Etudes techniques de compression de réseaux de neurones pour sa mise en place dans une architecture embarquée de type Smartphone

Anthony Berthelier

THÈSE DE DOCTORAT

Etudes Techniques de Compression de Réseaux de Neurones Pour sa Mise en Place dans une Architecture Embarquée de Type Smartphone

Auteur :
Anthony BERTHELIER

Directeur de thèse :
Thierry CHATEAU

Soutenue le :
9 Décembre 2021

Composition du jury :

Rapporteure : Alice CAPLIER, Professeure, Grenoble INP
Rapporteur: Donatello CONTE, MCF, Univ. de Tours
Examinatrice : Laure TOUGNE, Professeure, Univ. de Lyon
Examinateur : Gregoire LEFEBVRE, Docteur, Orange Labs Grenoble

Encadrant : Thierry CHATEAU, Professeur, Univ. Clermont Auvergne
Co-encadrant : Stefan DUFFNER, MCF-HDR, Univ. de Lyon
Co-encadrant : Christophe BLANC, MCF, Univ. Clermont Auvergne
Co-encadrant : Christophe GARCIA, Professeur, Univ. de Lyon

Cette thèse a été rédigée dans le but d'obtenir
le statut de Docteur de l'Université Clermont Auvergne
à

Institut Pascal

UNIVERSITÉ CLERMONT AUVERGNE
Ecole Doctorale des Sciences Pour l'Ingénieur
Institut Pascal

Résumé

Etudes Techniques de Compression de Réseaux de Neurones Pour sa Mise en Place dans une Architecture Embarquée de Type Smartphone

par Anthony BERTHELIER

Au cours de ces dernières années, les réseaux de neurones profonds se sont montrés être des éléments centraux dans le développement de solutions intelligentes. Ils ont atteint des performances remarquables au détriment de la grande taille de leurs modèles avec de nombreuses couches profondes et des millions de paramètres. Ainsi, utiliser ces modèles pour des applications en réalité augmentée devant fonctionner sur des plateformes possédant des ressources limitées comme des systèmes embarqués ou des smartphones est une tâche loin d'être évidente. Dans ce contexte, cette thèse s'intéresse au problème de la compression et de l'optimisation de réseaux de neurones dans le but de réduire la taille de leur modèle sur des systèmes ayant des ressources limitées.

Nous présentons dans une première partie différentes méthodes de compression de la littérature ainsi que leurs forces, leurs faiblesses et une brève comparaison entre elles. Nous nous intéressons également aux méthodes permettant d'optimiser la construction des architectures de réseaux de neurones profonds, allant de simples modules à la construction autonome de modèles.

Les deux parties suivantes sont consacrées au développement et à l'évaluation d'une nouvelle méthode de compression de réseaux de neurones convolutifs profonds. Basée sur un terme de régularisation défini sur les coefficients des filtres du modèle, notre approche permet d'introduire de la dispersion au sein des poids du réseau. Cela a pour effet de redistribuer l'information entre les filtres du modèle, nous permettant de supprimer les filtres ayant une valeur faible suite à cette opération. Nous évaluons les performances de notre méthode sur des tâches de classification classiques. Nous introduisons aussi l'efficacité de notre technique sur des modèles et des tâches de classification, segmentation et détection plus complexes en spécialisant les modèles sur certaines sous-catégories de jeux de données.

Nous montrons dans une dernière partie la faisabilité technique d'une application en réalité augmentée et en temps réel utilisant un modèle d'apprentissage profond pour de l'analyse de visages. En utilisant des frameworks adaptés et une architecture optimisée, nous arrivons à segmenter différents composants du visage de manière robuste en temps réel sur un iPhone X.

Mot clés : Compression ; Optimisation ; Systèmes embarqués ; apprentissage profond

UNIVERSITÉ CLERMONT AUVERGNE
Ecole Doctorale des Sciences Pour l'Ingénieur
Institut Pascal

Abstract

Technical Study of Neural Networks Compression for its Implementation on Embedded Systems Architecture

by Anthony BERTHELIER

Over the past years, deep neural networks have proved to be an essential element for developing intelligent solutions. They have achieved remarkable performances at a cost of a large size with deeper layers and millions of parameters. Therefore utilising these networks for developing augmented reality applications on limited resource platforms such as embedded devices or mobile phones is a challenging task. In this context, this thesis addresses the problem of neural networks compression and optimisation in order to enhance the performance of these models on limited resource systems.

In the first part of this manuscript, we present an overview of different compression methods present in the literature as well as their strengths, their weaknesses and a brief comparison of these techniques. We are also interested in the methods that are allowing the optimisation of deep neural networks structure design, from simple modules to autonomous models building.

The second and third parts are focused on the development and the evaluation of a new deep convolutional neural networks compression method. Based on a regularisation term which is defined on the filter coefficients of the model, our approach is inducing sparsity among the weights of the network. Thus, our method is redistributing the information between the model filters, enabling us to remove the filters with the smaller values. We show the performance of our method on classic classification tasks. We are also introducing the efficiency of our technique on more complex models and tasks such as classification, segmentation and detection problems while specialising these models on a subset of categories on several databases.

In the last part, we show the feasibility of an augmented reality application in real-time using a deep learning model in order to achieve a face parsing task. By using adapted frameworks and an optimised architecture, we achieve the segmentation of different face components in real-time with a high level of consistency on an iPhone X.

Keywords : Compression ; Optmization ; Embedded Devices ; Deep Learning

Remerciements

Avant toute chose, je tiens à remercier Thierry Chateau, mon principal encadrant lors de cette thèse, pour son aide précieuse durant ces dernières années. Cette thèse n'aurait pas été possible sans lui, sans sa patience, sans sa gentillesse et sans ses nombreux encouragements. L'expérience, aussi bien humaine que professionnelle, que j'ai pu engranger a ses cotés restera à vie avec moi.

Je tiens aussi à remercier profondément mes autres encadrants. Stefan Duffner pour tous ses nombreux conseils avisés lors de nos réunions et pour son support lors de l'élaboration de papiers et de leurs relectures à l'approche des échéances. Et enfin Christophe Garcia et Christophe Blanc, pour leur soutient sans failles pendant toute la durée de cette thèse et pendant les difficultés qu'elle a pu rencontrer. Mes plus profonds remerciements à vous trois.

De plus, j'aimerais exprimer mes remerciements les plus sincères à Alice Caplier et Donatello Conte qui ont accepté de rapporter ce manuscrit, ainsi qu'au reste des membres du jury, incluant Laure Tougne et Grégoire Lefebvre.

J'aimerais aussi remercier mon partenaire et co-doctorant Yongzhe Yan, ce fût un plaisir d'évoluer à tes cotés lors de nos thèses respectives. Mes remerciements vont aussi à l'Université Clermont Auvergne et aux membres de l'équipe ComSee : Céline, Ruddy, Rémi, Antoine, Simon, Gauthier et tous les autres pour nos nombreux échanges. Un grand merci aussi à l'équipe du Mésocentre pour sa réactivité et pour avoir rendu disponible de nombreuses ressources nécessaires à l'élaboration de cette thèse.

Ma sincère gratitude va aussi à l'ensemble des (ex)membres de Wisimage : Wendy, Aurélie, Alex, Isamel, Issam, Mohammad, Zesheng, Benjamin et Maxime. J'aimerais remercier plus particulièrement Xavier et Gael qui ont énormément apporté à cette thèse avant leur départ de Wisimage. Leur soutient et leur bienveillance sont un des piliers fondamentaux de cette thèse. Je tiens aussi à remercier Priyanka, Arnaud et Léo pour non seulement avoir été des collègues chaleureux, mais aussi des amis formidables.

Mes remerciements vont aussi à toutes les personnes que j'ai rencontrées au cours de ces années de thèse à Clermont-Ferrand et qui sont aujourd'hui des amis : Loic, Tiffany, Cecilia, Alexis, Deborah, Yohan, Lea, Arnaud et Pierrick. Votre compagnie fut un soutient essentiel lors de ces dernières années.

A mes amis de longue date. A Maxime et Benjamin, parce qu'il n'y a aucune raison de changer un trio qui fonctionne aussi bien. A Elise, la meilleure partenaire pour des discussions interminables. A Nadia, pour ne jamais m'oublier malgré la distance. A Théo, parce que chacune de nos rencontres est aussi rare qu'importante. A Adeline, pour les nombreuses comètes en terrasses. Merci à vous de m'avoir accompagné jusqu'ici et de continuer à m'accompagner encore longtemps je l'espère.

Enfin, merci infiniment à ma famille : mes parents, mon frère Grégory, Sandra et Chloé. Merci pour m'avoir toujours soutenu quoi que je fasse. Cette thèse vous est dédiée.

Table des Matières

Liste des Figures

Liste des Tableaux

Glossaire

BSN Budgeted Super Networks. xi, 40, 41

CNN Convolutional Neural Network. xi, 6, 8, 10, 11, 13, 15–19, 21, 25–27, 30, 31, 43, 46, 71, 73–75, 80, 83, 87–89, 93–95

DCNN Deep Convolutional Neural Network. 43, 45, 47, 48, 69, 71, 84

FC Fully Connected. 14, 16

FCNN Fully Convolutional Neural Network. 11, 16, 17, 33

FPGA Field Programmable Gate Array. xi, 6, 7

FPN Feature Pyramid Network. xiii, xv, 75, 83, 84

GAL Generative Adversarial Learning. 46

GAN Generative Adversarial Network. xi, 3, 4

GRU Gated Recurrent Unit. 13

IoU Intersection over Union. 80

LSH Locality-Sensitive Hashing. 20

LSTM Long Short-Term Memory. 13

mAP mean Average Precision. 80

MLP Multilayer Perceptron. xi, 14, 15

NAS Neural Architecture Search. 31, 35, 39, 41, 42

NEAT NeuroEvolution of Augmenting Topologies. 37, 38

R-CNN Regional Convolutional Neural Network. xiii, xv, 75, 83, 84

Chapitre 1

Introduction

1.1 Contexte Général

1.1.1 Contexte des applications de réalité augmentée

A travers l'histoire, on peut remarquer que le désir et le besoin de créer des expériences mettant en scène des réalités alternatives sont intimement liés à la nature imaginative des êtres humains. Il aura cependant fallu attendre la fin des années 60 et les innovations technologiques qu'elles ont amenées afin d'en voir les premières créations utilisant des outils informatiques. Appelée *Epée de Damoclès*, la création de Sutherland [Sutherland, 1968] constitue la première occurrence d'un dispositif ressemblant à un casque de réalité virtuelle tel qu'on pourrait le définir aujourd'hui. Bien qu'accroché au plafond par des câbles pour permettre son utilisation (d'où le nom), les utilisateurs pouvaient à travers cet appareil visualiser et interagir avec des images simples générées par ordinateur grâce à un suivi de la position de la tête.

Malgré cela, il faudra attendre le début des années 90 afin de voir apparaître le terme "Réalité Augmentée", introduit par les travaux de Caudel et al. [Caudell and Mizell, 1992]. Le premier appareil de réalité augmentée opérationnel a cependant été développé par Rosenberg et al. [Rosenberg, 1993]. Appelé *Virtual Fixtures*, l'appareil permet de rajouter une surcouche d'information à l'utilisateur afin de l'aider à améliorer son efficacité dans la réalisation de différentes tâches. Depuis ces années là, la réalité augmentée a commencé à se démocratiser et à être introduite dans différentes applications allant du simple divertissement à des simulations de champs de batailles utilisées par l'armée.

Les applications de réalité augmentée modernes sont donc désormais étendues à un grand nombre de domaines. On peut notamment cité le champ médical ou lors de certaines opérations à risques, les chirurgiens peuvent être assistés par des applications de réalité augmentée afin de mieux les guider [Bichlmeier et al., 2007]. Certains pays utilisent aussi des systèmes de réalité augmentée dans leur système éducatif. Certaines études comme celle de Radu [Radu, 2012] montrent que les méthodes éducatives basées sur de la réalité augmentée avaient pour effet d'aider les étudiants à les motiver à apprendre de nouvelles choses, à maintenir l'attention et à améliorer la rétention des informations. Impossible de parler de réalité augmentée sans aussi mentionner toutes les applications de divertissement qui en font l'usage, notamment sur téléphones mobiles. Le plus bel exemple est l'énorme succès de l'application *Pokemon GO* depuis plusieurs années, ayant aussi permis d'introduire les expériences en réalité augmentée à une partie du grand public.

Encore plus proche de nous, la démocratisation des applications de réalité augmentée, aussi bien sur internet que sur téléphones mobiles, a permis l'apparition de nombreuses

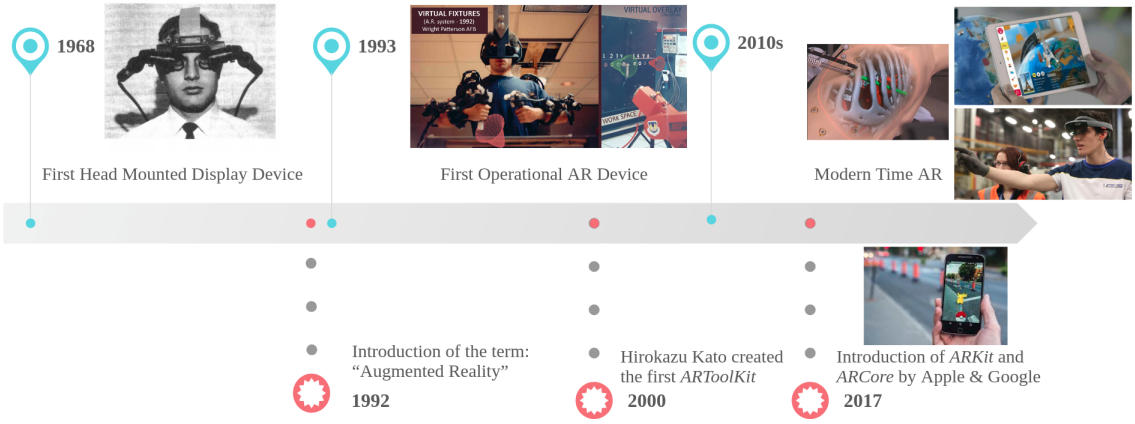
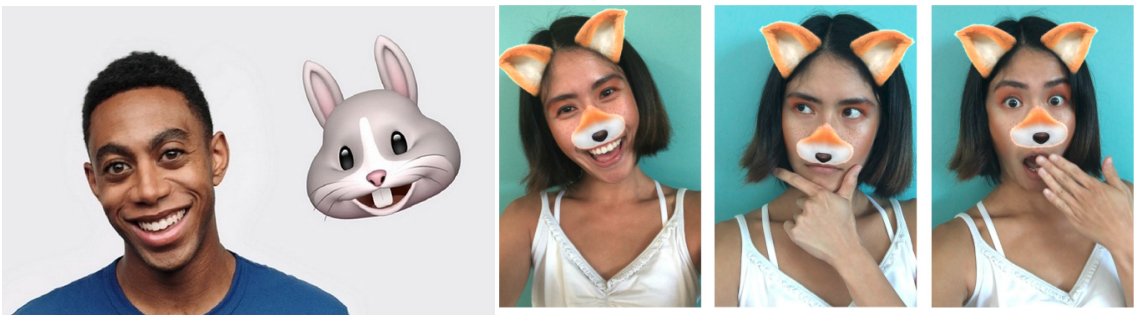


FIGURE 1.1 – Un bref résumé de l'évolution des technologies de réalité augmentée.

solutions logicielles pour développer ce genre d'application. En 2017, les entreprises responsables des deux logiciels d'exploitation les plus répandus sur téléphones mobiles, respectivement Apple avec iOS et Google avec Android, ont chacun mis à disposition leur solution logicielle pour développer des applications en réalité augmentée sur leur plateforme avec respectivement *ARKit* et *ARCore*. Du point de vue du développeur, ces deux solutions simplifient grandement l'intégration d'éléments en réalité augmentée dans des applications. Du point de vue de l'utilisateur, la démocratisation des smartphones et des applications en réalité augmentée accessibles en un clic fait qu'il n'est plus utile d'utiliser du matériel encombrant comme des casques ou des lunettes pour bénéficier d'expériences de qualité. Ainsi, le domaine de la réalité augmentée explose depuis quelques années en parallèle de toutes les avancées technologiques qui lui sont liées.

1.1.2 Estimation du haut du corps sur téléphones mobiles

Parmi toutes les applications de réalité augmentée, celles permettant la détection du haut du corps et plus particulièrement du visage sont les plus populaires. Avec l'avènement des téléphones mobiles et de leurs caméras de plus en plus performantes au fil des années, des milliers d'applications du genre sont désormais accessibles au grand public lui permettant ainsi de remodeler son visage, de lui ajouter des animations, d'échanger son visage avec un autre, d'ajouter des stickers, etc. Des exemples de ce qu'il est possible de faire avec les applications en réalité augmentée de Google et Apple sont visibles sur la Figure 1.2.



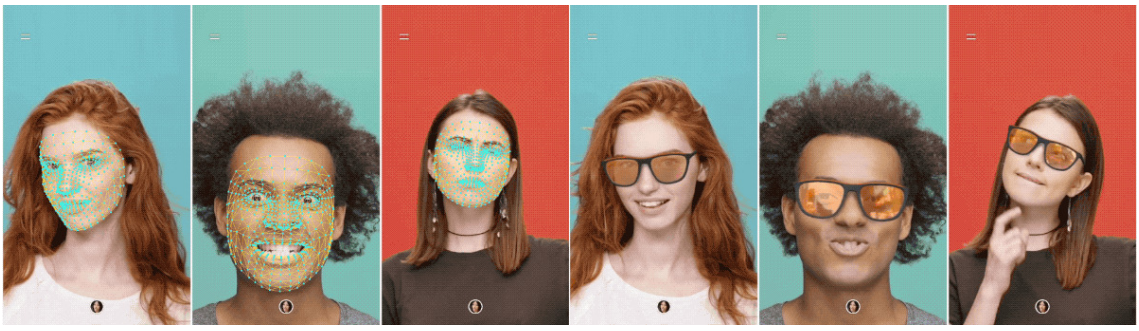
(A) *Animoji* introduit par Apple

(B) *Visages Augmentés* introduit dans *ARCore* de Google

FIGURE 1.2 – Exemples d'applications pour le visage en réalité augmentée sur plateformes mobiles.

Ces dernières années, de très nombreuses applications ont été développées pour appliquer des effets ou des objets virtuels sur des visages humains. Ce développement a particulièrement été aidé par les travaux récents sur les réseaux antagonistes génératifs (Generative Adversarial Network ou GAN) [Goodfellow et al., 2014] et sur le transfert de style [Gatys et al., 2016]. Parmi celles-ci, on peut citer 3 catégories de méthodes qui sont particulièrement utilisées pour réaliser des modifications sur des visages.

— **Edition de visages par superposition :** cette approche est une des plus simples afin de réaliser des modifications sur des visages à partir de photographies. Il suffit de commencer par détecter la position des visages ainsi que de ses principaux composants (comme les yeux, le nez et la bouche) pour superposer différents effets virtuels aux endroits voulus. Un exemple est visible sur la Figure 1.3 qui montre ce principe pour le positionnement de lunettes virtuelles. Il est aussi important de noter que la superposition d'éléments est généralement plus complexe que de la simple superposition d'objets et que des étapes de post-traitement minutieuses comme de la déformation 3D sont souvent nécessaires.



(A) Détection d'éléments du visage (B) Superposition de l'effet visuel

FIGURE 1.3 – Exemple d'éditions de visage par superposition. Cette figure est reprise de [Ablavatski and Grishchenko, 2019].

— **Edition de visages basée sur des GAN :** contrairement à l'édition par superposition, les méthodes basées sur des GAN permettent de modifier les photos intégralement via des méthodes d'apprentissages sans aucune intervention manuelle ou de patrons virtuels prédéfinis. Le principal avantage de ces méthodes est le réalisme accru des effets virtuels notamment car ils peuvent s'adapter à différentes conditions d'éclairages ou de poses de l'image entrante. Un exemple de maquillage virtuel via cette méthode est visible sur la Figure 1.4. On observe que contrairement aux méthodes par superposition (représentées par la colonne warping result), les méthodes basées sur des GAN gardent plus de détails de l'image entrante comme la couleur de l'iris des yeux, mais sont toujours capables d'appliquer des effets virtuels similaires à l'image de référence. Cependant, cette méthode nécessite de grandes bases de données d'images afin d'obtenir des résultats visuellement satisfaisants.

— **Edition de visages par transfert de style :** Liu et al. [Liu et al., 2016] ont proposés de considérer les effets virtuels comme des styles artistiques. Ainsi, appliquer des effets virtuels peut donc être considéré comme du transfert de style [Gatys et al., 2016, Liao et al., 2017]. Un transfert d'effet de fard à paupières est montré sur la Figure 1.5. De manière similaire à des méthodes basées sur des GAN, aucun patron d'effet virtuel n'est nécessaire. Cependant, il n'y a pas de garanti quand au fait que

le rendu de l'image finale sera aussi naturel que ceux des images de références.

FIGURE 1.4 – Exemple d'éditions de photos de visages basées sur des méthodesGAN. Le maquillage de l'image de référence est appliqué sur l'image entrante. L'edition de visages par superposition est nommée "Warping results" ici. La figure est reprise de [Chang et al., 2018].

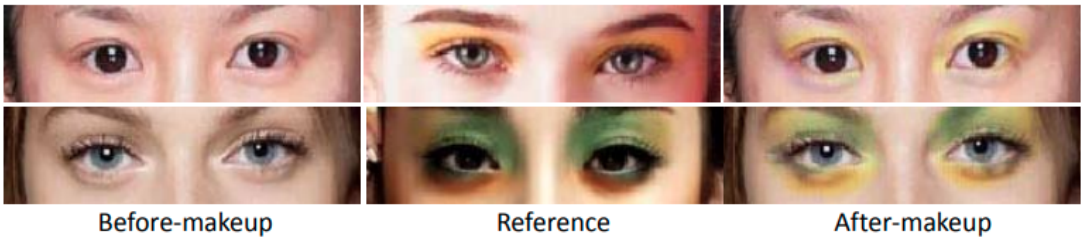


FIGURE 1.5 – Exemple d'éditions de visages par transfert de style. L'effet de maquillage de l'image de référence est appliqué sur l'image entrante. Cette figure est reprise de [Liu et al., 2016]

Dans toutes ces méthodes, la détection des composants faciaux doit être robuste et précise. Une thèse à donc été réalisée par Yonghze Yan en parallèle de celle-ci afin de parvenir à détecter et classifier avec précision des points caractéristiques ou des régions précises du visage en vue de pouvoir réaliser des applications de maquillage virtuel ou de coloration virtuelle de cheveux. Cependant, les modèles permettant d'obtenir une détection précise des points du visage sont des modèles lourds, aussi bien en poids mémoire qu'en temps de calculs, et doivent fonctionner en temps réel sur des plateformes mobiles de type smartphone ou tablette afin d'appliquer les effets virtuels voulus. Les ressources de calcul de ces appareils étant limitées, il était donc important de trouver des méthodes afin de réduire et simplifier ces modèles pour qu'ils soient aussi efficaces tout en demandant moins de ressources. C'est l'objet de cette thèse.

1.1.3 Contexte industriel de la thèse

Cette thèse a été réalisée en collaboration avec un partenaire industriel : la société *Wisimage*. Cette start-up est spécialisée dans l'élaboration et la mise sur le marché d'applications en réalité augmentée pour l'industrie cosmétique permettant des modifications du haut du corps (principalement le visage). *Wisimage* a déjà dirigé de nombreux travaux scientifique de recherche dans le domaine de l'analyse de visage [Vu, 2010, Schwab, 2013] allant de tâches de simples reconnaissances de visages jusqu'à des tâches de suivis complets de ces derniers dans l'espace. Désormais, *Wisimage* détient un éventail de produits diversifié (voir Figure 1.6) à destination de l'industrie cosmétique incluant de la simulation de maquillage virtuel, du diagnostic de peau, des bases de données de nombreux produits cosmétiques et des recommandations personnalisées de ces derniers. Plusieurs applications développées par *Wisimage* sont illustrées sur la Figure 1.7. Outre ces produits déja existants, *Wisimage* à de nombreux projets en cours tel (1) qu'un kit de développement logiciel pour la détection de points caractéristiques du visage, (2) des simulations de lentilles virtuelles de différentes couleurs et (3) des simulations de maquillage virtuel (incluant de la reconnaissance spécifique de maquillage pour permettre des opérations de transfert de maquillage entre visages).

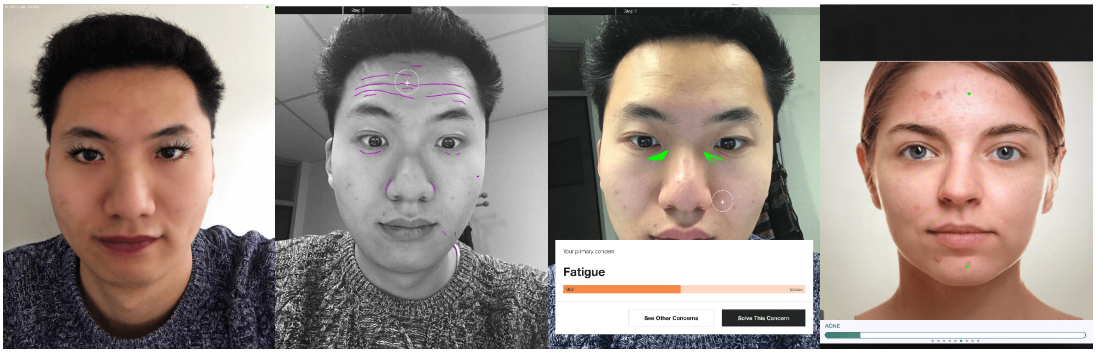
De manière plus précise, le projet de maquillage virtuel de *Wisimage* (voir Figure 1.7 (A)) produit des applications en réalité augmentée fonctionnant sur des plateformes mobiles (téléphones ou tablettes) en temps réel. Les algorithmes utilisés dans ce genre d'application pour détecter des points caractéristiques du visage sont clairement liés à des problèmes d'analyse de visages humain dans le domaine de recherche de la vision par ordinateur. En d'autres termes, ce sont des algorithmes souvent complexes avec de nombreux paramètres.

FIGURE 1.6 – Ligne de produits de *Wisimage*.

1.2 Contexte scientifique

1.2.1 Apprentissage profond et appareils mobiles

Depuis l'avènement des architectures de réseaux de neurones profonds et leur apport de nombreuses opérations entièrement parallélisées [Krizhevsky et al., 2012, LeCun et al., 2015], les méthodes basées sur de l'apprentissage profond ont atteint des performances au sommet de l'état de l'art dans de nombreuses applications telles que la détection d'objets, la reconnaissance faciale, la segmentation sémantique, etc. C'est donc ce genre de



(A) Maquillage virtuel (B) Détection de rides (C) Détection de fatigue (D) Detection d'acné

FIGURE 1.7 – Différents exemples d'applications en réalité virtuelle proposées par *Wisimage*.

modèle qui nous intéresse, et plus particulièrement les Convolutional Neural Network (CNN), afin d'être le plus précis possible dans le développement d'applications en réalité virtuelle liées à la détection du haut du corps.

Cependant, afin d'obtenir des performances aussi élevées, de grandes capacités de calcul sont nécessaires car ces modèles possèdent généralement des millions de paramètres. De plus, l'implémentation de ces méthodes sur des appareils disposant de ressources limitées comme des caméras intelligentes ou des smartphones est une tâche difficile liée à la grande consommation mémoire de ces méthodes et les restrictions mémoires très strictes imposées par ces appareils. Par exemple, un modèle comme AlexNet [Krizhevsky et al., 2012] pèse plus de 200MB et tous les modèles marquant des étapes importantes dans l'avancée des modèles profonds comme VGG [Simonyan and Zisserman, 2015], GoogleNet [Szegedy et al., 2015] et ResNet [He and Sun, 2015] ne sont pas forcément plus efficaces en terme de mémoire utilisée et en temps de calcul. Ainsi, trouver des solutions afin d'implémenter des modèles profonds sur des plateformes disposant de ressources limitées comme des smartphones ou des caméras intelligentes est devenu un sujet de recherche important de ces dernières années. Chaque appareil dispose de capacités de calcul qui lui sont propre, il n'y a donc pas de solutions universelles. Il est donc important de trouver des méthodes en amont pour réduire le nombre de paramètres de ces modèles profonds tout en maintenant leur haut niveau d'efficacité afin d'assurer le bon fonctionnement des applications sur des systèmes embarquées.

Quelques travaux ont déjà été effectués dans le passé en se concentrant sur des appareils ou des architectures spécifiques tel que les Field Programmable Gate Array ou FPGA (voir un exemple sur la Figure 1.8) . Avoir une architecture unique est une aide précieuse pour optimiser une application donnée. En revanche, généraliser une optimisation à plusieurs systèmes différents est une tâche difficile. Les architectures des CPU des smartphones sont différentes les unes des autres. Il est donc important de développer des méthodes génériques permettant d'optimiser les réseaux de neurones.

1.2.2 Travaux de recherches

Dans cette thèse, nous nous concentrons sur la problématique de comment compresser des réseaux de neurones afin qu'ils soient plus efficaces en terme de consommation mémoire et de temps de calcul tout en donnant des résultats visuels fiables et une précision semblable à des modèles non compressés. Bien que les travaux développés dans

FIGURE 1.8 – Les FPGA sont de petites cartes programmables composées de semi-conducteurs qui ont la particularité d'effectuer des actions spécifiques de manière rapide et efficace grâce à leur système logique fermé et maîtrisé.

cette thèse visent à être utilisés dans le cadre de modèles destinés à des applications d'estimation du haut du corps, les méthodes utilisées restent volontairement génériques afin d'être applicables à tous les types de modèles profonds et toutes applications confondus. C'est d'ailleurs pour cela que nos méthodes sont testées sur des modèles très connus de classification, segmentation ou de détection qui ne sont pas nécessairement des applications d'analyse de visages. Afin de mener ces travaux de réduction de modèles d'apprentissage profond à bien, nous avons considéré trois grand types d'approches :

— **Les techniques de compression** sont les méthodes qui vont être utilisées pendant ou après l'apprentissage d'un modèle profond dans le but de l'optimiser sans modifier l'architecture globale du modèle. Le but de ce genre de méthode va par exemple être de supprimer des paramètres apportant peu d'information, de regrouper des paramètres similaires, de réduire la précision des données, etc. Tout cela dans le but de réduire les temps de calcul et la consommation mémoire de ces modèles.

— **L'optimisation des architectures** vise directement la structure des modèles profonds en créant des architectures optimisées pour une tâche donnée dès le départ plutôt que d'abord construire une architecture et ensuite penser à l'optimiser avec diverses techniques de compression. Il va donc s'agir d'optimisation de différents modules, de comment ils interagissent entre eux, voire de réussir à automatiser des recherches d'architectures optimisées.

— **Les outils des frameworks** peuvent beaucoup nous aider dans l'optique de compresser un modèle. C'est pourquoi le choix du framework utilisé dans nos travaux est essentiel. De nombreux frameworks existaient au début de cette thèse, certains prédominent aujourd'hui grâce à leur flexibilité et grâce aux outils mis à disposition pour compresser et porter des modèles d'apprentissage profond sur des plateformes mobiles.

1.2.3 Challenges

Nous venons de présenter succinctement les trois principaux axes de recherches qui ont été explorés dans cette thèse. Cependant, la compression de modèles d'apprentissage profond implique de nombreux défis dans chacun de ces axes.

Challenge des techniques de compression : les techniques de compression sont des méthodes applicables à tous les types de modèles d'apprentissage profond. Il n'empêche qu'en pratique, leur développement et leur mise en application peuvent se montrer difficiles pour plusieurs raisons :

— **Choix de la méthode :** il existe de nombreuses manières de compresser des réseaux de neurones allant de la suppression de paramètres à la redéfinition du codage des données. Il est même souvent possible de combiner plusieurs de ces méthodes ensemble pour de meilleures performances. Cependant, les performances peuvent varier entre modèles et entre applications. Nous abordons plus en détail cette thématique dans la section 2. Dans le cadre de cette thèse, nous nous intéressons essentiellement à compresser des modèles CNN.

— **Vitesse :** de nombreuses méthodes de compression s'intéressent à supprimer des paramètres jugés comme non essentiels à un modèle. Cependant, une suppression non optimisée, même de paramètres inutiles, peut mener à des modèles ayant une mauvaise dispersion de leurs paramètres, limitant ainsi la parallélisation des calculs et ralentissant l'exécution des tâches [Anwar et al., 2017].

— **Précision :** le principal problème lors de la compression d'un réseau est de réduire sa taille en mémoire et d'accélérer ses calculs tout en gardant des performances similaires au modèle non compressé. Certains travaux [Ba and Caruana, 2014] rapportent que des petits réseaux de neurones seraient capables d'apprendre la même fonction que des réseaux plus profonds tout en ayant de meilleures performances mais la mise en pratique est loin d'être aussi simple.

Challenge des architectures optimisées : Construire des architectures optimisées dès le départ pour une tâche donnée est un problème complexe. Bien que le domaine ait été un peu délaissé dans un premier temps au profit de la conception de modèles imposants avec de nombreux paramètres (et de méthodes pour les compresser par la suite), il devient avec le temps un domaine de recherche de plus en plus important. Ce domaine reste néanmoins d'une grande complexité et est rempli de défis :

— **Choix des couches :** savoir comment agencer les différentes couches de modèles d'apprentissage profond afin d'obtenir le modèle plus optimisé possible est une tâche difficile car c'est une étape itérative manuelle qui n'est souvent viable que pour un certain type d'application donné. Des modèles comme SqueezeNet [Iandola et al., 2016] ont des modules composés de plusieurs couches de convolution optimisés pour de la classification par exemple et il est difficile de les réorienter vers d'autres tâches. Ce n'est pas une opération impossible cependant, comme le montre une version de SqueezeNet capable d'opérations de segmentation [Nanfack et al., 2017].

— **Espace de recherche :** concevoir un réseau de neurones est un processus itératif et complexe lorsqu'il est effectué de manière manuel. Il l'est encore plus lorsqu'il

s'agit de construire une architecture optimisée pour une tâche donnée de manière automatique, sans intervention humaine. Néanmoins ces méthodes se sont révélées très efficaces ces dernières années, permettant de trouver des modèles dépassant les performances de certains états de l'art [Zoph and Le, 2017]. Il est important de bien définir l'espace de recherche dans lequel le réseau va être construit et quelles couches sont utilisables sous peine de se retrouver avec une infinité de solutions possibles.

Challenge des outils de framework : bien que moins complexe et déterminant que les deux précédents axes de recherche, l'étude des frameworks d'apprentissage profond est une étape cruciale car elle permet de passer outre certaines limitations lors de la compression et l'optimisation de modèles, notamment sur téléphones mobiles :

— **Flexibilité :** de nombreux frameworks sont disponibles sur le marché, tous possédant des particularités spécifiques. Or, il nous faut un framework capable d'être assez flexible pour modifier des parties précises de certains modèles étant donné que nous allons modifier directement sa structure et ses paramètres afin de le rendre plus léger.

— **Portabilité :** n'oublions pas que le but de cette thèse est de faire fonctionner des modèles d'apprentissage profond sur téléphone mobile. Il est donc important que les frameworks proposent des outils simples et efficaces afin de transposer aisément les modèles sur ces plateformes. Tous n'en sont pas capable sans un certain nombre d'opérations fastidieuses.

1.3 Contributions de cette thèse

1.3.1 Avant-propos

Au cours de cette thèse, de nombreux travaux ont été menés. Néanmoins, nous ne sommes pas allez au bout de toutes ces pistes pour plusieurs raisons.

Différentes études de frameworks ont été effectuées afin de voir lesquels d'entres eux permettraient de développer de nouvelles approches pour la compression de réseaux de neurones sur téléphones mobiles. Peu après le début de cette thèse, de nouveaux outils de frameworks tel que TensorFlow avec Tensorlite, ARCore de Google ou CoreML d'Apple sont apparus. Ces outils ont permis non seulement de faciliter les processus de portage des modèles d'apprentissage profonds mais aussi d'intégrer des méthodes de compression (notamment de la quantification) dans ces processus. Le développement de ces outils a laissé en suspends notre intérêt pour le développement de méthodes de compression mais nous a permis de réaliser un modèle d'analyse de visage en temps réel pour des applications en réalité augmentée sur plateformes mobiles et de nous focaliser sur d'autres sujets de recherches.

Des contributions dans différents domaines ont essayé d'être apportées dans cette thèse, toutes en rapport avec les challenges cités dans la partie précédente. Ainsi, de nombreux travaux ont été menés dans le but de contribuer aux méthodes de compression ou à trouver de nouveaux moyens pour construire automatiquement des architectures optimisées. Néanmoins, beaucoup de ces pistes ne se sont pas révélées satisfaisantes au niveau des méthodes et des résultats obtenus. Ainsi sur les trois principaux sujets de recherches

présentés en amont, à savoir les techniques de compression, les architectures optimisés et les outils de framework, les contributions présentées dans ce mémoire sont essentiellement centrées sur le domaine des techniques de compression. Il serait sans doute intéressant de revenir sur toutes les pistes explorées durant cette thèse, même si nous n'avons pas été au bout de tous ces travaux. Cependant, nous faisons le choix de présenter dans ce manuscrit les contributions significatives, ayant permis des publications ou pouvant permettre des publications dans le futur.

1.3.2 Principales contributions

La structure de cette thèse s'oriente en quatre principaux chapitres détaillant autant de contributions différentes :

— **Un état de l'art sur la compression de modèles profonds et l'optimisation d'architectures :** Nous avons réalisé une bibliographie complète des différentes méthodes existantes permettant de compresser les modèles d'apprentissage profond, et en particulier ceux qui nous intéressent, c'est à dire les CNN. Nous avons aussi réalisé une bibliographie complète des méthodes de constructions d'architectures optimisées, allant de l'explication de certains modules spécialement optimisés pour des petits réseaux jusqu'aux méthodes de constructions automatiques de réseaux pour une tâche donnée. Toutes ces méthodes peuvent être utilisées pour compresser des réseaux de neurones sur des téléphones mobiles ou des systèmes embarqués et nous discutons des avantages et inconvénients propre à chaque méthode. Ce travail d'état de l'art complet a donné lieu a une publication dans *Journal of Signal Processing Systems*. Il nous sert de base pour le rapport bibliographique de ce manuscrit de thèse que nous avons complété de nouvelles références pour l'occasion.

— **Une méthode de compression supprimant des noyaux de CNN :** De nombreuses méthodes de compression sont utilisées après l'apprentissage afin de compresser des modèles, nécessitant souvent de nombreuses étapes supplémentaires de fine-tuning. Afin de palier à cela, nous proposons une nouvelle méthode afin de compresser des modèles d'apprentissage profond de type CNN pendant l'apprentissage. Nous avons effectué cela en évaluant l'importance des noyaux des CNN via une pseudo-norme basée sur les normes simples l_1 et l_2. La pseudo-norme que nous avons créée permet aussi d'induire de la dispersion entre les noyaux d'un modèle, permettant de transférer de l'information entre les noyaux. En d'autres termes certain noyaux vont posséder beaucoup d'information essentielle au bon fonctionnement du modèle alors que d'autres n'en posséderont quasiment aucune, entraînant la suppression de ces derniers pendant l'apprentissage. Nous montrons que sur des modèles et des jeux de données classiques de classification, la méthode que nous avons développée performe aussi bien, voire mieux, que d'autres méthodes de l'état de l'art.

— **Une méthode jointe de compression et de spécification sur des sous-classes de jeux de données :** Dans certains cas, nous n'avons pas besoin de toutes les classes d'un jeu de données mais seulement de certaines sous-classes de ce jeu de données. Par exemple dans une scène routière, nous ne voulons détecter que les camions ou sur une segmentation de parties du visage, nous sommes seulement intéressés par une segmentation du nez et non des autres parties du visage. Or les principaux modèles sont souvent déjà entraînés sur l'ensemble des classes des jeux

de données. Ainsi, en utilisant notre contribution permettant de supprimer des noyaux, nous avons développé une méthode permettant à la fois de ré-entraîner un modèle sur des sous-classes d'un jeu de données sur lequel il a déjà entraîné et à la fois de le compresser. Cela permet que ce modèle soit (1) spécialisé sur des sous-classes et (2) compressé par la même occasion car nécessitant moins de paramètres pour fonctionner sur des sous-classes plutôt que sur le jeu de données entier. Nous montrons que cette méthode est efficace sur des modèles et des jeux de données connus de classification, segmentation et détection.

— **Une démonstration d'analyse de visages pour des applications mobile en réalité augmentée en temps réel :** Nous présentons une démonstration d'analyse de visages pour des plateformes mobiles comme un iPhone ou des appareil Android. Nous avons construit un modèle FCNN sous forme de sablier qui est adapté pour effectuer de l'analyse de visages en temps réel sur téléphones mobiles. Le modèle est implémenté sur iPhone en utilisant le framework *CoreML*. Afin de visualiser les résultats de segmentation, nous superposons un masque avec des couleurs faussées pour que les utilisateurs puissent profiter immédiatement d'une expérience en réalité augmentée.

1.4 Organisation du manuscrit

La reste de cette thèse est organisé en six chapitres en fonction de nos principales contributions :

— ***Chapitre 2 :*** Une revue de la littérature concernant les récents développements des CNN, la compression de réseaux de neurones et la construction d'architectures optimisées.

— ***Chapitre 3 :*** Les contributions concernant nos méthodes de compression supprimant des noyaux de CNN durant l'apprentissage.

— ***Chapitre 4 :*** Nos premiers résultats concernant les méthodes de compression et de spécialisation sur des sous-classes de jeux de données.

— ***Chapitre 5 :*** Une démonstration d'analyse de visages pour des applications mobile en réalité augmentée en temps réel.

— ***Chapitre 6 :*** Les conclusions et les perspectives pour de futurs travaux de recherche.

Chapitre 2

Etat de l'art

Cette thèse s'intéresse essentiellement à la compression de modèles profonds de type CNN. Cependant, les méthodes présentées ici peuvent aussi s'appliquer à d'autres types de réseaux profonds plus simples comme les MLP ou des architectures plus complexes tels que les réseaux récurrents comme les Long Short-Term Memory (LSTM) ou les Gated Recurrent Unit (GRU) [Chuangxia et al., 2013, Chuangxia et al., 2016]. Ce chapitre présente une revue de l'état de l'art des différentes méthodes permettant de compresser et d'accélérer des modèles d'apprentissage profond mais aussi des méthodes s'intéressant à la construction et à la recherche d'architectures optimisées.

