouvoir différencier deux classes d'images différentes, et suffisamment générique pour pouvoir reconnaître facilement les images d'une même classe malgré la diversité des représentations.

### 1.4.3 Extraction des *features*

Il existe de nombreuses méthodes pour l'extraction de divers types de *features* :

#### Le gradient d'une image

Le gradient d'une image est un outil très utile pour l'extraction de *features*. Il s'agit d'un vecteur  $\mathbf{I}$  composé des dérivées partielles de la fonction d'intensité, et calculé en chaque pixel :

$$\nabla I(x, y) = \left( \frac{\partial I(x, y)}{\partial x}, \frac{\partial I(x, y)}{\partial y} \right)$$

#### La détection des bords avec le filtre de Canny

Les bords ou contours (*edges* en anglais) fournissent beaucoup d'information à propos d'une image : ils délimitent les objets présents dans la scène représentée, les ombres ou encore les différentes textures.

Un moyen pour détecter les bords est de segmenter l'image en objets, mais il s'agit d'un problème difficile. Le filtre de Canny développé en 1986 est une solution plus simple, qui repose sur l'étude du gradient.

Les bords se situent dans les régions de l'image qui présentent de forts changements. En effet, les contours des objets correspondent à des changements de profondeur (on passe d'un objet à un autre situé en arrière-plan), et les ombres et différentes textures à des changements d'illumination.

Mathématiquement, la détection des bords revient donc à chercher les points de l'image où la fonction d'intensité  $I$  varie brusquement. Or, nous savons qu'une amplitude du gradient élevée indique un fort changement d'intensité. Le but est donc de chercher les maxima locaux de  $\|\nabla I\|$ .

### La localisation des coins avec le détecteur de Harris-Stephens

Les coins (corners en anglais) sont d'autres *features* riches en informations. Ils se situent dans les régions où l'intensité varie fortement dans au moins deux directions :

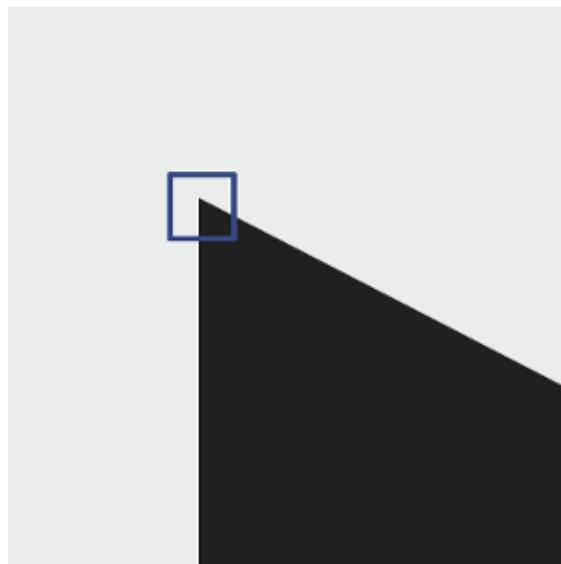


FIGURE 1.10 – Exemple de coin. On voit bien que l'intensité varie brutalement dans plusieurs directions (vers le haut et vers la gauche)

### SIFT

La *scale-invariant feature transform* (SIFT), que l'on peut traduire par "transformation de caractéristiques visuelles invariante à l'échelle", est un algorithme utilisé dans le domaine de la vision par ordinateur pour détecter et identifier les éléments similaires entre différentes images numériques (éléments de paysages, objets, personnes, etc.).

L'étape fondamentale de la méthode consiste à calculer ce que l'on appelle les "descripteurs SIFT" des images à étudier. Il s'agit d'informations numériques dérivées de l'analyse locale d'une image et qui caractérisent le contenu visuel de cette image de la façon la plus indépendante possible de l'échelle ("zoom" et résolution du capteur), du cadrage, de l'angle d'observation et de l'exposition (luminosité). Ainsi, deux photographies d'un même objet auront toutes les chances d'avoir des descripteurs SIFT similaires, et ceci d'autant plus si les instants de prise de vue et les angles de vue sont proches.

## 1.5 Les réseaux de neurones convolutifs (ou CNN)

L'extraction de *features* constitue la première étape pour entraîner un réseau de neurones pouvant reconnaître une image donnée. La précision de ce réseau dépendra de la quantité et de la qualité (les *features* extraits) des données d'entraînement. Dans la partie précédente, nous avons étudié la notion de *features* d'une image et les méthodes utilisées pour les extraire de nos images. Pour une très grande quantité de données, nous serons donc amener à passer un temps considérable dans cette première étape (extraction des *features*) sans oublier les étapes qui suivront.

Mais en 2012, une révolution s'est produite : lors de la compétition annuelle de vision par ordinateur ILSVRC [17], un nouvel algorithme de *deep learning* explose les records [10]. Il s'agit d'un réseau de neurones convolutifs appelé AlexNet [10].

### 1.5.1 Qu'est-ce que la convolution ?

La convolution est un outil mathématique simple qui est très largement utilisé pour le traitement d'images. Ce qui explique que les réseaux de neurones à convolution soient particulièrement bien adaptés à la reconnaissance d'image.

La convolution agit comme un filtrage. On définit une taille de fenêtre qui va se balader à travers toute l'image (rappelons qu'une image peut être vue comme étant un tableau). Au tout début de la convolution, la fenêtre sera positionnée tout en haut à gauche de l'image puis elle va se décaler d'un certain nombre de cases (c'est ce que l'on appelle le pas) vers la droite et lorsqu'elle arrivera au bout de l'image, elle se décalera d'un pas vers le bas ainsi de-suite jusqu'à ce que le filtre est parcourue la totalité de l'image :

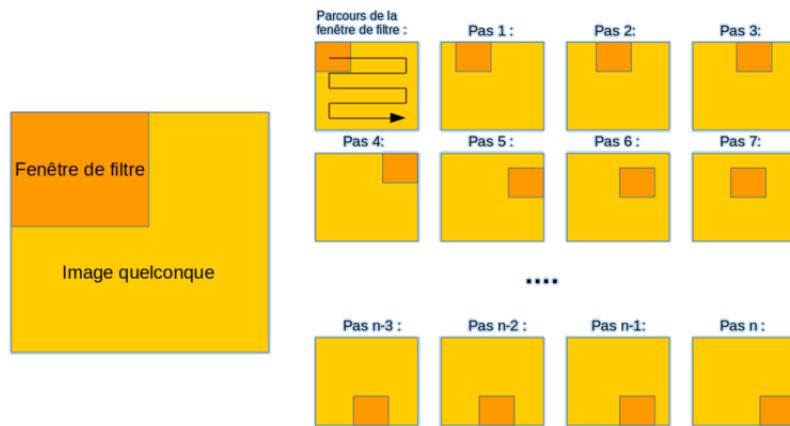


FIGURE 1.11 – Schéma du parcours de la fenêtre de filtre sur l'image.

Le but est de se servir des valeurs présentes dans le filtre à chaque pas. Par exemple si l'on définit une fenêtre 3 par 3, cela représentera 9 cases du tableau (c'est à dire 9 pixels). La convolution va effectuer une opération avec ces 9 pixels. Il peut s'agir de n'importe quelle opération, par exemple on extrait la valeur la plus grande (soit le pixel avec la plus grande valeur).

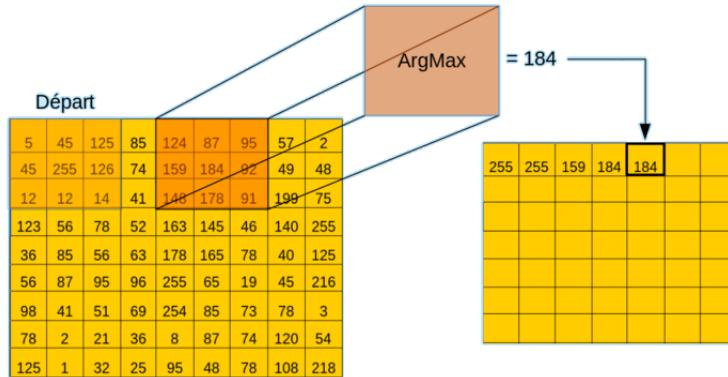


FIGURE 1.12 – Exemple d'une convolution dont la configuration est : Opération = Argument maximale, pas horizontale = 1 pixel, pas vertical = 1 pixel.

On fait glisser la fenêtre en orange et à chaque pas on récupère la valeur la plus grande parmi les 9 valeurs de pixels. On remarque que la sortie de la convolution, que l'on peut appeler "carte de caractéristiques", à des dimensions plus petites que celle de l'image en entrée.

### 1.5.2 Introduction aux réseaux de neurones convolutifs (CNN)

Les réseaux de neurones convolutifs ont une méthodologie similaire à celle des méthodes traditionnelles d'apprentissage supervisé : ils reçoivent des images en entrée, détectent les *features* de chacune d'entre elles, puis entraînent un classifieur dessus.

Cependant, les *features* sont apprises automatiquement. Les CNN réalisent eux-mêmes tout le travail fastidieux d'extraction et de description des *features* : lors de la phase d'entraînement, l'erreur de classification est minimisée afin d'optimiser les paramètres du classifieur. De plus, l'architecture spécifique du réseau permet d'extraire des *features* de différentes complexités, des plus simples au plus sophistiqués. L'extraction et la hiérarchisation automatiques des *features*, qui s'adaptent au problème donné, constituent une des forces des réseaux de neurones convolutifs : plus besoin d'implémenter un algorithme d'extraction "à la main", comme SIFT ou *Harris-Stephens*.

Aujourd'hui, les réseaux de neurones convolutifs, aussi appelés CNN ou ConvNet pour Convolutional Neural Network, sont toujours les modèles les plus performants pour la classification d'images.

### 1.5.3 Architecture d'un CNN

Les réseaux de neurones convolutifs sont directement inspirés du cortex des vertébrés. Un réseau de neurones à convolution se représente en général sous la forme suivante :

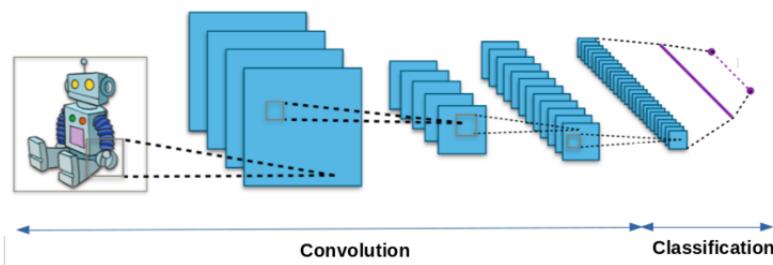


FIGURE 1.13 – Architecture d'un réseau de neurones convolutif.

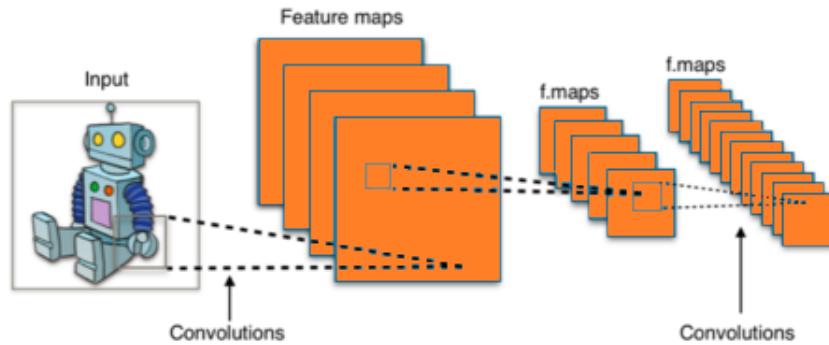
On distingue deux parties, une première partie que l'on appelle la partie convective du modèle et la seconde partie, que l'on va appeler la partie classification du modèle qui correspond à un modèle MLP (Multi Layers Perceptron - vu plus haut -).

