République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي

ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الإعلام الآلي سابقا)
Ecole nationale Supérieure d'Informatique
ex. INI (Institut National de formation en Informatique)

Graduation Thesis

Image deep learning techniques for resource limited devices, within the context of free parking slot detection.

Auteurs:

Mohammed Habib Allah Kechout

Sebti Tamraoui

Encadreurs:

Lotfi Bouziani

Ahmed Mostefaoui

Année universitaire : 2019/2020

Remerciements

Ici c'est le remerciement...

Abstract

Ici c'est l'abstract

Résumé

Ici c'est le résumé...

Introduction générale

Ici c'est l'introduction générale...

Table des matières

List of Algorithms

Table des figures

Liste des tableaux

Acronymes

CNN Convolutional Neural Network. 11, 29

R-CNN Region Based Convolutional Neural Network. 32

Première partie

Théorie Et Contexte

Chapitre 1

Présentation du projet

1.1 Introduction

Une grande partie des travaux de recherches récents sur les réseaux de neurones ont porté sur l'amélioration de la précision sur des taches de classification, segmentation et détection, et du coup Il apparaît qu'une tendance générale dans leurs conceptions était de trouver des modèles plus grands et plus profonds pour obtenir une meilleure précision sans tenir vraiment compte de la mémoire, temps de calcule/d'entraînement ou de la consommation d'énergie.
Vouloir transmettre les avancées des techniques du Deep Learning vers des appareils limités en ressources a soulevé le problème suivant : est ce qu'il est possible de trouver un bon compromis entre les performances du réseau et sa consommation de ressources ?
L'objectif de ce chapitre est de définir les notions de base sur les techniques du Deep Learning relatives au traitement d'image afin de cadrer notre travail et de présenter ses motivations. Nous commençons d'abord par présenter l'institution impliquée ensuite dans la section 1.3 nous détaillons quelques termes et notions utilisés tout au long de ce rapport, et finallement détailler le problème et les défis à surmonter.

1.2 Institutions impliquées

Ici on présente brievement les institutions impliquées.

1.3 Background

Dans cette section nous allons définir les notions de bases relatives à l'apprentissage profond dans le contexte du traitement d'image notamment sur les taches de classification et détection, ces termes vont être utilisés tout au long du document. Nous commençons par celles des réseaux de neurones. 1.3.1 ensuite celles des réseaux de neurones de convolution 1.3.2.

1.3.1 Réseaux de neurones

1.3.1.1 Introduction

Les réseaux de neurones formels sont des modèles théoriques de traitement de l'information inspirés des observations relatives au fonctionnement des neurones biologiques et du cortex cérébral.

Les réseaux de neurones constituent aujourd'hui une des techniques de classification les plus utilisées et les plus performantes dans différentes tâches. les réseaux de neurones encodent une fonction mathématique à appliquer sur un signal d'entrée (dans notre cas, une image) et permettant de prédire un signal de sortie. Cette fonction est une composition de plusieurs fonctions non-linéaires. Ils sont constitués d'un grand nombre de neurones artificiels connectés entre eux et modélisant le fonctionnement des neurones biologiques.

1.3.1.2 Perceptron

Le perceptron est l'unité de base des réseaux de neurones. Il s'agit d'une fonction, prenant un vecteur d'entrée x et qui produit une sortie y en fonction d'un vecteur de paramètres (souvent appelé poids) w. La sortie y correspond à une somme pondérée des entrées x et des poids w à laquelle est ajouté un biais b, voir figure 1.1.

$$\varphi(x, w) = h(\sum_i w_i * x + b) = y \tag{1.1}$$

L'équation 1.1 fait aussi apparaître une fonction notée h(). Cette dernière, appelée fonction d'activation est une fonction non linéaire, nous détaillerons son utilité ainsi que les différentes fonctions usuellement utilisées.

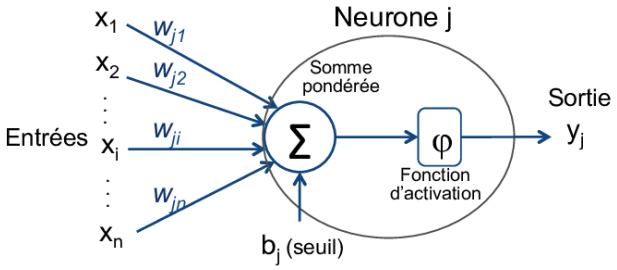


FIGURE 1.1 – Perceptron

Un perceptron ne donne qu'une seule valeur sortie y, or souvent on peut vouloir des prédictions plus complexes composées de plusieurs valeurs de sortie, regroupées sous la forme d'un vecteur y. Par exemple, une tâche pourrait consister à prédire des coordonnées x, y dans une image d'entrée. On peut aussi faire de la classification multi-classes d'objets en prédisant un vecteur de probabilités $y = [y_1, y_2, ..., y_n]$, où chaque valeur de sortie y_i

correspond à la probabilité d'appartenance à une classe d'indice i. Afin d'obtenir plusieurs sorties, il est nécessaire d'utiliser plusieurs perceptrons de manière parallèle, regroupés pour former ce que l'on appelle une couche de perceptrons. Une couche est dite entièrement connectée quand tous les perceptrons sont connectés à la totalité des entrées. Dans ce cas, les poids des perceptrons sont regroupés sous forme de matrice W et l'évaluation d'une couche de perceptrons revient à calculer le produit matriciel entre les entrées x et les poids W.

1.3.1.3 Activation Function

Les perceptrons, vus dans la partie précédente, modélisent des fonctions linéaires. Or, mathématiquement, la combinaison de plusieurs fonctions linéaires reste une fonction linéaire. Par conséquent, quel que soit le nombre de perceptrons et de couches de perceptrons utilisés, si l'on n'introduit pas de la non-linéarité dans les calculs, la fonction objective calculée par le réseau de neurones restera une fonction linéaire. C'est pourquoi l'on utilise une fonction appelée fonction d'activation ou fonction non linéaire, dont le rôle est d'introduire une non linéarité dans les calculs. La fonction d'activation $h(x)$ est généralement définie par l'une des fonctions suivante :

- sigmoid :

$$h(Z) = \frac{1}{1 + e^{-Z}} \tag{1.2}$$

- Tangente hyperbolique :

$$h(Z) = tanh(Z) = \frac{1 - e^{-Z}}{1 + e^{-Z}} \tag{1.3}$$

- ReLu :

$$h(Z) = ReLu(Z) = max(0, Z) \tag{1.4}$$

Avec $Z = X * W + b$

1.3.1.4 Architecture :

L'architecture d'un réseau de neurones détermine la façon par laquelle les neurones sont ordonnés et connectés au sein d'un même réseau. Quel que soit l'architecture, un réseau de neurones est composé de plusieurs couches de neurones successives : les entrées, les couches suivantes dites couches cachées (du fait qu'elles n'ont aucun contact direct avec l'extérieur du réseau) jusqu'à la couche de sorties. La profondeur d'une telle architecture dépend du nombre de couches cachées.

1.3.1.5 Architecture feedforward

Un réseau de neurones de type feedforward est un réseau où l'information qui le traverse ne circule que dans un sens unique : de l'entrée vers la sortie [DGR88].
Ce réseau est constitué de plusieurs couches successives : on intercale entre la couche d'entrée et celle de la sortie une ou plusieurs couches cachées. Chaque neurone d'une couche est relié à tous ceux de la suivante. Les sorties de la couche inférieure sont propagées à travers les différentes couches, de l'entrée vers la sortie, la figure 1.2 montre un exemple d'une architecture feedforward.

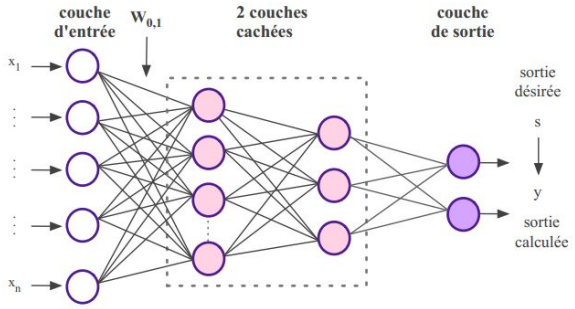


FIGURE 1.2 – Réseau de neurones feedforward avec deux couches cachées

En général : Un réseau de neurones correspond à une fonction $\Theta(x, \theta) = \hat{y}$, Cette dernière prend une entrée x et des paramètres θ, correspondant aux poids des différentes couches du réseau, et produit une sortie $\hat{y}$ modélisant le prédiction du réseau.

1.3.1.6 Entraînement d'un réseau de neurones :

L'entraînement d'un réseau de neurones est la manière de modifier les poids des réseaux de manière à ce que la fonction qu'ils modélisent permette de répondre à une tâche désirée.

1.3.1.7 Entraînement supervisé :

En apprentissage automatique supervisé, l'entraînement d'un réseau de neurones se fait en utilisant des bases d'entraînement B, ces dernières contiennent des données d'entraînement x associées à des annotations y (appelée vérités terrain) correspondant à la sortie désirée pour l'entrée x.

1.3.1.8 Fonction de perte :

Une fonction de perte $J(x_i, y_i, \theta) = \epsilon_i$ mesure la différence entre les estimation $\hat{y}_i = \Theta(x_i, \theta_i)$ et les vérités terrain y_i, l'objectif en apprentissage automatique est de trouver les paramètres θ minimisant la somme des pertes observées sur les exemples d'entraînement

de la base B. C'est-à-dire, de trouver les paramètres θ qui minimisent un risque empirique R tel que $R[\theta, B] = \frac{1}{N} \sum_i^N J(x_i, y_i, \theta)$.

1.3.1.9 Gradient descent :

Gradient descent est un algorithme itératif qui permet de minimiser la fonction de perte J_θ, paramétrée par des poids θ d'un réseau de neurones. Minimiser la fonction (c'est à dire trouver les paramètres θ qui minimisent l'erreur empirique) se fait en calculant le gradient $\nabla_\theta J(\theta)$ et en mettant a jour les paramètres θ dans la direction opposée du gradient de ce dernier. Intuitivement, le gradient $\nabla_\theta J(\theta)$ donne la direction de la pente de la fonction J au point θ, par conséquent, se diriger dans la direction opposée du gradient revient à descendre cette pente.

1.3.1.10 Pas d'apprentissage η :

Il s'agit d'un hyper-paramètre qui détermine la longueur du pas à faire dans la direction du gradient. Une fois les paramètres mis à jour, on calcule le nouveau gradient et on se dirige à nouveau dans la direction opposée. On minimise ainsi la fonction en avançant pas par pas dans la direction de la pente.
Ainsi les nouveaux poids de connexions θ du réseau de neurone seront calculer de la manière suivante :

$$\theta = \theta - \eta * \nabla_\theta J(\theta, B) \tag{1.5}$$

Il existe trois variantes de descente de gradient qui diffèrent par la quantité de données utilisées pour estimer le gradient de la fonction objectif. La première est la descente de gradient stochastique et utilise un seul exemple à la fois. La deuxième variante est la descente de gradient par "Mini-Batch", qui utilise un petit groupe d'exemples pour calculer le gradient. Finalement, la descente de gradient par "Batch" est la variante qui se sert de l'ensemble B complet pour faire son calcul, l'équation 1.5 montre la descente de gradient par "Batch".

1.3.1.11 Retropropagation :

Afin d'entraîner un réseau de neurone il faut calculer le gradient de perte en fonction de chacun des poids du modèle. Ce calcul est utilisé pour modifier la valeur des poids de façon à obtenir des prédictions qui se rapprochent des cibles. Calculer indépendamment la dérivée pour chaque paramètre peut s'avérer long et fastidieux, c'est pourquoi on utilise la technique de rétropropagation. En effet, cette technique nous permet de calculer de façon efficace les gradients de tous les poids d'un réseau de neurones. La rétropropagation porte bien son nom, car le flux de calculs fait le chemin inverse de la propagation avant. Elle commence donc par la sortie et se dirige vers l'entrée [Lau16].

L'idée est de calculer la dérivée de la fonction objectif par rapport à la couche de sortie pour ensuite propager cette information à travers le réseau jusqu'à l'entrée du modèle. Le gradient de chaque couche cachée est exprimé en réutilisant les dérivées des couches qui les suivent.

1.3.1.12 Sur-apprentissage

Lorsque l'on réalise un algorithme d'apprentissage automatique, on souhaite que celui-ci n'ait pas uniquement de bonnes performances sur les données d'apprentissages, mais sur toutes données qu'on lui présente.
On parle de sur-apprentissage lorsqu'on perd la capacité de généraliser, c'est à dire lorsqu'un réseau de neurones s'est trop spécialisé sur ses données d'apprentissages et qu'il a une forte erreur sur l'ensemble de test.

1.3.1.13 Régularisation

La régularisation est l'ensemble des méthodes qui permettent à l'algorithme de réduire son erreur de généralisation.
Une façon schématique de voir la régularisation est d'imaginer un ajout de bruit dans les réseaux pour les rendre plus robustes et moins spécialisés sur leurs données d'apprentissages. L'idée est donc de déstabiliser le réseau lors de l'apprentissage, pour qu'il ne soit pas limité aux exemples fournis pendant l'apprentissage, dans le but d'éviter un sur-apprentissage [Por18].

1.3.1.14 Conclusion :

Dans cette section, nous avons introduit les réseaux de neurones artificiels ainsi que leurs propriétés fondamentales.

Ces réseaux sont capable d'intervenir dans la résolution de nombreux problèmes de la modélisation ainsi que la classification, en revanche, leur performance est insuffisante en ce qui concerne le traitement d'image et ainsi l'introduction d'un nouveau type de réseaux de neurone est devenu une nécessité.

C'est pourquoi nous nous intéressons dans la section suivante à l'apprentissage profond et à des méthodes plus récentes à base de réseaux de neurones artificiels. Ces méthodes sont aujourd'hui appliquées dans la vision par ordinateur et offrent des performances comparables voire supérieures à celles de l'Homme.

1.3.2 Réseaux de neurones de convolution (CNN) :

1.3.2.1 Introduction :

Le Deep Learning, ou apprentissage profond, est une branche de l'apprentissage automatique. Ce sous-ensemble repose sur le principe des réseaux de neurones artificiels , utilisés cependant à une échelle beaucoup plus importante, reposant sur le fait qu'une augmentation du nombre de couches et de neurones d'un ANN augmente les performances de classification. Le concept d'apprentissage profond n'est devenu populaire que récemment, exigeant un très grand nombre de données annotées et une grande puissance de calculs.

Un des modèles d'apprentissage profond est le réseau de neurone de convolution. Le premier CNN a été introduit à la fin des années 80 par [LeC+89]. C'est le premier réseau de neurones pour la reconnaissance d'images. Depuis, les CNNs ont connus une amélioration révolutionnaire de ces résultats dans des divers domaines liés à la reconnaissance des formes. L'aspect le plus important des CNNs est la réduction du nombre de paramètres par rapport à un réseau de neurone classique.

L'avantage des CNNs est de pouvoir reconnaître une forme précise, cette forme ne dois pas forcement être spatialement dépendante. En d'autres termes, par exemple, dans une application de détection de visage, nous n'avons pas besoin de faire attention à l'emplacement des visages dans les images. La seule préoccupation est de les détecter quelle que soit leur position dans les images données.