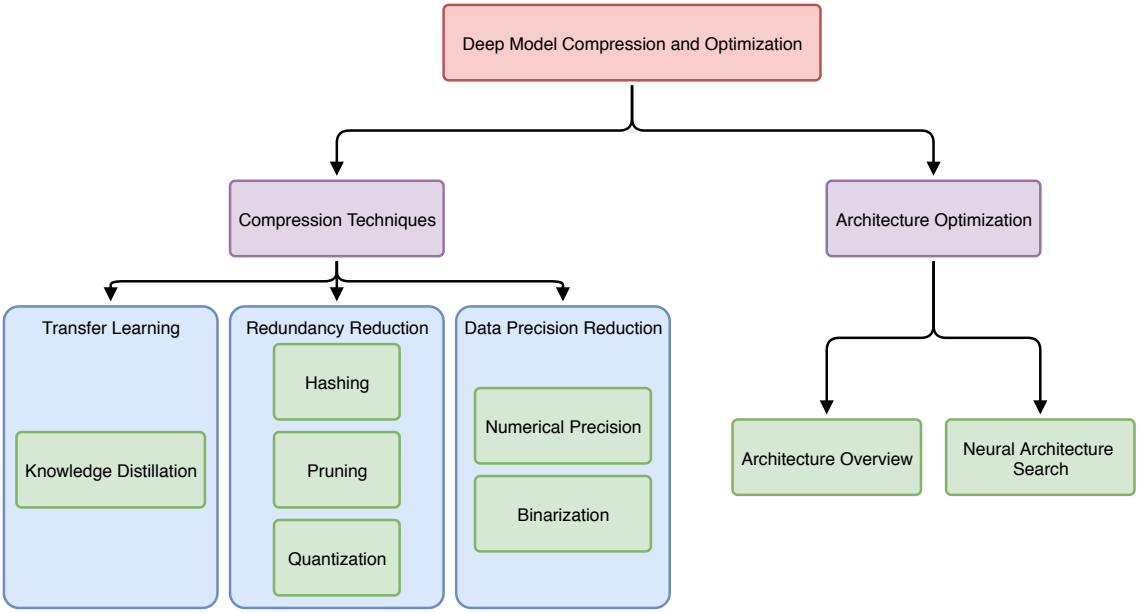


FIGURE 2.1 – Plan de l'état de l'art.

Avant toute chose et comme nous nous intéressons avant tout aux CNN, nous présentons dans une première partie une revue des principes de base des réseaux de neurones, en particulier des CNN, et de leurs évolutions 2.1. Dans une deuxième partie, les principales techniques de compression 2.2 visant à réduire la taille et l'exigence en ressources de calcul des modèles au travers de différents algorithmes. Les méthodes de *knowledge distillation* sont d'abord abordées pour discuter notamment de l'apprentissage par transfert 2.2.1. Les techniques de hachage 2.2.2, d'élagage 2.2.3 et de quantification 2.2.4 sont ensuite présentées, toutes permettant de diminuer la redondance des paramètres des modèles. Les algorithmes permettant de diminuer la précision des paramètres comme la précision numérique 2.2.5 ou la binarisation 2.2.6 sont enfin abordés pour conclure cette section sur les méthodes de compression. La deuxième partie de cet état de l'art

est consacrée à l'optimisation de l'architecture de modèles profonds 2.3. Nous commençons par présenter différentes implémentions architecturales permettant d'optimiser les modèles et comment les modules interagissent entre eux pour construire un design intelligent 2.3.1. Dans un deuxième temps, nous présentons un état de l'art des méthodes permettant la recherche et la construction automatique de modèles profonds optimisés 2.3.2. La structure de ce chapitre est détaillée dans la Figure 2.1.

2.1 Évolution des réseaux de neurones

2.1.1 Le perceptron multicouche

Un perceptron multicouche, ou Multilayer Perceptron [Rumelhart et al., 1988] comme on le trouve plus souvent dans la littérature, est un réseau de neurones à propagation avant composé d'unités (où neurones) que l'on appelle perceptrons [Rosenblatt, 1961]. Ce modèle est né en prenant exemple sur le système nerveux humain biologique, composé de neurones émettant un signal de sortie en fonction des signaux reçus par d'autres neurones. Dans un MLP, les neurones sont organisés en couches (comme on peut le voir sur la figure 2.2) avec une couche d'entrée, une ou plusieurs couches intermédiaires dîtes "cachées" et une couche de sortie. Dans chaque couche, tous les neurones sont connectés avec tous les neurones de la couche précédente. Lorsque que l'on a ce type de connexion entre les couches, on parle aussi de couches entièrement connectées (Fully Connected).

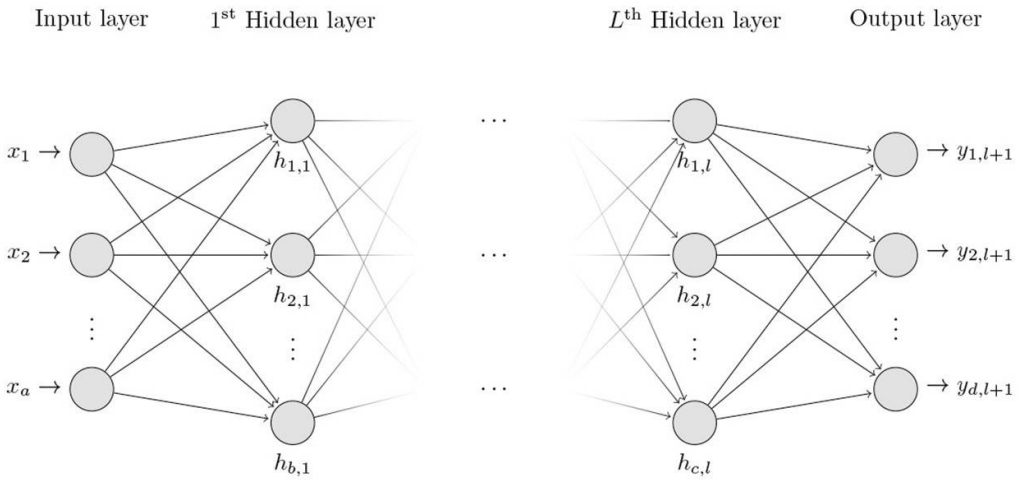


FIGURE 2.2 – Structure d'un MLP. Il est composé d'une couche d'entrée avec plusieurs entrées x, de l couches cachées h contenant chacune un certain nombre de neurones (perceptrons) et d'une couche de sortie contenant aussi plusieurs neurones y. Figure tirée de [Castro et al., 2017]

D'un point de vue mathématique, chaque neurone est modélisé comme suit :

$$f(\mathbf{x}) = \sigma\left(z(x)\right) = \sigma\left(\sum_i w_i x_i + b\right) = \sigma\left(w^T \mathbf{x} + b\right) \tag{2.1}$$

où $\mathbf{x}$ est l'entrée de la couche, w et b sont les poids et les biais des paramètres de cette couche avec i entrées. La fonction d'activation σ ajoute de la non-linéarité à la sortie de chaque neurone. Les fonctions d'activation usuelles sont ici la fonction sigmoide ou tanh.

Afin d'entraîner un réseau de neurones, il est nécessaire de rétro-propager l'erreur de la couche de sortie vers les couches intermédiaires. La non-linéarité introduite par

la fonction d'activation fait en sorte que l'optimisation du réseau de neurones est non convexe. Ainsi, l'apprentissage du réseau est principalement basé sur une méthode de descente de gradient itérative minimisant la fonction de coût [Rumelhart et al., 1988]. La méthode d'optimisation la plus utilisée pour les réseaux de neurones est sans doute la descente de gradient stochastique [Bottou, 2010]. Une simple mise à jour de la descente de gradient stochastique sur le réseau (dont les paramètres sont notés $\boldsymbol{\theta}$, incluant w et b) peut être définie comme suit :

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta_k \nabla f\left(\boldsymbol{\theta}_k\right), \tag{2.2}$$

où η_k est le taux d'apprentissage à l'étape k. Les paramètres d'un réseau de neurones ne sont appris qu'après de nombreuses itérations.

2.1.2 Réseau neuronal convolutif

Les premières apparitions de CNN remontent au milieu des années 70 et sont liées aux travaux de Fukushima [Fukushima, 1975]. Néanmoins, ils ont plutôt connu une première mise en avant avec les travaux de Lecun et al. [LeCun, 1989, LeCun et al., 1990a, LeCun et al., 1998] proposant un réseau neuronal convolutif permettant la reconnaissance de chiffres écrits à la main (voir la structure du réseau sur la Figure 2.3). Ils proposent aussi un apprentissage de bout en bout utilisant un algorithme de rétro-propagation. Cette technique est toujours utilisée actuellement.

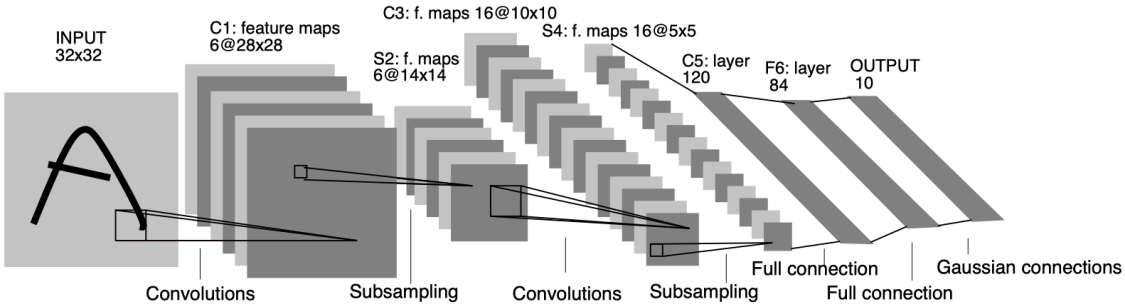


FIGURE 2.3 – Structure du réseau de neurones convolutif Lenet-5 pour la reconnaissance de chiffres manuscrits. Figure reprise de [LeCun et al., 1998].

En comparaison avec les MLP, les CNN incorporent des couches de convolution. Une couche de convolution permet aux poids du réseau de neurones d'être partagés à travers l'espace de dimension de son entrée. Cette opération s'est prouvée être bénéfique pour les tâches de vision par ordinateur grâce à une meilleure efficacité et généralisation. En effet, une opération de convolution s'exécute sur des entrées en deux dimensions, permettant à un réseau d'apprendre des caractéristiques visuelles importantes tout en maintenant les relations d'espaces entre les cartes de caractéristiques. Dans un CNN, les couches de convolution proches de l'entrée du réseau apprennent à reconnaître des caractéristiques générales comme des formes, des textures ou des couleurs, alors que les couches de convolution proches de la fin du réseau vont apprendre des caractéristiques plus précises comme la présence de certains objets.

Bien que ce type de réseaux a été découvert depuis longtemps, ils ne connaissent une réelle popularité qu'à partir de 2012. En effet, cette année là apparaît le réseau AlexNet [Krizhevsky et al., 2012] et apporte avec lui un regain d'intérêt pour les réseaux de neurones par la communauté. Les performances du modèle surpassent celles de tous les autres modèles qui ne sont pas des modèles d'apprentissage profond. L'apprentissage

profond connaît alors un nouveau souffle à partir de ce point et de nombreuses améliorations sont apportées à ces réseaux.

Comme nous l'avons vu précédemment, les premiers travaux utilisant des CNN sont principalement basés sur l'architecture Lenet. Les applications et réseaux restent donc assez condensés, ces derniers ne dépassant pas les 6 ou 7 couches au maximum. Néanmoins avec la démocratisation des calculs sur GPU et des modèles d'apprentissage profond, les réseaux deviennent de plus en plus profond. En conséquence, les performances des modèles sont aussi grandement améliorées. La figure 2.4 montre une comparaison de la précision de différents CNN sur le jeu de données ImageNet par rapport au nombre d'opérations qu'effectuent ces modèles. On remarque ainsi que plus le modèle est grand et dois effectuer d'opérations, plus sa précision augmente. Il est néanmoins nécessaire de nuancer ce résultat qui est surtout vrai lorsque les modèles peuvent s'appuyer sur des jeux de données de tailles conséquentes comme ImageNet [Krizhevsky et al., 2012] par exemple. C'est aussi une réalité qui commence à être faussée grâce à de nouvelles méthodes de constructions automatiques dont nous discuterons plus tard dans ce manuscrit. De plus, nous essayons de nous battre dans cette thèse contre cette tendance en essayant d'obtenir des modèles plus petits mais tout aussi précis.

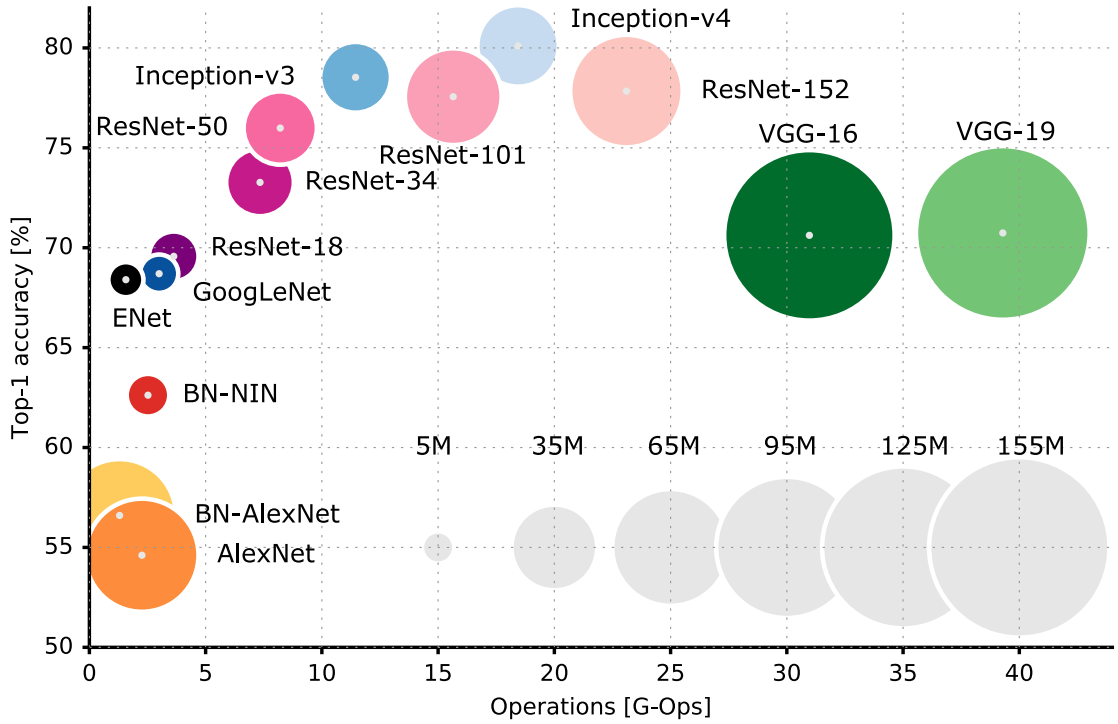


FIGURE 2.4 – Une comparaison de la précision de différents CNNs par rapport au nombre d'opérations qu'ils doivent effectuer pour classifier le jeu de données ImageNet.

De plus, les réseaux ne sont plus pensés comme des modèles linéaires avec des couches alignées les unes à la suite des autres. Des structures plus flexibles émergent comme celles faisant intervenir des sauts de connexions (*skip connections*) [He et al., 2016] ou des connexions denses (*dense connections*) [Huang et al., 2017]. Ces deux améliorations permettent au gradient de circuler dans des réseaux très profonds tout en évitant des problèmes liés à la perte d'informations. Enfin, les réseaux de neurones entièrement convolutifs ou Fully Convolutional Neural Network (FCNN) ont aussi émergés et ont été appliqués à tout un ensemble d'applications. Un FCNN est un modèle qui ne possède pas de couches FC. Il est donc capable de garder les informations spatiales des différentes cartes

de caractéristiques à travers tout le réseau tout en maintenant une capacité de stockage relativement basse. Bien que les FCNN ont été introduit pour des tâches de segmentation [Long et al., 2015], ils ont été utilisés par la suite sous forme d'encodeur-décodeur et appliqués à de nombreuses applications comme la génération d'images [Goodfellow et al., 2014], l'estimation de la pose [Wei et al., 2016], la reconstruction 3D [Feng et al., 2018]...

Au delà de ces évolutions de structure, des améliorations ont aussi été faites afin d'améliorer la régularisation des CNN pour limiter le sur-apprentissage. La méthode de Dropout [Srivastava et al., 2014] permet de laisser de coté certaines unités (neurones) et leurs connexions respectives pendant l'apprentissage, évitant ainsi aux différents neurones de trop se familiariser et de se co-apprendre entre eux. La normalisation par lots (ou *batch normalisation*) [Ioffe and Szegedy, 2015] a aussi été introduite pour stabiliser, accélérer l'apprentissage et régulariser les réseaux en normalisant l'entrée de la couche pour chaque lot (ou *batch*). Plus proche de nous, la normalisation de groupes (*group normalisation*) [Wu and He, 2018] remplace la normalisation par lots quand la taille du lot (*batch size*) est petite.

Les algorithmes d'optimisation ont aussi évolués avec les CNN. Leur apprentissage a ainsi pu être accéléré grâce à des gradients adaptatifs comme Adam [Kingma and Ba, 2014] ou RMSProp [Hinton et al., 2012]. Cependant, même si ces optimiseurs accèlerent l'apprentissage des CNN, il est toujours difficile de dire que ces optimiseurs sont capables de converger vers des solutions similaires. En effet, certains papiers tel que [Wilson et al., 2017] ont montré que les solutions trouvées par ces méthodes adaptatives ne sont pas capables de généraliser aussi bien qu'une méthode de descente de gradient stochastique.

Enfin, les fonctions d'activation ont aussi été sujettes à plusieurs améliorations ces dernières années. La fonction ReLU a été utilisée avec AlexNet [Krizhevsky et al., 2012] pour palier au problème de disparition du gradient auquel on peut être confronté en utilisant une simple fonction sigmoide avec un CNN. Cependant, comme toutes les valeurs négatives sont amenées à zéro, la dérivée de cette fonction prendra aussi une valeur zéro pour toute entrée ayant une valeur négative. Un phénomène appelé "*dead ReLU problem*" peut apparaître chez les CNN quand certains composants du réseau ne sont jamais mis à jour avec de nouvelles valeurs. Afin de palier ce problème, les fonctions d'activations Leaky ReLU [He et al., 2015] et ELU [Shah et al., 2016] ont été introduites. Utiliser des fonctions d'activations permet aux valeurs du gradient de ne plus être bloquées lors de valeurs négatives. Les formules mathématique de ces différentes fonctions sont listées ci-dessous et leurs comportements peuvent être visualisés sur la Figure 2.5.

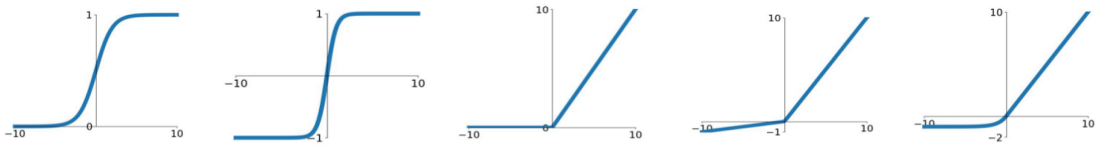
— **Sigmoid :** $\text{Sigmoid}(z) = \dfrac{1}{1+e^{-z}}$

— **Tanh :** $\text{Tanh}(z) = \tanh(z)$

— **ReLU :** $\text{ReLU}(z) = \max\{0, z\}$

— **Leaky ReLU :** $\text{LReLU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases}$

— **ELU :** $\text{ELU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha\left(e^{z}-1\right) & \text{if } z \leq 0 \end{cases}$



(A) Sigmoid (B) Tanh (C) ReLU (D) Leaky ReLU (E) ELU

FIGURE 2.5 – Représentation visuelle du comportement de différentes fonctions d'activation pour un CNN profond.

Le principal objectif des CNN et de ses nombreuses évolutions au cours des dix dernières années a surtout été d'améliorer la précision de ces modèles souvent au détriment de leur efficacité ou de leur taille. Néanmoins, de nombreux travaux ont émergé au fil du temps s'intéressant à la compression et à l'optimisation des CNN. Nous allons explorer ces travaux dans les prochaines pages.

2.2 Techniques de compression

Nous avons vu dans la section précédente les bases des réseaux de neurones et plus particulièrement des CNN. Nous allons désormais nous intéresser à explorer la littérature autour du sujet de cette thèse : comment obtenir des modèles d'apprentissage profond plus petits et rapides tout en ayant la même efficacité que des modèles très profonds ?

2.2.1 Apprentissage par transfert

Lors de la construction d'un réseau de neurones, une des données les plus importantes à prendre en compte est la profondeur que celui-ci doit avoir. Un réseau de neurones classique est composé d'une entrée, d'une sortie et d'un nombre de couches intermédiaires variant d'un réseau à l'autre. Un réseau de neurones peu profond est un réseau avec peu de couches intermédiaires au contraire d'un réseau de neurones profond. Ainsi, un réseau de neurones profond a plus de paramètres et peut potentiellement apprendre des fonctions plus complexes comme des représentations hiérarchiques précises [Dauphin and Bengio, 2013]. Les travaux théoriques provenant de [Dauphin and Bengio, 2013] ont permis de montrer les difficultés qu'impliquaient d'entraîner un réseau de neurones peu profond en espérant obtenir la même précision qu'un réseau plus complexe pour une même tâche. Par exemple, des tentatives ont été effectuées dans le but de classifier la base de données Imagenet [Krizhevsky et al., 2012] en utilisant un réseau avec un nombre de couches très limité. Les auteurs ont conclu que la tâche était trop complexe à apprendre pour un réseau peu profond démarrant avec aucunes connaissances. Le réseau n'arrivant jamais à s'approcher de la précision d'un réseau plus profond sur cette tâche de classification [Dauphin and Bengio, 2013].

Malgré cela, Ba et al. [Ba and Caruana, 2014] ont réussi à montrer par d'autres moyens que des réseaux de neurones avec une architecture peu profonde étaient capables d'apprendre la même fonction que des neurones avec beaucoup plus de couches. Parfois même en obtenant une meilleure précision et en gardant un nombre de paramètres similaire (voir Figure 2.6). Inspirée par [Buciluǎ et al., 2006], leur méthode de compression consiste à entraîner un réseau condensé afin qu'il copie la fonction apprise par un modèle plus complexe. C'est ce que l'on appelle l'apprentissage par transfert : transférer les connaissances apprises par un modèle vers un autre modèle. Afin d'arriver à ce résultat,

la première étape est d'entraîner un réseau profond (que l'on appellera le modèle "enseignant") sur un jeu de données spécifique. Une fois ce réseau entraîné, il va s'agir de lui faire étiqueter des données non étiquetées afin de créer un nouveau jeu de données synthétique. Ce jeu de données aura donc été généré par ce réseau profond "enseignant". Ensuite, ce jeu de données synthétique est utilisé afin d'entraîner un plus petit modèle (que l'on appellera modèle "élève"), afin qu'il assimile la fonction qui a été apprise par le modèle "enseignant" plus profond. En utilisant cette méthode, le modèle "élève" doit produire les mêmes prédictions et erreurs que le modèle "enseignant". Ainsi, Il est techniquement possible d'atteindre une précision similaire entre un ensemble de réseaux de neurones et leur modèle "élève" tout en ayant 1000 fois moins de paramètres. Dans [Ba and Caruana, 2014], les auteurs ont démontré cette théorie sur le jeu de données CIFAR-10. Un ensemble de CNN a été utilisé afin d'étiqueter des données non étiquetées du jeu de données. Ensuite, ces nouvelles données ont été utilisées afin d'entraîner un modèle peu profond ne possédant qu'une seule couche de convolution et de maxpooling suivi d'une couche entièrement connectée composée de 30k unitées non linéaires. Au final, l'ensemble de CNN et le modèle peu profond atteignent tous les deux le même niveau de précision. Plusieurs améliorations ont été apportées à cette méthode de modèles enseignant-élève, notamment grâce aux travaux de Hinton et al. [Hinton et al., 2014]. Leur *framework* intègre la sortie du réseau enseignant afin de pénaliser le réseau élève. De plus, il est aussi capable de récupérer un ensemble de réseaux enseignants afin de compresser leurs connaissances dans un réseau élève ayant une profondeur similaire.

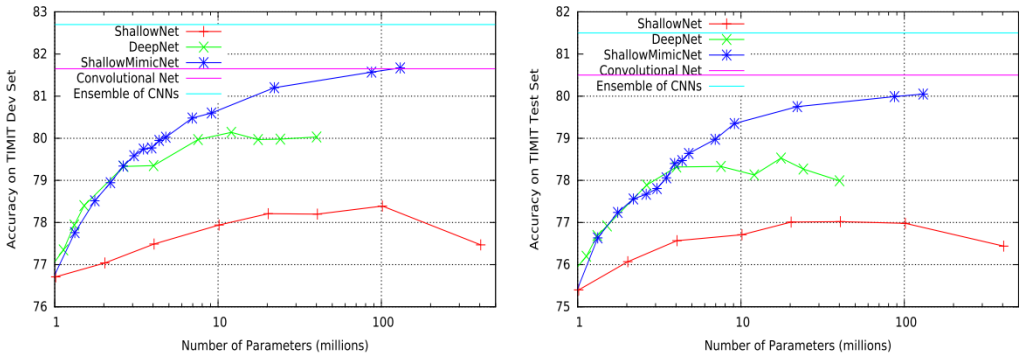


FIGURE 2.6 – Précision de différents réseaux de neurones profonds "enseignant" et réseaux de neurones peu profond "élèves" en fonction de leur nombre de paramètres sur le jeu de données de reconnaissance de parole TIMIT Dev (gauche) et Test (droite). Résultats et figures tirés de [Ba and Caruana, 2014].

Ces dernières années, d'autres méthodes de compression qui sont répertoriées dans cet état de l'art sont préférées. Cependant, l'apprentissage par transfert est une méthode qui se couple efficacement avec d'autres méthodes de compression afin d'obtenir des performances encore plus poussées. Par exemple, les travaux de Chen et al. [Chen et al., 2017] et de Huang et al. [Huang and Wang, 2017] suivent cette approche en la couplant avec leurs méthodes d'élagage (voir la section 2.2.3). En effet, les premiers utilisent cette méthode comme métrique d'apprentissage dans leur modèle alors que les deuxièmes gèrent la problématique enseignant-élève comme un problème de correspondance de distribution en essayant de faire correspondre des schémas de sélection d'activités de certains neurones entre eux pour améliorer les performances globales. Aguilar et al. [Aguilar et al., 2020] proposent de distiller les représentations internes d'un modèle "enseignant" dans une version simplifiée de lui-même afin d'améliorer l'apprentissage et les performances du modèle "élève" par la suite. Lee et al. [Lee et al., 2020] utilisent un algorithme

d'apprentissage auto-supervisé pour améliorer la méthode d'apprentissage par transfert. L'ensemble de ces méthodes est efficace. Cependant leurs performances peuvent varier de manière importante selon l'application visée. Les tâches de classification sont par exemple faciles à apprendre pour un modèle peu profond, mais des tâches de segmentation ou de suivi seront plus difficiles à appréhender, même pour des modèles bien plus profond. De plus, Muller et al. [Müller et al., 2019] ont récemment montré grâce à des expériences utilisant un lissage de l'étiquetage que les réseaux "enseignants" et "élèves" sont sensibles aux formats de données. Améliorer les méthodes d'apprentissage par transfert n'est ainsi pas qu'un travail d'amélioration des algorithmes mais est aussi dépendant de la sélection et du tri des données.

2.2.2 Hachage

Le hachage est une méthode employée afin de regrouper les paramètres dans un réseau de neurones pour, d'une part, éviter les redondances et, d'autre part, accéder plus rapidement aux données. A travers des études empiriques, les méthodes de hachage se sont surtout montrées être efficaces pour les problématiques de réduction dimensionnelle [Weinberger et al., 2009].

Par exemple, HashedNets [Chen et al., 2015] est un réseau utilisant une méthode de hachage développée par Nvidia. Dans ce modèle, une fonction de hachage est utilisée pour regrouper de manière aléatoire et uniforme les connexions des modèles dans des seaux de hachage. Ainsi, chaque connexion présente dans le i^{me} seau de hachage aura la même valeur w_i. Cette technique est particulièrement efficace sur des réseaux de neurones entièrement connectés. De plus, cette méthode peut aussi être facilement utilisable en complément d'autres méthodes de compression.

Plusieurs autres méthodes de hachages ont aussi été développées ces dernières années. Spring et al. [Spring and Shrivastava, 2017] ont proposé une approche utilisant du *dropout* adaptatif [Ba and Frey, 2013] (c'est à dire choisir des noeuds avec une probabilité proportionnelle à la valeur de leur fonction d'activation) et des tables de hachage basées sur un hachage sensible à la localité (Locality-Sensitive Hashing (LSH)) [Gionis et al., 1999, Shinde et al., 2010, Sundaram et al., 2013, Huang et al., 2015]. La combinaison de ces techniques a permis aux auteurs de construire une structure intelligente afin de maximiser la recherche de factorisation interne [Shrivastava and Li, 2014]. Cette technique montre de bons résultats, réduisant les coûts de calculs pendant les étapes d'entraînements mais aussi pendant l'inférence. De plus, ce genre de structure de modèle amène des mises à jour de gradient éparses permettant de développer des modèles asynchrones. Ainsi, les modèles peuvent être facilement parallélisables étant donné la possible plus grande dispersion des données. Cependant, une dispersion trop large peut aussi résulter en un ralentissement du modèle. Un compromis entre ces paramètres est donc nécessaire pour obtenir un modèle efficace.

2.2.3 Élagage

Parmi toutes les méthodes de compression de réseaux de neurones existantes, les techniques d'élagages sont celles qui ont été le plus étudiées. Le principe de ces techniques est de supprimer les paramètres d'un modèle qui sont jugés comme non nécessaires à la bonne inférence du réseau. Les premiers travaux dans le domaine visaient à réduire la complexité et le sur-apprentissage des réseaux [LeCun et al., 1990b, Hassibi and Stork, 1993]. Dans ces articles, les auteurs utilisent des algorithmes d'élagages basés

sur la Hessienne de la fonction de coût. Cela a pour conséquence de réduire le nombre de connexions à l'intérieur d'un réseau. La méthode trouve un ensemble de paramètres dont la délétion va entraîner le moins d'augmentation possible de la fonction objective en mesurant la saillance des paramètres. Pour se faire, les auteurs utilisent de nombreuses approximations. La fonction objective est par exemple approximée par une série de Taylor. Trouver les paramètres dont la délétion n'augmente pas la fonction objective est un problème difficile qui engendre, en règle générale, le calcul de matrices de tailles importantes ainsi que de nombreuses dérivées. Par ailleurs, ces méthodes suggèrent que réduire le nombre de poids via des méthodes de calculs de Hessienne de la fonction de coût serait plus performant que des méthodes d'élagages basées sur des ordres de grandeur tel que le *weight decay*. De plus, cela réduirait le sur-apprentissage du réseau ainsi que sa complexité. Néanmoins, le calcul de dérivées du second ordre concernant la série de Taylor introduit une complexité de calcul supplémentaire non négligeable.

Signorini et al. [Han et al., 2015] ont utilisé une méthode simple, intuitive et efficace afin de supprimer des paramètres. La première étape est d'apprendre les différentes connexions du réseau via un entraînement traditionnel de ce dernier. Cela permet dans un premier temps d'apprendre quels paramètres (ou connexions) sont plus importants que les autres. L'étape suivante consiste à élaguer (ou supprimer) les connexions dont la valeur des poids se situe en dessous d'un certain seuil. Cette étape de suppression permet aussi de convertir un réseau dense en un réseau éparse. La dernière étape de cette méthode, qui est aussi la plus importante, est de ré-entraîner le réseau (*fine-tuning*) afin de ré-apprendre les poids des connexions qui n'ont pas été supprimées. En effet, grâce à cette dernière étape, les poids du réseau sont réajustés afin de compenser la perte de certaines connexions. Sans cette étape, le réseau aurait une précision qui ne cesserait de chuter. Afin de mieux visualiser la méthode, les différentes étapes pour élaguer un modèle sont présentées dans la Figure 2.7.

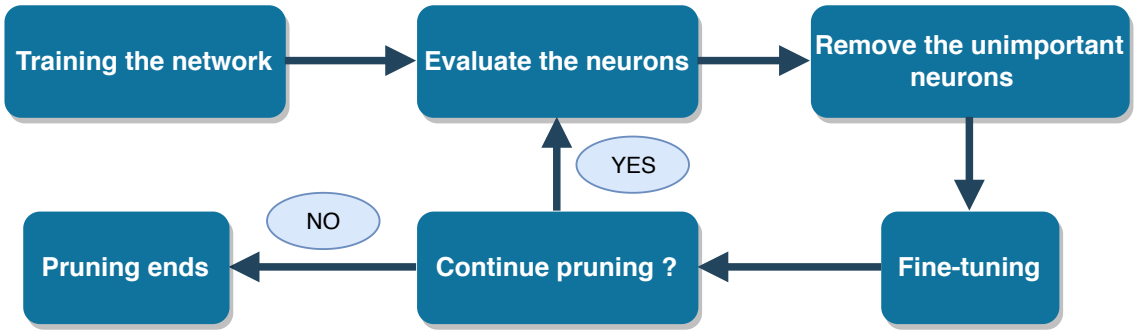


FIGURE 2.7 – Etapes de base afin d'élaguer un réseau de neurones profond. Figure inspiré par [Molchanov et al., 2017].

Anwar et al. [Anwar et al., 2017] utilisent une méthode similaire. Cependant, ils estiment aussi que l'élagage a le désavantage de créer des connexions irrégulières, dues au coté éparse des connexions restantes, pouvant entraîner des problèmes pour la parallélisation des calculs. Afin de contourner ce problème, les auteurs ont introduis une dispersion structurée à différents niveaux d'un CNN. Ainsi, l'élagage est effectué au niveau des cartes de caractéristiques (*feature map*) et des noyaux (*kernels*). L'idée est de forcer certains poids à zéro tout en créant de la dispersion dans les activations à des endroits définis du réseau. Pour se faire, la technique consiste à contraindre chaque connexion de sortie de convolution à avoir, pour une carte de caractéristique donnée, le même déplacement (*stride*) et décalage (*offset*). Cela a pour conséquence de réduire grandement la taille des

matrices des cartes de caractéristiques et des noyaux. Pour approfondir, ces problèmes de dispersion ont été étudiés dans de nombreux travaux ayant aussi pour but de pénaliser des paramètres jugés comme non-essentiels [Wen et al., 2016, Zhou et al., 2016, Alvarez and Salzmann, 2016, Lebedev and Lempitsky, 2016].

Une approche similaire d'élagage est aussi développée chez Molchanov et al. [Molchanov et al., 2017]. Cependant, des critères différents d'élagages ainsi que d'autres considérations techniques sont introduites pour supprimer des cartes de caractéristiques et des noyaux, comme par exemple le critère de poids minimum [Han et al., 2015]. Les auteurs partent du principe que si une valeur d'activation (ici la sortie d'une carte de caractéristiques) est faible, alors le détecteur de caractéristiques issue de cette carte n'est pas important pour l'application. Un autre critère impliqué dans cette méthode est l'information mutuelle qui mesure la quantité d'information qu'une variable possède à propos d'une autre. De plus, le développpement de Taylor est aussi utilisé, à la manière de LeCun [LeCun et al., 1990b], afin de minimiser le coût de calcul entre le réseau élagué et le réseau non élagué. Dans ce cas, l'élagage est alors traité comme un problème d'optimisation.

Une méthode d'élagage plus récente [Li et al., 2017] a pour but de supprimer les filtres qui ont un impact limité sur sur la précision finale du réseau. Cela a pour effet de supprimer automatiquement les cartes de caractéristiques et les noyaux liés à ce filtre dans la couche suivante. L'importance relative d'un filtre dans chaque couche est mesurée en calculant la somme de la valeur absolue de ses poids, ce qui permet d'obtenir une approximation de l'ordre de grandeur de la sortie de la carte de caractéristiques. A chaque itération, les filtres avec les plus petites valeurs sont ainsi supprimés. Jian-Hao et al. [Luo et al., 2017] ont développé une méthode d'élagage appelée ThiNet qui, au lieu de se servir de l'information d'une couche pour supprimer des filtres de cette même couche, va utiliser des informations et des statistiques effectuées sur la couche suivante afin de supprimer des filtres sur la couche actuelle. On peut aussi rajouter qu'il n'y a pas que les poids et les filtres qui peuvent être supprimés mais aussi des canaux entiers [Liu et al., 2017] en utilisant des méthodes de seuils plus complexes.

Ces dernières années, de nombreux algorithmes de compression de réseaux de neurones utilisant des méthodes d'élagages et atteignant des performances dignes de l'état de l'art ont émergés. Yu et al. [Yu et al., 2018] ont proposé une méthode de score basée sur le niveau d'importance de propagation des neurones (Neurons Importance Score Propagation ou NISP). Ce score est mesuré sur le niveau de réponse des dernières couches afin de mesurer l'impact de l'élagage sur des couches antérieurs. Zhuang et al. [Zhuang et al., 2018] ont développé une fonction de coût discriminante permettant de déterminer quels sont les canaux les plus utiles dans les couches intermédiaires. D'autres méthodes comme l'élagage de filtres via la médiane géométrique (Filter Pruning Via Geometrc Median ou FPGM) [He et al., 2019] ne se concentrent pas sur l'élagage de filtres apportant peu d'information mais plutôt sur ceux apportant trop de redondance. De manière similaire, Lin et al. [Lin et al., 2019] abordent le problème des structures redondantes en proposant une méthode d'apprentissage adverse générative (Generative Adversarial Learning ou GAL) non seulement pour supprimer des filtres, mais aussi pour supprimer des branches et des blocs entiers.

Des méthodes de factorisations peuvent aussi être utilisées comme de la décomposition de tenseurs ou de matrices [Sainath et al., 2013, Kolda and Bader, 2009]. Cependant, la décomposition de certaines opérations peut être très coûteuse en ressources de calculs

et les méthodes de factorisations sont aussi coûteuses en terme de temps comme le modèle a besoin d'être ré-entraîner de nombreuses fois. Nous n'allons donc pas entrer dans le détail de ces méthodes dans cet état de l'art. Néanmoins, un passage en revue de ces différentes techniques peut-être trouvé dans le papier de Cheng et al. [Cheng et al., 2017].

De nombreuses méthodes d'élagages existent et chacune d'entre elles possèdent ses forces et ses faiblesses. Le principal désavantage de ces méthodes étant qu'elles demandent un temps important pour élaguer un réseau, notamment à cause des ré-entraînements constants nécessaires au bon fonctionnement du modèle. Des techniques récentes comme Li et al. [Lin et al., 2017a] tentent de se passer de certaines étapes en élaguant le réseau pendant la phase d'apprentissage en utilisant des réseaux de neurones récurrents. Néanmoins, malgré quelques désavantages, la majorité de ces méthodes permettent une réduction considérable du nombre de paramètres d'un modèle. Les techniques d'élagages permettent d'éliminer de 10% à 30% des poids d'un réseau. Peu importe la méthode choisie, la taille d'un réseau peut toujours être réduite avec de l'élagage sans changement ou décroissance majeur de la précision du modèle. L'inférence du modèle qui en résulte gagnera aussi en rapidité (voir Figure 2.8) mais cette rapidité dépendra toujours de la méthode utilisée et de la dispersion des paramètres du modèle après l'élagage.

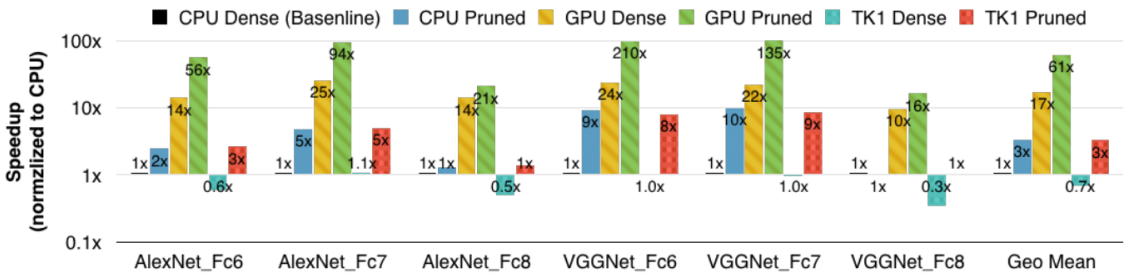


FIGURE 2.8 – Comparaison de la rapidité d'AlexNet et de VGG avant et après élagage sur CPU, GPU et TK1. Figure reprise de [Han et al., 2016].

2.2.4 Quantification

La quantification est, comme l'élagage, une technique de compression très utilisée pour compresser des réseaux de neurones profonds. Le but de cette famille de méthodes est de réduire le nombre de bits requis pour représenter chaque poids. En d'autres termes, le but de la quantification est de réduire le nombre de paramètres d'un réseau en exploitant la redondance de ce dernier. Cela va ainsi permettre de réduire la taille de stockage du réseau tout en limitant au minimum la perte de précision du modèle. Concrètement, dans un réseau de neurones, cela veut dire que les paramètres vont être rassemblés dans des groupes et partager les mêmes valeurs de paramètres au sein du même groupe.

Gong et al. [Gong et al., 2014] ont effectué des études sur des séries de méthodes de quantification vectoriel et ont montré qu'appliquer une quantification scalaire sur la valeur des paramètres en utilisant un simple k-means est suffisant pour compresser la taille d'un réseau de 8 à 16 fois sans faire décroître de manière significative la précision de ce réseau. Quelques années plus tard, Han et al. [Han et al., 2016] ont directement utilisé une méthode simple de classification k-means pour regrouper des paramètres. Ils ont aussi couplé cette méthode de quantification à une étape d'élagage et un algorithme de codage de Huffman afin d'obtenir la compression de modèles la plus poussée possible. Dans leurs expérimentations, les auteurs ont été capables de réduire les demandes

en stockage de différents réseaux de 35 à 49 fois. En effet, les méthodes d'élagages et de quantifications sont des techniques souvent utilisées ensemble afin d'atteindre des niveaux de compressions importants. Par exemple, pour un réseau de type Lenet-5 [LeCun et al., 1998], l'élagage et la quantification permettent d'obtenir un modèle 32 fois plus petit (en terme de stockage). Une étape de codage de Huffman supplémentaire permet même d'augmenter ce nombre de 32 à 40 fois plus petit.