#### Description de la partie convective des modèles CNN

Soit l'exemple suivant : imaginons qu'en entrée d'un réseau de neurones, nous avons une image quelconque de 512 pixels de coté. Cette image comporte donc 262 144 pixels ( $512 \times 512$ ). Un réseau de neurones classique (MLP par exemple) capable de reconnaître cette image présentera sur sa première couche cachée 512 neurones. Cela a pour conséquences le fait que cette première couche uniquement aura plus de 13 millions de poids à calculer ( $262\,144 \times 512$ ). Cela qui n'est pas envisageable dans

le sens où nous souhaitons diminuer la puissance de calculs nécessaire à l'utilisation des réseaux de neurones.

Le CNN et en particulier sa partie convulsive permet de r2soudre en partie ce problème en diminuant considérablement le nombre de poids à calculer dans un réseau. En effet, comme nous l'avons vu ci-dessus, la convolution va avoir pour effet de réduire la dimension de "la carte de caractéristiques" que l'on obtient après convolution. Si l'on répète ce processus plusieurs fois, en prenant comme nouvelle entrée la sortie de la convolution précédente, nous réduisons de plus en plus la taille de la "carte de caractéristiques", et donc le nombre de calculs :



---

FIGURE 1.14 – On observe la réduction de la taille de la carte de caractéristiques (*feature maps*)

## Chapitre 2

# Problématiques, motivations et objectifs de notre recherche

Dans ce chapitre, nous allons d'abord énoncer les problématiques liées à notre travail de recherche qui, rappelons-le, porte sur la compression et l'optimisation de modèles de réseaux de neurones pour les terminaux de faibles capacités. En effet, le *deep learning* est l'un des domaines qui a beaucoup évolué ces dernières années et de nos jours, ces produits entrent déjà dans notre vie quotidienne, mais présentent toutefois un certain nombre de limites quant à une utilisation grand public. Ensuite nous allons évoquer les raisons qui nous ont motivé à orienter nos recherches vers ce sujet et enfin terminer en présentant nos objectifs à atteindre.

### Sommaire

---

<b>2.1 Problématique</b>	21
2.1.1 <i>Deep learning</i> et ressources mises en jeu	21
2.1.2 L'exploitation des modèles de réseaux de neurones artificiels	22
<b>2.2 Motivations et objectifs</b>	22
2.2.1 Motivations	22
2.2.2 Objectifs de recherches	23

---

## 2.1 Problématique

Le *deep learning* alimente bon nombre de débats au seins des communautés de chercheurs. Bien que n'étant pas un concept nouveau, le *deep learning* n'est pas à la portée de tous que ce soit pour des recherches ou une utilisation dans des applications au profit de l'individu lambda.

### 2.1.1 *Deep learning* et ressources mises en jeu

Alors que les premières théories concernant le *deep learning* remontent aux années 1980, ce n'est que récemment que le *deep learning* connaît un grand engouement de la part des chercheurs. Les raisons qui pourraient expliquer cela sont par exemple le fait que :

1. Le *deep learning* nécessite un vaste jeu de données qui n'existaient pas auparavant. En effet, ces dernières années, nous assistons à un boom de données numériques [18].
2. Le *deep learning* exige non seulement une forte puissance de calcul mais aussi consomme énormément d'électricité. En témoignent les chiffres de *DeepMind* pour battre un humain au jeu de Go : il s'agit de 1500 CPU, environ 300 GPU, quelques TPU et 440 kWh [6].

A ces raisons s'ajoute la durée d'entraînement de réseaux de *deep learning* qui peut varier entre quelques heures à plusieurs semaines.

Mettre en place un modèle basé sur les réseaux de neurones n'est donc pas chose aisée, tout comme l'exploitation de ce modèle résultant.

### 2.1.2 L'exploitation des modèles de réseaux de neurones artificiels

En se basant toujours sur la reconnaissance d'images, un modèle capable de reconnaître au moins huit milles (8000) espèces différentes de plantes aura une taille considérable (plusieurs centaines de mégaoctets). En marge de la vision par ordinateur, ce problème est présent dans tous les autres champs d'application du *deep learning* et en particulier les réseaux de neurones.

Si l'utilisation d'un tel modèle de plusieurs mégaoctets comme taille dans une application pour un ordinateur est envisageable, cela demeure très difficile, voire impossible sur un smartphone bas/milieu ou même haut de gamme. A cela s'ajoute de sérieux problèmes de performance qui pourraient survenir compte tenu de la puissance de calcul de ces derniers sans oublier les problèmes de compatibilité.

Ces deux derniers points énoncent notre problématique. Ainsi donc, nous nous posons la question suivante : **Quelles techniques pour compresser et optimiser les modèles basés sur des réseaux de neurones pour les terminaux de faibles capacités tout en garantissant des performances comparable à un déploiement sur un ordinateur assez performant ?**

## 2.2 Motivations et objectifs

Faire fonctionner des applications intelligentes sur les terminaux de faibles capacités (mémoire, stockage, puissance de calcul) n'est pas nouveau en soi. En effet, face aux énormes capacités requises par ces applications, il existe des solutions telles que le *Machine Learning Kit for firebase* (Google) et le *Core Machine Learning* (Apple) qui décharge ce besoin en ressources sur le cloud. Ces solutions sont fournis sous forme de services Cloud (Le *cloud computing* consiste en l'utilisation de la puissance de calcul ou de stockage de serveurs informatiques distants par l'intermédiaire d'un réseau, généralement internet.). Cette solution n'est donc faite pour tout le monde particulièrement dans les zones avec des difficulté d'accès à internet (manque d'infrastructures, faible couverture et de débit, coût d'accès, ... ). Face, à ces difficultés de pouvoir déployer localement des applications basées sur des réseaux de neurones artificiels et plus généralement l'intelligence artificielle, nous présentons ici les motivations et les objectifs de notre travail.

### 2.2.1 Motivations

La prolifération des smartphones et systèmes embarqués en Afrique fait de ces derniers les seules ressources de calcul accessibles au grand public. Ce qui pourrait constituer un marché potentiel si nous arrivons à favoriser une tout autre utilisation de ces smartphones grâce à des applications basées sur le *deep learning*. Les champs d'applications possibles grâce aux réseaux de neurones artificiels sont vastes et diversifiés. Un support local de modèles de réseaux de neurones serait d'une grande utilité et indispensable dans le sens où les utilisateurs demandent des applications de plus en plus complexes.

### 2.2.2 Objectifs de recherches

Les objectifs de notre thème de recherche qui s'intitule "Optimisation et compression de modèles de réseaux de neurones artificiels : application à la reconnaissance d'images de plantes sur terminaux de faibles capacités" s'articulent comme suit :

- d'une part, apporter une optimisation et une compression des réseaux de neurones afin de garantir leur utilisation efficiente dans des applications pour terminaux de faibles capacités ;
- d'autres part, tester notre approche sur un cas pratique qui est celui de la reconnaissance d'images de plantes sur un terminal de faibles capacités, plus précisément tournant sur le système d'exploitation mobile Android.

Mais avant de proposer une approche pour parvenir à ces résultats, nous allons dans le chapitre suivant (Chapitre 3) faire le point sur les différentes approches existantes et celles ayant un lien avec notre sujet.

## Chapitre 3

# Etat de l'art

Dans cette partie, nous allons faire le point sur l'état des connaissances, les différents acquis et travaux en cours concernant l'optimisation et la compression de modèles de réseaux de neurones.

### Sommaire

<b>3.1</b>	<b>Les méthodes d'optimisation de modèle de <i>deep learning</i> . . . . .</b>	<b>24</b>
3.1.1	La régression linéaire univariée en rappel . . . . .	24
3.1.2	La descente du gradient . . . . .	26
3.1.3	La descente de gradient par mini-lot . . . . .	27
3.1.4	Momentum . . . . .	28
<b>3.2</b>	<b>Les travaux de recherches en optimisation de modèle de <i>deep learning</i> . . . . .</b>	<b>28</b>
<b>3.3</b>	<b>La compression de modèles de <i>deep learning</i> . . . . .</b>	<b>29</b>
3.3.1	L'élagage des réseaux de neurones profonds pour les rendre rapides et petits . . . . .	29
3.3.2	Le partage de poids : le partage pondéral . . . . .	30
3.3.3	Le codage de Huffman . . . . .	32
3.3.4	Les travaux de recherche sur la compression de modèles de <i>deep learning</i> . . . . .	33

### 3.1 Les méthodes d'optimisation de modèle de *deep learning*

La descente du gradient est la méthode traditionnelle utilisée pour mettre à jour les paramètres et minimiser les coûts dans bon nombre de problèmes d'optimisation. Dans cette partie, nous allons montrer d'autres méthodes d'optimisation plus avancées qui peuvent accélérer l'apprentissage et pouvant dans certains cas conduire à de meilleurs résultats. Avoir un bon algorithme d'optimisation peut faire la différence entre des jours d'attente et quelques heures seulement pour obtenir un bon modèle de *deep learning*. En *deep learning*, l'ensemble de ses algorithmes vise à mettre en place des modèles performants, peu gourmands en ressources CPU.

#### 3.1.1 La régression linéaire univariée en rappel

La régression linéaire univariée est un algorithme prédictif supervisé. Il prend en entrée une variable prédictive  $x$  et va essayer de trouver une fonction de prédiction  $h(x)$ . Cette fonction sera une droite qui s'approchera le plus possible des données d'apprentissage. La fonction de prédiction étant une droite, elle s'écrira mathématiquement sous la forme :

$$h(x) = \theta_0 + \theta_1 \cdot x$$

Avec  $\theta_0$  et  $\theta_1$  sont les coefficients de la droite.

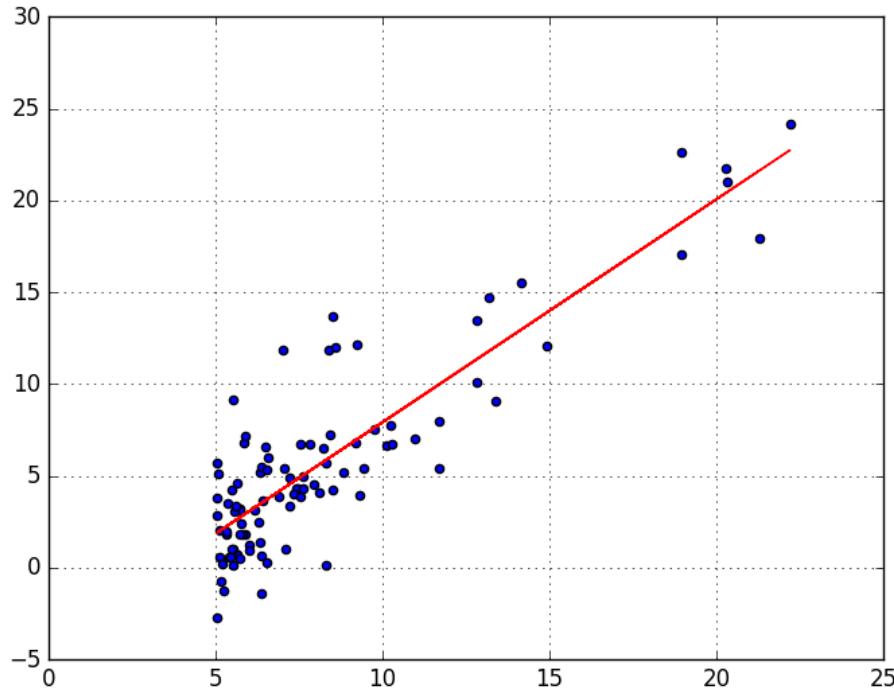


FIGURE 3.1 – La droite en rouge représente la meilleure approximation par rapport au nuage de points bleus. Cette approximation est rendue possible par ce qu'on a pu calculer les paramètres prédictifs  $\theta_0$  et  $\theta_1$  qui définissent notre droite rouge.