1.3.2.2 Convolution :

Une convolution est une opération mathématique qui consiste à multiplier, ou « convoluer » une matrice (qui représente par exemple une image) par une autre appelée matrice de convolution, ou « filtre ». Les paramètres de la couche de convolution consistent en un ensemble de filtres apprenables. Chaque filtre est petit spatialement (le long de la largeur et de la hauteur), mais s'étend à travers toute la profondeur de l'image d'entrée. Pendant l'apprentissage, chaque filtre est convolué avec l'image. Ceci produit une carte de caractéristiques (features map) de 2-dimensions qui donnent les réponses de ce filtre à chaque position spatiale dans l'image, la figure 1.3 montre un exemple de convolution. Intuitivement, le réseau va apprendre des filtres qui s'activent lorsqu'ils voient un type de caractéristique visuelle comme le contour, les orientations, etc., dans la première couche du réseau. L'ensemble de filtres dans la couche de convolution produisent chacun une carte de caractéristiques, qui sont ensuite empilées le long de la dimension de profondeur pour former la sortie[GHA17].

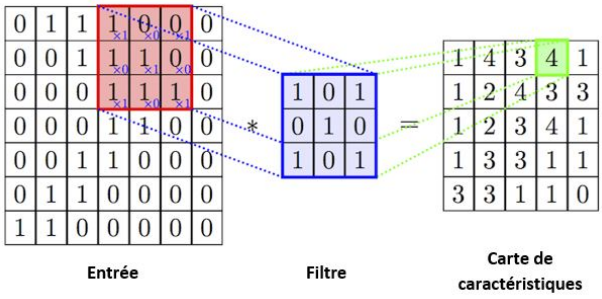


FIGURE 1.3 – convolution d'une entrée par un filtre 3x3

1.3.2.3 Padding :

L'un des inconvénients de l'étape de convolution est la perte des informations pouvant exister sur le bord de l'image, Parce qu'ils ne sont capturés que lorsque le filtre glisse, ils n'ont jamais la chance d'être vu. Une approche très simple pour résoudre le problème, consiste à ajouter des zéros supplémentaires autour des frontières de l'entrée, comme montrer dans la figure 1.4, augmentant ainsi la taille effective de l'entrée, cette opération est zero-padding.

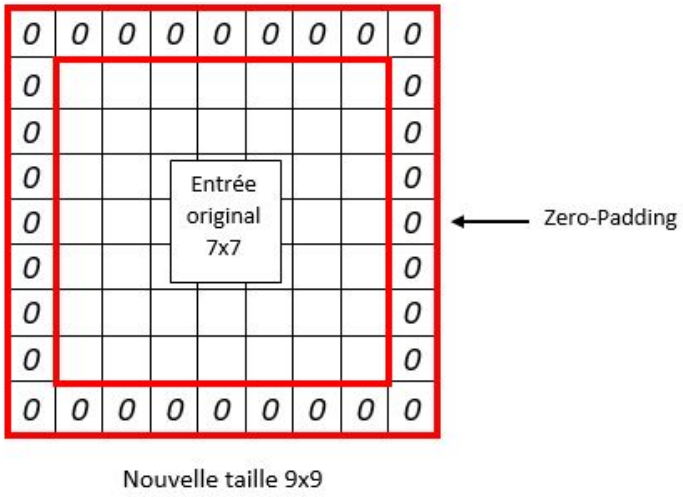


FIGURE 1.4 – application du zero-padding a une entrée 7x7

En général : soient e_h, e_l la hauteur et la largeur de l'entrée, f_h, f_l la hauteur et la largeur du filtre, p_h, p_l les lignes et les colonne ajouter lors du zero-padding et s_h, s_l le pas en hauteur et en largeur. et la taille de la sortie est :

$$[(e_h - f_h + p_h + s_h)/s_h]X[e_l - f_l + p_l + s_l)/s_l] \tag{1.6}$$

1.3.2.4 Pooling :

Soient n_i le nombre d'entrées de la i-ème couche convolutive, et h_i, w_i la hauteur et la largeur de la 'feature map' d'entrée notée $x_i \in \mathbb{R}^{n_i \times h_i \times w_i}$ qui est transformée par la couche convolutive en une feature map de sortie notée $x_{i+1} \in \mathbb{R}^{n_{i+1} \times h_{i+1} \times w_{i+1}}$ qui est utilisée à son tour comme entrée de la couche suivante. La transformation consiste en l'application de n_{i+1} filtres 3D qu'on note $F_{i,j} \in \mathbb{R}^{n_i \times k \times k}$ à l'ensemble des feature maps d'entrée pour donner une seule feature map de sortie et donc on aura n_{i+1} feature maps de sortie. L'ensemble des filtres est appelé 'kernel matrix' et est noté $F_i \in \mathbb{R}^{n_i \times n_{i+1} \times k \times k}$.

1.3.2.5 Dropout :

Pour éviter le sur-apprentissage (overfitting), la couche de dropout a été introduite [Srivastava2014]. Cette couche est utilisée pendant l'apprentissage. Elle permet de désactiver aléatoirement des neurones durant les différentes itérations de l'apprentissage. En d'autres termes, le dropout permet au réseau d'apprendre des sous-réseaux contenant moins de paramètres et donc moins sujets au sur-apprentissage. Cette manière de faire permet d'apprendre des paramètres plus génériques qui ne se focalisent pas sur des détails de la base d'apprentissage. Une fois l'apprentissage terminé, tous les neurones sont réactivés.

1.3.2.6 Architecture Classique :

Nous présentons ici une architecture de réseaux de convolution profonds utilisées couramment dans les travaux de recherche en vision par ordinateur.

AlexNet : Cette architecture est celle proposée par [KSH12] et ayant permis la recrudescence de l'étude des réseaux de neurones à partir de 2012, notamment, grâce à la victoire lors de la compétition de classification d'images ImageNet. Cette architecture utilise cinq couches de convolution et trois couches de pooling. La taille des filtres de convolution est variable (11×11, 5×5, 3×3) en fonction de la couche considérée. La fonction d'activation utilisée entre chaque couche est la fonction ReLU. Après le passage de l'image dans les couches de convolution, de pooling et d'activation, une carte de caractéristiques est obtenue. Celle-ci est envoyée dans un perceptron multicouches, qui permettra d'effectuer la classification, composé de deux couches cachées et d'une couche de sortie. La Figure 1.5 illustre l'architecture de l'AlexNet.

Transfer Leaning : L'apprentissage des réseaux profond est difficile du fait de leur grand nombre de paramètres et du manque de données d'apprentissage.
Transfer Learning est une pratique commune qui utilise des CNN déjà appris sur d'immenses bases de données pour ensuite les adapter à une tâche spécifique. C'est ce que

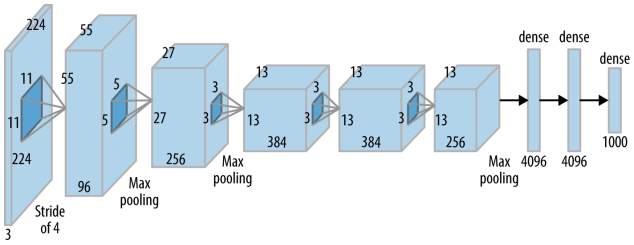


FIGURE 1.5 – Architecture AlexNet

l'on appelle le fine-tuning. Cette pratique permet d'apprendre des réseaux profonds en utilisant une initialisation des poids et des biais déjà très pertinente et générique. L'adaptation de ces paramètres est ensuite réalisée durant la phase d'apprentissage de la tâche spécifique que l'on souhaite réaliser. Cela a pour incidence une vitesse d'apprentissage bien plus rapide et une convergence quasiment garantie.

1.3.2.7 Détection d'objet :

La détection d'objet consiste à définir des régions d'intérêt (RoI), très souvent rectangulaire, dans une image (localisation) et de leur attribuer une classe d'objet (classification).

R-CNN : R-CNN est le premier détecteur basé sur les CNN, il a été introduit par Ross Girshick [Gir+14].
Le principe de cette approche est de proposer des boites avec une forte probabilité d'être une *région d'intérêt*. Ces régions proposées sont ensuite extraites et envoyées à un classifieur pour affecter une classe à chacune.
Un des algorithmes pour la proposition des régions d'intérêt est *Selective Search* introduit par [Uij+13] cette méthode se base sur une segmentation de l'image à différentes résolutions. En utilisant la méthode de segmentation introduite par [FH04], la *Selective Search* segmente l'image d'entrée sur plusieurs échelles. Cela produit un premier ensemble de régions d'intérêts. Les auteurs de [Uij+13] introduisent ensuite un calcul de similarité entre régions basé sur des informations de couleurs, de textures, de tailles et d'inclusions. Cette similarité permet de fusionner les régions redondantes (trop similaires) et de renvoyer un ensemble de propositions d'objets pertinents. La figure 1.6 illustre l'algorithme de selective search.

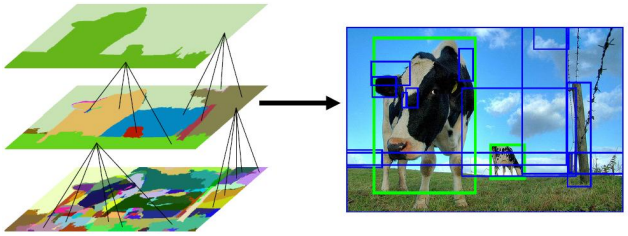


FIGURE 1.6 – Algorithme selective search

Les propositions d'objets provenant de la selective search sont extraites dans l'image et redimensionnées à taille fixe. Ces régions sont ensuite envoyées à un CNN pour déterminer leur classe.
R-CNN est très coûteuse en temps de calcul (pour l'apprentissage et pour le test) car chaque région passe dans toutes les couches du CNN.

Fast-RCNN : Puisque R-CNN est coûteuse en terme de temps de calcul Fast-RCNN [Gir15] a été introduit. Il permet d'extraire les régions provenant de la selective search sur une carte de caractéristiques profonde (feature map). En d'autres termes, l'image d'entrée entière est passée dans un réseau de neurones convolutif fournissant une carte de caractéristiques à basse résolution (du fait des Pooling successifs). Les propositions d'objets sont alors extraites sur cette carte et envoyées à un classifieur consistant généralement en deux couches cachées (complètement connectées) et une couche de sortie permettant la classification de l'objet (classe de l'objet ou fond). Dans ces deux approches, une fonction supplémentaire est apprise par le réseau permettant de transformer les propositions d'objets originelles produites par la selective search afin que celles-ci collent au mieux à l'objet. Cette fonction est appelée régression sur les boites.

Faster R-CNN : Il est possible de faire apprendre directement à un réseau les régions d'intérêt sur les images, si l'on dispose d'un corpus d'apprentissage avec les régions annotées. Des collections d'images proposent un grand nombre d'images avec des boîtes englobantes autour des objets. Ceci permet de construire un réseau qui va à la fois classifier image et proposer une région pour la localisation de l'objet.
L'approche Faster-RCNN [50] ajoute un Region Proposal Network après la dernière couche de convolution. Ce réseau est capable de regarder les caractéristiques de la dernière couche de convolution pour déterminer une région d'intérêt probable. Ensuite, la même méthode de classification que cette de Fast R-CNN.

1.3.2.8 Conclusion :

Dans ce chapitre, nous avons introduit les réseaux neuronaux de convolution profonds (CNN). Ces modèles permettent d'apprendre, grâce à l'apprentissage supervisé, les caractéristiques visuelles discriminantes à extraire sur les images afin de résoudre des problématiques de classification et/ou de détection.

Aujourd'hui, les CNN sont au centre de l'attention, en particulier en vision par ordinateur, de par les résultats impressionnants qu'ils produisent sur différentes tâches. Cependant, ils ne sont pas parfaits et leur étude au sein de la communauté est très active, notamment sur les problématiques de complexité calculatoire, d'optimisation pendant l'apprentissage et de choix des hyper-paramètres.

1.4 Position du problème

1.4.1 Motivations

Une grande partie des recherches récentes sur les réseaux de neurones convolutionnels (CNN) ont porté sur l'amélioration de la précision sur les taches de classification, segmentation et de détection. Pour un niveau de précision donné, il existe généralement de multiples architectures de CNN qui atteignent ce niveau de précision mais une architecture CNN avec moins de paramètres présente plusieurs avantages :

- Déploiement sur des systèmes embarqués : Afin de tirer profit des applications réussies du DL et de permettre à d'autre industries de s'améliorer, il est nécessaire de pouvoir déployer des DNN sur des systèmes embarqués entre autres pour des raisons économiques.
- Traitement local : parmi les raisons qui motivent le traitement local sont :
  — Fiabilité : Compter sur une connexion Internet n'est souvent pas une option viable.
  — Faible latence : De nombreuses applications nécessitent une réponse immédiate. Il se peut qu'une demande ne puisse pas tolérer le délai d'envoi des données à un autre endroit pour leur traitement.
  — Confidentialité : Les données peuvent être privées et ne doivent donc pas être transmises ou stockées à l'extérieur.
- Exportation rapide de nouveaux modèles vers les clients, par exemple Pour la conduite autonome, des entreprises comme Tesla copient périodiquement les nouveaux modèles de leurs serveurs vers les voitures des clients, avec AlexNet, cela nécessiterait 240MB de communication du serveur à la voiture ce qui n'est pas très efficace.
- Un apprentissage plus efficace, moins de paramètre implique un apprentissage plus rapide avec possibilité de distribution de calcule à cause du taux faible de communications.

Mais afin de pouvoir bénificier de tout ça, il est nécessaire de répondre à un nombre de défis et d'exigences qu'on discutera dans ce qui suit.

1.4.2 Défis

Il apparaît qu'une tendance générale dans la conception des réseaux de neurones était de trouver des modèles plus grands et plus profonds pour obtenir une meilleure précision sans tenir compte de la mémoire, temps de calcule ou de la consommation d'énergie. Une croyance largement répandue était que le nouveau matériel allait fournir une puissance de calcul et une mémoire suffisante pour permettre à ces réseaux de fonctionner avec des performances en temps réel dans les systèmes embarqués. Cependant, l'augmentation de

la vitesse des transistors due à l'amélioration des processus des semi-conducteurs a considérablement ralenti, et il semble peu probable que les processeurs mobiles répondent aux exigences de calcul avec un budget de puissance limité. Ceci a ouvert plusieurs nouvelles portes de recherche pour essayer de répondre à cette problématique.

1.5 Conclusion :

Le chapitre présenté a permis de dresser des généralités sur les réseaux de neurone et plus particulièrement les réseaux de neurones de convolution ainsi que leur rôle fondamental dans le vision par ordinateur.
La tendance générale a été de rendre les réseaux plus profonds et plus complexes afin d'obtenir une plus grande précision. Toutefois, ces progrès en matière de précision ne rendent pas nécessairement les réseaux plus efficaces en termes de taille et de vitesse. Dans de nombreuses applications du monde réel, telles que la robotique, les voitures autonomes et la réalité augmentée.
Dans le chapitre suivant on va introduire les différentes techniques qui permettront l'optimisation des CNN pour afin que les tâches de reconnaissance soit effectuées en temps utile sur des plateformes limitées par les calculs.