Il est possible d'appliquer différentes méthodes de quantifications sur des réseaux de neurones. Choi Y. et al. [Choi et al., 2017] ont construit une mesure de distorsion basée sur une matrice Hessiennne qui sert de fonction objective afin de réduire localement les coûts de quantification. De plus, un k-means aussi basé sur une Hessienne est utilisé dans un but de quantification pour minimiser les pertes de performances. De récents algorithmes d'optimisation de réseaux de neurones peuvent fournir des alternatives à la Hessienne et ainsi réduire les coûts de calculs, comme par exemple Adam [Kingma and Ba, 2014], AdaGrad [Duchi et al., 2011], Adadelta [Zeiler, 2012] ou RMSProp [Hinton et al., 2012]. Cependant, l'un des avantages des méthodes qui sont basées sur la Hessienne est que tous les paramètres de toutes les couches dans un réseau de neurones peuvent être quantifiés ensemble au même moment. Ceci n'est pas le cas avec des méthodes de quantification plus anciennes qui fonctionnent couche par couche [Han et al., 2016, Gong et al., 2014].

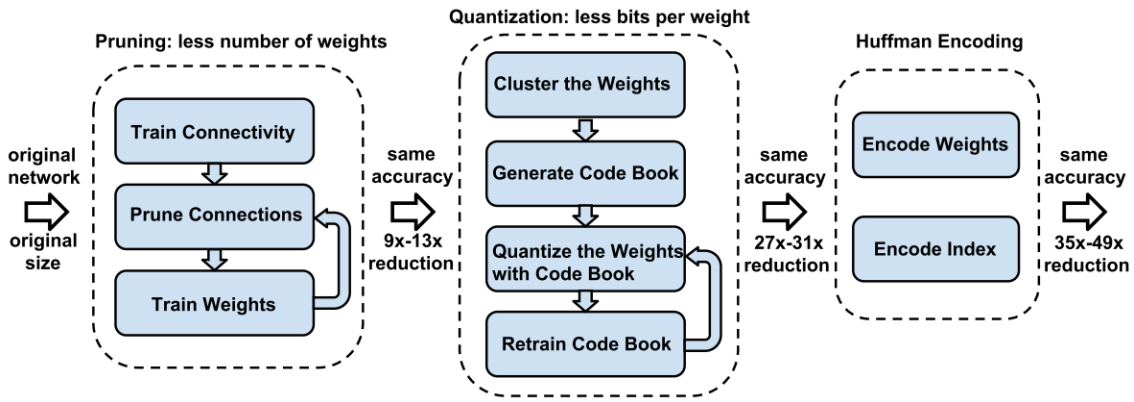


FIGURE 2.9 – Figure reprise de [Han et al., 2016] présentant leur méthode en 3 étapes utilisant élagage, quantification et codage de Huffman. L'accumulation de ces méthodes de compression permettent d'atteindre un taux de compression de plus en plus élevé pouvant aller jusqu'à une compression de 49x du modèle en bout de course. Cette compression se faisant sans perte significative de précision.

Ces techniques de quantification sont efficaces puisqu'elles permettent d'importants taux de compression et peuvent facilement être couplées avec d'autres méthodes (telles que les méthodes d'élagage) pour compresser des modèles de manière plus efficace. Il est d'ailleurs important de noter que des méthodes de quantification sont implémentées dans de nombreux Framework afin d'optimiser les modèles et de les porter sur des appareils mobiles sans aucune intervention supplémentaire de l'utilisateur [Abadi, 2015, Ahmad et al., 2017].

2.2.5 Réduire la précision numérique

Bien que le nombre de poids d'un modèle puisse être considérablement réduit via des méthodes d'élagage ou de quantification, les nombreuses multiplications de matrices de

grandes tailles peuvent toujours être un problème non négligeable pour obtenir un réseau optimisé. Une solution à ce problème est de diminuer la complexité des calculs en limitant la précision numérique des données. Les réseaux de neurones profonds sont généralement entraînés en utilisant une précision de 32-bits en virgules flottantes pour les paramètres et les activations. Le but est de diminuer le nombre de bits utilisés (vers 16, 8 ou même moins) et de changer les représentations en virgules flottantes pour des virgules fixes. Sélectionner la précision des données a toujours été un choix fondamental quand on aborde des systèmes embarqués. Lorsque l'on développe pour un système ou un appareil spécifique, les modèles et les algorithmes peuvent être optimisés pour l'architecture de cet appareil de manière très ciblée [Farabet et al., 2011, Gokhale et al., 2014, Vanhoucke et al., 2011].

Cependant, limiter la précision numérique des réseaux de neurones profonds n'est pas une tâche simple. En effet, des erreurs de précision peuvent facilement être propagées et amplifiées à travers le modèle et avoir un impact important sur ses performances globales. Depuis le début des années 90, des expérimentations ont été faites afin de limiter la précision des données dans un réseau de neurones, particulièrement durant les phases de rétro-propagation. Iwata et al. [Iwata et al., 1989] ont créé un algorithme de rétro-propagation avec des unités de traitement codées en virgules flottantes sur 24 bits. Hammerston [Hammerstrom, 1990] a présenté une architecture pour de l'apprentissage sur puce en utilisant un codage en points fixes sur 8 ou 16 bits. De plus, Holt et Hwang [Holt and Hwang, 1993] ont empiriquement montré qu'un codage de 8 à 16 bits était suffisant pour l'apprentissage de la rétro-propagation. Néanmoins, même si tous ces travaux ont aidé à comprendre l'impact du changement de la précision numérique dans les réseaux de neurones, ils ont été effectués sur des modèles de tailles limitées comme des perceptrons multi-couches avec une seule couche cachée et peu d'unités. Des algorithmes plus sophistiqués ont donc été nécessaires pour passer sur des modèles plus complexes.

En 2015, Gupta et al. [Gupta et al., 2015] ont entrainé des CNN profonds en utilisant une précision en points fixes sur 16 bits à la place d'une précision en virgules flottantes sur 32 bits. Cela permet de contraindre les paramètres d'un réseau de neurones tels que les biais, les poids et les autres variables utilisées pendant la rétro-propagation comme les activations, les erreurs rétro-propagées et les mises à jour des poids et des biais. Différentes expérimentations ont été faites avec cette précision en points fixes sur 16 bits comme faire varier le nombre de bits qui encodent la partie décimale (et entière) entre 8 (8), 10 (6) et 14 (2), respectivement. En d'autres termes, le nombre de bits codant la partie entière IL additionné au nombre de bits codant la partie décimal FL est toujours égal à 16. Testé avec les jeux de données MNIST et CIFAR-10 sur des réseaux entièrement connectés et des réseaux de convolutions, les résultats obtenus étaient semblables à un codage en virgules flottantes jusqu'à la diminution de la partie décimale à 12 bits.

La partie cruciale dans cette méthode est la conversion d'un nombre en virgules flottantes (ou d'un format de précision supérieure) en un niveau de précision inférieure. Pour atteindre cela, [Gupta et al., 2015] décrit deux méthodes d'arrondissement. La première est une méthode d'arrondissement au plus proche. Elle consiste à définir $\lfloor x \rfloor$ comme le plus grand entier multiple de $\epsilon = 2^{-FL}$ plus petit ou égal à x. Ainsi, à partir d'un x et de la précision voulue (IL,FL), l'arrondissement est effectué comme suit :

$$\begin{cases} \lfloor x \rfloor & if \lfloor x \rfloor \leq x \leq \lfloor x \rfloor + \frac{\epsilon}{2} \\ \lfloor x \rfloor + \epsilon & if \lfloor x \rfloor + \frac{\epsilon}{2} \leq x \leq \lfloor x \rfloor + \epsilon \ . \end{cases} \tag{2.3}$$

La seconde méthode d'arrondissement est un arrondissement stochastique. C'est un arrondissement statistique et non biaisé où la probabilité de x d'être arrondi à $\lfloor x \rfloor$ est proportionnelle à sa proximité à $\lfloor x \rfloor$:

$$\begin{cases} \lfloor x \rfloor & w.p. \quad 1 - \dfrac{x - \lfloor x \rfloor}{\epsilon} \\ \lfloor x \rfloor + \epsilon & w.p. \quad \dfrac{x - \lfloor x \rfloor}{\epsilon} \end{cases}. \tag{2.4}$$

Courbariaux et al. [Courbariaux et al., 2014] ont enquêté sur l'impact de la précision numérique, en particulier pour réduire les coût des calculs des nombreuses multiplications. Leurs expérimentations ont été effectuées sur trois formats : virgules flottantes, points fixes [Gupta et al., 2015] et points fixes dynamiques [Williamson, 1991] (qui est un compromis entre les deux premiers formats). Au lieu d'avoir un seul facteur d'échelle avec un nombre fixe pour la partie entière et un autre nombre fixe pour la partie décimale, plusieurs facteurs d'échelles sont partagés entre des variables groupées et sont mis à jour régulièrement. Les auteurs arrivent aux mêmes conclusions que [Gupta et al., 2015] : une précision minime est suffisante pour entraîner et faire fonctionner des réseaux de neurones profonds. Cependant, une précision limitée peut être délicate à implémenter lorsqu'elle est couplée à du matériel spécifique. Gupta et al. [Gupta et al., 2015] sont arrivés à obtenir des bons résultats en couplant la précision en points fixes avec du matériel de type FPGA mais la représentation en points fixes dynamiques n'est pas aussi concluante. Les réseaux de neurones avec des paramètres ayant une précision limitée et leur intégration matérielle est un sujet déja bien étudié. Par exemple, Mamalet et al. [Mamalet et al., 2007] et Roux et al. [Roux et al., 2007] ont développé des CNN optimisés sur systèmes embarqués pour détecter les visages et ses différents traits en temps réel. Ils ont utilisé une précision de paramètres en points fixes mais aussi optimisé l'algorithme d'inférence pour des plate-formes spécifiques. Cela leur permet d'exploiter la parallélisation des calculs et la gestion de la mémoire au maximum pour la plate-forme visée.

Pour conclure, une précision numérique limitée est suffisante pour entraîner des modèles profonds. Il est toujours utile de réduire le stockage mémoire et les coûts des calculs, surtout sur des systèmes embarqués. Cependant, toutes les étapes d'entraînement d'un modèle ne peuvent pas être effectuées avec une précision limitée. Une haute précision est généralement nécessaire afin de mettre à jour les paramètres durant la phase d'apprentissage, obligeant souvent à garder en mémoire une sauvegarde des valeurs réelles des paramètres.

2.2.6 Binarisation

Limiter la précision numérique de 32 à 16 ou 8 bits permet déjà d'obtenir une compression efficace. Cependant, cette méthode a été poussée plus loin jusqu'à du codage binaire fournissant des résultats intéressants. En effet, dans un réseau binaire, les poids et les activations, au minimum, sont contraints de prendre une valeur de $+1$ ou -1. Suivant la même idée que nous avons vue dans la section 2.2.5 sur la précision numérique limitée [Courbariaux et al., 2014], les mêmes auteurs ont décidé d'appliquer deux méthodes d'arrondissements différentes pour transformer une variable normale en une variable binaire. L'une des méthodes est un arrondissement déterministe tandis que l'autre est un arrondissement stochastique [Courbariaux et al., 2016]. L'arrondissement le plus commun est simplement de garder le signe de la variable. Ainsi pour une variable x, sa valeur binaire x^b sera le signe de x ($+1$ si $x \geq 0$, -1 autrement). La seconde méthode de binarisation est une méthode stochastique. Ainsi, $x^b = +1$ avec une probabilité $p = \sigma(x)$

et $x^b = -1$ avec une probabilité $1 - p$ où σ est une fonction sigmoide dure (*hard sigmoid function*) [Courbariaux et al., 2016]. Cette méthode stochastique est difficile à implémenter parce qu'elle requiert de générer aléatoirement des bits directement par le matériel sur lequel elle est implémentée. Par conséquent, la méthode déterministe est la plus utilisée. Cependant, des travaux plus récents comme ceux de Lin et al. [Lin et al., 2017b] se consacrent à trouver des méthodes alternatives pour approximer les valeurs des poids et obtenir des réseaux plus précis. Par exemple, les poids peuvent être approximés en utilisant une combinaison linéaire de plusieurs bases de poids binaires.

Néanmoins, tout comme lorsque l'on limite la précision numérique, garder en mémoire une précision supérieure est nécessaire pour certaines étapes et les valeurs réelles sont requises durant la phase de rétro-propagation. Ajouter du bruit aux poids et aux activations (comme du *dropout* [Srivastava, 2013, Srivastava et al., 2014]) est bénéfique pour la généralisation lorsque le gradient et les paramètres sont calculés. La binarisation peut d'ailleurs aussi être vue comme une méthode de régularisation [Courbariaux et al., 2016].

Grâce à toutes ces observations, Courbariaux et al. [Courbariaux et al., 2015] ont développé une méthode appelée *BinaryConnect* pour entraîner des réseaux de neurones profonds en utilisant des poids binaires pendant les propagations avant et arrière du réseau, tout en gardant en mémoire les vraies précisions des poids afin de pouvoir calculer le gradient. La méthode se déroule en 3 étapes successives :

— La propagation avant : couche par couche, les poids réels sont transformés en poids binaires, permettant au calcul de l'activation des neurones de se faire plus rapidement puisque le passage au binaire transforme les multiplications en additions.

— La propagation arrière : l'apprentissage du gradient objectif (*objective's gradient*) est calculé en fonction du niveau d'activation de chaque couche (en partant de la couche la plus profonde et en remontant couche par couche jusqu'à la première couche cachée).

— La mise à jour des paramètres : les paramètres sont mis à jour en fonction de leur valeur précédente et de leur gradient précédemment calculé. Durant cette étape, une plus grande précision est nécessaire. Ainsi, les valeurs réelles des paramètres sont utilisées (contrairement aux deux premières étapes où les poids ont des valeurs binaires).

Des tests sur des jeux de données tel que MNIST, CIFAR-10 ou SVNH ont été effectués avec cette méthode. Ils ont montré que les modèles utilisés pouvaient atteindre les performances de l'état de l'art tout en réduisant le nombre de multiplications que doit effectuer le réseau de près de deux tiers. De plus, la durée de l'entraînement est aussi accéléré d'un facteur 3 et le besoin en mémoire est au minimum 16 fois inférieur.

Dans un réseau avec des poids binaires, seulement la valeur des poids est approximée avec des valeurs binaires. On parle donc ici de réseaux de neurones entièrement connectés. Cependant, cette méthode fonctionne aussi très bien pour les CNN, où les modèles deviennent significativement plus petits (jusqu'à 32 fois plus petit). En effet, avec des opérations binaires, une opération de convolution peut être simplifiée comme cela :

$$I * W \approx (I \oplus B)\alpha \tag{2.5}$$

où I est l'entrée, W est le filtre contenant les poids avec les valeurs réelles, B est le filtre en binaire ($sign(W)$), α est un facteur d'échelle tel que $W \approx \alpha B$ et $\oplus$ indique une opération de convolution sans multiplication. Des améliorations approfondies ont été faites grâce à la méthode XNOR-Net proposée par Rastegari et al. [Rastegari et al., 2016] où à la fois les poids, les entrées des couches de convolution et les entrées des couches entièrement connectées sont passées en binaire. Dans ce cas là, toutes les opérations de convolution sont binaires, permettant ainsi d'effectuer ces opérations seulement avec des opérations XNOR et de comptage de bits :

$$I * W \approx (sign(I) \oplus sign(W)) \odot K\alpha \tag{2.6}$$

où, I est l'entrée, W est le filtre contenant les poids avec les valeurs réelles et K est composé de tous les facteurs d'échelle pour tous les sous-tenseurs de l'entrée I. Un récapitulatif de cette méthode est schématisé sur la Figure 2.10.

Le réseau résultant de cette méthode [Rastegari et al., 2016] arrive à être aussi précis qu'un réseau non binaire lorsque les poids sont codés sous forme binaire. De plus, le réseau arrive à être deux fois plus rapide tout en étant 32 fois plus petit en mémoire. Si les entrées du réseau sont aussi transformées sous forme binaire en revanche, la précision décroît de plus de 10% mais le réseau est 58 fois plus rapide que sa version non binaire. AlexNet peut être réduit à 7MB par exemple (voir les résultats de la Figure 2.11). Beaucoup de réseaux existants (comme le modèle hourglass [Newell et al., 2016]) ont été améliorés grâce la méthode XNOR-Net afin d'atteindre des performances équivalentes à celles de l'état de l'art [Bulat and Tzimiropoulos, 2017]. Plus récemment, la méthode XNOR-Net a été étudiée afin de la faire évoluer d'une tâche de binarisation à une tâche de ternarisation [Deng et al., 2017]. Les valeurs sont alors contraintes dans un espace ternaire -1, 0, $+1$. Cela permet, entre autres, de supprimer le besoin de garder toujours les valeurs réelles en mémoire durant la phase d'apprentissage grâce à une méthode de discrétisation des valeurs.

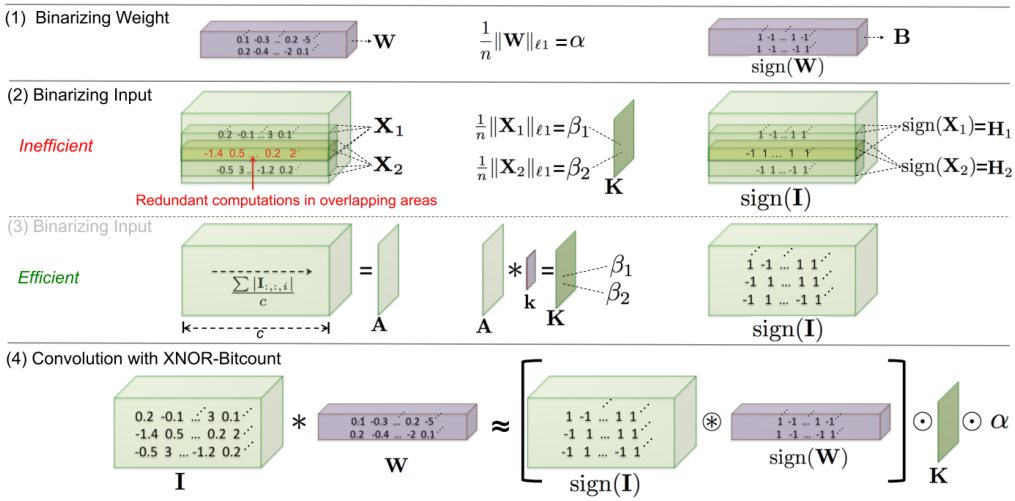
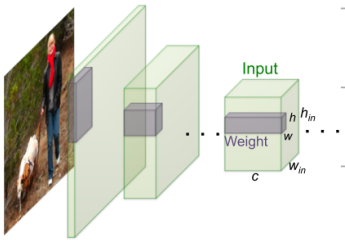


FIGURE 2.10 – Schéma explicatif de la méthode XNOR-Net afin d'approximer des opérations de convolutions en utilisant des opérations binaires. Figure reprise de [Rastegari et al., 2016]



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52 — Real-Value Weights 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52 — Binary Weights 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs 1 -1 ... -1 / -1 1 ... 1 — Binary Weights 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

FIGURE 2.11 – Résultats montrant l'efficacité de la transformation des poids et des entrées en binaires sur le réseau AlexNet et le jeu de données ImageNet. Figure reprise de [Rastegari et al., 2016]

2.2.7 Discussion

Dans la Table 2.1, une comparaison et une synthèse des différentes méthodes de compression de la littérature vues précédemment sont présentées. De cela, on en déduit plusieurs choses. Ces méthodes visent à réduire la taille, le temps de calcul et la mémoire demandée par les modèles profonds. Cependant, beaucoup d'entre elles mènent à des modèles éparses et un modèle trop éparse n'est pas forcément efficace. L'élagage et la quantification sont les méthodes les plus utilisées et parviennent à des résultats impressionnants sur des modèles déjà entraînés. Néanmoins, la dispersion des données du modèle est aussi à prendre en compte lors de l'utilisation de ces méthodes. Cette remarque est aussi valable avec les méthodes de hachage qui sont elles, en revanche, beaucoup moins répandues. Les méthodes changeant la précision numérique (binarisation comprise) ne sont elles pas concernées par ces problèmes de dispersions et permettent de gagner un temps non négligeable sur tous les calculs d'un modèle, surtout si l'architecture est construite intelligemment. Cependant, d'autres problèmes peuvent survenir. Une précision supérieure doit toujours être gardée en mémoire pour certains passages de l'apprentissage et la précision peut grandement varier selon l'application voulue. Au final, compresser un modèle profond, quelle que soit la technique, demandera toujours de faire un compromis entre la précision du modèle et l'efficacité de ses calculs. Cependant, et comme nous allons le voir maintenant, trouver des méthodes pour compresser des modèles n'est pas l'unique moyen de les optimiser. Sa construction et son architecture peuvent aussi être plus intelligemment pensées.

Technique	Méthode	Avantages	Désavantages
Apprentissage par transfert	Utiliser un CNN profond pour entraîner un plus petit CNN.	Obtention de petits modèles avec des performances comparables.	Les modèles ne peuvent être entraîner qu'à partir de zéro. Difficile pour des tâches qui ne sont pas de la classification.
Hachage	Indexer des poids dans une table de hachage.	Meilleur parallélisation ; Meilleur dispersion des données ; Réduit les temps de calcul.	Plus les valeurs des paramètres du modèle sont dispersés, plus ce dernier est lent.
Elagage	Supprimer les poids qui ont une influence mineure sur le modèle.	Accélération du modèle et réduction de sa taille ; Le taux de compression est de l'ordre du facteur 10x à 15x (pouvant même aller à 30x).	Le processus d'élagage prends du temps ; Moins intéressant pour des modèles dont les valeurs des paramètres sont déjà très dispersés.
Quantification	Réduire le nombre de neurones distincts en les rassemblant dans des groupes.	Taux de compression élevé : 10x à 15x ; Peut être couplé avec l'élagage.	Plus les valeurs des paramètres du modèle sont dispersés, plus ce dernier est lent.
Précision Numérique	Décroître la précision numérique des paramètres.	Accélération de l'inférence et taux de compression élevé.	Une plus grande précision est nécessaire durant la mise à jour des paramètres ; Nécessite parfois d'être appliqué sur du matériel spécifique
Binarisation	Décroître la précision numérique des paramètres jusqu'à seulement 2 bits.	Taux de compression élevé (30x) et accélération du modèle (50x à 60x).	Une plus grande précision est nécessaire durant la mise à jour des paramètres.

TABLE 2.1 – Résumé des différentes méthodes de compression vues précédemment.

2.3 Optimisation des architectures

Les méthodes de compression ont été vastement étudiées. Désormais, certaines d'entre elles font même partie intégrantes de *frameworks* d'apprentissages profonds populaires. Tensorflow Lite [Abadi, 2015] possède de nombreux outils pour quantifier des modèles, rendant par exemple des transferts vers des plate-formes mobiles plus simples. Core ML [Ahmad et al., 2017], qui est le framework d'Apple pour faire de l'apprentissage profond sur les appareils de la marque, est aussi capable d'appliquer ce genre de méthodes sur des IPhones par exemple. Ainsi, d'un coté, quelques techniques de compressions sont déjà intégrées dans des logiciels utiles pour les développeurs. Mais, d'un autre coté, nous sommes encore très loin de connaître toutes les particularités des modèles profonds afin de les optimiser au maximum.

Cependant, comme ces méthodes visent à réduire la complexité d'un modèle a posteriori, elles sont généralement appliquées sur des modèles déjà construits. Ainsi, des recherches plus récentes se concentrent directement sur l'architecture de ces modèles et plus précisément sur comment construire des architectures optimisées dès le départ plutôt que de trouver des méthodes pour les réduire en bout de chaîne. Cette section de ce chapitre de l'état de l'art s'intéresse à ces méthodes. Dans un premier temps, une revue de différentes architectures optimisées et de leurs modules permettant d'obtenir des architectures efficaces est présentée. Dans un deuxième temps, nous présentons des méthodes de recherche d'architectures neuronales (Neural Architecture Search (NAS)) afin de construire des modèles de manière plus ou moins autonome à partir de rien.

2.3.1 Passage en revue d'architectures

Pour commencer, les opérations de convolution sont responsables d'une grande partie des coûts de calcul d'un réseau profond. Dans les travaux préliminaires de Lecun et al. [LeCun, 1989], des filtres de tailles 5x5 et 3x3 sont utilisés pour ce type de couche. Ces tailles de filtres restent les plus répandus et sont généralement recommandées (comme on peut le remarquer avec l'enchaînement de filtres 3x3 dans le modèle VGG [Simonyan and Zisserman, 2015]) même si certains modèles très connus comme AlexNet [Krizhevsky et al., 2012] utilisent des résolutions de filtres plus grandes. Une architecture comme GoogleNet [Szegedy et al., 2015] se permet aussi de réduire la résolution de ses filtres jusqu'à une taille 1x1. De plus, GoogleNet introduit aussi l'idée de modules, qui sera reprise par la suite dans des architectures comme ResNet [He et al., 2016] ou DenseNet [Huang et al., 2017]. Les modules sont des blocs composés de plusieurs couches de convolution avec différentes tailles et une organisation spécifique qui va être répétée plusieurs fois dans le réseau.

En utilisant ce concept de modules, Iandola et al. [Iandola et al., 2016] ont construit une architecture appelée SqueezeNet qui repose sur une organisation particulière de ses différentes couches. Le 'fire module' (voir Figure 2.12) permet de décroître le nombre de paramètres du réseau, réduisant ainsi la taille du modèle. La stratégie d'élaboration de ce module s'organise en trois choix principaux :

— L'utilisation de filtres 1x1 pour remplacer la plupart des filtres 3x3 qui sont généralement utilisés dans les CNN.

— Décroître le nombre de canaux en entrée avec des filtres 3x3.

— Sous-échantillonner très tard dans le réseau afin d'avoir des couches de convolution avec des cartes d'activation plus larges.

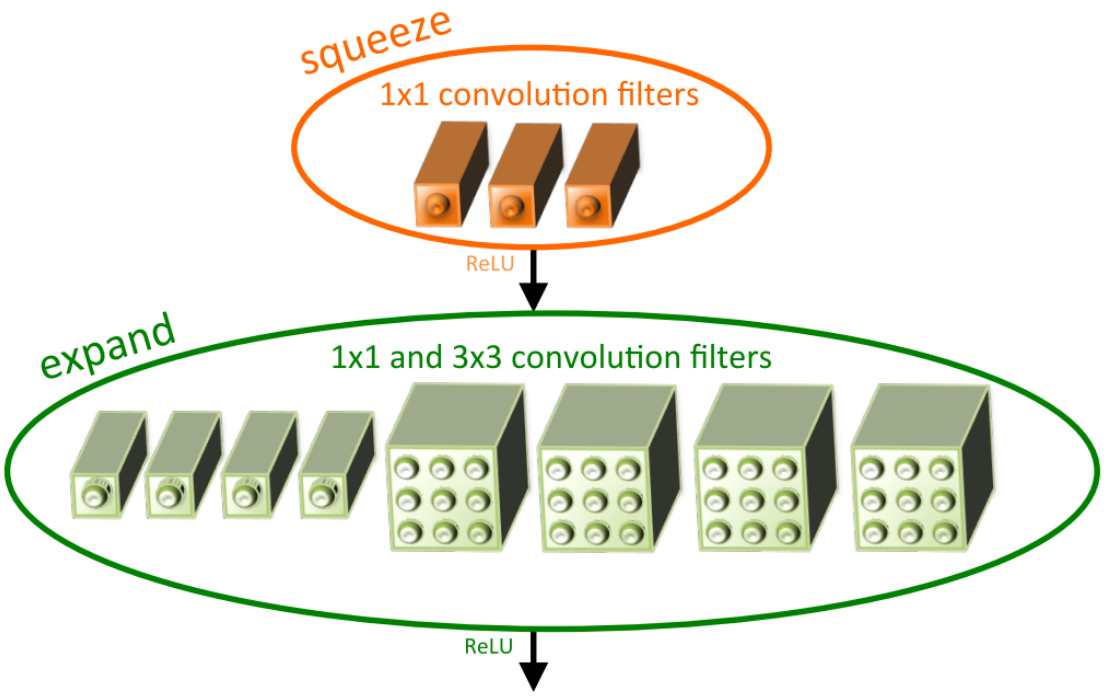


FIGURE 2.12 – Architecture du 'fire module' du modèle SqueezeNet. Figure reprise de [Iandola et al., 2016].

Les deux premiers choix visent à réduire le nombre global de paramètres. Le troisième point vise à améliorer la précision de la classification due aux cartes d'activation plus larges générées par les filtres de taille 1x1 et par le report des étapes de sous-échantillonnages. En pratique, le *fire module* est composé d'une série de couches de convolution groupées dans un bloc *squeeze* qui n'est constitué que de filtres 1x1 suivi d'une autre série de couches de convolution dans un bloc *expand* qui est constitué d'un mixte de filtres de tailles 1x1 et 3x3. Le modèle SqueezeNet répète ces couches plusieurs fois (voir Figure 2.13) afin d'obtenir un réseau 50 fois plus petit qu'AlexNet tout en atteignant la même précision.

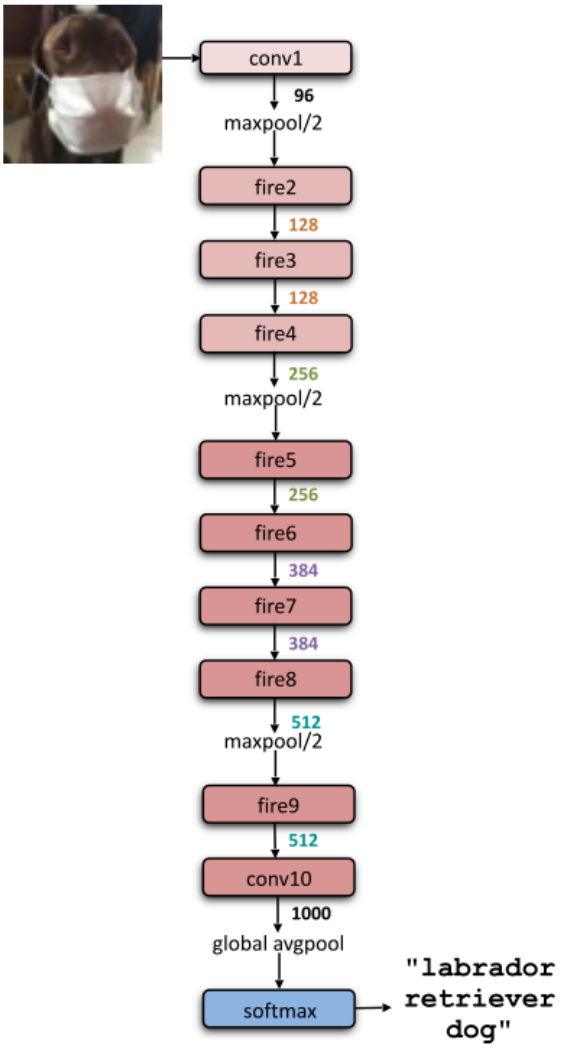


FIGURE 2.13 – Vue globale de l'architecture de SqueezeNet. Figure reprise de [Iandola et al., 2016].

Cette architecture a été poussée un peu plus loin par Nanafack et al. [Nanfack et al., 2017] afin de créer l'architecture Squeeze-SegNet, un FCNN pour de la segmentation sémantique pixel par pixel. Ce modèle est un réseau de type encodeur-décodeur et est présenté dans sa version simplifié sur la Figure 2.14. La partie encodeur du réseau est similaire à l'architecture SqueezeNet tandis que la partie décodeur est composée de *fire modules* inversés appelés *DFire modules*, qui sont aux aussi des séries de modules *expand* et *squeeze* alternés. Chacun des composants de ce modèle est repris de SqueezeNet et s'enchaînent dans l'ordre inverse de ce dernier au niveau du *DFire modules*. Les étapes de sous-échantillonages sont aussi remplacées par des étapes de sur-échantillonages comme le modèle doit construire des cartes d'activation denses. Inspiré aussi par SegNet [Badrinarayanan et al., 2017], le modèle Squeeze-SegNet est capable d'obtenir le même niveau de précision que ce dernier sur un jeu de données comme CamVid [Brostow et al., 2008] tout en diminuant le nombre de paramètres requis par le modèle d'un facteur 10.

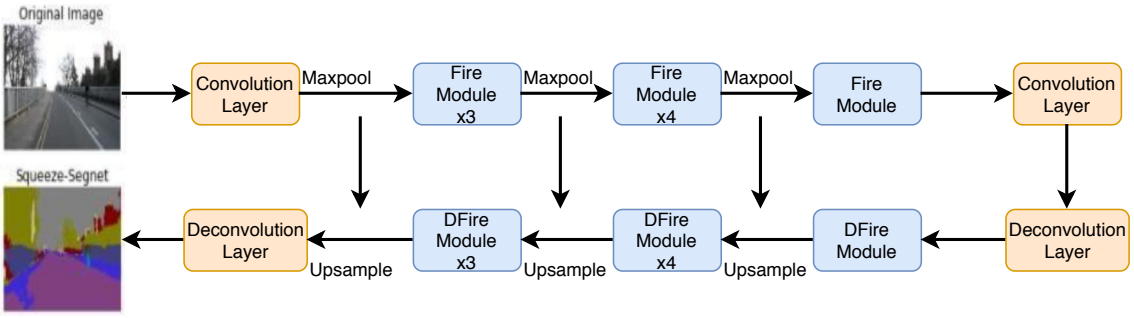
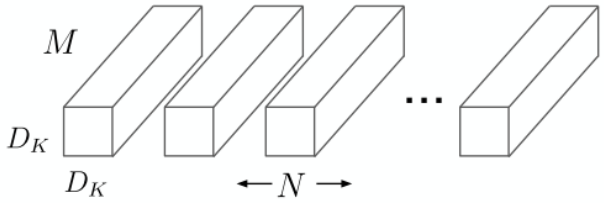
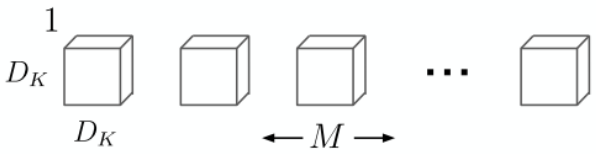


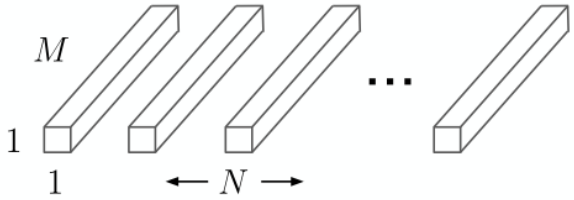
FIGURE 2.14 – Architecture simplifiée du réseau Squeeze-SegNet. Figure inspirée par [Nanfack et al., 2017].



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

FIGURE 2.15 – Figure reprise de [Howard et al., 2017]. La convolution standard (a) est remplacée par deux opérations successives : la convolution en profondeur (b) et la convolution point-par-point (c) pour construire un filtre séparable en profondeur.

En 2012, Mamalet et al. [Mamalet and Garcia, 2012] ont introduit une simplification des filtres de convolution en utilisant des couches de convolution séparables (consistant à séparer une matrice en deux matrices plus petites). Ces travaux ont été améliorés par Howard et al. [Howard et al., 2017] avec le modèle MobileNet. Inspirés par les travaux de Chollet [Chollet, 2016], les couches centrales de leur architecture sont basées sur des caractéristiques séparables en profondeur (*depthwise separable features*) [Sifre and Stephane, 2014]. Expliqué différemment, l'étape de convolution est factorisée en deux différentes étapes afin de faire décroître les temps de calcul générés par les différentes multiplications de matrices. Premièrement, une convolution en profondeur applique un seul filtre

sur chacun des canaux en entrée. Deuxièmement, une convolution point-par-point applique une convolution 1x1 afin de recombiner les sorties des convolutions en profondeur. Ces opérations peuvent être visualisées sur la Figure 2.15. Cette factorisation, introduite par [Sifre and Stephane, 2014] permet de réduire dramatiquement les coûts de calcul de toutes les convolutions.

Les couches de convolution séparables sont devenues une solution efficace pour accélérer les opérations de convolution. Zhang et al. [Zhang et al., 2017] ont aussi approfondi ces travaux avec un réseau de neurones appelé ShuffleNet. La particularité de ce réseau est qu'il ajoute aux caractéristiques séparables en profondeur une "unité de mélange" (*Shuffle Unit*). Cette unité permet au modèle de mélanger des canaux pour les convolutions de groupe. En général, chaque canal de sortie est lié à un groupe de canaux en entrée. Ici, nous considérons une couche de convolution avec $g * n$ canaux et g groupes. La dimension du canal de sortie est d'abord remodeler en (g, n) et ensuite transposée et aplatie comme entrée de la couche suivante. Comparé à d'autres modèles, la complexité est grandement réduite. Comparé à MobileNet [Howard et al., 2017], l'efficacité et la précision sont légèrement améliorées.

Dans la section 2.2.6, nous avons présenté des méthodes pour rendre des réseaux binaire. Bulat et al. [Bulat and Tzimiropoulos, 2017] ont développé une version binaire d'une architecture appelée *stacked hourglass network* [Newell et al., 2016], un modèle de l'état de l'art dans l'estimation de la pose humaine. La principale contribution du modèle binaire est qu'il améliore une couche définie comme goulot (*bottleneck layer*) en limitant le nombre de filtres 1x1 et en augmentant le nombre de sauts de couches (*skip layers*) pour limiter la perte d'information due au passage du réseau en binaire. Sur de l'alignement de visage en 3D, ce modèle surpasse les autres méthodes les plus performantes de 35%. Cependant, sur des tâches d'estimation de la pose humaine, la version binaire du modèle est loin derrière sa représentation réelle en terme de performance. Ainsi, il y a encore de la marge pour améliorer les méthodes binaires qui ne sont pas encore les plus au point. Il est aussi important de noter qu'une architecture peut être changée et améliorée afin d'utiliser des paramètres avec une précision numérique limitée. La compression et la construction d'une architecture sont donc intimement liées.

2.3.2 Neural architecture search

Ces dernières années, nos connaissances sur les réseaux profonds ont grandis grâce au développement de nouveaux modules et de nouvelles architectures. Cependant, savoir quel modèle utiliser pour une application donnée n'est pas toujours une tâche simple. De plus, plus la tâche visée est difficile, plus la construction d'un modèle "à la main" l'est aussi. Comme construire une architecture parfaite peut prendre un temps infini, des travaux ont été faits afin d'étudier la possibilité de laisser des réseaux grandir, s'adapter et même construire leur propre architecture de manière autonome. Il est intéressant de noter que les premiers travaux dans ce domaine étaient plus orientés vers des thématiques physique et biologique plutôt qu'informatique. Rosenblatt [Rosenblatt, 1962], Kohonen [Kohonen, 1982] ou Willshaw et al. [10., 1976] ont associé l'organisation structurelle du cerveau aux réseaux de neurones afin de trouver des mécanismes d'auto-organisation théoriques. Depuis lors, de nombreux travaux sur le sujet ont été effectués et il peuvent être regroupés sous le terme générique de Neural Architecture Search.

Nous effectuons une revue de ces différentes méthodes dans cette section. Nous commençons par introduire les NAS avec les travaux préliminaires dans le domaine sur les

gaz neuronaux, suivis par les méthodes de *neuroevolution*, inspirées par les algorithmes génétiques, avant de finir par les méthodes de morphismes de réseaux qui visent à transformer des architectures déja entrainées. Dans ces méthodes, les architectures construites sont majoritairement optimisées pour obtenir les meilleurs résultats pour une tâche particulière. Cependant, la taille et la consommation mémoire de ces structures peuvent ne pas être optimisées. Ainsi, dans une dernière section, nous nous intéresserons aux méthodes de *supergraph* capables de trouver des structures optimisées pour différents critères.

2.3.2.1 Gaz Neuronaux

A la suite des premiers travaux orientés vers le domaine physique, des méthodes de *gaz neuronaux* ont émergés. Introduites par Schulten [Martinetz et al., 1993], ces méthodes ont été les premières approches à pousser le concept de structures s'auto-organisant. Elles visaient à trouver une représentation optimale des données en se basant sur les vecteurs caractéristiques. Au début des années 90, les travaux de Fritzke [Fritzke, 1995, Fritzke, 1994b, Fritzke and Bochum, 1994, Fritzke, 1994a] ont exploré les bases de l'auto-organisation liée aux réseaux de neurones incrémentaux permettant d'améliorer les méthodes de gaz neuronaux en tant que structures pouvant croître. Un exemple d'interprétation de ces réseaux en graphe est visible sur la Figure 2.16. A travers leurs travaux, les auteurs ont principalement exploré deux idées.

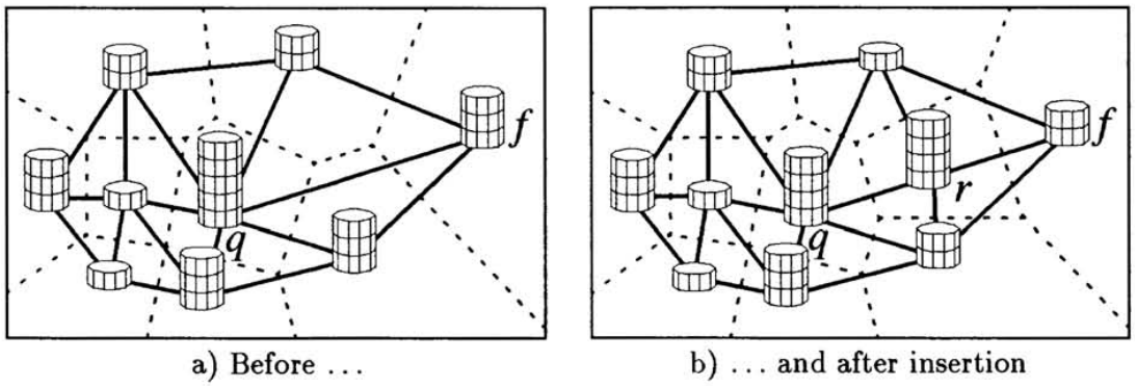


FIGURE 2.16 – Figure reprise de [Fritzke and Bochum, 1994]. Dans les méthodes de gaz neuronaux, les réseaux de neurones sont interprétés comme des graphes ou chaque noeud correspond à une unité. Ici on schématise l'insertion d'une nouvelle unité. L'unité présente en q ayant accumulée le plus d'erreurs, une nouvelle unité est insérée entre q et un de ses voisins direct.

La première de ces deux idées, décrite dans [Fritzke, 1994a], était de développer une approche d'apprentissage non-supervisée pour la visualisation, le regroupement et la quantification vectorielle de données afin de trouver une architecture convenable de manière automatique. Dans ce travail, un réseau de neurones peut être vu comme un graphe sur lequel un processus de contrôle de la croissance est appliqué. De plus, une approche d'apprentissage supervisée a aussi été développée, ajoutant une fonction de base radiale. Cette addition permet, pour la première fois, d'ajouter de nouvelles unités et de superviser l'entraînement des paramètres en même temps. De plus, les nouvelles unités ne sont plus ajoutées de manière aléatoire contrairement aux méthodes antérieures. Cela permet d'obtenir des réseaux qui généralisent de manière plus efficace. Leur méthode a été testée sur des problématiques de reconnaissances de voyelles et permettait d'obtenir de meilleurs résultats que l'algorithme des plus plus proches voisins (la technique à l'état

de l'art de l'époque).