La figure ci-dessus montre que la droite en rouge tente d'approcher le plus de points possibles (en réduisant l'écart avec ces derniers). En d'autres termes, elle minimise au maximum l'erreur globale. Le but du jeu revient à trouver un couple  $(\theta_0, \theta_1)$  optimal tel que  $h(x)$  soit le plus proche possible de  $y$  (la valeur qu'on essaie de prédire). Et ce, pour tous les couples  $(x, y)$  qui forment notre ensemble de données d'apprentissage.

**Note 1** : pensons à  $h(x_i)$  comme un imitateur de  $y_i$ . La fonction  $h$  va essayer de transformer au mieux  $x_i$  en  $y_i$  tel que  $h(x_i) \approx y_i$ .

**Note 2** : on définit “l'erreur unitaire” entre une valeur observée  $y_i$  et une valeur prédictée  $h(x_i)$ , comme suit :

$$(h(x_i) - y_i)^2$$

Trouver le meilleur couple  $(\theta_0, \theta_1)$  revient à minimiser le coût global des erreurs unitaires qui se définit comme suit :

$$\sum_{i=0}^m (h(x_i) - y_i)^2$$

$m$  est la taille des données d'entraînement. La fonction de coût est définie comme suit :

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=0}^m (h(x_i) - y_i)^2$$

En remplaçant le terme  $h(x)$  par sa valeur on obtient :

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=0}^m (\theta_0 + \theta_1 x_i - y_i)^2$$

Cette formule représente la fonction de coût (*cost function / Error function*) pour la régression linéaire univariée.

### 3.1.2 La descente du gradient

Une méthode d'optimisation simple dans l'apprentissage automatique est la descente du gradient (GD). Mathématiquement, il s'agit d'un algorithme d'optimisation itératif du premier ordre pour trouver les meilleures paramètres  $\theta_0$  et  $\theta_1$  : minimiser (trouver le minimum) la fonction du coût.

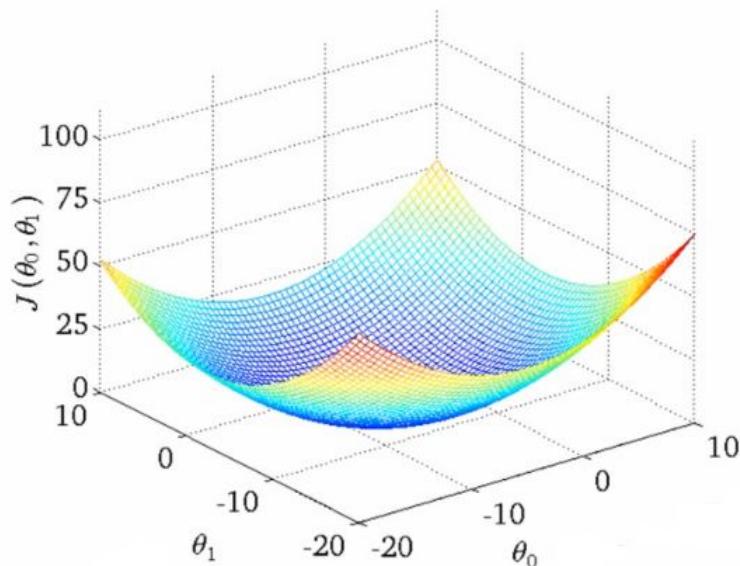


FIGURE 3.2 – Visualisation de la descente du gradient

Visuellement, on remarque que la fonction  $J(\theta_0, \theta_1)$  a la forme d'un bol. Mathématiquement, on dit que la fonction est convexe. La convexité d'une fonction implique que cette dernière possède un seul minimum global. Les valeurs de  $\theta_0$  et  $\theta_1$  qui sont au minimum global de  $J(\theta_0, \theta_1)$  seront les meilleures valeurs. Une façon de calculer le minimum de la fonction de coût  $J(\theta_0, \theta_1)$  est d'utiliser l'algorithme de la descente du gradient (Gradient descent) :

**Algorithm 1:** Algorithme de la descente du gradient

---

Début de l'algorithme : Gradient Descent initialiser aléatoirement les valeurs de  $\theta_0$  et  $\theta_1$ ;

répéter jusqu'à convergence au minimum global de la fonction coût

pour  $j \in \mathbb{N} \wedge \forall j \in 0, 1$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\lambda}{\lambda \theta_j} J(\theta_0, \theta_1)$$

réturner  $\theta_0$  et  $\theta_1$ ;

Fin algorithme

---

L'algorithme peut sembler compliqué à comprendre, mais l'intuition derrière est assez simple :

Imaginez que vous trouviez sur une colline, et que vous souhaitez la descendre (figure 3.3). A chaque nouveau pas (analogie à l'itération), vous regardez autour de vous pour trouver la meilleure pente pour avancer vers le bas. Une fois la pente trouvée, vous avancez d'un pas d'une grandeur  $\alpha$ .




---

FIGURE 3.3 – Illustration de la descente du gradient

Dans la définition de l'algorithme nous remarquons ces deux termes :

- le terme  $\alpha$  s'appelle le *Learning Rate* : il fixe la "grandeur" du pas de chaque itération de la descente du gradient.
- $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$  : Ce terme est la dérivée partielle pour chacun des termes  $\theta_0$  et  $\theta_1$ .

### 3.1.3 La descente de gradient par mini-lot

La descente de gradient par mini-lot est une variante de l'algorithme de descente de gradient qui divise le jeu de données d'apprentissage en petits lots utilisés pour calculer l'erreur du modèle et mettre à jour les coefficients du modèle.

Les mises en œuvre peuvent choisir de faire la somme du gradient sur le mini-lot ou de prendre la moyenne du gradient, ce qui réduit encore la variance du gradient.

La descente en gradient par mini-lots cherche à trouver un équilibre entre la robustesse et l'efficacité de la descente en gradient. C'est la mise en œuvre la plus courante de la descente en gradient utilisée dans le domaine de l'apprentissage en profondeur.

## Avantages

- La fréquence de mise à jour du modèle est supérieure à la descente de gradient, ce qui permet une convergence plus rapidement vers les minimas locaux.
- Les mises à jour par lots fournissent un processus de calcul plus efficace que la descente de gradient.
- Le traitement par lots permet à la fois de ne pas avoir toutes les données d'entraînement dans la mémoire.

## Inconvénients

- La descente du gradient par mini-lot nécessite la configuration d'un hyperparamètre *mini-batch size* supplémentaire pour l'algorithme d'apprentissage : la taille des mini-lots.
- Les informations d'erreur doivent être accumulées sur des mini-lots d'exemples de formation tels que la descente par gradient de lots.

### 3.1.4 Momentum

Étant donné que la descente de gradient en mini-lot effectue une mise à jour des paramètres après avoir uniquement vu un sous-ensemble d'exemples, la direction de la mise à jour présente une certaine variance. L'utilisation du moment peut réduire ces oscillations.

Momentum prend en compte les dégradés passés pour aplani la mise à jour. Nous allons stocker la "direction" des dégradés précédents dans la variable  $v$ . Formellement, ce sera la moyenne pondérée exponentiellement du dégradé aux étapes précédentes. La variable  $v$  pouvant être considérée comme la "vitesse" d'une balle en descente, augmentant la vitesse (et la quantité de mouvement) en fonction de la direction de la pente.

## 3.2 Les travaux de recherches en optimisation de modèle de *deep learning*

Une application réussie de l'apprentissage en profondeur dépend de la définition appropriée de paramètres pour obtenir des résultats de haute qualité en moins de temps. Le nombre de couches cachées et le nombre de neurones dans chaque couche d'un réseau de neurones sont deux paramètres clés qui ont une influence majeure sur les performances de l'algorithme. Les paramétrages manuels avec les méthodes définies plus haut facilitent quelque peu les tâches des utilisateurs dans la définition de ces paramètres importants. Néanmoins, cela peut prendre beaucoup de temps. La majorité des travaux de recherches pour l'optimisation des modèles de *deep learning* se concentre sur l'automatisation de ces tâches. Parmi ces travaux, nous pouvons citer :

- *Parameters Optimization of Deep Learning Models using Particle Swarm Optimization* [22] : En 2017, Basheer Qolomany et Al.[22] nous montrent que la technique d'optimisation des essaims de particules (PSO) offre un grand potentiel pour optimiser les réglages des paramètres et économise ainsi de précieuses

ressources informatiques lors du processus de réglage des modèles d'apprentissage en profondeur. Cette technique se fonde sur un algorithme simulant le déplacement des abeilles.

- *Adam : A Method for Stochastic Optimization*[15] : Proposé par Diederik P. Kingma et Jimmy Ba, Adam [15] (*Adaptive Moment Estimation*) est l'un des algorithmes d'optimisation de *deep learning* qui fonctionnent bien dans un large éventail d'architectures. Il est recommandé par de nombreux experts en algorithmes de réseaux neuronaux bien connus. L'algorithme d'optimisation d'Adam est une combinaison de descente de gradient avec les algorithmes de momentum de recherche opérationnelle (plus précisément, en optimisation).
- *TVM : An Automatic End-to-End Optimization Compiler for Deep Learning* [2] : Toujours dans le besoin croissant d'apporter l'apprentissage machine à une grande diversité de périphériques matériels, Tianqi Chen et Al. du groupe de recherche interdisciplinaire sur l'apprentissage automatique SAMPL mettent en place TVM.ai (figure 3.4). Il s'agit d'un compilateur d'optimisation de bout en bout pour la mise en place de modèles de *deep learning*. Cette approche présente des optimisations au niveau des graphes et au niveau de l'opérateur pour fournir une portabilité des performances aux charges de travail de *deep learning* sur divers composants matériels. TVM.ai résout les problèmes d'optimisation spécifiques au *deep learning*, tels que la fusion d'opérateurs de haut niveau, le mappage à des primitives matérielles arbitraires et le masquage de la latence de la mémoire. TVM offre également une optimisation automatisée des programmes de bas niveau aux caractéristiques matérielles en utilisant une nouvelle méthode de modélisation des coûts basée sur l'apprentissage pour l'exploration rapide des optimisations de code. Les résultats expérimentaux démontrent que TVM offre des performances sur les composants matériels qui sont compétitifs par rapport aux bibliothèques optimisées à la main pour les processeurs à faible consommation, les GPU mobiles et les GPU de classe serveur. Ainsi donc, seuls les processeurs et GPUs les plus récents présentent une meilleure compatibilité. Le système est en open source et en production dans plusieurs grandes entreprises. La taille des applications demeure également grande avec une consommation énergétique élevée lors de nos tests.

### 3.3 La compression de modèles de *deep learning*

La compression des réseaux de neurones profonds pour les représentations de fonctionnalités compactes efficaces en termes de mémoire et de calcul devient un problème critique, en particulier pour le déploiement des réseaux de neurones sur les plateformes à ressources limitées. En *deep learning*, la compression tient habituellement en un pipeline en trois étapes [9] : élagage, partage de poids et codage de Huffman [24], qui travaillent ensemble pour réduire les besoins de stockage pour nos modèles de réseaux de neurones.