Chapitre 2

État de l'art

2.1 Introduction

Une nouvelle direction dans les travaux de recherche est apparue, qui vise à optimiser les CNNs afin de pouvoir les déployer sur des appareils limités en ressources et sur des systèmes embarqués. On peut distinguer dans la littérature deux grandes familles d'approches : **(i) :** La conception à partir de zéro d'architectures CNNs modernes moins complexes et peu profondes avec moins de paramètres. **(ii) :** La compression des architectures déjà existantes et déjà entraînée selon plusieurs techniques.
Dans ce chapitre, nous allons passer en revue les deux familles d'approches et présenter les différentes techniques et travaux dans chacune d'elles, nous commençons en section 2.2 par présenter les nouvelles architectures CNNs les plus pertinentes. Ensuite, en section 2.3 nous présentons les techniques de compression de CNN.

2.2 Vers de nouvelles architectures

2.2.1 Introduction

Dans ce qui suit nous allons présenter les différentes architectures CNNs qui ont été appliquées avec succès à des taches de détection et classification et déployées sur des systèmes embarqués. Ces architectures comptent un nombre très bas de paramètres par rapport à celles classiques, ce qui leurs permet d'être adaptés aux machines ayant une mémoire très limitées, en plus ça réduit le temps d'inférence et par conséquent la consommation d'énergie. Nous essayons de présenter une diversité d'architectures qui suivent différentes stratégies de conception.

2.2.2 SqueezeNet

L'une des architectures les plus réussies proposées dans cette direction c'est SqueezeNet qui est une petite architecture CNN. SqueezeNet qui atteint la même précision que celle d'AlexNet sur ImageNet avec 50x moins de paramètres. Dans cette section on va présenter les travaux de [Ian+17], mais avant ça on va reprendre deux termes définis dans l'article :

- **Microarchitecture CNN :** On utilise le terme de micro-architecture CNN pour désigner l'organisation et les dimensions internes particulières des différents modules de plus haut niveau qui composent le réseaux global.
  Exemple : les dimensions des filtres, le pas (Stride), ...
- **Macroarchitecture CNN :** nous définissons la macroarchitecture CNN comme l'organisation au niveau du système de différents modules dans une architecture CNN de bout en bout.
  Exemple : Le nombre de couches, la nature des connexion,...

2.2.2.1 Les strategies de conception :

Afin de pouvoir atteindre les objjectifs de SquuezeNet, ils adoptent trois stratégies :

- **Remplacer les filtres** 3×3 **par des filtres** 1×1 **:** Etant donné un certain nombre de filtres de convolution, on choisit de faire la majorité de ces filtres 1×1, puisqu'un filtre 1×1 a $9\times$ moins de paramètres qu'un filtre 3×3.
- **Diminuer le nombre de canaux d'entrée aux filtres** 3×3 **:** pour maintenir un petit nombre total de paramètres dans un CNN, il est important non seulement de diminuer le nombre de filtres 3×3 (voir Stratégie 1 ci-dessus), mais aussi de diminuer le nombre de canaux d'entrée vers les filtres 3×3. Nous diminuons le nombre de canaux d'entrée vers les filtres 3x3 en utilisant des couches de *"squeeze layers"*, que nous décrivons plus tard.
- **"Downsample" tard dans le réseau :** la réduction des "activation maps" se fait en utilisant soit des pas de convolution > 1 soit par l'opération *pooling*, il est donc claire que les la réduction retardée donne des *activation maps* plus grande ce qui peut conduire selon les travaux de [HS15] à une plus grande précision de classification, toutes choses étant égales par ailleurs.

2.2.2.2 Fire Module :

Est composé de : **(i)** une couche de convolution *squeeze* qui n'a que des filtres 1×1, alimentant **(ii)** une couche *expand* qui a un mélange de filtres de convolution 1×1 et 3×3 ; Ceci est illustré dans la figure 2.1
L'utilisation des filtres 1×1 dans les modules Fire est une application de la stratégie 1. Ce module a trois paramètres ajustables (hyperparamètres) qui sont :

— $s_{1\times1}$: est le nombre de filtres 1×1 dans la couche *squeeze*.
— $e_{1\times1}$: est le nombre de filtres 1×1 dans la couche *expand*.
— $e_{3\times3}$: est le nombre de filtres 3×3 dans la couche *expand*.

Lorsqu'on utilise des modules *Fire*, nous définissons $s_{1\times1}$ comme étant inférieur à $(e_{1\times1} + e_{3\times3}$, de sorte que la couche de *squeeze* permet de limiter le nombre de canaux d'entrée vers les filtres 3×3, conformément à la stratégie 2.

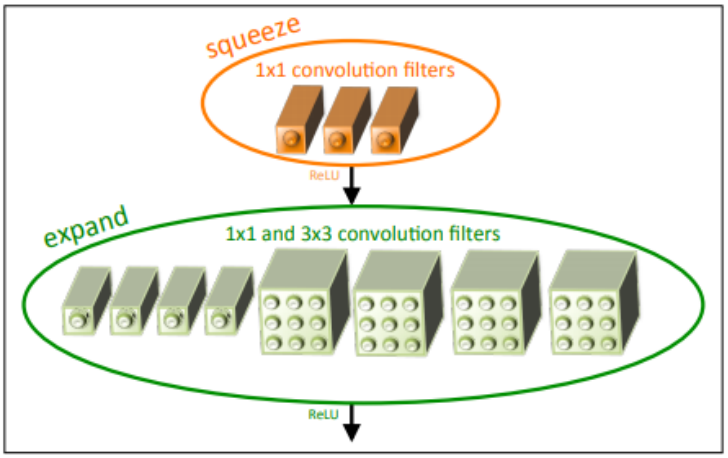


FIGURE 2.1 – Microarchitecture du *Fire Module* [Ian+17]
Sur cet example on a $s_{1\times1} = 3, e_{1\times1} = 4$ et $e_{3\times3} = 4$.

2.2.2.3 Architecture SqueezeNet :

Quant à la macroarchitecture. Le tableau 2.1 montre que SqueezeNet commence par une couche convolutionnelle (conv1), suivie de 8 modules Fire (fire2-9), pour finir par une couche convolutionnelle finale (conv10). On augmente progressivement le nombre de filtres par module *Fire* du début à la fin du réseau. SqueezeNet effectue le max-pooling avec un pas de 2 après les couches conv1, fire4, fire8, et conv10 ; ces placements relativement tardifs du pooling sont conformes à la stratégie 3. Nous présentons l'architecture complète de SqueezeNet dans le tableau 2.1.

Layer name	**Output size**	**filter size / stride (if not a fire layer)**	$\mathbf{s_{1\times1}}$	$\mathbf{e_{1\times1}}$	$\mathbf{e_{3\times3}}$
input image	224x224x3				
conv1	111x111x96	7x7/2 (x96)			
maxpool1	55x55x96	3x3/2			
fire2	55x55x128		16	64	64
fire3	55x55x128		16	64	64
fire4	55x55x256		32	128	128
maxpool4	27x27x256	3x3/2			
fire5	27x27x256		32	128	128
fire6	27x27x384		48	192	192
fire7	27x27x384		48	192	192
fire8	27x27x512		64	256	256
maxpool8	13x12x512	3x3/2			
fire9	13x13x512		64	256	256
conv10	13x13x1000	1x1/1 (x1000)			
avgpool10	1x1x1000	13x13/1			

TABLE 2.1 – Macroarchitecture complète de SqueezeNet [Ian+17]

2.2.2.4 Evaluation :

Afin de d'évaluer SqueezeNet, [Ian+17] ont utilisé AlexNet comme architecture de réference. Les deux modèles sont entraînés sur ImageNet. Les résultats de la comparaison sont résumés dans le tableau 2.2.

CNN architecture	**Original**	**Reduction in Model Size vs. AlexNet**	**Top-1 ImageNet Accuracy**	**Top-5 ImageNet Accuracy**
AlexNet	240 MB	1x	57.2%	80.3%
SqueezeNet	4.8 MB	50x	57.5%	80.3%

TABLE 2.2 – Comparaison entre SqueezeNet et AlexNet.

2.2.3 SqueezeNext

Dans cette section, nous présentons SqueezeNext [Gho+18], une nouvelle famille d'architectures de réseaux neuronaux dont la conception a été guidée par des architectures précédentes telles que SqueezeNet, ainsi que par les résultats de simulation sur un accélérateur de réseaux de neurones. Ce nouveau réseau est capable de reproduire la précision d'AlexNet sur le benchmark ImageNet avec 112× moins de paramètres

2.2.3.1 Les stratégies de conception

SqueezeNext utilise l'architecture SqueezeNet comme architecture de base, cependant on apporte les modifications suivantes.

- On utilise une réduction plus agressive des canaux en incorporant un module Squeeze à deux niveaux. Cela réduit considérablement le nombre total de paramètres utilisés avec les convolutions 3×3.
- On utilise des convolutions 3×3 séparables pour réduire davantage la taille du modèle, et on supprime la branche *expand* 1×1 supplémentaire après le module de *squeeze*
- On utilise une connexion de saut d'addition (*addition skip connection*) similaire à celle de l'architecture ResNet [He+16], ce qui nous permet d'entraîner un réseau beaucoup plus profond sans le problème de disparition du gradient (*gradiet vanishing problem*[1]).
- On optimise l'architecture SqueezeNext de base en simulant ses performances sur un système embarqué multiprocesseur. Les résultats de la simulation donnent un aperçu très intéressant de la localisation des goulots d'étranglement en matière de performances. Sur la base de ces observations, [Gho+18] peuvent alors effectuer

1. Gradiet vanishing problem : le gradient sera très petit, ce qui empêchera le poids de changer de valeur et donc relentit le processus d'apprentissage.

des variations sur le modèle de base et obtenir des performances plus élevées tant en termes de vitesse d'inférence que de consommation d'énergie, et parfois même une meilleure précision de classification.

2.2.3.2 SqueezeNext Block

[Gho+18] utilisent une variante de l'idée utilisée dans l'architecture SqueezeNet [Ian+17], où les auteurs ont utilisé une couche *Squeeze* avant la convolution 3×3 pour réduire le nombre de canaux d'entrée vers celle-ci. Et ce en utilisant une couche de *Squeeze* deux fois de suite (on appelle chacune *Bottleneck Module*), comme le montre la figure 2.3, réduisant chacune la taille du canal par un facteur de 2.
Qui est suivi de deux convolutions séparables, i.e. décomposer les convolutions $K \times K$ en deux convolutions séparables de taille $1 \times K$ et $K \times 1$ figure 2.2 (en orange), cela réduit effectivement le nombre de paramètres de K^2 à $2K$.
Ils incorporent également un module *expand* final de 1×1, qui réduit encore le nombre de canaux de sortie pour les convolutions séparables. En plus la figure 2.2 montre la connexion de saut d'addition déjà mentionnée.

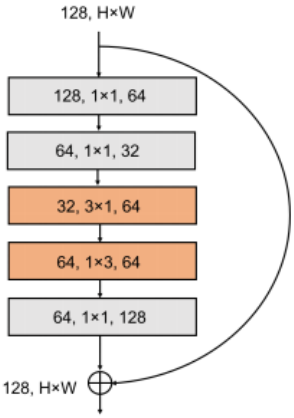


FIGURE 2.2 – Illustration d'un bloc SqueezeNext avec saut d'addition [Gho+18]

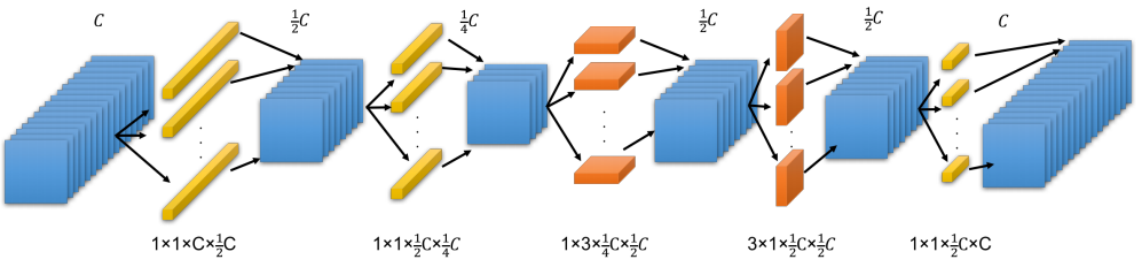


FIGURE 2.3 – Illustration d'un bloc SqueezeNext.[Gho+18]

2.2.3.3 Architecture SqueezeNext

Il s'agit d'arranger les blocs *SqueezeNext*. Nous présentons une version de cette famille de réseaux appelée 1.0-SqNxt-23 à travers un exemple (Figure 2.4).

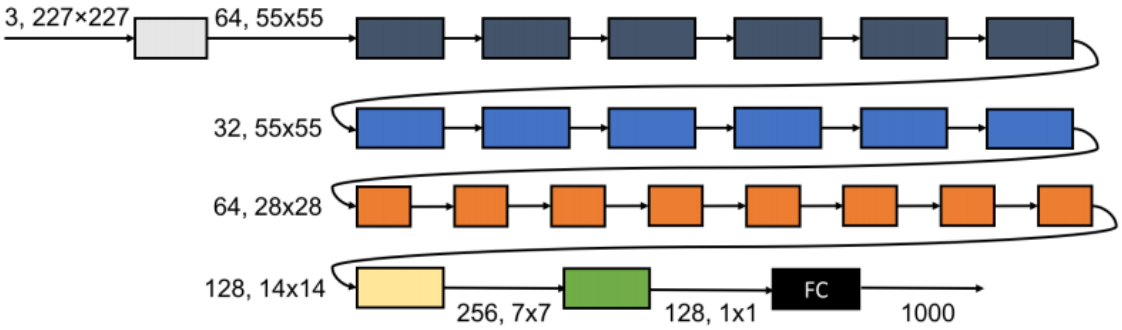


FIGURE 2.4 – Illustration de la disposition des blocs dans 1.0-SqNxt-23 [Gho+18]

Chaque changement de couleur correspond à une modification de la résolution du *Feature map* d'entrée. Le nombre de blocs après la première couche de convolution/pooling est Depth [2]= [6, 6, 8, 1], où le dernier nombre fait référence à la case jaune. Ce bloc est suivi d'un module de *bottleneck* avec avgpooling pour réduire la taille du canal et la résolution spatiale (boîte verte), suivi d'une couche entièrement connectée (boîte noire). Dans les variations optimisées de cette version, on modifie cette distribution de profondeur en diminuant le nombre de blocs dans les premiers stades (bleu foncé), et nous attribuons plutôt plus de blocs aux stades ultérieurs par exemple *1.0-SqNxt-23v5* avec Depth=[2,4,14,1].

2.2.3.4 Evaluation

La table 2.3 montre les résultats de simulation sur un accélérateur Hardware des différents modèles entraînés sur le dataset ImageNet. Le temps pour chaque configuration est normalisé par le nombre de cycles du réseau le plus rapide pour chaque configuration (plus petit, c'est mieux). Notez comment les variations du modèle SqueezeNext de base permettent d'obtenir une meilleure inférence et une meilleure consommation d'énergie tout en gardant des performance de classification meilleures par rapport à tout autre modèle. De plus, le modèle 1.0-SqNxt-23v5 est 17% plus rapide et 23% plus efficace sur le plan énergétique que le modèle 1.0-SqNxt-23 (de base) pour une configuration 16×16 (accélérateur hardware [Gho+18]). Ceci est obtenu par une redistribution efficace de la profondeur à chaque étape comme mentionné dans 2.2.3.3.