La seconde idée décrite par Fritzke [Fritzke, 1995] est une extension de la première méthode mais basée sur une règle de Hebb. Contrairement à [Martinetz et al., 1993], le modèle n'a aucun paramètre qui change au cours du temps mais est quand même capable d'apprendre en continu jusqu'à ce qu'un critère de performance soit atteint. D'un point de vue théorique, ces recherches ont permis d'aider à mieux appréhender et comprendre la circulation de l'information à travers un réseau, permettant ainsi de poser les bases pour les méthodes qui suivirent.

2.3.2.2 Neuroevolution

Les algorithmes génétiques sont des techniques bien connues permettant de trouver des solutions à des problèmes d'optimisations complexes. Construire un modèle optimisé pour une tâche donnée peut-aussi être vu comme une tâche d'optimisation. Il est donc normal que ces méthodes soient adaptées pour les réseaux de neurones profonds. Ces méthodes sont ainsi regroupées dans un domaine appelé *neuroevolution*. Le principe de base des méthodes de *neuroevolution* est le suivant : une topologie évoluant avec ses poids devrait permettre d'obtenir un avantage sur des poids évoluant dans une topologie fixe. Depuis des décennies, les méthodes de *neuroevolution* [Montana and Davis, 1989, Floreano et al., 2008] ont été appliquées avec succès sur des tâches de décisions séquentielles. Ce succès vient en partie du fait que les tâches de décisions séquentielles optimisent les poids des réseaux de neurones à la place d'optimiser la descente de gradient. Stanley et al. [Stanley and Miikkulainen, 2002] ont été plus loin avec une méthode appelée NeuroEvolution of Augmenting Topologies. Cette technique permet de minimiser les dimensions de l'espace de recherche des poids des différentes connexions, menant à une recherche rapide et efficace de nouvelles topologies.

Cependant, une importante difficulté dans l'implémentation de techniques de *neuroevolution* vient de ce que l'on appelle le problème de permutation [Radcliffe, 1993]. Dans les méthodes évolutives, il y a plus d'une seule manière d'exprimer une solution. Pour des réseaux de neurones, cela veut dire qu'il existe plus d'une seule architecture pouvant exprimer le même problème d'optimisation des poids. La base des algorithmes évolutifs est de générer et construire automatiquement des topologies qui vont entrer en compétition les unes avec les autres. Les meilleures topologies (ou solutions) sont ensuite mélangées ensemble afin d'obtenir une topologie encore meilleure. Cette dernière étape est ce qu'on appelle le *crossover* (voir Figure 2.17). Ces différents processus sont répétés jusqu'à ce que la solution ne puisse plus être améliorée. Durant ces étapes, le problème de permutation intervient quand des structures représentant la même solution ont un encodage différent. En effet, si deux topologies avec la même architecture mais un encodage différent sont mélangées l'une avec l'autre, la structure résultante peut mener à de futures structures endommagées entraînant de la perte importante d'informations (voir la Figure 2.18 pour une visualisation du problème). Ainsi, ces algorithmes doivent suivre des conventions strictes pour des topologies dites "fixes" ou "sous contraintes" comme un encodage génétique non-redondant [Thierens, 1996]. Formulé différemment, il devient impossible d'obtenir deux structures similaires. Néanmoins, il est difficile de respecter complètement ces conventions pour des réseaux de neurones où à la fois les poids et les topologies seraient en évolution constante. Ainsi, pour des modèles profonds, le problème de permutation est difficile à éviter. Les travaux de Stanley et al. [Stanley and Miikkulainen, 2002] ont trouvé une solution alternative en gardant en mémoire l'historique des réseaux afin de définir quelles parties des topologies doivent être mélangées sans

perdre d'informations. Cependant, cette méthode est extrêmement coûteuse en terme de mémoire à cause de la tenue permanente d'un historique des réseaux élaborés. Ainsi, les algorithmes NEAT [Stanley and Miikkulainen, 2002] ne performent bien que sur des petits modèles simples. Les modèles complexes entraînant tous les problèmes que nous venons de détailler.

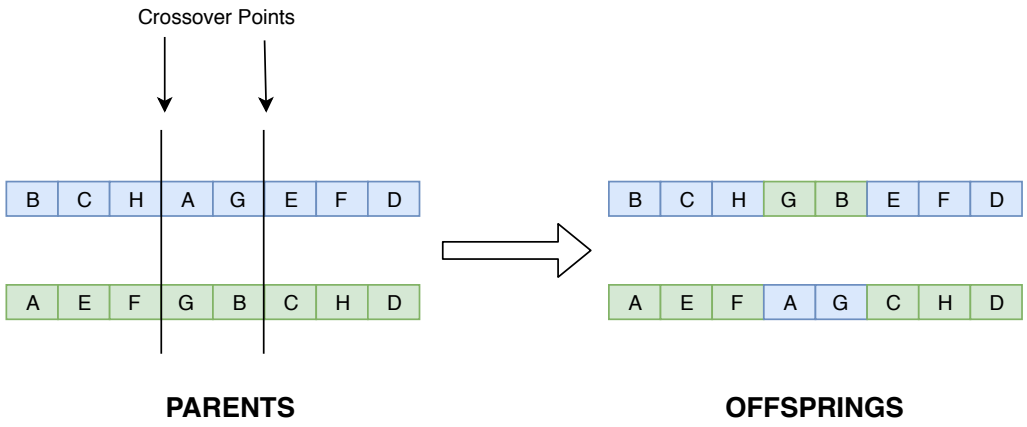


FIGURE 2.17 – Exemple d'une étape de crossover. Les deux structures dites "parentes" (gauche) sont découpées de manière aléatoire à certains points afin d'être mélangées et recomposées pour construire des structures dites "filles".

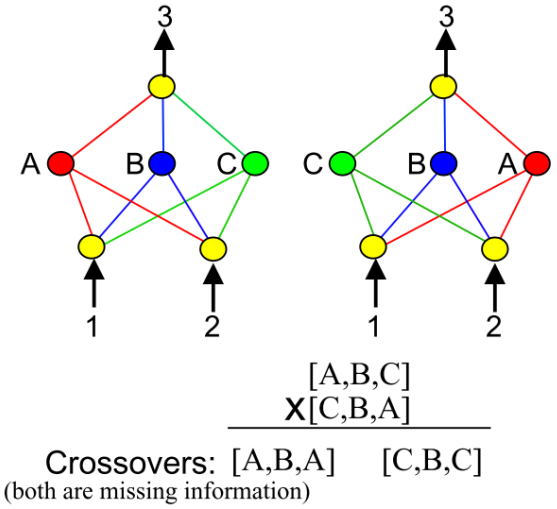


FIGURE 2.18 – Visualisation du problème de permutation. Les deux réseaux expriment la même solution, même si leurs unités apparaissent dans un ordre différent. Cela a pour effet de rendre le crossover impossible ou une des unités principales disparaît, entraînant une perte importante d'informations. Figure reprise de [Stanley and Miikkulainen, 2002].

Le modèle NEAT [Stanley and Miikkulainen, 2002] a été amélioré à travers une méthode que l'on appelle CoDeepNEAT [Miikkulainen et al., 2017]. Cette amélioration repose sur une co-évolution des composants, topologies et hyperparamètres. De plus, l'évolution et l'optimisation sont basées sur le gradient, au contraire des méthodes précédentes où l'optimisation était basée sur les poids du réseau. Sur le jeu de données de reconnaissance d'images CIFAR-10, CoDeepNEAT est capable de découvrir automatiquement des structures qui ont des performances comparables à celles des modèles de l'état de l'art. De plus, sur des problèmes d'étiquetages d'images, cette approche est aussi capable de trouver des meilleures structures que celles imaginées par un humain si on

lui laisse le temps de calcul nécessaire pour cela. On peut en conclure que ces méthodes fonctionnent et sont capables de trouver de manière automatique des structures adaptées et efficaces pour une tâche donnée sous certaines conditions. Cependant, l'efficacité de la recherche et le coût de calcul de ces méthodes restent les gros désavantages de ces approches.

2.3.2.3 Morphisme de réseaux

Les méthodes NAS ne sont pas seulement limitées aux approches de *neuroevolution*. Le morphisme de réseaux [Elsken et al., 2018,Cai et al., 2018a,Jin et al., 2018] est aussi une partie importante du domaine. Cette approche a pour but de modifier un réseau de neurones déjà entraîné en une autre architecture. Ainsi, des opérations de morphisme sont appliquées sur les réseaux comme par exemple de l'insertion de couches ou de sauts de connexions. La conséquence de cela, c'est qu'il est difficile de déterminer quelles opérations appliquer à un modèle. Dans [Jin et al., 2018], les auteurs utilisent des optimisations Bayesienne et sélectionnent dans un espace de recherche défini à chaque itération l'opération qui a le plus de probabilité d'améliorer les performances du réseau. L'avantage de cette technique est qu'elle ne nécessite pas un nombre important d'epochs supplémentaires pour fonctionner. Cependant, comme les opérations de morphisme s'effectuent au niveau des couches, les topologies de réseaux sont contraintes d'adopter une structure en chaîne. Comme la plupart des réseaux de l'état de l'art sont des structures à plusieurs chemins, c'est une limitation non négligeable. Néanmoins, Cai et al. [Cai et al., 2018b] ont réussi a étendre l'espace de recherche à des structures multi-chemins en permettant aux poids d'être réutilisés et en utilisant des architectures en forme d'arbres. Le désavantage est encore une fois les coûts excessifs des calculs qu'imposent l'exploration de ces grands espaces de recherche.

2.3.2.4 Supergraphes

Désormais, de nombreuses méthodes NAS sont développées et sont trouvables dans la littérature. Un des principaux problèmes comme nous l'avons souligné plusieurs fois est que ces méthodes sont souvent confrontées à un espace de recherche des solutions possibles bien trop grand. Théoriquement, cet espace de recherche est infini, entraînant un énorme coût en temps et en calculs. Il est donc nécessaire de limiter cet espace de recherche. Dans les méthodes décrites précédemment, cette limitation est principalement effectuée en limitant et en contrôlant le nombre d'opérations qui peuvent être faites pendant l'évolution des réseaux. Cependant, une approche alternative pourrait être de ne pas limiter les opérations qui peuvent être possibles mais plutôt l'espace de recherche dans lequel elles sont implémentées. C'est l'idée principale derrière les *supergraphes*.

Un *supergraphe* est un grand graphe opérationnel où chaque sous-graphe est un réseau de neurones [Saxena and Verbeek, 2016,Pham et al., 2018,Veniat and Denoyer, 2018] (voir la Figure 2.19 pour un exemple). Ainsi, le *supergraphe* devient l'espace de recherche et la solution à la tâche voulue sera un de ses sous-graphes. Le bénéfice de cette méthode est que les capacités de calculs nécessaires sont réduites de manière importante comme la solution est recherchée dans un espace bien plus contraint que dans les précédentes méthodes.

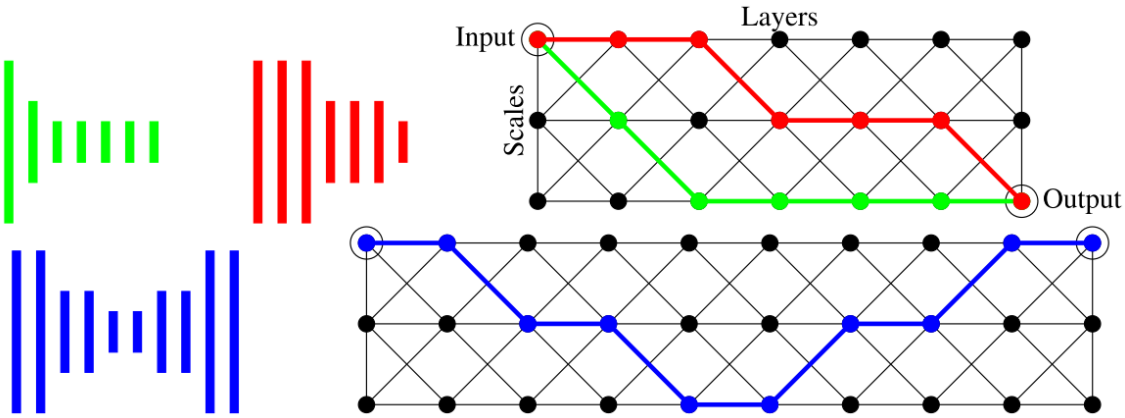
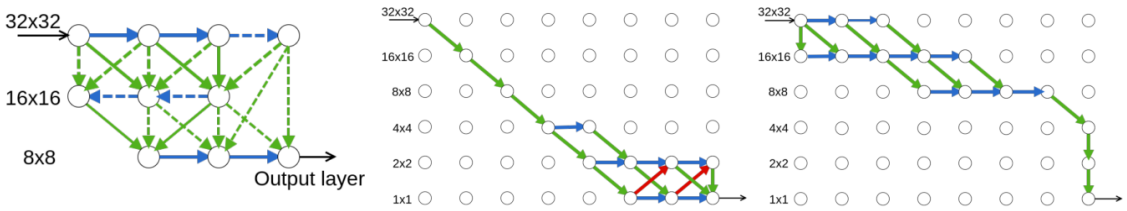


FIGURE 2.19 – Un exemple de sous-graphes trouvés via la méthode de *convolutional neural fabrics* [Saxena and Verbeek, 2016]. Tous les noeuds sont des couches de convolution. Tous les graphes sont orientés vers la droite. La taille des *feature map* de chaque couche est donnée par la hauteur du noeud dans le graphe. Ainsi, rouge et vert sont des réseaux avec sept couches de convolution et bleu est un réseau de convolution-déconvolution constitué de dix couches. Figure reprise de [Saxena and Verbeek, 2016].

Récemment, Pham et al. [Pham et al., 2018] ont réduit les ressources de calcul nécessaires pour construire des réseaux avec des performances élevées. Leur idée est la suivante : tous les sous-graphes produits par le *supergraphe* partagent les mêmes paramètres afin d'éviter les entraînements recommençant constamment à zéro. Il est possible d'argumenter que partager des paramètres entre différents modèles peut mener à des erreurs. Cependant, les méthodes de transferts d'apprentissages ont montré que les paramètres appris pour une tâche spécifique peuvent être utilisés par d'autres modèles sur d'autres tâches [Hinton et al., 2014, Zoph et al., 2016]. Ainsi, dans la pratique, le fait que tous les sous-graphes partagent les mêmes paramètres pour des buts différents n'est pas vraiment un problème.

Une approche différente a été développée par Veniat et al. [Veniat and Denoyer, 2018] dans leur méthode appelée Budgeted Super Networks. Ici, le *supergraphe* est défini comme un ensemble d'architectures possibles mais un coût maximum autorisé doit aussi être défini. Ce coût est directement lié aux performances, aux capacités de calcul et à la consommation mémoire du réseau. Cela permet aux auteurs de décider quels critères l'algorithme doit privilégier pendant la recherche d'une solution. Afin d'atteindre ce but, un modèle stochastique est proposé et optimisé en utilisant des méthodes inspirées des gradients. De plus, les auteurs ont aussi démontré que la solution de leur modèle stochastique correspond à l'architecture optimale la plus contrainte du *supergraphe*. Quelques exemples de réseaux trouvés via cette méthode sont visualisables sur la Figure 2.20. Testée sur le jeu de données CIFAR pour la classification et la segmentation d'images, leur solution est capable de converger pour construire des architectures similaires au modèle ResNet [He et al., 2016]. De plus, les modèles qui en ressortent ont le même coût de calcul et la même consommation mémoire que le modèle ResNet original mais obtiennent une meilleure précision.



(a) B-ResNet (b) B-CNF & computation cost (c) B-CNF & memory consumption cost

FIGURE 2.20 – Exemple d'architectures découvertes avec la méthode de BSN [Veniat and Denoyer, 2018]. (a) est un petit ResNet demandant peu de ressources de calcul. (b) et (c) sont des réseaux de convolutions découvert par l'algorithme BSN. (b) est un réseau peu demandeur en ressource de calcul : les convolutions avec des filtres larges sont évitées pour préférer les convolutions à faibles résolutions. (c) est un réseau privilégiant une faible consommation mémoire et garde les convolutions à hautes résolutions car elles possèdent des cartes de caractéristiques plus détaillées et le même nombre de paramètres que celles ayant une résolution plus faible. Figure reprise de [Veniat and Denoyer, 2018].

Plusieurs conclusions peuvent être tirées de ces expérimentations. Premièrement, contraindre la recherche d'architectures optimisées mène à une réduction des ressources de calcul nécessaires pour fouiller l'espace de recherche. Deuxièmement, réduire l'espace de recherche mène aussi à des modèles limitant leur consommation des ressources, aussi bien en terme de calculs qu'en terme de mémoire utilisée, sans compromettre la précision des modèles.

Tan et al. [Tan et al., 2019] ont poussé leur recherche dans cette direction. Les auteurs ont proposé une méthode de NAS optimisée spécifiquement pour plate-formes mobiles appelée MnasNet. Dans leurs travaux, la précision et le temps de latence de l'inférence du modèle sur une plate-forme mobile sont tous les deux pris en compte pour l'optimisation du modèle. Ainsi, le modèle essaie de trouver le meilleur compromis entre ces critères. Afin d'atteindre cet équilibre, deux espaces de recherche hiérarchique sont utilisés : un espace pour factoriser un modèle profond en une séquence de blocs et un autre espace pour déterminer l'architecture détaillée des couches qui composent ce bloc. Ainsi, différentes couches sont capables d'utiliser des opérations différentes mais toutes les couches qui sont dans le même bloc vont partager la même structure. Le bloc adapté à un endroit donné est réévalué à différentes profondeurs du réseau pour réduire le temps de latence d'inférence général du modèle. Ces améliorations permettent de réduire l'espace de recherche tout en trouvant des architectures qui sont plus performantes et rapides que MobileNets par exemple [Howard et al., 2017, Sandler et al., 2018a].

2.3.3 Discussion

Nous avons vu dans la première partie de cet état de l'art que la recherche de nouvelles méthodes pour compresser des modèles déjà entraînés est diversifiée et efficace. Cependant, nous venons aussi de voir que ce n'est pas la seule voie possible. Se concentrer sur l'architecture des modèles et leur construction préalable est aussi une voie en pleine expansion et permet d'obtenir des modèles déjà optimisés. En effet, malgré les optimisations matérielles et algorithmiques qui peuvent être développées, des travaux récents comme Mobile-Net [Howard et al., 2017] et Shuffle-Net [Zhang et al., 2017] ont montré qu'il est prometteur de ne pas simplement compresser les modèles mais aussi de les construire de manière intelligente. Une architecture bien pensée est la première étape vers un réseau optimisé.

Les travaux sur les NAS permettent de construire des architectures optimisées (du point de vue de la performance et de l'efficacité des calculs) pour une tâche bien définie. Bien que ce ne soit pas une tâche facile, des travaux commencent à montrer des résultats prometteurs à travers de nouveaux algorithmes et théories comme l'hypothèse du ticket de loterie par exemple [Frankle and Carbin, 2019]. Ces travaux permettent de mettre sur le devant de la scène et de pousser la recherche bien au delà de l'optimisation de réseaux en permettant de comprendre les mécanismes derrière les modèles profonds dont nous ne maîtrisons pas encore tous les codes.

Les contributions de ce chapitre ont débouché à des publications à la conférence *ORASIS* et dans *Journal of Signal Processing Systems* [Berthelier et al., 2020a].

Chapitre 3

Compression de réseaux convolutifs par utilisation d'un terme de clarté $\frac{l_1}{l_2}$ sur les noyaux

Dans ce chapitre, nous présentons un terme de régularisation induisant de la dispersion basée sur un rapport de pseudo-norme l_1/l_2 définie sur les coefficients de filtres de CNN. En définissant cette pseudo-norme de manière appropriée pour les différents filtres des noyaux de CNN, le nombre de noyaux de chaque couche peut-être considérablement réduit amenant à des structures compact de DCNN. Contrairement à beaucoup de méthodes existantes, l'approche présentée ne nécessite pas d'étapes de ré-entraînement itératives et, grâce à un terme de régularisation, produit directement un modèle dispersé durant la phase d'apprentissage. De plus, cette approche est aussi plus facile et simple à implémenter que les méthodes déjà existantes. Nous montrons que sur des modèles et des jeux de données classiques de classification, cette méthode se comporte aussi bien, voire mieux, que d'autres méthodes de l'état de l'art.

3.1 Introduction

Dans le chapitre précédent, nous avons vu que les réseaux de neurones profonds se sont montrés efficaces dans différentes situations. Leurs implémentations facilement parallèlisables [Krizhevsky et al., 2012] [LeCun et al., 2015] ont permis à de nombreux modèles de repousser les performances de l'état de l'art dans des tâches de visualisations comme de la reconnaissance de visages, de la segmentation sémantique, de la classification et de la détection d'objets, etc. [Krizhevsky et al., 2012] [Simonyan and Zisserman, 2015] [Szegedy et al., 2015] [He and Sun, 2015] [He et al., 2016]. Nous avons aussi vu que ces modèles sont bien souvent composés de nombreux paramètres, faisant de ces réseaux des modèles coûteux en mémoire et en temps de calcul.

Dans le cadre de cette thèse, nous cherchons à mettre en oeuvre ce genre de modèle sur des appareils ayant des ressources limitées comme des systèmes embarqués ou des smartphones. Il est donc indispensable d'optimiser et de réduire la mémoire et les temps de calculs de ces modèles. Comme nous l'avons montré dans le chapitre précédent, de nombreuses contributions ont été publiées dans ce domaine, aussi bien en terme de compression qu'en terme d'optimisation d'architectures, permettant d'obtenir des réseaux neuronaux profonds toujours plus compacts.

Les réseaux de neurones profonds les plus performants sont des réseaux composés essentiellement de couches de convolution et les opérations de convolution sont celles

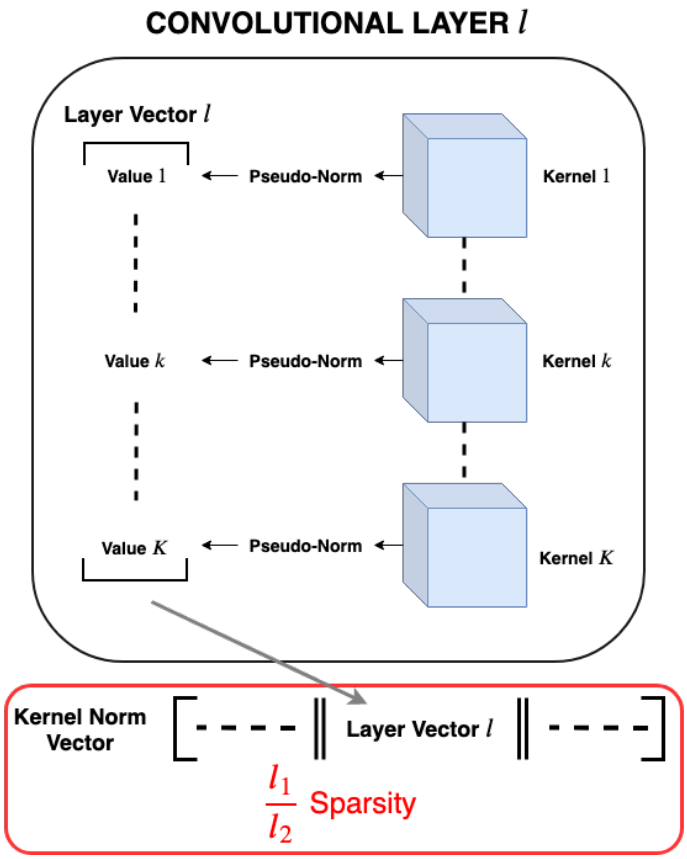


FIGURE 3.1 – Représentation visuelle de la méthode et du calcul du vecteur de normes de noyaux via une pseudo-norme.

demandant le plus de temps de calculs. Nous avons ainsi décidé de proposer une nouvelle méthode de compression en se focalisant sur ce type de couche. Bien entendu, de nombreuses méthodes permettent déjà de réduire ou supprimer ces couches. On pense notamment à des méthodes d'élagages, que nous avons vu dans la partie précédente, afin de supprimer des filtres. Ces stratégies de réduction pouvant aller de la suppression simple de filtres jugés comme non-essentiels et ayant un effet minime sur la précision de sortie [Li et al., 2017] à des stratégies plus complexes basées sur l'évaluation des filtres grâce à des méthodes d'optimisations et de statistiques [Luo et al., 2017].

L'approche développée ici est inspirée par toutes ces méthodes mais est aussi motivée par deux idées qui ressortent de cette littérature. Premièrement, de nombreux travaux montrent qu'il existe de la redondance entre les poids d'un réseau de neurones convolutif profond [Jaderberg et al., 2014]. Et deuxièmement, de nombreuses méthodes de parcimonie (*sparsity methods*) existent dans la littérature [Bach et al., 2012] mais ces méthodes ont été très rarement utilisées afin de supprimer directement des noyaux pendant l'apprentissage [Wen et al., 2016]. Nous proposons ainsi une nouvelle stratégie de réduction, basée sur une norme l_1/l_2, afin de forcer certains noyaux à avoir tous leurs poids égaux à zéro (de manière à ce que les filtres associés puissent être supprimés).

L'idée principale de cette contribution est d'exprimer le problème de réduction de filtres en introduisant une structure éparse sur un ensemble de pseudo-normes calculées sur chaque noyau. La Figure 3.1 illustre l'idée générale se cachant derrière cette méthode. Chaque noyau du réseau est transformé en une valeur unique en utilisant une pseudo-norme. Toutes ces valeurs sont ensuite concaténées dans un vecteur global (la taille du

vecteur étant le nombre de filtres) appelé vecteur de normes des noyaux. La parcimonie globale sur les noyaux est définie par la parcimonie sur ce vecteur et est estimée par un ratio de normes l_1/l_2. Comme un noyau avec tous ses poids à zéro produit une pseudo-norme égale à zéro, le nombre de filtres peut être réduit en accentuant la structure éparse du vecteur de normes des noyaux. Ici, nous proposons la norme l_1/l_2 pour deux raisons : (1) la norme l_1/l_2 est une norme de groupe simple à implémenter et (2) l'utilisation de la norme l_1 permet d'améliorer la performance, l'interprétabilité et la parcimonie d'un modèle [Huang and Zhang, 2010, Turlach et al., 2000, Yuan and Lin, 2006a] combinée avec la norme l_2 qui permet de converger vers une solution stable et de maintenir une structure éparse à un bon niveau.

Nous proposons une norme l_1/l_2 calculée sur le vecteur global (le vecteur des pseudo-normes des noyaux) de façon à ce que l'ajout de ce terme de clarté pour minimiser la fonction de coût globale réduise le nombre de filtres (non à zéro) d'un réseau de neurones convolutif profond. L'élagage se fait au niveau de noyaux 3D entiers permettant ainsi de compresser de manière plus efficace les modèles par rapport au fait d'enlever seulement des canaux 2D. Comparé à d'autres approches, cette méthode présente plusieurs avantages :

1. Toutes les étapes sont effectuées pendant l'apprentissage. Aucune opération supplémentaire de *fine-tuning* n'est nécessaire.

2. Cette méthode est basée sur des normes l_1 et l_2 qui sont des normes simples, faciles à implémenter et à calculer contrairement à d'autres méthodes supprimant des poids durant l'apprentissage.

3. Comme nous gardons en mémoire l'évolution du réseau à chaque étape de l'apprentissage, il est possible de choisir le meilleur modèle en se basant sur un équilibre entre la compression et la précision voulue.

Dans les prochaines sections, nous allons faire un court rappel des méthodes d'élagages et de structuration de poids éparses nous ayant inspirées pour cette contribution dans la section 3.2. Dans la section 3.3 nous expliquons cette contribution basée sur la norme l_1/l_2. Enfin, dans la section 3.4, nous montrons les résultats expérimentaux de cette méthode avec les architectures LeNet et VGG entraînées sur les jeux de données MNIST et CIFAR-10. Nous montrons que cette méthode est capable d'améliorer significativement la structure éparse des couches de convolution dans les réseaux de neurones convolutif profonds (DCNN) sans diminution significative de leur précision.

3.2 Travaux liés

De nombreuses méthodes de compression ont été détaillées dans le chapitre 2. Nous n'allons donc pas nous focaliser de nouveaux sur l'ensemble de ces méthodes dans cette section mais uniquement sur celles ayant influencées l'élaboration de cette contribution : l'élagage et les poids éparses.

3.2.1 Elagage de réseaux

Les méthodes d'élagage sont les méthodes permettant de supprimer les paramètres jugés comme non essentiels à la propagation de l'information dans un réseau de neurones. Han et al. [Han et al., 2016, Han et al., 2015] ont proposés une méthode d'élagage basée sur un seuil d'activation afin de supprimer tous les paramètres sous ce seuil. Contrairement à la contribution de ce chapitre, la plupart des suppressions sont faites au niveau des couches entièrement connectées et non pas sur les couches de convolution. Cependant, la compression des couches de convolutions est essentielle de nos jour comme les nouveaux réseaux de neurones profonds sont des réseaux de neurones convolulif profonds avec de moins en moins de couches entièrement connectées (seulement 3.99% des paramètres de ResNet par exemple) [He et al., 2016]. Plus proche de notre approche, les méthodes d'élagages structurées suppriment directement des parties entières afin de maintenir une structure globale, comme des noyaux ou des couches, pour compresser des CNN. Li et al. [Li et al., 2017] ont utilisé une norme l_1 pour supprimer des filtres. He et al. [He et al., 2017b] ont utilisé une sélection de canaux basée sur une régression LASSO pour supprimer des filtres. L'élagage de canaux est majoritairement utilisé sur des réseaux de neurones convolutifs profonds très utilisés. Par exemple, la sélection de cartes de caractéristiques non-importantes peut-être faite en utilisant une régularisation l_1 [Liu et al., 2017].

Ces dernières années, de nombreux algorithmes de compression de réseaux utilisant des méthodes d'élagages et atteignant l'état de l'art ont émergés. Yu et al. [Yu et al., 2018] ont proposé une méthode de score d'importance de propagation des neurones (NISP) basée sur la réponse des dernières couches afin d'évaluer l'impact de la suppression des couches antérieures. Zhuang et al. [Zhuang et al., 2018] ont développé une méthode de coût discriminante afin de déterminer les canaux les plus utiles dans les couches intermédiaires. D'autres méthodes comme les FPGM (Filter Pruning Via Geometric Median) [He et al., 2019] ne se concentrent pas sur la suppression de filtres ayant peu d'importance mais évaluent seulement leur redondance. De manière similaire, Lin et al. [Lin et al., 2019] abordent le problème des structures redondantes en proposant une méthode d'apprentissage antagoniste génératrice GAL (non seulement pour supprimer des filtres, mais aussi pour supprimer des branches et des blocs).

Néanmoins, les méthodes d'élagages standards aboutissent généralement à des réseaux non structurés et ayant des connexions irrégulières, menant à des accès irréguliers de la mémoire. Dans la plupart de ces approches, le réseau de neurones profond est entraîné préalablement. Puis chaque paramètre est évalué pour savoir s'il apporte de l'information au réseau. Si ce n'est pas le cas, le paramètre est supprimé. Ainsi, une étape de *fine-tuning* est nécessaire par la suite afin que le modèle puisse retrouver une précision suffisante. Ces étapes prennent du temps. La plupart d'entres-elles sont faites hors ligne et nécessitent des réitérations coûteuses de décomposition et de *fine-tuning* afin de trouver la meilleure répartition des poids tout en maintenant une précision et un taux de compression suffisants. A l'inverse de ces méthodes, notre approche est capable de rendre un réseau éparse durant l'apprentissage et d'identifier quels noyaux doivent être supprimés sans étapes de calcul supplémentaires après l'apprentissage.

3.2.2 Structures de poids éparses

Un facteur important pour la compression d'un modèle va être sa structure éparse, c'est à dire le nombre de ses paramètres mis à zéro. Cependant, cette structure éparse

doit être structurée afin de pouvoir être efficace au niveau de l'accès en mémoire et du temps de calculs. Liu et al. [Baoyuan Liu et al., 2015] ont réussi à mettre à zéro près de 90% des paramètres du réseau AlexNet avec une perte de précision de seulement 2% en utilisant des algorithmes de décompositions et de multiplications de matrices éparses. Cette méthode emploie aussi un LASSO groupé [Yuan and Lin, 2006b] et une régularisation efficace afin d'apprendre des structures éparses. Elle est aussi utilisée par Wen et al. [Wen et al., 2016] pour régulariser la structure d'un réseau de neurones profond à différents niveaux (par exemple au niveau des filtres, des canaux, de la taille des filtres ou de la profondeur des couches). Cette approche mène aussi à des réseaux de neurones profond ayant des coûts de calculs réduits et une accélération efficace due à la structure éparse induite par la méthode. Nous proposons d'utiliser un type différent de régularisation basée sur une norme qui est un ratio de normes l_1/l_q [Liu and Ye, 2010, Bach et al., 2012]. Cela permet de maximiser dynamiquement l'aspect éparse d'un modèle avec un hyper-paramètre (q) sans itérations additionnelles dans l'apprentissage et sans perte de précision significative tout en étant simple à implémenter.

3.3 Entraîner un modèle avec la méthode de noyaux éparses

Nous nous concentrons sur le fait d'augmenter la structure éparse des couches de convolution pour régulariser la structure des DCNN durant l'étape d'apprentissage. Nous proposons une méthode générique afin de régulariser les DCNN en utilisant une pseudo-norme $\frac{l_1}{l_2}$.

3.3.1 Méthode de régularisation de la structure éparse des noyaux

Déclarons $\mathcal{N}$ un DCNN composé de L couches de convolution. Nous définissons $W^{l,k}$ le $k^{ieme} \in \{1, ..N_k^l\}$ tenseur 3D (noyau) associé à la l^{ieme} couche de convolution. Ainsi, un poids du kernel k dans la couche de convolution l est défini comme :

$$W_{w,h,c}^{l,k} \in \mathbb{R}^{N_w^l, N_h^l, N_c^l} \tag{3.1}$$

Ici, $w \in \{1, ..N_w^l\}$ est la largeur (colonne), $h \in \{1, ..N_h^l\}$ est la hauteur (ligne) et $c \in \{1, ..N_c^l\}$ est l'index (profondeur) du canal de la matrice du noyau k dans la couche de convolution l. L'idée principale de cette contribution est d'induire une structure éparse sur des pseudo-normes de noyaux. Notons n_k^l comme étant la pseudo-norme définie par la norme l_1 des composants du noyau $W^{l,k}$:

$$n_k^l \doteq \sum_{w=1}^{N_w^l} \sum_{h=1}^{N_h^l} \sum_{c=1}^{N_c^l} \frac{|W_{w,h,c}^{l,k}|}{N_k^l} \tag{3.2}$$

Le vecteur $\mathbf{N}^l$ concatène, pour la couche l, les N_k^l normes n_k^l :

$$\mathbf{N}^l \doteq \Big\|_{k=1}^{N_k^l} n_k^l \tag{3.3}$$

Nous introduisons la structure éparse d'un noyau pour une couche comme étant une valeur liée au nombre de noyaux de cette couche avec tous les poids égaux à zéro. Ainsi, la structure éparse de noyaux de la couche l peut être liée au nombre de valeurs du vecteur $\mathbf{N}^l$ qui est égal à zéro. La structure éparse globale des noyaux peut être exprimée à partir de la concaténation des vecteurs $\mathbf{N}^l$ de toutes les couches :

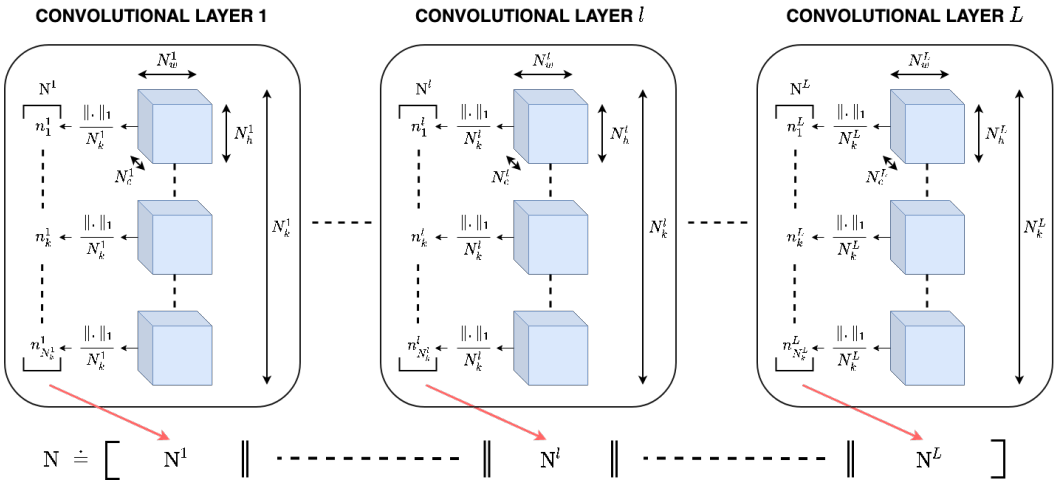


FIGURE 3.2 – Visualisation du calcul des pseudo-normes des noyaux et de la construction du vecteur global des pseudo-normes des noyaux $\vec{\mathbf{N}}$.

$$\mathbf{N} \doteq \Big\|_{l=1}^{L} \mathbf{N}^l \tag{3.4}$$

Pour une meilleure compréhension, il est possible de visualiser ces opérations sur la Figure 3.2. Afin de normaliser les valeurs du vecteur $\mathbf{N}$, chacun de ses composants est divisé par le nombre de valeurs (ou de normes) qu'il contient. Finalement, la structure éparse globale est définie par la structure éparse de $\mathbf{N}$ et est estimée par une fonction qui est un ratio l_1/l_2 :

$$\mathcal{L}_s \doteq \frac{\mathbf{N}_1}{\mathbf{N}_2} \tag{3.5}$$

Minimiser ce terme va pousser des coefficients à prendre une valeur proche de zéro (numérateur), tout en poussant d'autres coefficients à prendre des valeurs plus grandes (dénominateur), produisant ainsi des couches de convolution avec des noyaux possédant des poids proches de zéro.

3.3.2 Entraîner un modèle avec la méthode de régularisation noyaux éparses

Notons $\mathcal{L}_\mathcal{N}$ comme étant la fonction de coût qui est minimisée pour trouver la configuration des poids optimale pour une tâche donnée (une fonction d'entropie croisée par exemple). Nous proposons de lui ajouter le terme de régularisation de la structure éparse pondéré par le coefficient $\lambda \in \mathbb{R}$:

$$\mathcal{L}_{all} = \mathcal{L}_\mathcal{N} + \lambda \mathcal{L}_s \ . \tag{3.6}$$

Nous discutons du réglage de la valeur de ce coefficient dans la section 3.4.

3.3.3 Mettre des noyaux à zéro

Cet algorithme induit une structure éparse dans un DCNN. C'est à dire que la régularisation des pseudo-normes permet de pousser certains noyaux à avoir des poids mis à zéro. Cependant, dans la pratique et pendant l'optimisation, la valeur réelle de ces noyaux ne sera pas exactement zéro mais une valeur très petite. Ainsi, pour compresser un réseau de manière efficace, l'algorithme identifie ces noyaux durant l'apprentissage et

les force à prendre une valeur nulle pour pouvoir les supprimer.

De manière plus spécifique, l'algorithme se déroule comme suit : chaque pseudo-norme des noyaux du modèle est contenue dans le vecteur global $\mathbf{N}$. Ainsi, à chaque epoch, nous normalisons les valeurs de $\mathbf{N}$ de telle sorte que $\sum_{i=1}^{K} \mathbf{N}_i = 1$. Trier ces valeurs dans l'ordre croissant va permettre de déterminer de manière objective quelles pseudo-normes sont les plus petites. Nous définissons ensuite un pourcentage (ou un seuil) sous lequel la somme cumulée des valeurs rangées les plus petites est jugée trop petite pour être pertinente. Les filtres correspondant à ces valeurs sont donc considérés comme non importants et mis à zéro. Une fois les poids d'un noyau mis à zéro, ils conservent cette valeur jusqu'à la fin de l'apprentissage et ne sont plus mis à jour. Cela permet d'assurer que les erreurs potentielles et l'imprécision introduite par la suppression de ces noyaux soient compensées par les noyaux restants pour la suite de l'apprentissage, permettant ainsi de converger vers une solution stable.

Pour résumer, la contribution présentée ici consiste en deux étapes à chaque epoch :

1. La pseudo-norme l_1/l_2 est calculée pour chaque noyau du modèle et est intégrée à la fonction de coût. Ainsi, la phase d'apprentissage minimise la fonction de coût et induit une structure éparse au niveau des noyaux, poussant ainsi certains poids à avoir une valeur proche de zéro.

2. Trier les noyaux dans l'ordre croissant des pseudo-normes et calculer un vecteur de sommes cumulées à partir du vecteur de pseudo-normes normalisées. Les noyaux participant à cette somme cumulée et qui sont sous un seuil t donné sont supprimés. Cet enchaînement d'opérations vise à garder les noyaux qui produisent plus de t% de la norme globale.

3.4 Expérimentations

Nous avons évalué les performances de la pseudo-norme l_1/l_2 sur deux modèles de classification (*LeNet* et *VGG*) et deux jeux de données : MNIST et CIFAR-10. La méthode est implémentée en *Pytorch*, et entraînée sur différents GPU NVIDIA utilisant CUDA. Les poids du réseaux sont initialisés aléatoirement et les hyper-paramètres sont sélectionnés manuellement pour des résultats optimaux. La valeur choisie pour l'hyper-paramètre λ est une valeur permettant au modèle d'obtenir une précision équivalente ou proche de la précision du modèle de base tout en rendant éparse le plus de normes de noyaux possible. Dans toutes les expérimentations, le seuil en-dessous duquel les noyaux sont supprimés en évaluant la somme cumulée des plus petites pseudo-normes est réglée à 1%. Nous trouvons que cette valeur est la meilleure balance entre une précision qui converge pour les modèles et une suppression progressive des noyaux durant la phase d'apprentissage.

3.4.1 Expérimentations sur *LeNet*

Dans nos expérimentations avec *Lenet* [LeCun et al., 1998], nous nous intéressons à l'efficacité de la pseudo-norme l_1/l_2 sur les jeux de données MNIST et CIFAR-10. Afin de pouvoir comparer nos résultats avec d'autres méthodes de l'état de l'art comme SSL [Wen

et al., 2016], NISP [Yu et al., 2018] et GAL [Lin et al., 2019], nous avons décidé de choisir le modèle *LeNet* implémenté par *Caffe*. La méthode SSL [Wen et al., 2016] utilise une régression de Lasso groupée à différents niveaux d'une couche de convolution pendant l'apprentissage afin de compresser sa structure. La méthode NISP [Yu et al., 2018] calcule un score de propagation de l'information pour chaque neurone afin d'évaluer leur importance. La méthode GAL [Lin et al., 2019] utilise une méthode d'apprentissage antagoniste générative afin de supprimer les structures redondantes (filtres, branches ou blocs). Il n'y a pas de *data augmentation* pour l'apprentissage sur les deux jeux de données.