#### 3.3.1 L'élagage des réseaux de neurones profonds pour les rendre rapides et petits

L'élagage des réseaux neuronaux est une ancienne idée qui remonte à 1990 (avec le travail optimal de Yan Lecun sur les lésions cérébrales) et avant. L'idée est que

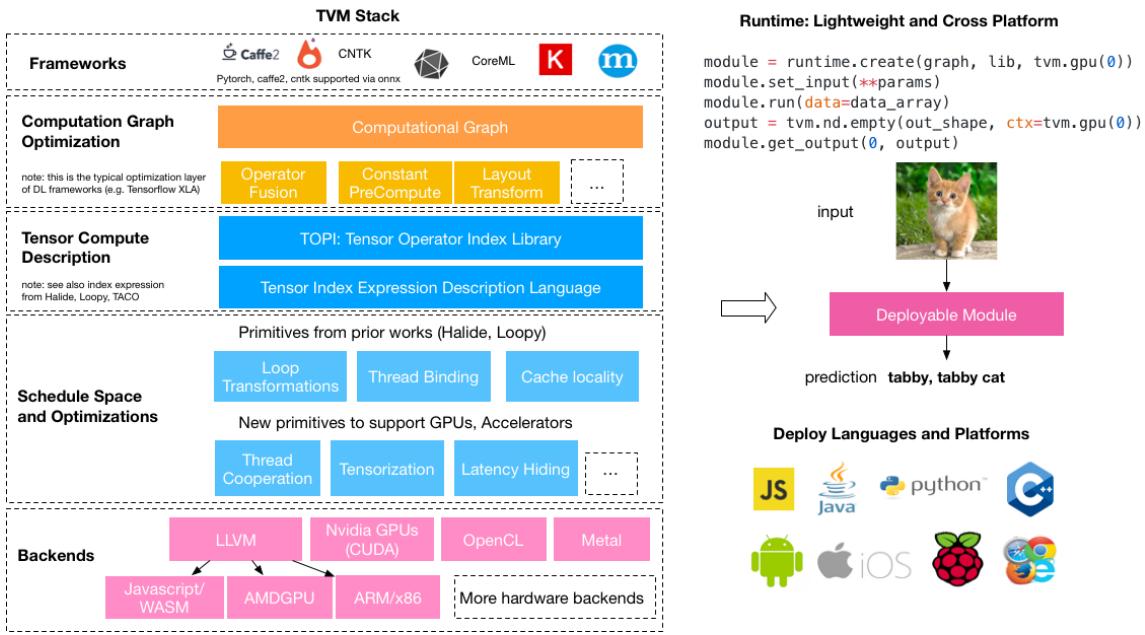


FIGURE 3.4 – Architecture TVM.ai).

parmi les nombreux paramètres du réseau, certains sont redondants et ne contribuent pas beaucoup à la production.

Si vous pouviez classer les neurones du réseau en fonction de leur contribution, vous pourriez alors supprimer les neurones de rang inférieur du réseau, ce qui entraînerait un réseau plus petit et plus rapide. Obtenir des réseaux plus rapides / plus petits est important pour exécuter ces réseaux d'apprentissage approfondi sur des appareils mobiles.

Le classement peut être effectué en fonction de la moyenne des poids des neurones, de leurs activations moyennes, du nombre de fois où un neurone n'est pas nul sur certains ensembles de validation et d'autres méthodes créatives. Après la taille, la précision diminuera (heureusement pas trop si le classement est malin), et le réseau est généralement mieux entraîné pour récupérer. Si nous taillons trop en même temps, le réseau risque d'être endommagé à un point tel qu'il ne pourra pas se rétablir.

En pratique, il s'agit donc d'un processus itératif - souvent appelé "taille itérative" : élagage / train / répétition.

### 3.3.2 Le partage de poids : le partage pondéral

Il s'agit d'une autre façon d'appréhender une notion bien connue en programmation (langage C notamment) : les pointeurs. Le partage pondéral se veut simple : utilisez le même vecteur de poids pour effectuer nos convolutions.

Ainsi, le partage du poids comprime davantage le réseau élagué en réduisant le nombre de bits requis pour représenter chaque poids.

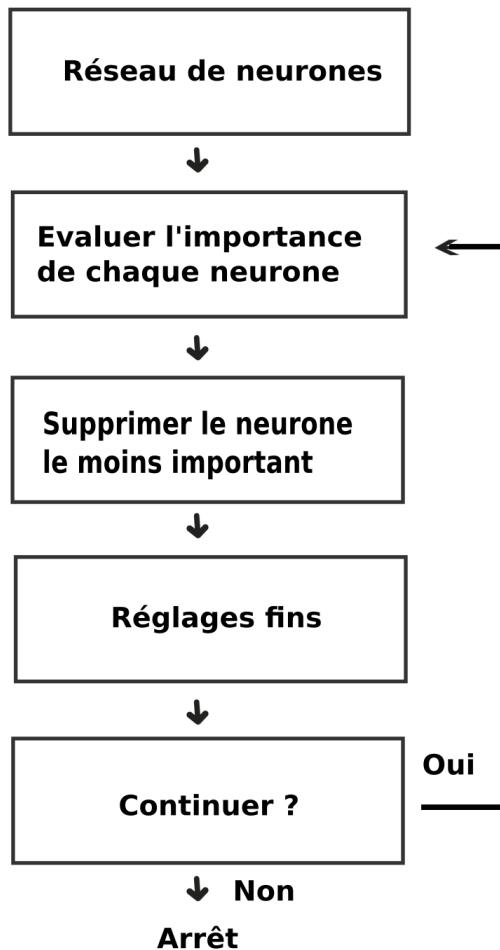


FIGURE 3.5 – Principe d'élagage

Suivant l'approche proposée par Song Han et Al, cela consiste en termes simples, à utiliser le regroupement de k-means [qui vise à partitionner n observations en k grappes dans lesquelles chaque observation appartient au cluster avec la moyenne la plus proche] pour identifier les poids partagés pour chaque couche d'un réseau formé, de sorte que tous les poids qui entrent dans le même cluster partagent le même poids, la même valeur en introduisant la notion de pointeur. Cela évite d'avoir en mémoire des valeurs identiques. Par exemple si cent (100) entrées d'un neurone ont la même valeur  $x$  comme poids, au lieu d'avoir en mémoire la valeur  $x$  pour chaque entrée (ce qui équivaut à une occupation mémoire de  $100x$  (*taille de x*))), cette méthode permet de garder en mémoire une seule valeur de  $x$  et faire pointer cette valeur à chacune des cent (100) entrées.

En nous référant à l'illustration ci-dessus nous avons :

- la couche d'entrée est  $x = [x_1, x_2, x_3, x_4, x_5, x_6, x_7]$
- la couche cachée est  $h = [h_1, h_2, h_3]$
- le vecteur de poids est  $w = [w_1, w_2, w_3] = [1 \ 0 \ -1]$  qui est utilisé par tous (c'est à dire partagés pour le calcul de  $h_1, h_2, h_3$ )

Ainsi donc  $h_1 = w.x[1 : 3]; h_2 = w.x[3 : 5]; h_3 = w.x[5 : 7]$ .

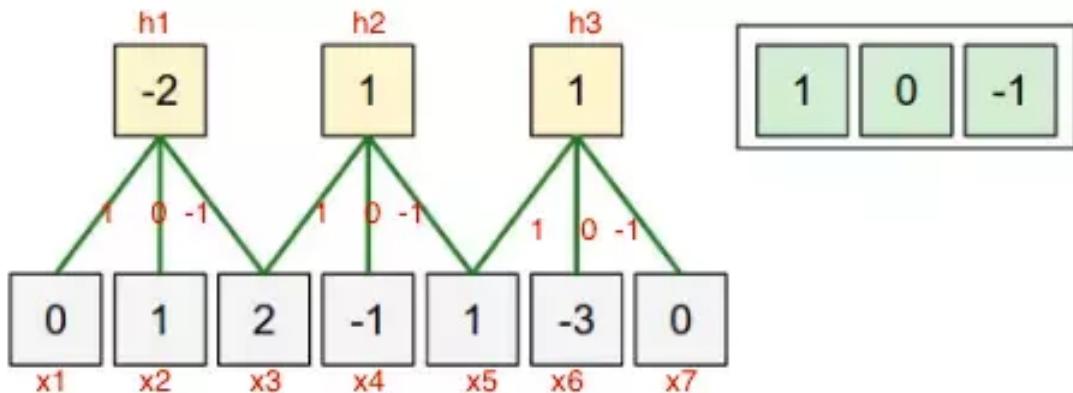


FIGURE 3.6 – Notion de partage pondéral

### 3.3.3 Le codage de Huffman

Un code Huffman est un code de préfixe optimal couramment utilisé pour la compression de données sans perte (Van Leeuwen, 1976). Il utilise des mots de code de longueur variable pour coder les symboles source. Le tableau est dérivé de la probabilité d'occurrence pour chaque symbole. Les symboles les plus courants sont représentés avec moins de bits.

#### Fonctionnement

Soit la phrase suivante : "COMME\_CHARMANT\_E". Les fréquences d'apparitions des lettres sont les suivants :

TABLE 3.1 – Codage de Huffman - Occurrence des lettres

M	A	C	E	_	H	O	N	T	R
3	2	2	2	2	1	1	1	1	1

Cela peut être représenté par l'arbre suivant :

Les codes correspondants à chaque caractère sont tels que les codes des caractères les plus fréquents soient courts et ceux correspondant aux symboles les moins fréquents soient longs :

TABLE 3.2 – Codage de Huffman - Codes de correspondance

M	A	C	E	_	H	O	N	T	R
00	100	110	010	011	1110	1111	1010	10110	10111

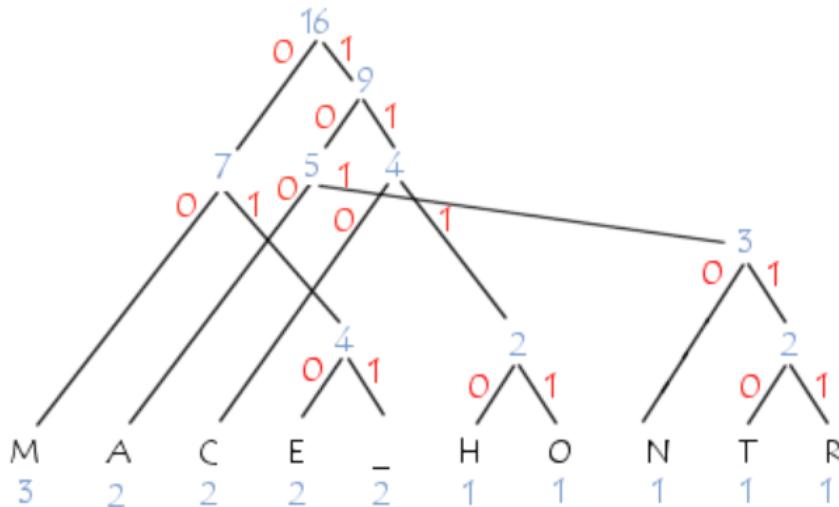


FIGURE 3.7 – Arbre

### 3.3.4 Les travaux de recherche sur la compression de modèles de deep learning

En plus des méthodes, nous présentons ici trois (03) publications ayant rétenu notre attention en matière de compression de modèles de *deep learning*.

- "Pruning Convolutional Neural Networks for Resource Efficient Inference" [20] : Dans ce papier, Pavlo Molchanov et Al. traite le problème d'élagage des réseaux de neurones comme un problème d'optimisation combinatoire. Cela consiste donc à choisir un sous-ensemble de pondérations B, dans le but d'introduire uniquement des modifications mineures dans le réseau de la fonction de coût du réseau.
- "Universal Deep Neural Network Compression" [3] : Yoojin Choi et Al. [3] étudient dans cet article la compression avec perte des réseaux de neurones par la quantification du poids et le codage source sans perte pour une inférence efficace en mémoire. Introduisant ainsi pour la première fois la compression universelle de réseau de neurones par quantification vectorielle universelle et par codage source universel.
- "Deep compression : compressing deep neural Networks with pruning, trained quantization And huffman coding" [9] : Song Han et Al. [9] introduisent en 2016 la notion de "compression profonde". Il s'agit d'un procédé en trois étapes : élagage, partage de poids et codage de Huffman, qui travaillent ensemble pour réduire la nécessité de stockage de réseaux de neurones. Il s'agit d'une approche sans perte de précision sur les modèles de *deep learning*.

Ces approches visent pour la plupart une compression sans perte. Ainsi, nous sommes contraints avec ces solutions à faire un choix entre un taux de compression faible ou le risque de *overfitting* (surapprentissage) du modèle résultant. Il apparaît donc nécessaire d'aller vers une autre approche qui garantisse un réseau de neurones fiable et pouvant être utilisé sur les terminaux de faibles capacités.