2. Depth : Indique la disposition des blocs

Modèle	**Paramètres (Millions)**	**Top-1 (%)**	**Top-5 (%)**	**Depth**	**rapport du temps d'inférence**	**énergie**
AlexNet	60.9	57.10	80.3		x5.46	1.6E+10
SqueezeNet	1.2	57.50	80.30		x3.42	6.7E+09
1.0-SqNxt-23	0.72	59.05	82.60	[6,6,8,1]	x1.17	3.2E+09
1.0-SqNxt-23v3	0.74	58.18	81.96	[4,8,8,1]	**x1.00**	2.7E+09
1.0-SqNxt-23v5	0.94	59.24	82.41	[2,4,14,1]	x1.00	2.6E+09

TABLE 2.3 – Résultat de simulation sur un accélérateur Hardware. [Gho+18]

2.2.4 MobileNet

Il existe deux version de l'architecture "MobileNets" version 1 [G+17] et version 2 [M+]. Ces réseaux ont été crées pour une utilisation embarquée sur smartphones. Entraînés à l'origine à la reconnaissance d'objets très divers, à travers 1000 classes et à partir de millions d'images pour le challenge ImageNet, leurs couches cachées sont capables de détecter des caractéristiques très variées et donc ces réseaux sont facilement réutilisables pour des tâches très différentes.

2.2.4.1 stratégies de conception

La particularité de MobileNet est de remplacer les convolutions conventionnelles par de nouvelles appelées "Depthwise Separable Convolutions". Une convolution standard filtre les données d'entrées via une fenêtre glissante sur tous les canaux simultanément et combine les résultats en une nouvelle série de données. Avec un but identique, les convolutions "depthwise separable" divisent ces étapes en deux couches distinctes : une convolution appelée "depthwise" suivie d'une convolution 1x1 appelée "pointwise". Cette factorisation permet de grandement réduire le nombre de calculs et la taille du modèle. Illustrée par la figure 2.5(a), la convolution "depthwise" applique un filtre sur chaque canal de l'image d'entrée. La convolution "pointwise", figure 2.5(b), applique ensuite une simple convolution 1x1 afin de créer une combinaison linéaire de l'entrée.

Une comparaison directe entre les principes des convolutions classiques et des convolutions "depthwise separable" est illustrée figure 2.6, où "M" correspond au nombre de canaux d'entrée ("input depth", 3 pour une image couleur, dépend de la couche précédente pour une carte de caractéristiques "feature map"), N au nombre de filtres (et donc nombre de canaux de sortie "ouput depth") et D_k la taille du noyau de convolution. En considérant un pas de un pixel, le nombre de multiplications d'une convolution "depthwise separable" est définit par $[D_F * D_F * M * (N + D_K * D_K)]$, et $[D_F * D_F * M * N * D_K * D_K]$ définit

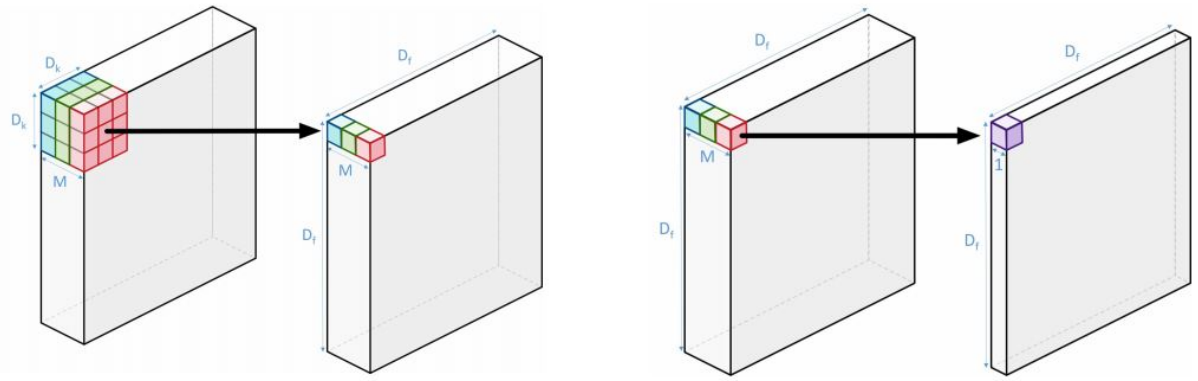


FIGURE 2.5 – MobileNet separable convolution $D_k = 3, M = 3, N = 1$

le nombre de multiplications pour une convolution standard, où D_F correspond à la taille de l'image traitée. On observe bien que les convolutions "depthwise separable" sont moins coûteuses en ressources. Les réseaux MobileNet utilisent des convolutions "depthwise" de taille $D_K = 3$. Contrairement aux réseaux conventionnels, les réseaux MobileNet n'in-

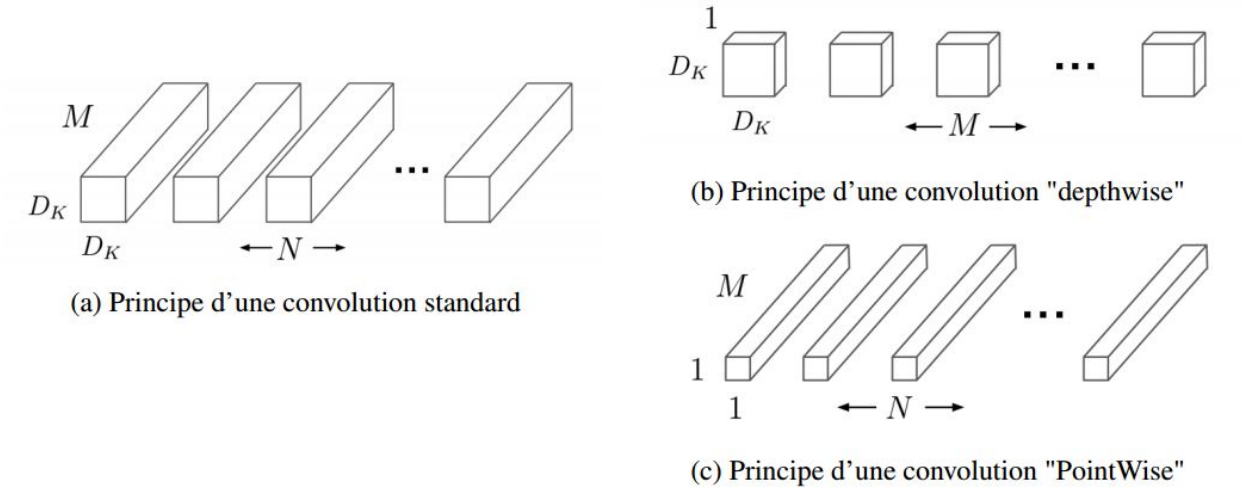


FIGURE 2.6 – Comparaison des principes des différentes convolutions

tègrent pas de couches de "pooling" entre ces blocs de convolutions. Ils utilisent également pour chacune de ces couches de convolutions une normalisation ainsi qu'une fonction d'activation de type ReLU. La version 2 de MobileNet intègre quelques différences dans son bloc de convolution "depthwise separable", illustrées figure 2.7, et désormais appelé bloc de "bottleneck" résiduel ("residual bottleneck bloc"). Le bloc de la version 2 contient une nouvelle première couche. Appelée couche d'expansion, c'est également une convolution 1x1 mais son objectif est d'étendre le nombre de canaux avant l'entrée de la convolution

"depthwise". Cette augmentation est gérée par un hyperparamètre appelé le facteur d'extension. Afin d'effectuer une somme entre l'entrée et la sortie de ce bloc via la connexion résiduelle, une dernière couche 1x1, appelée couche de projection, va réduire le nombre de canaux en sortie de la couche de convolution "depthwise", permettant de retrouver le même nombre de canaux qu'en entrée avant l'extension.

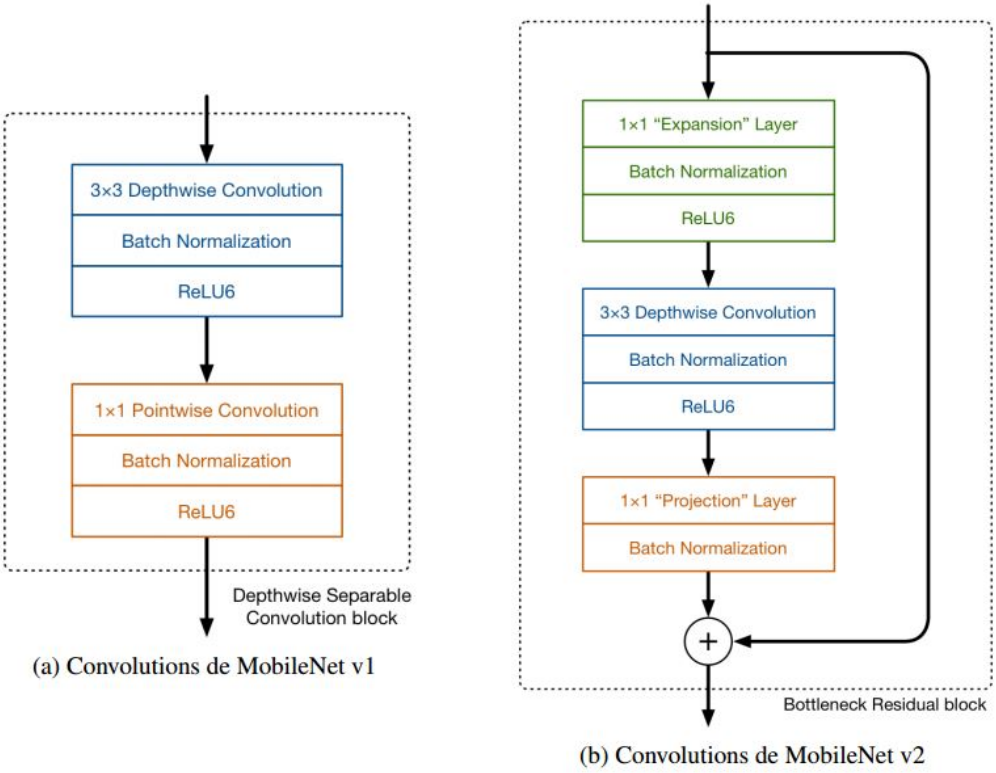


FIGURE 2.7 – Structure des convolutions des réseaux MobileNet v1 et v2

2.2.4.2 architecture MobileNet

L'architecture du réseau MobileNet v1 est présentée par le tableau 2.4. En comptant les convolutions depthwise et pointwise comme des couches distinctes, ce réseau dispose de 28 couches cachées. En comparaison, l'architecture du réseau MobileNet v2, illustrée dans le tableau 2.5, est initialisée par une convolution complète, suivie par 19 couches de bottlenecks résiduels (remplacent du bloc de convolution depthwise separable de MobileNet v1). Dans ce tableau, le paramètre "t" représente le facteur d'extension de la première couche du bloc résiduel. Le paramètre "c" détermine le nombre de canaux de sortie de chaque séquence. Chaque couche présentée est répétée un certain nombre de fois déterminé par le paramètre "n". Le paramètre " s" détermine le "stride", c'est à dire le pas avec lequel le noyau de convolution se déplace sur l'image.

Layer name	**Filtre**	**Entrée**
conv / s2	3x3x3x32	224^2x3
Conv dw / s1	3x3x32 dw	112^2x32
Conv / s1	1x1x32x64	112^2x32
Conv dw / s2	3x3x64 dw	112^2x64
Conv / s1	1x1x64x128	56^2x64
Conv dw / s1	3x3x128 dw	56^2x128
Conv / s1	1x1x128x128	56^2x128
Conv dw / s2	3x3x128 dw	56^2x128
Conv / s1	1x1x128x256	28^2x128
Conv dw / s2	3x3x256 dw	28^2x256
Conv / s1	1x1x256x256	28^2x256
Conv dw / s2	3x3x256 dw	28^2x256
Conv / s1	1x1x256x512	14^2x256
conv dw / s1 5x conv / s1	3x3x512 dw 1x1x512x512	14^2x256 14^2x256
Conv dw / s2	3x3x512 dw	14^2x512
Conv / s1	1x1x512x1024	7^2x512
Conv dw / s2	3x3x1024 dw	7^2x1024
Conv / s1	1x1x1024x1024	7^2x1024
Avg Pool / s1	Pool 7x7	7^2x1024
FC / s1	1024x1000	1^2x1024
Softmax / s1	Classifieur	1^2x1000

TABLE 2.4 – Architecture des couches de MobileNet v1 [G+17]

Layer Type	**Entrée**	**t**	**c**	**n**	**s**
Conv2D	224^2x3	-	32	1	2
Bottleneck	112^2x32	1	16	1	1
Bottleneck	112^2x16	6	24	2	2
Bottleneck	56^2x24	6	32	3	2
Bottleneck	28^2x32	6	64	4	2
Bottleneck	14^2x64	6	96	3	1
Bottleneck	14^2x96	6	160	3	2
Bottleneck	7^2x160	6	320	1	1
conv2D 1x1	7^2x320	-	1280	1	1
avgpool 7x7	7^2x1280	-	-	1	-
conv2D 1x1	1x1x1280	-	k	-	-

TABLE 2.5 – Architecture des couches de MobileNet v2 [M+]

2.2.4.3 Évaluation

Bien que les deux versions de base de MobileNet soient déjà très légères, l'utilisation grandissante de l'intelligence artificielle embarquée, comme dans nos smartphones, requiert des modèles toujours plus légers et rapides. Afin de construire à partir de Mobile-

Réseaux	α	dimension image définie par ρ	Nombre de Paramètres	Accuracy [%]	Taille [Mo]
MobileNet_v1	0.25	128	$0{,}47.10^2$	41.5	2.00
MobileNet_v1	0.25	160	$0{,}47.10^2$	45.5	2.00
MobileNet_v1	0.25	192	$0{,}47.10^2$	47.7	2.00
MobileNet_v1	0.25	224	$0{,}47.10^2$	49.8	2.00
MobileNet_v1	0.5	128	$1{,}34.10^2$	56.3	5.20
MobileNet_v1	1.0	224	$4{,}24.10^2$	70.9	16.7
MobileNet_v2	0.35	96	$1{,}66.10^2$	45.5	6.60
MobileNet_v2	0.35	128	$1{,}66.10^2$	50.8	6.60
MobileNet_v2	0.35	160	$1{,}66.10^2$	55.7	6.60
MobileNet_v2	0.35	192	$1{,}66.10^2$	58.2	6.60
MobileNet_v2	0.35	224	$1{,}66.10^2$	60.3	6.60
MobileNet_v2	0.5	96	$1{,}96.10^2$	57.7	7.75
MobileNet_v2	1.0	224	$3{,}47.10^2$	71.8	13.76

TABLE 2.6 – Caractéristiques des réseaux MobileNet [19a] v1 et v2 [19b] et leurs performances originales en terme de accuracy de reconnaissance d'objet sur les 1000 classes du challenge ImageNet

Net ces réseaux plus légers et moins gourmands en puissance de calcul, deux paramètres, α et ρ, ont été introduits [G+17]. Appelé "facteur de largeur" ("width multiplieur"), le paramètre α permet d'affiner uniformément le réseau à chaque couche, ce qui réduit les calculs nécessaires et le nombre de paramètres par approximativement 2. Pour une couche et un paramètre α donnés, le nombre de canaux d'entrée "M" devient "$\alpha\times$M" et le nombre de canaux de sortie "N" (nombre de filtres) devient "$\alpha\times$N", avec $\alpha \in]0\,;\,1]$. Différentes valeurs de ce paramètres ont été définies : $\{1\,;\,0{,}75\,;\,0{,}5\,;\,0{,}35\,;0{,}25\}$. Le réseau original correspond donc à $\alpha = 1$ et les réseaux réduits à $\alpha < 1$.
Le second paramètre, appelé facteur de résolution ("resolution multiplier"), permet également de réguler le nombre de calculs nécessaires. Appliqué aux images d'entrée, la constitution interne de chaque couche est ensuite réduite par ce même facteur $\rho(\in]0\,;\,1])$, ce qui a pour effet de réduire les besoins en calculs d'environ ρ^2 . Le coût calculatoire d'une convolution depthwise separable (pour MobileNet v1), prenant en compte les paramètres α et ρ est alors de $[\rho^2 \times D_F^2 \times \alpha \times M \times (\alpha \times N + D_K^2)]$. Différentes valeurs de ρ ont été définies de manière à implicitement correspondre à certaines dimensions d'images d'entrées : $\{224\,;\,192\,;\,160\,;\,128\,;\,96\,\}$. Le réseau original correspond donc à $\rho = 1$, et les réseaux réduits à $\rho < 1$. Le tableau 2.6 récapitule brièvement les caractéristiques de ces réseaux. On y retrouve les deux versions de MobileNet ainsi que leurs différentes déclinaisons en réseaux réduits de par les variations des paramètres α et ρ. Permettant avant tout de moduler la vitesse et l'encombrement au niveau stockage selon les besoins, ces paramètres ont évidemment un impact sur les performances prédictives des réseaux.