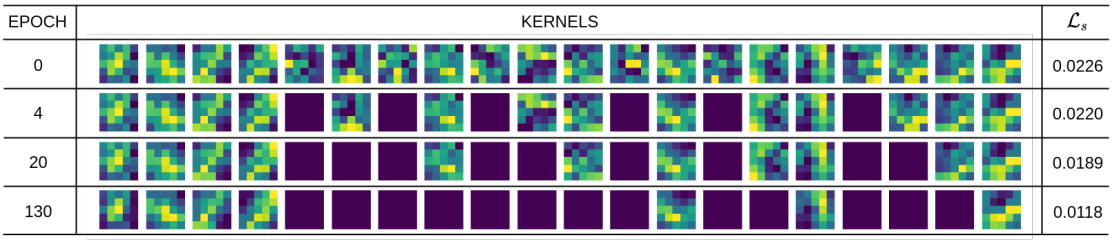


FIGURE 3.3 – Evolution des 20 noyaux de la première couche de convolution de *LeNet* et du terme de régularisation de la structure éparse des noyaux $\mathcal{L}_s$ durant un apprentissage sur le jeu de données MNIST. Pour une meilleure visualisation, chaque noyau est aplati de 3 dimensions vers 2 dimensions.

LeNet sur MNIST : comme précédemment expliqué, la pseudo-norme l_1/l_2 est appliquée aux filtres d'un DCNN afin de les pénaliser. Ainsi, cette méthode introduit de la structure éparse parmi les filtres des couches de convolution de *LeNet*. Afin de visualiser les effets de cette approche sur les noyaux, la Figure 3.3 montre l'évolution des noyaux de la première couche de convolution de *LeNet* durant l'entraînement de MNIST avec cette méthode de régularisation de la structure éparse des noyaux. Nous voyons que le terme de régularisation $\mathcal{L}_s$ décroît epoch après epoch et que le nombre de filtres de la couche décroît aussi en parallèle. $\mathcal{L}_s$ étant calculé sur les pseudo-normes des noyaux et les noyaux étant mis à zéro au cours du temps : ces résultats montrent l'efficacité de cette méthode.

La Table 3.1 résume les résultats sur MNIST de différentes méthodes. Dans le meilleur des scénarios que nous avons testé, la pseudo-norme l_1/l_2 avec une valeur λ mise à 0.5 est capable d'atteindre une meilleure précision que le modèle de base de 0.2%. De plus, le nombre de filtres diminue drastiquement dans les deux couches de convolution respectivement de 20 à 5 et de 50 à 18. Comparé à d'autres méthodes de l'état de l'art, la pseudo-norme l_1/l_2 est capable d'atteindre une précision supérieure tout en pénalisant plus de filtres. Nous avons aussi comparé cette méthode à la norme l_1 et la norme l_2. Pendant nos évaluations, ces deux normes ont été capables d'atteindre une structure de noyaux plus éparse que cette méthode. C'est à dire que plus de noyaux ont été mis à zéro. Cependant, elles n'ont jamais été capables d'atteindre une précision au moins aussi bonne que celle du modèle de base.

Afin de visualiser l'effet de cette contribution sur les paramètres, nous montrons les filtres appris par la première couche de convolution sur la Figure 3.4. Avec $\lambda = 0.5$ et pour différents niveaux de structures éparses, nous pouvons voir que le nombre de filtres restants peut-être diminué jusqu'à seulement deux ou quatre. Ainsi, entre le modèle de base et celui utilisant la pseudo-norme l_1/l_2, la précision est la même ou meilleure avec la pseudo-norme. Cela montre qu'il y a effectivement une grande redondance entre les filtres et que la plupart d'entre eux ne sont donc pas essentiels. De plus, comparé au

modèle de base, il semble que les filtres restants soient plus structurés, avec des motifs plus réguliers. La supposition semble particulièrement vraie lorsqu'il y a seulement deux filtres restants. Ainsi, nous arrivons à la même conclusion que [Wen et al., 2016] : le modèle de base à une grande liberté dans son espace de recherche de paramètres et la pseudo-norme l_1/l_2 est capable d'obtenir la même précision en optimisant les filtres pour avoir des motifs plus réguliers.

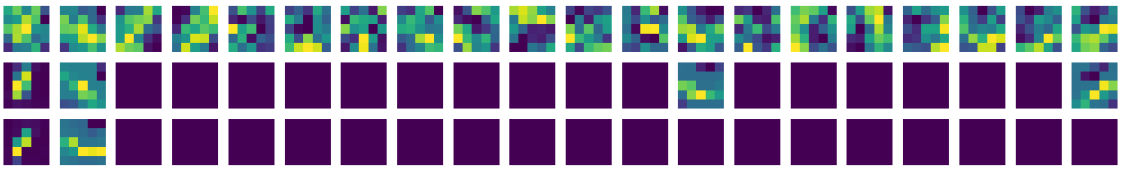


FIGURE 3.4 – Filtres appris par la première couche de convolution de *LeNet* sur MNIST. La première ligne est le modèle *LeNet* de base, celles du milieu et du bas sont la pseudo-norme l_1/l_2 avec $\lambda = 0.5$ et différents niveaux de structure éparse.

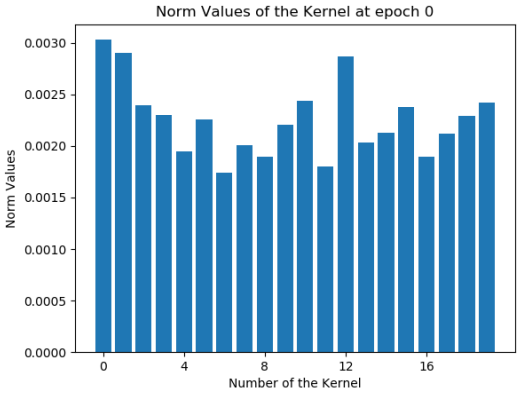
Method	λ	Error	Conv1 Filter # (Sparsity)	Conv2 filter # (Sparsity)	Total Sparsity
Baseline	-	**0.9%**	20	50	**0%**
l_1	0.5	1.2%	4 (80%)	5 (90%)	87.1%
l_2	0.5	1.2%	3 (85%)	5 (90%)	88.6%
SSL 1	-	**0.8%**	5 (75%)	19 (62%)	**65.7%**
SSL 2	-	1.0%	3 (85%)	12 (76%)	78.6%
NISP	-	**0.8%**	10 (50%)	25 (50%)	**50.0%**
GAL	-	1.0%	2 (90%)	15 (70%)	75.7%
l_1/l_2	0.5	**0.7%**	5 (75%)	18 (64%)	**67.1%**

TABLE 3.1 – Résultats après avoir pénalisé des filtres non-essentiels de *LeNet* sur MNIST. Baseline est le modèle simple *LeNet* provenant de Caffe. l_1 et l_2 sont les meilleurs résultats trouvés en utilisant la régularisation par la norme l_1 ou la norme l_2 sur les noyaux. SSL, NISP et GAL sont les méthodes d'élagage provenant respectivement de [Wen et al., 2016], [Yu et al., 2018] et [Lin et al., 2019]. l_1/l_2 est la contribution de ce chapitre avec $\lambda = 0.5$.

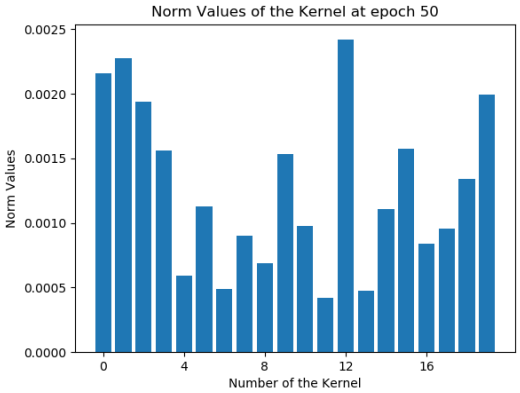
Dans le but de montrer d'une manière plus visuelle l'effet de la pseudo-norme l_1/l_2 sur un apprentissage, nous proposons de montrer en détails, et à travers différentes figures, un apprentissage effectué avec le réseau *LeNet* sur MNIST. Dans cet apprentissage, la pseudo-norme l_1/l_2 est effective dès le début (avec une valeur $\lambda = 0.5$) mais nous avons décidé de mettre la valeur des filtres à zéro à partir de l'epoch 50 afin de mieux visualiser l'effet de dispersion des valeurs induit par la pseudo-norme l_1/l_2 dans les premières epochs. Ainsi il est possible de trouver différentes figures dans les pages suivantes. Pour chacune des deux couches de convolution de *LeNet*, nous avons tracé

des histogrammes de la valeur de chacun des noyaux à différentes étapes de l'apprentissage. La Figure 3.5 présente les valeurs des noyaux de la première couche de convolution tandis que la Figure 3.7 présente les valeurs des noyaux de la première couche de convolution de *LeNet*. Il est aussi possible de voir, en complément, la fréquence des valeurs de ces noyaux au cours de l'apprentissage, respectivement sur la Figure 3.6 pour la première couche de convolution et sur la Figure 3.8 pour la deuxième couche de convolution. Enfin, la précision du modèle et la valeur de la fonction de coût tout au long de l'apprentissage sont visibles respectivement sur la Figure 3.9 et la Figure 3.10, aussi bien pour le jeu de données d'entraînement que pour le jeu de données de validation.

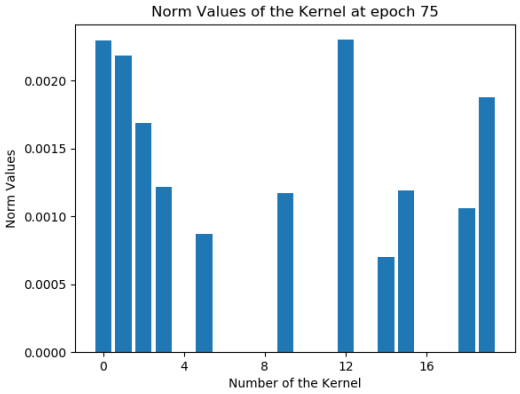
Nous pouvons remarquer sur l'ensemble de ces figures plusieurs effets de la pseudo-norme l_1/l_2. Bien qu'aucun filtre ne soit mis à zéro au début de l'apprentissage jusqu'à l'epoch 50, on peut voir que la pseudo-norme l_1/l_2 permet déjà d'introduire de la dispersion dans la valeur des filtres, certains prenant une valeur plus élevée et d'autres prenant une valeur plus faible. Après l'epoch 50, des filtres commencent à être supprimés. Plus on avance dans le temps et plus le nombre de filtres supprimés est réduit. Autrement dit, s'il reste peu de filtres, la méthode aura tendance a moins en supprimer rapidement. On remarque aussi que la méthode est capable de garder un niveau de précision et une valeur de fonction de coût stables même après la suppression d'un bon nombre de filtres. Nous arrivons tout de même toujours à un point de décrochage où trop de filtres sont supprimés et il n'est plus possible pour le modèle de retrouver un niveau de performance satisfaisant.



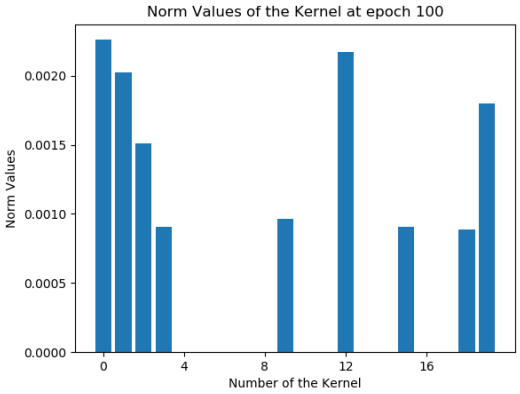
(A) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 0.



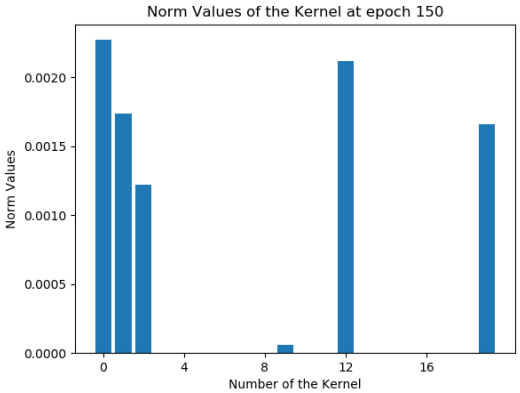
(B) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 50.



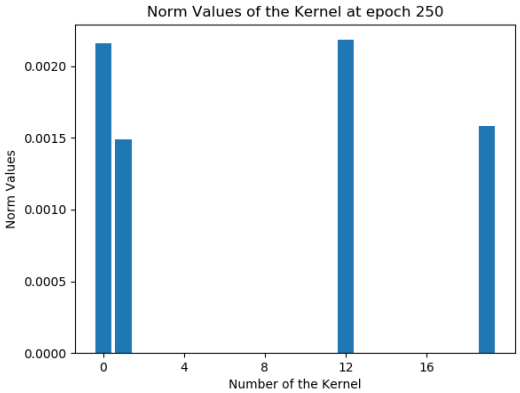
(C) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 75.



(D) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 100.

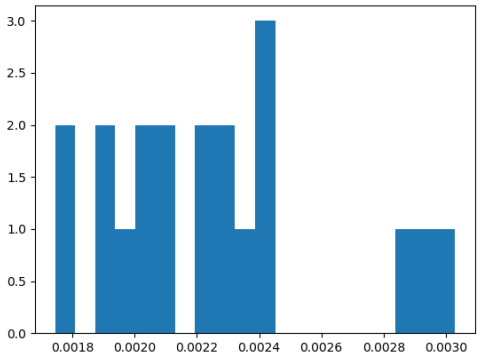


(E) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 150.

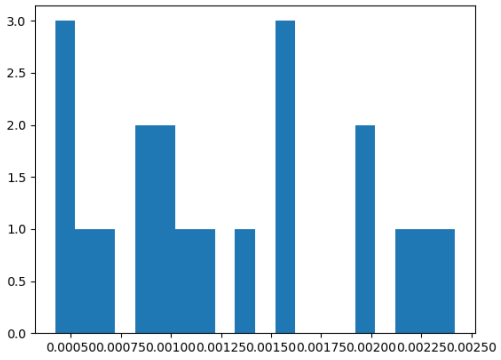


(F) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 250.

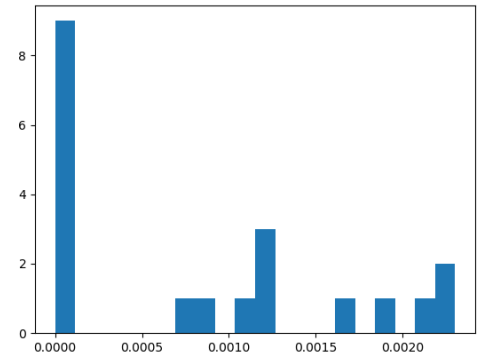
FIGURE 3.5 – Évolution de la valeur de chacun des noyaux de la première couche de convolution de *LeNet* pendant un apprentissage sur MNIST avec la norme l_1/l_2. Les noyaux sont mis à zéro à partir de l'epoch 50 ($\lambda = 0.5$). La couche est composée de 20 noyaux.



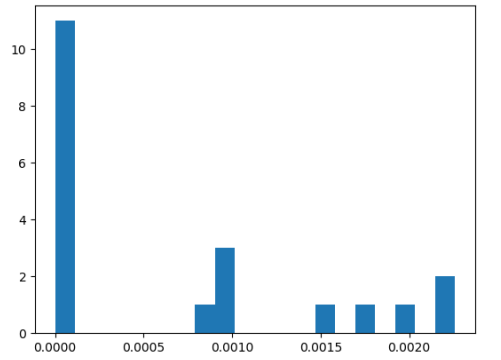
(A) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 0.



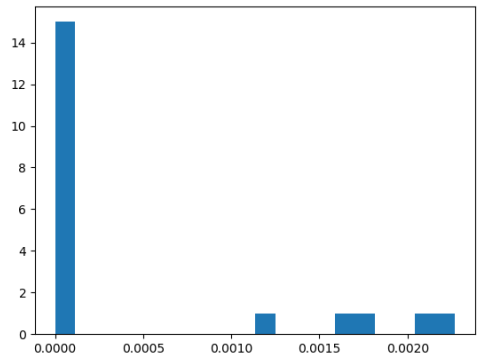
(B) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 50.



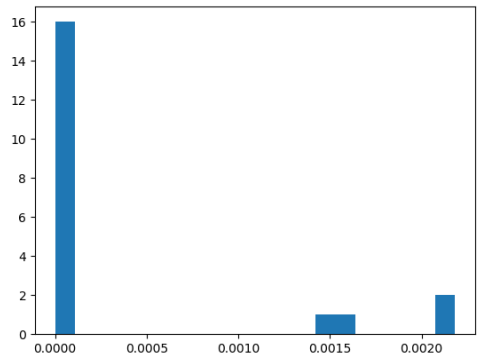
(C) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 75.



(D) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 100.

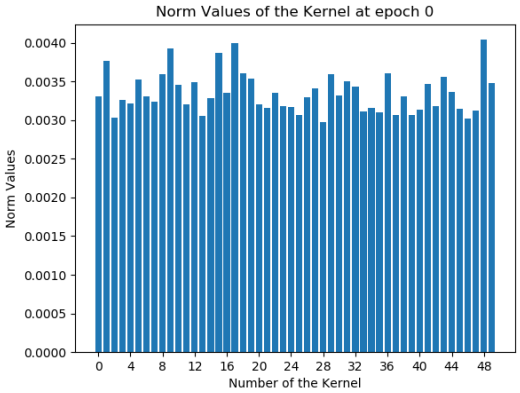


(E) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 150.

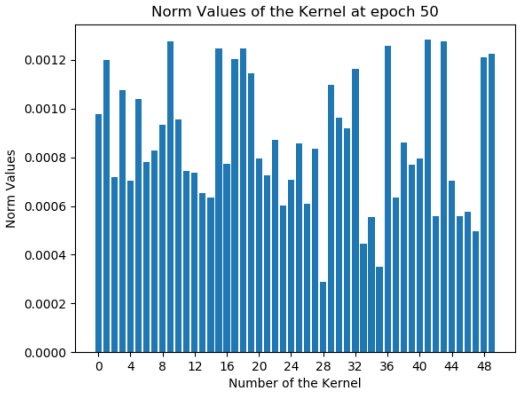


(F) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 250.

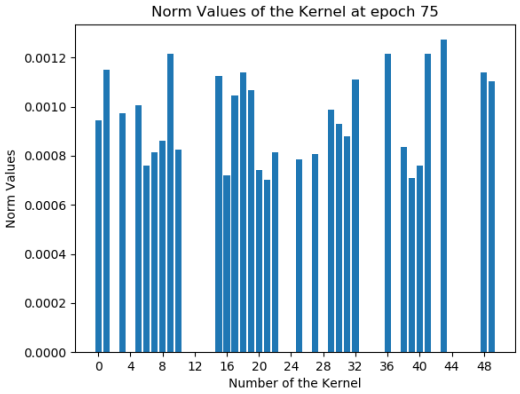
FIGURE 3.6 – Fréquence de la valeur de chacun des noyaux de la première couche de convolution de *LeNet* pendant un apprentissage sur MNIST avec la norme l_1/l_2. Les noyaux sont mis à zéro à partir de l'epoch 50 ($\lambda = 0.5$). La couche est composée de 20 noyaux.



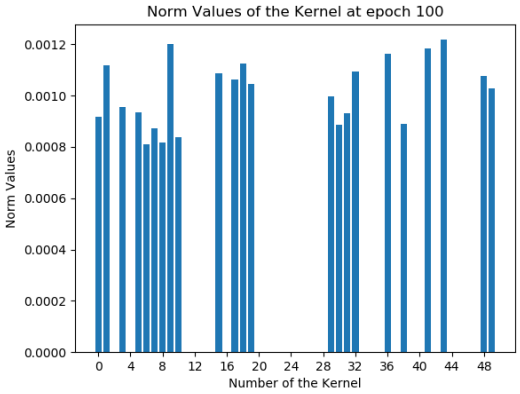
(A) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 0.



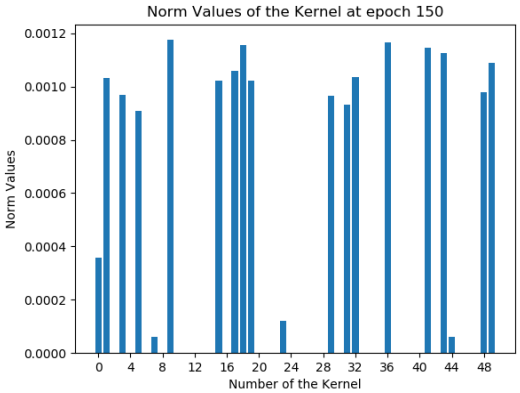
(B) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 50.



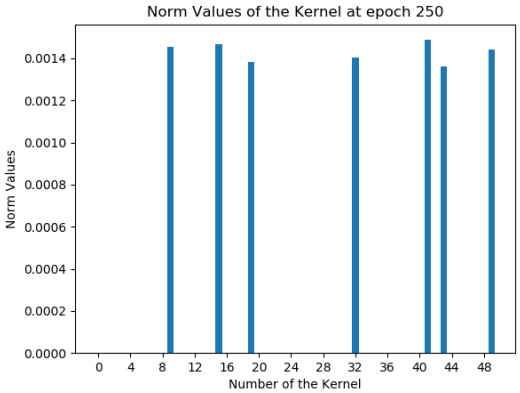
(C) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 75.



(D) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 100.

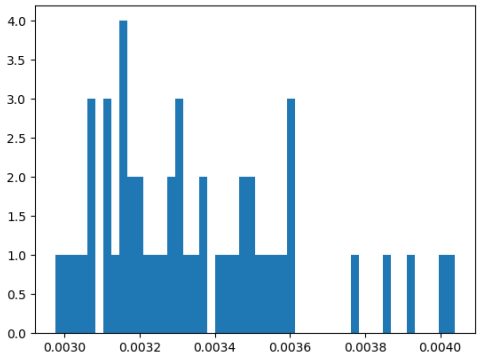


(E) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 150.

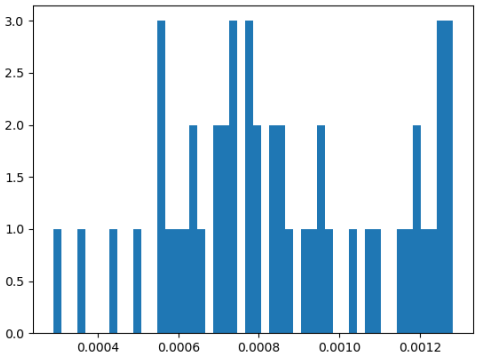


(F) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 250.

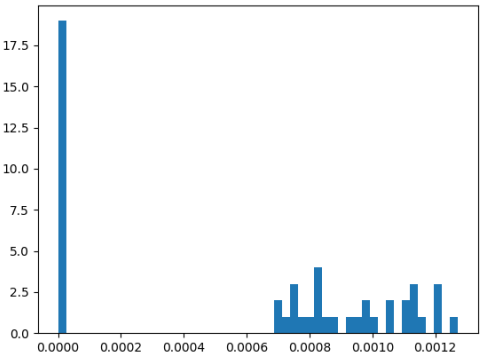
FIGURE 3.7 – Évolution de la valeur de chacun des noyaux de la première couche de convolution de *LeNet* pendant un apprentissage sur MNIST avec la norme l_1/l_2. Les noyaux sont mis à zéro à partir de l'epoch 50 ($\lambda = 0.5$).La couche est composée de 50 noyaux.



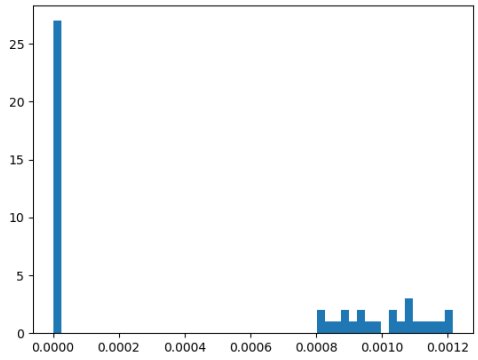
(A) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 0.



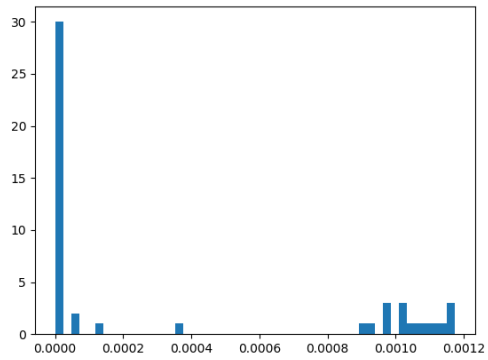
(B) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 50.



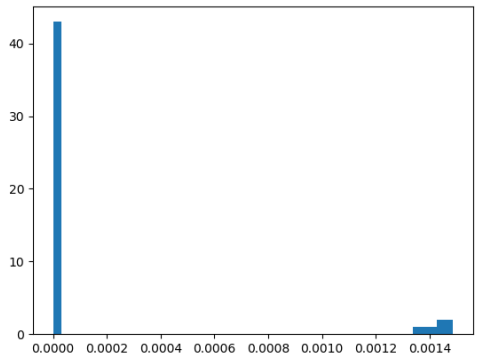
(C) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 75.



(D) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 100.

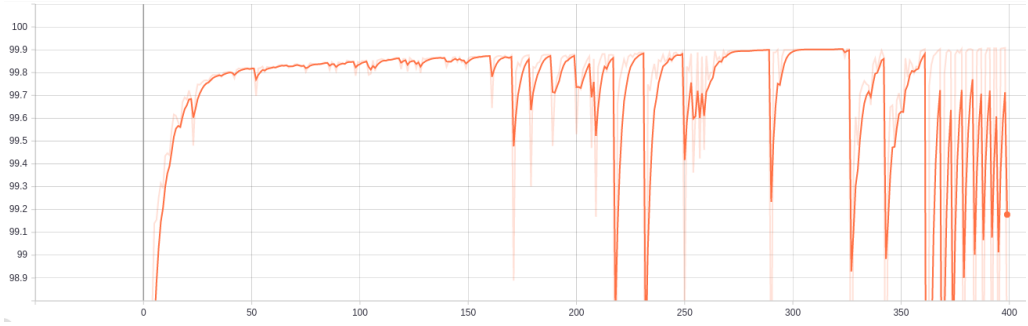


(E) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 150.

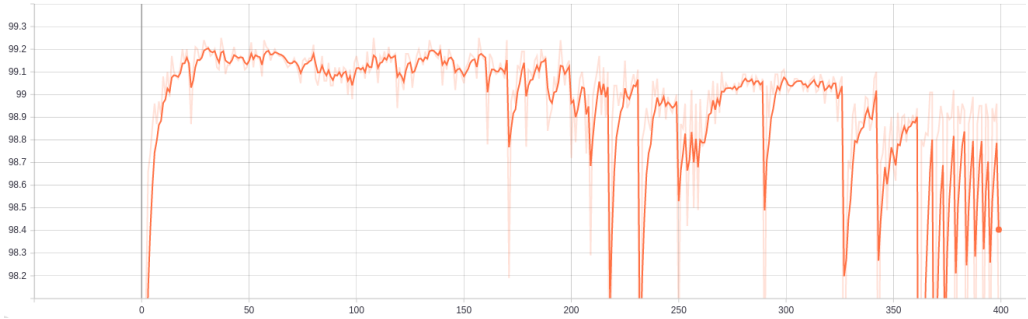


(F) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 250.

FIGURE 3.8 – Fréquence de la valeur de chacun des noyaux de la première couche de convolution de *LeNet* pendant un apprentissage sur MNIST avec la norme l_1/l_2. Les noyaux sont mis à zéro à partir de l'epoch 50 ($\lambda = 0.5$). La couche est composée de 50 noyaux.

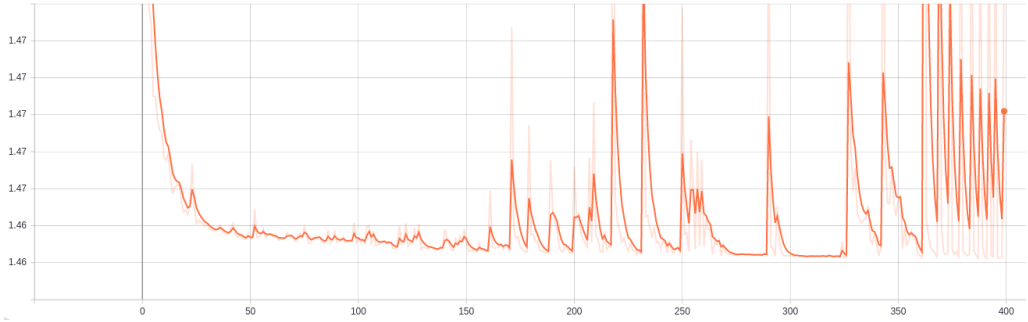


(A) Précision de modèle (Ordonnée) sur la base de données d'entraînement en fonction du nombre d'epoch écoulé (Abscisse).

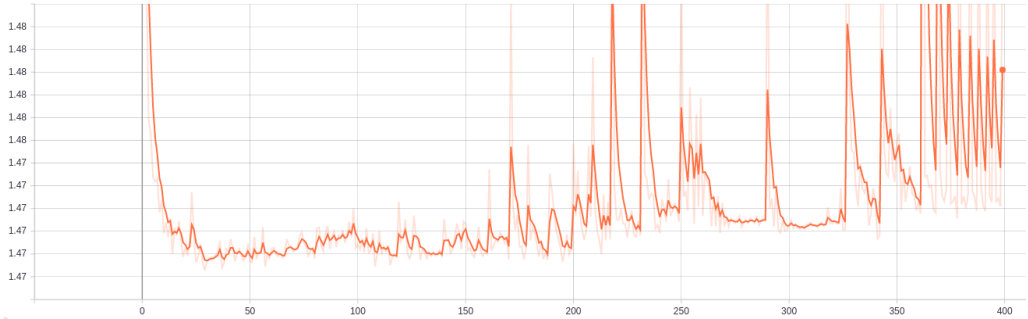


(B) Précision de modèle (Ordonnée) sur la base de données de validation en fonction du nombre d'epoch écoulé (Abscisse).

FIGURE 3.9 – Précisions d'un modèle *LeNet* pendant un apprentissage sur MNIST avec la norme l_1/l_2. Les noyaux sont mis à zéro à partir de l'epoch 50 ($\lambda = 0.5$).



(A) Valeur de la fonction de coût du modèle (Ordonnée) sur la base de données d'entraînement en fonction du nombre d'epoch écoulé (Abscisse).



(B) Valeur de la fonction de coût du modèle (Ordonnée) sur la base de données de validation en fonction du nombre d'epoch écoulé (Abscisse).

FIGURE 3.10 – Valeurs des fonctions de coût du modèle *LeNet* pendant un apprentissage sur MNIST avec la norme l_1/l_2. Les noyaux sont mis à zéro à partir de l'epoch 50 ($\lambda = 0.5$).

LeNet sur CIFAR-10 : afin de tester la pseudo-norme l_1/l_2 et de visualiser ses effets sur une tache de classification plus difficile que le jeu de données MNIST, nous avons décidé d'utiliser le jeu de données CIFAR-10 avec le même modèle *LeNet*. Les résultats sont consignés dans la Table 3.2. Le modèle *LeNet* de base ne performe pas aussi bien sur le jeu de données CIFAR-10 que sur le jeu de données MNIST. En effet, la précision de la classification est seulement aux alentours de 70%. Ainsi, la précision de la méthode diminue aussi mais la pseudo-norme l_1/l_2 est tout de même capable de bonnes performances sur ce modèle, même sur cette tâche de classification. Avec une valeur de λ égale à 0.7, nous sommes capables de diminuer le nombre de filtres de la première et de la deuxième couche de convolution respectivement de 20 à 10 et de 50 à 25. La moitié des filtres de *LeNet* peut donc être supprimée. Dans cette configuration, la méthode basée sur la pseudo-norme obtient une précision inférieure de 1.7% par rapport aux performances du modèle de base. Nous avons été capables de supprimer jusqu'à 80% des filtres dans nos expérimentations, mais les précisions associées à ces réductions sont trop basses pour être intéressantes (plus de 20% de perte par rapport au modèle de base). Ainsi, plus de filtres sont nécessaires afin de classifier correctement le jeu de données CIFAR-10 par rapport au jeu de données MNIST. Le meilleur compromis entre nombre de filtres et précision du modèle que nous avons trouvé est toujours avec une valeur de λ égale à 0.7. Dans les deux couches de convolution, le nombre de filtres chute respectivement de 20 à 14 et de 50 à 30. Cela veut dire que cette contribution est capable de mettre à zéro plus d'un tiers des filtres avec une baisse de précision de seulement 0.9%. Comparé à la norme l_1 et à la norme l_2, la pseudo-norme l_1/l_2 montre aussi de bons résultats. En effet, la norme l_1 et la norme l_2 sont toutes les deux incapables d'atteindre un niveau de précision équivalent à la pseudo-norme l_1/l_2 tout en supprimant autant de filtres.

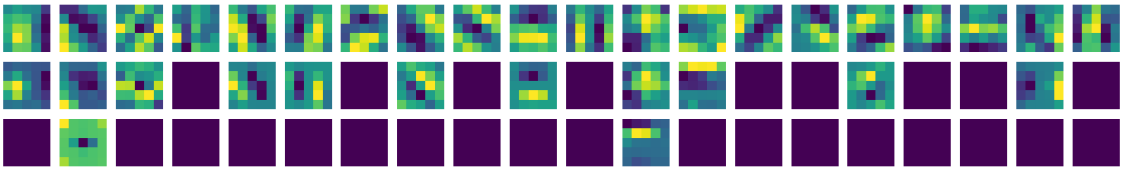


FIGURE 3.11 – Filtres appris de la première couche de convolution de filtres *LeNet* sur CIFAR-10. La première ligne est le modèle *LeNet* de base, celles du milieu et du bas sont la norme l_1/l_2 avec $\lambda = 0.7$ et différents niveaux de structure éparse.

Comme fait précédemment avec le jeu de données MNIST, nous pouvons visualiser les filtres appris par la première couche de convolution sur la Figure 3.11. A partir de cette visualisation, nous pouvons tirer les mêmes conclusions qu'avec le jeu de données MNIST. Plus nous enlevons de filtres, plus les filtres restants semblent avoir une structure définie, à l'opposé des filtres du modèle de base où tous les filtres semblent bruités. Cela est d'autant plus remarquable lorsque nous laissons l'algorithme tourner jusqu'à ce qu'il ne reste plus qu'une paire de filtres dans la couche. Même si le modèle n'atteint pas une précision satisfaisante, les deux filtres restants ont appris des motifs remarquables. Ainsi, la pseudo-norme l_1/l_2 est toujours capable de lisser un grand espace des paramètres possibles dans un plus petit nombre de filtres possédant des motifs plus régularisés.

3.4.2 *VGG* sur CIFAR-10

Afin de montrer la capacité de généralisation de cette contribution sur des réseaux de neurones plus profonds, nous avons évalué les performances de la pseudo-norme l_1/l_2 sur le modèle *VGG* [Simonyan and Zisserman, 2015], un réseau plus profond que *LeNet*,

Method	λ	Error	Conv1 Filter # (Sparsity)	Conv2 filter # (Sparsity)	Total Sparsity
Baseline	-	**28.4%**	20	50	**0%**
l_1	0.7	**35.4%**	7 (65%)	14 (72%)	**70.0%**
l_2	0.7	29.8%	12 (40%)	24 (52%)	48.6%
l_1/l_2	0.7	30.1%	10 (50%)	25 (50%)	50.0%
l_1/l_2	0.7	**29.3%**	14 (30%)	30 (40%)	**37.1%**

TABLE 3.2 – Résultats après avoir pénalisé des filtres non-essentiels de *LeNet* sur CIFAR-10. Baseline est le modèle simple *LeNet* provenant de Caffe. l_1 et l_2 sont les meilleurs résultats trouvés en utilisant la régularisation par la norme l_1 ou la norme l_2 sur les noyaux. l_1/l_2 est la contribution de ce chapitre avec $\lambda = 0.7$.

avec plusieurs couches de convolution. Un modèle *VGG* peut avoir différentes tailles, influençant donc le nombre de couches qu'il contient. Nous avons choisi un modèle *VGG11* qui contient un total de huit couches de convolution. Nous avons implémenté le modèle en *Pytorch* et l'avons entraîné sur divers GPU NVIDIA en utilisant CUDA. Le modèle est entraîné sans augmentation de données et évalué sur le jeu de données CIFAR-10. Dans cette expérimentation, les pseudo-normes des noyaux ne sont pas normalisées sur tout le réseau, ce qui explique pourquoi les valeurs du coefficient λ sont plus petites que celles utilisées avec *LeNet*.

Method	λ	Error	Conv1 to conv 8 Filter # (sparsity)	Total Sparsity
Baseline	-	**17.6%**	64 - 128 - 256 - 256 - 512 - 512 - 512 - 512	**0%**
l_1	0.0001	19.3%	52 (18.3%) - 128 (0.0%) - 255 (0.4%) - 256 (0.0%) - 175 (65.8%) - 147 (71.3%) - 97 (81.1%) - 123 (75.9%)	55.2%
l_2	0.005	**18.2%**	64 (0.0%) - 128 (0.0%) - 256 (0.0%) - 256 (0.0%) - 511 (0.2%) - 474 (7.4%) - 434 (15.2%) - 299 (41.6%)	**12%**
l_1/l_2	0.005	18.8%	35 (45.3%) - 115 (10.2%) - 238 (7.0%) - 176 (31.3%) - 354 (30.9%) - 195 (61.9%) - 190 (62.9%) - 175 (65.8%)	46.3%
l_1/l_2	0.001	**16.8%**	64 (0.0%) - 128 (0.0%) - 256 (0.0%) - 256 (0.0%) - 512 (0.0%) - 512 (0.0%) - 510 (0.4%) - 380 (25.8%)	**5%**

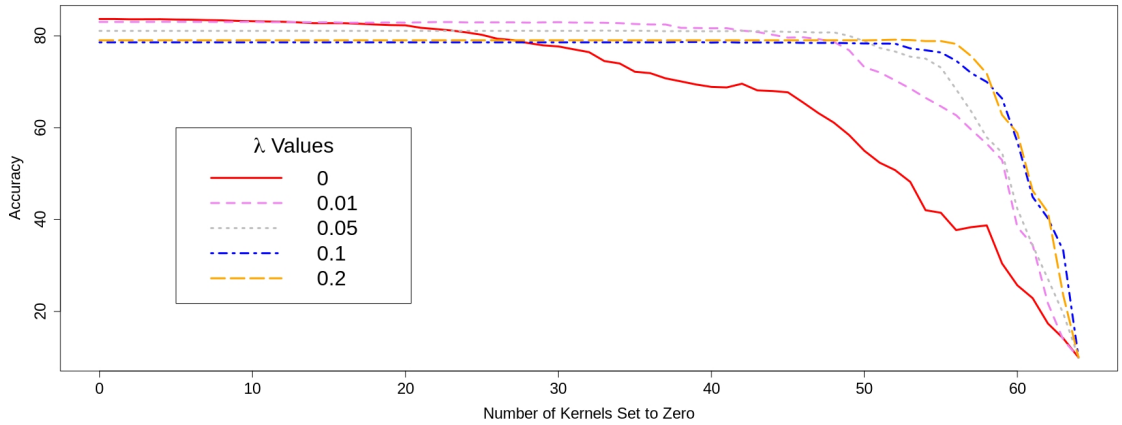
TABLE 3.3 – Résultats après avoir pénalisé des filtres non-essentiels de *VGG11* sur CIFAR-10. Baseline est le modèle simple *VGG11*. l_1/l_2 est la contribution de ce chapitre avec différents coefficients de régularisation λ.

Sur le modèle *LeNet*, la méthode de pseudo-norme l_1/l_2 est appliquée sur seulement deux couches de convolution, avec 50 filtres au plus dans la seconde couche de convolution. Sur le modèle *VGG11*, la méthode est appliquée sur huit couches de convolution

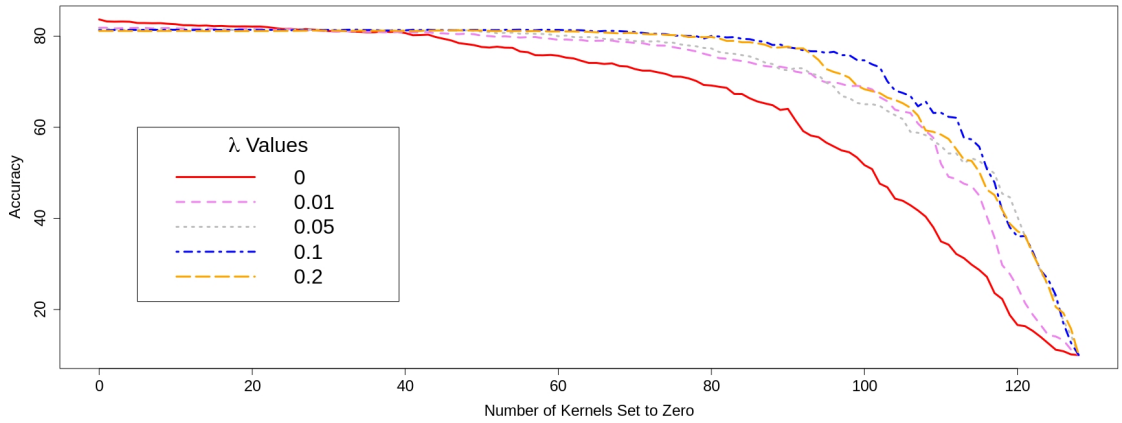
différentes avec un nombre de filtres allant de 64 pour la première couche de convolution jusqu'à 512 pour les 4 dernières couches de convolution. Les résultats sont consignés dans la Table 3.3. Le modèle de base, avec tous les filtres et une fonction de coût classique (une fonction de cross-entropie), obtient une erreur de 17.6% sur la base de test. En utilisant la pseudo-norme l_1/l_2 avec un coefficient λ égal à 0.005, le modèle arrive à une précision de classification environ 1% inférieure au modèle de base. Cependant, le nombre de filtres est grandement réduit. De plus, il semble que plus nous allons profondément dans le réseau, plus la proportion de filtres mis à zéro est importante. Par exemple, la seconde couche de convolution a environ 10% de ses filtres mis à zéro tandis que la dernière couche de convolution voit plus de 65% de ses filtres mis à zéro. Ainsi, nous pouvons en déduire que les dernières couches de convolution gardent moins d'information pour le modèle que les toutes premières ou bien plus simplement qu'il y a plus de redondances dans les dernières couches. Cependant, la première couche de convolution semble être une exception comme approximativement la moitié de ses filtres peuvent être supprimés. Nous supposons que les formes apprises dans la première couche ne sont pas décisives pour le modèle et peuvent être rattrapées par les couches suivantes et par les formes plus précises qu'elles apprennent.