## Chapitre 4

# Approche

Dans ce chapitre, nous parlerons d'abord de l'approche retenue pour l'optimisation des réseaux de neurones, puis notre proposition pour une compression efficace des réseaux de neurones pour en faciliter le déploiement sur des terminaux de faibles capacités. Etant parti du principe de pouvoir proposer, une nouvelle approche pour l'optimisation de l'entraînement des réseaux de neurones, nous nous sommes résolus à faire plutôt un choix. En effet, de par la communauté et nos différentes travaux, Adam demeure à l'heure actuelle l'un des algorithmes d'optimisation les plus efficaces pour l'entraînement des réseaux neuronaux. C'est ainsi qu'à travers l'étude comparative conduite par Diederik et Al dans leur publication [15], nous retenons Adam comme méthode d'optimisation. Pour la compression, nous proposons notre propre approche.

## Sommaire

---

<b>4.1 La méthode Adam pour l'optimisation de modèle de <i>deep learning</i></b>	<b>34</b>
4.1.1 Le choix de Adam . . . . .	34
4.1.2 Fonctionnement et points clés de Adam . . . . .	35
<b>4.2 Elagage massif et progressif, partage pondéral et codage de Huffman</b>	<b>35</b>
4.2.1 Présentation . . . . .	36
4.2.2 Algorithme et équations . . . . .	37

---

## 4.1 La méthode Adam pour l'optimisation de modèle de *deep learning*

Nous avons fait cas de la méthode Adam dans l'état de l'art (chapitre 3). Mais, concrètement qu'est-ce qui place Adam au-dessus des autres méthodes d'optimisation ?

### 4.1.1 Le choix de Adam

Nous confirmons le choix de Adam par une étude comparative basée sur une étude empirique et des simulations (figure 4.1) regroupant différentes méthodes d'optimisation. Un des résultats qui a majoritairement impacté notre choix est donné par cette (figure figure4.1) extraite de l'article "ADAM : A METHOD FOR STOCHASTIC OPTIMIZATION"[15] :

Pour un même problème d'optimisation, sur la même configuration (même taux d'apprentissage et les mêmes poids initiaux), Adam a tendance à faire converger l'erreur vers zéro beaucoup plus rapidement.

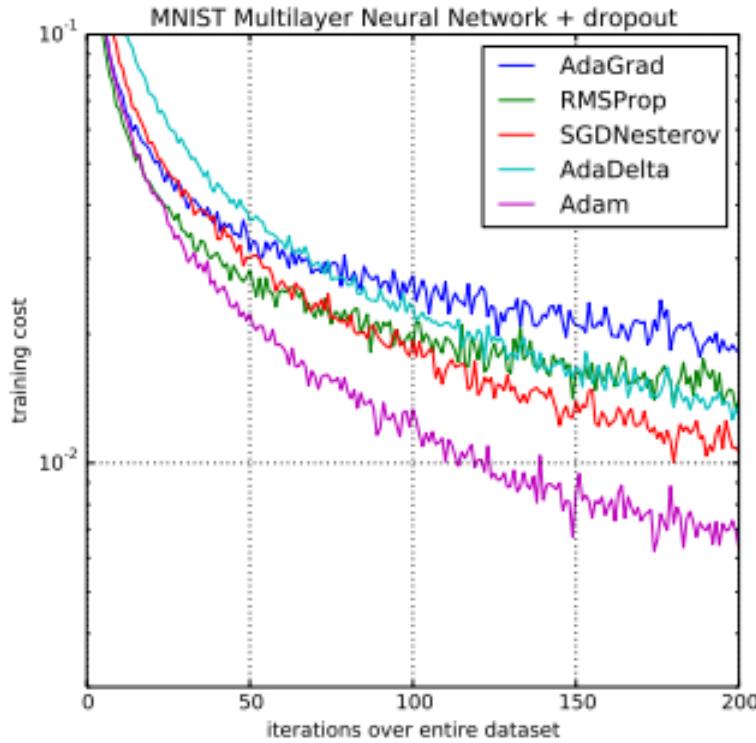


FIGURE 4.1 – Entrainement d'un réseaux de neurones multicouches sur des images de chiffres écrits à la main (MNIST) (extrait de l'article de Diederik P. Kingma et Al. [15])

#### 4.1.2 Fonctionnement et points clés de Adam

Les points clés de cet algorithme sont les suivants :

1. Calcul de la moyenne exponentiellement pondérée des dégradés passés et la stocké dans les variables  $v$  (avant correction du biais) et  $v^{corrected}$  (avec correction du biais).
2. Calcul de la moyenne pondérée exponentiellement des carrés des dégradés passés et la stocké dans les variables  $s$  (avant correction du biais) et  $s^{corrected}$  (avec correction du biais).
3. Mise à jour des paramètres dans une direction basée sur la combinaison des informations de "1" et "2".

## 4.2 Elagage massif et progressif, partage pondéral et codage de Huffman

Nous proposons dans cette partie notre approche pour la compression de réseau de neurones pour les terminaux de faibles capacités. Réposant sur la pipeline en trois étapes [9], la nouveauté de notre approche réside dans l'étape 1 : l'élagage.

---

**Algorithm 1:** Adam, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation.  $g_t^2$  indicates the elementwise square  $g_t \odot g_t$ . Good default settings for the tested machine learning problems are  $\alpha = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 10^{-8}$ . All operations on vectors are element-wise. With  $\beta_1^t$  and  $\beta_2^t$  we denote  $\beta_1$  and  $\beta_2$  to the power  $t$ .

---

```

Require:  $\alpha$ : Stepsize
Require:  $\beta_1, \beta_2 \in [0, 1]$ : Exponential decay rates for the moment estimates
Require:  $f(\theta)$ : Stochastic objective function with parameters  $\theta$ 
Require:  $\theta_0$ : Initial parameter vector
 $m_0 \leftarrow 0$  (Initialize 1st moment vector)
 $v_0 \leftarrow 0$  (Initialize 2nd moment vector)
 $t \leftarrow 0$  (Initialize timestep)
while  $\theta_t$  not converged do
     $t \leftarrow t + 1$ 
     $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )
     $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)
     $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)
     $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)
     $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)
     $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)
end while
return  $\theta_t$  (Resulting parameters)

```

---

FIGURE 4.2 – Algorithme de Adam tel que présenté dans l'article de Diederik P. Kingma et Jimmy Lei Ba

### 4.2.1 Présentation

Notre approche vise à supprimer progressivement certains neurones de notre réseau. L'idée est que s'il est possible d'évaluer la contribution de chaque neurone dans une prise de décision, nous pouvons éliminer ceux qui sont neutres ou qui ont un impact minimal sur notre réseau. Le résultat sera ainsi un réseau plus petit et plus rapide. L'évaluation peut être faite sur le poids des neurones, leurs activations moyennes, le nombre de fois qu'un neurone n'est pas égal à zéro sur un ensemble donné lors de la validation. Toutefois, si nous supprimons un grand nombre de neurones à la fois, le réseau peut être tellement "endommagé". En pratique, il s'agit donc d'un processus itératif - souvent appelé "élagage itératif" : élagage / entraînement / répétition [9].

Pour éviter d'endommager notre réseau, notre approche vise à assurer en sortie un modèle de taille minimale et avec une précision acceptable. Ainsi, malgré les risques d'endommagement de notre réseau, nous effectuons un nombre maximal d'itérations permettant d'enlever le maximum de neurones. À partir d'un seuil de précision défini dès le début, nous prévoyons par ces itérations la création d'un réseau offrant le meilleur rapport taux de compression précision.

Pour atteindre ces résultats, nous procédons par deux séries d'élagages :

- le premier est inspiré du travail de Pavlo Molchanov et al. [20] et consiste à évaluer l'importance de chaque neurone et à supprimer le moins important de manière itérative [20].
- le second suit la première étape et consiste en un élagage progressif en évaluant l'impact de la suppression d'un neurone du réseau sur la précision lors de tests avec des données externes.

Théoriquement, cette approche consiste donc en une itération d'opérations d'évaluation de l'impact de la suppression d'un neurone du réseau sur la précision du réseau résultant lors de tests avec des données externes. La précision issue de ces tests nous permet soit de restaurer le neurone ; soit le neurone demeure supprimé. Ainsi, nous faisons converger le réseau vers un réseau de taille réduite tout en garantissant une précision acceptable.

#### 4.2.2 Algorithme et équations

---

##### Algorithm 2: Elagage progressif et massif

---

**Entrée :**  $a$ , seuil de précision minimale pour la validation

**Entrée :**  $b$ , seuil de précision minimale pour le test

**Entrée :**  $T$ , nouvelles données pour le test

**Sortie :**  $resultat$

```

reseau_initial ← entrainement();
p ← valeur_precision_validation(reseau);
if  $a \geq p$  then
    reseau ← reseau_initial;
    l ← valeur_precision_test(reseau, T);
    while  $b \geq l$  do
        sauvegarder(reseau);
        reseau ← elagage(reseau);
        l ← valeur_precision_test(reseau, T);
        p ← valeur_precision_validation(reseau);
        if  $a < p$  then
            reseau ← restaurer(reseau);
        nouveau ← restaurer(reseau);
        details.taille ← taille_de(nouveau_reseau);
        details.taux_compression ← (1 -
            taille_de(nouveau_reseau)/taille_de(reseau_initial));
        details.precision_test ← valeur_precision_test(nouveau_reseau, T);
        resultat.reseau ← nouveau;
        resultat.details ← details;
return resultat;
```

---

Nous avons utilisé les formules standard suivantes pour évaluer la précision et le taux de compression :

$$PRECISION = \frac{NOMBRE\ DE\ PREDICTIONS\ CORRECTES}{TOTAL\ DE\ PREDICTION} \quad (4.1)$$

$$TAUX\ COMPRESSION = 1 - \frac{TAILLE\ NOUVEAU\ RESEAU}{TAILLE\ RESEAU\ INITIAL} \quad (4.2)$$

## Chapitre 5

# Réalisations

Dans ce chapitre nous allons aborder les grandes étapes de la réalisation de notre solution en commençant par le jeu de données et les outils nécessaires. Pour rappel, nous mettrons en place notre solution sur la base d'une application de reconnaissance d'images de plantes. Le choix de la reconnaissance d'images de plante s'est fait suivant la disponibilité du jeu de données. Ainsi, nous aurions tout aussi bien pu le faire dans tout autre domaine.

### Sommaire

---

<b>5.1</b>	<b>Constitution du jeu de données</b>	38
5.1.1	La compétition " <i>iNaturalist</i> "	38
5.1.2	<i>ImageNet</i>	39
<b>5.2</b>	<b>Les outils</b>	39
5.2.1	Les outils provenant d'éditeurs tierces	39
5.2.2	Outils développés dans le cadre de notre travail	40
<b>5.3</b>	<b>Démarche</b>	40
5.3.1	Entrainement du modèle	41
5.3.2	Compression : études expérimentales et résultats	41
5.3.3	Conversions du modèle	44
5.3.4	Application mobile de reconnaissance d'images de plantes sur Android	45

---

### 5.1 Constitution du jeu de données

Constitué de plus d'un million d'images pour près de quatre milles (4000) espèces de plantes, notre jeu de données (en anglais *dataset*) provient de deux sources qui sont la compétition '*iNaturalist*' 2018 et la banque d'images, *ImageNet*.

#### 5.1.1 La compétition "*iNaturalist*"

*iNaturalist* [28] est une compétition de classification des espèces à grande échelle. Il est estimé que le monde naturel contient plusieurs millions d'espèces de plantes et d'animaux. Sans connaissances spécialisées, beaucoup de ces espèces sont extrêmement difficiles à classer avec précision en raison de leur similarité visuelle. Le but de ce concours est de repousser les limites de la classification automatique des images en utilisant des données réelles. Le jeu de données "*iNat Challenge 2018*" contient plus de 8 000 espèces de plantes et d'animaux. Nous nous sommes intéressés dans ce jeu de données aux images des plantes uniquement.