2.2.5 ShuffleNet :

ShuffleNet [Zha+18b] est une architecture correspondant à un CNN extrêmement efficace en termes de calcul, conçue spécialement pour les appareils mobiles avec une puissance de calcul très limitée. Cette architecture est basé sur deux opérations principale, la convolution de groupées pointwise et le mélange de canaux "Shuffle Channels", pour réduire considérablement les coûts de calcul tout en maintenant précision.

2.2.5.1 Convolutions groupées :

Les convolutions groupées [Ioa+17] ont d'abord été introduites pour permettre de distribuer les calculs pendant l'apprentissage du réseau. On sépare les canaux en un certain nombre g de groupes, et les convolutions ne vont être utilisées que sur un groupe particulier. Si l'on a C canaux d'entrée avec g groupes, on a une entrée qui va être composée de groupes de taille $\frac{C}{g}$, où chaque convolution ne va être appliquée que sur un seul de ces groupes. Comme illustré sur la figure 2.8, avec un nombre de groupes égal à deux, on divise les convolutions en deux blocs, celle du bloc 1 n'étant effectuée que sur les canaux numérotés $[1; \frac{C}{g}]$ n'ayant pas accès aux canaux $]\frac{C}{g}; C]$

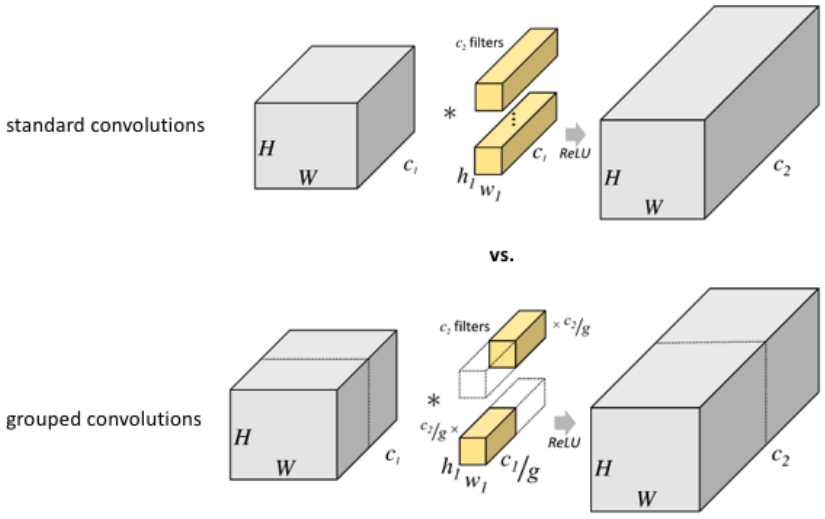


FIGURE 2.8 – Illustration des convolutions groupées [Ioa+17]

2.2.5.2 Mélange des canaux pour la convolution regroupée :

Les convolutions groupées peuvent être apprises séparément (notamment sur plusieurs GPUs) et ça présente l'avantage de réduire le nombre de paramètres, mais en contrepartie on perd toute forme d'information entre les groupes, puisque chaque canal de sortie ne se relie qu'aux canaux d'entrée au sein du groupe. Cette propriété bloque le flux d'informations entre les groupes de canaux et affaiblit la représentation, la figure 2.9(a) illustre

une situation de deux couches de convolution groupée empilées. Il est clair qu'aucune information est partagé entre les groupes.
ShuffleNet traite ce problème en permettant à la convolution groupée d'obtenir des données d'entrée de différents groupes, comme montré dans la figure 2.9(b), et ainsi les canaux d'entrée et de sortie seront entièrement liés.Plus précisément, pour la carte des caractéristiques d'entités générée à partir de la couche de groupe précédente, nous pouvons d'abord diviser les canaux de chaque groupe en plusieurs sous-groupes, puis alimenter chaque groupe de la couche suivante avec des sous-groupes différents, comme illustré dans la figure 2.9(c).

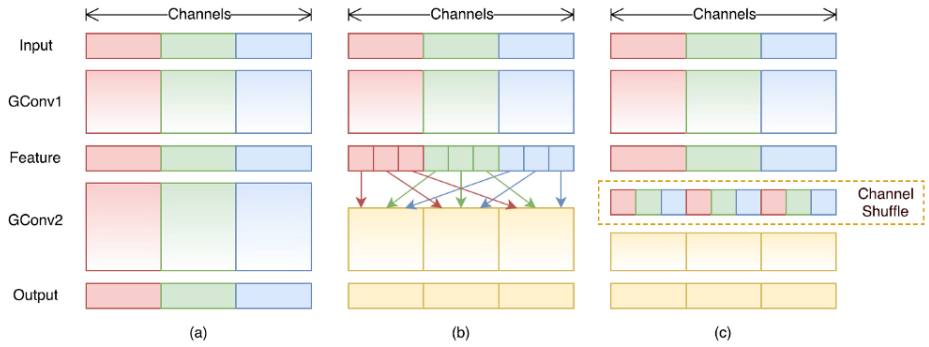


FIGURE 2.9 – Mélange des canaux avec deux couches de convolution groupée [Zha+18b]

2.2.5.3 Bloc ShuffleNet :

ShuffleNet utilise un type de bloc proche de celui de Mobile, mais ajoute une convolution 1×1 groupée pour réduire le nombre de canaux en entrée de la convolution 3×3. La convolution 1×1 de fusion des informations est remplacée par une convolution 1×1 groupée (figure 2.10). Mais pour que celle-ci ait quand même un rôle de fusion d'information, une opération de mélange des canaux est réalisée entre les deux convolutions, en plus d'ajouter une skip-connection. Cette approche permet d'obtenir des résultats légèrement supérieurs à MobileNet, mais avec environ $4,5$ fois moins de paramètres grâce aux convolutions groupées.

2.2.5.4 Archtecture ShuffleNet :

ShuffleNet est principalement composé d'une pile de bloc ShuffleNet regroupées en trois Stages. Dans un bloc ShuffleNet le nombre de groupe g contrôle la densité de convolution Pontwise. Le Tableaux 2.7 exploite l'architecture avec diffèrent nombre de groupe, des nombres de groupe plus importants entraînent plus de canaux de sortie (donc plus de filtres convolutionnels).

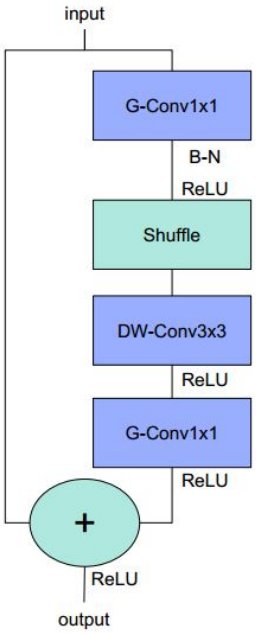


FIGURE 2.10 – Unité ShuffleNet [Zha+18b]

Layer	Taille de Sortie	Taille Filtre	Pas	Répéter	Canaux de Sortie (g groups) $g=1$	$g=2$	$g=3$	$g=4$	$g=8$
Image	224×224				3	3	3	3	3
Conv	112×112	3×3	2	1	24	24	24	24	24
MaxPool	56×56	3×3	2						
Stage2	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	544	768
	14×14		1	7	288	400	480	544	768
Stage4	7×7		2	1	576	800	960	1088	1536
	7×7		1	3	576	800	960	1088	1536
GlobalPool	1×1	7×7							
FC					1000	1000	1000	1000	1000

TABLE 2.7 – Architecture des couches de ShuffleNet [Zha+18b]

2.2.5.5 Évaluation :

Afin d'evaluer les performance de ShuffleNet, on va comparer cette architecture avec MobileNet (figure 2.8) sur deux facteurs principales, l'erreur de classification et la complexité, cette dernière sera mesurer par FLOPS (Nombre d'opérations en virgule flottante par seconde). L'ensemble de validation du compétition ImageNet 2012 été utilisé [How+17].

Pour personnaliser le réseau selon la complexité souhaitée, nous pouvons simplement appliquer un facteur d'échelle s au nombre de canaux. Par exemple, nous désignons les réseaux dans la table 2.8 comme «ShuffleNet 1×», puis «ShuffleNet s×» signifie une mise

à l'échelle du nombre de filtres dans ShuffleNet 1× par s fois, ainsi la complexité globale sera à peu près s^2 fois de ShuffleNet 1× [Zha+18b].

Model	Complexité ($10^6 \times FLOPS$)	Erreur Classification (%)	△ Erreur (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet 2 × ($g = 3$)	524	26.3	3.1
0.75 MobileNet-224	325	31.6	-
ShuffleNet 1.5 × ($g = 3$)	292	28.5	3.1
0.5 MobileNet-224	149	36.3	-
ShuffleNet 1 × ($g = 8$)	140	32.4	3.9
0.25 MobileNet-224	41	49.4	-
ShuffleNet 0.5 × ($g = 4$)	38	41.6	7.8

TABLE 2.8 – ShuffleNet vs. MobileNet [How+17] sur la classification ImageNet

2.2.6 SqueezeDet

La détection d'objet est la principale tâche dans un système intelligent de supervision d'un parking ou dans le pilotage des véhicules autonomes etc... Dans cette section, on présente SqueezeDet [Wu+17], un réseau de neurones entièrement convolutif pour la détection d'objets. Dans ce réseau, on utilise des couches convolutionnelles non seulement pour extraire des *feature maps*, mais aussi comme couche de sortie pour calculer les rectangles de délimitation et les probabilités de classe. Le pipeline de détection de ce modèle ne contient qu'un seul passage direct (*forward pass*) d'un réseau de neurones, donc il est extrêmement rapide. Le modèle est entièrement convolutif, ce qui permet d'obtenir un modèle de petite taille et une meilleure efficacité énergétique, et donc de déployer des systèmes embarqués.

2.2.6.1 Conception de l'architecture

On adopte un pipeline de détection à une seule passe : la proposition et la classification des régions sont effectuées par un seul réseau simultanément. Comme le montre la figure 2.11 un réseau de neurones convolutionnels prend d'abord une image en entrée et en extrait une *feature map* à basse résolution et à haute dimension. Ensuite, la carte de caractéristiques est introduite dans la couche ConvDet pour calculer des rectangles de délimitation (*anchors*) centrés sur W×H grilles spatiales uniformément distribuées. W et H sont le nombre de centres de grille horizontales et verticales.

Choix du CNN de base : Plusieurs architectures CNN sont possibles pour l'extraction de caractéristiques VGG16, ResNet ...etc, Mais l'objectif étant de minimiser la taille, accé-

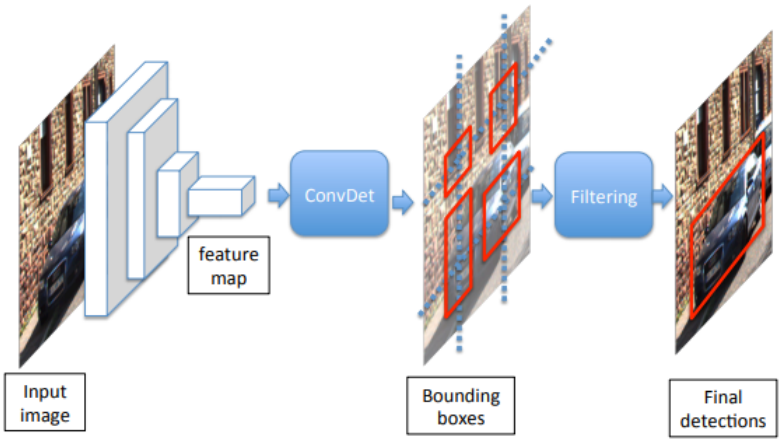


FIGURE 2.11 – Illutration du pipeline de detection de SqueezeDet [Wu+17]

lérer les inférence et minimiser la consommation d'énergie,[Wu+17] ont choisi SqueezeNet comme architecture de base.

ConvDet : ConvDet est essentiellement une couche convolutionnelle qui est entraînée pour calculer les coordonnées des rectangles de délimitation et les probabilités des classes. elle fonctionne comme une fenêtre glissante qui se déplace à travers chaque position spatiale sur la carte de caractéristiques. À chaque position, elle calcule $K \times (4+1+C)$ valeurs, ou **K** est le nombre de rectangles avec des formes prédéterminées appelés ancres (*anchors*). Chaque position sur la carte correspond à un centre de grille dans l'image originale, donc chaque ancre peut être décrite par **4** scalaires $(x_i, y_j, w_k, h_k)i \in [1, W], j \in [1, H], k \in [1, K]$. Ici $x_i et y_i$ sont les coordonnées du centre. w_k, h_k sont la largeur et la hauteur de la k-ième ancre.**C** est le nombre de classe (C probabilités de classes sont calculées), le **1** correspond au score de confience (confidence score) qui indique la probabilité d'existence d'objet sur une ancre.

2.2.7 Conclusion :

Dans cette section on a introduit des architectures CNN optimales qui ont été conçues afin de les déployer sur des systèmes avec des ressources limitées, SqueezeNet qui réduit le nombre de paramètres et de calculs tout en maintenant les performances de la classification, MobileNet utilise des convolutions séparables en profondeur pour construire des réseaux neuronaux légers et profonds et ShuffleNet propose d'utiliser des convolutions de groupe et de mélanger les canaux afin de réduire les coûts de calcul.
En terme de précision les architectures plus profondes (ShuffleNet, MobileNet V2) donnent de meilleurs résultats que les architectures moins profondes (SqueezeNet, MobileNet V1). Par conséquent, les CNNs optimal ne devraient pas être conçus avec une faible profondeur afin de ne pas affecter la performance.

Même si tous les modèles présentés fonctionne en temps réel, il apparaît que SqueezeNet a la meilleure performance d'exécution grâce aux bibliothèques qui sont spécifiquement optimisée pour les convolutions standard,
La complexité chaque modèle est ajustable grâce avec des hyper-paramètres qui lui sont spécifique ce qui permet de trouver le bon compromis entre la précision et le nombre de paramètres.