En réduisant le coefficient λ à 0.001, nous confirmons le résultat montrant que les dernières couches de convolution semblent contenir plus de filtres avec de l'information non importante ou redondante que les premières couches de convolution du modèle. En effet, seulement les deux dernières couches possèdent des filtres mis à zéro. Mais plus important encore, la suppression de quelques filtres dans les deux dernières couches de convolution amènent à une erreur de classification de seulement 16.8%, ce qui est une précision 0.8% meilleure que celle du modèle de base. Ainsi, cette contribution, en supprimant seulement quelques filtres, est capable d'atteindre une meilleure précision que celle du modèle de base. Comparée à la norme l_1 et à la norme l_2, la pseudo-norme l_1/l_2 se comporte aussi très bien. La norme l_1 est capable de mettre à zéro de nombreux filtres mais est incapable d'atteindre un niveau de précision satisfaisant, performant toujours moins bien que le modèle de base ou que notre approche. A peu près les mêmes conclusions peuvent être tirées des expérimentations avec la norme l_2. Sous certaines conditions, la norme l_2 est capable de mettre à zéro un peu plus de filtres que la pseudo-norme l_1/l_2 dans la dernière couche de convolution. Cependant, les modèles ne sont pas capables d'atteindre une précision satisfaisante, en restant toujours environ 1% derrière la précision du modèle de base.

Afin de conclure cette étude, nous visualisons sur la Figure 3.12 l'évolution de la précision du modèle par rapport au nombre de noyaux qui sont mis à zéro dans la première et la deuxième couche de convolution pour différents coefficients de régularisation λ. Quand $\lambda = 0$, la pseudo-norme l_1/l_2 n'est pas prise en compte, ce qui revient au modèle de base. L'ordre selon lequel les filtres sont mis à zéro est déterminé par la pseudo-norme des filtres rangée en ordre croissant. Les filtres correspondant aux plus petites pseudo-normes sont supprimés en premier. Ces tests sont effectués sur une seule couche de convolution. Ce qui veut dire que pendant l'apprentissage, les seuls filtres qui sont évalués et mis à zéro sont ceux appartenant à la couche étudiée. Les autres couches ne sont pas modifiées. On peut remarquer que pour les deux couches, nous sommes capables de mettre de nombreux filtres à zéro sans avoir une diminution importante de la précision, même lorsque la méthode basée sur la pseudo-norme l_1/l_2 n'est pas active. Ce résultat montre qu'il y a de l'information non-essentielle dans la couche et qu'il est possible de la supprimer, même s'il n'y a pas de méthode définie pour accentuer ce phénomène. Avec l'implémentation de la pseudo-norme l_1/l_2 ($\lambda > 0$), nous pouvons voir



(A) Première couche de convolution de *VGG11*



(B) Deuxième couche de convolution de *VGG11*

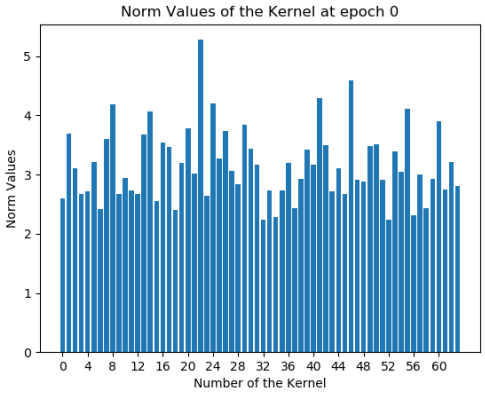
FIGURE 3.12 – Visualisation de l'effet de la mise à zéro de noyaux dans les deux premières couches de convolution de *VGG11* par rapport à la précision du modèle. Chaque courbe représente une valeur différente du coefficient λ pour la méthode basée sur la pseudo-norme l_1/l_2.

que (1) plus de noyaux sont mis à zéro avant le début de la chute de la précision du modèle par rapport au modèle de base, et (2) qu'une plus grande valeur de λ veut dire que plus de noyaux sont mis à zéro mais au prix d'une chute plus importante de la précision. A partir de ces conclusions, nous pouvons dire que la contribution de ce chapitre permet d'augmenter la structure éparse des filtres dans une couche, transférant de l'information entre eux afin de mieux la centraliser dans certains filtres. Cependant, cette structure éparse à ses limites. Plus nous essayons de l'augmenter (en augmentant la valeur de λ), plus nous augmentons aussi les chances de perdre de l'information qui ne pourra jamais être retrouvée.

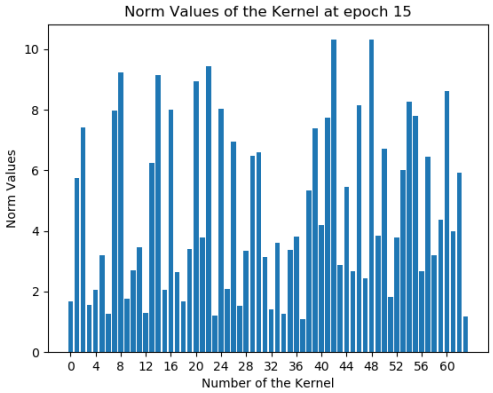
De la même manière que pour le modèle *LeNet*, nous proposons de montrer visuellement l'effet de la pseudo-norme l_1/l_2 sur un modèle *VGG11* pendant l'apprentissage du jeu de données CIFAR-10. Dans cet apprentissage, la pseudo-norme l_1/l_2 est effective dès le début (avec une valeur $\lambda = 0.005$) et les filtres sont aussi supprimés dès la première epoch de l'apprentissage. Dans les prochaines pages, il est ainsi possible de trouver les figures suivantes : des histogrammes de la valeur de chacun des noyaux au cours de l'apprentissage pour la première (Figure 3.13), la deuxième (Figure 3.15) et la dernière

(Figure 3.17) couche de convolution du modèle. La fréquence des valeurs de ces noyaux au cours de l'apprentissage est aussi visible pour la première et la deuxième couche de convolution respectivement sur la Figure 3.14 et la Figure 3.16. La fréquence des valeurs des noyaux de la dernière couche de convolution n'est pas montrée car les noyaux sont plus nombreux et ont de nombreuses valeurs différentes rendant ainsi les graphiques illisibles, bien que beaucoup des noyaux de la couche soient mis à zéro. Enfin, la précision du modèle et la valeur de la fonction de coût tout au long de l'apprentissage sont visibles respectivement sur la Figure 3.18 et la Figure 3.19, aussi bien pour le jeu de données d'entraînement que pour le jeu de données de validation.

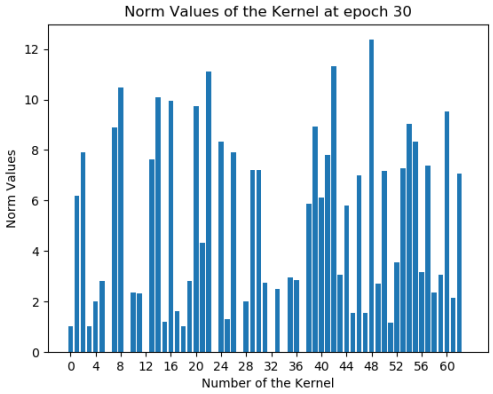
Nous pouvons remarquer plusieurs choses sur ces figures. Comme nous l'avons évoqué précédemment, peu de filtres semblent être mis à zéro dans les premières couches et les filtres sont supprimés de manière progressive tout au long de l'apprentissage. A l'opposé, de nombreux filtres sont supprimés dans la dernière couche et beaucoup d'entre eux sont supprimés dès les premières epochs. Les suppressions sont ensuite moins nombreuses et plus progressives. Une redondance de l'information est potentiellement plus présente dans les dernières couches du modèle que dans les premières permettant à la méthode de supprimer plus de filtres. La précision du modèle ne semble pas décroître au cours du temps malgré la suppression de nombreux filtres, elle semble même ne pas être tout à fait stabilisée et continue de croître. Cela laisse penser que l'apprentissage n'est pas terminé et que nous aurions pu continué l'apprentissage de cet exemple pour obtenir de meilleurs performances. Les mêmes conclusions peuvent être tirées en regardant les courbes représentant les valeurs des fonctions de coût.



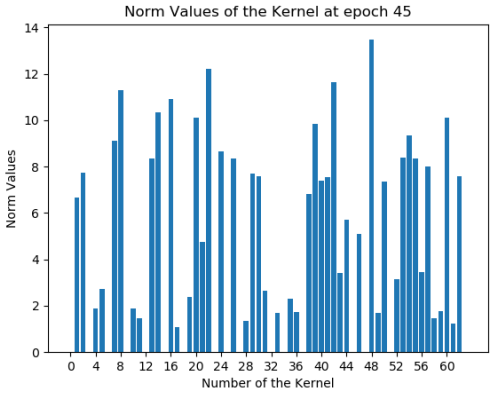
(A) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 0.



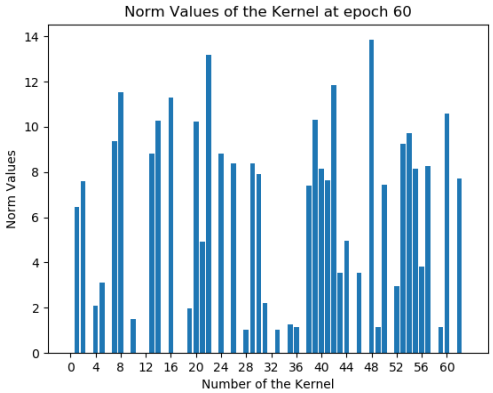
(B) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 15.



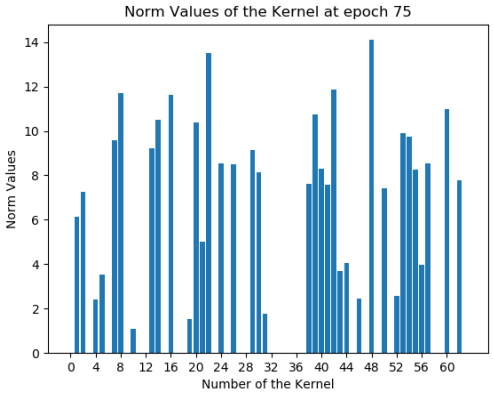
(C) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 30.



(D) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 45.

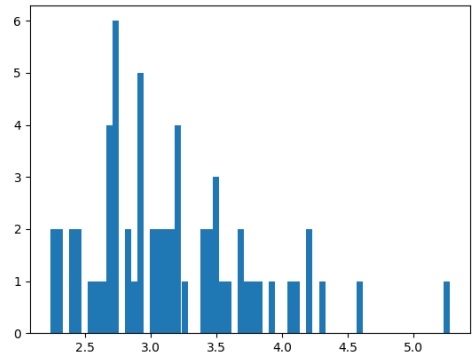


(E) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 60.

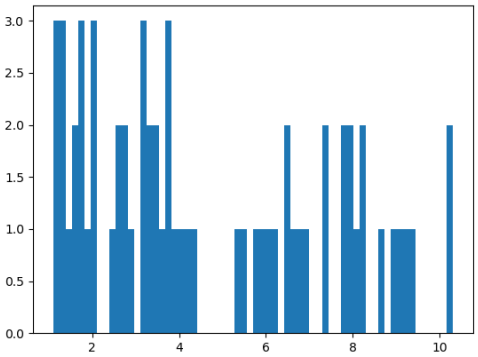


(F) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 75.

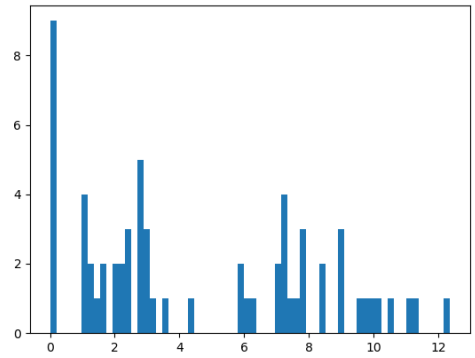
FIGURE 3.13 – Évolution de la valeur de chacun des noyaux de la première couche de convolution de *VGG11* pendant un apprentissage sur CIFAR-10 avec la norme l_1/l_2 ($\lambda = 0.005$). La couche est composée de 64 noyaux.



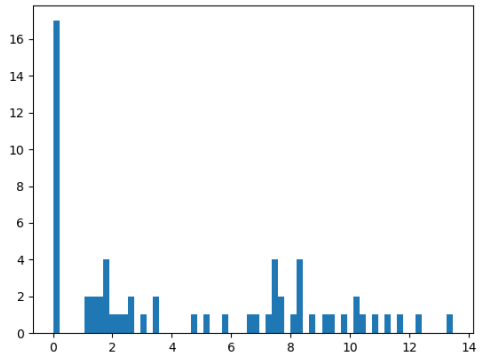
(A) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 0.



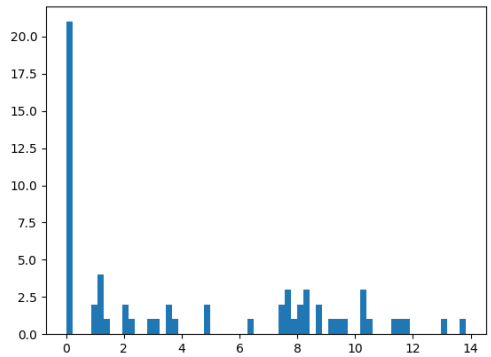
(B) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 15.



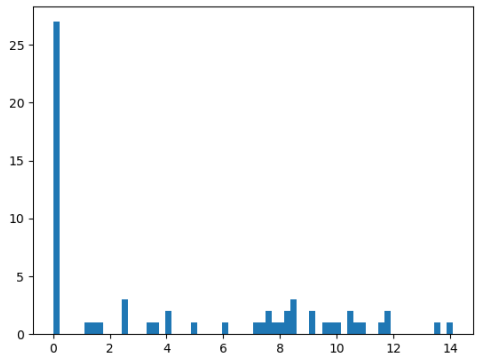
(C) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 30.



(D) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 45.

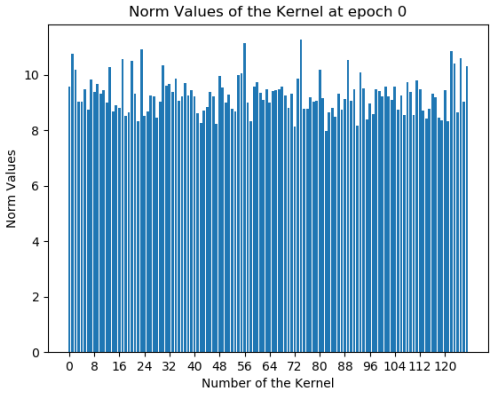


(E) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 60.

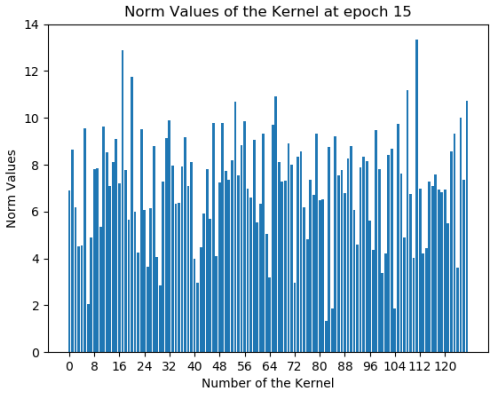


(F) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 75.

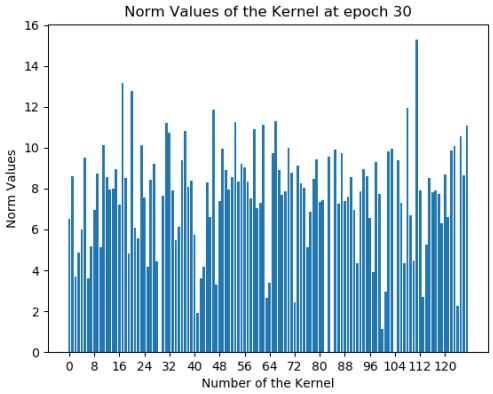
FIGURE 3.14 – Fréquence de la valeur de chacun des noyaux de la première couche de convolution de *VGG11* pendant un apprentissage sur CIFAR-10 avec la norme l_1/l_2 ($\lambda = 0.005$). La couche est composée de 64 noyaux.



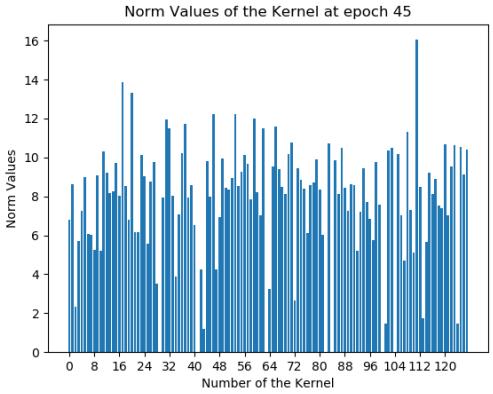
(A) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 0.



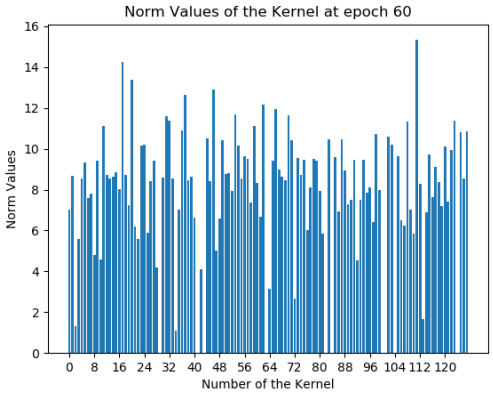
(B) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 15.



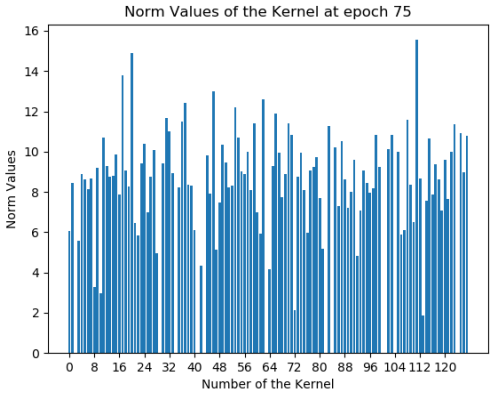
(C) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 30.



(D) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 45.

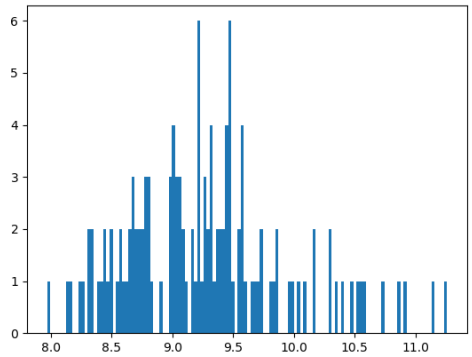


(E) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 60.

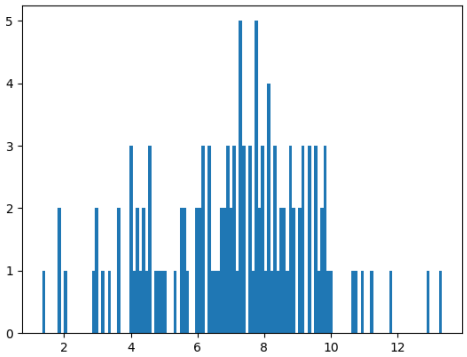


(F) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 75.

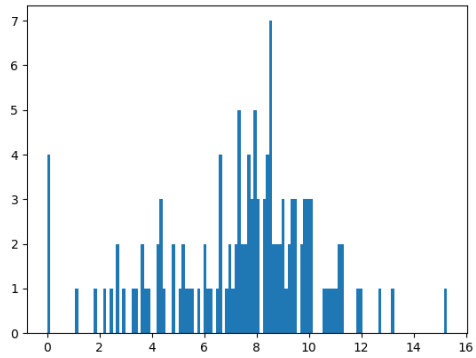
FIGURE 3.15 – Évolution de la valeur de chacun des noyaux de la deuxième couche de convolution de *VGG11* pendant un apprentissage sur CIFAR-10 avec la norme l_1/l_2 ($\lambda = 0.005$). La couche est composée de 128 noyaux.



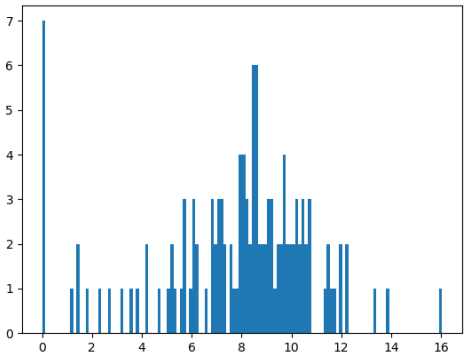
(A) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 0.



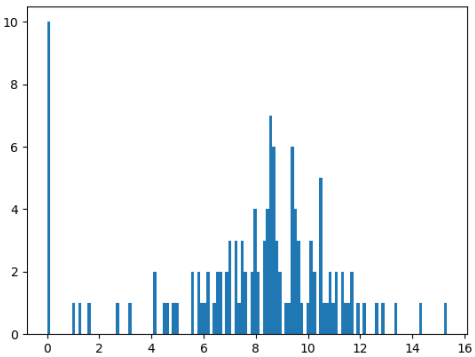
(B) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 15.



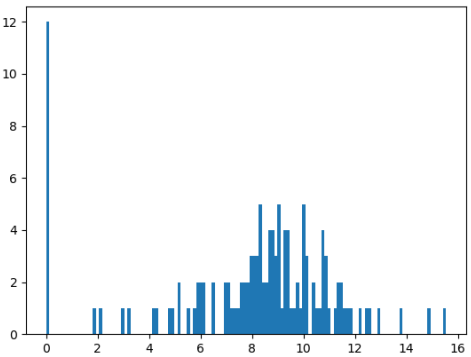
(C) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 30.



(D) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 45.

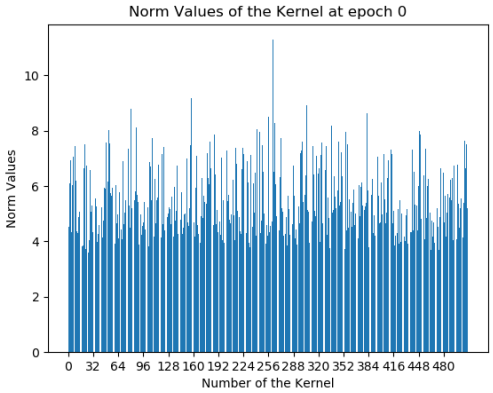


(E) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 60.

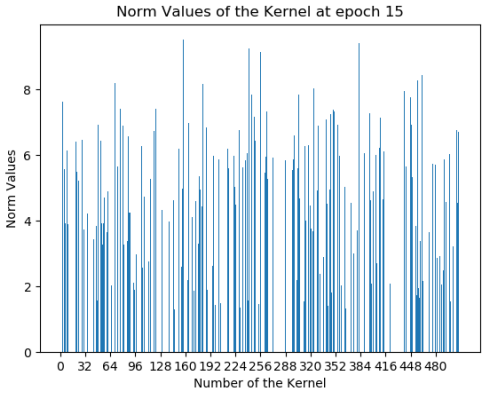


(F) Valeur des noyaux (Abscisse) et le nombre de noyaux possédant cette valeur (Ordonnée) à l'epoch 75.

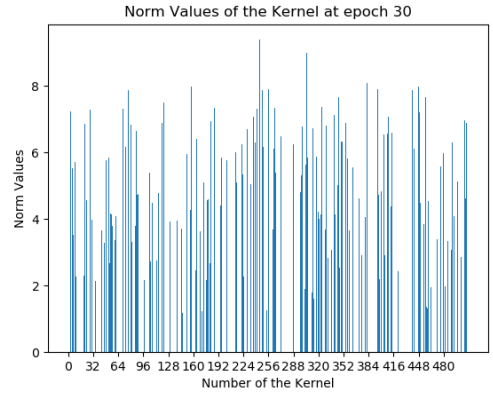
FIGURE 3.16 – Fréquence de la valeur de chacun des noyaux de la deuxième couche de convolution de *VGG11* pendant un apprentissage sur CIFAR-10 avec la norme l_1/l_2 ($\lambda = 0.005$). La couche est composée de 128 noyaux.



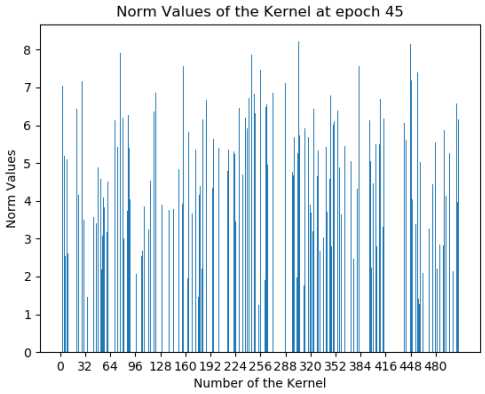
(A) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 0.



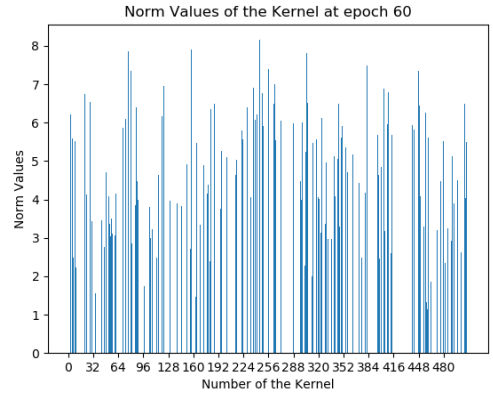
(B) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 15.



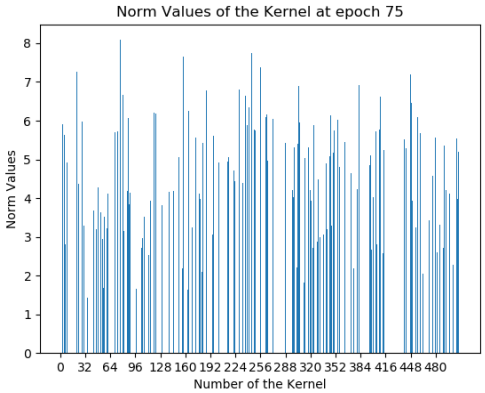
(C) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 30.



(D) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 45.

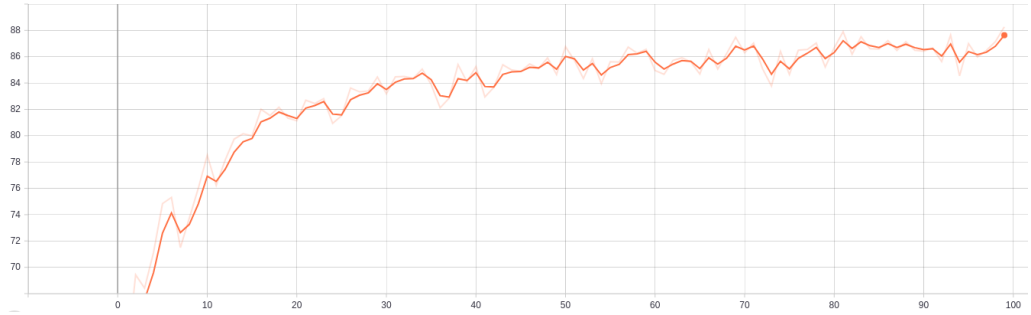


(E) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 60.

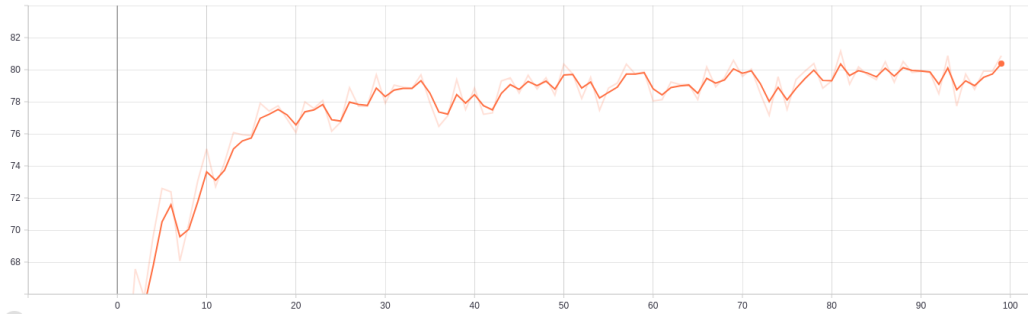


(F) Numéros des noyaux (Abscisse) et leurs valeurs associées (Ordonnée) à l'epoch 75.

FIGURE 3.17 – Évolution de la valeur de chacun des noyaux de la dernière (8ème) couche de convolution de *VGG11* pendant un apprentissage sur CIFAR-10 avec la norme l_1/l_2 ($\lambda = 0.005$). La couche est composée de 512 noyaux.

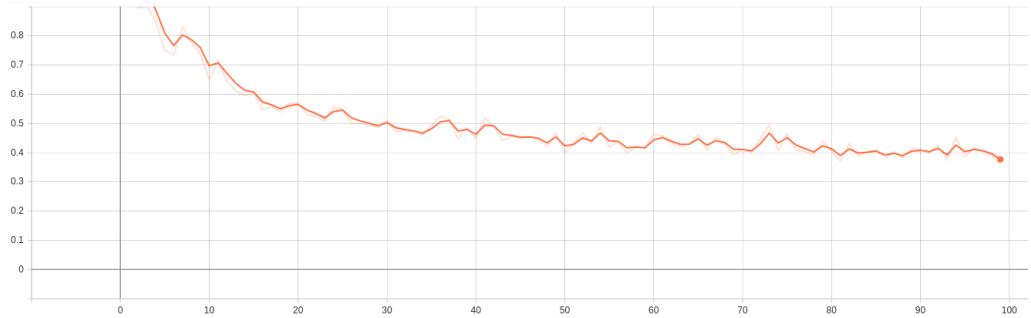


(A) Précision de modèle (Ordonnée) sur la base de données d'entraînement en fonction du nombre d'epoch écoulé (Abscisse).

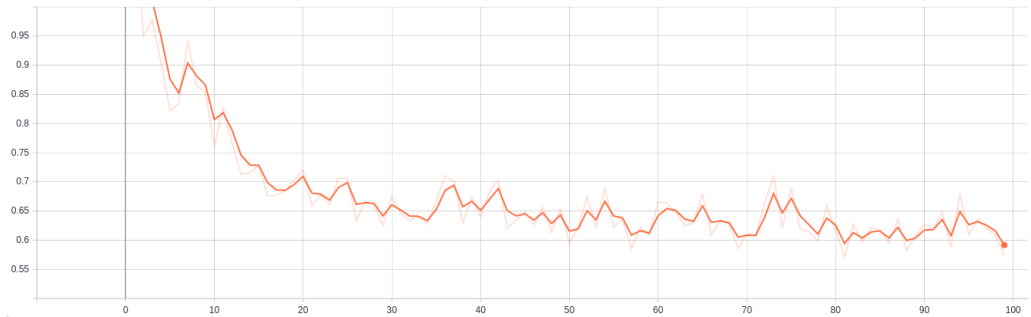


(B) Précision de modèle (Ordonnée) sur la base de données de validation en fonction du nombre d'epoch écoulé (Abscisse).

FIGURE 3.18 – Précisions d'un modèle *VGG11* pendant un apprentissage sur CIFAR-10 avec la norme l_1/l_2 ($\lambda = 0.005$).



(A) Valeur de la fonction de coût du modèle (Ordonnée) sur la base de données d'entraînement en fonction du nombre d'epoch écoulé (Abscisse).



(B) Valeurs de la fonction de coût du modèle (Ordonnée) sur la base de données de validation en fonction du nombre d'epoch écoulé (Abscisse).

FIGURE 3.19 – Valeurs des fonctions de coût du modèle *VGG11* pendant un apprentissage sur CIFAR-10 avec la norme l_1/l_2 ($\lambda = 0.005$).

3.5 Conclusion

Dans ces travaux, nous proposons une nouvelle approche de régularisation afin d'induire une structure plus éparse dans les DCNN basée sur une pseudo-norme l_1/l_2. Cette méthode réorganise les poids des couches de convolution afin d'apprendre des structures plus compacts durant l'apprentissage. Ces DCNN plus compacts peuvent atteindre le même niveau de précision que les modèles originaux et même obtenir de meilleures performances dans certains cas. Nos expérimentations ont démontré les bénéfices de notre approche et de sa généralisation à des structures profondes : elle est simple à implémenter, elle opère directement pendant l'apprentissage et il est aussi possible de choisir une balance entre précision et compression du modèle. Dans cette partie, nous avons appliqué la méthode à des problèmes de classification. Dans la prochaine partie, nous allons tester cette contribution sur des modèles de tailles plus importantes comme Resnet [He et al., 2016] mais aussi sur des plus grands jeux de données et des tâches plus complexes de segmentation et de détection dans le cadre d'une application différente.

Les contributions de ce chapitre ont débouché à des publications à la conférence *RFIAP* [Berthelier et al., 2019] et dans un *Workshop ICPR* [Berthelier et al., 2020b].

Chapitre 4

Spécialisation et compression de réseaux de neurones sur des sous-classes

Le chapitre précédent a montré qu'il était possible d'introduire de la dispersion entre les valeurs des filtres des CNN dans le but de compresser le modèle directement pendant sa phase d'apprentissage. La méthode proposée est efficace et simple à utiliser. Ce chapitre présente une extension de cette méthode dans le cadre d'une autre problématique : celle de la spécialisation sur des sous-classes. Ainsi, ce chapitre montre que la méthode est capable de compresser de manière efficace des DCNN préalablement entraînés sur un jeu de données complet en le ré-entraînant et en le spécialisant sur un ensemble de sous-classes de ce même jeu de données. Nous montrons que cette spécialisation fonctionne également sur des DCNN de tailles plus importantes que dans le chapitre précédent, notamment sur des applications de classification et de segmentation. Nous montrons aussi quelques résultats sur des tâches de détection.

4.1 Introduction

Il est parfois difficile de bien choisir le réseau de neurones à utiliser pour une application donnée. En effet de nombreux modèles ont été développés au fil des années, de complexité variable, et donc, de profondeur variable. Comme nous l'avons vu à travers les précédents chapitres de ce manuscrit, un réseau de neurones possédant une grande profondeur possède plus de paramètres et peut donc potentiellement apprendre des fonctions plus complexes que des réseaux peu profonds [Dauphin and Bengio, 2013]. Cependant, ces réseaux possèdent généralement beaucoup d'informations redondantes et une bonne partie des paramètres de ces réseaux peuvent être retirés sans avoir un impact significatif sur les performances de ces derniers. C'est ce que nous avons accompli dans le chapitre précédent avec la méthode de suppression de noyaux de CNN.

Les modèles d'apprentissage profond sont généralement entraînés sur des bases de données de très grandes tailles avec de nombreux échantillons d'apprentissage et souvent, beaucoup de classes différentes à apprendre. Par exemple, un modèle comme Alexnet possède environ soixante millions de paramètres afin de classer les mille classes du jeu de données ImageNet [Krizhevsky et al., 2012]. Il est aujourd'hui nécessaire d'avoir des modèles d'une grande profondeur afin d'apprendre efficacement des tâches complexes. Les travaux théoriques de Dauphin et al. [Dauphin and Bengio, 2013] ont révélé les difficultés que l'on peut rencontrer à entraîner un réseau peu profond pour obtenir le même niveau de précision qu'un réseau beaucoup plus profond. Ils révèlent néanmoins que cette tâche est théoriquement faisable, mais expérimentalement très difficile.

C'est aussi pour cela que les méthodes de compression post-apprentissage permettent de supprimer une grande partie des paramètres d'un réseau profond : de nombreux paramètres dans ce dernier sont redondants, voir inutiles, allant une fois de plus dans le sens qu'un réseau ayant une profondeur plus limitée pourrait théoriquement effectuer la même tâche avec autant de précision.

La compression de réseaux profonds est une tâche facilement réalisable et cela même pour des modèles devant effectuer des tâches complexes ou devant différencier de très nombreuses classes. Grâce aux nombreux travaux de recherche de ces dernières années, de très nombreux modèles d'apprentissage profond pré-entraînés sur des jeux de données reconnus sont accessibles, soit parce qu'ils ont été mis à disposition par leurs auteurs, soit parce qu'il est possible de les retrouver directement intégrés dans certains frameworks. Lorsque que l'on veut utiliser ces modèles déjà entraînés pour nos applications, il est toujours possible de les compresser en utilisant des méthodes classiques couplées à des phases de ré-entraînement successives sur l'ensemble du jeu de données d'origine par exemple. Néanmoins, il arrive aussi que dans certains cas, nous n'ayons pas besoin de toutes les classes sur lesquelles le jeu de données a été entraîné mais seulement de certaines sous classes de ce jeu de données. Par exemple, imaginons qu'un modèle d'apprentissage a été conçu et entraîné pour apprendre et différencier tous les éléments d'une scène routière (piétons, voitures, camions, motos...) et que nous voulions au final ne détecter qu'une seule de ces classes dans notre application. Ou bien, pour un exemple plus proche de l'objectif initial de cette thèse, que nous ayons un modèle qui a appris à segmenter l'ensemble des parties du visage mais que nous ne sommes intéressés que par une segmentation des yeux ou du nez. C'est ce que nous appelons la spécialisation sur des sous-classes.

Dans ce chapitre, la spécialisation sur des sous-classes de modèles d'apprentissage profond pré-entraînés sur des jeux de données complets est abordée. Ce travail est motivé par plusieurs raisons. La première est celle que nous avons explicitée dans le paragraphe précédent, à savoir que dans la pratique, il est rare d'avoir besoin de l'ensemble des classes sur lesquelles un réseau a déjà été entraîné lorsqu'il est recyclé dans une application d'un autre type. En effet, il est souvent utile de ne se servir que d'une partie des classes, voir même de se spécialiser sur d'autres types de classes, c'est ce qu'on appelle l'apprentissage par transfert [Bozinovski, 2020]. La deuxième raison est que nous pensons qu'un modèle pré-entraîné sur un jeu de données va être davantage compressible lorsqu'il va être spécialisé sur certaines de ses sous-classes (car il aura moins de classes à différencier qu'à l'origine) tout en ayant une bonne précision grâce à ses connaissances acquises lors du premier apprentissage et des apprentissages successifs pour mettre en oeuvre sa spécialisation et sa compression. Ainsi, nous comptons utiliser le modèle de compression présenté dans le chapitre précédent afin de compresser des modèles pendant la phase de ré-entraînement sur des sous-classes pour assurer leur spécialisation.

Cette méthode comporte deux avantages principaux :

1. La spécialisation sur des sous-classes va permettre une précision plus efficace sur ces sous-classes après la spécialisation du modèle.

2. Compresser le modèle de manière efficace car nous supposons qu'il nécessite moins de paramètres pour fonctionner sur un ensemble de sous-classes plutôt que sur le

jeu de données entier.

Les pages suivantes montrent que cette méthode est efficace sur des modèles et des jeux de données connus de classification et de segmentation. Quelques résultats obtenus sur des tâches de détection d'objets sont également présentés.

4.2 Travaux liés

4.2.1 Apprentissage par transfert et distillation des connaissances

Une spécialisation d'un modèle sur certaines des sous-classes d'un jeu de données sur lequel il a déjà été entraîné peut-être vu comme une méthode d'apprentissage par transfert ou de distillation de connaissances. A l'exception que dans ce chapitre, les connaissances déjà engrangées par le modèle vont être utiles pour réduire le même modèle et améliorer son apprentissage sur la même base de données ou les mêmes catégories. En effet, dans la grande majorité des techniques d'apprentissage par transfert, les connaissances d'un modèle d'apprentissage profond sont utilisées sur un jeu de données afin de le rendre performant sur un autre jeu de données [Bozinovski, 2020]. Le modèle reste le même, mais le jeu données diffère. A l'opposé, les techniques de distillation de connaissances sont des méthodes gardant le même jeu de données mais permettant de transférer les connaissances acquises sur ce jeu de données par un modèle profond sur un modèle moins profond [Ba and Caruana, 2014]. Ainsi, le modèle diffère mais le jeu de données reste le même. Dans le cas présenté ici, le modèle est compressé mais ne change pas, et le jeu de données est le même mais seulement quelques sous-classes de ce dernier sont gardées.

Néanmoins de nombreux travaux dans ces domaines se rapprochent de ceux présentés dans ce chapitre. Par exemple, certains travaux couplent les méthodes d'apprentissage par transfert avec des méthodes de compression. Chen et al. [Chen et al., 2017] utilisent une méthode d'apprentissage par transfert en tant que métrique d'apprentissage pour élaguer des réseaux. Huang et al. [Huang and Wang, 2017] utilisent le système de modèles enseignant-élève de la distillation de connaissances comme un système de correspondance de distributions pour comprendre les schémas d'activités de neurones afin de pouvoir en supprimer certains et améliorer les performances globales des modèles. Cependant, bien que ces méthodes fonctionnent très bien pour des tâches simples comme de la classification, elles sont difficiles à mettre en place sur des tâches plus complexes comme de la segmentation, de la détection, etc. Plus de détails sur ces méthodes sont disponibles dans le chapitre 2.2.1 de cette thèse.

4.2.2 Élagage

Les méthodes d'élagage sont les méthodes permettant de supprimer les paramètres jugés comme non essentiels à la propagation de l'information dans un réseau de neurones. C'est une des méthodes de compression les plus répandues et les plus étudiées. Han et al [Han et al., 2016] ont proposé une méthode d'élagage basée sur un seuil d'activation afin de supprimer tous les paramètres sous ce seuil. Plus proche de la méthode que nous avons développé, Li et al. [Li et al., 2017] utilisent une simple norme l_1 pour supprimer des filtres de réseaux convolutifs. Une régularisation basée sur la norme l_1 est aussi utilisée par Liu et al. [Liu et al., 2017] pour effectuer une sélection de cartes de caractéristiques non importantes dans les CNN.