### 5.1.2 *ImageNet*

*ImageNet* [11] est une base de données d'images annotées produit par l'organisation du même nom, à destination des travaux de recherche en vision par ordinateur. En 2016, plus de dix millions de liens de photos ont été annotées à la main pour indiquer quels objets sont représentés dans l'image. Sur *imageNet*, nous nous sommes intéressés principalement à l'extraction des photos des plantes les plus populaires dans notre pays.

## 5.2 Les outils

Nous nous intéressons dans cette partie aux outils que nous utiliserons tout au long de nos travaux.

### 5.2.1 Les outils provenant d'éditeurs tierces

Il s'agit là des outils que nous avons eu à télécharger ou utiliser sur internet. Nous retrouvons ainsi les outils suivants :

#### Keras

Développé par François Chollet, Keras [14] est une bibliothèque open source écrite en python et permettant d'interagir avec les algorithmes de réseaux de neurones profonds et de *machine learning*

#### Tensorflow

TensorFlow™ [12] est une bibliothèque logicielle open source pour le calcul numérique haute performance. Son architecture flexible permet de déployer facilement les calculs sur diverses plates-formes (processeurs, GPU, TPU). Développé à l'origine par les chercheurs et les ingénieurs de l'équipe Google Brain au sein de l'organisation d'intelligence artificielle de Google, il intègre un support puissant en matière d'apprentissage automatique et d'apprentissage approfondi.

#### `keras_to_tensorflow.py`

`Keras_to_tensorflow.py` [1] est un outil permettant de convertir un modèle keras en un modèle TensorFlow prêt à être utilisé. Il a été développé par Amir, doctorant Ingénierie informatique.

#### *Deep Playground*

*Deep playground* permet une visualisation interactive des réseaux de neurones, écrite en *TypeScript*.

#### *TOCO : TensorFlow Lite Optimizing Converter*

*TOCO* [13] est un convertisseur de graphe Tensorflow développé par Google pour une compatibilité sur Android et IOS.

## Android Studio

Android Studio est un environnement de développement pour créer des applications Android. Il est développé par Google.

## Android Development Kit

Le kit de développement (SDK) d'Android est un ensemble complet d'outils de développement. Il inclut un débogueur, des bibliothèques logicielles, un émulateur...

## Android Native Development Kit

Le NDK Android est un ensemble d'outils qui permet d'implémenter tout ou parties d'applications android en code natif, en utilisant des langages tels que C et C++.

## Android Neural Networks

L'API *Android Neural Networks* (NNAPI) est une API Android conçue pour exécuter des opérations de calcul intensif pour l'apprentissage automatique sur des appareils mobiles. NNAPI fournit une couche de fonctionnalités de base pour les structures d'apprentissage automatique de haut niveau sur Android.

### 5.2.2 Outils développés dans le cadre de notre travail

Tout au long de notre travail, nous avons mis en place quelques outils pour simplifier et automatiser certaines tâches.

#### Téléchargement automatique d'image

Afin d'automatiser le téléchargement des images sur ImageNet sur la base de mots clés fournis en entrée, nous avons créé un outils de fouilles et d'extraction des liens de téléchargement d'images sur le site *imagenet*. Cet outils exploite les différentes annotations.

#### Comparison d'algorithmes d'optimisation de *deep learning*

Basé sur Deep Playground, nous avons écrit un programme qui à partir de simulations, dresse un comparatif croisé entre différentes méthodes d'optimisation.

## 5.3 Démarque

Dans cette partie, nous donnerons les étapes pratiques de résolution de notre problème d'optimisation et de compression de réseau de neurones avec une application à la reconnaissance d'images de plantes sur smartphone (Android). Ces étapes sont les suivantes :

1. Optimisation de l'entraînement de notre modèle de réseaux de neurones pour la reconnaissance des plantes avec Keras;
2. Etudes expérimentales pour une compression utilisant notre approche ;
3. Conversions de réseaux de neurones pour une compatibilité sur smartphone Android ;

4. Mise en place de notre application mobile de reconnaissance d'images de plantes.

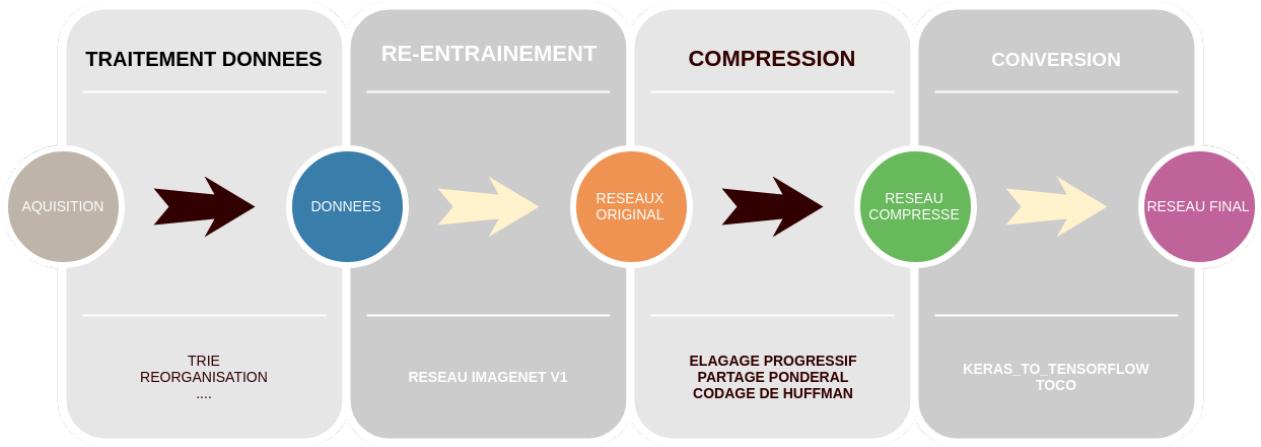


FIGURE 5.1 – Etapes de réalisation

### 5.3.1 Entrainement du modèle

Plutôt que d'opter pour un entraînement d'un modèle depuis le plus bas niveau, nous avons opté pour le recyclage d'un modèles pré-entraîné. L'entraînement de modèles n'étant pas l'objet principal de notre travail, nous avons préféré cette approche. En effet, contrairement au capacités humaines, un réseau de neurones capable de reconnaître mille (1000) objets différents, peut apprendre à en reconnaître de nouveaux en mobilisant très peu d'effort. Cela fonctionne parce que bien souvent, ces mille (1000) objets sont assez utiles pour faire la distinction entre de nouveaux types d'objets. Ainsi donc, nous avons opté de baser notre entraînement sur un modèle pré-entraîné : *imageNet v2*. L'algorithme Adam sera utilisé lors de ce ré-entraînement. Grâce à la comparaison faite des différentes méthodes d'optimisation et vu l'état de l'art, nous avons implémenté au lors de cette étape la méthode Adam (figure 4.2)

En remarques, nous nous sommes rabattus sur seulement une centaine d'espèces de plante compte tenu des ressources de calcul disponible pour nos travaux.

### 5.3.2 Compression : études expérimentales et résultats

Nos expériences visent d'abord à évaluer la faisabilité de notre approche et, éventuellement, la possibilité de déployer efficacement le réseau résultant dans un environnement contraint. D'autre part, ils concerteront également une évaluation de la solution en situation réelle. Nous étudions les questions de recherche suivantes :

RQ1 Quel est l'impact de la compression de réseau neuronal sur le temps de réponse et la précision ?

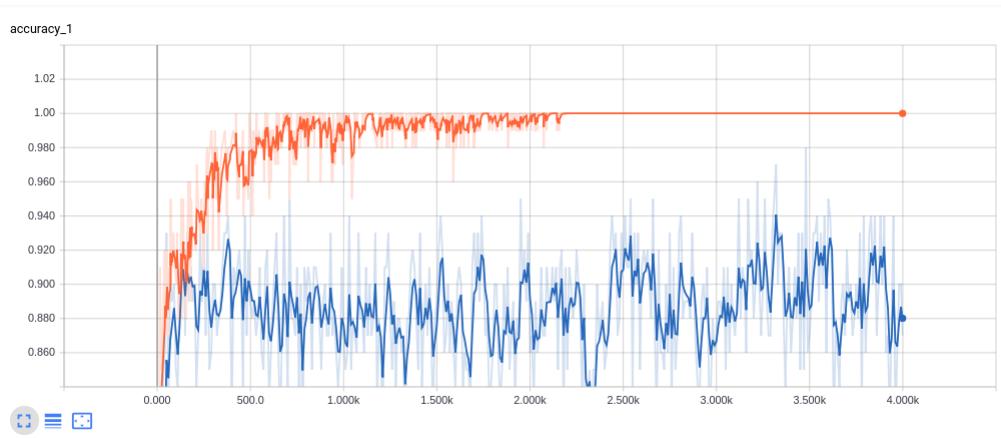


FIGURE 5.2 – Précision atteinte - Training (couleur orange) et Testing (couleur bleue)

```
root@localhost:~/M2TR/my-results# ls -l output_graph
-rw-r--r-- 1 root root 87564034 Nov 28 17:46 output_graph
root@localhost:~/M2TR/my-results# 
```

FIGURE 5.3 – Taille du modèle

RQ2 Quel est le meilleur rapport taux de compression / précision permettant de déployer efficacement un réseau neuronal sur un terminal de faibles capacités ?

### Elagage "aveugle" et monitoring des performances

Pour se faire, nous avons évalué et "classé" par ordre d'importance tous les neurones de notre réseau. De ces résultats, nous procédons à une suppression une par une par ordre décroissant des neurones de notre réseau. À chaque suppression, nous évaluons les performances de notre nouveau réseau en y extrayant les informations suivantes :

- taux de compression par rapport au réseau initial
- le temps moyen de réponse
- la précision moyenne

Ces expériences, similaires aux travaux antérieurs [3], nous montrent d'abord un impact mineur sur le réseau. Cependant, à mesure que cette opération progresse, nous constatons une diminution significative de la précision du réseau de neurones.

Après cette première expérience, nous avons ensuite essayé de nous orienter vers un compromis entre la taille et la précision de notre réseau. Pour ce faire, nous nous limitons à l'élagage d'un petit nombre de neurones qui impacte très faiblement notre réseau. Ce nouveau réseau constitue notre entrée pour la prochaine étape.

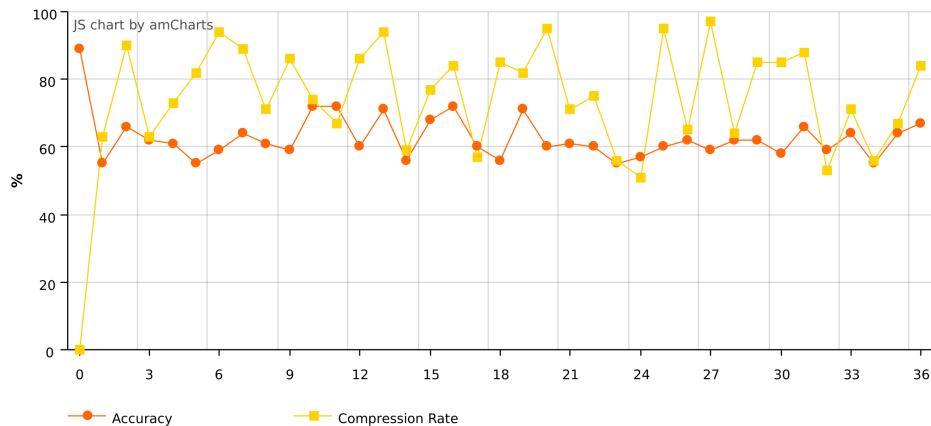


FIGURE 5.4 – Métriques d'élagage "à l'aveugle"

### Un compromis entre taux de compression et précision

Dans ces expériences, nous avons essayé de trouver un compromis entre le taux de compression et la précision du réseau. Pour cela, nous avons implémenté notre algorithme (Algorithme 2). C'est ainsi que lors de son execution, nous définissons une plage de valeurs pour une précision "acceptables". Ainsi, la suppression d'un neurone lors de cette nouvelle manière d'élagage n'est confirmée que si la précision obtenue lors des tests avec des données externes est "acceptable". Ainsi progressivement, nous faisons le maximum de suppression. En comparant les résultats (figure 5.5) de cette série d'expériences, nous faisons un choix des trois réseaux avec le meilleur taux de compression afin de les évaluer en situation de test (tableau 5.1).