2.3 Compression des CNN

2.3.1 Introduction

La plupart des applications réussies des réseaux de neurones à des problèmes du monde réel ont été réalisées en utilisant des réseaux de neurones hautement structurés et de taille assez importante. Vu que les applications sont de plus en plus complexes, les réseaux seront probablement encore plus grands et plus structurés, d'autres techniques seront alors nécessaires pour minimiser la taille du réseau même après son entraînement.
Dans ce qui suit on présente les différentes techniques de compression des CNNs, en mettant l'accent sur la technique d'élagage qui s'avérait être très prometteuse en terme de résultat et en terme de quality du réseau (par exemple la précision sur une tache de classification)

2.3.2 Elagage

L'élagage des réseaux neuronaux est une vieille idée qui remonte aux années 1990 avec les travaux de Yann Lecun [LeC+89] sur les dommages optimaux au cerveau et avant. L'idée est que parmi les nombreux paramètres du réseau, certains sont redondants et ne contribuent pas beaucoup à la sortie. Plus important encore, à mesure que le nombre de paramètres dans le système augmente, des problèmes de surapprentissage peuvent survenir avec des effets dévastateurs sur la performance de la généralisation. Nous présentons de nouvelles techniques utilisées pour réduire la taille d'un réseau d'apprentissage en supprimant sélectivement les poids.

2.3.2.1 position du problème

Considérons un ensemble d'entrainement $D = \{X = \{x_0, x_1...x_n\}, Y = \{y_0, y_1...y_n\}\}$, ou x et y représentent respectivement une entrée et une sortie. Les paramètres du modèle $W = \{(w_1^1, b_1^1), (w_1^2, w_1^2)...(w_l^{cl}, b_l^{cl})\}$ ou l représentent le nombre de couches et Cl le nombre de filtres dans la couche l. Ces paramètres sont optimisés pour minimiser une fonction de cout qu'on note $C(^D/_W)$ qui indépendante de l'opération de l'élagage. Trouver un bon sous-ensemble de paramètres W' tout en maintenant une valeur de coût aussi proche que

possible de l'original est un problème combinatoire formulé comme suit :

$$\min_{W'}(C(D/W') - C(D/W))$$

Pour les réseaux actuels, il serait impossible de trouver ce sous ensemble en d'autres termes la solution optimale ne sera pas atteinte en un temps raisonnable, du coup nous allons présenter dans ce qui suit une classe de méthodes gourmandes.

2.3.2.2 Pruning filters for effecient convnets

Dans ce qui suit on présente les travaux de [Li+17] qui se sont intéressé à élaguer des filtres des couches convolutives et non pas ceux de la couche totalement connectée (fully connected).On va garder les notations du chapitre 1.
Ils ont remarqué que le nombre d'opérations dans une couche convolutive est $n_i \times n_{i+1} \times k^2 \times h_{i+1} \times w_{i+1}$ comme le montre la figure 2.12, quand un filtre F_{ij} est élagué sa feature map $x_{i+1,j}$ correspondante est supprimée ce qui réduit $n_i \times k^2 \times h_{i+1} \times w_{i+1}$ opérations, de plus les noyaux de la couche suivante appliqués au feature map supprimée sont à leurs tour enlevés ce qui réduit $n_{i+2} \times k^2 \times h_{i+2} \times w_{i+2}$ opérations. En total élaguer m filtre de la couche i réduit m/n_{i+1} du pourcentage du cout de calcul pour les couches i et i+1.

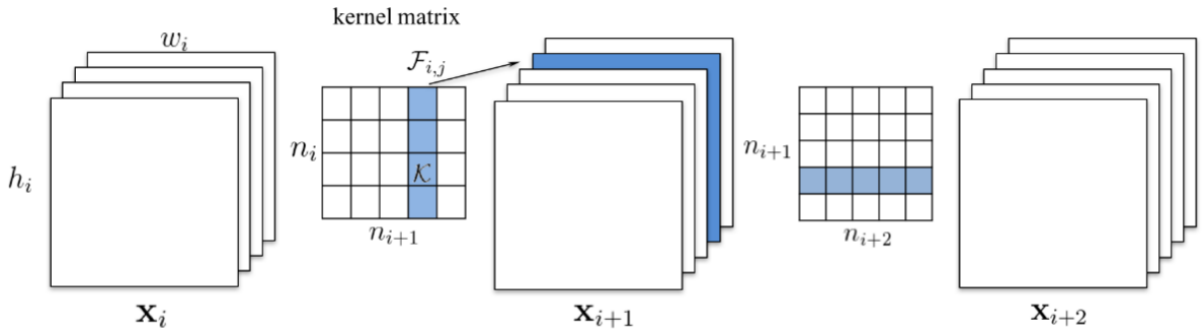


FIGURE 2.12 – Example d'élagage [Li+17]

Leur méthode élague les filtres les moins utiles d'un modèle déjà entrainé, ils mesurent l'importance relative d'un filtre dans chaque couche en calculant la somme absolue de ses composantes en d'autres termes calculer la norme l_1 du filtre $F_{i,j}$. Puisque les canaux d'entrée sont les mêmes pour tous les filtres, $\sum |F_{i,j}|$ représente également la magnitude moyenne de ses poids de noyaux. Cette valeur donne une espérance de l'ampleur de la feature map de sortie. Ils ont constaté que l'élagage des filtres les plus petits fonctionne mieux que l'élagage du même nombre de filtres aléatoires ou de filtres les plus grands. Comparativement aux autres critères d'élagage ils ont constaté que la norme 1 est un bon critère de sélection de filtre tout en étant indépendant des données en entrée.

Ci-dessous un pseudo algorithme de la méthode citée dans les travaux de [Li+17]

Algorithm 1: Elagage selon [Li+17]

Pour chaque filtre calculer $sj = \sum_1^{n_i} \sum |K|$;

Trier les filtres selon sj ;

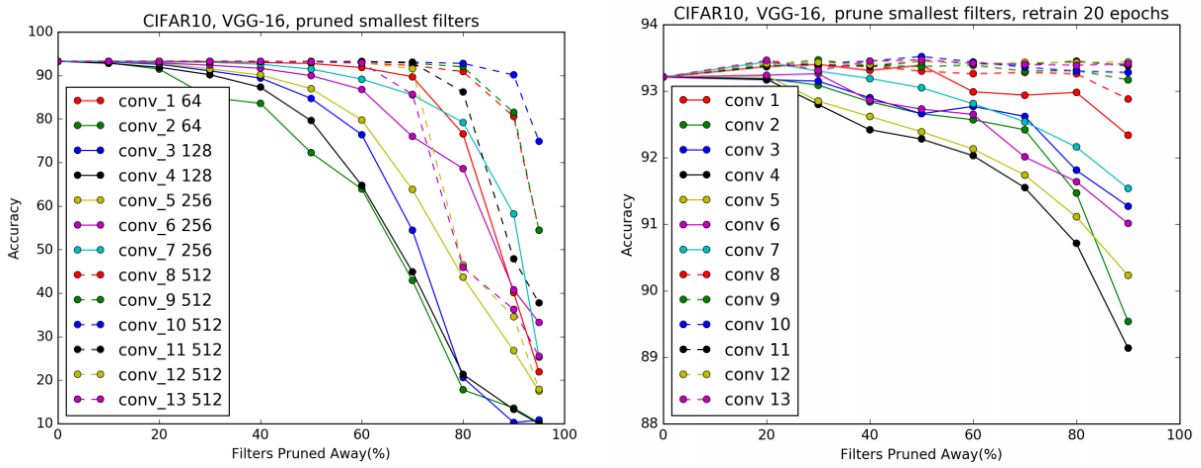
Elaguer les m filtres ayant les plus faibles sj et leur feature map correspondantes, et les noyaux de la couche suivante.;

Copier les noyaux restants de l'élagage dans une nouvelle matrice de noyaux;

Après l'élagage des filtres, la dégradation des performances doit être compensée par un nouvel entrainement (apprentissage) du réseau. Il existe deux stratégies pour élaguer les filtres sur plusieurs couches :

- Élaguer une fois et réentraîner : Élaguer les filtres de plusieurs couches à la fois et entraîner le modèle jusqu'à ce que la précision d'origine soit rétablie.
- Élaguer et réentraîner de façon itérative : Élaguer les filtres couche par couche ou filtre par filtre et entraîner le modèle de manière itérative. Le modèle est entraîné avant d'élaguer la couche suivante pour que les poids s'adaptent aux changements du processus d'élagage.

La figure 2.13 montre les résultats d'élagage basé sur la norme-1, appliqué au réseau VGG-16 et entrainé sur CIFAR10. [3]



(a) élagage sans réapprentissage (b) élagage avec ré-apprentissage

FIGURE 2.13 – Résultats de l'élagage. [Li+17]

2.3.2.3 Structured Pruning of Deep Convolutional Neural Networks

Dans cette partie on va présenter une nouvelle approche d'élagage introduite par [AHS17] basée sur l'exploitation des creux des matrices des noyaux, filtres et feature

3. CIFAR-10 : Ue jeu de données qui comprend 60000 images couleur 32x32 réparties en 10 classes, avec 6000 images par classe. Il y a 50000 images d'entraînement et 10000 images de test.

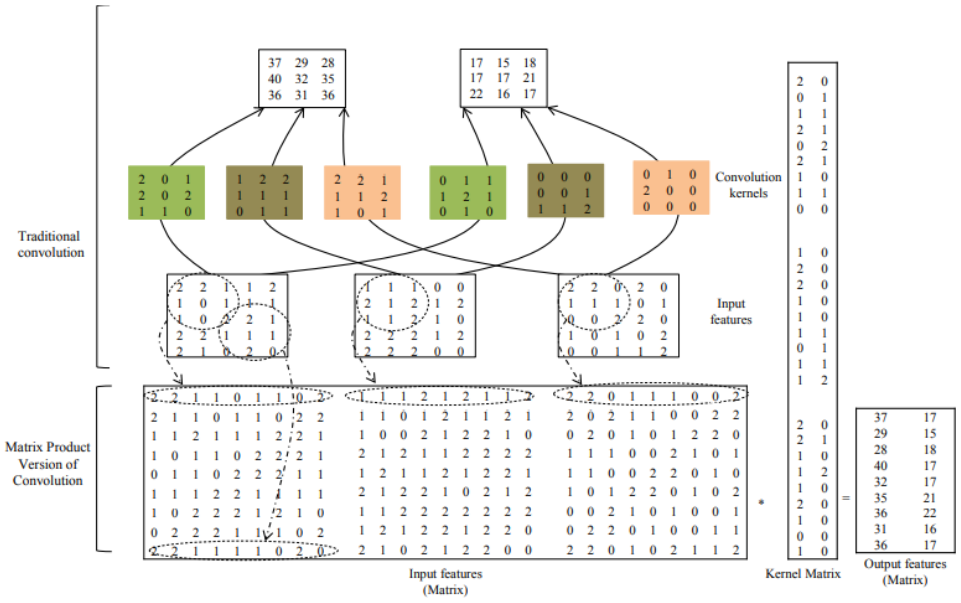
maps et sur l'algorithme Particle Filter (PF) pour la sélection des candidats de l'élagage. Il est bien claire que les éléments nuls des matrices mentionnées ci-dessus ne contribuent pas au résultat à cause de l'opération de convolution (produit scalaire), et du coup éviter l'opération dès le début réduit le nombre d'opération et le coût de calcul ce qui va résulter en des inférences plus rapides.
Mais il est très difficile de tirer parti de cet avantage à cause de la dispertion aléatoire des zéros dans les matrices, car cela exige de nombreuses opérations conditionnelles et une représentation supplémentaire pour indiquer l'emplacement des paramètres zéro ou non zéro.
Dans [Son+15] on force à zéro les paramètres proches de zéro, mais dans [AHS17] ce procédé est appliqué dans des parties bien choisies, en utilisant un masque défini par deux paramètres : un pas (Stride en anglais) et un déplacement (offset en anglais). L'idée qu'ils proposent contraint chaque connexion de convolution sortante pour la même feature map source à avoir un pas et un déplacement similaires. Le déplacement montre l'indice du premier poids élagué (mis à zéro). Dans la figure 2.14a on donne un exmple de l'idée de "Convolution Lowering" [4] sur laquelle on applique l'élagage dans la figure 2.14b. La contrainte est montrée avec les carrés de couleur similaire. Ceci réduit de manière significative la taille de la matrice de feature map et de la matrice du noyau. Les 9 premières colonnes de la ligne 1 de la matrice des feature maps d'entrée passent de 2 $\underline{2}$ 1 $\underline{1}$ 0 $\underline{1}$ 1 $\underline{0}$ 2 à 2 1 0 1 2 avec les éléments soulignés élagués. Seuls les éléments colorés en rouge dans les feature maps et les noyaux survivent et le reste est élagué. Dans cet exemple, la taille de la matrice de feature maps est réduite de 9×27 à 9×15 et la taille de la matrice de noyaux est réduite de 27×2 à 15×2.

Le processus d'élagage doit sélectionner des combinaisons de connexions moins importantes comme candidats à l'élagage. Ces connexions, une fois élaguées, ont le moins d'effets négatifs sur les performances du réseau, ce qui peut être compensé par un réentraînement. Dans ce travail, ils proposent de localiser les connexions candidates à l'élagage avec la méthode séquentielle de Monte Carlo (en anglais Sequential Montre Carlo SMC) aussi connue sous le nom du Filtrage Particulaire (Particle Filter en anglais) [Aru+02]. Les algorithmes de Filtrage Particulaire (PF) trouvent leurs applications dans plusieurs domaines [Aru+02], [Num+03] [Nak+09].le PF représente la distribution de filtrage avec un ensemble de particules pondérées. est généralement appliqué au système modélisé par les équations

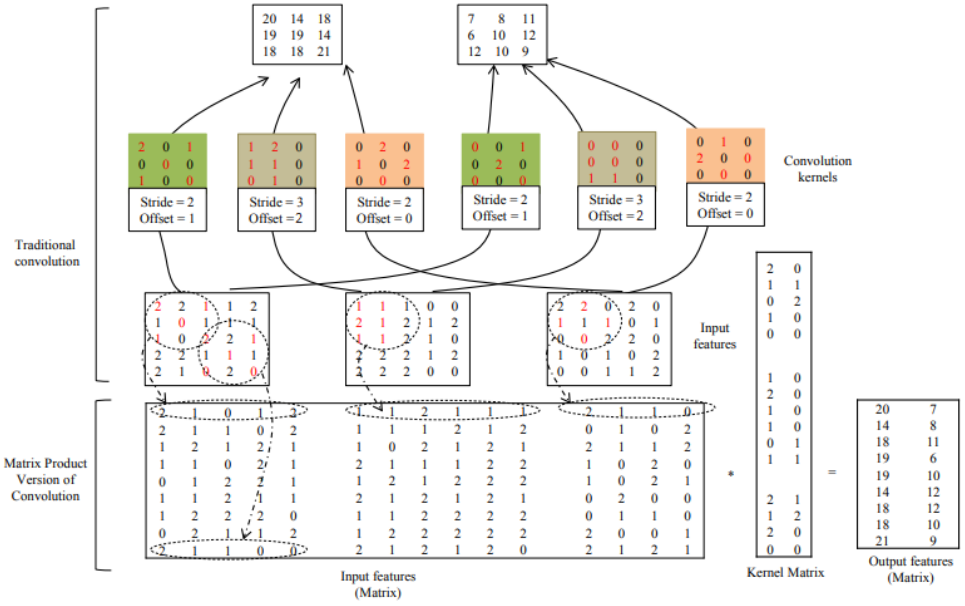
$$x_k = f(x_k - 1) + \underset{\sim}{\mathrm{t}}_k \tag{2.1}$$

$$z_k = h(x_k) + v_k \tag{2.2}$$

4. Convolution Lowering : Vectorisation des opérations dans un CNN : utilisé entre autre pour le calcule parallèle



(a) Exemple de "Convolution Lowering"



(b) Elagage appliqué à l'éxemple 2.14a

FIGURE 2.14 – Exemple d'élagage selon [AHS17]

Le vecteur d'état est représenté par x, k indice de temps ou d'itération, z_k est le vecteur d'observation, v_k est le bruit d'observation et ζ_k est le bruit d'état. La fonction d'observation est représentée par $h(.)$ tandis que $f(.)$ représente la fonction de transition.