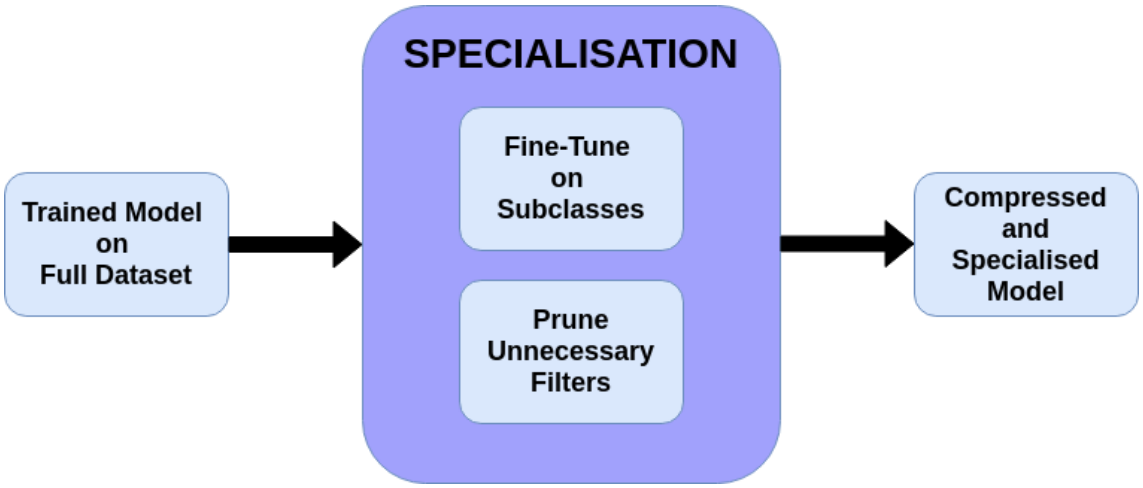


FIGURE 4.1 – Résumé des grandes étapes de la méthode de spécialisation de modèles d'apprentissage profond.

De nombreuses autres méthodes d'élagage ont été développées et continues de l'être. Plusieurs de ces méthodes ont déjà été évoquées dans ce manuscrit dans le chapitre 2.2.3 et le chapitre 3.2.2. Ici, nous reprenons la méthode que nous avons développée dans le chapitre 3.3. Contrairement à de nombreuses méthodes d'élagage standards, la technique que nous avons développée est capable de directement supprimer des filtres de CNN durant l'apprentissage et ne nécessite pas d'étapes de fine-tuning supplémentaires. Bien que la spécialisation sur des sous-classes peut-être vue comme une simple opération de fine-tuning, cette méthode va permettre à ce deuxième apprentissage d'être d'autant plus efficace en supprimant les filtres non-essentiels du modèle simultanément.

4.3 Spécialisation et élagage de modèles

Les grandes étapes de l'approche de spécialisation sont résumées sur la Figure 4.1. On peut la résumer en deux étapes principales : une étape de spécialisation et une étape d'élagage.

4.3.1 Algorithme de spécialisation

La méthode de spécialisation est simple et suit le principe d'algorithmes de fine-tuning classiques, à savoir un ré-entraînement du modèle afin d'améliorer ses performances. En effet les connaissances déjà acquises par un modèle entraîné sur l'ensemble d'un jeu de données sont réutilisées afin d'améliorer ses performances sur un plus petit jeu données qui est un sous-ensemble du jeu de données initiale. On peut donc résumer la méthode de spécialisation en trois étapes :

1. Entraîner un modèle d'apprentissage profond sur un jeu de données complet et complexe.

2. Sélectionner un ensemble de sous-classes de ce jeu de données.

3. Reprendre l'apprentissage du modèle sur l'ensemble de sous-classes en ajoutant le terme de compression dans la fonction de coût.

La dernière étape contient la méthode de suppression de noyaux, permettant de compresser et de supprimer des noyaux du modèle pendant la deuxième phase d'apprentissage sur des sous-classes.

4.3.2 Méthode d'élagage de noyaux

La méthode de suppression de noyaux de CNN est basée sur une pseudo-norme l_1/l_2 permettant d'obtenir un sous-échantillon de noyaux qui auront tous leurs poids égaux à zéro. L'idée de la méthode est d'exprimer le problème de la réduction du nombre de filtres dans un CNN en introduisant de la dispersion sur une sélection de pseudo-normes calculées sur chaque noyau et pas directement sur les valeurs réelles des noyaux. Chaque noyau du modèle va ainsi être réduit à une seule valeur via une pseudo-norme et toutes ces valeurs sont concaténées dans un vecteur global. La dispersion globale des noyaux du modèle est calculée par un ratio de normes l_1/l_2 sur ce vecteur global. Comme un noyau avec tous ses poids à zéro produit une pseudo-norme égale à zéro, le nombre de filtres d'un CNN peut être réduit en renforçant la dispersion des valeurs dans le vecteur global de pseudo-normes. Plus des valeurs seront proches de zéro, plus les filtres associés auront de chances d'être supprimés.

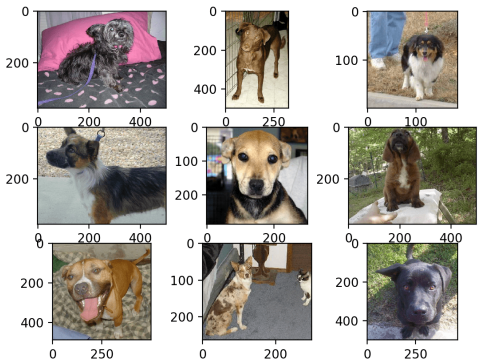
Cette approche présente 3 principaux avantages : (1) toutes les étapes sont faites durant l'apprentissage, il n'y a pas d'opérations supplémentaires requises, (2) la méthode est basée sur de simples normes l_1 et l_2 ce qui la rend simple à implémenter et à exécuter contrairement à d'autres méthodes d'élagage durant l'apprentissage et enfin (3) une trace des apprentissages est gardée en mémoire permettant de suivre l'évolution du modèle à chaque étape. Il est donc possible de choisir le meilleur modèle basé sur un compromis entre compression et performance. Cette méthode d'élagage a été largement développée dans le Chapitre 3 pour plus de détails et a fait l'objet d'une publication dans un *Worshop ICPR* en 2020 [Berthelier et al., 2020b].

4.4 Expériences de spécialisation

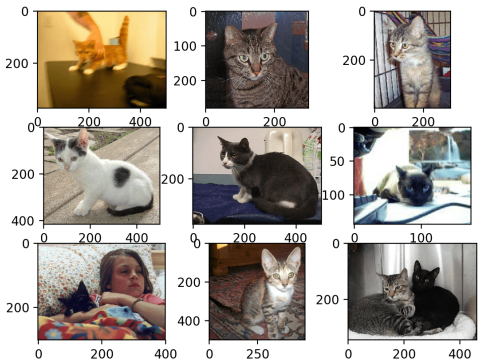
Les performances de la contribution sont évaluées sur trois réseaux différents. Le premier est un réseau AlexNet [Krizhevsky et al., 2012] entraîné sur ImageNet pour des tâches de classification. Le deuxième réseau est un FCN ResNet-50 [Wu et al., 2016] entraîné sur COCO pour de la segmentation d'images. Le dernier réseau est un Mask R-CNN ResNet-50 FPN (Feature Pyramid Network) [Dai et al., 2016] aussi entraîné sur le jeu de données COCO mais pour des tâches de détection d'objets. La méthode a été implémentée en Pytorch et les modèles déjà entraînés sur les bases de données sont aussi issus de ce framework. Les modèles sont entraînés et testés sur divers GPU Nvidia utilisant CUDA. La valeur choisie pour l'hyper-paramètre λ de la méthode est une valeur permettant aux modèles d'obtenir une précision maximale selon nos conditions de test tout en supprimant le plus de filtres possible. Dans toutes les expérimentations, le seuil en-dessous duquel les filtres sont supprimés en évaluant la somme cumulée des plus petites pseudo-normes est réglé à 1%. Pour plus de précision concernant le rôle de ces paramètres, veuillez-vous référer au chapitre 3.

4.4.1 Spécialisation sur des tâches de classification

La première expérimentation de la méthode de spécialisation est réalisée sur une tâche de classification. Pour ce faire, le modèle AlexNet a été choisi [Krizhevsky et al., 2012]. Le modèle a été entraîné sur le jeu de données ImageNet [Deng et al., 2009] datant de 2012. Il a donc été entrainé à reconnaître et à classifier 1000 classes distinctes allant de simples objets à des véhicules, en passant par des animaux, etc. C'est d'ailleurs cette dernière catégorie qui nous intéresse puisqu'elle contient deux classes qui vont être d'intérêt pour nous ici : les chiens et les chats.



(A) Visualisation des 9 premières photos de chiens.



(B) Visualisation des 9 premières photos de chats.

FIGURE 4.2 – Visualisation des 9 premières photos de chiens et de chats du jeu de données Cats vs. Dogs.

En effet, nous décidons de spécialiser le modèle AlexNet sur deux classes, les chiens et les chats. Ces deux classes : le modèle les connaît très bien puisqu'elle font parties des 1000 classes du jeu de données ImageNet. Néanmoins, les images de chiens et de chats du jeu de données ImageNet ne sont pas utilisées afin de spécialiser le modèle sur ces deux classes. C'est le jeu de données Dogs vs. Cats qui est utilisé à la place, possédant des images de chiens et de chats semblables à celle que l'on peut trouver dans ImageNet mais en plus grand nombre (25000 images en tous, 12500 de chaque classe) et plus variées. Une visualisation de ce jeu de données est visible sur la Figure 4.2. L'expérimentation consiste ainsi à spécialiser et compresser AlexNet sur le jeu de données Dogs vs. Cats.

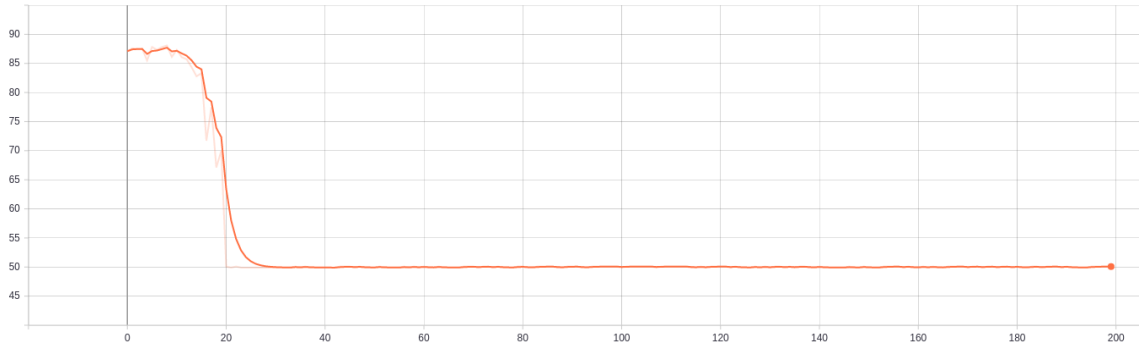
La Table 4.1 résume les résultats obtenus. On peut remarquer que sans la méthode de spécialisation, le modèle AlexNet présente de bons résultats. En effet, uniquement avec le jeu de données ImageNet, sans fine-tuning, le modèle est capable d'obtenir une précision de 87% sur le jeu de données Dogs vs. Cats. Si le modèle est fine-tuné sur le jeu de données Dogs vs. Cats, le modèle est capable d'atteindre une précision de 89%. Si on applique maintenant la méthode de spécialisation, à savoir fine-tuning sur le jeu de données Dogs vs. Cats tout en réduisant le nombre de noyaux du modèle via la méthode basée sur la pseudo-norme l_1/l_2, nous obtenons également de bons résultats. En effet, si nous prenons en compte la meilleure précision que nous ayons été capable d'obtenir, le modèle atteint 88.6% de classifications correct tout en ayant supprimé 13.9% des filtres du réseau. Le modèle spécialisé et compressé a une précision de seulement 0.4% inférieure au modèle fine-tuné sur le jeu de données Dogs vs. Cats tout en étant plus petit,

Model (epoch)	Training Dataset	Fine-Tuned	l_1/l_2	Accuracy	Sparsity (Filter #)
AlexNet	Imagenet	No	No	87%	0% (1152)
AlexNet	Imagenet	Yes	No	89%	0% (1152)
AlexNet (54)	Imagenet	Yes	Yes	88.6%	13.9% (992)
AlexNet (88)	Imagenet	Yes	Yes	87.2%	21% (910)

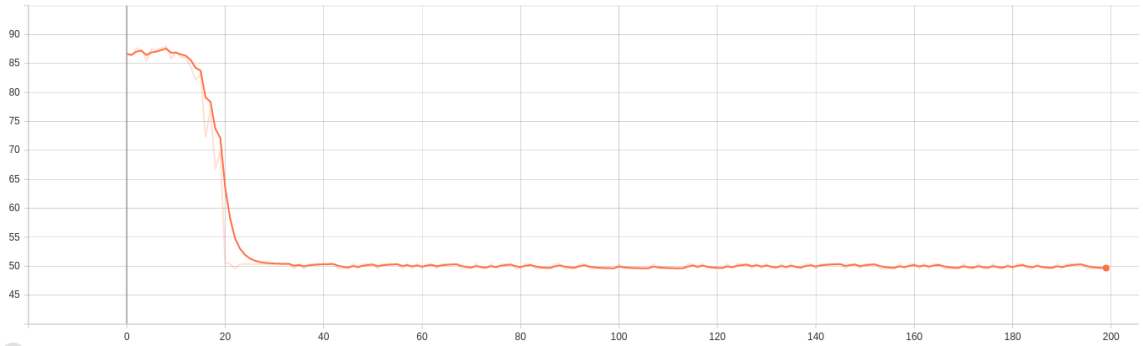
TABLE 4.1 – Résultat de la spécialisation du modèle AlexNet, préalablement entraîné sur le jeu de données ImageNet, sur le jeu de données Dogs vs. Cats issue de Kaggle. La colonne *Fine-Tuned* indique si le jeu de donnée a été re-entraîné (fine-tuné) sur le jeu de données Dogs vs. Cats. La colonne l_1/l_2 indique si le modèle est fine-tuné avec la méthode d'élagage de filtres ($\lambda = 0.1$). Les deux dernières lignes correspondent au même apprentissage mais avec des résultats à des epochs différentes.

ce qui est un bon premier résultat. En supprimant plus de filtres (21% des filtres du réseau) nous sommes capables d'obtenir une précision légèrement supérieure à celle du modèle uniquement entraîné sur ImageNet (87.2% contre 87%). Supprimer plus de filtres fait descendre la précision du modèle en dessous de ce seuil de précision. La méthode de spécialisation est donc capable d'obtenir une précision équivalente au modèle classique ou au modèle fine-tuné tout en ayant moins de filtres en son sein. Néanmoins, le nombre de filtres supprimés reste faible, ne dépassant jamais environ 20% pour une précision satisfaisante et étant même plus proche des 10% en règle générale. Nous pouvons expliquer cela par le fait que classifier ces deux catégories nécessite une grande partie des filtres du réseau, ou tout simplement que le réseau AlexNet n'est pas si profond par rapport à d'autres modèles et qu'il possède moins d'informations redondantes.

Nous avons tracé les données relatives à la précision, la valeur de la fonction de coût et le nombre de filtres supprimés en fonction du nombre d'epoch écoulée du modèle Alexnet lors de sa spécialisation sur le jeu de données Dogs vs. Cats. Ces données sont visibles respectivement sur la Figure 4.3, la Figure 4.4 et la Figure 4.5. Nous pouvons remarquer sur ces figures qu'un grand nombre de filtres est supprimé dans les premières epochs avant de décroitre progressivement dans les epochs suivantes. La précision et la valeur de la fonction de coût du modèle restent hautes et stables dans les premières epochs avant de décroître un peu avant la 20ème epoch. Nous pouvons en déduire que c'est environ à ce moment là que de trop nombreux filtres sont supprimés dans le modèle et que ses performances commencent à décrocher. Il serait possible de diminuer un peu la valeur de λ afin de supprimer un plus petit nombre de filtres dans les premières epochs afin de rendre ce décrochage moins violent.

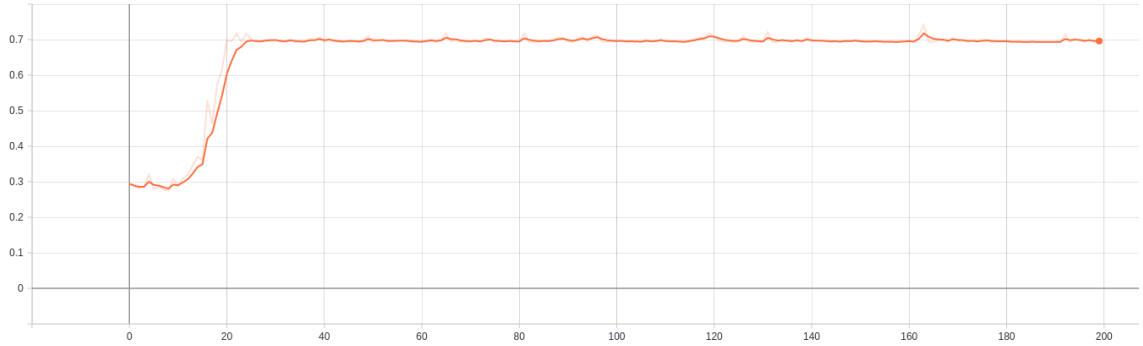


(A) Précision de modèle (Ordonnée) sur la base de données d'entraînement en fonction du nombre d'epoch écoulé (Abscisse).

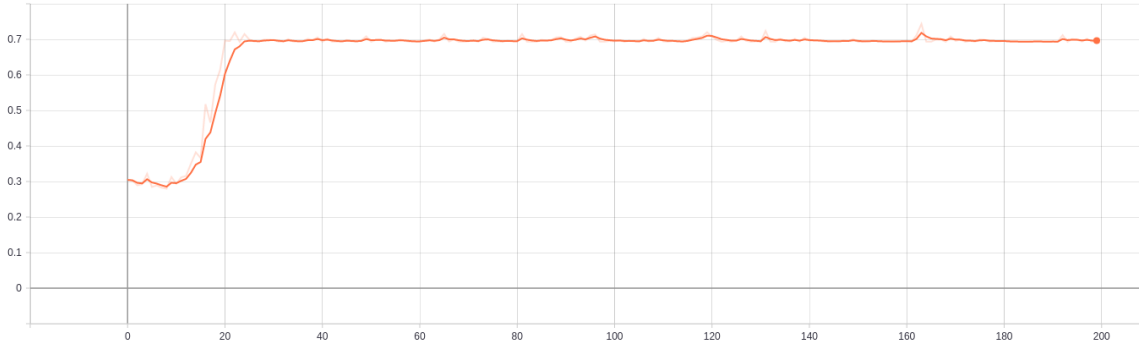


(B) Précision de modèle (Ordonnée) sur la base de données de validation en fonction du nombre d'epoch écoulé (Abscisse).

FIGURE 4.3 – Précisions d'un modèle AlexNet pendant sa spécialisation sur le jeu de données Dogs vs. Cats avec la pseudo-norme l_1/l_2 ($\lambda = 0.1$).

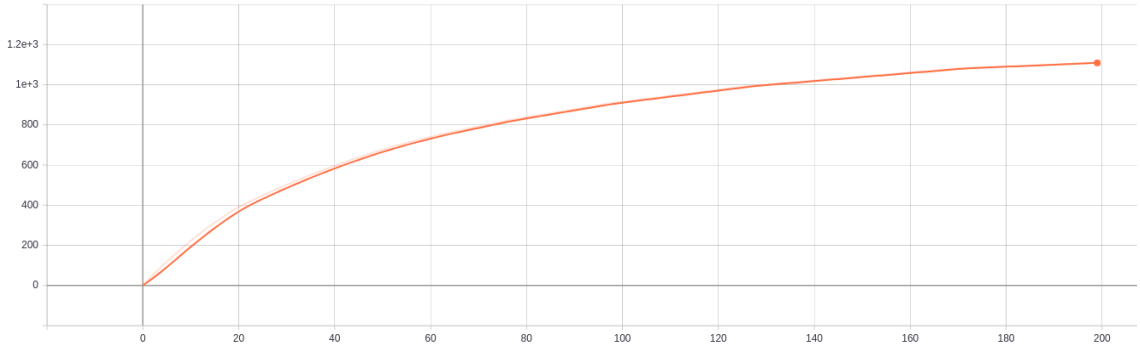


(A) Valeur de la fonction de coût du modèle (Ordonnée) sur la base de données d'entraînement en fonction du nombre d'epoch écoulé (Abscisse).

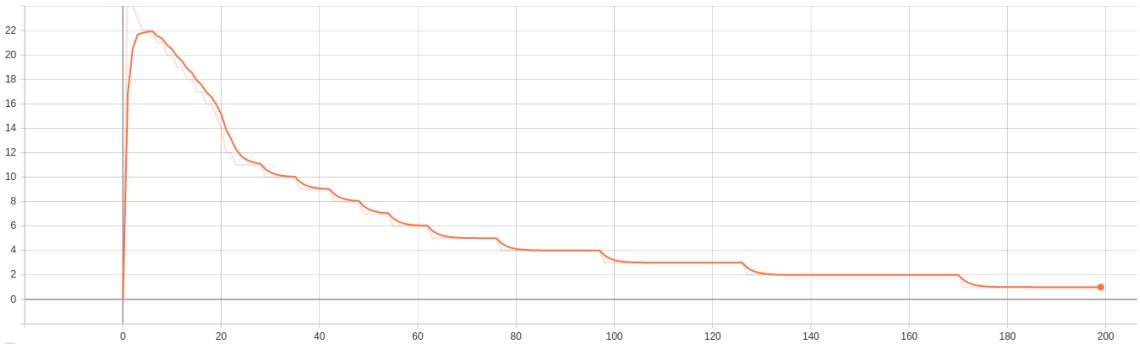


(B) Valeurs de la fonction de coût du modèle (Ordonnée) sur la base de données de validation en fonction du nombre d'epoch écoulé (Abscisse).

FIGURE 4.4 – Valeurs des fonctions de coût du modèle AlexNet pendant sa spécialisation sur le jeu de données Dogs vs. Cats avec la pseudo-norme l_1/l_2 ($\lambda = 0.1$).



(A) Nombre de filtres à zéro (Ordonnées) en fonction du nombre d'epochs écoulé (Abscisse).



(B) Nombre de filtres mis à zéro (Ordonnées) à chaque epoch (Abscisse).

FIGURE 4.5 – Evolution du nombre de filtres du modèle AlexNet en fonction du nombre d'epochs lors de sa spécialisation sur le jeu de données Dogs vs. Cats avec la pseudo-norme l_1/l_2 ($\lambda = 0.1$).

4.4.2 Spécialisation sur des tâches de segmentation

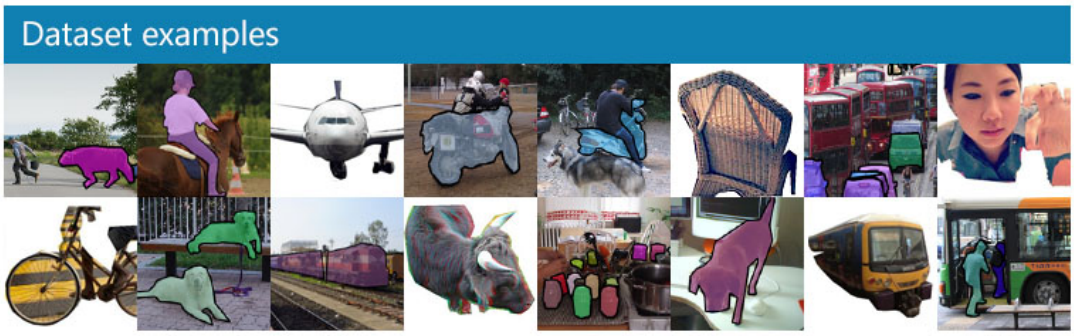


FIGURE 4.6 – Visualisation de quelques exemples d'images de segmentation du jeu de données COCO.

La méthode de spécialisation est maintenant testée sur un CNN plus profond et sur une application plus complexe. Pour ce faire un modèle FCN ResNet50 [Wu et al., 2016] est utilisé afin de réaliser des tâches de segmentation. Le jeu de données change lui aussi afin d'utiliser le jeu de données COCO (plus précisément la version de 2017). Le jeu de données COCO contient près d'une centaine de catégories d'objets qu'il est possible d'utiliser pour entraîner des modèles sur des tâches de segmentation. Dans nos expérimentions, nous n'utilisons pas l'ensemble de ces catégories. En effet, le modèle est déjà pré-entraîné et a appris à segmenter 20 catégories de COCO : aéroplane, vélo, oiseau, bateau, bouteille, bus, voiture, chat, chaise, vache, table, chien, cheval, moto, personne, plante en pot, mouton, sofa, train et télévision. Il est possible de visualiser quelques exemples de ces catégories et leur segmentation associée pour ce jeu de données sur la Figure 4.6. Le modèle est aussi entraîné à reconnaître une catégorie supplémentaire qui est une classe poubelle regroupant tout ce qui n'appartient pas à une des catégories définies du jeu de données. On appelle cette catégorie *background* même si nous n'allons pas la mentionner explicitement ou la compter dans les catégories que nous choisissons, elle est toujours présente et essentielle au bon fonctionnement des tâches de segmentation.

Une fois le modèle FCN ResNet50 entraîné sur ces 20 catégories, deux types de test sont effectués. Le premier test est une spécialisation et compression du modèle sur 10 de ces 20 catégories. Le deuxième test est une spécialisation et compression sur 2 de ces 20 catégories. Les catégories sur lesquelles le modèle est spécialisé sont choisies arbitrairement : peu importe les catégories choisies, les résultats et tendances restent similaires. Dans les tests suivants, les 2 catégories sur lesquelles le modèle a été spécialisé sont les classes bouteille et mouton. La précision des modèles est évaluée via une méthode globale pixel par pixel : un masque binaire représentant la vérité terrain est appliquée sur l'image segmentée prédite par le modèle. Un vrai positif représente un pixel qui est correctement attribué à la classe voulue et un vrai négatif correspond à un pixel qui est correctement identifié comme n'appartenant pas à la classe voulue. Bien que n'étant pas la meilleure méthode d'évaluation de tâches de segmentation (on préfère usuellement les méthodes Intersection over Union ou mean Average Precision), elle a l'avantage d'être simple et de permettre d'avoir une idée de la précision atteinte par une méthode par rapport à une autre.

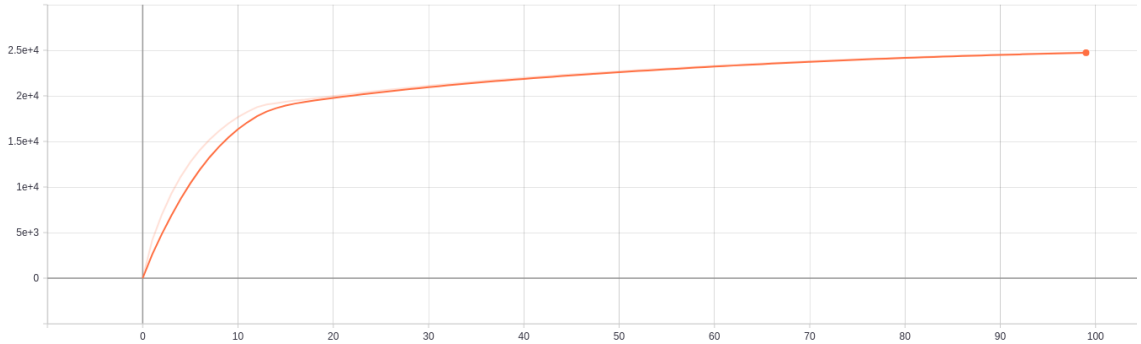
Le résultat des expérimentations est visible dans la Table 4.2. On remarque que le modèle simple, entraîné sur les 20 catégories de COCO atteint une précision de 91.4%. Si le modèle est fine-tuné sur 2 catégories (sans utiliser la méthode de spécialisation), la

Model (epoch)	Training Dataset	Fine-Tuned	l_1/l_2	Accuracy	Sparsity (Filter #)
FCN ResNet50	COCO (20)	No	No	91.4%	0% (26560)
FCN ResNet50	COCO (20)	COCO (2)	No	98.5%	0% (26560)
FCN ResNet50	COCO (20)	COCO (2)	Yes	99.5%	26.2% (19606)
FCN ResNet50 (3)	COCO (20)	COCO (10)	Yes	96.8%	25.5% (19793)
FCN ResNet50 (5)	COCO (20)	COCO (10)	Yes	95.9%	39.3% (16113)

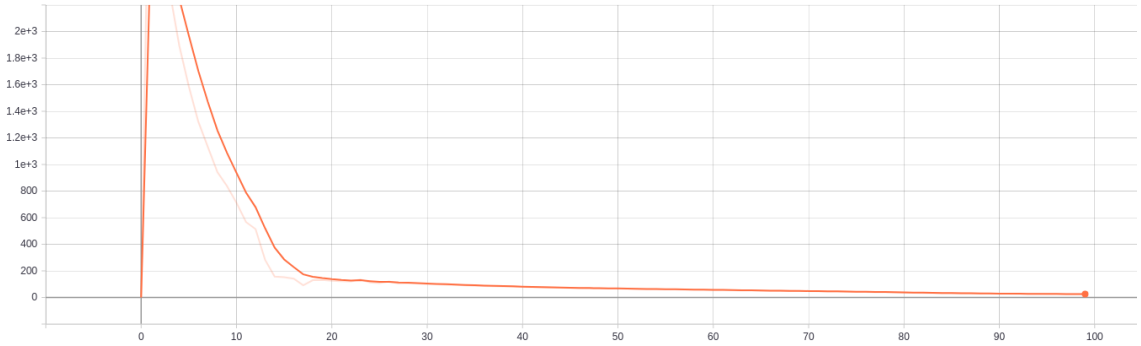
TABLE 4.2 – Résultat de la spécialisation du modèle FCN ResNet50, préalablement entraîné sur 20 Catégories du jeu de données COCO, puis spécialisé sur 2 ou 10 de ces catégories. La colonne *Fine-Tuned* indique si le jeu de donnée a été re-entraîné (fine-tuné) et sur combien de catégories. La colonne l_1/l_2 indique si le modèle est fine-tuné avec la méthode d'élagage de filtres ($\lambda = 0.5$). La précision est évaluée via une méthode méthode pixel par pixel. Les deux dernières lignes correspondent au même apprentissage mais avec des résultats à des epochs différentes.

précision du modèle augmente, lui permettant d'atteindre une précision de 98.5%. Si le modèle est spécialisé sur 2 catégories avec la méthode de spécialisation permettant de réduire le nombre de filtres du réseau, la précision augmente encore tout en permettant de supprimer environ 26% des filtres du modèle. Ces hauts niveaux de précision sont à prendre avec du recul, nous n'utilisons pas la meilleure méthode d'évaluation qui existe pour des tâches de segmentation. Cependant, ils permettent de dire que la méthode de spécialisation performe aussi bien si ce n'est mieux qu'un fine-tuning classique tout en supprimant plus d'un quart des filtres du modèle. Le modèle étant très profond, cela peut indiquer qu'il l'est trop pour bien catégoriser seulement 2 catégories, et qu'il existe beaucoup d'informations redondantes dans le modèle. En passant à une spécialisation sur 10 catégories au lieu de 2, soit la moitié de la taille du jeu de données sur lequel le modèle a été entraîné à la base, les résultats restent aussi encourageant. En effet le modèle est capable d'atteindre une précision de 96.8% tout en supprimant plus de 25% des filtres du modèle, et peut même atteindre la suppression de 39% à 40% des filtres du modèle tout en maintenant une précision de 95.9%. Il est donc possible de garder une précision élevée et de supprimer une bonne partie des filtres du réseau tout en gardant un bon nombre des différentes catégories initiales à segmenter. On remarque cependant que plus le nombre de catégories gardées est élevé, plus la précision diminue. Cela parait logique, il est plus facile de différencier 2 classes plutôt que 10, pour un humain comme pour un modèle.

Un dernier résultat intéressant que l'on peut voir sur la Figure 4.7 est que la grande majorité des filtres sont supprimés dans les premières itérations (epochs) d'apprentissage du modèle indiquant que beaucoup de filtres sont jugés comme non important très tôt dans l'apprentissage et sont supprimés sans détériorer la précision du modèle. Ce phénomène peut-être diminué en baissant la valeur du coefficient de régularisation λ de la méthode de spécialisation. Nous avons décidé de le laisser dans ces tests prendre une valeur haute ($\lambda = 0.5$), la suppression d'un grand nombre de filtres dès les premières étapes de l'apprentissage ne heurtant pas les performances globales du modèle.



(A) Nombre de filtres mis à zéro (Ordonnées) en fonction du nombre d'epochs écoulées (Abscisse).



(B) Nombre de filtres mis à zéro (Ordonnées) à chaque epoch (Abscisse).

FIGURE 4.7 – Evolution du nombre de filtres du modèle FCN ResNet50 en fonction du nombre d'epochs lors de sa spécialisation sur 2 classes du jeu de données COCO avec la pseudo-norme l_1/l_2 ($\lambda = 0.5$).

4.4.3 Premiers tests sur des tâches de détection

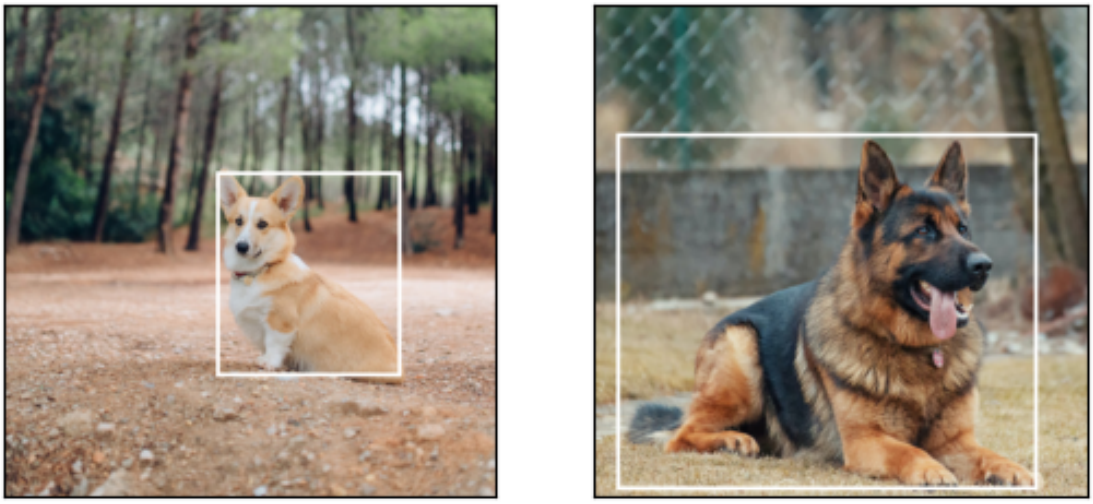


FIGURE 4.8 – Visualisation de quelques exemples d'images de détection du jeu de données COCO (avec la boite englobante).

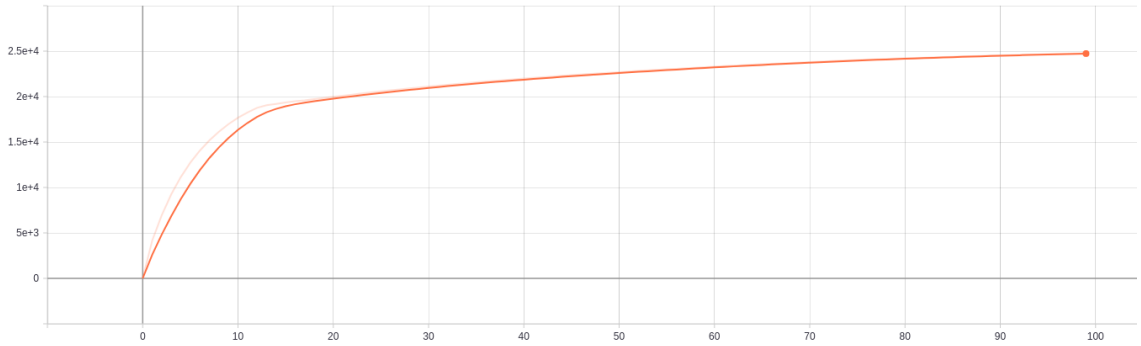
Afin de finir ces expérimentations sur une application et un CNN plus complexe (bien que basé sur l'architecture du ResNet50), un mask R-CNN ResNet-50 FPN [Dai et al., 2016] est utilisé afin de réaliser des tâches de détection. Le jeu de données reste le même que pour les tests de segmentation, à savoir le jeu de données COCO. Dans cette expérimentation, le modèle à été entraîné sur 91 catégories du jeu de données et à donc appris à détecter autant d'objets différents. Il est possible de visualiser un exemple d'images disponibles dans le jeu de données, ainsi que les boites englobantes associées à la tâche de détection dans la Figure 4.8.

Les premiers résultats obtenus sont visibles dans la Table 4.3. La méthode choisie pour évaluer la précision du modèle est une méthode de précision moyenne de la boite englobante (Bounding Box Average Precision). On remarque que le modèle de base, sans la pseudo-norme l_1/l_2 et sur l'ensemble du jeu de données, atteint une précision de 37%. En ajoutant la méthode basée sur la pseudo-norme l_1/l_2, toujours sur l'ensemble du jeu de données, le modèle atteint au mieux une précision de 32% tout en supprimant 16.4% des filtres du réseau (l'évolution du nombre de filtres supprimés au cours de l'apprentissage est visible sur la Figure 4.9). Ainsi, bien que quelques filtres soient supprimés, la précision reste en dessous de celle du modèle de base. De plus, ce résultat est obtenu dès la première itération de l'apprentissage, la précision baissant drastiquement par la suite. On peut aussi remarquer l'absence de tests de spécialisation pour ce modèle alors que c'est l'objet du chapitre. Cela est dû au fait que nous n'ayons pas réussi à faire converger un modèle spécialisé avec la méthode de spécialisation. Aucun des modèles n'a atteint une précision suffisante. Cela peut être dû à plusieurs facteurs. Une explication possible est que le modèle est bien plus complexe que les précédents que nous avons utilisé. En effet, le modèle étant un Mask R-CNN [He et al., 2017a], il possède plusieurs branches : une pour la classification et une pour la régression de la boite englobante. Il est donc possible que la méthode ait des difficultés à fonctionner parfaitement sur ce type de modèle. Une autre explication possible peut venir du paramétrage de la méthode. Certaines couches doivent peut-être être laissées intactes afin de garantir une bonne performance du modèle et les paramètres ont peut-être des valeurs trop hautes, supprimant trop de filtres trop vite. Une solution serait donc de réaliser des tests complémentaires, chose que

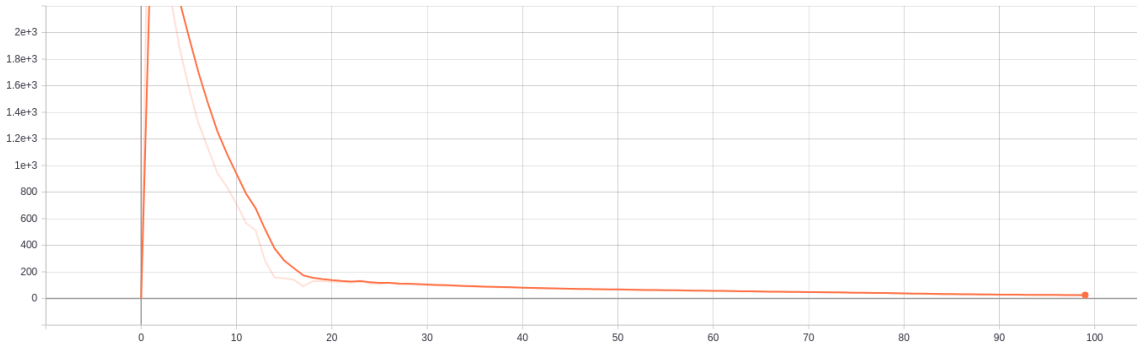
Training Dataset	l_1/l_2	Accuracy	Sparsity (Filter #)
COCO (91)	No	37%	0% (26560)
COCO (91)	Yes	32%	16.4% (22210)

TABLE 4.3 – Résultat des premiers travaux le modèle de détection Mask R-CNN ResNet-50 FPN entrainés sur 91 classes du jeu de données COCO. La colonne l_1/l_2 indique si le modèle est fine-tuné avec la méthode d'élagage de filtres ($\lambda = 0.1$). La précision est évaluée via une méthode de précision moyenne de la boite englobante (Bounding Box Average Precision).

nous n'avons pas eu le temps d'effectuer au moment d'écrire ces lignes, le temps d'apprentissage de ce modèle étant relativement long. Une autre solution serait de changer de modèle de détection pour un modèle de détection avec une architecture plus simple, comme le modèle YOLO [Redmon et al., 2015] ou une de ses nombreuses évolutions. Les résultats de modèles de détection assistés par la méthode de spécialisation ont matière à être améliorés dans le futur.



(A) Nombre de filtres mis à zéro (Ordonnées) en fonction du nombre d'epochs écoulées (Abscisse).



(B) Nombre de filtres mis à zéro (Ordonnées) à chaque epoch (Abscisse).

FIGURE 4.9 – Evolution du nombre de filtres du modèle mask R-CNN ResNet-50 FPN en fine-tunant le modèle avec la pseudo-norme l_1/l_2 sur le jeu de données COCO ($\lambda = 0.1$).

4.5 Conclusion

Dans ce chapitre, une approche de spécialisation sur des sous-classes de modèles d'apprentissage profond est proposée. Cette approche permet non seulement de spécialiser des DCNN sur un petit ensemble de catégories du jeu de données initiale mais aussi de compresser et de supprimer des filtres du modèle via la méthode détaillée dans le chapitre 3. L'approche est efficace pour des tâches de classification et de segmentation utilisant des modèles profonds, arrivant dans certains cas à améliorer la précision du modèle de base sur certaines sous-catégories tout en obtenant un modèle plus compressé.

Cependant ces premiers résultats peuvent encore être améliorés en multipliant les tests et en choisissant d'autres méthodes d'évaluation. Bien que la méthode soit assez spécifique et difficile à comparer avec d'autres méthodes de l'état de l'art, il serait aussi intéressant de la comparer à des méthodes d'élagage ou d'apprentissage par transfert ayant des objectifs similaires. Enfin, et comme nous l'avons vu pour le modèle de détection d'objets, les premiers tests réalisés ne sont pas satisfaisants. Il serait donc intéressant d'étendre les recherches dans ce domaine.

Chapitre 5

Analyse de visages pour des applications mobiles en réalité augmentée

Dans ce chapitre, nous présentons une démonstration pour montrer la faisabilité et les performances de l'analyse de visages basée sur des méthodes d'apprentissage profond et en réalité augmentée. La démonstration fonctionne sur un iPhone X en temps réel et fournit des résultats consistants entre chaque image.