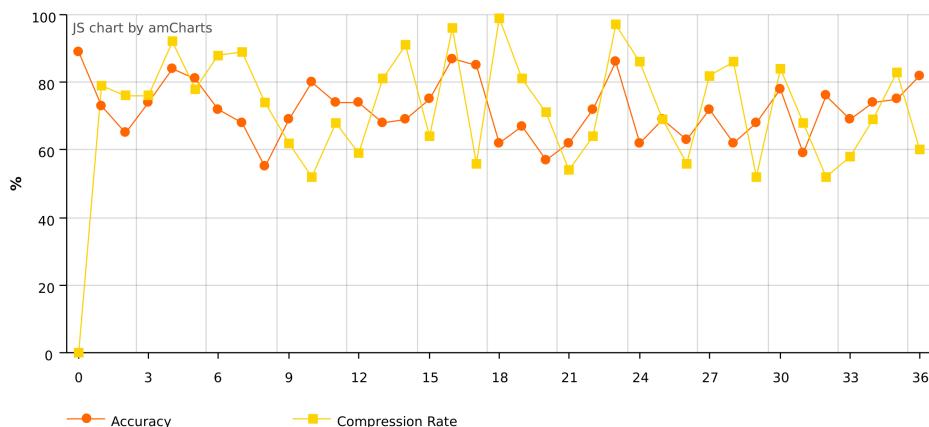


FIGURE 5.5 – ‘Métriques d’élagage “à l’aveugle” avec précision = 55%

TABLE 5.1 – Résultats de l’évaluation lors de tests

Réseau	Taux de compression	Précision (validation)	Précision (test)	Temps de réponse
1	97%	59%	44%	277ms
2	95%	60%	61%	401ms
3	94%	71%	63%	467ms

A l’issus de l’étude expérimentale, nous procedons au choix du meilleur des réseaux de notre tableau (tableau 5.1). Par la suite nous appliquons un partage pondéré et le codage de Huffman sur ce réseau. Cette étape reprend l’approche de Song Han et Al.[9].

### 5.3.3 Conversions du modèle

Après la compression de notre réseau, nous procédons par deux conversions qui sont :

1. Une première conversion de notre réseau Keras en un modèle Tensorflow : pour se faire, nous avons exploité l’outils de **keras\_to\_tensorflow** développé par Amir.
2. Une deuxième conversion du réseau résultant de 1 en un modèle Tensorflow lite : nous utilisons pour cette opération, l’outils TOCO.

Le réseau résultant de cette dernière conversion est compatible sur Android et IOS. Sur Android, l’utilisation de ce réseau tirera davantage de performances si nous l’utilisons avec l’API Android Neural Networks.

Pour un déploiement sur un système embarqué ou sur un OS différent de ceux cité plus haut, la dernière conversion se fera avec d’autres outils. Par exemple, Tensorflow.js, une bibliothèque JavaScript pour l’entraînement et le déploiement de modèles sur un navigateur web.

### 5.3.4 Application mobile de reconnaissance d'images de plantes sur Android

Afin de tester notre réseau final, nous avons mis en place une application basique sur Android. Cette application exploite notre réseau pour reconnaître des photos de plantes capturées grâce à la caméra du terminal hôte. Nous avons utilisé l'environnement de développement, Android Studio pour mettre en place notre application.

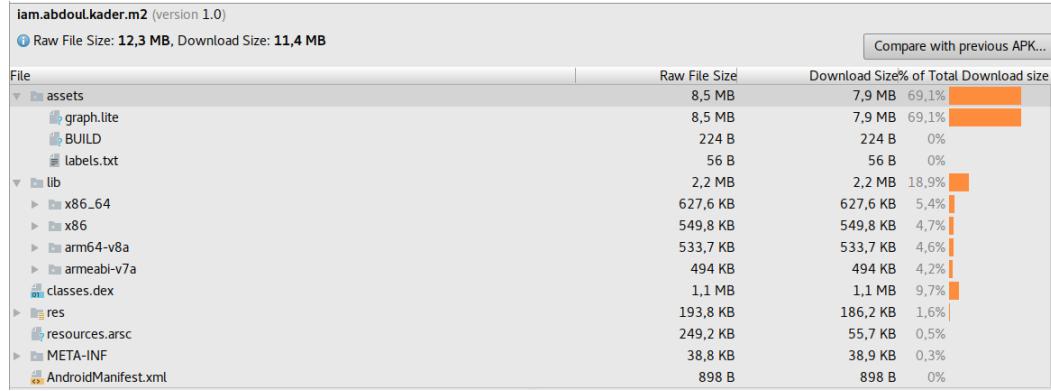


FIGURE 5.6 – Analyse de notre APK (version debug)

Cette étape qui marque la fin de ce chapitre, met également en évidence une troisième question de recherche qui est la suivante : **RQ3 : Quel est l'impact du déploiement d'une application basée sur un modèle de réseau de neurones sur le fonctionnement des autres applications et de l'OS du terminal cible ?**.

## Chapitre 6

# Résultats et évaluations

Nous présentons dans cette partie, les résultats de nos travaux suivi d'une évaluation de notre proposition en matière d'optimisation et de compression de modèles de *deep learning* pour les terminaux de faibles capacités. Cette partie sera également l'occasion de revenir sur nos questions de recherche et faire le point sur les difficultés rencontrés tout au long de notre travail.

### Sommaire

---

<b>6.1 Résultats . . . . .</b>	<b>46</b>
6.1.1 Performances . . . . .	46
6.1.2 Utilisation des ressources par notre application . . . . .	46
6.1.3 Evaluation de notre approche . . . . .	47
<b>6.2 Difficultés . . . . .</b>	<b>48</b>

---

## 6.1 Résultats

Au termes de nos travaux, nous avons atteint avec l'étape d'élagage un taux de compression de 94,04%. En terminant les deux dernières étapes de la pipeline (partage pondéral et codage de Huffman), nous atteignons un taux de compression de 97,76%. L'utilisation de TOCO pour assurer la conversion de notre réseau afin de permettre le déploiement sur Android et IOS fait passer ce taux à 98,13%.

### 6.1.1 Performances

En terme de performances de notre application Android de reconnaissance d'image de plantes, nous avons annoté manuellement les résultats de tentatives de reconnaissance sur une centaine de photos de plantes (feuilles, fruits, fleurs...) capturées directement depuis l'application mobile. Nous obtenons avec ces résultats une précision de 93%.

Comme l'indique cette capture d'écran (figure 6.1), l'application propose à chaque tentative, suivant l'ordre, les quatre (04) meilleurs résultats.

### 6.1.2 Utilisation des ressources par notre application

Comme présentée dans cette capture d'écran 6.2, la consommation mémoire de notre application est en moyenne de 1,3 *mgaoctes*. La consommation de la batterie reste également faible malgré une utilisation prolongée (environ 1%). Ces données sans une étude approfondie suggèrent un impact minimum de cette application sur le fonctionnement des autres applications installées et du système d'exploitation en particulier.

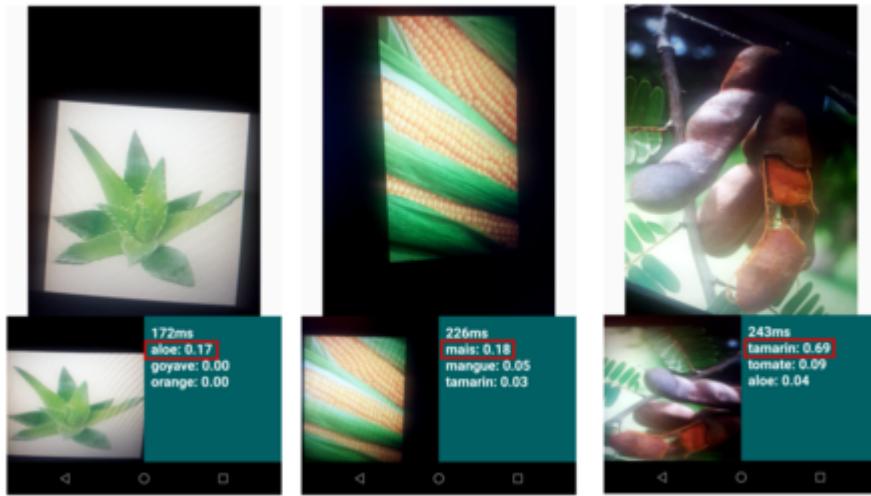


FIGURE 6.1 – Captures d’écran de notre application

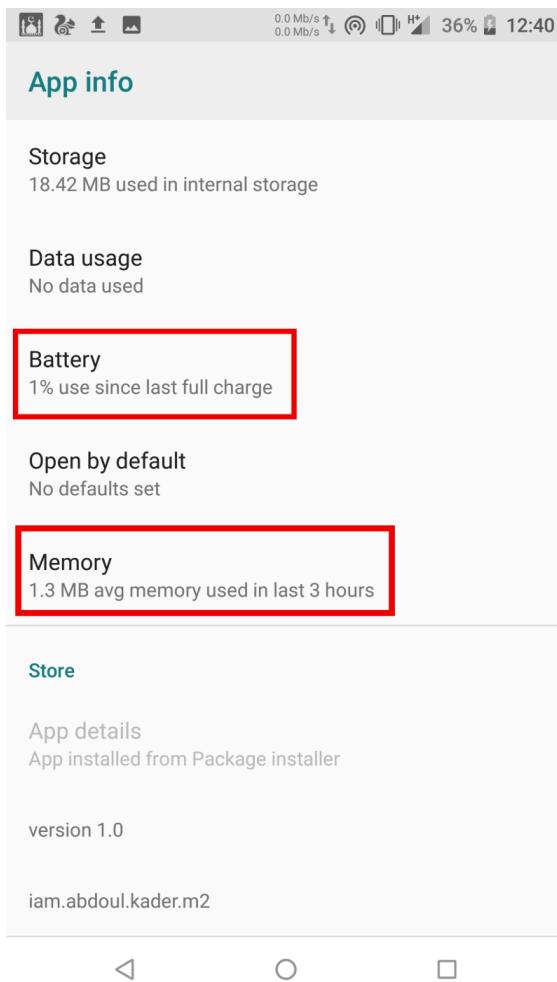
### 6.1.3 Evaluation de notre approche

Pour évaluer notre approche, nous avons procéder à une série de comparaison entre notre réseau compressé et le réseau original d’une part et d’autre part entre le réseau final et le réseau compressé. Les résultats de ces évaluations sont donnés par les captures d’écran (figures 6.3 et 6.4).

La comparaison entre le réseau original et celui compressé (figure 6.3) montre nettement une différence dans le temps de réponse. Pour le réseau original, ce temps est de 3,671 *scondes* contre 1,821 *scondes* pour le modèle compressé. Ce qui représente une baisse de 50,40%. Toutefois, nous ne remarquons aucune baisse majeure dans les résultats. En réitérant cela plusieurs fois et constatant les mêmes résultats, notre approche ne semble donc présenter aucun impact négatif sur la précision liée à l’utilisation du réseau compressé. Contrairement, il permet un gain sur le temps de réponse lors d’une exécution sur un ordinateur.