Lors de l'élagage de n connexions, les combinaisons possibles sont de l'ordre de $O(2n)$ ce qui signifie qu'une recherche exhaustive n'est pas possible. Avec N particules, nous simulons plusieurs combinaisons de connexions possibles. La Figure 2.15 montre un exemple de vecteur d'état d'une particule. Le réseau lui même est utilisé comme fonction d'observation qui est bruyante car le taux d'erreur de classification est supérieur à 0%. Nous évaluons le taux d'erreur de classification (Missclassification Rate (MCR)) sur un ensemble d'évaluation pour chaque particule. De cette façon, le poids d'importance (une probabilité en utilisant les termes de PF) est calculé par 1 - MCR. Les poids de grande importance survivent à travers plusieurs itérations tandis que les autres sont élaguées. MCR compare la classe attribuée par le réseau avec la vraie classe. Une fois que toutes les particules sont assignées à leurs probabilités, on construit la fonction de distribution cumulative (CDF) et on rééchantillonne avec le rééchantillonnage d'importance séquentiel (Sequantial Importance Resampling (SIR)) [Aru+02]. La fonction de transition est simulée en perturbant le masque d'élagage (Pas et déplacement).

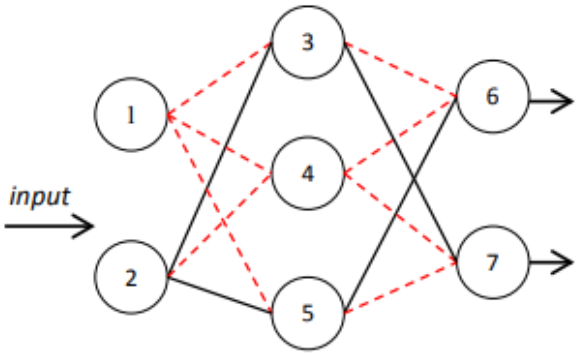


FIGURE 2.15 – Exemple du vecteur d'état [AHS17]
w_{ij} c'est le poids allant du neuron i au neuron j

Les PF explorant un nombre fini d'échantillons souffrent de problèmes de dégénérescence et de famine lorsque les particules moins probables sont remplacées par les particules très probables [Tia+14]. Le filtrage particulaire évolutif (Evolutionary Particle Filter (EPF)) propose une approche hybride où l'algorithme génétique (GA) est combiné avec le PF [Kwo+05]. Les particules sont similaires aux chromosomes et la survie du plus apte est équivalente à l'algorithme de rééchantillonnage. Avec l'approche hybride, le but est d'augmenter l'aptitude de toute la population en effet elle redéfinit les particules dans les régions les moins probables tout en maintenant les gènes les plus probables dans son

chromosome. L'EPF réduit les coûts de calcul car il nécessite moins de particules que le PF classique [Kwo+05]. Pour aider à réduire le coût de la recherche de candidats à l'élagage, l'attribution de poids d'importance utilise des ensembles d'évaluation de petite taille. On considère également que les techniques de SMC explore d'une meilleur façon l'espace des paramètres réseau.

2.3.2.4 Learning to Prune Filters in Convolutional Neural Networks

On ne sait jusqu'ici si les techniques orientées données ou basées sur l'apprentissage automatique donnent de meilleurs résultat, de plus les techniques existantes ne donnent pas un contrôle sur le compromis entre la précision de classification et l'efficacité de l'élagage (taux d'élagage). Les travaux [Hua+18] proposent justement une nouvelle technique qui essaye de répondre à ces besoins, elle apprend à élaguer les filtres redondants en entraînant un agent sous forme de réseau de neurones.Ils ont montré que cette approche d'élagage donne de meilleures performances.
On suppose qu'on a un CNN déja entraîné noté f ayant L couches convolutionnelles. Soient N_l le nombre de filtres dans la couche l, et l'ensemble des filtre de la couche l $W^l = \{w_1^l, w_2^l, ..., w_{N_l}^l\}$ le but c'est d'élaguer des filtres redondant en entier de l'ensemble W^l.
On y parvient en entraînant un agent noté π^l qui prend W^l en entrée et fait une suite de choix binaires notée $A^l = \{a_1^l, a_2^l, ..., a_{N_l}^l)\}$ où $a_i^l \in \{0, 1\}$, ou $a_i^l = 0$ veut dire que l'agent traite le filtre comme non nécessaire et $a_i^l = 1$ le contraire. On modélise l'agent par un réseau de neurones qui a pour objectif de maximiser la fonction récomponse notée χ :

$$\chi = R(A^l, X_{val}) = \psi(A^l, X_{val}, b, p^*) \times \varphi(A^l) \tag{2.3}$$

$$\psi(A^l, X_{val}, b, p^*) = \frac{b - (p^* - p)}{b} \tag{2.4}$$

$$\varphi(A^l) = \log\left(\frac{N^l}{C(A^l)}\right) \tag{2.5}$$

Le terme de précision ψ (calculé selon l'équation 2.4) garantit que la baisse de performance (précision de classification) évaluée sur un ensemble de données de validation X_val est limitée par b qui est un hyperparamètre contrôlant le compromis déjà cité. p^* et p sont respectivement la performance du modèle de base f et le nouveau modèle f_{A^l}.Plus le p est grand, plus ψ contribue à la récompense. Cependant, lorsque la baisse de performance $(p^* - p)$ est supérieure à la borne b, ψ contribue négativement à la récompense finale. Cela oblige l'agent d'élagage à maintenir la performance du réseau élagué au-dessus d'un niveau spécifié.
Le terme d'efficacité φ est calculé par l'équation 2.5. Il incite l'agent π à élaguer

davantage de filtres. $C(A^l)$ indique le nombre de 1-actions dans Al qui est aussi le nombre de filtres conservés. Un petit $C(A^l)$ signifie que seuls quelques filtres sont conservés et que la plupart des filtres sont éliminés. Plus le $C(A^l)$ est petit, plus le modèle élagué est efficace et plus φ contribue à la récompense finale. L'opérateur logarithme garantit que deux termes de l'équation 2.3 sont du même ordre de grandeur.
La table 2.9 montre un résultat d'élagage selon [Hua+18] en changeant le seuil de perte en précision b et notant le taux d'élagage de filtre, le taux de FLOPs [5] et le gain en terme d'acceleration. On remarque que l'hyperparametre b controle plutôt bien la perte en précision de classification, en plus cette perte n'est relativement pas trop importante vu les gains observés.

b	Taux d'élagage(%)	FLOPs évités(%)	Acceleration GPU(%)	Acceleation CPU(%)	Précision perdue(%)
0.5	83.3	45	32.4	48.8	0.6
1	82.7	55.2	37.8	56.6	1.1
2	86.5	64.5	44.6	63.4	1.9
4	92.8	80.6	63.4	79.2	3.4

TABLE 2.9 – Résultat d'élagage sur VGG16 sur CIFAR-10. [Hua+18]

2.3.3 Arithmetic approximation

Plusieurs études ont démontré que la précision des opérations et des opérandes dans les CNN4 peuvent être réduites sans que leur performance prédictive soit affectée de manière critique. Cette réduction peut être réalisée en quantifiant l'une ou l'autre ou les deux entrées CNN, les poids et/ou les FM en utilisant une représentation numérique à point fixe et en mettant en œuvre des multiplicateurs et des additionneurs approximatifs.

2.3.3.1 Fixed Point Arithmetic :

D'une manière générale, les modèles CNN sont déployés dans les CPU et les GPU avec la même précision numérique que celle avec laquelle ils ont été formés, en s'appuyant sur une représentation en virgule flottante de simple précision. Ce format utilise 32 bits, disposés selon la norme IEEE754.
Néanmoins, plusieurs études dans [AHS15], [Gup+15] et [LTA16] démontrent que l'inférence des CNN peut être réalisée avec une précision réduite des opérandes. De plus, les travaux de [CBD14], [Hub+16], [Zho+17b] et [Wu+16] démontrent l'applicabilité de l'arithmétique du point fixe [6] à l'entraînement des CNN. Dans les deux cas, les Feature maps et/ou les poids sont quantifiés à l'aide d'un schéma de représentation de point fixe.

5. FLOP : Float Operation, opération en virgule flottante.
6. Point Fixe : Fixed point en anglais, représentation en virgule fixe des nombres sur machine.

Dans la version la plus simple de ce format, les nombres sont codés avec la même largeur de bit (bw) qui est définie en fonction de la plage numérique et de la précision souhaitée. Plus particulièrement, tous les opérandes partagent le même exposant (c'est-à-dire le facteur d'échelle) ou la meme position du point de base [7]. Dans cet partie, nous appelons cette représentation Point fixe Statique (SFP).
Par rapport à la virgule flottante, le calcul SFP avec une largeur de bits compacte est connu pour être plus performant en termes d'utilisation du matériel et de consommation d'énergie. Cela a motivé les premières mises en œuvre à utiliser la SFP pour construire des accélérateurs CNN basés sur FPGA, en [Gok+14], où les auteurs utilisent un format 16 bits (Q8.8) pour représenter les features maps et les poids. Pour éviter les dépassements de capacité, la largeur de bit est élargie lors du calcul des sommes pondérées des convolutions et des produits internes selon plusieurs stratégies.

2.3.3.2 Dynamic fixed point

Dans les topologies profondes, on peut observer que des parties distinctes d'un réseau peuvent avoir une plage numérique de données significativement différente. Plus particulièrement, les features maps des couches profondes ont tendance à avoir un étendue numérique plus large que les premieres Feature Maps, alors que les poids sont généralement beaucoup plus petits que les Feature maps. En conséquence, la largeur de bit est élargie pour garder la même précision tout en évitant le dépassement de capacité. En conséquence, et comme indiqué [CBD14], le SFP avec son exposant fixe partagé, est souvent mal adapté à l'apprentissage profond.
Pour résoudre ce problème, les travaux dans [CBD14] et [GMG16] préconisent l'utilisation du point fixe dynamique (DFP) premièrement introduit dans [Wil91]. Dans le DFP, différents facteurs d'échelle sont utilisés pour traiter différentes parties du réseau. Plus particulièrement, des facteurs d'échelle distincts sont attribués aux poids, aux sommes pondérées et aux sorties de chaque couche. Les facteurs d'échelle et les largeurs de bits optimaux (c'est-à-dire ceux qui offrent le meilleur compromis entre la perte de précision et la charge de calcul) pour chaque couche peuvent être dérivées après une exploration en force brute en utilisant des frameworks dédiés qui supportent la DFP comme [GMG16], [Guo+17] pour Caffe et d'autre pour TensorFlow. De plus, ces outils peuvent affiner (réentraîner) le modèle CNN pour améliorer la précision des réseau.

2.3.3.3 Low Rank Approximation

Ces méthodes sont basées sur une observation clé, à savoir que la plupart des filtres ou éléments CNN sont de rang inférieur et peuvent être décomposés en couches légères

7. Point de base : le symbole utilisé dans les représentations numériques pour séparer la partie entière d'un nombre de sa partie fractionnaire.

par factorisation matricielle [6, 17] et donc une façon de réduire les calculs effectués dans les CNN est de maximiser le nombre de filtres séparables dans les modèles CNN. Un filtre séparable 2D θ^{sep} a un rang unitaire (c'est-à-dire que le $rang(\theta^{sep}) = 1$), et peut être exprimé sous la forme de deux filtres 1D successifs $\theta_{J\times 1}$ et $\theta_{1\times K}$. En étendant ceci aux filtres 3D, une convolution 3D séparable nécessite des multiplications $n_i + h + w$ alors qu'une convolution 3D standard nécessite des multiplications $n_i \times h \times w$
Néanmoins, seule une petite partie des filtres des modèles CNN sont séparables.[Den+14] a fait l'une des premières tentatives d'application de méthodes LRA telles que la décomposition (SVD) [8] pour la simplification des réseaux.
[JVZ14] a décomposé les filtres k × k en filtres k × 1 et 1 × k, ce qui a permis un gain de calculs.Ils ont étudié deux schémas d'optimisation différents, l'un pour l'approximation des filtres et l'autre pour l'approximation des feature maps.
[Zha+16] a utilisé la SVD généralisée pour traiter la non-linéarité dans les réseaux et a obtenu des résultats prometteurs dans les réseaux CNN très profonds.
Récemment, Pour augmenter la proportion des poids séparables, [Wen+17] a proposé d'utiliser la "Force Régularization" pour entraîner les réseaux afin de forcer les noyaux de convolution à être séparables en pénalisant les filtres de rang élevé lors de l'apprentissage du réseau.[Sir+15], Alternativement, et après l'apprentissage, les poids W_l d'une couche donnée peuvent être approchés par un petit ensemble de r filtres de rang inférieur qui peuvent être implémentés comme une succession de filtres entièrement séparables. Dans ce cas, $r \times (n_i \times h \times w)$ multiplications sont nécessaires pour traiter une seule convolution 3D. Pour les couches entièrement connectées, dans lesquelles le traitement se résume à un produit de matrice vectorielle, une approximation de rang inférieur peut être obtenue en utilisant, par exemple, la décomposition SVD de la matrice de poids.
Enfin, et de la même manière que pour l'élagage, l'approximation de rang inférieur des poids est suivie d'un réglage fin afin de compenser la baisse de précision de la classification.