5.1 Introduction

Détecter différents composants du visage est une tâche de grande importance pour de nombreuses applications en réalité augmentée comme l'embellissement d'images de visages ou l'édition d'images de visages. Par exemple, en détectant la zone des lèvres, il est possible d'appliquer un effet de rouge à lèvres virtuel par dessus en coloriant la région avec des couleurs adéquates. Dans notre cas, nous nous concentrons sur la construction de méthodes efficaces pour de l'analyse de visages en utilisant des réseaux de neurones étant donné que beaucoup de ces applications ont pour cible les plateformes mobiles comme iOS et Android.

Les CNN se sont montrés être des méthodes de pointe pour de nombreuses tâches de vision par ordinateur, et notamment pour la segmentation sémantique [Badrinarayanan et al., 2017, Ronneberger et al., 2015]. Néanmoins, ces méthodes sont soit coûteuses en terme de temps soit en terme de mémoire à cause des nombreux paramètres et d'étapes de calcul qu'elles contiennent. La plupart des méthodes de segmentation sémantique sont conçues pour des scènes généralement complexes comme de l'analyse de rues. Cependant, l'analyse de visages est une tâche assez différente pour les raisons suivantes.

— L'analyse de visages est généralement effectuée sur une région d'intérêt déterminée par une détection du visage en amont. La segmentation sémantique est généralement effectuée sur l'image entière.

— Des frontières nettes sont demandées pour les applications de visages en réalité augmenté afin d'avoir un meilleur rendu des effets visuels.

— Les composants faciaux ont une variance de déformation plus grande mais une variance de position et de taille plus basse en comparaison avec la segmentation sémantique d'objets classiques.

FIGURE 5.1 – Résultats visuels de notre méthode sur iPhone. Notre méthode est efficace même sur des expressions et poses extrêmes du visage.

Dans ce chapitre, une démonstration d'analyse de visages est présentée pour des applications mobiles en réalité augmentée et en temps réel sur iPhone avec un CNN profond efficace. Contrairement aux méthodes d'analyse de visages précédentes, l'adaptation de réseaux de neurones sur des vidéos est prise en compte afin de produire un rendu fluide et cohérent entre chaque image. Les utilisateurs sont capables de visualiser les résultats de segmentation via un masque qui indique différentes régions du visage. Un résultat de rendu est montré sur la Figure 5.1. Différentes applications en réalité augmentée peuvent être réalisées en se basant sur les résultats obtenus.

5.2 Travaux liés à la segmentation sémantique

Un consensus communément partagé dans le domaine de la segmentation sémantique via des méthodes d'apprentissage profond est qu'il existe deux sortes de méthodes dominantes [Chen et al., 2018] : les structures basées sur le module *Spatial Pyramid Pooling* (SPP) et les structures de type encodeur-décodeur [Badrinarayanan et al., 2017, Ronneberger et al., 2015]. Les structures de type encodeur-décodeur adoptent une méthode de sur-échantillonnage progressif avec des sauts de connexions afin de reconstruire les bordures des objets de la manière la plus précise possible. Les sauts de connexions jouent un rôle important dans la structure du réseau de manière à ce que le CNN puisse transférer les informations détaillées de bas niveau vers les couches de sortie.

5.3 Travaux liés à l'accélération de réseaux

Afin d'être sûr d'obtenir la meilleure expérience en réalité augmentée possible pour l'utilisateur, un temps d'inférence court et une très faible latence sont les deux principaux pré-requis. Certains travaux ont proposé d'utiliser le flux optique et de ré-utiliser les cartes de caractéristiques des images précédentes si la scène globale persistait. Une autre manière d'accélérer le temps d'inférence est d'utiliser des méthodes d'accélération et de compression de réseaux de neurones comme de la quantification ou de l'élagage. Des travaux récents comme MobileNet [Sandler et al., 2018b] et Shuffle-net [Zhang et al., 2017] ont proposé d'utiliser une convolution séparable en profondeur afin de réduire la complexité de calcul de CNN tout en gardant le même niveau de performance.

5.4 Description de la démo d'analyse de visages pour mobile

Dans cette partie, la structure du réseau de segmentation est présentée, ainsi que son adaptation à la vidéo avec quelques détails d'implémentation.

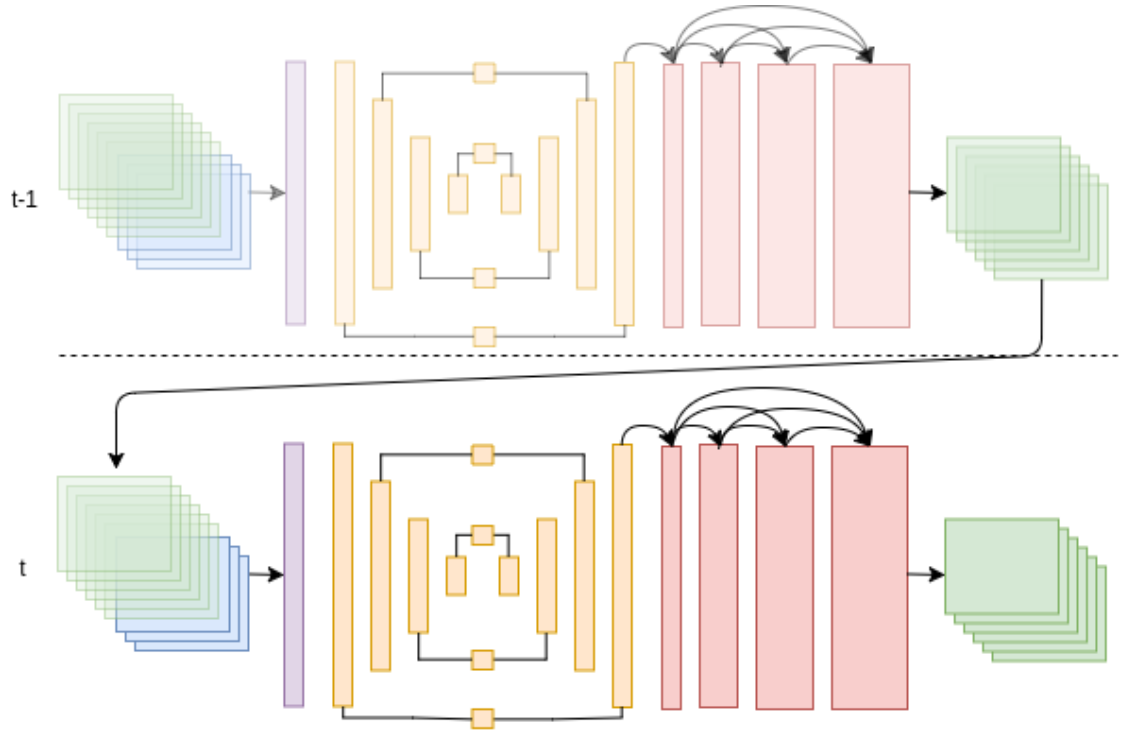


FIGURE 5.2 – Aperçu de la méthode pour l'analyse de visages vidéo. Canaux bleus : images du visage en RGB, Canaux verts : prédictions des masques. Plus de transparence signifie des prédictions tôt dans le temps. Les prédictions en sortie à $t-1$ sont réinjectées au temps t pour plus de robustesse. Blocs pourpres : couche de convolution + couche de batch norm. Blocs jaunes : le modèle Hourglass composé de blocs Mobile-net. Blocs rouges : blocs dense [Huang et al., 2017] Blocks.

5.4.1 Réseau Hourglass pour mobile

L'architecture développée suit celle du modèle Hourglass pour l'estimation de la pose humaine [Newell et al., 2016]. La structure du réseau est présentée sur la Figure 5.2. Le modèle Hourglass est un réseau encodeur-décodeur symétrique entièrement convolutif avec une profondeur de 4, ce qui veut dire que l'encodeur sous-échantillonne 4 fois et que le décodeur sur-échantillonne la carte de caractéristiques 4 fois afin de reconstruire la sortie. Chaque bloc jaune représente un bloc du réseau qui permet au CNN d'apprendre le flux d'information à chaque étape et à chaque saut de connexions.

Plusieurs couches de convolution et de max-pooling sont laissées de coté au début du réseau et la taille de la carte de sortie est augmentée à 256, ce qui est la même taille que l'image en entrée. Cela va avoir pour effet de rendre les bordures des composants faciaux plus nettes mais va aussi augmenter la complexité des calculs en parallèle. Afin d'accélérer l'inférence, tous les blocs ResNet dans le réseau Hourglass sont remplacés par des blocs bottleneck inversés provenant du réseau MobileNet [Sandler et al., 2018b]. Le facteur d'expansion τ est un hyper paramètre important qui indique combien de fois le nombre de canaux de caractéristiques est étendu dans son propre bloc. Nous fixons le nombre de caractéristiques f à 32 et le facteur d'expansion τ à 6 en trouvant le meilleur compromis entre la vitesse et la qualité de la segmentation. Une comparaison des modèles utilisant différents f et τ est visible dans la Table 5.1.

Model	Overall F-score	Num. of parameters
SegNet [Badrinarayanan et al., 2017]	92.90	29.45M
Unet [Ronneberger et al., 2015]	93.72	13.40M
Mobile-Hourglass-f16-τ3	93.00	0.09M
Mobile-Hourglass-f16-τ6	93.08	0.12M
Mobile-Hourglass-f32-τ3	93.18	0.17M
Mobile-Hourglass-f32-τ6	93.55	0.27M

TABLE 5.1 – Évaluation quantitative des résultats d'analyse de visages sur le jeu de données Helen.

5.4.2 Adaptation à la vidéo

En s'inspirant de ce poste de blog [Bazarevsky and Tkachenka, 2018], deux stratégies on été implémentées afin d'adapter notre modèle à de l'analyse de visages vidéo.

Pour l'inférence, le masque à l'image $t-1$ est pris comme entrée pour l'image t afin de stabiliser la segmentation. A cause de l'absence de jeux de données vidéo, un masque transformé est appliqué de manière aléatoire en tant que faux masque précédent pour l'apprentissage. Selon nos expérimentations, cela va éliminer le bruit de segmentation aléatoire qui est présent sans prendre le masque de l'image précédente en entrée.

Quatre couches denses [Huang et al., 2017] sont ajoutées avec un taux de croissance de 8 à la fin du réseau pour un rendu plus robuste.

5.5 Expérimentations

Le modèle est entraîné sur le jeu de données Helen [Smith et al., 2013] qui contient 2330 images manuellement étiquetées en 10 classes incluant peau, cheveux, nez, sourcil gauche/droit, oeil gauche/droit, lèvre supérieur/inférieur et l'intérieur de la bouche. Les modèles sont entraînés sur toutes les étiquettes à l'exception des cheveux car l'annotation des cheveux n'est pas précise. Les images sont découpées avec une marge de 30% à 70% de la taille de la boite englobante en fonction des annotations des repères faciaux.

RMSprop est utilisé comme algorithme d'optimisation et la fonction d'entropie croisée softmax comme fonction de coût. Un taux d'apprentissage de 0.0005 est appliqué avec une décroissance de 0.1 toutes les 40 epochs durant l'apprentissage jusqu'à ce que le total de 190 epochs soit fini. ONNX est utilisé comme format intermédiaire afin de transformer notre modèle Pytorch en modèle CoreML, qui est optimisé pour appareils iOS. Le framework Vision est également adopté pour l'étape de détection du visage qui est l'étape antérieure à l'analyse du visage.

Les méthodes que nous avons développées sont comparées avec plusieurs réseaux de segmentation de l'état de l'art [Ronneberger et al., 2015, Badrinarayanan et al., 2017]. Les résultats sont mesurés avec le F1-score dans la Table 5.1. Le nombre de paramètres est aussi listé afin de fournir plus d'information à propos de la taille du modèle, information qui est critique pour des plateformes mobiles.

La durée d'exécution des blocs de MobileNet est mesurée avec différents paramètres en changeant le nombre de canaux f et la valeur du facteur d'expansion τ. Une analyse

Model	Face Detection	Inference	Colorization	Total
Unet [Ronneberger et al., 2015]	7	203	54	269
Mobile-Hourglass-f16-τ3	8	77	18	106
Mobile-Hourglass-f16-τ6	7	75	18	103
Mobile-Hourglass-f32-τ3	7	76	18	104
Mobile-Hourglass-f32-τ6	7	78	18	106
Dense Mobile-Hourglass-f32-τ6	7	75	18	110

TABLE 5.2 – Durée d'exécution (en ms) sur un iPhone X.

du temps qu'il faut pour effectuer la détection du visage est également fournie, ainsi que du temps nécessaire pour le calcul de la matrice de transformation et la colorisation dans la Table 5.2. Les résultats montrent que l'analyse de visages est capable de fonctionner sur des téléphones mobiles en temps réel.

5.6 Conclusion

Dans ce chapitre, une démonstration en temps réel d'un encodeur-décodeur vidéo pour de l'analyse de visages en réalité augmentée est présentée. En utilisant une architecture neuronale spécifique qui prends en compte l'estimation des précédents pas de temps et en augmentant l'efficacité des calculs avec des convolutions factorisées en profondeurs issues du réseau MobileNet, nous avons montré qu'il est possible de réaliser des applications d'analyse de visages en temps réel pour mobile. De plus :

— L'analyse de visages est cruciale et réalisable pour de nombreuses applications de modification du visage comme le maquillage virtuel, la coloration de cheveux, l'analyse de peau, la transformation et la reconstruction du visage etc.

— La méthode utilisée n'est pas seulement limitée à des applications en réalité augmentée pour le visage mais est aussi intéressante pour d'autres applications en réalité augmentée basées sur de la segmentation fine, comme l'extraction du premier plan ou du fond d'une image.

Les contributions de ce chapitre ont débouché sur la publication d'un papier de démonstration à *ISMAR* [Yan et al., 2018].

Chapitre 6

Conclusion

6.0.1 Résumé des contributions

Dans cette thèse, nous avons exploré, développé et contribué autour de différents aspects sur le thème de la compression de réseaux de neurones. Diverses méthodes ont été présentées afin de permettre à des réseaux de neurones d'être plus efficaces, aussi bien en terme de consommation mémoire qu'en terme de temps de calcul, permettant ainsi de faciliter l'incorporation de ce genre de modèle dans des systèmes embarqués comme des smartphones par exemple. Nous avons proposé une méthode de compression permettant de mieux comprendre le rôle de certains filtres de CNN tout en optimisant leur nombre directement pendant l'apprentissage. Un résumé des contributions de chacun des chapitres de ce manuscrit est présenté ci-dessous.

Architectures optimisées et Outils de framework : dans le Chapitre 2, un état de l'art complet a été effectué concernant l'optimisation de réseaux de neurones profonds. La contribution se situe principalement sur le fait d'avoir mélanger un très large ensemble de méthodes différentes dans cet état de l'art. En effet nous avons effectué d'une part un passage en revue de nombreuses méthodes de compression de modèles d'apprentissage profond ainsi que de leurs inconvénients et de leurs avantages les unes par rapport aux autres. D'autres part, un état de l'art complet a été aussi effectué sur les méthodes relatives à la construction d'architectures optimisées, allant de l'explication et de l'intégration de certains modules spécifiques à la revue de méthodes et d'algorithmes permettant la construction automatique de modèles d'apprentissage profond optimisés pour certaines tâches spécifiques. L'ensemble de ces méthodes est utilisable pour compresser ou optimiser des réseaux de neurones afin de faciliter leur portage vers des systèmes embarqués ou des plateformes mobiles de type iOS ou Android par exemple.

Le Chapitre 5 a montré ce dont était capable l'alliance entre des outils de frameworks performants, une architecture optimisée et une tâche de segmentation de visages en proposant une démonstration d'analyse de visages pour des applications mobiles en réalité augmentée et en temps réel. En effet, en reprenant l'architecture performante du modèle Hourglass [Newell et al., 2016] et en la couplant avec l'efficacité du flux de données du modèle MobileNet [Sandler et al., 2018a], nous avons pu construire un réseau optimisé pour une application en réalité augmentée basée sur l'analyse de visages. L'utilisation de frameworks comme Pytorch, ONNX et CoreML a permis un portage et un niveau de compression optimisé pour son passage sur mobile. Enfin, l'incorporation de couches denses dans notre architecture et la ré-injection de l'image précédente pour l'évaluation de chaque image a permis de maintenir la cohérence nécessaire entre chaque pas de temps pour une application en temps-réel.

Méthode de compression : dans le chapitre 3, une nouvelle approche pour la compression de CNN a été proposée. Cette approche est construite autour d'une régularisation permettant d'induire une structure éparse entre les filtres de CNN en se basant sur une pseudo-norme l_1/l_2. La régularisation permet une réorganisation des poids du modèle, permettant de supprimer des filtres jugés comme non importants par la méthode. Contrairement à d'autres approches similaires, toutes les étapes de cette méthode sont effectuées pendant l'apprentissage initial, elle est basée sur des normes simples l_1 et l_2 qui sont faciles à implémenter et à calculer, et enfin, comme l'évolution de l'architecture à chaque étape de l'apprentissage est gardée en mémoire, il est possible de choisir le meilleur modèle selon un compromis entre compression et précision. Cette méthode est capable de performances similaires, voir supérieures à d'autres méthodes de l'état de l'art sur des jeux de données classiques de classification.

Dans le Chapitre 4, la méthode développée dans le Chapitre 3 est étendue à des CNN plus profonds et des tâches plus complexes de classification, segmentation et détection dans le but de spécialiser des modèles sur des sous-classes de jeux de données sur lesquels ils ont déjà été entraînés. Nous montrons que ré-entraîner des modèles sur certaines de ses sous-classes permet d'avoir de meilleurs résultats sur ces classes choisies et que l'ajout de notre méthode permet d'avoir des performances au minimum similaires aux modèles d'origines tout en supprimant une grande partie des filtres.

6.0.2 Futurs travaux

Dans cette section, nous présentons quelques directions que peuvent prendre les travaux présentés dans cette thèse dans le futur.

Application en temps réel : nous avons développé une application en temps réel sur mobile pour de l'analyse de visages. Une piste pour aller plus loin avec ce projet serait de développer de vraies applications pour le domaine de l'esthétique avec cette méthode, comme du maquillage virtuel par exemple, afin d'évaluer les performances du modèle développé sur une application concrète. Une autre idée serait de tester des méthodes de compression sur ce modèle déjà optimisé pour plateformes mobiles afin d'évaluer leurs effets sur ces performances. On peut aussi évoquer le sujet de la performance de ces modèles sur des applications vidéos en réalité augmentée. La question de savoir comment exploiter de manière efficace les informations temporelles pour la détection de points du visage et la segmentation de cheveux est de première importance pour les futures recherches autour de ce sujet.

Méthodes de compression : concernant la méthode présentée dans cette thèse au Chapitre 3 et étendue à une autre problématique dans le Chapitre 4, elle peut être encore affinée. De plus nombreux tests sur des réseaux profonds sont possibles afin d'affiner et de valider les tendances obtenues sur plus de modèles. Nous n'avons pas intégré sur des applications mobiles ou des systèmes embarqués des modèles résultants de cette méthode comme nous nous sous sommes plus intéressés au développement globale de cette nouvelle approche. Il serait intéressant d'avoir des résultats complémentaires sur ces appareils.

Le développement de méthodes similaires à la notre pourrait aussi être envisagé dans le cadre d'approches s'apparentant à de la distillation de connaissances. En effet, transférer la connaissance déjà apprise par un réseau dans un autre réseau de taille similaire permettrait d'obtenir une précision équivalente tout en supprimant de plus nombreux

paramètres pendant l'apprentissage par transfert via des méthodes de compression adéquates.

Nous pensons que sur le long terme, le sujet le plus important de recherche à développer est celui de l'interprétation et de la compréhension précise des mécanismes d'un CNN. Une meilleure compréhension des mécanismes des CNN pourrait nous mener à une meilleure compréhension du stockage et de la circulation de l'information dans ces modèles. Nous pensons que les futures études dans ce domaine permettront de mieux comprendre les comportements de ces modèles et permettront le développement de méthodes de compression et d'architectures plus efficaces et optimisées.

Références

[10., 1976] (1976). How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 194(1117) :431–445.

[Abadi, 2015] Abadi, M. (2015). TensorFlow : Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Ablavatski and Grishchenko, 2019] Ablavatski, A. and Grishchenko, I. (2019). Real-time ar self-expression with machine learning. https ://ai.googleblog.com/2019/03/real-time-ar-self-expression-with.html.

[Aguilar et al., 2020] Aguilar, G., Ling, Y., Zhang, Y., Yao, B., Fan, X., and Guo, C. (2020). Knowledge distillation from internal representations.

[Ahmad et al., 2017] Ahmad, J., Beers, J., Ciurus, M., Critz, R., Katz, M., Pereira, A., Pringle, M., and Rames, J. (2017). *iOS 11 by Tutorials : Learning the New iOS APIs with Swift 4*. Razeware LLC, 1st edition.

[Alvarez and Salzmann, 2016] Alvarez, J. M. and Salzmann, M. (2016). Learning the number of neurons in deep networks. pages 2270–2278.

[Anwar et al., 2017] Anwar, S., Hwang, K., and Sung, W. (2017). Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems*.

[Ba and Caruana, 2014] Ba, J. and Caruana, R. (2014). Do deep nets really need to be deep ? *NIPS*, pages 2654–2662.

[Ba and Frey, 2013] Ba, J. and Frey, B. (2013). Adaptive dropout for training deep neural networks. pages 3084–3092.

[Bach et al., 2012] Bach, F., Jenatton, R., Mairal, J., and Obozinski, G. (2012). Optimization with sparsity-inducing penalties. *Found. Trends Mach. Learn.*, page 1–106.

[Badrinarayanan et al., 2017] Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). SegNet : A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Baoyuan Liu et al., 2015] Baoyuan Liu, Min Wang, Foroosh, H., Tappen, M., and Penksy, M. (2015). Sparse convolutional neural networks. *CVPR*, pages 806–814.

[Bazarevsky and Tkachenka, 2018] Bazarevsky, V. and Tkachenka, A. (2018). Mobile real-time video segmentation. https ://ai.googleblog.com/2018/03/mobile-real-time-video-segmentation.html.

[Berthelier et al., 2020a] Berthelier, A., Chateau, T., Duffner, S., Garcia, C., and Blanc, C. (2020a). Deep Model Compression and Architecture Optimization for Embedded Systems : A Survey. *Journal of Signal Processing Systems*.

[Berthelier et al., 2019] Berthelier, A., Yan, Y., Chateau, T., Blanc, C., Duffner, S., and Garcia, C. (2019). Compression de réseaux convolutifs par utilisation d'un terme de clarté l1/l2 sur les noyaux. In *RFIAP*.

[Berthelier et al., 2020b] Berthelier, A., Yan, Y., Chateau, T., Blanc, C., Duffner, S., and Garcia, C. (2020b). Learning sparse filters in deep convolutional neural networks with a l1/l2 pseudo-norm. *ICPR Workshop*.

[Bichlmeier et al., 2007] Bichlmeier, C., Wimmer, F., Heining, S. M., and Navab, N. (2007). Contextual anatomic mimesis hybrid in-situ visualization method for improving multi-sensory depth perception in medical augmented reality. In *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*.

[Bottou, 2010] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Computational Statistics*, pages 177–186. Springer.

[Bozinovski, 2020] Bozinovski, S. (2020). Reminder of the first paper on transfer learning in neural networks, 1976. *Informatica*, 44.

[Brostow et al., 2008] Brostow, G., Shotton, J., Fauqueur, J., and Cipolla, R. (2008). Segmentation and recognition using structure from motion point clouds.

[Buciluǎ et al., 2006] Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. pages 535–541.

[Bulat and Tzimiropoulos, 2017] Bulat, A. and Tzimiropoulos, G. (2017). Binarized convolutional landmark localizers for human pose estimation and face alignment with limited resources.

[Cai et al., 2018a] Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. (2018a). Efficient Architecture Search by Network Transformation. pages 2787–2794.

[Cai et al., 2018b] Cai, H., Yang, J., Zhang, W., Han, S., and Yu, Y. (2018b). Path-level network transformation for efficient architecture search. In *Proceedings of the 35th International Conference on Machine Learning*, pages 678–687.

[Castro et al., 2017] Castro, W., Oblitas, J., Santa-Cruz, R., and Avila-George, H. (2017). Multilayer perceptron architecture optimization using parallel computing techniques. *PLOS ONE*, 12 :1–17.

[Caudell and Mizell, 1992] Caudell, T. P. and Mizell, D. W. (1992). Augmented reality : an application of heads-up display technology to manual manufacturing processes. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, volume ii, pages 659–669 vol.2.

[Chang et al., 2018] Chang, H., Lu, J., Yu, F., and Finkelstein, A. (2018). Pairedcyclegan : Asymmetric style transfer for applying and removing makeup. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 40–48.

[Chen et al., 2018] Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. In *European Conference on Computer Vision (ECCV)*.

[Chen et al., 2015] Chen, W., Wilson, J., Tyree, S., Weinberger, K., and Chen, Y. (2015). Compressing neural networks with the hashing trick. pages 2285–2294.

[Chen et al., 2017] Chen, Y., Wang, N., and Zhang, Z. (2017). Darkrank : Accelerating deep metric learning via cross sample similarities transfer.

[Cheng et al., 2017] Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv :1710.09282*.

[Choi et al., 2017] Choi, Y., El-Khamy, M., and Lee, J. (2017). Towards the limit of network quantization. *ICLR*.

[Chollet, 2016] Chollet, F. (2016). Xception : Deep learning with depthwise separable convolutions. *arXiv preprint arXiv :1610.02357*.

[Chuangxia et al., 2013] Chuangxia, H., Hanfeng, K., Xiaohong, C., and Fenghua, W. (2013). An lmi approach for dynamics of switched cellular neural networks with mixed delays. *Abstract and Applied Analysis*.

[Chuangxia et al., 2016] Chuangxia, H., Jie, C., and Peng, W. (2016). Attractor and boundedness of switched stochastic cohen-grossberg neural networks. *Discrete Dynamics in Nature and Society*.

[Courbariaux et al., 2014] Courbariaux, M., Bengio, Y., and David, J.-P. (2014). Training deep neural networks with low precision multiplications. *ICLR*.

[Courbariaux et al., 2015] Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Binaryconnect : Training deep neural networks with binary weights during propagations. pages 3123–3131.

[Courbariaux et al., 2016] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks : Training deep neural networks with weights and activations constrained to+ 1 or-1.

[Dai et al., 2016] Dai, J., Li, Y., He, K., and Sun, J. (2016). R-FCN : object detection via region-based fully convolutional networks. *NIPS*.

[Dauphin and Bengio, 2013] Dauphin, Y. N. and Bengio, Y. (2013). Big neural networks waste capacity.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet : A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[Deng et al., 2017] Deng, L., Jiao, P., Pei, J., Wu, Z., and Li, G. (2017). Gated xnor networks : Deep neural networks with ternary weights and activations under a unified discretization framework.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, pages 2121–2159.

[Elsken et al., 2018] Elsken, T., Metzen, J. H., and Hutter, F. (2018). Simple and efficient architecture search for convolutional neural networks.

[Farabet et al., 2011] Farabet, C., Martini, B., Corda, B., Akselrod, P., Culurciello, E., and Lecun, Y. (2011). NeuFlow : A runtime reconfigurable dataflow processor for vision. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*.

[Feng et al., 2018] Feng, Y., Wu, F., Shao, X., Wang, Y., and Zhou, X. (2018). Joint 3d face reconstruction and dense alignment with position map regression network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 534–551.

[Floreano et al., 2008] Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution : from architectures to learning. *Evolutionary Intelligence*, 1(1) :47–62.

[Frankle and Carbin, 2019] Frankle, J. and Carbin, M. (2019). The lottery ticket hypothesis : Finding sparse, trainable neural networks. In *ICLR*.

[Fritzke, 1994a] Fritzke, B. (1994a). Growing cell structures-A self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9) :1441–1460.

[Fritzke, 1994b] Fritzke, B. (1994b). Supervised Learning with Growing Cell Structures. *Advances in Neural Information Processing Systems 6*, (1989) :255–262.

[Fritzke, 1995] Fritzke, B. (1995). A Growing Neural Gas Learns Topologies. *Advances in Neural Information Processing Systems*, 7 :625–632.

[Fritzke and Bochum, 1994] Fritzke, B. and Bochum, R.-u. (1994). Fast learning with incremental RBF Networks 1 Introduction 2 Model description. *Processing*, 1(1) :2–5.

[Fukushima, 1975] Fukushima, K. (1975). Cognitron : A self-organizing multilayered neural network. *Biological cybernetics*, 20(3-4) :121–136.

[Gatys et al., 2016] Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423.

[Gionis et al., 1999] Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity Search in High Dimensions via Hashing. *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529.

[Gokhale et al., 2014] Gokhale, V., Jin, J., Dundar, A., Martini, B., and Culurciello, E. (2014). A 240 G-ops/s mobile coprocessor for deep neural networks. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 696–701.

[Gong et al., 2014] Gong, Y., Liu, L., Yang, M., and Bourdev, L. (2014). Compressing deep convolutional networks using vector quantization.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Gupta et al., 2015] Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). Deep Learning with Limited Numerical Precision. *ICML*, pages 1737–1746.

[Hammerstrom, 1990] Hammerstrom, D. (1990). A VLSI architecture for high-performance, low-cost, on-chip learning. *IJCNN International Joint Conference on Neural Networks*, pages 537–544.

[Han et al., 2016] Han, S., Mao, H., and Dally, W. J. (2016). Deep Compression - Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *ICLR*, pages 1–13.

[Han et al., 2015] Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both weights and connections for efficient neural network. *NIPS*, pages 1135–1143.

[Hassibi and Stork, 1993] Hassibi, B. and Stork, D. G. (1993). Second order derivatives for network pruning : Optimal brain surgeon. pages 164–171.

[He et al., 2017a] He, K., Gkioxari, G., Dollár, P., and Girshick, R. B. (2017a). Mask R-CNN. *ICCV*.

[He and Sun, 2015] He, K. and Sun, J. (2015). Convolutional neural networks at constrained time cost. pages 5353–5360.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers : Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *CVPR*, pages 770–778.

[He et al., 2019] He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. *CVPR*.

[He et al., 2017b] He, Y., Zhang, X., and Sun, J. (2017b). Channel pruning for accelerating very deep neural networks. *ICCV*, pages 1398–1406.

[Hinton et al., 2014] Hinton, G., Vinyals, O., and Dean, J. (2014). Distilling the Knowledge in a Neural Network. *NIPS 2014 Deep Learning Workshop*, pages 1–9.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., and Swersky, K. (2012). Lecture 6a-overview of mini-batch gradient descent. *COURSERA : Neural Networks for Machine Learning*, page 31.

[Holt and Hwang, 1993] Holt, J. L. and Hwang, J. N. (1993). Finite precision error analysis of neural network hardware implementations. *IEEE Transactions on Computers*, pages 281–290.

[Howard et al., 2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets : Efficient convolutional neural networks for mobile vision applications.

[Huang et al., 2017] Huang, G., Liu, Z., Weinberger, K. Q., and Maaten, L. V. D. (2017). Densely connected convolutional networks. *CVPR*.

[Huang and Zhang, 2010] Huang, J. and Zhang, T. (2010). The benefit of group sparsity. *Ann. Statist.*, pages 1978–2004.

[Huang et al., 2015] Huang, Q., Feng, J., Zhang, Y., Fang, Q., and Ng, W. (2015). Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, pages 1–12.

[Huang and Wang, 2017] Huang, Z. and Wang, N. (2017). Like what you like : Knowledge distill via neuron selectivity transfer.

[Iandola et al., 2016] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet : Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization : Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

[Iwata et al., 1989] Iwata, A., Yoshida, Y., Matsuda, S., Sato, Y., and Suzumura, Y. (1989). An artificial neural network accelerator using general purpose 24 bit floating point digital signal processors. *Proc. IJCNN*, pages 171–175.

[Jaderberg et al., 2014] Jaderberg, M., Vedaldi, A., and Zisserman, A. (2014). Speeding up convolutional neural networks with low rank expansions. *BMVC*.

[Jin et al., 2018] Jin, H., Song, Q., and Hu, X. (2018). Efficient Neural Architecture Search with Network Morphism.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam : A method for stochastic optimization.

[Kohonen, 1982] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1) :59–69.

[Kolda and Bader, 2009] Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, pages 455–500.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *NIPS*, pages 1097–1105.

[Lebedev and Lempitsky, 2016] Lebedev, V. and Lempitsky, V. (2016). Fast convnets using group-wise brain damage. pages 2554–2564.

[LeCun, 1989] LeCun, Y. (1989). Generalization and network design strategies. *Connections in Perspective. North-Holland, Amsterdam*, pages 143–155.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, pages 436–444.

[LeCun et al., 1990a] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990a). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, pages 2278–2324.

[LeCun et al., 1990b] LeCun, Y. L., Denker, J. S., and Solla, S. a. (1990b). Optimal Brain Damage. *Advances in Neural Information Processing Systems*, pages 598–605.

[Lee et al., 2020] Lee, H., Hwang, S. J., and Shin, J. (2020). Self-supervised label augmentation via input transformations. *ICML*.

[Li et al., 2017] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2017). Pruning filters for efficient ConvNets. *ICLR*, pages 1–10.

[Liao et al., 2017] Liao, J., Yao, Y., Yuan, L., Hua, G., and Kang, S. B. (2017). Visual attribute transfer through deep image analogy. *ACM Transactions on Graphics (TOG)*, 36(4) :1–15.

[Lin et al., 2017a] Lin, J., Rao, Y., Lu, J., and Zhou, J. (2017a). Runtime neural pruning. pages 2178–2188.

[Lin et al., 2019] Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F., and Doermann, D. S. (2019). Towards optimal structured CNN pruning via generative adversarial learning. *CVPR*, pages 2790–2799.

[Lin et al., 2017b] Lin, X., Zhao, C., and Pan, W. (2017b). Towards accurate binary convolutional neural network. pages 344–352.

[Liu and Ye, 2010] Liu, J. and Ye, J. (2010). Efficient l1/lq norm regularization.

[Liu et al., 2016] Liu, S., Ou, X., Qian, R., Wang, W., and Cao, X. (2016). Makeup like a superstar : deep localized makeup transfer network. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2568–2575.

[Liu et al., 2017] Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. (2017). Learning Efficient Convolutional Networks through Network Slimming. *ICCV*, pages 2736–2744.

[Long et al., 2015] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. pages 3431–3440.

[Luo et al., 2017] Luo, J.-H., Wu, J., and Lin, W. (2017). Thinet : A filter level pruning method for deep neural network compression. *ICCV*.

[Mamalet and Garcia, 2012] Mamalet, F. and Garcia, C. (2012). Simplifying convnets for fast learning. *ICANN 2012*, pages 58–65.

[Mamalet et al., 2007] Mamalet, F., Roux, S., and Garcia, C. (2007). Real-time video convolutional face finder on embedded platforms. *Eurasip Journal on Embedded Systems*.

[Martinetz et al., 1993] Martinetz, T. M., Berkovich, S. G., and Schulten, K. J. (1993). "Neural-Gas" Network for Vector Quantization and its Application to Time-Series Prediction.

[Miikkulainen et al., 2017] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., and Hodjat, B. (2017). Evolving Deep Neural Networks.

[Molchanov et al., 2017] Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2017). Pruning convolutional neural networks for resource efficient transfer learning. *ICLR*.

[Montana and Davis, 1989] Montana, D. J. and Davis, L. (1989). Training feedforward neural networks using genetic algorithms. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 762–767.

[Müller et al., 2019] Müller, R., Kornblith, S., and Hinton, G. (2019). When does label smoothing help ? *NIPS*.

[Nanfack et al., 2017] Nanfack, G., Elhassouny, A., and Thami, R. O. H. (2017). Squeeze-segnet : A new fast deep convolutional neural network for semantic segmentation. *CoRR*.

[Newell et al., 2016] Newell, A., Yang, K., and Deng, J. (2016). Stacked hourglass networks for human pose estimation. pages 483–499.

[Pham et al., 2018] Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80.

[Radcliffe, 1993] Radcliffe, N. J. (1993). Genetic set recombination and its application to neural network topology optimisation. *Neural Computing & Applications*, 1(1) :67–90.

[Radu, 2012] Radu, I. (2012). Why should my students use ar ? a comparative review of the educational impacts of augmented-reality. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*.

[Rastegari et al., 2016] Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). Xnor-net : Imagenet classification using binary convolutional neural networks. pages 525–542.

[Redmon et al., 2015] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. (2015). You only look once : Unified, real-time object detection.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net : Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241. Springer.

[Rosenberg, 1993] Rosenberg, L. B. (1993). Virtual fixtures : Perceptual tools for telerobotic manipulation. In *IEEE Virtual Reality Annual International Symposium (VR)*.

[Rosenblatt, 1961] Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY.

[Rosenblatt, 1962] Rosenblatt, F. (1962). Perceptrons and the Theory of Brain Mechanics. page 621.

[Roux et al., 2007] Roux, S., Mamalet, F., Garcia, C., and Duffner, S. (2007). An embedded robust facial feature detector. *Proceedings of the 2007 IEEE Signal Processing Society Workshop, MLSP*, pages 170–175.

[Rumelhart et al., 1988] Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive Modeling*, 5(3) :1.

[Sainath et al., 2013] Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., and Ramabhadran, B. (2013). Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets. *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6655–6659.

[Sandler et al., 2018a] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018a). Inverted Residuals and Linear Bottlenecks : Mobile Networks for Classification, Detection and Segmentation.

[Sandler et al., 2018b] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018b). Mobilenetv2 : Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520.

[Saxena and Verbeek, 2016] Saxena, S. and Verbeek, J. (2016). Convolutional Neural Fabrics. *NIPS 2016*.

[Schwab, 2013] Schwab, S. (2013). *Suivi visuel multi-cibles par partitionnement de détections : application à la construction d'albums de visages*. PhD thesis, Clermont-Ferrand 2.

[Shah et al., 2016] Shah, A., Kadam, E., Shah, H., Shinde, S., and Shingade, S. (2016). Deep residual networks with exponential linear unit. In *Proceedings of the Third International Symposium on Computer Vision and the Internet*, pages 59–65. ACM.

[Shinde et al., 2010] Shinde, R., Goel, A., Gupta, P., and Dutta, D. (2010). Similarity search and locality sensitive hashing using ternary content addressable memories. pages 375–386.

[Shrivastava and Li, 2014] Shrivastava, A. and Li, P. (2014). Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). pages 2321–2329.

[Sifre and Stephane, 2014] Sifre, L. and Stephane, M. (2014). Rigid-Motion Scattering For Image Classification. *CoRR*.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale Image recognition. *CoRR*, pages 1–14.

[Smith et al., 2013] Smith, B. M., Zhang, L., Brandt, J., Lin, Z., and Yang, J. (2013). Exemplar-based face parsing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3484–3491.

[Spring and Shrivastava, 2017] Spring, R. and Shrivastava, A. (2017). Scalable and sustainable deep learning via randomized hashing. pages 445–454.

[Srivastava, 2013] Srivastava, N. (2013). Improving Neural Networks with Dropout. *Master's thesis*.

[Srivastava et al., 2014] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout : a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, pages 1929–1958.

[Stanley and Miikkulainen, 2002] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*.

[Sundaram et al., 2013] Sundaram, N., Turmukhametova, A., Satish, N., Mostak, T., Indyk, P., Madden, S., and Dubey, P. (2013). Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *Proceedings of the VLDB Endowment*, pages 1930–1941.

[Sutherland, 1968] Sutherland, I. E. (1968). A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 757–764, New York, NY, USA. ACM.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. *CVPR*, pages 1–9.

[Tan et al., 2019] Tan, M., Chen, B., Pang, R., Vasudevan, V., and Le, Q. V. (2019). MnasNet : Platform-Aware Neural Architecture Search for Mobile. *CVPR*.

[Thierens, 1996] Thierens, D. (1996). Non-redundant genetic coding of neural networks. pages 571–575.

[Turlach et al., 2000] Turlach, B., Venables, W., and Wright, S. (2000). Simultaneous variable selection. *Technometrics*.

[Vanhoucke et al., 2011] Vanhoucke, V., Senior, A., and Mao, M. Z. (2011). Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*.

[Veniat and Denoyer, 2018] Veniat, T. and Denoyer, L. (2018). Learning time/memory-efficient deep architectures with budgeted super networks. In *Conference on Computer Vision and Pattern Recognition*, pages 3492–3500.

[Vu, 2010] Vu, N.-S. (2010). *Towards unconstrained face recognition from one sample*. PhD thesis, Institut National Polytechnique de Grenoble-INPG.

[Wei et al., 2016] Wei, S.-E., Ramakrishna, V., Kanade, T., and Sheikh, Y. (2016). Convolutional pose machines. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4732.

[Weinberger et al., 2009] Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. (2009). Feature hashing for large scale multitask learning. pages 1113–1120.

[Wen et al., 2016] Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). Learning structured sparsity in deep neural networks. *NIPS*, page 2082–2090.

[Williamson, 1991] Williamson, D. (1991). Dynamically scaled fixed point arithmetic. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference Proceedings*, pages 315–318.

[Wilson et al., 2017] Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158.

[Wu and He, 2018] Wu, Y. and He, K. (2018). Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19.

[Wu et al., 2016] Wu, Z., Shen, C., and van den Hengel, A. (2016). High-performance semantic segmentation using very deep fully convolutional networks.

[Yan et al., 2018] Yan, Y., Bout, B., Berthelier, A., Naturel, X., and Chateau, T. (2018). Face parsing for mobile ar applications. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*.

[Yu et al., 2018] Yu, R., Li, A., Chen, C., Lai, J., Morariu, V. I., Han, X., Gao, M., Lin, C., and Davis, L. S. (2018). NISP : pruning networks using neuron importance score propagation. *CVPR*.

[Yuan and Lin, 2006a] Yuan, M. and Lin, Y. (2006a). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society : Series B (Statistical Methodology)*, pages 49–67.

[Yuan and Lin, 2006b] Yuan, M. and Lin, Y. (2006b). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B*, pages 49–67.

[Zeiler, 2012] Zeiler, M. D. (2012). ADADELTA : An Adaptive Learning Rate Method. page 6.

[Zhang et al., 2017] Zhang, X., Zhou, X., Lin, M., and Sun, J. (2017). Shufflenet : An extremely efficient convolutional neural network for mobile devices.

[Zhou et al., 2016] Zhou, H., Alvarez, J. M., and Porikli, F. (2016). *Less Is More : Towards Compact CNNs*, pages 662–677. Springer International Publishing, Cham.

[Zhuang et al., 2018] Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., and Zhu, J. (2018). Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems 31*, pages 875–886.

[Zoph and Le, 2017] Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning.

[Zoph et al., 2016] Zoph, B., Yuret, D., May, J., and Knight, K. (2016). Transfer Learning for Low-Resource Neural Machine Translation.