La capture d’écran (figure 6.4) présentant d’une part notre application et d’autre part l’utilisation du modèle compressé sur un ordinateur montre un contraste entre le temps de réponse et la précision. En effet, le modèle final déployé sur notre application présente un temps de réponse nettement inférieur (243 *milliscondes* contre 1821 *milliscondes*). Toutefois la précision des résultats de notre application qui utilise le réseau final présente en baisse (69%) par rapport au modèle compressé (88%).

En définitive, si le modèle compressé ne présente pas une baisse de précision vis-à-vis du modèle d’origine, son déploiement après conversion sur une application Android montre une baisse de la précision malgré un gain considérable sur le temps de réponse. Néanmoins, cette baisse de précision n’empêche pas de reconnaître efficacement (93% des cas) les plantes prises en photo.



---

FIGURE 6.2 – Capture d'écran des détails de notre application

## 6.2 Difficultés

Tout au long de notre travail, nous avons fait face à un certain nombre de difficultés dont celle qui retient le plus notre attention est l'absence de ressources informatiques pouvant supporter notre travail sur les réseaux de neurones. Bien qu'ayant tout mis en oeuvre pour pouvoir faire le maximum, cette contrainte nous a quand même contraint à travailler avec une partie de notre jeu de données sur les plantes. En effet, l'absence de serveurs mis à la disposition des étudiants pour nous permettre d'avoir des résultats en un temps raisonnable a entraîné ce choix.

```
(venv) root@localhost:~/M2TR# time python3 scripts/label_image.py --graph=my-results/  
output_graph.pb -labels=my-results/output_labels.txt --image=image.jpg  
tamarin 0.99793434  
goyave 0.00066314056  
mangue 0.0005577018  
aloë 0.00048271214  
papaye 0.00019585634  
  
real 0m3.671s  
user 0m4.007s  
sys 0m0.688s  
(venv) root@localhost:~/M2TR# time python3 scripts/label_image.py --graph=my-results/  
output_compressed_graph --labels=my-results/output_labels.txt --input_height=224 --in  
put_width=224 --image=image.jpg  
tamarin 0.9982666  
mangue 0.0007121214  
goyave 0.0003997253  
aloë 0.00038941402  
orange 8.642309e-05  
  
real 0m1.821s  
user 0m1.849s  
sys 0m0.349s
```

FIGURE 6.3 – Modèle d’origine et modèle compressé

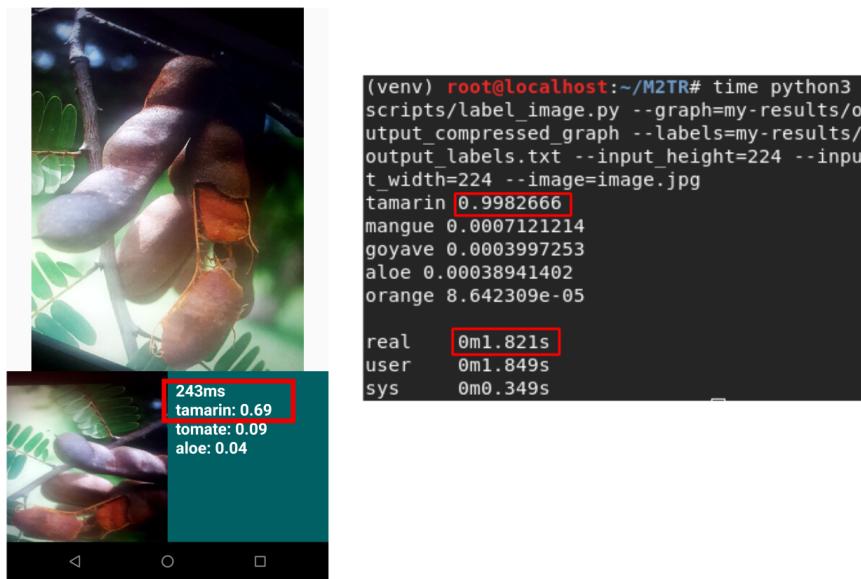


FIGURE 6.4 – Modèle compressé et modèle final

# Conclusions et perspectives

Dans ce mémoire, bien que traitant de l'optimisation et de la compression de modèles de réseaux de neurones pour les terminaux de faibles capacités, nous avons d'abord fait le point sur un certain nombre de généralités sur le *deep learning* et les réseaux de neurones en particulier pour mieux introduire le sujet.

Par la suite, à travers une étude plus approfondie des différentes approches d'optimisation et de compression des réseaux de neurones, nous avons orienter nos recherches beaucoup plus vers la compression ou il reste encore beaucoup de choses à faire. Nous avons fait ce choix parce que les différentes méthodes d'optimisation déjà existantes sont bien avancées et fournissent de meilleurs résultats. C'est ainsi qu'à travers une étude comparative de ces méthodes d'optimisation, nous avons décider de travailler avec la méthode Adam et ainsi continuer avec l'étape de la compression.

Dans ce domaine nous avons proposé une approche s'appuyant sur un pipeline en trois étapes : élagage, partage pondéral et codage de Huffman. La nouveauté de notre approche réside dans l'étape 1 du pipeline : en lieu et place de la méthode traditionnelle d'élagage pour supprimer certains neurones des réseaux de neurones, nous avons opté pour une tout autre approche qui nous permettra une suppression massive des neurones. Cela entraîne une réduction considérable de la taille des réseaux. Nous avons mise en oeuvre notre approche pour prouver sa validité à travers la mise en place d'une application Android de reconnaissance d'images de plantes.

Notre première perspective concerne une étude de faisabilité et de mise en place une application complète exploitant la reconnaissance d'images de plantes au profit de la filière phytosanitaire. En plus de cela, nous aimerais mettre en application notre approche dans d'autres champs de l'apprentissage en profondeur tels que traitement du langage naturel.

# Bibliographie

- [1] Amir. General code to convert a trained keras model into an inference tensorflow model. [https://github.com/amir-abdi/keras\\_to\\_tensorflow](https://github.com/amir-abdi/keras_to_tensorflow), 2018. [En ligne, dernier accès en Octobre 2018].
- [2] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Tvm : An automated end-to-end optimizing compiler for deep learning. In *OSDI 2018(to appear)*, 4 2018.
- [3] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Universal deep neural network compression, 2018. cite arxiv :1802.02271.
- [4] H. Cuillandre. *Un monde meilleur : Et si l'intelligence artificielle humanisait notre avenir?* Maxima, 2018.
- [5] Université de LIEGE. L'Intelligence artificielle au service de la « nouvelle physique. [https://www.news.uliege.be/cms/c\\_10359904/fr/l-intelligence-artificielle-au-service-de-la-nouvelle-physique,-](https://www.news.uliege.be/cms/c_10359904/fr/l-intelligence-artificielle-au-service-de-la-nouvelle-physique,-). [En ligne, dernier accès en Octobre 2018].
- [6] Journal du Geek. “L'intelligence artificielle n'existe pas” : interview de Luc Julia, le cocréateur de Siri. <https://www.journaldugeek.com/dossier/l-intelligence-artificielle-nexiste-interview-de-luc-julia-cocreateur-de-siri/>, 2019. [En ligne, dernier accès en Janvier-2019].
- [7] Eyra. Horus. [http://horus.tech/?l=en\\_us](http://horus.tech/?l=en_us), -. [En ligne, dernier accès en Octobre 2018].
- [8] Katsuyuki Hagiwara. On a fitting of a heaviside function by deep relu neural networks. In *Neural Information Processing - 25th International Conference, ICNIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I*, pages 59–69, 2018.
- [9] Song Han, Huizi Mao, and William Dally. Deep compression : Compressing deep neural networks with pruning, trained quantization and huffman coding. 10 2016.
- [10] Neurone Hive. AlexNet – ImageNet Classification with Deep Convolutional Neural Networks. <https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/>, 2018. [En ligne, dernier accès en Novembre 2018].
- [11] Imagenet. Imagenet. <http://www.image-net.org/>, -. [En ligne, dernier accès en Octobre 2018].
- [12] Google inc. An Open Source Machine Learning Framework for Everyone. <https://tensorflow.org/>, 2015. [En ligne, dernier accès en Octobre 2018].

- [13] Google inc. TensorFlow Lite Converter. <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/toco>, 2018. [En ligne, dernier accès en Octobre 2018].
- [14] Keras-Team. Keras : The Python Deep Learning library. <https://keras.io/>, 2014. [En ligne, dernier accès en Octobre 2018].
- [15] Diederik P Kingma and Ba J Adam. a method for stochastic optimization. 2014. *arXiv preprint arXiv:1412.6980*, 2015.
- [16] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990.
- [17] LSVRC. LSVRC. <http://www.image-net.org/challenges/LSVRC/2012/index>, 2012. [En ligne, dernier accès en Octobre 2018].
- [18] IT Nation. Boom des données : la prochaine grande crise? <https://www.itnation.lu/boom-des-donnees-la-prochaine-grande-crise/>, 2016. [En ligne, dernier accès en Octobre 2018].
- [19] Michael Nielsen. *Neural Networks and Deep Learning*. <https://www.goodreads.com/book/show/24582662-neural-networks-and-deep-learning>, 2013.
- [20] Tero Karras Timo Aila Jan Kautz Pavlo Molchanov, Stephen Tyree. Pruning convolutional neural networks for resource efficient inferenc. 10 2018.
- [21] Jeffrey Pennington, Samuel S. Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry : theory and practice. *CoRR*, abs/1711.04735, 2017.
- [22] Basheer Qolomany, Majdi Maabreh, Ala Al-Fuqaha, Ajay Gupta, and D Ben-haddou. Parameters optimization of deep learning models using particle swarm optimization, 06 2017.
- [23] Peter E. Hart Richard O. Duda. *Pattern Classification*. John Wiley Sons, 1973.
- [24] David Salomon. *Huffman Coding*. In : *A Concise Introduction to Data Compression. Undergraduate Topics in Computer Science*. [https://link.springer.com/chapter/10.1007/978-1-84800-072-8\\_2](https://link.springer.com/chapter/10.1007/978-1-84800-072-8_2), 2008.
- [25] Elite Data Science. Handling overfitting in deep learning models. <https://towardsdatascience.com/handling-overfitting-in-deep-learning-models-c760ee047c6e>, -. [En ligne, dernier accès en Octobre 2018].
- [26] H. Tripier. *Les fonctions circulaires et les fonctions hyperboliques : étudiées parallèlement en partant de la définition géométrique*. Vuibert, 1923.
- [27] The Verge. A pioneering scientist explains deep learning. <https://www.theverge.com/platform/amp/2018/10/16/17985168/deep-learning-revolution-terrence-sejnowski-artificial-intelligence-technology>, 2018. [En ligne, dernier accès en Octobre 2018].
- [28] Visipedia. iNaturalist 2018 Competition. [https://github.com/visipedia/inat\\_comp#data](https://github.com/visipedia/inat_comp#data), 2018. [En ligne, dernier accès en Septembre 2018].

- [29] Brian A. Wandell and Jonathan Winawer. Imaging retinotopic maps in the human brain. *Vision Research*, 51(7) :718–737, 4 2011.
- [30] Wikipedia. Frank Rosenblatt. [http://en.wikipedia.org/wiki/Frank\\_Rosenblatt](http://en.wikipedia.org/wiki/Frank_Rosenblatt), -. [En ligne, dernier accès en Septembre 2018].
- [31] Wikipedia. Walter Pitts. [http://en.wikipedia.org/wiki/Walter\\_Pitts](http://en.wikipedia.org/wiki/Walter_Pitts), -. [En ligne, dernier accès en Septembre 2018].
- [32] Wikipedia. Warren McCulloch. [http://en.wikipedia.org/wiki/Warren\\_McCulloch](http://en.wikipedia.org/wiki/Warren_McCulloch), -. [En ligne, dernier accès en Septembre 2018].