2.3.3.4 Weight Quantization

La quantification des CNN est un domaine de recherche actif avec des approches et des objectifs variés. Il s'agit de réduire le nombre de bits requis pour représenter chaque paramètre Dans cette section, nous passons brièvement en revue les stratégies les plus prometteuses de différentes catégories. Nous classons les approches en fonction de deux critères [Sch+18] : **(i)** les approches qui quantifient uniquement les poids et les approches qui quantifient à la fois les poids et les activations, et **(ii)** les approches "flexibles" qui permettent différentes largeurs de bits et les approches "non flexibles" qui sont conçues pour des représentations de poids spécifiques

8. SVD : Single Value Decomposition : Décomposition en valeur singulière, appelée aussi la diagonalisation

1. **Quantification non flexible des poids seulement :** [CBD15] ont introduit "Binary Connect" (BC) dans lequel ils limitent les poids à -1 ou 1. [Ras+16] ont proposé des réseaux de poids binaires (Binary Weight Networks - BWN), qui améliorent la BC en introduisant des facteurs d'échelle par canal. [LL16] ont introduit les réseaux de poids ternaires ("Ternary Weight Networks" : TWN), afin de tenir compte de la dégradation de la précision des BWN. [Zhu+16] ont proposé la quantification ternaire entraînée (Ternary Quantization : TTQ), qui étend TWNs par des facteurs d'échelle non uniformes et entraînables.
2. **Quantification flexibles des poids seulement :** [HMD16] ont proposé la Deep Compression, une méthode qui met à profit l'élagage,la quantification et le partage des poids et ensuite le codage de Huffman pour la compression du modèle.[Zho+17a] ont proposé la quantification incrémentale du réseau (INQ) pour quantifier des DNN de pleine précision pré-entraînés à des zéros et des puissances de deux. Pour ce faire, on divise itérativement les poids en deux ensembles, dont l'un est quantifié tandis que l'autre est réentraîné pour compenser la dégradation de la précision.
3. **Quantification non flexible des poids et des activations :** ABC-Net [LZP17] quantifient les poids et les activations à -1 ou 1.En se basant les travaux de [Zhu+16], [Far+18] ont introduit la quantification symétrique (SYQ) par mise à l'échelle au niveau du pixel et la quantification de l'activation à virgule fixe
4. **Quantification flexible des poids et des activations :** [Zha+18a] ont proposé un quantificateur apprenable de manière adaptative (Learnable Quantizer - LQ-Nets) et atteint une précision de classification des CNN actuels. [Sch+18] proposent une nouvelle approche basée sur des facteurs d'échelle entraînables en combinaison avec une approche de clustering (nested-means clustering).

Weight sharing : Le partage des poids est utilisé après la quantification, il s'agit de limiter le nombre de poids effectifs que nous devons stocker en faisant en sorte que les connexions qui appartiennent au même groupe (cluster) partagent le même poids.
le partage des poids utilisé dans [HMD16] est illustré dans la figure 2.16. Supposons que nous ayons une couche qui a 4 neurones d'entrée et 4 neurones de sortie, le poids est une matrice 4×4. En haut à gauche se trouve la matrice de poids 4×4, et en bas à gauche la matrice de gradient 4×4. Les poids sont quantifiés à 4 groupes (désignés par 4 couleurs) en utilisant l'algorithme K-means, tous les poids dans le même groupe partagent la même valeur, donc pour chaque poids, nous devons alors stocker seulement un petit indice dans un tableau de poids partagés. Lors de la mise à jour, tous les gradients sont regroupés par couleur et additionnés, multipliés par le taux d'apprentissage et soustraits des centroïdes partagés de la dernière itération. Pour AlexNet élagué, Ils ont été capables de quantifier

à 8 bits (256 poids partagés) pour chaque couche CONV, et à 5 bits (32 poids partagés) pour chaque couche FC sans aucune perte de précision.

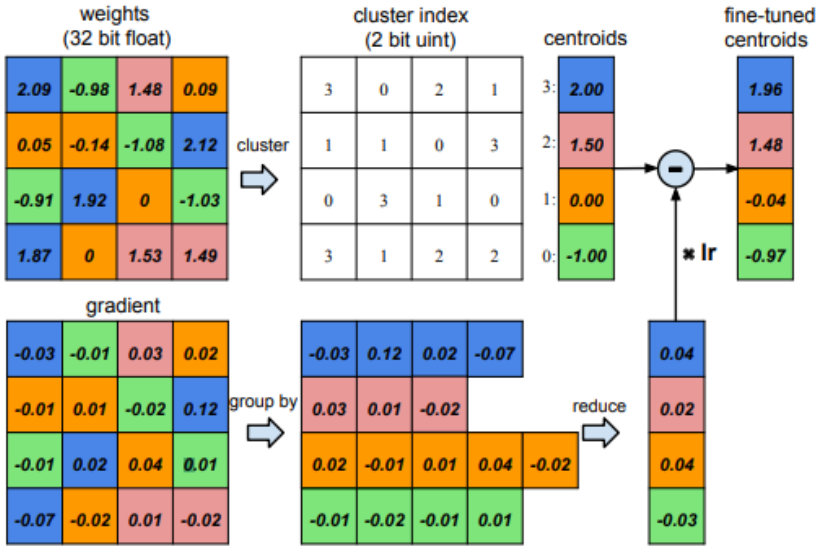


FIGURE 2.16 – Quantification selon [HMD16]

2.3.4 Conclusion :

L'efficacité des calculs reste une contrainte pour les algorithmes d'apprentissage profond. Il n'existe pas de méthodes universelles pour optimiser un modèle profond car chaque modèle est spécifique et conçu pour une application particulière. Des travaux supplémentaires doivent être réalisés dans ce sens pour exploiter toute leur puissance sur les appareils limités en ressources. L'introduction des architectures optimisées a montré qu'il est prometteur non seulement de compresser les modèles mais aussi de les construire intelligemment. Ainsi, une architecture bien conçue est la première clé des réseaux optimisés. Des travaux comme la recherche d'architecture neurale *Neural Architecture Search* qui vise à construire et à rechercher la meilleure conception architecturale du réseau, est la prochaine étape vers des modèles optimisés.

2.4 Conclusion

Bibliographie

Sources primaires

[18] « Accès à de l'information en mobilité par l'image pour la visite de Musées : Réseaux profonds pour l'identification de gestes et d'objets. », *in* : (2018).

[19a] *Mobilenet v1 pre-trained models.* `https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md`, 2019.

[19b] *Mobilenet v2 pre-trained models.* `https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet`, 2019.

[AHS15] Sajid Anwar, Kyuyeon Hwang et Wonyong Sung., « Fixed point optimization of deep convolutional neural networks for object recognition », *in* : *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP* (2015).

[AHS17] Sajid Anwar, Kyuyeon Hwang et Wonyong Sung, « Structured Pruning of Deep Convolutional Neural Networks », *in* : Department of Electrical Engineering et Computer Science, Seoul National University, 2017.

[Bic01] Pierre Bicon, *De la memoire*, Paris : Serigraph, 1901.

[CBD14] Matthieu Courbariaux, Yoshua Bengio et Jean-Pierre David., « Training deep neural networks with low precision multiplications. », *in* : *The International Conference on Learning Representations (ICLR)* (2014).

[CBD15] Matthieu Courbariaux, Yoshua Bengio et Jean-Pierre David, « Binaryconnect : Training deep neural networks with binary weights during propagations. », *in* : *CoRR, abs/1511.00363* (2015).

[Den+14] E L Denton, W Zaremba, J Bruna, Y LeCun et R Fergus, « Exploiting linear structure within convolutional networks for efficient evaluation. », *in* : *Advances in Neural Information Processing Systems* (2014), p. 1269-1277.

[DGR88] Rumelhart D.E., Hinton G.E. et Williams R.J, « Learning representations by back-propagating errors. Cognitive modeling, 5(3) :1. », *in* : (1988).

[FH04] P. Felzenszwalb et D. Huttenlocher, « Efficient Graph-Based Image Segmentation », *in* : (2004).

[G+17] Howard A. G., Zhu M., Chen B., Kalenichenko D., Wang W., Weyand T., Andreetto M. et Adam H., « Mobilenets : Effcient convolutional neural networks for mobile vision applications. », *in* : *arXiv preprint arXiv :1704.04861* (2017).

[GHA17] Sahar GHANNAY, « Étude sur les représentations continues de mots appliquées à la détection automatique des erreurs de reconnaissance de la parole. Informatique et langage [cs.CL] », *in* : Université du Maine, 2017.

[Gir+14] Ross GIRSHICK, Jeff DONAHUE, Trevor DARRELL et Jitendra MALIK, « Rich feature hierarchies for accurate object detection and semantic segmentation. In Computer Vision and Pattern Recognition », *in* : (2014).

[Gir15] Ross GIRSHICK, « Fast R-CNN », *in* : (2015).

[GMG16] Philipp GYSEL, Mohammad MOTAMEDI et Soheil GHIASI, « Hardware-oriented Approximation of Convolutional Neural Networks », *in* : (2016).

[Gok+14] Vinayak GOKHALE, Jonghoon JIN, Aysegul DUNDAR, Berin MARTINI et Eugenio CULURCIELLO., « A 240 G-ops/s mobile coprocessor for deep neural networks. », *in* : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR* (2014).

[Guo+17] Shasha GUO, Lei WANG, Baozi CHEN, Qiang DOU, Yuxing TANG et Zhisheng LI, « FixCaffe : Training CNN with Low Precision Arithmetic Operations by Fixed Point Caffe », *in* : *Proceedings of the International Workshop on Advanced Parallel Processing Technologies - APPT* (2017).

[Gup+15] Suyog GUPTA, Ankur AGRAWAL, Pritish NARAYANAN et Kailash GOPALAKRISHNAN, « Deep Learning with Limited Numerical Precision », *in* : *the International Conference on Machine Learning* (2015).

[HMD16] Song HAN, Huizi MAO et William J. DALLY, « Deep compression : Compressing deep neural network with pruning, trained quantization and huffman coding », *in* : *The International Conference on Learning Representations (ICLR)* (2016).

[Hua+18] Qiangui HUANG, Kevin ZHOU, Suya YOU et Ulrich NEUMANN, « Learning to Prune Filters in Convolutional Neural Networks », *in* : *IEEE Winter Conference on Applications of Computer Vision (WACV)* (2018).

[Hub+16] Itay HUBARA, Matthieu COURBARIAUX, Daniel SOUDRY, Ran EL-YANIV et Yoshua BENGIO, « Quantized Neural Networks : Training Neural Networks with Low Precision Weights and Activations. », *in* : *arxiv :1609.07061* (2016).

[Ian+17] Forrest Iandola, Song Han, Matthew Moskewicz, Khalid Ashraf, William Dally et Kurt Keutzer, « SqueezeNet : AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size », *in* : *The International Conference on Learning Representations (ICLR)* (2017).

[Ioa+17] Y Ioannou, D Robertson, R Cipolla et A Criminisi, « . Deep roots : Improving cnn efficiency with hierarchical filter groups.In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017. IEEE. », *in* : (2017).

[JVZ14] M Jaderberg, A Vedaldi et A Zisserman., « Speeding up convolutional neural networks with low rank expansions. », *in* : *CoRR, abs/1405.3866* (2014).

[Lau16] Stanislas Lauly, « EXPLORATION DES RÉSEAUX DE NEURONES À BASE D'AUTOENCODEUR DANS LE CADRE DE LA MODÉLISATION DES DONNÉES TEXTUELLES », *in* : (2016).

[LeC+89] Y. LeCun, B Boser, J S Denker, D Henderson, R E Howard, W Hubbard et L D Jackel, « Handwritten digit recognition with a backpropagation network », *in* : (1989).

[Li+17] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet et Hans Peter Graf, « Pruning Filters For Efficient Convnets », *in* : 2017.

[LTA16] Darryl Lin, Sachin Talathi et V Annapureddy., « Fixed Point Quantization of Deep Convolutional Networks. », *in* : *Proceedings of the International Conference on Machine Learning - ICML* (2016).

[M+] Sandler M., Howard A., Zhu M., Zhmoginov A. et Chen L.-C., « MobileNetV2 : Inverted Residuals and Linear Bottlenecks - Mobile Networks for Classification, Detection and Segmentation. », *in* : ().

[OEG00] Duda R O, Hart P E et Stork D G, « Pattern classification », *in* : (2000).

[Por18] Maxime Portaz, « Information Access in mobile environment for museum visits : Deep Neraul Networks for Instance and Gesture Recognition », *in* : (2018).

[Ras+16] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon et Ali Farhadi, « Xnor-net : Imagenet classification using binary convolutional neural networks. », *in* : *CoRR, abs/1603.05279* (2016).

[Sch+18] Günther Schindler, Wolfgang Roth, Franz Pernkopf et Holger Fröning, « N-Ary Quantization for CNN Model Compression and Inference Acceleration », *in* : 2018.

[Sir+15] Amos Sironi, Bugra Tekin, Roberto Rigamonti, Vincent Lepetit et Pascal Fua, « Learning separable filters. », *in* : *IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(1) :94–106,* (2015).

[Uij+13] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers et A. W. M. Smeulders, « Selective search for object recognition. International Journal of Computer Vision », *in* : (2013).

[Wen+17] W. Wen, C Xu, C Wu, Y Wang, Y Chen et H Li., « Coordinating filters for faster deep neural networks », *in* : *The IEEE International Conference on Computer Vision (ICCV)* (2017).

[Wil91] D. Williamson, « Dynamically scaled fixed point arithmetic », *in* : *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference* (1991).

[Wu+16] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu et Jian Cheng., « Quantized Convolutional Neural Networks for Mobile Devices », *in* : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition - CVPR* (2016).

[Zha+16] X Zhang, J Zou, K He et J Sun., « Accelerating very deep convolutional networks for classification and detection », *in* : *IEEE transactions on pattern analysis and machine intelligence, 38(10) :1943–1955* (2016).

[Zha+18a] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye et Gang Hua, « Lq-nets : Learned quantization for highly accurate and compact deep neural networks », *in* : *In European Conference on Computer Vision (ECCV)* (2018).

[Zho+17a] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu et Yurong Chen, « Incremental network quantization : Towards lossless cnns with low-precision weights », *in* : *The International Conference on Learning Representations (ICLR)* (2017).

[Zho+17b] Shuchang Zhou, Yuzhi Wang, He Wen, Qinyao He et Yuheng Zou, « Balanced Quantization : An Effective and Efficient Approach to Quantized Neural Networks. », *in* : *Journal of Computer Science and Technology.* (2017).

Sources secondaires

[Aru+02] Arulampalam, M Sanjeev, Simon Maskell, Neil Gordon et Tim Clapp, *A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking*, Signal Processing, IEEE Transactions on 50, no. 2, 2002.

[Far+18] Julian Faraone, Nicholas Fraser, Michaela Blott et Philip H.W. Leong., « SYQ : Learning symmetric quantization for efficient deep neural networks », *in* : *Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).

[He+16] K He, X Zhang, S Ren et J Sun, « Deep residual learning for image recognition », *in* : *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), p. 770-778.

[HS15] Kaiming He et Jian Sun, « Convolutional Neural Networks at Constrained Time Cost », *in* : *Conference on Computer Vision and Pattern Recognition* (2015).

[LL16] Fengfu Li et Bin Liu, « Ternary weight networks », *in* : *CoRR, abs/1605.04711* (2016).

[LZP17] Xiaofan Lin, Cong Zhao et Wei Pan, « Towards accurate binary convolutional neural network », *in* : *Advances in Neural Information Processing Systems 30* (2017).

[Nak+09] Nakamura, Kazuyuki, Ryo Yoshida, Masao Nagasaki, Satoru Miyano et Tomoyuki Higuchi., *Parameter estimation of in silico biological pathways with particle filtering towards a petascale computing.* In Pacific Symposium on Biocomputing, vol. 14, 2009, p. 227-238.

[Num+03] Nummiaro, Katja, Esther Koller-Meier et Luc Van Gool, *An adaptive color-based particle filter.* Image et vision computing 21, no. 1., 2003, p. 99-110.

[Son+15] Han Song, Jeff Pool, John Tran et William Dally, « Learning both Weights and Connections for Efficient Neural Network », *in* : 2015.

[Wu+17] B. Wu, A. Wan, F. Iandola, P. H. Jin et K. Keutzer, « SqueezeDet : Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving », *in* : *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, juil. 2017, p. 446-454.

[Zhu+16] Chenzhuo Zhu, Song Han, Huizi Mao et William J. Dally, « Trained ternary quantization », *in* : *CoRR, abs/1612.01064* (2016).